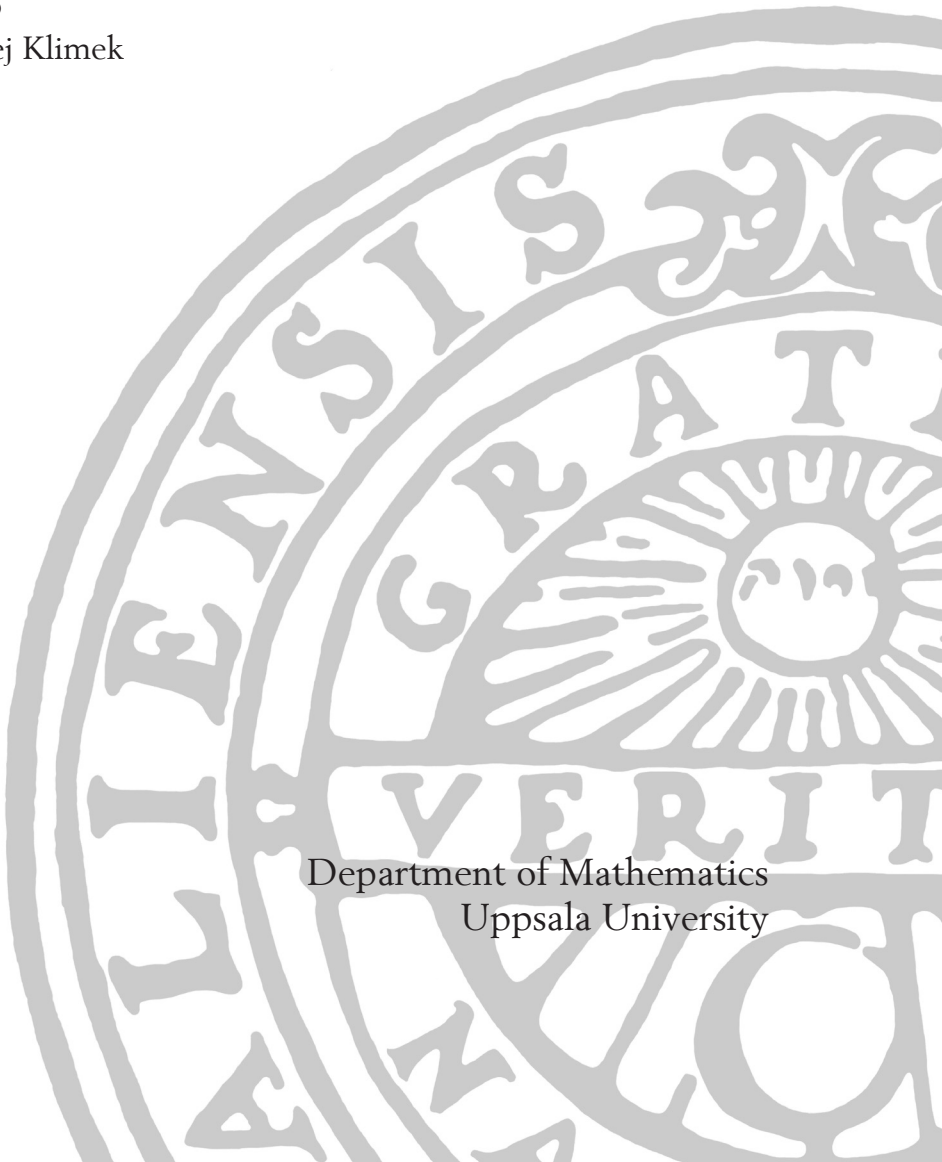
UPPSALA
UNIVERSITET

U.U.D.M. Project Report 2014:24

Calibration of stochastic volatility models

Yavor Kovachev

Examensarbete i matematik, 30 hp
Handledare och examinator: Maciej Klimek
Juni 2014



Department of Mathematics
Uppsala University

Calibration of stochastic volatility models

Master's Thesis

Author:
Yavor Kovachev

Supervisor:
Prof. Maciej Klimek

A thesis submitted in fulfilment of the requirements
for the degree of M. Sc. Mathematics - Financial Mathematics

at the

Department of Mathematics

Uppsala University

June 2014

Declaration of Authorship

I, Yavor KOVACHEV, declare that this thesis titled, 'Calibration of stochastic volatility models' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

Abstract

Calibration of stochastic volatility models

by Yavor Kovachev

This thesis examines the performance of three methods for calibrating advanced option pricing models incorporating stochastic volatility. We examine the Heston, Bates, Barndorff-Nielsen-Shephard (BNS) and the stochastic time change Normal Inverse Gaussian - Cox Ingersoll Ross (NIG-CIR) models. The calibration methods used are hybrid particle swarm optimization (PSO) combined with a gradient based interior point algorithm, a genetic algorithm (GA) combined with a gradient based interior point algorithm and an interior point algorithm alone. We show that while standard methods can in general be successfully applied, they also fail catastrophically and are not suitable for the calibration of the NIG-CIR model. Additionally, we demonstrate that perfect calibration in term of minimizing the difference between market and model prices as well as between synthetic and model parameters can be achieved only in the case of the Heston and NIG-CIR models. These findings suggest that unlike the Bates and BNS models, the Heston and NIG-CIR models are well specified and lead to stable Greek values making them suitable for the pricing, hedging and risk management of exotic derivatives.

Keywords: Stochastic Volatility Models, Calibration, Particle Swarm Optimization, Genetic Algorithm, Model Risk

Acknowledgements

I extend my gratitude to Prof. Maciej Klimek for his guidance as a teacher and a supervisor.

Contents

List of Figures

Dedicated to my mother, father, and brother. Their continued support made this thesis possible.

Introduction

Since it's introduction in 1973, the Black-Scholes model has become a standard benchmark for option pricing in the financial industry and is taught in virtually every financial mathematics course in universities around the world. As research into option pricing continued to grow in the decades following the model's publication, it became increasing clear that some of the assumptions on which it is based do not efficiently capture the realities of financial markets. Examining empirical data of log returns of major indicies like the S&P 500 or the Nasdaq-Composite shows that these returns do not follow a Gaussian distribution as is assumed in the model (see e.g. [1] Chapter 3). Additionally, the assumption of constant volatility of returns which predicts a flat implied volatility surface is unrealistic as it is a well known empirical fact that implied volatility is not constant as a function of strike nor as a function of time to maturity and generally exhibits some skewness commonly referred to as a volatility smile ([2], [3], [4]). Finally, in the Black-Scholes model log price follows a Brownian motion process which has continuous sample paths. In general, however, prices (especially for equities) tend to move by jumps even for the most highly traded stocks and their discontinuous behavior is therefore not adequately captured by Brownian motion. To address these issues a number of new models have been introduces over time in the financial literature. We focus on four advanced models: the Heston stochastic volatility model [5] and its generalization allowing for jumps in the stock price known as the Bates model [6], the Barndorff-Nielsen-Shephard model introduced in [7] and the Lévy models with stochastic time introduced by Carr, Geman, Madan and Yor [8]. The main idea we explore is the efficient calibration of the models through particle swarm optimization. The rest of this thesis is structured as follows: in section 2 we provide an overview of the models, section 3 discusses how to price options using their characteristic functions, section 4 describes the problem of model calibration and how to use particle swarm optimization to perform calibration, section 5 discusses empirical experiments and results and section 6 concludes the thesis.

Overview of the models

We consider four stochastic volatility models: the Heson, Bates, Barndorff-Nielsen-Shephard and the Normal Inverse Gaussian Cox Ingersoll Ross (NIG-CIR) stochastic time change model. This section provides an overview of each model along with an analytical formula for the characteristic function which is essential for performing computationally efficient option pricing.

2.1 The Heston Model

The Heston model [5] introduced in 1993 is a stochastic volatility model in which the risk neutral stock price dynamics are given by:

$$dS_t = (r-q)S_t dt + \sigma_t S_t dW_t^{(1)} \tag{2.1a}$$

$$d\sigma_t^2 = k(\eta - \sigma_t^2)dt + \theta\sigma_t dW_t^{(2)} \tag{2.1b}$$

$$Cov[dW_t^{(1)} dW_t^{(2)}] = \rho dt \tag{2.1c}$$

Here r is the risk neutral interest rate and $W_t^{(1)}$ and $W_t^{(2)}$ are two correlated standard Brownian motion processes. While volatility is assumed to be constant in the Black-Scholes model volatility in the Heston model follows a Cox–Ingersoll–Ross (CIR) stochastic process which was first successfully used in the area of interest rate modeling. The CIR process is mean reverting and the parameter k controls the rate at which the process returns to its long run average η. High values of k make the process essentially deterministic as any deviations from η are quickly smoothed out. The inclusion of stochastic volatility makes the model flexible enough to capture the empirically observed skewness in implied volatility. The parameter ρ is the correlation between the underlying and the volatility while θ is the volatility of volatility (i.e. the volatility of the variance of returns). The skewness of the distribution of returns of the price process in the model is governed by ρ while θ affects the kurtosis. Consequently, ρ determines the skewness of the volatility smile produced by the Heston model while θ determines its steepness as illustrated in Figure 2.1.

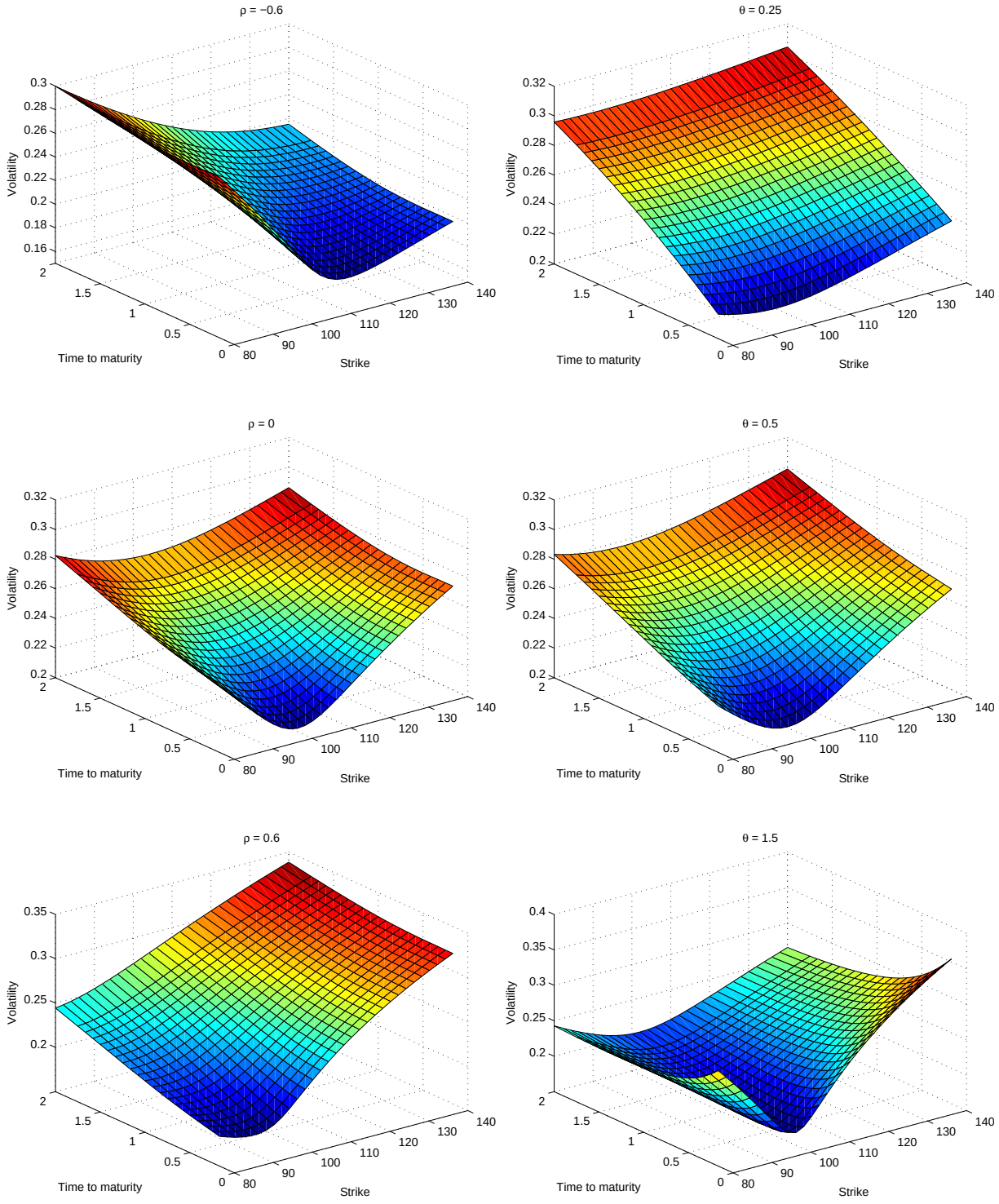


FIGURE 2.1: The effects of changes in ρ and θ on the volatility smile. In the left column the skew of the smile changes as ρ changes from negative to positive (-0.6 down to 0 and down to 0.6). In the right column the steepness of the smile increases as θ increases. Note that with the five parameters (ρ, η, θ, k, and σ_0) the model can produce a flexible volatility smile. The implied volatility surfaces in the figure were generated using direct integration techniques.

The characteristic function of the log price process in the Heston model, a derivation of which can be found in [9] and [10] is given by:

$$\phi(u,t) = E[e^{iu\log(S_t)}|S_0, \sigma_0^2)] = e^{A(u,t)+B(u,t)+C(u,t)} \tag{2.2}$$

where

$$\begin{aligned}
A(u,t) =& iu\log S_0 + iu(r-q)t \\
B(u,t) =& \frac{\eta k}{\theta^2}((k-\rho\theta iu - d)t - 2\log(\frac{1-ge^{-dt}}{1-g})) \\
C(u,t) =& \frac{\frac{v_0}{\theta^2}(k-\rho\theta iu - d)(1-e^{-dt}}{1-ge^{-dt}}) \\
d(u) =& \sqrt{(\rho\theta iu - k)^2 + \theta^2(iu+u^2)} \\
g(u) =& \frac{k-\rho\theta ui - d}{k-\rho\theta ui + d}
\end{aligned}$$

2.2 The Bates model

This model was introduced by David Bates [6] in 1996 and extends the Heston model by including jumps in the price process of the underlying. The dynamics are specified by

$$dS_t = (r - q - \lambda\mu_J)S_t dt + \sigma_t S_t dW_t^{(1)} + J_t S_t dN_t \tag{2.3}$$

$$d\sigma_t^2 = k(\eta - \sigma_t^2)dt + \theta\sigma_t dW_t^{(2)} \tag{2.4}$$

$$Cov[dW_t^{(1)} dW_t^{(2)}] = \rho dt \tag{2.5}$$

Here N_t is a Poisson process with intensity λ indicating the probability of a jump of size one is given by λdt. J_t is the random jump size and the logarithm of J_t follows a Gaussian distribution

$$\log(1+J_t) \sim N(\log(1+\mu_J) - \frac{\sigma_J^2}{2}, \sigma_J^2) \tag{2.6}$$

Both J_t and N_t are independent of each either as well as of $W_t^{(1)}$ and $W_t^{(2)}$.

As the Bates model is essentially an extension of the Heston model, the parameters ρ and θ have the same effect on the implied volatility surface as in the Heston model. The mean rate of reversion is again denoted by k. The jump parameter μ_J affects the skewness of the distribution of price returns with positive values of μ_J producing positively skewed distributions and negative values having the opposite effect. The standard deviation of the random jump size σ_J influences the kurtosis of the distribution of returns as higher values of σ_J lead to a higher variance in

the size of the jumps exhibited by the price process. Finally, the intensity parameter λ of the Poisson process N_t determines the frequency of jump occurrence. Higher values of λ lead to higher number of jumps in the price process and consequently to a higher overall volatility. Thus, λ affects the kurtosis of the distribution of returns similar to σ_J. Figure 2.2 provides an illustration.

It is important to observe that 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 as the jump terms allow for more flexibility in fitting the short end of the skew. Combining jumps and stochastic volatility therefore leads to more flexible models which can better fit market data. Thus, the advantage of jump models is twofold: not only are jumps supported by empirical data as financial security prices move by jumps but including jumps also produces models which are more flexible and more accurately fit observable market data.

