The Heston Model: A Practical Approach

with

Matlab Code

Nimalin Moodley

2005

An Honours Project

submitted to the

Faculty of Science,
University of the Witwatersrand,
Johannesburg,
South Africa,

in partial fulfillment
of the requirements of the

Bachelor of Science Honours,
Programme in Advanced Mathematics of Finance

by

Nimalin Moodley
`nimalin@yahoo.com`

Acknowledgements
My gratitude and appreciation goes out to the following people: My Internal Supervisor and 'Chief' of *Maths of Finance*, Prof. David Taylor, for running the slickest department ever, his advice and motivation. Dr Graeme West (aka Supa G), for his help, advice, selflessness, and inspiration. Grant Lotter, the Don of LaTeX, for inspiring me to further my LaTeX skills. Ulrich Nögel of Fraunhofer-Institut für Technound Wirtschaftsmathematik for his advice and encouragement. The Class of 2005 for creating an environment that encouraged learning and challenged my thinking thus enabling me to achieve greater things. My family and Adisha, for all their love and support. Oscar (PhD Chaos Theory), for sitting by me while I researched. Marchand van Rooyen, for suggesting the topic.

“If I have seen further it is by standing on the shoulders of Giants.”

-Sir Isaac Newton

This project is dedicated to my Grandfather.
Thank you for being that intellectual Giant.

ABSTRACT

This document covers various aspects the Heston model. The structure and topics covered is as follows:

Chapter 1 introduces the model and provides theoretical and graphical motivation for its robustness and hence popularity. It also discusses pricing using the Partial Differential Equation and Equivalent Martingale Measure techniques

Chapter 2 discusses how the different components of the model can be evaluated computationally and how this can be achieved with different methods. These methods are then compared to each other.

Chapter 3 addresses the calibration problem. Different methods are presented as well as practical implementation, results thereof, and comparisons.

All the MATLAB code required to implement the model is provided in the appendix

Contents

Introduction

The Heston Model is one of the most widely used stochastic volatility (SV) models today. Its attractiveness lies in the powerful duality of its tractability and robustness relative to other SV models.

This project initially begun as one that addressed the calibration problem of this model. Attempting to solve such a problem was an impossible task due to the lack of exposure to such 'advanced' models.

I, therefore, decided to take a slight digression into the world of Heston and stochastic volatility. Enroute I realised that fundamental information that one would require to gain an intuitive understanding of such a model was very disjoint and hence incomplete. This project, therefore, evolved into something that could fill this gap.

A practical approach has been adopted since the focus of calibration is quite practical itself. All the relevant tools are provided to facilitate this calibration process, including MATLAB code. This code has been confined to the appendix to keep the main body clutter free and 'quick-to-read'.

All the functions and spreadsheets herein mentioned are included on the attached CD so that the reader is not confined to this document and has some sort of practical reference when approaching this subject.

CHAPTER I

A Brief Overview

The purpose of this section is to give the reader a succinct overview of the Heston Model. A 'broad-brush', pragmatic approach is adopted to give the reader an intuitive understanding of the model, rather than an overly technical one, so that the sections that follow are easily absorbed. If further technical details are desired, the reader is directed to the relevant references.

1.1 The Heston Model

(Heston 1993) proposed the following the model:

$$\begin{aligned} dS_t &= \mu S_t dt + \sqrt{V_t} S_t dW_t^1 && (1.1)\\ dV_t &= \kappa(\theta - V_t)dt + \sigma\sqrt{V_t} dW_t^2 && (1.2)\\ dW_t^1 dW_t^2 &= \rho dt && (1.3) \end{aligned}$$

where $\{S_t\}_{t\geq 0}$ and $\{V_t\}_{t\geq 0}$ are the price and volatility processes, respectively, and $\{W_t^1\}_{t\geq 0}, \{W_t^2\}_{t\geq 0}$ are correlated Brownian motion processes (with correlation parameter ρ). $\{V_t\}_{t\geq 0}$ is a square root mean reverting process, first used by (Cox, Ingersoll & Ross 1985), with long-run mean θ, and rate of reversion κ. σ is referred to as the volatility of volatility. All the parameters, viz. $\mu, \kappa, \theta, \sigma, \rho$, are time and state homogenous.

1.2 Motivation and Parameters

There are many economic, empirical, and mathematical reasons for choosing a model with such a form (see (Cont 2001) for a detailed statistical/ empirical analysis).

Empirical studies have shown that an asset's log-return distribution is non-Gaussian. It is characterised by heavy tails and high peaks (leptokurtic). There is also empirical evidence and economic arguments that suggest that equity returns and implied volatility are negatively correlated (also termed 'the leverage effect'). This departure from normality plagues the Black-Scholes-Merton model with many problems.

In contrast, Heston's model can imply a number of different distributions. ρ, which can be interpreted as the correlation between the log-returns and volatility of the asset, affects the heaviness of the tails. Intuitively, if $\rho > 0$, then volatility will increase as the asset price/return increases. This will spread the right tail and squeeze the left tail of the distribution creating a fat right-tailed distribution. Conversely, if $\rho < 0$, then volatility will increase when the asset price/return decreases, thus spreading the left tail and squeezing the right tail of the distribution creating a fat left-tailed distribution (emphasising the fact that equity returns and its related volatility are negatively correlated). ρ, therefore, affects the skewness of the distribution. Figure 1.1 shows this effect for different values ρ. MATLAB code for this simulation can be found in the appendix, § **A.1**.

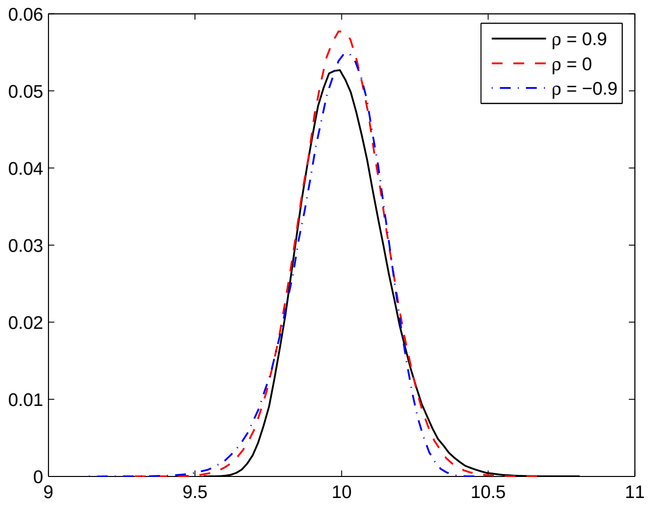

Figure 1.1: *The effect of ρ on the skewness of the density function*

The effect of changing the skewness of the distribution also impacts on the shape of the implied volatility surface. Hence, ρ also affects this. Figures 1.2, 1.3 and 1.4 show the effect of varying ρ. As can be seen, the model can imply a variety of volatility surfaces and hence addresses another shortcoming of the Black-Scholes-Merton model, viz., constant volatility across differing strike levels. MATLAB code for generation of the volatility surfaces can be found in § **A.2**.

σ affects the kurtosis (peak) of the distribution. When σ is 0 the volatility is deterministic and hence the log-returns will be normally distributed. Increasing σ will then increase the kurtosis only, creating heavy tails on both sides. Figure 1.5 shows the effect of varying σ. MATLAB code for this simulation can be found in the appendix, § **A.1**.

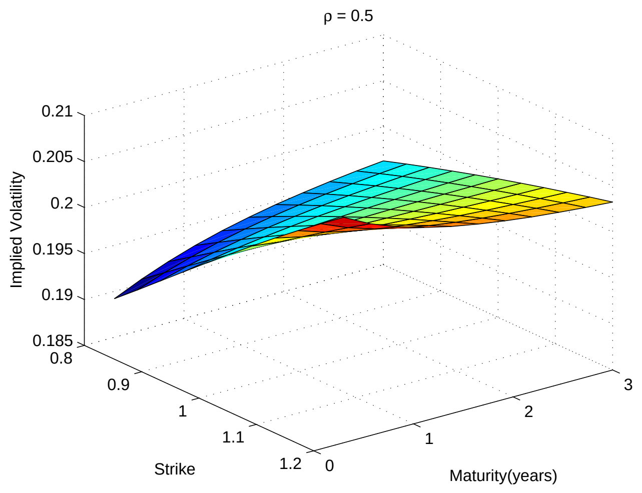

Figure 1.2: *Implied volatility surface,* $\rho = \frac{1}{2}$, $\kappa = 2$, $\theta = 0.04$, $\sigma = 0.1$, $V_0 = 0.04$, $r = 1\%$, S_0 =*1, strikes* : $0.8 - 1.2$, *maturities* : $0.5 - 3$ *years*

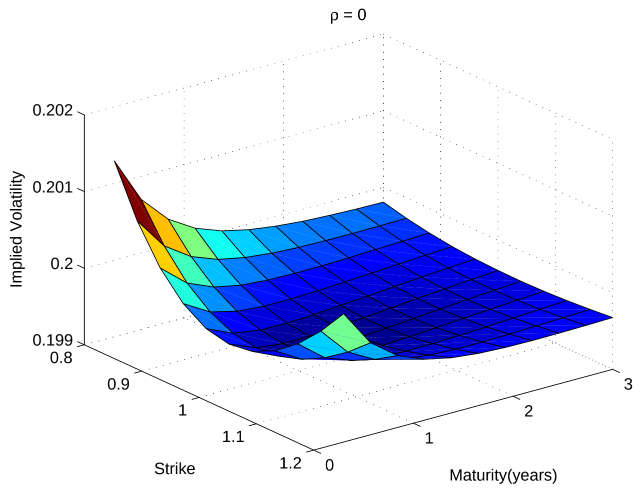

Figure 1.3: *Implied volatility surface,* $\rho = 0$, $\kappa = 2$, $\theta = 0.04$, $\sigma = 0.1$, $V_0 = 0.04$, $r = 1\%$, S_0 =*1, strikes* : $0.8 - 1.2$, *maturities* : $0.5 - 3$ *years*

Again, the effect of changing the kurtosis of the distribution impacts on the implied volatility. Figures 1.6, 1.7 and 1.8 show how σ affects the 'significance' of the smile/skew. Higher σ makes the skew/smile more prominent. This makes sense relative to the leverage effect. Higher σ means that the volatility is more volatile. This means that the market has a greater chance of extreme movements. So, writers of puts must charge more and those of calls, less, for a given strike.