The characteristic function for the Bates model is given by [11]:

$$\phi(u,t) = E[e^{iu\log(S_t)}|S_0,\sigma_0^2)] = e^{A(u,t)+B(u,t)+C(u,t)+D(u,t)} \tag{2.7}$$

where

$$\begin{aligned}
A(u,t) =& iu\log S_0 + iu(r-q)t \\
B(u,t) =& \frac{\eta k}{\theta^2}((k-\rho\theta iu-d)t - 2\log(\frac{1-ge^{-dt}}{1-g})) \\
C(u,t) =& \frac{\frac{v_0}{\theta^2}(k-\rho\theta iu-d)(1-e^{-dt}}{1-ge^{-dt}}) \\
D(u,t) =& -\lambda\mu_J iut + \lambda t((1+\mu_J)^{iu}e^{\frac{1}{2}\sigma_J^2 iu(iu-1)} - 1) \\
d(u) =& \sqrt{(\rho\theta iu-k)^2 + \theta^2(iu+u^2)} \\
g(u) =& \frac{k-\rho\theta ui-d}{k-\rho\theta ui+d}
\end{aligned}$$

2.3 The Barndorff-Nielsen-Shephard model

The Barndorff-Nielsen-Shephard (BNS) model is a Lévy type model with a stochastic volatility component. It addresses the shortcomings of the Black-Scholes model in three ways. First, the price dynamics in BNS follow more general Lévy processes than the Brownian motion process used in Black-Scholes. Instead of a Gaussian distribution, these processes are based on more flexible distributions which can accurately model the skewness and excess kurtosis present in financial data. Examples include the Variance Gamma (VG), the Negative Inverse Gaussian (NIG), CGMY (named after Carr, Geman, Madan and Yorr) and the Meixner distribution. Second, similar to the Heston model the volatility parameter in BNS is stochastic. Instead

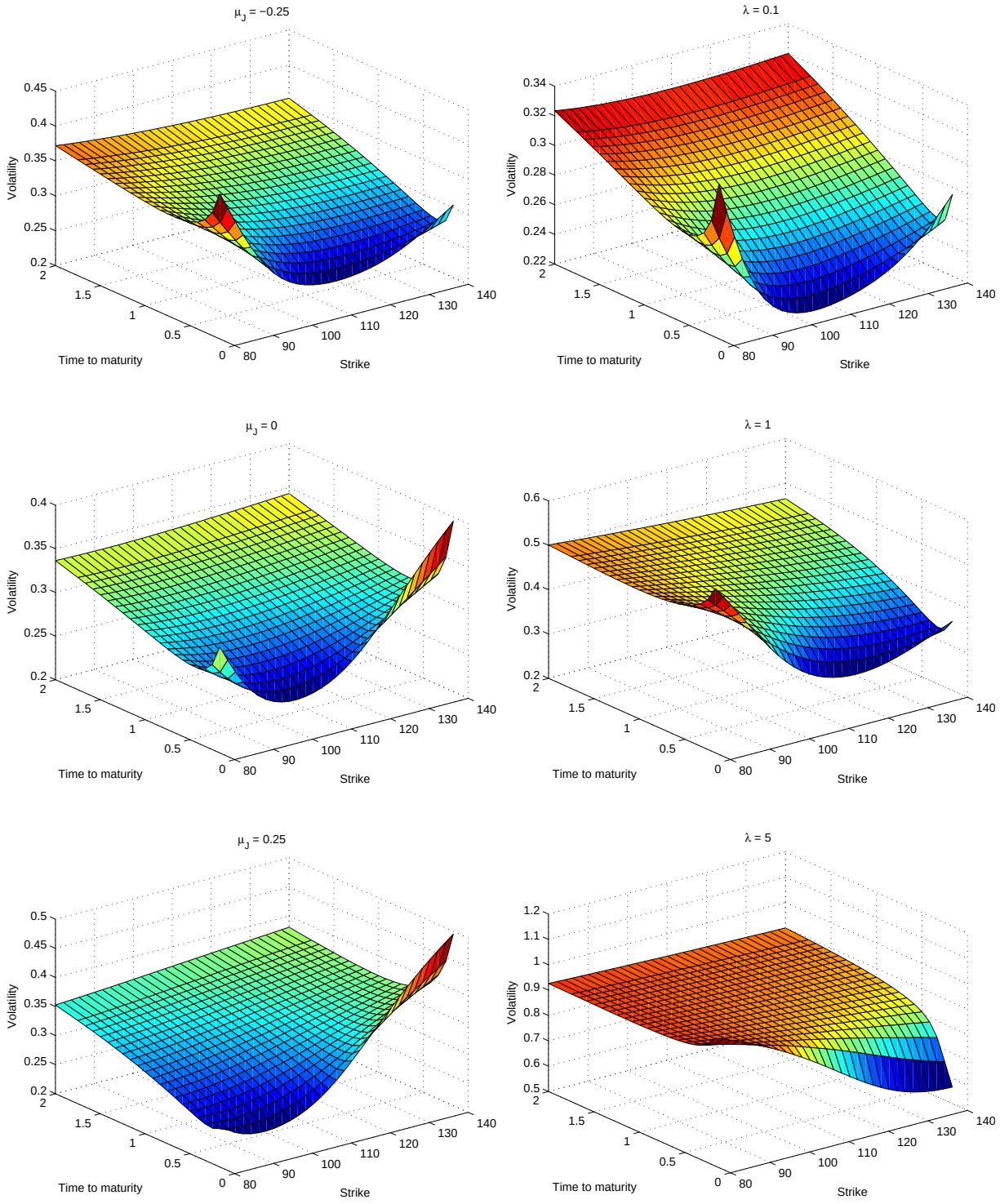


FIGURE 2.2: The effects of changes in μ_J and λ on the volatility smile. The implied volatility surfaces in the figure were generated using direct integration techniques.

of a CIR process, however, the BNS model uses an Ornstein Uhlenbeck (OU) process driven by a subordinator (a subordinator is defined as a non-negative non-decreasing Lévy process). Finally, BNS incorporates jumps in both the stock price and the volatility of returns similar to the Bates model. In BNS jumps are not independent but occur simultaneously to reflect

the empirical observation that periods of increased volatility result in decreases in stock prices. Thus, up-jumps in volatility produce down jumps in the underlying price process [1].

In this thesis we focus on a version of the BNS model for which the risk neutral dynamics of the log spot are given by

$$d(\log S_t) = (r - \lambda k(-\rho) - \sigma_t^2/2)dt + \sigma_t dW_t + \rho dz_{\lambda t} \tag{2.8}$$

and the squared volatility follows a Gamma-OU process specified by

$$d\sigma_t^2 = -\lambda\sigma_t^2 dt + dz_{\lambda t} \tag{2.9}$$

where the subordinator $z_{\lambda t}$ is a compound Poisson process

$$z_{\lambda t} = \sum_{n=1}^{N_t} x_n \tag{2.10}$$

N_t is a Poisson process with intensity a so that $E[N_t] = a$ and $\{x_n, n = 1, 2, 3...\}$ is an independent and identically distributed sequence of random variables each of which follows an exponential law with mean $1/b$. Graphs illustrating the effect of model parameters on the return distribution of the underlying price process similar to the ones in Figures 2.1 and 2.2 can be generated for the BNS process as well but have not been included here for the sake of brevity. The code section includes scripts which can be used to generate such graphs.

The characteristic function of the model is given by [1, 12]:

$$\phi(u,t) = E[e^{iu\log(S_t)}|S_0, \sigma_0^2)] = e^{A(u,t)-B(u,t)+C(u,t)} \tag{2.11}$$

where

$$\begin{aligned}
A(u,t) &= iu\log S_0 + iu(r - q - a\lambda\rho\frac{1}{b-\rho})t \\
B(u,t) &= \frac{1}{\lambda}(u^2 + iu)(1 - e^{-\lambda t})\sigma_0^2/2 \\
C(u,t) &= a\frac{1}{b-f_2}\left(b\log\left(\frac{b-f_1}{b-iu\rho}\right) + f_2\lambda t\right) \\
f_1(u) &= iu\rho - \frac{1}{\lambda}(u^2 + iu)(1 - e^{-\lambda t})/2 \\
f_2(u) &= iu\rho - \frac{1}{\lambda}(u^2 + iu)/2
\end{aligned}$$

2.4 The Normal Inverse Gaussian - Cox Ingersoll Ross (NIG-CIR) stochastic time change model

The NIG-CIR process is a Lévy model with stochastic time. It creates a stochastic volatility effect similar to the one present in the Heston, Bates and BNS models by the use of a time change in the underlying price process. The model is comprised of two processes: a Lévy process the exponential of which determines the dynamics of the asset price and a stochastic time clock process which governs the passage of time. The Lévy process is subordinated or time-changed by the stochastic clock so that time can be sped up or slowed down to reflect periods of high and low volatility respectively. The risk-neutral asset price process is given by

$$S_t = S_0 \frac{e^{rt}}{E[e^{X_{Y_t}}]} e^{X_{Y_t}} \tag{2.12}$$

where X_t is the Lévy process and Y_t is the subordinator process governing business time

$$Y_t = \int_0^t y_s ds \tag{2.13}$$

where y_t is the rate of time change. Since a subordinator is a non-negative non-decreasing Lévy process the rate of time change y_t has to be positive which in turns makes Y_t in (2.13) an increasing process. Different choices for the Lévy process X_t are possible with two of the most popular being the Normal Inverse Guassian (NIG) and the Variance Gamma (VG) process. Typical choices of the rate of time change y_t are the CIR stochastic clock as in Heston

$$d\sigma_t^2 = k(\eta - \sigma_t^2)dt + \theta\sigma_t dW_t^{(2)} \tag{2.14}$$

or the Gamma-OU as in BNS

$$d\sigma_t^2 = -\lambda\sigma_t^2 dt + dz_{\lambda t} \tag{2.15}$$

Similar to BNS, Lévy processes with stochastic time incorporate a jump component and a stochastic volatility component through X_t and Y_t respectively. Graphs illustrating the effect of model parameters on the return distribution of the underlying price process similar to the ones in Figures 2.1 and 2.2 can be generated for the NIG-CIR process as well but have not been included here for the sake of brevity. The code section includes scripts which can be used to generate such graphs.

In this thesis we have chosen to use a NIG-CIR process. In the general case, the characteristic function of a Lévy process driven by a stochastic time clock is given by [1]:

$$\begin{aligned}\phi(u,t) =& E[e^{iu\log(S_t)}|S_0, y_0] = \\ =& e^{iu((r-q)t+\log S_0)}\frac{\varphi(-i\psi_X(u);t,y_0)}{\varphi(-i\psi_X(i);t,y_0)^{iu}}\end{aligned} \tag{2.16}$$

where

$$\psi_X = \log E[e^{iuX_1}] \tag{2.17}$$

is referred to as the characteristic exponent of the Lévy processes X_t and $\varphi(u;t,y_0)$ is the characteristic function of the stochastic clock Y_t given the initial value y_0. For the specific case of a NIG-CIR process the characteristic function of the integrated CIR process is derived in [13] and is given by:

$$\begin{aligned}\psi_{CIR}(u,t;k,\eta,\lambda,y_0) =& E[e^{iuY_t}|y_0] \\ =& \frac{e^{k^2\eta t/\lambda^2}e^{2y_0iu/(k+\gamma\coth(\gamma t/2))}}{(\cosh(\gamma t/2)+k\sinh(\gamma t/2))^{2k\eta/\lambda^2}}\end{aligned} \tag{2.18}$$

$$\gamma = \sqrt{k^2 - 2\lambda^2 iu}$$

while the characteristic exponent of the NIG-Lévy process is:

$$\psi_X(\xi) = \frac{1-\sqrt{1+v\xi(-2i\mu+\xi\sigma^2)}}{v} \tag{2.19}$$

The NIG process introduced by Barndorff-Nielsen [14] is obtained through subordinating a Brownian motion process with drift by an Inverse Gaussian (IG) process with variance v and mean 1. The NIG process is therefore a time changed Brownian motion process and the parameters μ and σ in the characteristic function (2.20) are the drift and volatility of that Brownian process. Some authors define the characterstic exponent of the NIG process as:

$$\psi_X(\xi) = \delta(\sqrt{\alpha^2-\beta^2} - \sqrt{\alpha^2-(\beta+i\xi)^2}) \tag{2.20}$$

where $\beta = \mu/\sigma^2$, $\delta = \sigma/\sqrt{v}$, and $\alpha^2 = 1/v\sigma^2 + \beta^2$.

Option pricing with characteristic functions

3.1 Direct Integration

The problem of model calibration is discussed in detail in the next section but the basic idea is to find parameters which make model prices consistent with market prices. Calibration depends on the choice of objective function which measures the discrepancy between modeled and observed prices but regardless of the form of the objective function to achieve efficient calibration we must be able to price vanilla put and call options over a range of strikes and maturities in the most computationally efficient manner possible. In the standard Black-Scholes model this is easily achieved as closed form solutions for the prices of such options are readily available which is the reason why the model is so popular in practice. In the case of more advanced models, however, analytical formulas rarely exist and pricing is achieved mainly through: (1) the fast Fourier transform (FFT) method of Carr and Madam [15], (2) the fractional FFT method of Chourdakis [16] and (3) direct integration. Although the FFT method has gained a lot of traction since its publication Kilin [17], [18], has shown that with correct implementation for a specific level of accuracy direct integration can be 30 times faster than FFT and 15 times faster than fractional FFT.

Direct integration entails the use of one dimensional numerical quadrature techniques to compute vanilla option prices. There are two ways to use the method: either in conjunction with the formula of Bakshi and Madan [19] or with the formula of Attari [20]. In the both cases it is assumed that the characteristic function of the risk neutral price is known in closed form. Bakshi and Madan show that the price of a European option is given by

$$C(K,T) = e^{-qT} S_0 \Pi_1 - K e^{-rT} \Pi_2 \tag{3.1}$$

where S_0 denotes the spot price of the underlying security, K the strike price, r the risk free interest rate, q the dividend yield, Π_1 corresponds to the delta of the option (the derivative with respect to the price of the underlying $\partial C/\partial S$ i.e. rate of change of the option value with respect to change in the price of the underlying) and Π_2 is the probability of finishing in the

money. Π_1 and Π_2 are obtained by computing the following integrals

$$\begin{aligned}\Pi_1 &= \frac{1}{2} + \frac{1}{\pi}\int_0^\infty \text{Re}\left(\frac{\exp(-iu\ \log K)E[\exp((i(u-i)\ \log S_T)]}{iuE[S_T]}\right)du \\ &= \frac{1}{2} + \frac{1}{\pi}\int_0^\infty \text{Re}\left(\frac{\exp(-iu\ \log K)\phi(u-i)}{iu\phi(-i)}\right)du \end{aligned} \tag{3.2}$$

$$\begin{aligned}\Pi_2 &= \frac{1}{2} + \frac{1}{\pi}\int_0^\infty \text{Re}\left(\frac{\exp(-iu\ \log K)E[\exp((iu\ \log S_T)]}{iu}\right)du \\ &= \frac{1}{2} + \frac{1}{\pi}\int_0^\infty \text{Re}\left(\frac{\exp(-iu\ \log K)\phi(u)}{iu}\right)du \end{aligned} \tag{3.3}$$

Here ϕ stands for the characteristic function of the log stock price. Thus, given a characteristic function ϕ we can compute Π_1 and Π_2 numerically, using Gaussian quadrature for example, and substitute back into (3.1) to obtain the price of the vanilla option. Attari starts off with (3.1) and simplifies the formula to

$$\begin{aligned}C(K,T) &= S_0 - \frac{1}{2}e^{-rT}K \\ &- e^{-rT}K\frac{1}{\pi}\int_0^\infty \frac{(\text{Re}(\phi(u)) + \frac{\text{Im}(\phi(u))}{u})\cos(ul) + (\text{Im}(\phi(u)) - \frac{\text{Re}(\phi(u))}{u})\sin(ul)}{1+u^2}du\end{aligned} \tag{3.4}$$

where ϕ is again the characteristic function of the risk neutral price process and $l = Ke^{-rT}/S_T$. As Attari and Kilin point out equation (3.4) has two main advantages over (3.1) i.e. it contains only one integral which leads to only one numerical integration and it has a quadratic term in the denominator which leads to a faster rate of decay.

While the decay of the function under the integral in (3.3) is faster, numerical experiments showed that this function exhibits much higher oscillatory behavior than the two functions under the integrals of Π_1 and Π_2 in (3.1). To deal with this problem one can employ adaptive quadrature methods such as the Gauss-Kronrod technique or the simpler adaptive Simpson's rule used in Matlab's built in `quad` function with the former being a lot more computationally effective. Instead of an adaptive algorithm, after examining the behavior of Π_1 and Π_2 for the four models we decided to use the formula of Bakshi and Madan and numerically integrate via standard Gaussian quadrature similar to Gilli and Schumann [21].

The behavior of the function under the integral in Π_1 for the four models is shown in Figure 3.1 (the behavior of Π_2 is similar and is displayed in Appendix A).The strike in all four cases is \$100 and as can be seen from the figure oscillations are present in Π_1 but only for options which are extremely out of the money. Oscillations can be further exacerbated as volatility and time to maturity approach zero, but they still remain limited to cases of extremely high or low strikes which are unlikely to occur in practice. Further numerical experiments confirmed the

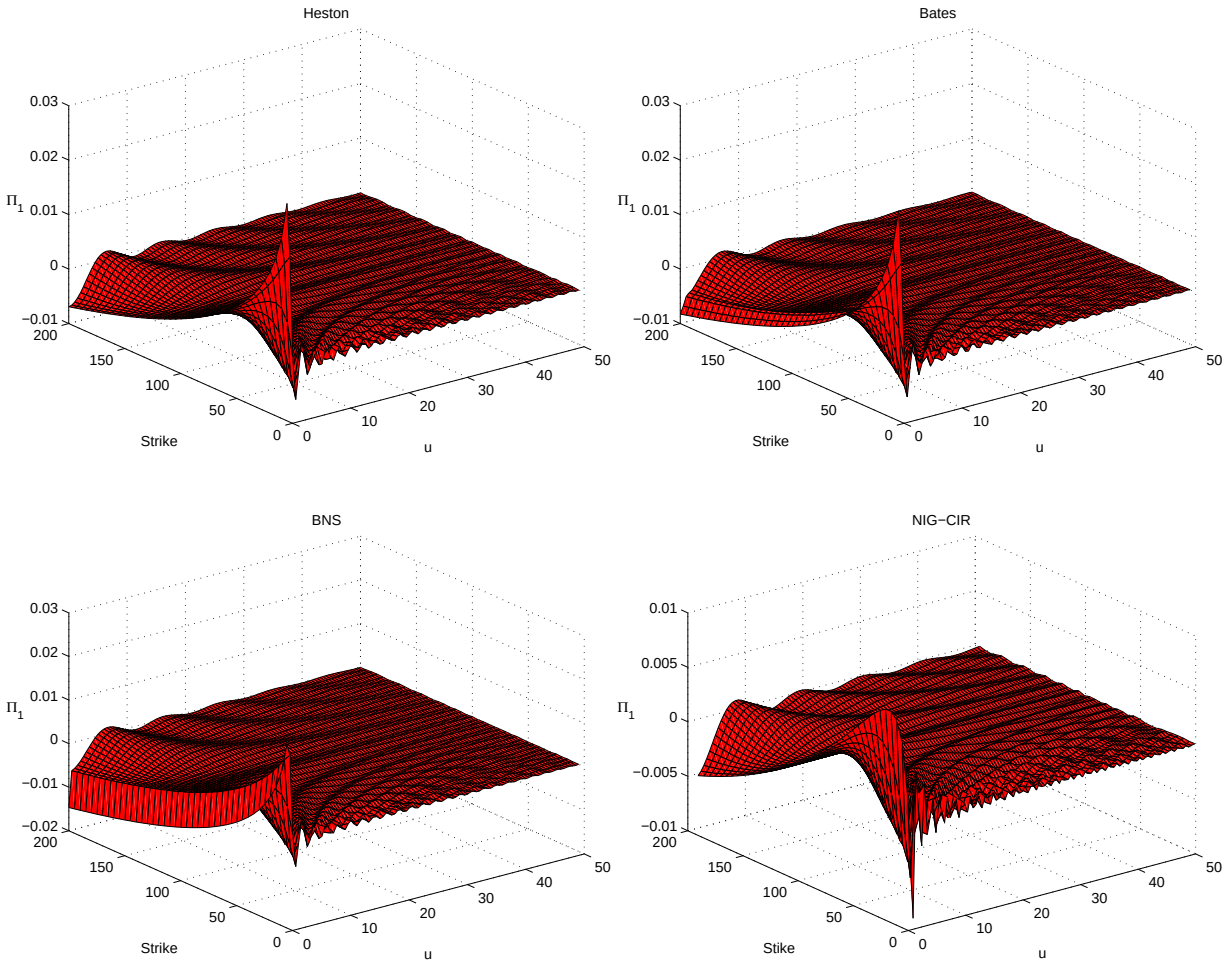


FIGURE 3.1: The behavior of Π_1 for the four models. Parameters used S=100, K=100, t=1/12, q=0, r=0.1. (Heston model: k=0.9, η =0.3,θ =0.9, ρ = -0.9, $\sigma_0^2 = 0.1$. Bates model: k=2.0, η =0.3,θ =0.9, ρ = -0.9, $\sigma_0^2 = 0.1$, $\lambda = 0.9$, μ_J = -0.9, $\sigma_J = 0.9$. BNS model: ρ = -0.9, a =1.0, b =0.1, $\lambda = 0.3$, $\sigma_0^2 = 0.3$. CIR-NIG: $\alpha = 0.2$, $\beta = 0.1$, $\delta = 0.2$, k=0.9, η =0.3, $\lambda = 0.9$, $y_0 = 0.5$)

same holds regardless of the magnitude of the strike and stock price e.g. for strikes and stock prices in the \$1000 to \$10000 range. Figure 3.2 shows Π_1 for strikes in the range [50,150].