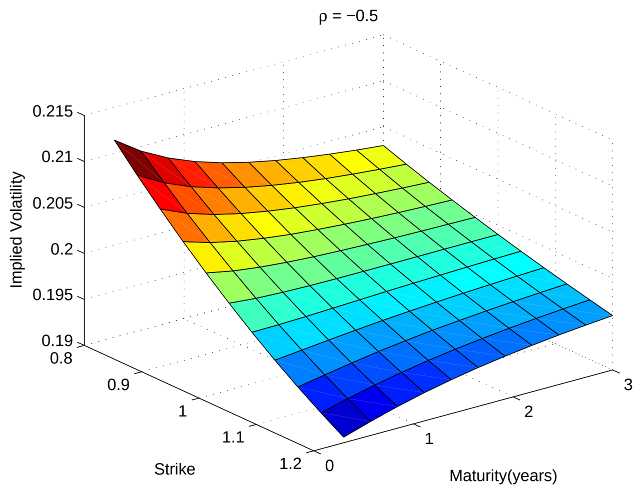

Figure 1.4: *Implied volatility surface,* $\rho = -\frac{1}{2}$, $\kappa = 2$, $\theta = 0.04$, $\sigma = 0.1$, $V_0 = 0.04$, $r = 1\%$, $S_0 = 1$, *strikes* : $0.8 - 1.2$, *maturities* : $0.5 - 3$ *years*

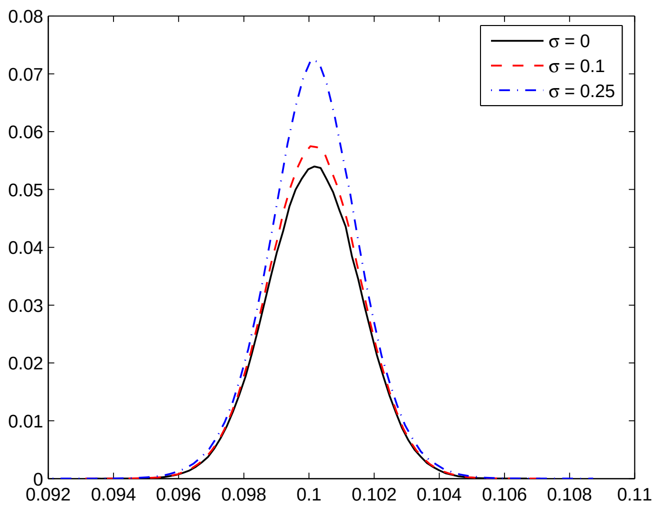

Figure 1.5: *The effect of* σ *on the kurtosis of the density function*

MATLAB code for generation of the volatility smiles/skews can be found in § **A.**$_{\mathbf{2}}$.

κ, the mean reversion parameter, can be interpreted as representing the degree of 'volatility clustering'. This is something that has been observed in the market, viz., large price variations are more likely to be followed by large price variations

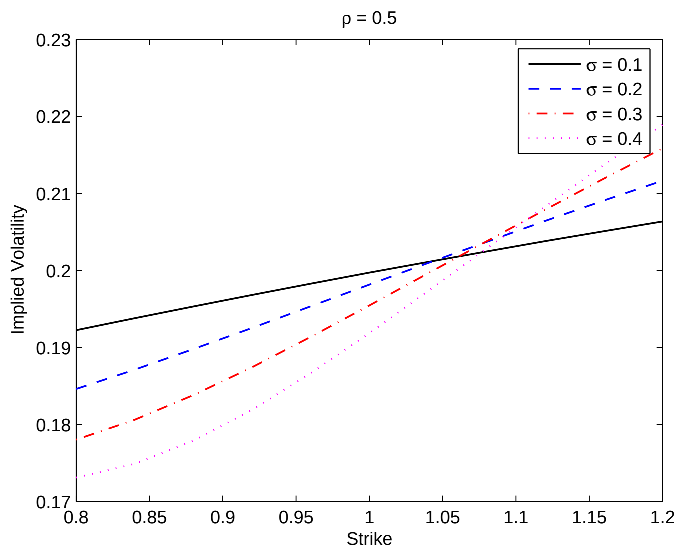

Figure 1.6: *Implied volatility surface,* $\rho = \frac{1}{2}$, $\kappa = 2$, $\theta = 0.04$, $\sigma = 0.1$, $V_0 = 0.04$, $r = 1\%$, $S_0 = 1$*, strikes* : $0.8 - 1.2$, *maturities* : $0.5 - 3$ *years*

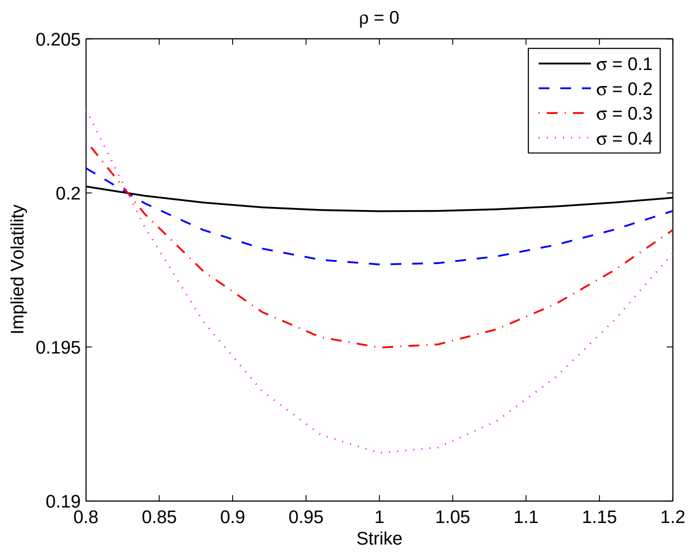

Figure 1.7: *Implied volatility surface,* $\rho = 0$, $\kappa = 2$, $\theta = 0.04$, $\sigma = 0.1$, $V_0 = 0.04$, $r = 1\%$, $S_0 = 1$*, strikes* : $0.8 - 1.2$, *maturities* : $0.5 - 3$ *years*

(Cont 2001).

A computationally convenient feature of the model is that it provides a closed-

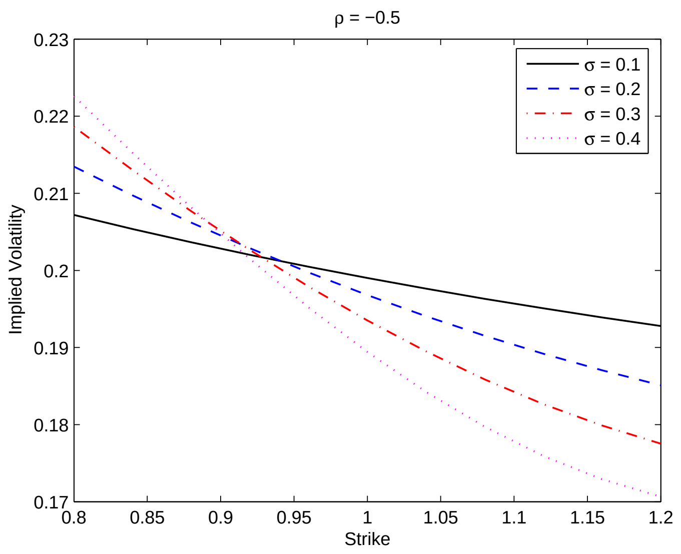

Figure 1.8: *Implied volatility surface,* $\rho = -\frac{1}{2}$, $\kappa = 2$, $\theta = 0.04$, $\sigma = 0.1$, $V_0 = 0.04$, $r = 1\%$, S_0 *=1, strikes :* $0.8 - 1.2$, *maturities :* $0.5 - 3$ *years*

form solution for European options, making it more tractable and easier to implement than other stochastic volatility models[1].

The aforementioned features of this model enables it to produce a barrage of distributions. This makes the model very robust and hence addresses the shortcomings of the Black-Scholes-Merton model. It provides a framework to price a variety of options that is closer to reality.

1.3 Pricing Options

In the Black-Scholes-Merton model, a contingent claim is dependent on one or more *tradable* assets. The randomness in the option value is solely due to the randomness of these assets. Since the assets are tradable, the option can be hedged by continuously trading the underlyings. This makes the market complete, i.e., every contingent claim can be replicated.

In a stochastic volatility model, a contingent claim is dependent on the randomness of the asset ($\{S_t\}_{t\geq 0}$) and the randomness associated with the volatility of the asset's return ($\{V_t\}_{t\geq 0}$). Only one of these is tradable, viz., the asset. Volatility is not a traded asset. This renders the market incomplete and has many

[1]more specifically, it provides a closed-form solution for any value of ρ. Models like Hull and White only provide such closed-form solutions for certain values of ρ.

implications to the pricing of options.

1.3.1 PDE approach

For purposes of brevity, the derivation of the PDE is not done here. The reader is directed to (Black & Scholes 1973), (Gatheral 2004), (Majmin 2005) or (Xu 2003) for further details. Under the Heston model, the value of any option, $U(S_t, V_t, t, T)$, must satisfy the following partial differential equation,

$$\begin{aligned}&\frac{1}{2}VS^2\frac{\partial^2 U}{\partial S^2} + \rho\sigma VS\frac{\partial^2 U}{\partial S\partial V} + \frac{1}{2}\sigma^2 V\frac{\partial^2 U}{\partial V^2} + rS\frac{\partial U}{\partial S}\\ &+\{\kappa[\theta - V] - \Lambda(S,V,t)\sigma\sqrt{V}\}\frac{\partial U}{\partial V} - rU + \frac{\partial U}{\partial t} = 0\end{aligned} \tag{1.4}$$

$\Lambda(S, V, t)$ is called the *market price of volatility risk*. Heston assumes that the market price of volatility risk is proportional to volatility, i.e.

$$\begin{aligned}\Lambda(S,V,t) &= k\sqrt{V} && \text{for some constant } k\\ \Rightarrow \Lambda(S,V,t)\sigma\sqrt{V} &= k\sigma V_t\\ &= \lambda(S,V,t), && \text{say}\end{aligned}$$

$\lambda(S, V, t)$ therefore represents the market price of volatility risk. This parameter does not appear in (1.1), (1.2) or (1.3) but does appear in the pricing formula, (2.1), and hence needs to be approximated. This is no easy task as it well known that the market price of volatility risk is nearly impossible to estimate. This problem is overcome due to the parametric nature of the model and the existence of a closed-form solution (see § **3.1** and § **2.1**).

1.3.2 Risk Neutral approach

Risk neutral valuation is the pricing of a contingent claim in an equivalent martingale measure (EMM). The price is evaluated as the expected discounted payoff of the contingent claim[2], under the EMM $\mathbb{Q}$, say. So,

$$\text{Option Value} = \mathbb{E}_t^{\mathbb{Q}}[e^{r(T-t)}H(T)] \tag{1.5}$$

where $H(T)$ is the payoff of the option at time T and r is the risk free rate of interest over $[t, T]$ (we are assuming, of course, that interest rates are deterministic and pre-visible, and that the numeraire is the money market instrument).

[2]this is referred to as the 'Law of the Unconscious Financial Mathematician'

Moving from a real world measure to an EMM is achieved by Girsavov's Theorem. In particular, we have

$$\begin{aligned}
d\tilde{W}_t^1 &= dW_t^1 + \vartheta_t dt && (1.6)\\
d\tilde{W}_t^2 &= dW_t^2 + \Lambda(S,V,t)dt && (1.7)\\
\frac{d\mathbb{Q}}{d\mathbb{P}} &= \exp\bigg\{-\frac{1}{2}\int_0^t ({\vartheta_s}^2 + \Lambda(S,V,s)^2)ds - \\
&\qquad \int_0^t \vartheta_s dW_s^1 - \int_0^t \Lambda(S,V,t)dW_s^2\bigg\} && (1.8)\\
\vartheta_t &= \frac{\mu - r}{\sqrt{V_t}} && (1.9)
\end{aligned}$$

where $\mathbb{P}$ is the real world measure and $\{\tilde{W}_t^1\}_{t\geq 0}$ and $\{\tilde{W}_t^2\}_{t\geq 0}$ are $\mathbb{Q}$-Brownian Motions. Under measure $\mathbb{Q}$, (1.1), (1.2), and (1.3) become,

$$\begin{aligned}
dS_t &= rS_t dt + \sqrt{V_t} S_t d\tilde{W}_t^1 && (1.10)\\
dV_t &= \kappa^*(\theta^* - V_t)dt + \sigma\sqrt{V_t}d\tilde{W}_t^2 && (1.11)\\
d\tilde{W}_t^1 d\tilde{W}_t^2 &= \rho dt && (1.12)
\end{aligned}$$

where,

$$\begin{aligned}
\kappa^* &= \kappa + \lambda\\
\theta^* &= \frac{\kappa\theta}{\kappa + \lambda}
\end{aligned}$$

This is an important result. Under the risk-neutral measure, λ has effectively been 'eliminated'.