3.1.1 Gaussian quadrature

As explained in the previous section the integrals in (3.2) and (3.3) are evaluated numerically using a Gaussian quadrature rule. The basic idea behind numerical quadrature is to approximate the definite integral of a function as a weighted sum of function values at specific points in the domain of integration e.g.

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} \omega_i f(x_i) \tag{3.5}$$

where the ω_i's are the weights and and the x_i are the nodes (the specific points in the domain). The nodes can be found using the following procedure. If we let $\varphi_0, \varphi_1, ..., \varphi_n$ be a sequence of

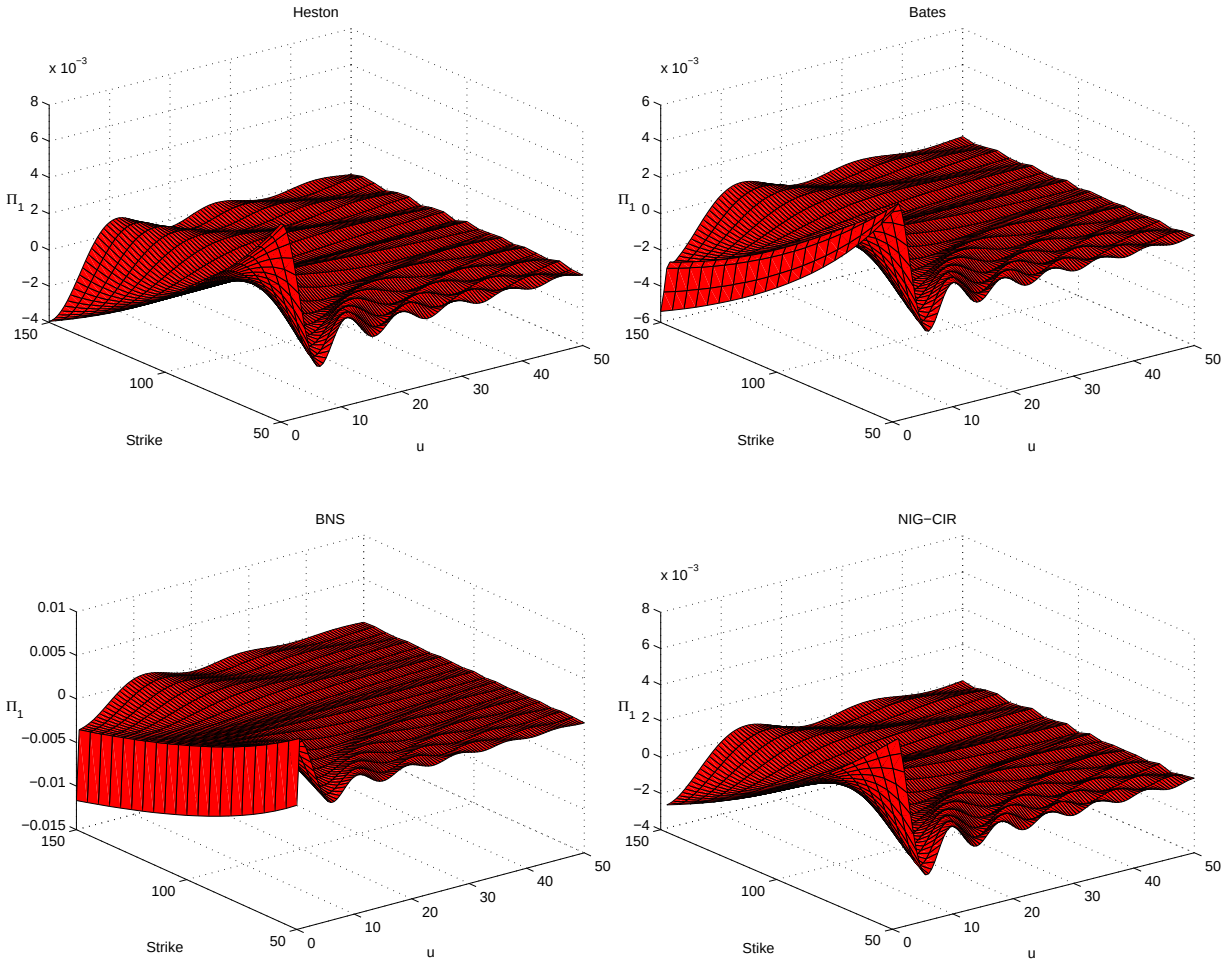


FIGURE 3.2: The behavior of Π_1 for strikes between \$50 and \$150. Same parameters as in 3.1

orthogonal polynomials e.g.

$$\int_a^b \varphi_i(x)\varphi_j(x)\omega(x)dx = 0, \text{ for all } i \neq j \tag{3.6}$$

then by the fundamental theorem of Gaussian quadrature the optimal nodes for an n-point Gaussian rule are given by the zeros of the polynomial $\varphi_n(x)$ of order n. The following recurrence holds for orthogonal polynomials:

$$\varphi_n(x) = (\alpha_n x + \beta_n)\varphi_{n-1}(x) - \gamma_n\varphi_{n-2}(x) \ , \ n \geq 1 \tag{3.7}$$

where α_n, β_n and γ_n are functions of the coefficients of $\varphi_0, \varphi_1, ..., \varphi_n$. If the polynomials are also orthonormal, i.e. if

$$\int_a^b \varphi_i(x)\varphi_i(x)\omega(x)dx = 1 \tag{3.8}$$

then $\gamma_n = \alpha_n/\alpha_{n-1}$ and (3.7) can be rewritten as

$$x\varphi_{n-1}(x) = \frac{1}{\alpha_n}\varphi_n(x) + \delta_n\varphi_{n-1}(x) + \frac{1}{\alpha_{n-1}}\varphi_{n-2}(x) \tag{3.9}$$

where $\delta_n = -\frac{\beta_n}{\alpha_n}$. In matrix notation (3.9) becomes

$$x\underbrace{\begin{bmatrix}\varphi_0(x)\\ \varphi_1(x)\\ \varphi_2(x)\\ \vdots\\ \varphi_{n-1}(x)\end{bmatrix}}_{\Phi(x)} = \underbrace{\begin{bmatrix}\delta_1 & \frac{1}{\alpha_1} & 0 & 0 & \cdots & 0\\ \frac{1}{\alpha_1} & \delta_2 & \frac{1}{\alpha_2} & 0 & \cdots & 0\\ 0 & \frac{1}{\alpha_2} & \delta_3 & \frac{1}{\alpha_3} & \cdots & 0\\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & 0 & 0 & \frac{1}{\alpha_{n-1}} & \delta_n\end{bmatrix}}_{A}\underbrace{\begin{bmatrix}\varphi_0(x)\\ \varphi_1(x)\\ \varphi_2(x)\\ \vdots\\ \varphi_{n-1}(x)\end{bmatrix}}_{\Phi(x)} + \underbrace{\begin{bmatrix}0\\ 0\\ \vdots\\ 0\\ \frac{1}{\alpha_n}\varphi_n(x)\end{bmatrix}}_{b} \tag{3.10}$$

and if we insert x^* into (3.10) such that x^* is a zero of φ_n then the term b in (3.10) vanishes leaving us with

$$x^*\Phi(x^*) = A\Phi(x^*) \tag{3.11}$$

From (3.11) we immediately see that x^* is an eigenvalue of the matrix A and consequently the nodes of the n-point Gaussian rule are given by the eigenvalues of A. Furthermore, the weight ω_i corresponding to node x_i is given by

$$\omega_i = q_{1,i}^2 \int_a^b \omega(x)dx \tag{3.12}$$

where $q_{1,i}$ is the first element of the eigenvector associated with node i and ω is the weight function. For many quadrature rules the α and δ in (3.9) are known explicitly and the matrix A can be constructed which essentially reduces the problem of finding the nodes of an n-point rule to an eigenvalue problem.

To evaluate equations (3.2) and (3.3) numerically we have chosen to use Gauss-Legendre quadrature. For this rule the weight function is defined as $\omega(x) \equiv 1$ on the interval [-1,1] and $\alpha_n = \left(4 - \frac{1}{n^2}\right)^{0.5}$ and $\delta_n = 0$. As numerical integration is only applicable over finite intervals, the domain of integration for (3.2) and (3.3) needs to be truncated. We can see from figure 3.2 that the integrals decay very rapidly and at $\omega = 50$ the values of the functions under the integral are very close to 0. The cutoff values were set at 200, 200, 20, and 250 for the Heston, Bates, BNS and NIG-CIR models respectively. Finally, the domain of integration was changed from [-1,1] to [a, b] using the the following identity

$$\int_a^b f(x)dx = \frac{b-a}{2}\int_{-1}^1 f\left(\frac{b-a}{2}z + \frac{a+b}{2}\right)dz \tag{3.13}$$

In a four cases 100 nodes were used to compute the approximation to Π_1 and Π_2 and the values obtained with the Gauss-Legendre rule were compared with Matlab's `quad` function to ensure accuracy.

3.2 Computing price surfaces over a range of maturities and strikes

Evaluating the characteristic function ϕ is the most costly part of any numerical integration procedure used for computing Π_1 and Π_2 in (3.1). To reduce that cost we observe that the characteristic function of each of the four models (Heston (2.2), Bates (2.7), BNS (2.11), NIG-CIR (2.16)) depends only on the time to maturity and is completely independent of the strike price of a given option. This is where the main computational advantage of using characteristic functions lies and to exploit it we can, for a fixed maturity, pre-compute the values of ϕ at the nodes specified by the Gauss-Legendre rule and then use those values to compute the prices over all strikes (Kilin [18]). The process is repeated by fixing a different maturity, evaluating ϕ and computing the prices for all strikes at the new maturity until all maturities are exhausted. Algorithm 1 describes how to generate a complete price surface using this idea:

Algorithm 1 Computing the prices for a given surface

Set: parameters r, q etc., set T = maturities, set K = strikes
for $t \in T$ **do**
    Compute the characterstic function ϕ
    for $k \in K$ **do**
        Compute the price for strike k, maturity t
    end for
end for

Model Calibration

4.1 Problem formulation

The main motivation behind the development of advanced stochastic volatility pricing models with or without jumps is the need to better fit market data. The process of fitting models to market data, otherwise referred to as model calibration, entails the use of optimization techniques aimed at identifying the set of model parameters for which model prices are consistent with market prices. The better the calibration that can be achieved, the higher the predictive validity of a model is and the more valuable it becomes as a tool for risk management and portfolio optimization.

Calibration typically begins with the identification of an objective function to be minimized. Different objective functions such the root mean square error (*rmse*), the average relative percentage error (*arpe*) and the relative percentage error (*rpe*) are available and depending on the application the selection of an objective function can be an interesting empirical problem in itself. Similar to [21], we have chosen to use the *rpe* and the objective function is given by

$$\min \sum_{i=1}^{N} \frac{|\mathrm{C}_i^{model} - \mathrm{C}_i^{market}|}{\mathrm{C}_i^{market}} \tag{4.1}$$

where N is the number of market prices.

With the objective function identified, the types of optimization schemes that can be used to solve (4.1) fall into three main categories: local, global and hybrid. Local schemes start with an initial parameter set and try to move in the direction of a minimum of the objective function by use of its gradient. Examples of such methods are all gradient based techniques such as the steepest descent, the conjugate gradient and the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithms. Although they are fairly easy to implement and can be highly efficient the main limitation of these algorithms is the fact that they maintain only a local view of the search space by following the objective function's gradient from the location of the initial parameter set. For this reason, their performance is highly sensitive to the choice of initial parameters and they can easily become trapped in a local optimum rather than a global one. Global optimization

schemes, as the name suggest, try to maintain a global view of the search space. They achieve this, as in the case of the particle swarm optimization (PSO) algorithm presented at the end of this section, by using a population of individuals exploring the search space simultaneously. Population-based methods such as PSO are generally classified as heuristic techniques due to the fact that they approximate the exact solution to a problem. They are extremely effective at exploring the search space and identifying areas where potential global optima are located but once such areas are found the algorithms converge rather slowly. This necessitates the use of hybrid algorithms combining global and local schemes e.g. using particle swarm to identify the general location of a global optimum and then using an interior-point algorithm or a standard gradient based method to pinpoint its exact location or, as another example, using differential evolution in combination with the Lavenberg-Marquard algorithm.

4.2 The need for heuristics

To understand why heuristic approaches are necessarily for performing calibration we can examine the objective function for the different models. The behavior of the function for the Heston model when volatility of volatility θ and mean reversion speed k vary is shown in Figure 4.1. On the left we have a search space for which the global minimum can easily be located using a standard optimization technique. Increasing the value of σ_0, however, produces the objective function in the right where a gradient based method would easily become trapped in the valley around $k = 0.5$ leading to a poor calibration. For this reason we have chosen to compare the performance of local and hybrid schemes when calibrating the models.

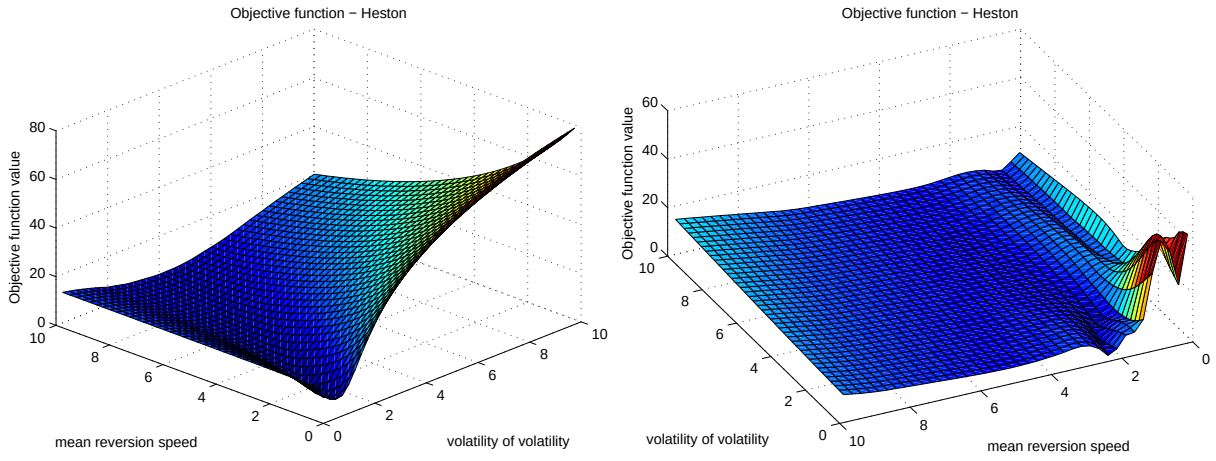


FIGURE 4.1: Parameters used: Left - $\sigma_0 = 0.3$,$\theta = 0.3$, $\rho = -0.3$, k=2 $\eta = 1.5$ Right - $\sigma_0 = 50$,$\theta = 0.3$, $\rho = -0.3$, k=2 $\eta = 1.5$

Figure 4.2 shows the objective function on a log scale for the NIG-CIR model. It is easy to see that optimization with a gradient based method would produce unsatisfactory results as any such method is bound to become trapped in a suboptimal region of the search space.

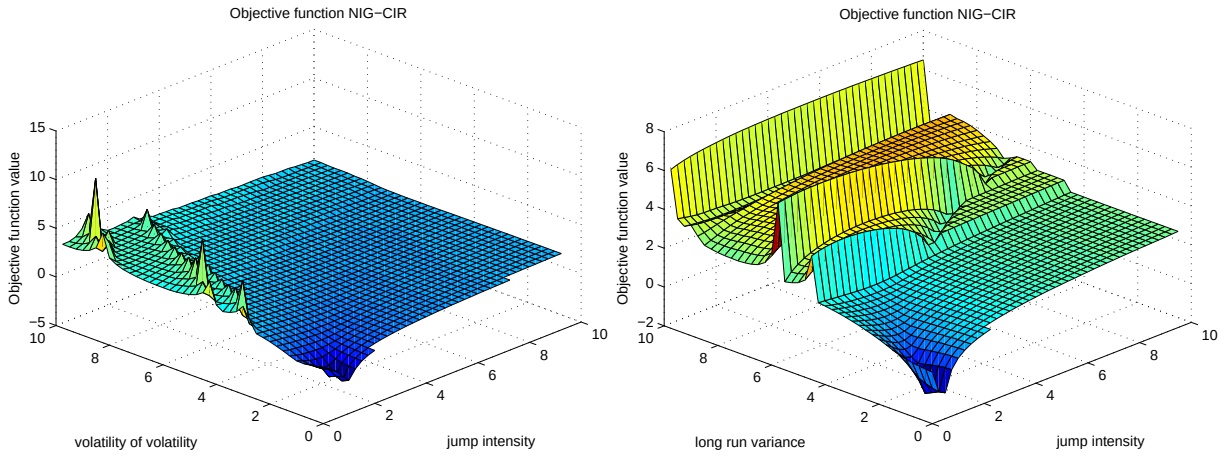


FIGURE 4.2: Parameters used: $\alpha = 10, \beta = 1, \delta = 0.5, k = 1, \lambda = 0.5y_0 = 1$

4.3 Particle swarm optimization

Particle swarm optimization (PSO) was introduced in 1995 by Eberhart and Kennedy [22]. It is a stochastic search method inspired by the coordinated movement of animals living in social groups. Each individual in the population explores its surroundings guided by of cognitive, social and stochastic interactions while the population as a whole tries to find the optimal location in the global search space. As explained in [23] the algorithm can be formulated as

$$v_i^{k+1} = \phi^k v_i^k + \alpha_1[\gamma_{1,i}(P_i - x_i^k)] + \alpha_2[\gamma_{2,i}(G - x_i^k)] \tag{4.2}$$

$$x_i^{k+1} = x_i^k + v_i^{k+1} \tag{4.3}$$

where the vectors x_i^k and v_i^k are the current position and velocity of the i-th particle in the k-th generation. The swarm consists of N particles with P_i denoting the personal best of particle i and G the global best observed among all the particles, $\gamma_{1,i}, \gamma_{2,i} \in [0, 1]$ are uniformly distributed random variables and α_1, α_2 are acceleration constants. Finally, the function ϕ is the particle inertia.

Figure 4.3 reproduced from [23] illustrates the idea of the algorithm graphically. The new velocity v_i^{k+1} of particle i is the sum of three components: a momentum term $\phi^k v_i^k$ which tends to keep the particle moving along its current path, an attraction component $\alpha_i[\gamma_{1,i}(P_i - x_i^k)]$ which moves the particle towards its personal best and an attraction component $\alpha_2[\gamma_{2,i}(G - x_i^k)]$ which guides it towards the global best G. The G term is referred to as the social component

while the cognitive component is comprised of the first two terms in (4.2). The interactions between particles are made stochastic through the uniform random variables $\gamma_{1,i}, \gamma_{2,i} \in [0,1]$.

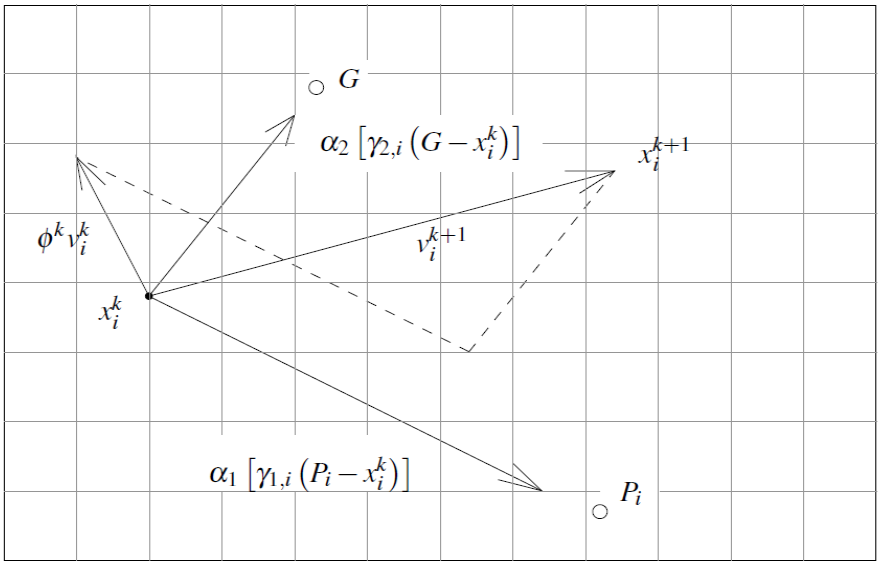


FIGURE 4.3: Graphical representation of PSO.

Once the particles have been initialized i.e. they have been randomly positioned in the search space and their initial velocities v_i^0 have been chosen, we have a population consisting of n_p positions each of which is a solution to the optimization problem (4.1) and is stored in a real-valued vector. For every successive generation a solution is updated according to (4.2) with the particle associated with the solution moving towards its personal best and the global best through perturbations brought about by multiplication with $\gamma_{1,i}, \gamma_{2,i}$. The solution is then updated according to (4.3). The procedure is summarized in Algorithm 2:

Algorithm 2 Particle Swarm

Set: parameters n_P = number of particles, n_G = number of generations, δ = inertia, α_1, α_2 = cognitive and social interaction parameters

Initialize particles x_i^0 and velocities v_i^0, $i = 1, ..., n_P$
Evaluate the objective function $F_i = F(x_i^0)$, $i = 1, ..., n_P$
$P = x^0$, $Fbest$ = F, G=min$_i(F_i)$, $gbest$ = argmin$_i(F_i)$
for k = 1 to n_G **do**
    for i = 1 to n_P **do**
        $v_i^k = \phi^k v_i^{k-1} + \alpha_1[\gamma_{1,i}(P_i - x_i^{k-1})] + \alpha_2[\gamma_{2,i}(G - x_i^{k-1})]$
        $x_i^k = x_i^{k-1} + v_i^{k-1}$
    end for
    Evaluate the objective function $F_i = F(x_i^k)$, $i = 1, ..., n_P$
    for i = 1 to n_P **do**
        if $F_i < Fbest_i$ **then** $P_i = x_i^k$ and $Fbest_i = F_i$
        end if
        if $F_i < G_i$ **then** $G = F_i$ and $gbest = i$
        end if
    end for
end for
Solution = $x_{gbest}^{n_G}$

4.4 Genetic algorithm optimization

Similar to PSO, genetic algorithm (GA) optimization is a global search method based on the natural selection processes which drive biological evolution. GA successively modifies a population of individuals which, as in the case of PSO, represent individual solutions to the optimization problem. At each iteration, the algorithm selects individuals from the current population to serve as parents for the offspring forming the next generation. Over successive generations the algorithm evolves toward the optimal solution in the same fashion as a population of animals evolves towards the optimal genetic makeup necessary for survival in a specific habitat.

The objective function value associated with which each individual in the population represents a fitness score by which the individual is ranked. Better objective function values lead to higher fitness scores and the individuals with the highest fitness form the current population are chose as the parents from which a successive generation is created. The method by which parents are chosen, i.e. roulette, tournament, stochastic uniform, is referred to as selection. In roulette selection, for example, each individual is assigned an area of a roulette wheel corresponding to its fitness score and parents are randomly chosen by spinning the roulette wheel, with fittest individuals having the highest probability of selection [1]. Once selection of the parents has taken place, the algorithm creates three type of children: elite, crossover and mutation children. Elite children are the individuals in the current population with the highest fitness who are allowed to survive onto the next generation. Crossover children are created by combining information from current parents, for example, if the information about parents is stored in bit strings cutting two bit strings and attaching them at the cut is an example of single point crossover. Finally, mutation children are created by introducing random changes to single parents. The procedure is summarized in Algorithm 3:

Algorithm 3 Genetic Algorithm (GA)

Set: parameters n_P = number of individuals (genomes), n_G = number of generations, selection, elitism, crossover, mutation rules
1: Initialize individuals x_i^0
2: Evaluate the objective function $F_i = F(x_i^0)$, $i = 1, ..., n_P$ and rank individuals
3: Select parents
4: **for** k = 1 to n_G **do**
5: Apply elitism rule, perform crossover and mutation to form a new population
6: Evaluate the objective function $F_i = F(x_i^k)$, $i = 1, ..., n_P$ and rank individuals
7: Select parents
8: **end for**
9: Solution = $x_{best fitness}^{n_G}$

[1] For more information on selection methods and GA in general refer to http://www.mathworks.se/help/gads/what-is-the-genetic-algorithm.html

Experiments and Results

5.1 Parameters and constraints

To calibrate the Heston, Bates, BNS and NIG-CIR models we create a synthetic data set for which the spot price S_0 equals \$100. The prices for strikes K from \$80 to \$120 and maturities T of $1/12, 2/12, 3/12, 4/12, 5/12, 6/12, 7/12, 8/12, 9/12, 10/12, 11/12, 1, 2$ years are computed resulting in calibration price surfaces of 21 x 13 = 273 points. For simplicity the dividend yield q is set to zero and the risk free interest rate is assumed constant at 2%. In practical applications we would obtain key interest rates and interpolate the yield curve using smoothing spline techniques or by fitting a term structure model such as the Nelson-Siegel model obtaining unique interest rates over the whole span of maturities. However, since the focus is on calibration this step has been omitted. The aforementioned parameters are used to generate prices serving as the market prices against which we calibrate. In all four cases we compare the performance of a hybrid algorithm of PSO combined with MATLAB's built in `fmincon` function, a hybrid genetic algorithm combined with `fmincon` and `fmincon` by itself. For the hybrid algorithms PSO and GA are used to identify the general location of the global minimum while `fmincon` to pinpoint its exact location. In all three cases the `fmincon` function uses the `interior-point` [1] setting.

The parameters for the Heston model are shown in the following table:

	1	2	3	4	5	6	7	8	9	10
σ_0	0.3	0.3	0.3	0.3	0.4	0.2	0.5	0.6	0.7	0.8
η	0.3	0.3	0.2	0.2	0.4	0.4	0.5	0.3	0.3	0.3
ρ	-0.3	-0.7	-0.9	0.0	-0.5	-0.5	0.0	-0.5	-0.5	-0.5
k	2.0	0.2	3.0	3.0	0.2	0.2	0.5	3.0	2.0	1.0
θ	1.5	1.0	0.5	0.5	0.8	0.8	3.0	1.0	1.0	1.0

while those for the Bates model are shown below:

[1] The `interior-point` approach to constrained minimization is to solve a sequence of approximate minimization problems. The algorithm uses a combination of Newton-Rhapson and conjugate gradient steps during optimization. For more information see: http://www.mathworks.se/help/optim/ug/constrained-nonlinear-optimization-algorithms.html#brnpd5f

	1	2	3	4	5	6	7	8	9	10
σ_0	0.3	0.3	0.3	0.3	0.4	0.2	0.5	0.6	0.7	0.8
η	0.3	0.3	0.2	0.2	0.4	0.4	0.5	0.3	0.3	0.3
ρ	-0.3	-0.7	-0.9	0.0	-0.5	-0.5	0.0	-0.5	-0.5	-0.5
k	2.0	0.2	3.0	3.0	0.2	0.2	0.5	3.0	2.0	1.0
θ	0.3	0.5	0.5	0.5	0.8	0.8	1.0	1.0	1.0	1.0
λ	0.1	0.1	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
μ_J	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1
σ_J	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1

For the BNS model we use the following parameters:

	1	2	3	4	5	6	7	8	9	10
a	0.3	0.3	0.3	0.3	0.4	0.2	0.5	0.6	0.7	0.8
b	3.0	3.0	5.0	10	14	12	12	12	13	6
ρ	-0.3	-0.7	-0.9	0.0	-0.5	-0.5	0.3	-0.5	-0.5	-0.5
λ	2.0	0.2	3.0	3.0	0.2	0.2	2.0	3.0	2.0	1.0
σ	1.5	1.0	0.5	0.5	0.8	0.8	3.0	1.0	1.0	1.0

and for the NIG-CIR model:

	1	2	3	4	5	6	7	8	9	10
α	3.0	4.0	5.0	6.0	7.0	12	16	16	13	18
β	-1.0	1.0	-2.0	-4.0	3.0	1.0	1.0	13	-11	1.7
δ	0.3	0.2	0.8	0.1	0.3	0.3	1.0	0.5	1.0	0.9
k	2.0	1.0	0.5	3.0	6.0	3.0	1.2	3.0	2.0	1.0
η	0.1	0.1	0.2	0.2	0.2	0.3	0.5	0.2	0.5	0.2
λ	0.1	0.2	0.4	0.4	0.5	0.6	1.7	0.8	0.9	2.0
y_0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

With ten different parameter sets for each model we compute the estimated time required for the calibration of an individual parameter set as well as the value of the objective function and the absolute error in each parameter. The absolute error is defined as:

$$\text{error} = |\text{estimated parameter value - true parameter value}|$$

For the PSO optimization we experimented with different combinations of swarm sizes and generation limits and found that swarms of size 50 particles and 30, 50, 30, and 50 generations for the Heston, Bates, BNS and NIG-CIR models respectively lead to optimal convergence. Unless otherwise stated the social and cognitive attraction parameters for PSO were set at 0.5. Additionally, the default value of the particles' velocity is 0.5.

The paremeter settings for the GA algorithm are listed in the code section.

5.1.1 Constraints

We require that variance is non-negative, correlation between -1 and 1 and parameters like η, θ, β are also non-negative. There are three distinct methods of enforcing constraints with PSO: penalize, absorb and nearest [23]. The first method assigns a very high objective function value to any particle which wanders outside of the feasible region (defined for every model in the following sub-sections) thus attracting it back to the feasible region, the second absorbs stray particles into the boundary of the feasible region according to the direction of their velocities while the third absorbs the particles into the boundary nearest to their current position. For all four models we use the penalize method.

5.2 Results

All computations were carried out on an Intel Core i5 - 2450, 2.5GHz processor in MATLAB. Unless otherwise stated, the calibration times reported in this section are for non-vectorized objective functions. Vectorizing the objective functions used for calibration significantly speeds up calibration times.

5.2.1 Results for the Heston model

The feasible region for the Heston model is defined as follows:

$$\sigma_0 \in [0.01, 150], \qquad \eta \in [0.01, 10]$$
$$\rho \in [-1, 1], \qquad k \in [0.01, 10]$$
$$\theta \in [0.01, 10]$$

The calibration results for the ten parameter sets from the hybrid PSO + `fmincon` algorithm and `fmincon` alone are shown in table 5.1 and 5.2 respectively. In the tables, F denotes the value of the objective function while the rest of the reported values denote the absolute errors in the model parameters and the overall calibration time.

	1	2	3	4	5	6	7	8	9	10
σ_0	5.27E-08	6.71E-08	1.37E-04	8.93E-08	4.15E-08	3.08E-09	1.80E-07	8.67E-08	1.44E-09	1.11E-07
η	9.04E-08	1.33E-06	4.99E-04	1.09E-07	3.36E-06	3.18E-06	1.09E-06	2.71E-07	1.97E-08	4.30E-07
ρ	1.36E-07	4.14E-07	6.29E-02	2.36E-08	1.27E-06	2.84E-07	6.64E-08	3.74E-06	4.05E-07	3.69E-06
k	4.45E-06	1.82E-06	4.91E-02	1.34E-05	3.24E-06	2.05E-06	1.77E-07	3.42E-06	9.37E-07	4.79E-06
θ	1.03E-06	1.95E-08	3.62E-02	7.04E-07	1.68E-06	6.20E-07	3.61E-06	8.21E-06	8.79E-07	6.50E-06
F	3.95E-06	9.01E-06	3.70E-02	8.62E-06	4.96E-06	6.14E-06	1.02E-05	7.97E-06	1.32E-06	4.13E-06
time	68.8	67.9	65.4	72.2	72.9	79.9	75.2	64.3	67.2	70.8

TABLE 5.1: PSO calibration of the Heston model

	1	2	3	4	5	6	7	8	9	10
σ_0	6.83E-08	2.15E-08	1.46E-04	2.56E-07	5.31E-08	7.01E-09	1.42E-07	4.89E-08	7.63E-08	1.52E-08
η	8.68E-07	1.53E-06	5.35E-04	2.23E-08	3.97E-08	2.02E-06	3.62E-06	8.16E-08	1.75E-07	1.25E-06
ρ	3.39E-07	5.30E-07	6.68E-02	5.72E-07	1.12E-07	1.29E-07	1.51E-07	4.30E-07	3.83E-06	6.23E-06
k	1.78E-05	1.46E-06	5.25E-02	5.98E-05	6.79E-07	1.60E-06	3.73E-06	1.02E-06	6.87E-06	6.29E-06
θ	1.42E-06	6.65E-07	3.86E-02	2.30E-05	6.13E-07	2.63E-07	4.59E-06	1.26E-06	7.25E-06	1.22E-05
F	1.32E-05	4.75E-06	3.95E-02	2.44E-05	5.57E-06	8.16E-06	1.12E-05	3.20E-06	4.31E-06	7.32E-06
time	21.95	18.35	15.94	16.64	18.85	16.61	27.50	15.45	18.48	18.33

TABLE 5.2: `fmincon` calibration of the Heston model

The most important observation we can make is that a perfect calibration is achieved in all ten cases. The worst calibration result occurs for the third parameter set where the objective function value is 3.70E-02 which translates to a relative percentage error of 0.037% over the whole price surface consisting of 273 points. The result is shown graphically in Figure 5.1 where we can see the market and model prices are virtually identical. In fact, the largest absolute difference over the whole surface is 0.0019.