In a complete market, it can be shown that every asset/option has the same market price of risk (i.e., $\frac{\mu_F - r}{\sigma_F}$ is constant for any asset F). Since volatility isn't a traded asset, the market is incomplete and $\Lambda(S,V,t)$ isn't constant. It is clear that (1.6), (1.7), and (1.8) solely determine the EMM. This implies that the EMM is not unique and is dependent on the value of $\Lambda(S,V,t)$.

The implications are that the different EMM's will produce different option prices, depending on the value of $\Lambda(S,V,t)$. This, initially, poses a problem. Again, due to the parametric nature of the model and the existence of a closed-form solution, we can work around this (see § **2.1** and § **3.1**).

CHAPTER II

Computational Evaluation

In order to practically implement such a model, one needs to be able to compute the price of vanilla options implied by it. This is used in the calibration process to obtain parameter values, which in turn will be used to price other exotic options. This section, therefore, provides the tools required to perform such calculations.

2.1 The Closed-Form Solution

The closed-form solution of a European call option on a non-dividend paying asset for the Heston model is:

$$C(S_t, V_t, t, T) = S_t P_1 - K e^{-r(T-t)} P_2 \tag{2.1}$$

where,

$$\begin{aligned}
P_j(x, V_t, T, K) &= \frac{1}{2} + \frac{1}{\pi}\int_0^\infty \operatorname{Re}\left(\frac{e^{-i\phi\ln(K)} f_j(x, V_t, T, \phi)}{i\phi}\right) d\phi \qquad (2.2)\\
x &= \ln(S_t)\\
f_j(x, V_t, T, \phi) &= \exp\{C(T-t,\phi) + D(T-t,\phi)V_t + i\phi x\}\\
C(T-t,\phi) &= r\phi i r + \frac{a}{\sigma^2}\left[(b_j - \rho\sigma\phi i + d)\tau - 2\ln\left(\frac{1-ge^{dr}}{1-g}\right)\right]\\
D(T-t,\phi) &= \frac{b_j - \rho\sigma\phi i + d}{\sigma^2}\left(\frac{1-e^{dr}}{1-ge^{dr}}\right)\\
g &= \frac{b_j - \rho\sigma\phi i + d}{b_j - \rho\sigma\phi i - d}\\
d &= \sqrt{(\rho\sigma\phi i - b_j)^2 - \sigma^2(2u_j\phi i - \phi^2)}
\end{aligned}$$

for *j* = 1,2, where,

$$u_1 = \frac{1}{2}, \quad u_2 = -\frac{1}{2}, \quad a = \kappa\theta, \quad b_1 = \kappa + \lambda - \rho\sigma, \quad b_2 = \kappa + \lambda$$

Such a formula looks intimidating, but in reality is quite ‘explicit’ and easy to evaluate in MATLAB. The only part that poses a slight problem is the limits of the integral in (2.2). This integral cannot be evaluated exactly, but can be approximated with reasonable accuracy by using some numerical integration technique, e.g., Gauss Lagendre or Gauss Lobatto integration.

Under the EMM $\mathbb{Q}$ some parameter simplification takes place viz,

$$a = \kappa^*\theta^*, \qquad b_1 = \kappa^* - \rho\sigma, \qquad b_2 = \kappa^*$$

Again, it can be seen that, effectively, the parameter λ has been eliminated.

A method to evaluate formulas in the form of (2.1) has been proposed by (Carr & Madan 1999). This method is much quicker than using a numerical method for the said integrals.

2.2 Fast Fourier Transform (FFT)

This method has been proposed by (Carr & Madan 1999). We first begin with some basic definitions.

Definition 2.1. *The Fourier transform ($\mathcal{F}\{\cdot\}$) and inverse Fourier transform ($\mathcal{F}^-1\{\cdot\}$) of an integrable function, $f(x)$, is*

$$\mathcal{F}\{f(x)\} = \int_{-\infty}^{\infty} e^{i\phi x} f(x)dx = F(\phi) \tag{2.3}$$

$$\mathcal{F}^{-1}\{F(\phi)\} = \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{-i\phi x} F(\phi)d\phi = f(x) \tag{2.4}$$

In our framework, $f(x)$ is the risk neutral density function of the log-return of the underlying and $F(\phi)$ is called the characteristic function of $f(x)$. It should be noted that even if a random variable is not *absolutely continuous*[3], its characteristic function exists. For absolutely continuous random variables (say X), the characteristic function can be expressed as an expectation, viz.,

$$\mathbb{E}[e^{i\phi X}]$$

FFT is an efficient algorithm used to calculate summations of the following form:

$$w(k) = \sum_{j=1}^{N} e^{-i\frac{2\pi}{N}(j-1)(k-1)} x(j) \tag{2.5}$$

It is a type of discrete approximation to a Fourier transform. Carr and Madan have the following to say regarding their method:

> *'...find that the use of the FFT is considerably faster than most available methods and furthermore, the traditional method described in Heston , Bates , Bakshi and Madan , and Scott can be both slow and inaccurate...'*

[3]A random variable is *absolutely continuous* if it has a density function.

Option pricing lends itself well to this sort of application due to the form of (1.5). Let $x_t := \ln S_t$ and $k := \ln K$, where K is the strike price of the option. Then the value of a European call, with maturity T, as a function of k is,

$$C_T(k) = e^{-rT} \int_k^\infty (e^{x_T} - e^k) f_T(x_T) dx_T \tag{2.6}$$

where $f_T(x)$ is the risk neutral density function of x, defined above. Carr and Madan define a modified call price function[4],

$$c_T(k) = e^{\alpha k} C_T(k), \qquad \alpha > 0 \tag{2.7}$$

We now expect $c_T(k)$ to be square integrable for a range of α values and $\forall\ k$. α is referred to as the *dampening factor*. The Fourier transform and inverse Fourier transform of $c_T(k)$ is,

$$\begin{aligned} F_{c_T}(\phi) &= \int_{-\infty}^\infty e^{i\phi k} c_T(k) dk & (2.8)\\ c_T(k) &= \frac{1}{2\pi} \int_{-\infty}^\infty e^{-i\phi k} F_{c_T}(\phi) d\phi & (2.9) \end{aligned}$$

Substituting (2.9) into (2.7),

$$\begin{aligned} C_T(k) &= e^{-\alpha k} c_T(k) & (2.10)\\ &= e^{-\alpha k} \frac{1}{2\pi} \int_{-\infty}^\infty e^{-i\phi k} F_{c_T}(\phi) d\phi & (2.11)\\ &= e^{-\alpha k} \frac{1}{\pi} \int_0^\infty e^{-i\phi k} F_{c_T}(\phi) d\phi & (2.12) \end{aligned}$$

Substituting (2.6) and (2.7) into (2.8),

$$\begin{aligned} F_{c_T}(\phi) &= \int_{-\infty}^\infty e^{i\phi k} e^{\alpha k} e^{-rT} \int_k^\infty (e^{x_T} - e^k) f_T(x_T) dx_T dk\\ &= \int_{-\infty}^\infty e^{-rT} f_T(x_T) \int_{-\infty}^{x_T} (e^{x_T + \alpha k} - e^{(\alpha+1)k}) e^{i\phi k} dk dx_T\\ &= \frac{e^{-rT}}{\alpha^2 + \alpha - \phi^2 + i(2\alpha+1)\phi} \int_{-\infty}^\infty e^{(\alpha+1+i\phi)x_T} f_T(x_T) dx_T\\ &= \frac{e^{-rT}}{\alpha^2 + \alpha - \phi^2 + i(2\alpha+1)\phi} \int_{-\infty}^\infty e^{(-\alpha i - i + \phi) i x_T} f_T(x_T) dx_T\\ &= \frac{e^{-rT} F_{C_T}(\phi - (\alpha+1)i)}{\alpha^2 + \alpha - \phi^2 + i(2\alpha+1)\phi} \end{aligned}$$

[4]because $C_T(k)$ is not square integrable

where $F_{C_T}(\phi)$ is the characteristic function of x_T, under $\mathbb{Q}$, given by (Hong 2004):

$$\begin{aligned}
F_{C_T}(\phi) &= e^{A(\phi)+B(\phi)+C(\phi)} \\
A(\phi) &= i\phi(x_0 + rT) \\
B(\phi) &= \frac{2\zeta(\phi)(1-e^{-\psi(\phi)T})V_0}{2\psi(\phi)-(\psi(\phi)-\gamma(\phi))(1-e^{-\psi(\phi)T})} \\
C(\phi) &= -\frac{\kappa\theta}{\sigma^2}\left[2\log\left(\frac{2\psi(\phi)-(\psi(\phi)-\gamma(\phi))(1-e^{-\psi(\phi)T})}{2\psi(\phi)}\right)+\right. \\
&\quad \left.(\psi(\phi)-\gamma(\phi))T\right] \\
\zeta(\phi) &= -\frac{1}{2}(\phi^2 + i\phi) \\
\psi(\phi) &= \sqrt{\gamma(\phi)^2 - 2\sigma^2\zeta(\phi)} \\
\gamma(\phi) &= \kappa - \rho\sigma\phi i
\end{aligned}$$

(Carr & Madan 1999) show that,

$$C_T(k_u) \approx \frac{e^{-\alpha k_u}}{\pi}\sum_{j=1}^{N} e^{-i\frac{2\pi}{N}(j-1)(u-1)}\underbrace{e^{ibv_j}F_{c_T}(v_j)\frac{\eta}{3}(3+(-1)^j-\delta_{j-1})}_{\circledast} \tag{2.13}$$

where[5],

$$\begin{aligned}
v_j &= \eta(j-1) \\
\eta &= \frac{c}{N} \\
c &= 600 \\
N &= 4096 \\
b &= \frac{\pi}{\eta} \\
k_u &= -b + \frac{2b}{N}(u-1), \qquad u = 1,2,\ldots,N+1
\end{aligned}$$

This is now a direct application of (2.5) and hence FFT. MATLAB can perform FFT using the `fft(X)` function, where `X` is the vector created from $\circledast$. `fft(X)` returns a $(1 \times N)$ vector whose j^{th} column corresponds to a log-strike of $-b + \frac{2b}{N}(j-1)$. The appendix, § **A.3.1**, contains the code for pricing a European Call using FFT.

[5] these parameters have been specified by Carr and Madan so that a balance is struck between computational time and accuracy.