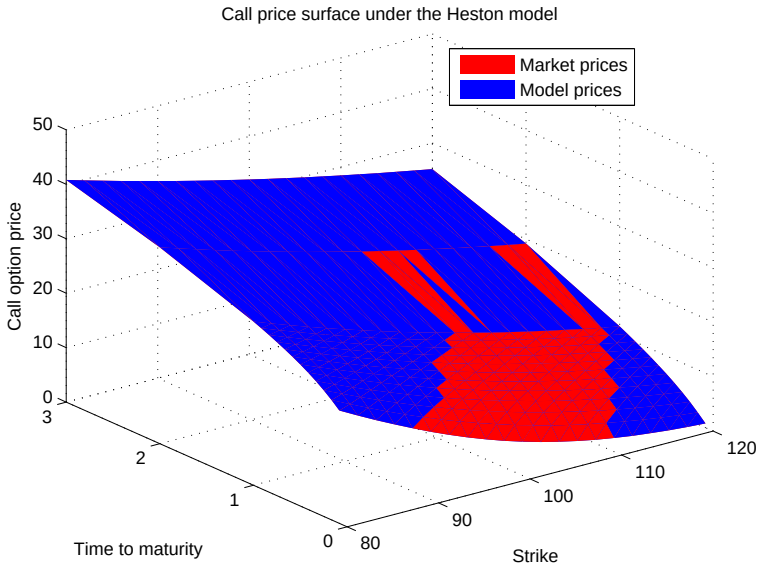


FIGURE 5.1: Calibration result for the Heston model. Parameters used: $\sigma_0 = 0.3, \eta = 0.2, \rho = -0.9, k = 3, \theta = 0.5$

In all ten cases `fmicon` is approximately 3.5 times faster than the PSO+`fmincon` hybrid and produces the same level of calibration accuracy as the hybrid. This is not surprising given the smooth nature of the search space as demonstrated in the left part of Figure 4.1. Of course, by decreasing the size of the swarm from 50 to 25 particles and decreasing the number of generations from 30 to 15 we can decrease the number of objective function evaluations required for the PSO optimization from 1500 to 375 and consequently reduce the calibration time. The results for PSO+`fmincon` and GA+`fmincon` with 25 individuals and 15 populations are included in tables B.1 and B.2 in Appendix B. We note that the calibration times for both hybrid algorithms in

this case are close to the `fmincon` times in Table 5.2 and that GA tends to be faster than PSO while also leading to higher calibration accuracy.

The parameter set considered in the right part of Figure 4.1 is of particular interest because we expect PSO to outperform `fmincon` due to the complex behavior of the objective function. As the results in Table 5.3 demonstrate our intuition is correct and while `fmincon` is faster it produces an objective function value of 9.86 which translates to a 9.86% relative error between market and model prices indicating that calibration of the model with `fmincon` is far from perfect. Clearly, in this case, a heuristic algorithm like PSO and GA must be used.

	PSO	GA	`fmincon`
σ_0	3.99E-08	6.17E-07	1.80E+01
η	1.14E-07	9.18E-07	6.52E-01
ρ	1.12E-07	2.99E-06	5.73E-01
k	3.38E-06	2.67E-06	6.00E-01
θ	3.59E-07	8.45E-06	3.34E+00
F	2.37E-06	1.28E-04	9.86E+00
time	31.853	29.728	18.14

TABLE 5.3: Comparing PSO, GA and `fmincon` (the hybrid algorithms used 15 generations x 25 individuals)

5.2.2 Results for the Bates model

The feasible region for the Bates model is defined as follows:

$$\sigma_0 \in [0.01, 10], \qquad \eta \in [0.01, 2]$$
$$\rho \in [-1, 1], \qquad k \in [0.01, 10]$$
$$\theta \in [0.01, 10], \qquad \lambda \in [0.01, 10]$$
$$\mu_J \in [-1, 10], \qquad v_J \in [0.01, 10]$$

The calibration results for the ten parameter sets from the hybrid PSO + `fmincon` algorithm and `fmincon` alone are shown in table 5.4 and 5.5 respectively.

We can see from the two tables (5.4 and 5.5) that perfect calibration is achievable for the Bates model in the sense of minimizing the objective function F. In all cases but one i.e. parameter set 7 `fmincon` produces more accurate results than its hybrid counterpart and, although the computational times in Table 5.5 vary, they are generally smaller than the times reported in Table 5.4. The results for the GA+`fmincon` hybrid are included in Table 5.6 and are similar to ones obtained with PSO, the immediately noticeable difference being that the computational times are slightly larger for GA.

	1	2	3	4	5	6	7	8	9	10
σ_0	0.0043	0.0047	0.0057	0.0027	0.0079	0.0005	0.0139	0.0068	0.0082	0.0188
η	0.0097	0.0129	0.0063	0.0050	0.0082	0.1066	0.0001	0.0211	0.0241	0.0177
ρ	0.0191	0.0146	0.1074	0.0247	0.0742	0.0037	0.0044	0.0553	0.0289	0.0243
k	0.7706	0.0169	0.0180	0.1423	0.0246	0.1002	0.0321	0.1302	0.0900	0.0303
θ	0.0672	0.0043	0.0369	0.3323	0.0401	0.0157	0.0284	0.1140	0.0677	0.0059
λ	0.0669	0.2111	0.1447	0.1411	0.1896	0.0554	0.6520	0.0460	0.0494	1.3486
μ_J	0.0382	0.0638	0.3315	0.0346	0.9000	0.0429	0.0618	0.0047	0.0035	0.0800
v_J	0.0135	0.0539	0.0830	0.3888	1.5338	0.0542	0.0609	0.1624	0.1797	0.0758
F	0.0815	0.0180	0.0420	0.0717	0.4112	0.0405	0.0118	0.0419	0.0474	0.0174
time	188.31	147.86	159.76	105.98	97.78	186.00	135.84	187.61	187.27	172.55

TABLE 5.4: PSO calibration of the Bates model

	1	2	3	4	5	6	7	8	9	10
σ_0	0.0099	0.0009	0.0022	0.0002	0.0035	0.0004	0.0077	0.0049	0.0030	0.0019
η	0.0122	0.0005	0.0016	0.0003	0.1010	0.0800	0.0827	0.0129	0.0016	0.0102
ρ	0.0125	0.0043	0.1203	0.0001	0.0025	0.0027	0.0009	0.0243	0.0060	0.0123
k	0.5604	0.0015	0.1263	0.0029	0.1031	0.0672	0.2365	0.0697	0.0121	0.0488
θ	0.0713	0.0002	0.0869	0.0006	0.0454	0.0100	0.0544	0.0505	0.0049	0.0450
λ	0.3483	0.0201	0.1349	0.0059	0.1324	0.0398	0.0639	0.0173	0.1083	0.1206
μ_J	0.0703	0.0217	0.0276	0.0021	0.1947	0.0276	0.0157	0.0106	0.0384	0.0721
v_J	0.0505	0.0066	0.0404	0.0014	0.2256	0.0340	0.2237	0.0776	0.0258	0.2144
F	0.0718	0.0082	0.0355	0.0014	0.0270	0.0279	0.0806	0.0270	0.0028	0.0161
time	58.44	104.96	79.73	104.86	105.01	103.81	61.39	104.28	98.68	104.97

TABLE 5.5: `fmincon` calibration of the Bates model

	1	2	3	4	5	6	7	8	9	10
σ_0	0.0017	0.0008	0.0018	0.0067	0.0037	0.0000	0.0054	0.0066	0.0077	0.0019
η	0.0051	0.0401	0.0015	0.0043	0.1017	0.0003	0.0599	0.0204	0.0029	0.0161
ρ	0.0812	0.0254	0.1179	0.0219	0.0042	0.0000	0.0263	0.0352	0.1102	0.0211
k	0.4416	0.0618	0.1151	0.0577	0.1041	0.0002	0.1469	0.1238	0.0407	0.0700
θ	0.0782	0.0172	0.0828	0.0491	0.0457	0.0000	0.0560	0.0917	0.1338	0.0684
λ	0.0656	0.0365	0.1120	0.3430	0.1341	0.0001	0.1650	0.0340	0.1885	0.1346
μ_J	0.1463	0.0099	0.0227	0.0569	0.2141	0.0001	0.3692	0.0146	0.7681	0.1009
v_J	0.2755	0.1424	0.0366	0.0534	0.2169	0.0001	0.6870	0.1332	1.3787	0.3405
F	0.0151	0.0559	0.0344	0.0193	0.0263	0.0001	0.0239	0.0381	0.0678	0.0219
time	195.48	188.95	192.00	123.05	191.79	138.70	191.73	183.76	129.70	189.43

TABLE 5.6: GA calibration of the Bates model

Unlike the Heston model in Bates' case we observe instances of significant absolute errors in the model parameters particularly for μ_J, v_J and k even though the objective function indicates a perfect fit. This is a cause of concern as achieving a perfect calibration with a number of different parameter values which are not equal to those used for generating market prices would indicate a model is misspecified and is therefore a source of risk. The risk stems from the fact that if the values of the model parameters are not fixed then the values of the sensitivities

of the options to changes in the underlying parameters, otherwise known as the Greeks (the delta, vega, theta etc.), are also not fixed. Unstable Greek values, in turn, inevitably lead to detrimental variability in the prices of exotic options which consequently undermines the model's reliability as a hedging and risk management tool.

To reduce the absolute errors in the model parameters and consequently to determine whether the Bates model is stable we can employ one of following of strategies: increase the number of generations used for the PSO and GA parts of the hybrid algorithms, increase the size of the populations used in the PSO and GA parts of the algorithms or, finally, run a series of optimization runs and use the values obtained at the end of each run as the initial values for the run which immediately follows thus resetting the optimization until the model parameters converge to the true values used for generating synthetic market prices. The results of the first two strategies in the case of PSO are included in Tables B.3 and B.4 in Appendix B. It suffices to note that in both cases, with 50 generations of size 100 and 100 generations of size 50, significant absolute errors in the model parameters are still observable and the calibration results are therefore unsatisfactory.

Table 5.7 illustrates the result of resetting GA for three optimization runs with 25 generations of 50 individuals. While the calibration performance has improved in terms of absolute error of model parameters in the case of sets 4 and 5, the errors have not been successfully minimized for the rest of the parameter sets. These findings are consistent with results obtained by Gilli and Schuman [21] and suggest that convergence in the model parameters cannot be obtained for the Bates model confirming its instability.

	1	2	3	4	5	6	7	8	9	10
σ_0	0.0037	0.0014	0.0020	0.0007	0.0005	0.0000	0.0140	0.0162	0.0173	0.0148
η	0.0092	0.0761	0.0014	0.0017	0.0038	0.0004	0.0002	0.0113	0.0136	0.0185
ρ	0.0009	0.0618	0.1198	0.0166	0.0013	0.0000	0.0044	0.0306	0.0269	0.0307
k	0.8047	0.0969	0.1182	0.0602	0.0032	0.0002	0.0324	0.0735	0.0463	0.0050
θ	0.0525	0.0602	0.0847	0.0496	0.0009	0.0000	0.0286	0.0227	0.0137	0.0127
λ	0.0487	0.0715	0.1195	0.0538	0.0160	0.0001	0.6649	1.1500	1.1877	0.2148
μ_J	0.0434	0.2459	0.0238	0.0324	0.0077	0.0001	0.0618	0.0910	0.0856	0.0920
v_J	0.0242	0.4760	0.0375	0.0455	0.0054	0.0001	0.0614	0.0746	0.0742	0.0067
F	0.0769	0.0320	0.0360	0.0123	0.0014	0.0002	0.0119	0.0124	0.0148	0.0290
time	434.74	303.24	417.82	414.61	295.74	400.89	264.67	266.58	428.27	272.30

TABLE 5.7: GA calibration of the Bates model with resetting: 3 optimization runs with 25 generations of 50 individuals

5.2.3 Results for the BNS model

The feasible region for the BNS model is defined as follows:

$$a \in [0.01, 10], \qquad b \in [0.01, 20]$$
$$\rho \in [-1, 1], \qquad \lambda \in [0.01, 10]$$
$$\sigma \in [0.01, 10]$$

The calibration results for the ten parameter sets from the hybrid PSO + `fmincon` algorithm, the hybrid GA + `fmincon` and `fmincon` alone are shown in table 5.8, 5.9 and 5.10 respectively. As in the case of the Bates model, the absolute errors present in the parameters a and b cannot be minimized and a perfect calibration even by resetting the optimization procedure cannot be achieved.

	1	2	3	4	5	6	7	8	9	10
a	0.98962	3.62518	0.00015	0.00050	2.33853	0.73857	0.05617	0.00039	0.00923	0.87015
b	9.96988	8.29061	0.00052	0.01834	1.81014	0.86910	1.29852	0.00722	0.16583	6.57323
ρ	0.03187	0.59602	0.00074	0.00001	0.46003	0.40158	0.00042	0.00005	0.00069	0.03452
λ	0.00314	0.08026	0.00011	0.00000	0.09922	0.02062	0.00001	0.00000	0.00003	0.00163
σ	0.00240	0.00210	0.00001	0.00000	0.00045	0.00013	0.00002	0.00000	0.00002	0.00229
F	0.00846	0.06415	0.00081	0.00019	0.06996	0.01496	0.00003	0.00002	0.00018	0.01786
time	62.96	65.58	82.31	64.72	67.81	84.70	97.42	66.97	67.08	60.04

TABLE 5.8: PSO calibration of the BNS model

	1	2	3	4	5	6	7	8	9	10
a	0.32923	0.85792	0.00016	0.06711	1.36585	1.52270	8.63637	0.24178	0.32135	0.02905
b	3.31642	3.08931	0.00078	2.43999	7.77634	5.76976	10.38252	4.41237	5.78757	0.21851
ρ	0.02214	0.44218	0.00085	0.00194	0.47125	0.48354	0.58156	0.05758	0.04455	0.00251
λ	0.00232	0.02220	0.00016	0.00091	0.14179	0.14238	6.42305	0.00274	0.00324	0.00014
σ	0.00164	0.00155	0.00002	0.00002	0.00055	0.00053	6.71995	0.00183	0.00153	0.00016
F	0.00610	0.02720	0.00093	0.03123	0.09551	0.09790	47.13365	0.02036	0.01347	0.00141
time	35.93	47.85	26.10	17.92	19.16	20.77	26.19	16.44	17.11	21.11

TABLE 5.9: `fmincon` calibration of the BNS model

	1	2	3	4	5	6	7	8	9	10
a	0.98784	3.62538	0.00007	0.00004	0.23193	0.74022	0.71626	0.05447	0.00857	0.87299
b	9.95265	8.29429	0.00033	0.00150	13.29620	0.87755	1.04104	1.00237	0.15433	6.59449
ρ	0.03183	0.59599	0.00038	0.00000	0.41510	0.40170	0.13799	0.00661	0.00062	0.03458
λ	0.00314	0.08021	0.00007	0.00000	0.05545	0.02066	0.00215	0.00011	0.00003	0.00163
σ	0.00240	0.00210	0.00001	0.00000	0.00033	0.00013	0.00040	0.00024	0.00002	0.00229
F	0.00845	0.06412	0.00042	0.00002	0.04111	0.01499	0.05074	0.00225	0.00016	0.01788
time	107.94	98.82	91.15	100.26	103.75	119.52	103.88	96.59	96.73	105.16

TABLE 5.10: GA calibration of the BNS model

5.2.4 Results for the NIG-CIR model

The feasible region for the NIG-CIR model is defined as follows:

$$\alpha \in [0.01, 20], \qquad \beta \in [-19, 19]$$
$$\delta \in [0.01, 10], \qquad k \in [0.01, 10]$$
$$\eta \in [-5, 10], \qquad \lambda \in [0.01, 10]$$

We also impose the additional constraint $-\alpha < \beta < \alpha$ which is a feature of the NIG distribution. The calibration results for the ten parameter sets from the hybrid PSO + `fmincon` algorithm and `fmincon` alone are shown in table 5.11 and 5.12 respectively.