All that remains is to choose the value of α. (Carr & Madan 1999) suggest that α be chosen such that

$$\begin{aligned} \mathbb{E}[S_T^{\alpha+1}] &< \infty \\ \Rightarrow F_{C_T}(-(\alpha+1)i) &< \infty \end{aligned} \tag{2.14}$$

Carr goes further to provide FFT for out-of the money options. He argues that the above formula depends on the intrinsic value of the option. Since out-of-the-money options have no intrinsic value, an FFT based on the time value of the option has to be derived. The formula is presented here without derivation, for completeness, as the author finds very little difference between the solutions of the two methods. The MATLAB code can also be found in § **A.3.1**.

$$\begin{aligned} C_T(k_u) &\approx \frac{1}{\pi \sinh(\alpha k_u)} \sum_{j=1}^{N} e^{-i\frac{2\pi}{N}(j-1)(u-1)} e^{ibv_j} \gamma_T(v_j) \frac{\eta}{3}(3 + (-1)^j - \delta_{j-1}) \\ \gamma_T(v) &= \frac{\varphi_T(v - i\alpha) - \varphi_T(v + i\alpha)}{2} \\ \varphi_T(v) &= e^{-rT}\left[\frac{1}{1+iv} - \frac{e^{rT}}{iv} - \frac{F_{C_T}(v-i)}{v^2 - iv}\right] \end{aligned}$$

2.3 Numerical Integration - Quadrature Rules

Numerical integration can be used to evaluate the integral in (2.2). Two techniques are briefly discussed together with their implementation.

2.3.1 Adaptive Simpson's Rule

The following is adapted from (Cheney & Kincaid 1999). The *basic Simpson's Rule* for numerical integration of a function, $f(x)$, over two equal subintervals, with partition points $a, a+h$, and $a+2h$ (we choose this form of partition for ease of calculation and notation. A general formula will be given later), is

$$\int_a^{a+2h} f(x)dx \approx \frac{h}{3}[f(a) + 4f(a+h) + f(a+2h)] \tag{2.15}$$

The error of the above approximate can be established with a basic Taylor[6]

[6]no relation to the internal supervisor of this document.

series. Consider,

$$\begin{aligned} f(a+h) &= f(a) + hf'(a) + \frac{1}{2!}h^2 f''(a) + \frac{1}{3!}h^3 f'''(a) + \frac{1}{4!}h^4 f^{(4)}(a) + \dots \\ f(a+2h) &= f(a) + 2hf'(a) + 2h^2 f''(a) + \frac{4}{3}h^3 f'''(a) + \frac{2^4}{4!}h^4 f^{(4)}(a) + \dots \end{aligned}$$

So,

$$\text{RHS of (2.15)} = 2hf + 2h^2 f' + \frac{4}{3}h^3 f'' + \frac{2}{3}h^4 f''' + \frac{20}{3.4!}h^5 f^{(4)}(a) + \dots \tag{2.16}$$

Now, define

$$F(x) := \int_a^x f(t)dt$$

Again, using a Taylor series,

$$F(a+2h) = F(a) + 2hF'(a) + 2h^2 F''(a) + \frac{4}{3}h^3 F'''(a) + \frac{2}{3}h^4 F^{(4)}(a) + \frac{2^5}{5!}h^5 F^{(5)}(a) \dots$$

Noting, by the Fundamental Theorem of Calculus, that $F^{(n+1)}(a) = f^{(n)}(a)$ and $F(a) = 0$ we have

$$\begin{aligned} \int_a^{a+2h} f(x)dx &= F(a+2h) \\ &= 2hf(a) + 2h^2 f'(a) + \frac{4}{3}h^3 f''(a) + \frac{2}{3}h^4 f'''(a) + \dots \end{aligned} \tag{2.17}$$

Subtracting (2.16) from (2.17), we get,

$$\text{Error in approximation} = -\frac{h^5}{90} f^{(4)}(\xi) = O(h^5), \qquad \xi \in (a, a+2h)$$

We now state a general result for the *basic Simpson's Rule*,

$$\begin{aligned} \int_a^b f(x)dx &\approx \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right] \\ \text{Error} &= -\frac{1}{90}\left(\frac{b-a}{2}\right)^5 f^{(4)}(\xi), \qquad \xi \in (a,b) \end{aligned}$$

An *Adaptive Simpson's Rule* divides the interval of integration into more than 2 subintervals. The number of subintervals is dependent on the required accuracy of the estimate. This is quantified by the error term given above. The function will, in general, behave differently over different parts of the interval and hence the intervals chosen will not be of uniform length.

The MATLAB function `quad(@fun,a,b)` uses an *Adaptive Simpson's Rule* to numerically integrate `@fun` over [a,b]. It produces a result that has an error less than 10^{-6} or a user defined tolerance level which is prescribed by a fourth argument.

2.3.2 Adaptive Gaussian Quadrature Rules

Gaussian quadrature rules approximate the integral of a function, $f(x)$, over $[a, b]$ by:

$$\int_a^b f(x)dx \approx \sum_{i=0}^{n} A_i f(x_i) \tag{2.18}$$

where,

$$\begin{aligned} A_i &= \int_a^b \ell_i(x)dx \\ \ell_i(x) &= \prod_{\substack{j=0 \\ j\neq i}}^{n} \left(\frac{x - x_j}{x_i - x_j} \right) \end{aligned}$$

where $\{x_i\}_{0\leq i\leq n}$, termed *nodes* or *abscissae*, are the zeros of a *Lagrange interpolating polynomial*.

A *Lagrange interpolating polynomial* is a polynomial of order $n-1$ that passes through n given points. Examples can be seen in figure 2.1.

The approximation in (2.18) is a weighted average of the function at discrete nodes, $\{x_i\}_{0\leq i\leq n}$, with weights $\{A_i\}_{0\leq i\leq n}$. The interpolating polynomial, $p_{n+1}(x)$, of degree $n + 1$ must satisfy:

$$\int_a^b x^k p(x)dx = 0, \qquad (0 \leq k \leq n)$$

It is quite intuitive that the choice of the polynomials will determine the accuracy of the estimate in (2.18). The most popular choice of these polynomials are the *Legendre polynomials*. These are generated by setting the interval of integration to $[-1, 1]$ and standardising $p_n(x)$ such that $p_n(1) = 1$. The roots of these polynomials will then be used as nodes in the Gaussian quadrature over $[-1, 1]$. A simple linear transformation can translate the original integral over $[a, b]$ to $[-1, 1]$.

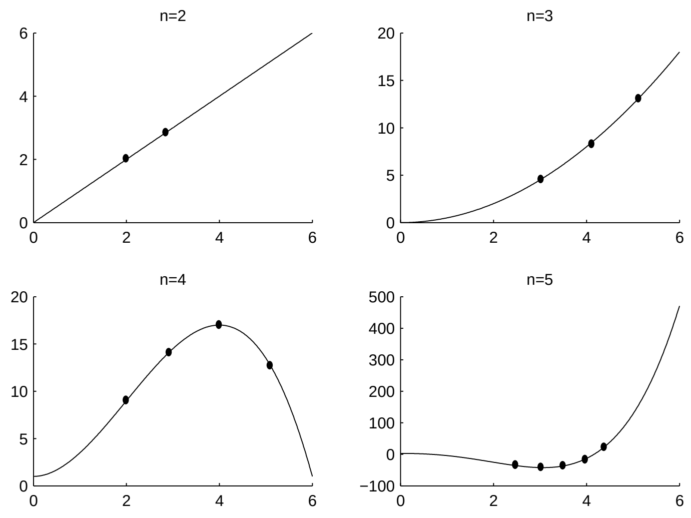

Figure 2.1: *Lagrange interpolating polynomials for n = 2, 3, 4 and 5 points. These polynomials are of order 1, 2, 3 and 4 respectively.*

The MATLAB function `quadl(@fun,a,b)` implements an adaptive *Gauss Lobatto* quadrature rule on the function `@fun` over the interval $[a, b]$. It's defined as the Gaussian quadrature presented above, with a specific form for the weights and nodes. The integral is evaluated over $[-1, 1]$ with nodes that are from the *Legendre polynomials*. The nodes are symmetric about the origin and also include the endpoints of the interval. The error in the estimate is given by,

$$\text{Error} = -\frac{n(n-1)^3 2^{2n-1}((n-2)!)^4}{(2n-1)((2n-2)!)^3} f^{(2n-2)}(\xi)$$

where n = number of nodes, and $\xi \in (-1, 1)$. The rule is adaptive as it divides the original integral into smaller subintervals and performs *Gauss Lobatto* integration with varying node quantities. This is performed to achieve an estimate whose error is less than 10^{-6} or a user defined tolerance level prescribed by a fourth argument.

2.3.3 Computation of the 'Heston Integral'

In order to evaluate (2.1) we need to compute the integral in (2.2), viz.:

$$P_j(x, V_t, T, K) = \frac{1}{2} + \frac{1}{\pi}\int_0^\infty \text{Re}\left[\frac{e^{-i\phi \ln(K)} f_j(x, V_t, T, \phi)}{i\phi}\right] d\phi \qquad (2.19)$$

for $j = 1, 2$. This can be done in MATLAB with the 2 functions, `quad(@fun a,b)` and `quadl(@fun,a,b)`, discussed above. A problem arises as (2.19) is an improper integral and the argument `b` cannot be specified as 'infinity' i.e., `quad(@fun,a,b)` evaluates only proper integrals. A quick plot of the integrand will reveal that it converges very quickly to zero. Figure 2.2 shows the effect of changing the parameters on the integrand. The parameters that were changed were those that had the most significant effect on the integrand.

So, for sufficiently large `b`, the integral of (2.19) can be evaluated with the required accuracy. I have chosen `b` = 100 to be prudent.

MATLAB code to price a European call option using Heston's model via *Adaptive Gauss Lobatto* integration/*Simpson's Rule* can be found in the appendix, § **A.3.2**.

2.3.4 Comparison of Quadrature Rules

The purpose of this section is to gauge which quadrature rule to use. For a complete discussion refer to (Gander & Gautschi 1998).

MATLAB defines the functions, `quad` and `quadl`, as low order and high order quadrature rules. One would therefore expect `quadl` to be superior.

(Gander & Gautschi 1998) say that for extremely small tolerance levels `quadl` outperforms `quad` in:

- **efficiency** as measured by the number of recursive steps required to compute an answer within a certain tolerance level.
- **reliability** as measured by the extent to which the required tolerance is achieved.

For large tolerance levels, `quad` is more efficient (faster) than `quadl`, but less reliable. Keeping in mind that we're pricing options, we require an extremely low tolerance level. Hence `quadl` would be better suited for our purposes.

2.4 Comparison of FFT and Numerical Integration

It now remains to find which method is better suited for our purposes.

FFT is faster in a sense that it generates a matrix of prices with differing strikes. Thousands of European call option prices are immediately available. The quadrature rule evaluates the option price only for a given set of parameters, i.e.,

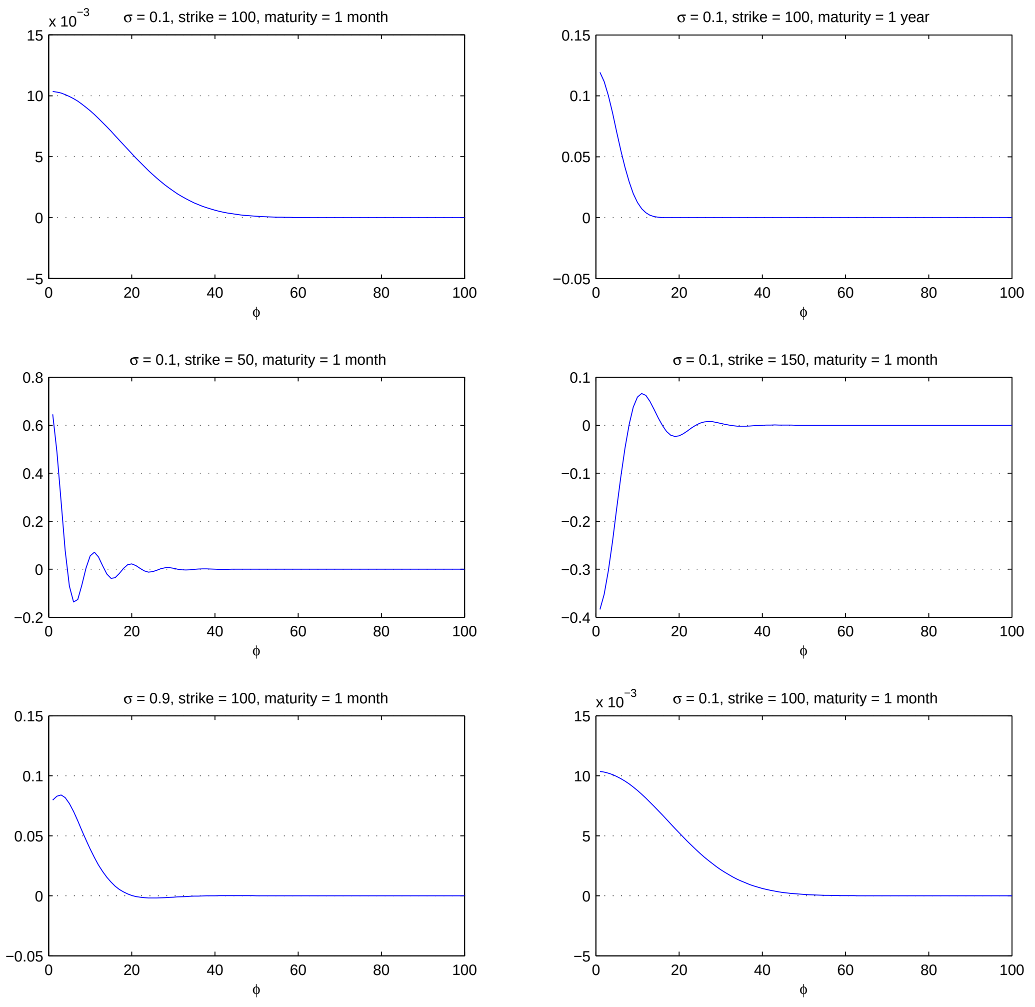

Figure 2.2: *The Integrand of (2.19). Notice how quickly this integrand converges to zero. Parameters that were kept constant :* $\kappa = 2, \theta = 0.04, \rho = -0.5, V_0 = 0.05, S_0 = 100, r = 0.1$

only for one strike price at a time. The time difference is extremely significant. For a given set of parameters, FFT calculates prices for 4097 different strikes in approximately 0.14 seconds. The quadrature rule takes approximately 281.89 seconds to perform the same calculation. FFT is faster by a factor of 2014.

FFT, in this application, suffers from one drawback, i.e., accuracy. The solutions produced are depend on the choice of α (figure 2.3 shows the price produced by FFT as a function of α). From (2.14) we can also see that α is depend on the set of parameters. The relationship between α and these parameters is not as trivial to identify as is done in (Carr & Madan 1999) for the Variance-Gamma model.

We are therefore left with the quadrature rule as our primary tool for evaluation. If an α can be found that satisfies (2.14), then FFT is far superior.

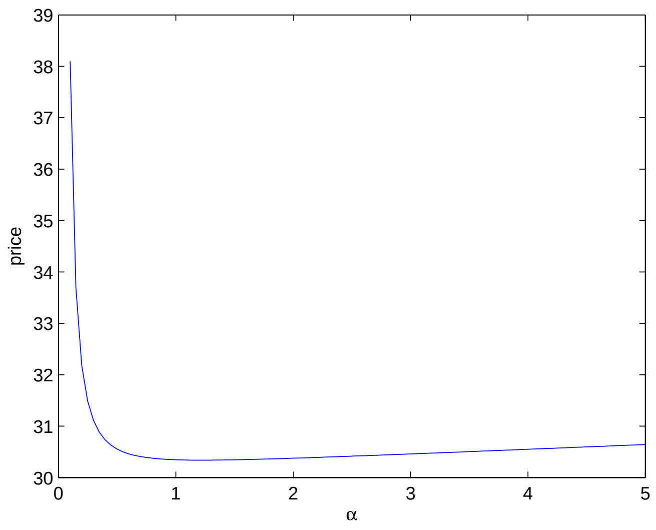

Figure 2.3: *This illustrates how the price of an option is depend on the choice of* α

CHAPTER III

Calibration of the Model

3.1 The Calibration Problem

The price to pay for more realistic models is the increased complexity of model calibration. Often, the estimation method becomes as crucial as the model itself (Cont 2005).

The Heston model has six parameters that need estimation, viz., $\kappa, \theta, \sigma, V_0$, ρ, λ. Research has shown that the implied parameters (i.e. those parameters that produce the correct vanilla option prices) and their time-series estimate counterparts are different (see (Bakshi, Cao & Chen 1997)). So one cannot just use empirical estimates for the parameters.

This leads to a complication that plagues stochastic volatility models in general[7]. A common solution is to find those parameters which produce the correct *market prices* of vanilla options. This is called an *inverse* problem, as we solve for the parameters indirectly through some implied structure [8].

The most popular approach to solving this inverse problem is to minimise the error or discrepancy between model prices and market prices. This usually turns out to be a non-linear least-squares optimisation problem. More specifically, the squared differences between vanilla option market prices and that of the model are minimised over the parameter space, i.e., we evaluate

$$\min_{\mathbf{\Omega}} S(\mathbf{\Omega}) = \min_{\mathbf{\Omega}} \sum_{i=1}^{N} w_i \left[C_i^{\mathbf{\Omega}}(K_i, T_i) - C_i^M(K_i, T_i) \right]^2 \tag{3.1}$$

where $\mathbf{\Omega}$ is a vector of parameter values, $C_i^{\mathbf{\Omega}}(K_i, T_i)$ and $C_i^M(K_i, T_i)$ are the i^{th} option prices from the model and market, respectively, with strike K_i and maturity T_i. N is the number of options used for calibration, and the w_i's are weights (the choice of these weights will be discussed later).

The question now arises as to what market prices to use in this calibration process, as for any given option there exists a bid and ask price. This may seem as a problem but it actually allows flexibility in the calibration process.

[7]On the positive side, we need not estimate the market price of volatility risk, which is near impossible to do in practice.

[8]eg. solving for a polynomial with given roots is an inverse problem.

We will use the mid-price of the option but accept a parameter set, $\mathbf{\Omega_0}$, given that

$$\sum_{i=1}^{N} w_i \left[C_i^{\mathbf{\Omega_0}}(K_i, T_i) - C_i^M(K_i, T_i)\right]^2 \leq \sum_{i=0}^{N} w_i[\text{bid}_i - \text{ask}_i]^2 \qquad (3.2)$$

where bid$_i$/ask$_i$ are the bid/ask prices of the i^{th} option. This means that we do not require the model to match the mid-prices exactly, but fall, on average, within the bid-offer spread. This is not an unreasonable relaxation of the calibration process. We should bare in mind that the modeling process should produce the required *estimates* within a certain tolerance level. Accuracy beyond this could be spurious and hence produce less accurate exotic prices.

The minimisation mentioned above is not as trivial as it would seem. In general, $S(\mathbf{\Omega})$ is neither convex nor does it have any particular structure. This poses some complications:

- Finding the minimum of $S(\mathbf{\Omega})$ is not as simple as finding those parameter values that make the gradient of $S(\mathbf{\Omega})$ zero. This means that a gradient based optimisation method will prove to be futile.
- Hence, finding a global minimum is difficult (and very dependent on the optimisation method used).
- Unique solutions to (3.1) need not necessarily exist, in which case only local minima can be found. This has some implications regarding the stationarity of parameter values which are important in these types of models. This is discussed later.

The last two points make this an *ill-posed* problem. This is therefore an *inverse, ill-posed* problem termed the **calibration problem**.

Figure 3.1 plots $S(\mathbf{\Omega})$ as a function of κ and σ. It is easy to see that gradient based optimisers will struggle to find a global minimum. Notice, also, the number of points that are non-differentiable. This poses a further problem for gradient based optimisers. The graph presents a graphical idea of the nature of $S(\mathbf{\Omega})$ as a function of two of its parameters, but it is important to remember that $S(\mathbf{\Omega})$ is 5-dimensional and as such could be even nastier in its 'true' form.

The aim is therefore to identify optimisation methods that can deal with the peculiarities of the calibration problem.

3.2 Calibration methods

Before discussing the calibration methods, we address some technical issues.

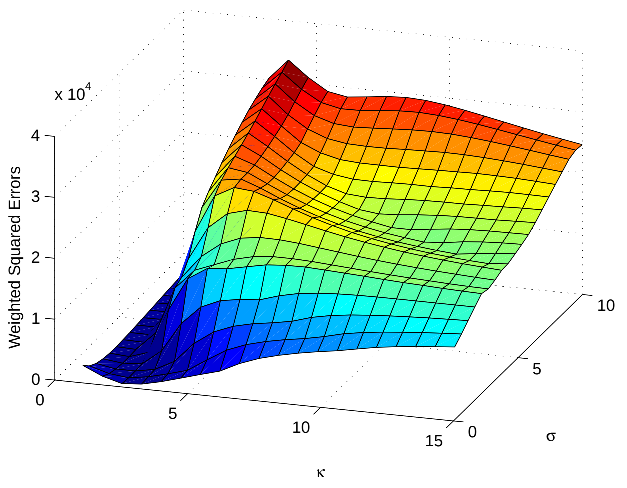

Figure 3.1: *The Heston error surface as a function of κ and σ*

The calibration problem has a slight simplification when we price options under an EMM, i.e., we can evaluate (3.1) using the $C_i^{\mathbf{\Omega}}(K_i, T_i)$'s under an EMM. As mentioned in § **1.3.2**, evaluation under an EMM effectively reduces the number of estimated parameters to five. The following must therefore hold:

$$\text{OptionValue}^{\mathbb{P}}(\kappa, \theta, \sigma, V_0, \rho, \lambda) = \text{OptionValue}^{\mathbb{Q}}(\kappa^*, \theta^*, \sigma, V_0, \rho, 0)$$

We also address the choice of the weights in (3.1). We choose the w_i's to be $\frac{1}{|\text{bid}_i - \text{ask}_i|}$. This choice is quite intuitive. If the spread is great, we have a wider range of prices that the model can imply. In other words the model is allowed to imply a wider range of prices around the mid-price. This means that less weight should be placed on such a price, and vice-versa. (Cont 2005) suggests using the implied volatilities as weights.

We also need a constraint on the parameters to prevent the volatility reaching zero. (Mikhailov & Nögel 2003) suggest that this constraint be $2\kappa\theta > \sigma^2$.

3.2.1 Regularisation

This method is discussed briefly for completeness. For a detailed discussion refer to (Chiarella, Craddock & El-Hassan 2000).

Regularisation involves adding a *penalty* function, $p(\mathbf{\Omega})$, to (3.1) such that

$$\sum_{i=1}^{N} w_i \left[C_i^{\mathbf{\Omega}}(K_i, T_i) - C_i^M(K_i, T_i)\right]^2 + \alpha p(\mathbf{\Omega}) \tag{3.3}$$

is convex. α, here, is called the *regularisation parameter*.

The philosophy behind this strategy is one of pragmatism. We cannot hope to determine the exact solution to our problem because of its very nature. Consequently, we attempt to find an approximation which is as close to the true solution as possible. To achieve this we are moved to replace our problem with one which is close to the original, but does not possess the ill conditioning which the makes the original intractable. In other words: Don't try and solve the given problem, try and solve a different one whose solution is close to that of your problem. This is the essence of all regularisation methods (Chiarella et al. 2000).