	1	2	3	4	5	6	7	8	9	10
α	3.37529	7.18708	0.00000	10.95581	0.00000	0.00003	0.00004	0.12148	6.35321	3.54400
β	1.66527	0.57618	0.00000	7.65344	0.00000	0.00001	0.00000	0.14127	3.83199	2.20693
δ	0.10615	4.10953	0.00000	0.01937	0.00000	0.00000	0.00000	0.04461	0.88097	0.13824
k	1.27579	8.52029	0.00000	2.52984	0.00003	0.00003	0.00000	0.46476	4.37985	0.95746
η	0.18416	0.10607	0.00000	0.94933	0.00000	0.00000	0.00000	1.02508	4.42714	0.34948
λ	0.30929	3.48553	0.00000	0.13342	0.00000	0.00009	0.00000	1.10782	3.55329	4.83985
F	7.31127	58.79805	0.00001	17.62816	0.00003	0.00004	0.00002	0.15289	218.29362	8.58555
time	338.04	216.21	299.42	318.95	283.46	313.31	268.76	216.87	222.46	248.35

TABLE 5.11: PSO calibration of the NIG-CIR model

	1	2	3	4	5	6	7	8	9	10
α	2.33781	3.29810	4.37934	5.14158	6.32589	11.23704	15.27710	15.11080	12.56837	17.17921
β	2.56523	0.55842	3.57073	5.51615	1.43665	0.54493	0.55420	11.49523	12.57534	0.17135
δ	0.19966	0.29966	0.30034	0.39771	0.19966	0.19965	0.50414	0.00433	0.50035	0.40036
k	1.49993	0.49993	0.00007	2.49948	5.49993	2.49993	0.69905	2.49899	1.49992	0.49992
η	0.39993	0.39993	0.29993	0.29957	0.29993	0.19993	0.00078	0.29919	0.00007	0.29993
λ	0.39980	0.29981	0.19979	0.09822	0.00020	0.10019	1.20317	0.30377	0.40025	2.49980
F	8E+73	7E+74	2E+73	1E+79	5E+74	2E+76	3E+74	1E+80	3E+76	1E+78
time	1.903	1.850	4.201	5.328	7.830	8.327	3.098	260.766	5.522	8.326

TABLE 5.12: `fmincon` calibration of the NIG-CIR model

It is clear from table 5.12 that using `fmincon` to calibrate the NIG-CIR model leads to catastrophic results. The algorithm fails in all ten cases with the best objective function value being 1E+73. The hybrid on the other hand performs well for parameter sets 3, 5, 6, 7 and 8. To improve the results for the rest of the parameter sets we can either increase the number of generations used for PSO or increase the size of the particle swarm. In either case, the PSO algorithm has to perform a larger number of functional evaluations which leads to higher calibration time. Table 5.13 shows the result of increasing the number of generations from 50 to 100 for parameter sets 1,2,4,9 and 10:

	1	2	4	9	10
α	3.378982	0.000003	0.000003	6.037131	0.000011
β	1.665694	0.000001	0.000001	2.622387	0.000000
δ	0.106450	0.000000	0.000000	5.862407	0.000001
k	1.269597	0.000023	0.000003	3.777842	0.000000
η	0.189093	0.000016	0.000000	0.263967	0.000000
λ	0.513943	0.000025	0.000002	2.655327	0.000000
F	7.312243	0.000046	0.000033	141.984225	0.000009
time	563.39	504.38	476.61	428.50	490.54

TABLE 5.13: PSO calibration of the NIG-CIR model with 100 generations

	1	9
α	0.000011	3.877540
β	0.000001	0.295764
δ	0.000001	0.546877
k	0.000003	1.549361
η	0.000001	0.117242
λ	0.000079	2.977533
F	0.000041	86.182563
time	488.20	502.9

TABLE 5.14: PSO calibration of the NIG-CIR model with a swarm of size 100

We can see that while the computational time has roughly doubled perfect calibration is achieved for parameter sets 2,4 and 10.

Table 5.14 shows the effect of increasing the swarm size to 100 for parameter sets 1 and 9. We conclude from the results in the table that the only parameter set for which a perfect calibration has not been achieved is 9. To understand what is happening in this specific case we examine the corresponding price surface and implied volatility surface shown in Figure 5.2.

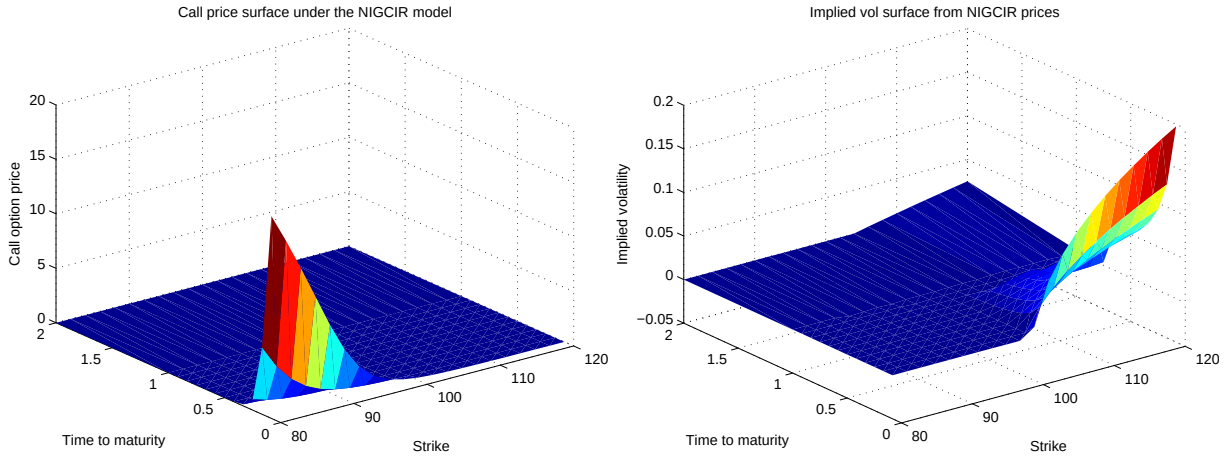


FIGURE 5.2: Price surface and implied vol surface for the NIG-CIR model with parameter set 9. Parameter values: $\alpha = 13, \beta = -11, \delta = 1, k = 3, \eta = 0.5\lambda = 0.9 y_0 = 1$

Intuitively, after examining the two surfaces shown in Figure 5.2 we can imagine that the search space for parameter set 9 is quite flat. In this case, all the particles in the swarm most likely tend to oscillate around their original locations as the global best of the swarm is not significantly better than their personal best. Thus, one way to minimize the objective function is to run a large number of calibrations and hope that one of the of the particles lands at or near the global minimum by chance. Of course, a much simpler solution is to just isolate the part of the prices surface where prices are actually non zero, in this case for maturities in the range [0, 0.4] and strikes [80, 100], and perform a calibration only over this subset of the surface.

Results from the GA calibration are listed in Table B.5. We see that PSO is faster than GA when 50 generations of 50 individuals are used. Additionally, GA optimization fails for parameter sets 5,7,8,9 and 10. The result of increasing the number of generations and number individuals to 100 is show in Table B.6. We experimented with a large number of algorithm settings e.g. selection, elitism, mutation, crossover etc. but, unlike the PSO case, were unable to calibrate parameter sets 5, 8, and 10 perfectly.

The most important observation which can be made is that unlike the Bates model when using NIG-CIR we can achieve perfect calibration in terms of the objective function as well as in terms of the absolute error in the model parameters. These findings suggest that NIG-CIR contains no model risk and is therefore a better pricing tool than its jump process based counterpart.

Section 6
Conclusion

We have demonstrated how to perform efficient calibration of four stochastic volatility models: the Heston, Bates, Barndorff-Nielsen-Shephard (BNS) and the Normal Inverse Gaussian Cox-Ingersoll-Ross (NIG-CIR) model. Our findings suggest that MATLAB's built in `quad` function with an `interior-point` setting can be a computationally efficient calibration tool as long as the search space of the objective function exhibits smooth behavior. In more general cases, however, hybrid schemes such as particle swarm optimization (PSO) combined with an `interior-point` algorithm strongly outperform standard techniques. Due to the complexity of the search space, `quad` cannot be applied to the calibration of the stochastic time changed NIG-CIR model making the use of heuristic optimization schemes such as PSO and genetic algorithms (GAs)s necessary. Most importantly we have shown that the Heston and NIG-CIR models can be perfectly calibrated not only in the sense of minimizing the relative percentage error between market prices and model prices but also in the sense of minimizing the absolute error between true parameters and model parameters indicating the two models are well specified and risk free and making them suitable for pricing, hedging and risk-management of exotic derivatives.

Appendix A
The behavior of Π_2 for the four models

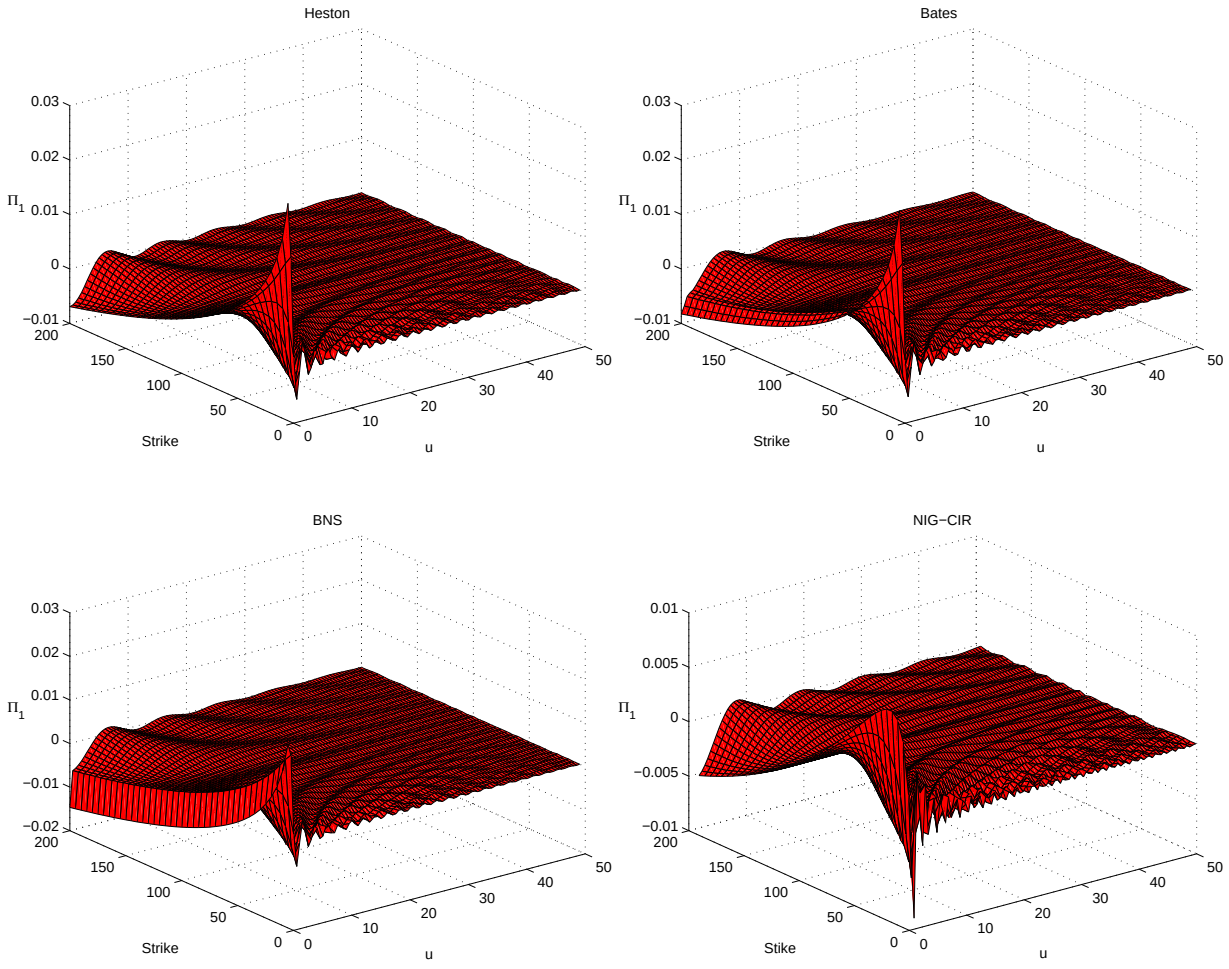


FIGURE A.1: The behavior of Π_2 for the four models

Appendix B
Results

B.1 Calibration results of the Heston model

B.1.1 PSO and GA results

	1	2	3	4	5	6	7	8	9	10
σ_0	2.84E-08	1.30E-07	1.37E-04	1.28E-07	1.88E-08	4.60E-08	1.46E-07	9.20E-08	5.88E-08	3.98E-08
η	2.98E-07	2.81E-06	4.96E-04	6.26E-08	1.26E-06	8.72E-07	9.37E-08	2.23E-07	1.94E-07	1.93E-07
ρ	1.20E-07	3.54E-07	6.27E-02	1.42E-07	1.05E-06	9.98E-07	3.11E-08	3.30E-06	1.26E-06	5.17E-07
k	3.72E-06	4.40E-06	4.91E-02	2.07E-05	1.90E-06	5.73E-07	5.05E-07	3.58E-06	4.81E-07	9.36E-08
θ	1.81E-06	3.78E-07	3.60E-02	5.17E-06	7.80E-07	2.54E-06	2.04E-06	7.37E-06	2.87E-06	1.17E-06
F	7.16E-06	1.79E-05	3.68E-02	9.93E-06	8.20E-06	1.81E-05	7.22E-06	7.07E-06	3.97E-06	2.35E-06
time	31.322	29.977	27.119	28.478	30.848	39.719	34.469	29.158	30.896	36.847

Table B.1: PSO calibration of the Heston model: 15 generations of 25 partciles

	1	2	3	4	5	6	7	8	9	10
σ_0	2.10E-07	4.08E-09	1.29E-07	1.92E-08	3.63E-08	5.51E-08	2.03E-07	5.00E-08	8.45E-08	5.66E-08
η	9.64E-07	5.80E-07	2.15E-07	1.30E-06	5.37E-06	3.25E-06	2.86E-06	1.78E-07	2.63E-07	3.41E-07
ρ	1.16E-07	4.19E-07	1.46E-06	3.49E-06	1.51E-06	2.78E-08	6.04E-08	3.13E-06	3.00E-06	3.14E-06
k	2.73E-05	1.22E-06	5.03E-06	1.68E-05	3.45E-06	3.02E-06	2.34E-06	6.27E-06	4.76E-06	3.38E-06
θ	2.71E-06	3.79E-07	5.80E-08	5.38E-05	3.42E-06	9.54E-07	5.20E-06	6.35E-06	5.73E-06	5.13E-06
F	1.81E-05	1.34E-05	3.67E-05	8.05E-05	1.24E-05	9.89E-06	1.35E-05	5.49E-06	7.11E-06	4.40E-06
time	23.401	28.796	28.640	28.364	27.087	25.884	26.418	26.028	27.567	23.620

Table B.2: GA calibration of the Heston model: 15 generations of 25 genomes

B.2 Calibration results of the Bates model

B.2.1 PSO calibration of the Bates model

	1	2	3	4	5	6	7	8	9	10
σ_0	0.0017	0.0008	0.0018	0.0067	0.0037	0.0000	0.0054	0.0066	0.0077	0.0019
η	0.0051	0.0401	0.0015	0.0043	0.1017	0.0003	0.0599	0.0204	0.0029	0.0161
ρ	0.0812	0.0254	0.1179	0.0219	0.0042	0.0000	0.0263	0.0352	0.1102	0.0211
k	0.4416	0.0618	0.1151	0.0577	0.1041	0.0002	0.1469	0.1238	0.0407	0.0700
θ	0.0782	0.0172	0.0828	0.0491	0.0457	0.0000	0.0560	0.0917	0.1338	0.0684
λ	0.0656	0.0365	0.1120	0.3430	0.1341	0.0001	0.1650	0.0340	0.1885	0.1346
μ_J	0.1463	0.0099	0.0227	0.0569	0.2141	0.0001	0.3692	0.0146	0.7681	0.1009
v_J	0.2755	0.1424	0.0366	0.0534	0.2169	0.0001	0.6870	0.1332	1.3787	0.3405
F	0.0151	0.0559	0.0344	0.0193	0.0263	0.0001	0.0239	0.0381	0.0678	0.0219
time	195.48	188.95	192.00	123.05	191.79	138.70	191.73	183.76	129.70	189.43

Table B.3: PSO calibration of the Bates model: 50 generations with swarm size of 100

	1	2	3	4	5	6	7	8	9	10
σ_0	0.0017	0.0008	0.0018	0.0067	0.0037	0.0000	0.0054	0.0066	0.0077	0.0019
η	0.0051	0.0401	0.0015	0.0043	0.1017	0.0003	0.0599	0.0204	0.0029	0.0161
ρ	0.0812	0.0254	0.1179	0.0219	0.0042	0.0000	0.0263	0.0352	0.1102	0.0211
k	0.4416	0.0618	0.1151	0.0577	0.1041	0.0002	0.1469	0.1238	0.0407	0.0700
θ	0.0782	0.0172	0.0828	0.0491	0.0457	0.0000	0.0560	0.0917	0.1338	0.0684
λ	0.0656	0.0365	0.1120	0.3430	0.1341	0.0001	0.1650	0.0340	0.1885	0.1346
μ_J	0.1463	0.0099	0.0227	0.0569	0.2141	0.0001	0.3692	0.0146	0.7681	0.1009
v_J	0.2755	0.1424	0.0366	0.0534	0.2169	0.0001	0.6870	0.1332	1.3787	0.3405
F	0.0151	0.0559	0.0344	0.0193	0.0263	0.0001	0.0239	0.0381	0.0678	0.0219
time	195.48	188.95	192.00	123.05	191.79	138.70	191.73	183.76	129.70	189.43

Table B.4: GA calibration of the Bates model: 100 generations with swarm size of 50

B.3 Calibration results of the NIG-CIR model

B.3.1 GA calibration of the NIG-CIR model

	1	2	3	4	5	6	7	8	9	10
α	0.00000	0.00000	0.00001	0.00000	5.23404	0.00004	2.62380	15.23960	10.93020	3.54474
β	0.00000	0.00000	0.00000	0.00000	3.01287	0.00001	1.47919	14.00875	8.91838	2.20738
δ	0.00000	0.00000	0.00000	0.00000	0.16137	0.00000	0.10714	0.47404	0.18205	0.13815
k	0.00000	0.00001	0.00001	0.00002	0.89723	0.00003	0.52087	0.70591	1.20373	0.95868
η	0.00000	0.00000	0.00001	0.00000	0.34629	0.00000	0.47979	0.27764	0.09858	0.34928
λ	0.00008	0.00000	0.00000	0.00000	0.39826	0.00009	3.69774	4.93223	2.89856	0.83803
F	0.00002	0.00001	0.00001	0.00006	23.83234	0.00005	3.13711	6.36756	215.22945	8.58539
time	458.70	454.64	467.69	463.56	273.56	470.48	472.80	309.02	236.14	378.92

TABLE B.5: GA calibration of the NIG-CIR model: 50 generations of 50 individuals

	5	7	8	9	10
α	6.51693	0.00003	14.93334	11.68388	3.54522
β	3.51693	0.00000	14.29820	9.56854	2.20764
δ	0.18530	0.00000	0.48415	0.29572	0.13818
k	1.72293	0.00000	0.06784	0.09902	0.95853
η	0.36885	0.00000	0.27466	2.43171	0.34928
λ	4.34227	3.40000	1.15361	2.80254	0.83811
F	26.86186	0.00002	6.64212	216.45433	8.58541
time	751.68	714.31	722.01	836.97	775.31