When applied to a given set of market prices, these methods yield a single set of model parameters calibrated to the market but also require the extra step of determining the regularisation parameter, α (Cont 2005).

(Mikhailov & Nögel 2003) suggest using $\alpha p(\mathbf{\Omega}) = ||\mathbf{\Omega} - \mathbf{\Omega_0}||^2$, where $\mathbf{\Omega_0}$ is an initial estimate of the parameters. This method is therefore dependent on the choice of the the initial parameters. It is, in a sense, a local minimum optimiser.

Equation (3.3) can be minimised in MATLAB using the function `lsqnonlin()`.

3.2.2 MATLAB'S `lsqnonlin`

MATLAB'S least-squares, non-linear optimiser is the function `lsqnonlin(fun, x0,lb,ub)`. It minimises the vector-valued function, `fun`, using the vector of initial parameter values, `x0`, where the lower and upper bounds of the parameters are specified in vectors `lb` and `ub`, respectively.

`lsqnonlin` uses an *interior-reflective Newton method* for *large scale* problems. MATLAB defines a *large scale* problem as one containing bounded / unbounded parameters, where the system is not under-determined, i.e., where the number of equations to solve is more than the required parameters. Given that the underlying of the model is liquidly traded, there should exist a rich set of market prices for calibration. So, hopefully, our system will never be under-determined[9]. MATLAB suggests (Coleman & Li 1996) and (Coleman & Li 1994) for further reference on these methods.

The result produced by `lsqnonlin` is dependent on the choice of `x0`, the initial estimate. This is, therefore, not a global optimiser, but rather, a local one. We have no way of knowing whether the solution is a global/local minimum, but if (3.2) is satisfied, the solution is acceptable. If not, then the calibration process would have to be rerun with a different `x0`.

[9]in fact, all one needs is five or more market prices

The appendix, § **A.4.1**, contains information and code on using `lsqnonlin` for calibration. § **3.3** contains a comparison of this method to the others presented here.

3.2.3 Excel's Solver

The standard *Solver* supplied with Excel contains an optimiser that can be used for calibration under specific circumstances. It uses a *Generalized Reduced Gradient* (GRG) method and hence is a local optimiser (for more information go to `www.solver.com`). The calibration results are therefore extremely sensitive to the initial estimates of the parameters. This optimiser should only be used when one is sure that the initial estimates are quite close to the optimal parameter set.

The author has used MATLAB'S *Excel Link* to link the function `HestonCallQuad` between Excel and MATLAB. The spreadsheets `HestonCalibration1` and `HestonCalibration2` use this function to calculate the model prices. Solver can then be run to calibrate the model.

The attractiveness of using Excel and Solver is that most people are comfortable with them and they are available on almost every computer in a workplace. On the down side, Excel is useless at handling calculations with very small numbers. These sort of calculations do appear during the calibration process, e.g., calculating the bound $2\kappa\theta > \sigma^2$, and hence can make your result suspect!

Overall, Excel provides a quick and dirty solution to our problem, under certain conditions. These conditions must be met in order to motivate its use. An advanced version of Solver can be purchased which contains a global optimiser. § **3.3** contains a comparison of this method to the others presented here.

3.2.4 Simulated Annealing (SA)

Simulated Annealing is a probability-based, non-linear, optimiser inspired by the physical process of *annealing*. Its attractiveness lies in the fact that it can:

- process objective functions (eg. (3.1)) possessing quite arbitrary degrees of nonlinearities, discontinuities, and stochasticity.
- process quite arbitrary boundary conditions and constraints imposed on these objective functions.
- be implemented quite easily with the degree of coding quite minimal, relative to other nonlinear optimisation algorithms.

- statistically guarantee finding an optimal solution.

Annealing is a formal term for the ancient art of heating and/or cooling materials to forge pottery, tools, weapons, and works of art (Frost & Heineman 1997). In a typical annealing process melted metal is allowed to cool subject to differing temperatures and pressures. The different combinations of these temperatures and pressures determine the structural properties of the cooled product. Simulated Annealing was borne out of the need to model such processes thereby advancing the types of said materials produced. Annealing has played a key role in human history: entire nations have been won and lost on the abilities of craftsmen to produce fit materials (Frost & Heineman 1997).

The algorithm was first developed by (Metropolis, Rosenbluth, Rosenbluth, Teller & Teller 1953) as a means of finding the equilibrium configuration of a collection of atoms at a given temperature for the actual annealing process. It was (Kirkpatrick, Jr. & Vecchi 1983) who realised the algorithms application to optimisation in general.

The algorithm works in the following way:

1. First the objective function is evaluated at the user-specified initial parameter estimates.

2. Next a random set of parameters is generated based on point 1 above.

3. If the value of the objective function is less than that of point 1, then we 'accept' the parameter set from point 2, else, the parameter set is accepted with probability $\exp\{-\frac{\delta f}{T_k}\}$, where δf is the difference between the objective functions using the parameter sets in points 1 and 2, and T_k is the *temperature*[10] parameter at iteration k, specified by the algorithm.

4. This process is iterated, with the *temperature* parameter decreased at each iteration, until a termination condition is met (usually a specified value of the *temperature* parameter).

The above algorithm not only 'accepts' parameter sets that decrease the objective function, but also that which increases it (subject to the probability constraints). This ensures that the algorithm does not get stuck in a local minimum.

It helps to imagine the objective function, that we want to minimise, as a geographical terrain. We want to find the deepest valley of this terrain. Simulated Annealing approaches this problem in the same way that a bouncing ball can

[10]termed this because of obvious historical reasons.

bounce over mountains, from valley to valley. Initially, when the *temperature* is high, the ball has a lot of energy and can bounce from any valley to any other valley. As time progresses, and the *temperature* parameter decreases, the ball loses energy and settles down in relatively smaller ranges of valleys.

The algorithm requires only the value of the objective function for a given set of parameters. It does not require the form of this objective function. This, in a sense, makes the function a 'black-box' to the optimiser. Constraints are encapsulated within the 'black-box'. This means that the objective function should be able to tell the optimiser whether a set of parameters lies within the required parameter space (in our case, whether $2\kappa\theta > \sigma^2$), and hence limits the search to a feasible space.

Parameters that are determined by the model (eg. the *temperature* parameter) are collectively known as the *annealing scheme*. The annealing scheme broadly determines the efficiency and accuracy of the algorithm. For example, it determines the degree of 'uphill' movement and the rate of decrease of *temperature*, which in turn affects how long the algorithm runs for. It is therefore obvious that the annealing scheme be optimally specified. Such a specification isn't obvious since parameters like *temperature* don't have an explicit/implicit mathematical relationship with the objective function. This sort of specification has therefore become an art form.

To reduce the subjective nature of the aforementioned specification, adaptive methods of Simulated Annealing have been developed. The most famous and widely used of these is *Adpative Simulated Annealing*.

3.2.5 Adaptive Simulated Annealing (ASA)

ASA was developed by the theoretical physicist Lester Ingber. ASA is similar to SA except that it uses statistical measures of the algorithm's current performance to modify its control parameters i.e. the annealing scheme.

A proof is provided by that Ingber shows that ASA is a *global* optimiser. He also provides arguments for ASA's computational effeciency and accuracy.

The C++ code is open-source and available from `www.ingber.com`. It has always been open-source and hence Ingber has had a lot of feedback from users of ASA. Over the years it has, therefore, been modified to a point where it is now perfect. ASA works quite well in it's 'standard' form, but also allows the user to tweak the annealing scheme for greater efficiency. For a detailed discussion on ASA the reader is referred to (Ingber 1993) and (Ingber 1995).

ASA can be implemented in MATLAB by downloading the function `asamin`, written by Shinichi Sakata. `asamin` is a MATLAB gateway function to ASA. This means that `asamin` uses the actual C++ code of Ingeber's ASA through MATLAB. Detailed instructions on how to install and use ASA on one's computer and `asamin` into MATLAB can be found in (Moins 2002). The MATLAB code for calibrating the model using ASA can be found in § **A.4.2**.

Given the thin requirements of the objective function, ASA lends itself well to a variety of applications in different disciplines. Ingber has consulted on projects, where he has successfully implemented ASA, in the following fields:

- Circuit design
- Mathematics/combinatorics
- Data analysis
- Imaging
- Neural networks
- Biology
- Physics
- Geophysics
- Finance
- Military

3.3 Calibration Results

Having an artillery of calibration methods at our disposal we can proceed to calibrate the model and compare the results. Vanilla calls on *Anglo American* shares, listed on the LSE, was used as market data. An example of this data can be found at the end of § **A.4.1**.

`lsqnonlin`, Solver and ASA were run with the same initial estimates on 2 sets of data. The results are as follows:

Method	κ	θ	σ	ρ	$\mathbf{V_0}$	$S(\mathbf{\Omega})$	**Time**
11 October 2005							
Initial Estimate	3	0.05	0.5	-0.5	0.15	386.10	
`lsqnonlin`	4.0481	0.0487	0.6066	-0.8061	0.1450	87.50	62.63 sec
ASA	21.3108	0.1034	2.0099	-0.4998	0.1583	81.80	4.7 hrs
Solver	5.7369	0.0579	0.8141	-0.74783	0.1568	78.10	10 mins
20 October 2005							
Initial Estimate	5	0.057	0.7	-0.75	0.16	148.32	
`lsqnonlin`	15.096	0.1604	2.0859	-0.7416	0.1469	88.58	6 mins
ASA	10	0.1072	1.4189	-0.8236	0.1829	77.38	3.5 hours
Solver	7.3284	0.0745	1.0227	-0.7670	0.1938	89.88	20 mins

where **Time** is the running time of the calibration process[11]. As can be seen, ASA gives the best result in minimising $S(\mathbf{\Omega})$, on average, but is far more computationally demanding. At this point it should be noted that ASA *statistically* guarantees to find an optimal solution. On the 11 October ASA doesn't produce the lowest $S(\mathbf{\Omega})$, but is very close. Although Solver produces the best parameter set, the results will be quite different for a different set of initial estimates. ASA will provide the best estimate if the annealing scheme is optimised.

The reason that ASA runs so long is because it searches the entire parameter space for a global minimum, unlike `lsqnonlin` and Solver which settles down quite quickly into a local minimum. Figures 3.2 and 3.3 illustrate this point. Notice that `lsqnonlin` drops down very quickly to a local minimum whereas ASA keeps on searching for other minima and hence oscillates.

3.4 Comparisons, Considerations and Shortcomings

Relative to ASA, `lsqnonlin` and Solver don't perform that badly in minimising $S(\mathbf{\Omega})$. Given that they are far quicker than ASA, the difference in $S(\mathbf{\Omega})$ is acceptable. This conclusion can only be made for the given set of initial parameters. One should be careful and not generalise such a conclusion as the solutions presented by `lsqnonlin` and Solver are very dependent on the initial parameters. If these were chosen differently, then `lsqnonlin` and Solver's solutions could have been very far off from ASA's. There have been trial runs where the differences between `lsqnonlin`, Solver and ASA were quite significant (ASA producing the best estimate). As mentioned above, `lsqnonlin` and Solver should only be used when one is confident that the solution to (3.1) isn't very far from the initial estimates.

[11]This was run on a Pentium M 1.6 Ghz, 512MB RAM.