TABLE B.6: GA calibration of the NIG-CIR model: 100 generations of 100 individuals

Appendix C
Code

Appendix C contains sample code ranging from procedures for evaluating the characteristic functions of the models, functions for generating price surfaces with the Gauss-Legendre quadrature rule as well as scripts for calibrating the models. All code included here is in MATLAB but a C++ version can be obtained by contacting the author at yavorkovachev2011@gmail.com

C.1 Charactersitc functions

Below are the characteristic functions for the Heston, Bates and BNS models. The characteristic exponent for the NIG process, the characteristic function for the CIR process and characteristic function for the stochastic time changed NIG-CIR process are also included.

```
function cf = cfHeston(om,S,tau,r,q,v0,vT,rho,k,sigma)
d   = sqrt((rho * sigma * 1i*om - k).^2 + sigma^2 * (1i*om + om .^ 2));
g2  = (k - rho*sigma*1i*om - d) ./ (k - rho*sigma*1i*om + d);
cf1 = 1i*om .* (log(S) + (r - q) * tau);
cf2 = vT * k / (sigma^2) * ((k - rho*sigma*1i*om - d) * tau - 2 * log((1 - g2 ↩
    .* exp(-d * tau)) ./ (1 - g2)));
cf3 = v0 / sigma^2 * (k - rho*sigma*1i*om - d) .* (1 - exp(-d * tau)) ./ (1 - ↩
    g2 .* exp(-d * tau));
cf  = exp(cf1 + cf2 + cf3);
end
```

```
function cf = cfBates(om,S,tau,r,q,v0,vT,rho,k,sigma,lambda,muJ,vJ)
d   = sqrt((rho * sigma * 1i*om - k).^2 + sigma^2 * (1i*om + om .^ 2));
%
g2  = (k - rho*sigma*1i*om - d) ./ (k - rho*sigma*1i*om + d);
%
cf1 = 1i*om .* (log(S) + (r - q) * tau);
cf2 = vT * k / (sigma^2) * ((k - rho*sigma*1i*om - d) * tau - 2 * log((1 - g2 ↩
    .* exp(-d * tau)) ./ (1 - g2)));
```

```
cf3 = v0 / sigma^2 * (k - rho*sigma*1i*om - d) .* (1 - exp(-d * tau)) ./ (1 - g2 .* exp(-d * tau));
% jump
cf4 = -lambda*muJ*1i*tau*om + lambda*tau*(  (1+muJ).^(1i*om) .* exp( vJ*(1i*om/2) .* (1i*om-1) )-1  );
cf  = exp(cf1 + cf2 + cf3 + cf4);
end
```

```
function cf = cfBNS(om,S,tau,r,q,alpha,beta,rho,lambda,sigma)
f1  = 1i*om*rho - (1/lambda)*(om.^2 + 1i*om)*(1 - exp(-lambda*tau))/2;
f2  = 1i*om*rho - (1/lambda)*(om.^2 + 1i*om)/2;
cf1 = 1i*om.*(log(S) + (r - q - alpha*lambda*rho*(1/(beta-rho))) * tau);
cf2 = (1/lambda)*(om.^2 + 1i*om)*(1 - exp(-lambda*tau))*(sigma^2)/2;
cf3 = alpha*(1./(beta-f2)).*(beta.*log((beta-f1)./(beta-1i*om*rho))+f2*lambda*tau);
temp = cf1 - cf2 + cf3;
cf  = exp(temp);
end
```

```
function ce = ceNIG(xi, alpha, beta, delta)
ce = delta.*(sqrt(alpha^2 - beta^2) - sqrt(alpha^2 - (beta + 1i*xi).^2));
end
```

```
function cf = cfCIR(om,tau,k,eta,lambda,y0)
gamma = sqrt(k^2 - 2*lambda^2*1i.*om);
cf = exp(k^2*eta*tau/lambda^2)*exp(2*y0*1i.*om./(k+gamma.*coth(gamma*tau/2)))./...
     (cosh(gamma*tau/2) + k*sinh(gamma*tau/2)./gamma).^(2*k*eta/lambda^2);
end
```

```
function cf = cfNIGCIR(om,S,r,q,tau,alpha,beta,delta,k,eta,lambda,y0)
cf = exp(1i.*om*((r-q)*tau+log(S))).*cfCIR(-1i*ceNIG(om,alpha,beta,delta),y0,tau,k,eta,lambda)./...
     (cfCIR(-1i*ceNIG(-1i,alpha,beta,delta),y0,tau,k,eta,lambda).^(1i*om));
end
```

C.2 Computing prices with the Heston model

```
function [X, tau, prices ] = price_surface_heston(S,X,tau,r,q,v0,vT,rho,k,sigma)
% This function computes a call option price surface of size X strikes by
```

```
% tau maturities under the Heston model. The value of each option in the
% price surface is computed according to the formula of Baskhi and Madan
% which uses the characteristic function cfHeston. The value of Pi_1 and
% Pi_2 are computing using a Gauss-Legendre quadrature rule where the nodes
% and weights are determined by findinding the eigenvalues of the recursion
% matrix A.
%
% S     = spot price
% X     = vector of strikes
% tau   = vector of maturities
% r     = riskfree rate
% q     = dividend yield
% v0    = initial variance
% vT    = long run variance
% rho   = correlation
% k     = speed of mean reversion
% sigma = volatility of volatility


% set up weights for Gauss-Legendre
num_nodes = 100;
m = num_nodes; a = 0; b = 200; h = (b-a)/m;
n = num_nodes + 1;
aux = 1./sqrt(4 - (1:(n-1)).^(-2));
% construct matrix A
A = diag(aux,1) + diag(aux, -1);
% find eigenvalues
[V,D] = eig(A);
x = diag(D);
[x,j] = sort(x);
om = (2*V(1,j).^2);
%shift weights to cover interval 0 to b instead of the standard -1 to 1
a0 = -1; b0 = 1;
x = ((b-a)*x + a*b0 - b*a0)/(b0 - a0);
om = om*(b-a)/(b0 - a0);
assignin('base', 'abcissas', x);

n = length(tau);
m = length(X);
prices = zeros(n,m);
for counter1=1:n
        cfHeston_values = cfHeston(x,S,tau(counter1),r,q,v0,vT,rho,k,sigma);
        cfHeston_values1 = cfHeston(x-1i,S,tau(counter1),r,q,v0,vT,rho,k,sigma)↩
    ;
      for counter2=1:m
        vP1 = P1(x,S,X(counter2),tau(counter1),r,q,v0,vT,rho,k,sigma,↩
    cfHeston_values1);
```

```
48          vP2 = P2(x,S,X(counter2),tau(counter1),r,q,v0,vT,rho,k,sigma,↩
    cfHeston_values);
49          vP1 = om*vP1;
50          vP2 = om*vP2;
51          vP1 = 0.5 + 1/pi * vP1;
52          vP2 = 0.5 + 1/pi * vP2;
53          prices(counter1,counter2) = exp(-q * tau(counter1)) * S * vP1 - exp(-r ↩
    * tau(counter1)) * X(counter2) * vP2;
54          if prices(counter1,counter2) < 0
55              prices(counter1,counter2) = 0;
56          end
57      end
58  end
59  X = repmat(X, length(tau), 1);
60  tau = repmat(tau', 1, size(X,2));
61  end
62  %
63  function p = P1(om,S,X,tau,r,q,v0,vT,rho,k,sigma, cfHeston_values)
64  i=1i;
65  p = real(exp(-i*log(X)*om) .* cfHeston_values ./ (i * om * S * exp((r-q) * tau)↩
    ));
66  end
67  %
68  function p = P2(om,S,X,tau,r,q,v0,vT,rho,k,sigma, cfHeston_values)
69  i=1i;
70  p = real(exp(-i*log(X)*om) .* cfHeston_values ./ (i * om));
71  end
72  %
73  function cf = cfHeston(om,S,tau,r,q,v0,vT,rho,k,sigma)
74  d   = sqrt((rho * sigma * 1i*om - k).^2 + sigma^2 * (1i*om + om .^ 2));
75  g2  = (k - rho*sigma*1i*om - d) ./ (k - rho*sigma*1i*om + d);
76  cf1 = 1i*om .* (log(S) + (r - q) * tau);
77  cf2 = vT * k / (sigma^2) * ((k - rho*sigma*1i*om - d) * tau - 2 * log((1 - g2 ↩
    .* exp(-d * tau)) ./ (1 - g2)));
78  cf3 = v0 / sigma^2 * (k - rho*sigma*1i*om - d) .* (1 - exp(-d * tau)) ./ (1 - ↩
    g2 .* exp(-d * tau));
79  cf  = exp(cf1 + cf2 + cf3);
80  end
```

Objective function

```
1  function [value] = obj_function(market_prices, model_prices )
2  diff = abs(model_prices - market_prices);
3  %market_prices(market_prices == 0) = 1;
4  diff = diff./market_prices;
5  value = sum(sum(diff));
6  end
```

The following script computes an implied volatility surface from a given price surface under the Heston model and illustrates the objective function when the speed of mean reversion and volatility of volatility vary.

```
%% Generate a sample price surface
clear;clc;
S       = 100;
q       = 0.00;
r       = 0.1;
X       = 80:2:120;
%tau     = [1/12 2/12 3/12 4/12 5/12 6/12 7/12 8/12 9/12 10/12 11/12 1 2];
tau     = [1/12 3/12 6/12 9/12 1 2 3];

v0      = 0.3^2;    % current variances
vT      = 0.3^2;    % long-run variance
rho     = -0.3;     % correlation
k       = 2;        % mean reversion speed
sigma   = 1.5;      % vol of vol

tic, [X,Y,Z] = price_surface_heston(S,X,tau,r,q,v0,vT,rho,k,sigma);
fprintf('Heston price surface generated:\t required time: %f seconds\n',toc)
disp(' -- ')
surf(X,Y,Z, 'LineStyle', 'none')
title('Call price surface under the Heston model')
ylabel('Time to maturity')
xlabel('Strike')
zlabel('Call option price')

%Generate an implied volatility surface
[n,m] = size(Z);
impl_vol = zeros(n,m);
tic
for i=1:n
    for j=1:m
        impl_vol(i,j) = fzero(@(x) Z(i,j) - callBSM(S, X(i,j), Y(i,j), r, q, x)↩
    , 0.5);
    end
end
fprintf('Implied vol surface from Heston prices generated:\t required time: %f ↩
    seconds\n',toc)
disp(' -- ')

figure()
surf(X,Y,impl_vol, 'LineStyle', 'none')
title('Implied vol surface from Heston prices')
ylabel('Time to maturity')
xlabel('Strike')
```

```
42 zlabel('Implied volatility')
43
44 %% Generate a sample search space illustrating the objective function when
45 %speed of mean reversion and vol of vol vary
46 clear; clc;
47 S       = 100;
48 q       = 0.00;
49 r       = 0.02;
50 X       = 80:2:120;
51 %tau     = [1/12 2/12 3/12 4/12 5/12 6/12 7/12 8/12 9/12 10/12 11/12 1 2];
52 tau     = [1/12 3/12 6/12 9/12 1 2 3];
53
54 v0      = 100;      % current variance
55 vT      = 0.3;      % long-run variance
56 rho     = -0.3;     % correlation
57 k       = 2;        % mean reversion speed
58 sigma   = 1.5;      % vol of vol
59
60 [X1,Y1,market_prices] = price_surface_heston(S,X,tau,r,q,v0,vT,rho,k,sigma);
61
62 step_size =0.25;
63 k = 0.1:step_size:10;
64 sigma = 0.1:step_size:10;
65 m= length(sigma);
66 n= length(k);
67 obj_fun_surface = zeros(n,m);
68 tic;
69 for counter1=1:m
70     for counter2=1:n
71         [~,~,model_prices] = price_surface_heston(S,X,tau,r,q,v0,vT,rho,k(↩
   counter2),sigma(counter1));
72         obj_fun_surface(counter2,counter1) = objective_function(market_prices, ↩
   model_prices);
73     end
74 end
75 fprintf('Time for computing the objective function: \t %f \n', toc)
76 k = repmat(k', 1, m);
77 sigma= repmat(sigma, n, 1);
```

Computing the implied volatility surface can be done with the following function as well.

```
1 function [X,Y,implied_vol] = implied_vol_heston(S,X,tau,r,q,v0,vT,rho,k,sigma)
2 % Computes an implied volatility surface from a price surface generated with
3 % the Heston model
4 [X,Y,Z] = price_surface_heston(S,X,tau,r,q,v0,vT,rho,k,sigma);
5 [n,m] = size(Z);
6 implied_vol = zeros(n,m);
```

```
for i=1:n
    for j=1:m
        implied_vol(i,j) = fzero(@(x) Z(i,j) - callBSM(S, X(i,j), Y(i,j), r, q,↩
    x), 0.5);
    end
end

end
```

```
function call = callBSM(S,X,tau,r,q,sigma)
% Standard Black Scholes formula
% S     = spot
% X     = strike
% tau   = time to mat
% r     = riskfree rate
% q     = dividend yield
% sigma = volatility
%
d1 = ( log(S/X) + (r-q+sigma^2/2)*tau ) / (sigma*sqrt(tau));
d2 = d1 - sigma*sqrt(tau);

call = S*exp(-q*tau)*normcdf(d1,0,1) - X*exp(-r*tau)*normcdf(d2,0,1);
```

C.3 Calibrating the Heston model

Objective function

```
function [value] = objective_function_heston(S,X, tau, r, q, v0, vT, rho, k, ↩
    sigma, x)
% Objective funcion needed for the PSO calibration
[~,~,market_prices] = price_surface_heston(S,X,tau,r,q,v0,vT,rho,k,sigma);
%price_surface(S,X,tau,r,q,v0,vT,rho,k,sigma)
%model_prices = price_surface(S,X,tau,r,q,x(1), x(2), x(3), x(4), x(5));
[~,~,model_prices] =  price_surface_heston(S,X,tau,r,q,x(1), x(2), x(3), x(4), ↩
    x(5));
diff = abs(model_prices - market_prices);
market_prices(market_prices == 0) = 1;
diff = diff./market_prices;
value = sum(sum(diff));
end
```

To use this calibration script you must provide a PSO function. I used my own version which I wrote as well as the version from Ebbesen, Kiwitz and Guzzella [23] which handles constraints particularly well. It can be downloaded from:

http://www.idsc.ethz.ch/Downloads/DownloadFiles

The script currently uses a for loop iterating over the ten sets of parameter values and storing the calibration results along with the absolute errors. To run it, the user can create an array of model parameters called `heston_parameters` and save it as a `model_parameters.mat` file in the same directory as the script. Otherwise, after commenting out the `load model_parameters;` line and the four lines after it, the loop can be removed or set to 1 and the model parameter values can be supplied manually. The second part of the script plots, for the parameter set provided in the first part, the synthetic and calibrated price surface on top of each other. Calibration of other models is achieved by modifying the objective function.

```
1  % EXAMPLE_Heston    This script demonstrates PSO, GA and fmincon
2  % applied to the Heston problem
3  %
4  %    find x = [v0,vT,rho,k,sigma]
5  %
6  %    min: f(x) = |market_prices - model_prices|/market_prices
7  %
8  %    subject to:   0.01 <= v0    <= 150
9  %                  0.01 <= vT    <= 2
10 %                  -1   <= rho   <= 1
11 %                  0.01 <= k     <= 10
12 %                  0.01 <= sigma <= 10
13 %
14 %
15 % PSO Author(s):    Soren Ebbesen, 14-Sep-2011
16 %                   sebbesen@idsc.mavt.ethz.ch
17 %
18 % Version:    1.1 (19-Jun-2013)
19 %
20 % Institute for Dynamic Systems and Control, Department of Mechanical and
21 % Process Engineering, ETH Zurich
22 %
23 % This Source Code Form is subject to the terms of the Mozilla Public License,
24 % v. 2.0. If a copy of the MPL was not distributed with this file, You can
25 % obtain one at http://mozilla.org/MPL/2.0/.
26     clear;clc;
27     load model_parameters
28     [n,m] = size(heston_parameters);
29     results_hest_pso = zeros(n,m);
30     results_hest_fmincon = zeros(n,m);
31     results_hest_ga = zeros(n,m);
```

```
32  for i= 1:m
33      addpath(genpath('functions'))
34      % EXAMPLE 3: Heston model calibration
35      % Options
36      options = pso;
37      options.PopulationSize = 25;
38      %options.Vectorized = 'on';
39      options.BoundaryMethod = 'penalize';
40      %options.PlotFcns = @psoplotbestf;
41      options.PlotFcns = [];
42      options.Display = 'iter';
43      options.Vectorized      = 'off';
44      options.HybridFcn = @fmincon;
45      options.StallGenLimit   = 100;
46      options.Generations     = 15;
47
48      S       = 100;
49      q       = 0.00;
50      r       = 0.02;
51      X       = 80:2:120;
52      tau     = [1/12 2/12 3/12 4/12 5/12 6/12 7/12 8/12 9/12 10/12 11/12 1 2];
53      % Set model parameters manually
54      % ------------------------------------------------------------------
55       v0      = 10^2;     % current variance
56       vT      = 0.3^2;    % long-run variance (theta in paper)
57       rho     = -0.3;     % correlation
58       k       = 2;        % mean reversion speed (kappa in paper)
59       sigma   = 1.5;      % vol of vol
60      % ------------------------------------------------------------------
61      v0      = heston_parameters(1,i);     % current variance
62      vT      = heston_parameters(2,i);     % long-run variance (theta in paper)
63      rho     = heston_parameters(3,i);     % correlation
64      k       = heston_parameters(4,i);     % mean reversion speed (kappa in ↩
        paper)
65      sigma   = heston_parameters(5,i);     % vol of vol
66
67      %Problem
68      problem = struct;
69      problem.fitnessfcn = @(x)objective_function_heston(S,X, tau, r, q, v0, vT,↩
         rho, k, sigma, x);
70      problem.nvars = 5;
71      problem.Aineq = [];
72      problem.bineq = [];
73      problem.lb = [0.01, 0.01, -1,  0.01, 0.01];
74      problem.ub = [150, 2, 1, 10,   10];
75      problem.nonlcon = @mynonlcon;
76      problem.options = options;
77      %%% PSO optimization
```

```
78      tic
79      [x_pso,fval,exitflag,output] = pso(problem)
80      fprintf('Elapsed time for PSO calibration\t: %f seconds\n',toc)
81      results_hest_pso(1,i) = abs(v0 - x_pso(1));
82      results_hest_pso(2,i) = abs(vT - x_pso(2));
83      results_hest_pso(3,i) = abs(rho - x_pso(3));
84      results_hest_pso(4,i) = abs(k - x_pso(4));
85      results_hest_pso(5,i) = abs(sigma - x_pso(5));
86      results_hest_pso(6,i) = fval;
87      results_hest_pso(7,i) = toc;
88
89      %%% FMINCON optimization
90      options = optimset('Algorithm', 'interior-point'); %can use sqp algorithm ↩
        here
91      tic
92      x_fmincon = fmincon(@(x)objective_function_heston(S,X,tau,r,q,v0,vT,rho,k,↩
        sigma,x), ...
93      [ones(1,5)*0.5], [], [], [], [], [0.01 0.01 -1 0.1 0.1], [150 2 1 10 10], ↩
        [], options)
94      fval_fmincon = objective_function_heston(S,X, tau, r, q, v0, vT, rho, k, ↩
        sigma, x_fmincon)
95      fprintf('Elapsed time for FMINCON calibration\t: %f seconds\n',toc)
96      results_hest_fmincon(1,i) = abs(v0 - x_fmincon(1));
97      results_hest_fmincon(2,i) = abs(vT - x_fmincon(2));
98      results_hest_fmincon(3,i) = abs(rho - x_fmincon(3));
99      results_hest_fmincon(4,i) = abs(k - x_fmincon(4));
100     results_hest_fmincon(5,i) = abs(sigma - x_fmincon(5));
101     results_hest_fmincon(6,i) = fval_fmincon;
102     results_hest_fmincon(7,i) = toc;
103
104     %%% GA optimization
105     popSize = 25;
106     generations = 15;
107     options = gaoptimset('PopulationSize', popSize , 'Generations', generations↩
        , 'UseParallel', 'always',...
108         'EliteCount', round(0.1*popSize), 'Vectorized', 'off', 'HybridFcn', ↩
        @fmincon);
109     tic;
110     x_ga = ga(@(x)objective_function_heston(S,X,tau,r,q,v0,vT,rho,k,sigma,x), ↩
        ...
111         5, [], [], [], [], [0.01 0.01 -1 0.1 0.1], [150 2 1 10 10], [],options)
112     fval_ga = objective_function_heston(S,X, tau, r, q, v0, vT, rho, k, sigma, ↩
        x_ga)
113     fprintf('Elapsed time for GA calibration\t: %f seconds\n',toc)
114     results_hest_ga(1,i) = abs(v0 - x_ga(1));
115     results_hest_ga(2,i) = abs(vT - x_ga(2));
116     results_hest_ga(3,i) = abs(rho - x_ga(3));
117     results_hest_ga(4,i) = abs(k - x_ga(4));
```

```
118       results_hest_ga(5,i) = abs(sigma - x_ga(5));
119       results_hest_ga(6,i) = fval_ga;
120       results_hest_ga(7,i) = toc;
121
122
123   end
124
125   %% Generate a price surface
126   addpath(genpath('functions'))
127   S       = 100;
128   q       = 0.00;
129   r       = 0.02;
130   X       = 80:2:120;
131   tau     = [1/12 2/12 3/12 4/12 5/12 6/12 7/12 8/12 9/12 10/12 11/12 1 2 3];
132
133   v0      = 6^2;      % current variances
134   vT      = 0.1^2;    % long-run variance (theta in paper)
135   rho     = 0.9;      % correlation
136   k       = 6;        % mean reversion speed (kappa in paper)
137   sigma   = 1;        % vol of vol
138   v0      = heston_parameters(1,i);     % current variance
139   vT      = heston_parameters(2,i);     % long-run variance (theta in paper)
140   rho     = heston_parameters(3,i);     % correlation
141   k       = heston_parameters(4,i);     % mean reversion speed (kappa in paper)
142   sigma   = heston_parameters(5,i);     % vol of vol
143   % Problem
144
145   tic, [X1,Y1,Z1]  =  price_surface_heston(S,X,tau,r,q,v0,vT,rho, k,sigma);
146   fprintf('Heston price surface generated:\t required time: %f seconds\n',toc)
147   disp(' -- ')
148
149
150   h = figure();
151   surf(X1,Y1,Z1, 'LineStyle', 'none', 'FaceColor', 'red')
152   title('Call price surface under the Heston model')
153   ylabel('Time to maturity')
154   xlabel('Strike')
155   zlabel('Call option price')
156   hold on
157
158   tic, [X2,Y2,Z2]  = price_surface_heston(S,X,tau,r,q,x(1), x(2), x(3), x(4), x↩
          (5));
159   fprintf('Heston price surface generated:\t required time: %f seconds\n',toc)
160   disp(' -- ')
161   surf(X2,Y2,Z2, 'LineStyle', 'none', 'FaceColor', 'blue')
162   hold off
163   legend('Market prices', 'Model prices','Location','Best')
164
```

```
ti = get(gca,'TightInset');
set(gca,'Position',[ti(1) ti(2) 1-ti(3)-ti(1) 1-ti(4)-ti(2)]);
set(gca,'units','centimeters')
pos = get(gca,'Position');
ti = get(gca,'TightInset');
set(gcf, 'PaperUnits','centimeters');
set(gcf, 'PaperSize', [pos(3)+ti(1)+ti(3) pos(4)+ti(2)+ti(4)]);
set(gcf, 'PaperPositionMode', 'manual');
set(gcf, 'PaperPosition',[0 0 pos(3)+ti(1)+ti(3) pos(4)+ti(2)+ti(4)]);
saveas(h,'heston_cal.pdf');

% title('Call price surface under the Heston model')
% ylabel('Time to maturity')
% xlabel('Strike')
% zlabel('Call option price')
```

Script illustrating how to calibrate the Bates model with hybrid PSO + `fmincon`, `fmincon` alone, and hybrid GA + `fmincon`.

```
% EXAMPLE 3: Bates model calibration. Hybrid PSO + fmincon, fmincon only,
% hybrid GA + fmincon
    clear; clc;
    load model_parameters
    [n,m] = size(bates_parameters);
    results_bates_pso = zeros(n,m);
    results_bates_fmincon = zeros(n,m);
    results_bates_ga = zeros(n,m);
for i= 1:m
    addpath(genpath('functions'))


    % PSO options
    options = pso;
    options.PopulationSize = 50;
    %options.Vectorized = 'on';
    options.BoundaryMethod = 'penalize';
    options.PlotFcns = @psoplotbestf;
    options.PlotFcns = [];
    options.Display = 'iter';
    options.Vectorized = 'off';
    options.HybridFcn = @fmincon;
    options.StallGenLimit = 100;
    options.Generations = 50;


    S = 100;
    q = 0.00;
```

```
29      r       = 0.02;
30      X       = 80:2:120;
31      tau     = [1/12 2/12 3/12 4/12 5/12 6/12 7/12 8/12 9/12 10/12 11/12 1 2];
32  %    v0      = 0.3^2;    % current variances
33  %    vT      = 0.3^2;    % long-run variance (theta in paper)
34  %    rho     = -0.3;     % correlation
35  %    k       = 2.0;      % mean reversion speed (kappa in paper)
36  %    sigma   = 0.3;      % vol of vol
37  %    lambda  = 0.3;      % intensity of jumps;
38  %    muJ     = -0.1;     % mean of jumps;
39  %    vJ      = 0.3^2;    % variance of jumps;
40      v0      = bates_parameters(1,i);        % current variances
41      vT      = bates_parameters(2,i);        % long-run variance (theta in paper↩
        )
42      rho     = bates_parameters(3,i);        % correlation
43      k       = bates_parameters(4,i);        % mean reversion speed (kappa in ↩
        paper)
44      sigma   = bates_parameters(5,i);        % vol of vol
45      lambda  = bates_parameters(6,i);        % intensity of jumps;
46      muJ     = bates_parameters(7,i);        % mean of jumps;
47      vJ      = bates_parameters(8,i);        % variance of jumps;
48
49      %%% PSO optimization
50      problem = struct;
51      problem.fitnessfcn = @(x)objective_function_bates(S,X,tau,r,q,v0,vT,rho,k,↩
        sigma,lambda,muJ,vJ,x);
52      problem.nvars = 8;
53      problem.Aineq = [];
54      problem.bineq = [];
55      problem.lb = [0.01, 0.01, -1,  0.01, 0.01, 0.01, -1, 0.01];
56      problem.ub = [10, 2, 1, 10, 10, 10, 10, 10];
57      problem.nonlcon = @mynonlcon;
58      problem.options = options;
59
60      %%% PSO optimization
61      tic
62      [x,fval,exitflag,output] = pso(problem)
63      fprintf('Elapsed time for PSO calibration\t: %f seconds\n',toc)
64      results_bates_pso(1,i) = abs(v0 - x(1));
65      results_bates_pso(2,i) = abs(vT - x(2));
66      results_bates_pso(3,i) = abs(rho - x(3));
67      results_bates_pso(4,i) = abs(k - x(4));
68      results_bates_pso(5,i) = abs(sigma - x(5));
69      results_bates_pso(6,i) = abs(lambda - x(6));
70      results_bates_pso(7,i) = abs(muJ - x(7));
71      results_bates_pso(8,i) = abs(vJ - x(8));
72      results_bates_pso(9,i) = fval;
73      results_bates_pso(10,i) = toc;
```

```
74
75      %%% FMINCON optimization
76      options = optimset('Algorithm', 'interior-point'); %can use sqp algorithm ↩
        here
77      tic
78      x_fmincon = fmincon(@(x)objective_function_bates(S,X,tau,r,q,v0,vT,rho,k,↩
        sigma,lambda,muJ,vJ,x), ...
79      [ones(1,8)*0.5], [], [], [], [], [0.01, 0.01, -1, 0.01, 0.01, 0.01, -1, ↩
        0.01], [10, 2, 1, 10, 10, 10, 10, 10], [], options)
80      fval_fmincon = objective_function_bates(S,X,tau,r,q,v0,vT,rho,k,sigma,↩
        lambda,muJ,vJ, x_fmincon)
81      fprintf('Elapsed time for FMINCON calibration\t: %f seconds\n',toc)
82      results_bates_fmincon(1,i) = abs(v0 - x_fmincon(1));
83      results_bates_fmincon(2,i) = abs(vT - x_fmincon(2));
84      results_bates_fmincon(3,i) = abs(rho - x_fmincon(3));
85      results_bates_fmincon(4,i) = abs(k - x_fmincon(4));
86      results_bates_fmincon(5,i) = abs(sigma - x_fmincon(5));
87      results_bates_fmincon(6,i) = abs(lambda - x_fmincon(6));
88      results_bates_fmincon(7,i) = abs(muJ - x_fmincon(7));
89      results_bates_fmincon(8,i) = abs(vJ - x_fmincon(8));
90      results_bates_fmincon(9,i) = fval_fmincon;
91      results_bates_fmincon(10,i) = toc;
92
93      %%% GA optimization
94      popSize = 50;
95      generations = 50;
96      HybridOptions = optimset('Simplex','off','LargeScale','off',...
97      ____,____,____,____,____'Algorithm','interior-point','Display','none');
98      options = gaoptimset('PopulationSize', popSize , 'Generations', generations↩
        , 'UseParallel', 'always',...
99          'EliteCount', round(0.1*popSize), 'Vectorized', 'off', 'HybridFcn', {↩
        @fmincon,HybridOptions});
100     tic;
101     x_ga = ga(@(x) objective_function_bates(S,X,tau,r,q,v0,vT,rho,k,sigma,↩
        lambda,muJ,vJ, x), ...
102         8, [], [], [], [], [0.01 0.01 -1 0.01 0.01 0.01 -1 0.01], [10 2 1 10 ↩
        10 10 10 10], [], options)
103     fval_ga = objective_function_bates(S,X,tau,r,q,v0,vT,rho,k,sigma,lambda,↩
        muJ,vJ, x_ga)
104     fprintf('Elapsed time for GA calibration\t: %f seconds\n',toc)
105     results_bates_ga(1,i) = abs(v0 - x_ga(1));
106     results_bates_ga(2,i) = abs(vT - x_ga(2));
107     results_bates_ga(3,i) = abs(rho - x_ga(3));
108     results_bates_ga(4,i) = abs(k - x_ga(4));
109     results_bates_ga(5,i) = abs(sigma - x_ga(5));
110     results_bates_ga(6,i) = abs(lambda - x_ga(6));
111     results_bates_ga(7,i) = abs(muJ - x_ga(7));
112     results_bates_ga(8,i) = abs(vJ - x_ga(8));
```

```
113         results_bates_ga(9,i) = fval_ga;
114         results_bates_ga(10,i) = toc;
115 
116 end
```

Bibliography

[1] Wim Schoutens. *Lévy Processes in Finance: Pricing Financial Derivatives.* John Wiley & Sons, Ltd., 2003. ISBN 0-470-85156-2.

[2] Rama Cont and José Da Fonseca. Dynamics of implied volatility surfaces. *Quantitative finance*, 2(1):45–60, 2002.

[3] R. Cont, J. d. Fonseca, and V. Durrleman. Stochastic models of implied volatility surfaces. *Economic Notes*, 31(1):361–377, 2002.

[4] Bernard Dumas, Jeff Fleming, and Robert E Whaley. Implied volatility functions: Empirical tests. *The Journal of Finance*, 53(6):2059–2106, 1998.

[5] Steven L. Heston. A closed-form solution for options with stochastic volatility with applications to bond and currency options. *Review of Financial Studies*, 6:327–343, 1993.

[6] David S Bates. Jumps and stochastic volatility: exchange rate processes implicit in deutsche mark options. *Review of Financial Studies*, 9(1):69–107, 1996. doi: 10.1093/rfs/9.1.69. URL http://rfs.oxfordjournals.org/content/9/1/69.abstract.

[7] Ole E. Barndorff-Neilsen and Neil Shephard. Non-Gaussian Ornstein-Uhlenbeck-based models and some of their uses in financial economics. *Journal of the Royal Statistical Society, Series B*, 63:167–241, 2001.

[8] Hélyette Geman, Peter Carr, Dilip B. Madan, and Marc Yor. Stochastic volatility for levy processes. Economics Papers from University Paris Dauphine 123456789/1392, Paris Dauphine University, July 2003. URL http://ideas.repec.org/p/dau/papers/123456789-1392.html.

[9] Jim Gatheral. *The Volatility Surface: A Practitioners Guide.* John Wiley & Sons, Ltd., 2006.

[10] Hansjörg Albrecher, Philipp Mayer, Wim Schoutens, and Jurgen Tistaert. The little heston trap. *WILMOTT*, 6:83–92, 2007.

[11] W. Schoutens, E. Simons, and J. Tistaert. A perfect calibration! Now what? *Wilmott*, pages 66–78, 2004.

[12] Ole E. Barndorff-Nielsen, Elisa Nicolato, and Neil Shephard. Some recent developments in stochastic volatility modelling. Economics Papers 2001-W25, Economics Group, Nuffield College, University of Oxford, Dec 2001. URL http://ideas.repec.org/p/nuf/econwp/0125.html.

[13] John C Cox, Jr Ingersoll, Jonathan E, and Stephen A Ross. A Theory of the Term Structure of Interest Rates. *Econometrica*, 53(2):385–407, March 1985. URL http://ideas.repec.org/a/ecm/emetrp/v53y1985i2p385-407.html.

[14] Ole E. Barndorff-Nielsen. Normal inverse gaussian distributions and stochastic volatility modelling. *Scandinavian Journal of Statistics*, 24(1):1–13, 1997. ISSN 1467-9469. doi: 10.1111/1467-9469.00045. URL http://dx.doi.org/10.1111/1467-9469.00045.

[15] Peter Carr and Dilip B. Madan. Option valuation using the fast fourier transform. *Journal of Computational Finance*, 2:61–73, 1999.

[16] Kyriakos Chourdakis. Option pricing using the fractional fft. *Journal of Computational Finance*, 8(2):1–18, 1999. URL http://ssrn.com/abstract=707566.

[17] Fiodar Kilin. Accelerating the calibration of stochastic volatility models. *The Journal of Derivatives*, 18(3):7–16, 2011.

[18] Fiodar Kilin. Accelerating the calibration of stochastic volatility models. CPQF Working Paper Series 6, Frankfurt School of Finance and Management, Centre for Practical Quantitative Finance (CPQF), 2007. URL http://ideas.repec.org/p/zbw/cpqfwp/6.html.

[19] Gurdip Bakshi and Dilip Madan. Spanning and derivative-security valuation. *Journal of Financial Economics*, 55(2):205–238, February 2000. URL http://ideas.repec.org/a/eee/jfinec/v55y2000i2p205-238.html.

[20] Mukarram Attari. Option Pricing Using Fourier Transforms: A Numerically Efficient Simplification. 2004. URL http://ssrn.com/abstract=520042.

[21] Manfred Gilli and Enrico Schumann. Calibrating Option Pricing Models with Heuristics. Working Papers 030, COMISEF, Mar 2010. URL http://ideas.repec.org/p/com/wpaper/030.html.

[22] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948 vol.4, Nov 1995. doi: 10.1109/ICNN.1995.488968.

[23] S. Ebbesen, P. Kiwitz, and L. Guzzella. A generic particle swarm optimization matlab function. In *American Control Conference (ACC), 2012*, pages 1519–1524, June 2012.