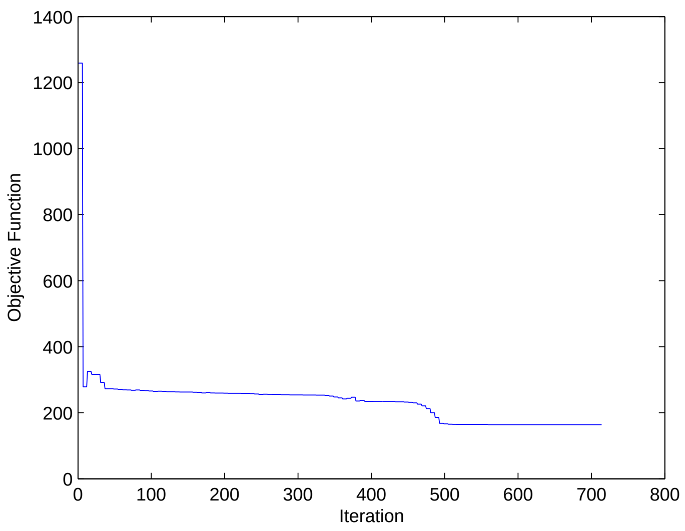

Figure 3.2: *This illustrates how the objective function is minimised when using* `lsqnonlin`.

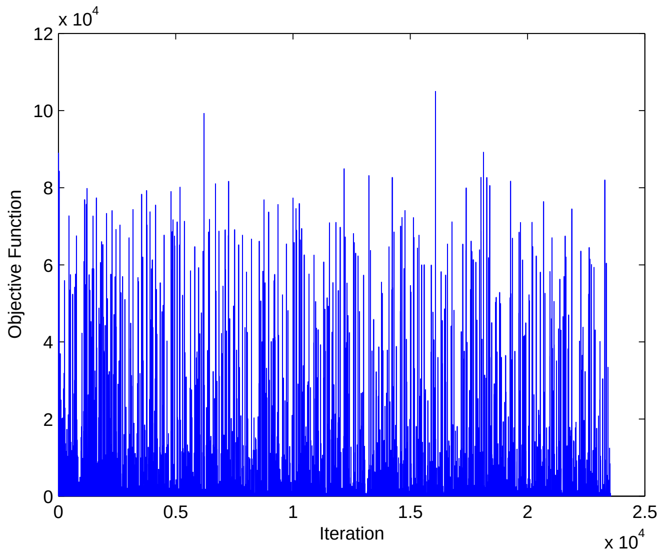

Figure 3.3: *This illustrates how the objective function is minimised when using ASA.*

It should also be noted that ASA is being used in its raw form. There has been no tweaking of the annealing schedule which can result in a significant decrease in run-time. Also, for a random guess of the initial parameters (e.g. this would be done when the model is calibrated for the very first time), ASA is the best option

The obvious question is, which method do we use? There isn't a straight answer to this. The choice of method should be dependent on the amount of information available related to parameter values and the state of the market. If a global minimum was obtained yesterday (by using ASA), and the market conditions are quite normal (i.e. no crashes or dramatic movements since yesterday) then `lsqnonlin` and Solver can quite safely be used. We expect that today's parameters are in a small neighborhood around yesterday's. If there has been a crash or dramatic market movement then such an expectation is unreal. In this case ASA would have to be used. So, there should be a marriage of the different methods to create efficiency and not sacrifice accuracy.

Another issue to consider is the stationarity of parameters. Due to time constraints placed on such an honours project, such a task wasn't able to perform. But, (Mikhailov & Nögel 2003) say that the parameters aren't stationary, and hence the comment above about calibrating the model each day. The stationarity that most practitioners on this topic refer to is that of the parameters that solve (3.1). But, a parameter set that we use need not be this set. An acceptable set is one that satisfies (3.2). An initial problem with ASA, `lsqnonlin` and Solver is that they don't tell us anything about the multiplicity of solutions of the calibration problem. But we can employ a clever trick and adapt ASA and `lsqnonlin` to do this.

First, calculate the RHS of (3.2). Call this value `SumBidOffer`. Remember that the objective function is the LHS of (3.2) that is evaluated at each iteration. By adding the following logical test in functions `HestonDifference` and `HestonCostFunc` we can store those parameter sets that satisfy (3.2):

```
if (sum(PriceDifference.^2) < SumBidOffer
    PossibleSolutions{j} = input;
    j = j+1
end
```

where `j` is the appropriate index of the *cell matrix*[12] `PossibleSolutions`. The model should now be calibrated every day/month until enough `PossibleSolutions` matices are available. Comparing the `PossibleSolutions` matrices over the different days/months it could be possible to extract a set of parameters that are more or less an element of each `PossibleSolutions`. This creates a stationary set of parameters that satisfy (3.2).

Extracting the common set of parameters is not as trivial as it may seem. We seek a 5-dimensional *set* of stationary parameters and hence the usual arguments

[12]a cell matrix in MATLAB is a matrix of matrices

that are used in such stationarity tests, in the 1-dimensional case, are not that helpful.

3.5 Other Methods

For completeness a few of the other calibration methods will be presented.

(Shu & Zhang 2004) propose a 2-step **Indirect Inference** method of calibrating the model. First, the recently developed simulation-based "indirect inference method" (time-series relative) is used to estimate the parameters for the dynamics of the asset. A non-linear least squares method is then used to calibrate the remaining parameters, similar to what is presented above.

As mentioned before, the parameters have no relation to their time-series counterparts, so it seems counter-intuitive to estimate some of the parameters using some time-series method. But it is exactly for this reason that we can estimate a *few* of the parameters by some other method and then use a non-linear least squares method to fit the model. This effectively reduces the dimension over which (3.1) has to be minimised.

This seems like an extension to the method presented here and could be an interesting extension to this project.

(Johnson & Lee 2004) proposes calibrating the model using an **Ensemble** method. This method does not involve fitting the parameters to market data as most methods do. Rather, it uses an ensemble of parameter sets. Each set of the ensemble is assigned a weight which is attained by just one matrix inversion. The calibrated ensemble will then reproduce exact market prices. (Johnson & Lee 2004) state that the attractiveness in the model lies in it's simplicity and robustness in being able to calibrate to any type of instrument, e.g., barrier options, forward starting options, etc. Also, hedge ratios are easily calculated.

Lastly, while reading through the *Wilmott* forums, it seems that the calculation of (3.1) can be carried out using **genetic algorithms**. MATLAB contains some genetic algorithms in the *Direct Search Toolbox* which is an extension of the *Optimisation Toolbox*. However, (Ingber 1993) notes that these algorithms are much slower that ASA, though.

Conclusion

As Mathematicians (or students thereof) we can get caught up in the magic of Mathematics. It is important to remember that the Heston model is, essentially, a model. It is a mathematical tool that models something that is infinitely complex. It therefore cannot wholly capture the complicated and diverse dynamics that exist in reality, of volatility.

Calibration of the model is an important task. Energy should not be wasted, though, on over-fitting the model. Such efforts can lead to spurious results. After all, one is trying to perfectly fit a model that doesn't perfectly explain the real world.

This doesn't make stochastic models, in general, useless. If the underlying assumptions are carefully understood and its application carefully applied then stochastic models are powerful tools. They are the best that we have in an unpredictable world without which we would be far worse off!

Appendix

A.1 Simulation of Probability Densities

Monte Carlo simulation was performed by discretising the stochastic processes using the Euler-Maruyama method. This resulted in,

$$\begin{aligned} S_t &= S_{t-1} + rS_{t-1}dt + \sqrt{V_{t-1}}S_{t-1}\sqrt{dt}Z_t^1 \\ V_t &= V_{t-1} + \kappa(\theta - V_{t-1})dt + \sigma\sqrt{V_{t-1}}\sqrt{dt}Z_t^2 \end{aligned}$$

where $\{Z_t^1\}_{t\geq 0}$ and $\{Z_t^2\}_{t\geq 0}$ are standard normal random variables with correlation ρ. The above can be made computationally easier by expressing $\{Z_t^1\}_{t\geq 0}$ and $\{Z_t^2\}_{t\geq 0}$, as a function of independent standard normal random variables, using the Cholesky decomposition,

$$\begin{aligned} Z_t^1 &= \phi_t^1 \\ Z_t^2 &= \rho\phi_t^1 + \sqrt{1-\rho^2}\phi_t^2 \end{aligned}$$

where $\{\phi_t^1\}_{t\geq 0}$ and $\{\phi_t^2\}_{t\geq 0}$ are independent standard normal random variables. MATLAB code for the simulation of figure 1.1 is as follows:

```
S0 = 10;        V0 = .01;       r = 0;       k = 2;
theta =.01;     sigma= .1;      delT = .02;  rho =-0.9;
numberOfSimulations = 1000000;

i = 1:numberOfSimulations;
NormRand1 = randn(1,numberOfSimulations);
NormRand2 = randn(1,numberOfSimulations);

   S = zeros(1,numberOfSimulations);
   V = zeros(1,numberOfSimulations);
   V(i) = V0 + k*(theta - V0)*delT + sigma*sqrt(V0)* ...
   (rho*NormRand1 + sqrt(1- rho^2)*NormRand2)*sqrt(delT);
   V = abs(V);     %prevents negative volatilities
   S(i) = S0 + r*S0*delT + S0*V.^(0.5).*NormRand1*sqrt(delT);
```

The code is broadly similar for the simulation of figure 1.5.

A.2 Generating Volatility Surfaces and Skews

Prices of the European call options are calculated, using the Heston closed-form solution (§ **2.1**), at different maturities and strikes. These prices are then equated to the Black-Scholes-Merton (BSM) solution and the volatility that satisfies this equation is evaluated. The function `blsimpv` returns the BSM implied volatility for a given set of parameters. The following script file, `VolSmile.m` generates the volatility surfaces of § **1.2** (figures 1.3, 1.2, 1.4).

```
strikes = linspace(.8,1.2,11);
mats = linspace(.3,3,11);       %maturities


for i = 1:11
   for j = 1:11
       price = HestonCallQuad(2,.04,.1,0.5,.04,.01,mats(i),1...
                                                ,strikes(j));
       prices(i,j) = price;
       Volatility(i,j) = blsimpv(1, strikes(j), 0.01 , ...
                                               mats(i), price);
   end
end

[strike mat] = meshgrid(strikes,mats);
surf(mat,strike,Volatility),xlabel('Maturity(years)'), ...
ylabel('Strike'),Title('\rho = 0.5'),zlabel('Implied ...
                                                Volatility');
figure;

for i = 1:11
   for j = 1:11
       price = HestonCallQuad(2,.04,.1,0,.04,.01,mats(i),1...
                                              ,strikes(j));
       prices(i,j) = price;
       Volatility(i,j) = blsimpv(1, strikes(j), 0.01 , ...
                                            mats(i), price);
   end
end

surf(mat,strike,Volatility),xlabel('Maturity(years)'),
ylabel('Strike'),Title('\rho = 0'),zlabel('Implied ...
                                              Volatility');
figure;

for i = 1:11
   for j = 1:11
       price = HestonCallQuad(2,.04,.1,-0.5,.04,.01,mats(i),...
                                               1,strikes(j));
```

```
        prices(i,j) = price;
        Volatility(i,j) = blsimpv(1, strikes(j), 0.01 ,...
                                                mats(i), price);
    end
end

surf(mat,strike,Volatility),xlabel('Maturity(years)'),
ylabel('Strike'),Title('\rho = -0.5'),zlabel('Implied...
                                                Volatility');
```

The following script, `HestonVolSmileVolVol.m`, generates figures 1.6, 1.7 and 1.8 of § **1.2**.

```
strikes = linspace(.8,1.2,11);
volvols = [.1:.1:.4];
styleV = {['-']['--'] ['-.'] [':']};
colourV = {['k'] ['b'] ['r'] ['m']};

   for i = 1:4
       for j = 1:11
           price = HestonCallQuad(2,.04,volvols(i),0.5,.04, ...
                                 .01,1,1,strikes(j));
           prices(i,j) = price;
           Volatility(i,j) = blsimpv(1, strikes(j), 0.01 , ...
                                                   1, price);
       end
   plot(strikes,Volatility(i,:),[char(colourV(i)), ...
   char(styleV(i))]),ylabel('Implied Volatility'), ...
                     xlabel('Strike'),Title('\rho = 0.5');
   hold on;
   end

   legend('\sigma = 0.1','\sigma = 0.2','\sigma = 0.3', ...
                                '\sigma = 0.4') figure;

   for i = 1:4
       for j = 1:11
           price = HestonCallQuad(2,.04,volvols(i),0,.04, ...
                               .01,1,1,strikes(j));
           prices(i,j) = price;
           Volatility(i,j) = blsimpv(1, strikes(j), 0.01 , ...
                                                   1, price);
       end

       plot(strikes,Volatility(i,:),[char(colourV(i)), ...
       char(styleV(i))]),ylabel('Implied Volatility'), ...
                         xlabel('Strike'),Title('\rho = 0');
```

```
hold on;
end

legend('\sigma = 0.1','\sigma = 0.2','\sigma = 0.3', ...
                                   '\sigma = 0.4') figure;

for i = 1:4
    for j = 1:11
        price = HestonCallQuad(2,.04,volvols(i),-0.5,.04,...
                                        .01,1,1,strikes(j));
        prices(i,j) = price;
        Volatility(i,j) = blsimpv(1, strikes(j), 0.01 , ...
                                                   1, price);
    end

    plot(strikes,Volatility(i,:),[char(colourV(i)), ...
    char(styleV(i))]),ylabel('Implied Volatility'), ...
                  xlabel('Strike'),Title('\rho = -0.5');
    hold on;
end

legend('\sigma = 0.1','\sigma = 0.2','\sigma = 0.3', ...
                                        '\sigma = 0.4')
```

A.3 Evaluation of the Closed-Form Solution

Presented here are the two methods discussed in § **2.2** and § **2.3** for computationally evaluating the closed-form solution for a European call.

A.3.1 European Call using FFT

The following function calculates at-the-money and in-the-money European call prices using FFT.

```
function CallValue = HestonCallFft(kappa,theta,sigma,rho,r ...
                                              ,v0,s0,strike,T)

%kappa = rate of reversion
%theta = long run variance
%sigma = Volatility of volatility
%v0    = initial Variance
%rho   = correlation
%T     = Time till maturity
```

```
%r     = interest rate
%s0    = initial asset price

x0 = log(s0);
alpha = 1.25;
N= 4096;
c = 600;
eta = c/N;
b =pi/eta;
u = [0:N-1]*eta;
lambda = 2*b/N;
position = (log(strike) + b)/lambda + 1;  %position of call
                                          %value in FFT
                                          %matrix

   v = u - (alpha+1)*i;
   zeta = -.5*(v.^2 +i*v);
   gamma = kappa - rho*sigma*v*i;
   PHI = sqrt(gamma.^2 - 2*sigma^2*zeta);
   A = i*v*(x0 + r*T);
   B = v0*((2*zeta.*(1-exp(-PHI.*T))))./(2*PHI - ...
                       (PHI-gamma).*(1-exp(-PHI*T))));
           C = -kappa*theta/sigma^2*(2*log((2*PHI - ...
                       (PHI-gamma).*(1-exp(-PHI*T)))./ ...
                             (2*PHI))  +  (PHI-gamma)*T);

       charFunc = exp(A + B + C);
       ModifiedCharFunc = charFunc*exp(-r*T)./(alpha^2 ...
                        + alpha - u.^2 + i*(2*alpha +1)*u);
       SimpsonW = 1/3*(3 + (-i).^[1:N] - [1, zeros(1,N-1)]);
       FftFunc = exp(i*b*u).*ModifiedCharFunc*eta.*SimpsonW;
       payoff = real(fft(FftFunc));
       CallValueM = exp(-log(strike)*alpha)*payoff/pi;
       format short;
       CallValue = CallValueM(round(position));
```

The following function calculates out-of-the-money European call prices using FFT.

```
function CallValue = HestonCallFftTimeValue(kappa,theta,...
                                  sigma,rho,r,v0,s0,strike,T)

%kappa = rate of reversion
%theta = long run variance
%sigma = Volatility of volatility
%v0    = initial Variance
```

```
%rho   = correlation
%T     = Time till maturity
%r     = interest rate
%s0    = initial asset price

   x0 = log(s0);
   alpha = 1.25;
   N= 4096;
   c = 600;
   eta = c/N;
   b =pi/eta;
   u = [0:N-1]*eta;
   lambda = 2*b/N;
   position = (log(strike) + b)/lambda + 1;  %position of call
                                             %value in FFT matrix

   w1 = u-i*alpha;
   w2 = u+i*alpha;
   v1 = u-i*alpha -i;
   v2 = u+i*alpha -i;

       zeta1 = -.5*(v1.^2 +i*v1);
       gamma1 = kappa - rho*sigma*v1*i;
       PHI1 = sqrt(gamma1.^2 - 2*sigma^2*zeta1);
       A1 = i*v1*(x0 + r*T);
       B1 = v0*((2*zeta1.*(1-exp(-PHI1.*T)))./(2*PHI1 - ...
                          (PHI1-gamma1).*(1-exp(-PHI1*T))));
       C1 = -kappa*theta/sigma^2*(2*log((2*PHI1 - ...
              (PHI1-gamma1).*(1-exp(-PHI1*T)))./(2*PHI1)) ...
                                  +  (PHI1-gamma1)*T);
       charFunc1 = exp(A1 + B1 + C1);
       ModifiedCharFunc1 = exp(-r*T)*(1./(1+i*w1) - ...
                exp(r*T)./(i*w1) - charFunc1./(w1.^2 - i*w1));

       zeta2 = -.5*(v2.^2 +i*v2);
       gamma2 = kappa - rho*sigma*v2*i;
       PHI2 = sqrt(gamma2.^2 - 2*sigma^2*zeta2);
       A2 = i*v2*(x0 + r*T);
       B2 = v0*((2*zeta2.*(1-exp(-PHI2.*T)))./(2*PHI2 - ...
                        (PHI2-gamma2).*(1-exp(-PHI2*T))));
       C2 = -kappa*theta/sigma^2*(2*log((2*PHI2 - ...
              (PHI2-gamma2).*(1-exp(-PHI2*T)))./(2*PHI2)) ...
                                      +  (PHI2-gamma2)*T);
       charFunc2 = exp(A2 + B2 + C2);
       ModifiedCharFunc2 = exp(-r*T)*(1./(1+i*w2) - ...
               exp(r*T)./(i*w2) - charFunc2./(w2.^2 - i*w2));

       ModifiedCharFuncCombo = (ModifiedCharFunc1 - ...
                                    ModifiedCharFunc2)/2 ;
```

```
        SimpsonW = 1/3*(3 + (-1).^[1:N] - [1, zeros(1,N-1)]);
        FftFunc = exp(i*b*u).*ModifiedCharFuncCombo*eta.*...
                                                    SimpsonW;
        payoff = real(fft(FftFunc));
        CallValueM = payoff/pi/sinh(alpha*log(strike));
        format short;
        CallValue = CallValueM(round(position));
```

A.3.2 European Call using Numerical Integration

The function `HestonCallQuad(kappa,theta,sigma,rho,v0,r,T,s0,K)` calculates the value of a European call. It calls `HestonP(kappa,theta,sigma, rho,v0,r,T,s0,K,type)`, where `type` = 1,2, which evaluates (2.19), either using an adaptive *Gauss Lobatto* rule or adaptive *Simpson's Rule*. This, in turn, calls `HestonPIntegrand(phi,kappa,theta,sigma,rho,v0,r,T,s0,K,type)`, which evaluates the integrand of (2.19). This, in turn calls `Hestf(phi,kappa,theta, sigma,rho,v0,r,T,s0,type)`, which evaluates the 'f' function in the integrand of (2.19). All variable names are the same as used in (Heston 1993).

```
function call = HestonCallQuad(kappa,theta,sigma,rho,v0,r,T,...
                                                        s0,K)
warning off;
call = s0*HestonP(kappa,theta,sigma,rho,v0,r,T,s0,K,1) - ...
      K*exp(-r*T)*HestonP(kappa,theta,sigma,rho,v0,r,T,s0,K,2);


function ret = HestonP(kappa,theta,sigma,rho,v0,r,T,s0,K,type)
ret = 0.5 + 1/pi*quadl(@HestonPIntegrand,0,100,[],[],kappa, ...
                             theta,sigma,rho,v0,r,T,s0,K,type);


function ret = HestonPIntegrand(phi,kappa,theta,sigma,rho, ...
                                            v0,r,T,s0,K,type)
ret = real(exp(-i*phi*log(K)).*Hestf(phi,kappa,theta,sigma, ...
                                 rho,v0,r,T,s0,type)./(i*phi));


function f = Hestf(phi,kappa,theta,sigma,rho,v0,r,T,s0,type);
   if type == 1
       u = 0.5;
       b = kappa - rho*sigma;
   else
       u = -0.5;
       b = kappa;
   end
a = kappa*theta; x = log(s0);
d = sqrt((rho*sigma*phi.*i-b).^2-sigma^2*(2*u*phi.*i-phi.^2));
g = (b-rho*sigma*phi*i + d)./(b-rho*sigma*phi*i - d);
C = r*phi.*i*T + a/sigma^2.*((b- rho*sigma*phi*i + d)*T - ...
2*log((1-g.*exp(d*T))./(1-g)));
D = (b-rho*sigma*phi*i + d)./sigma^2.*((1-exp(d*T))./ ...
                                         (1-g.*exp(d*T)));
f = exp(C + D*v0 + i*phi*x);
```

A.4 Calibration

The following contains the MATLAB code for calibrating the Heston model.

A.4.1 Using MATLAB'S `lsqnonlin`

The script file `HestonLsCalibration.m` initiates the calibration process. It creates a handle on the function `HestonDifferences` that calculates the differences between the model and market prices within `lsqnonlin`. The '`load OptionData.txt`' line imports the strikes, maturities, market prices, bid and offers, etc., of the options and underlying from the file `OptionData.txt`[13]. The first parameter of `input` that MATLAB sends into `HestonDifference` is $2\kappa\theta - \sigma^2$. It is done here in this way because it is easier to encapsulate the constraint $2\kappa\theta - \sigma^2 > 0$. Reasonable bounds on the parameters were chosen relative to this.

```
clear;
global OptionData;
global NoOfOptions;
global NoOfIterations;
global PriceDifference;

NoOfIterations = 0;

load OptionData.m ;
%OptionData = [r,T,S0,K,Option Value,bid,offer]

Size = size(OptionData);
NoOfOptions = Size(1);
```

[13]an example of the composition of `OptionData.m` is presented at the end of this subsection

```
%input sequence in initial vectors [2*kappa*theta - sigma^2,...
%                                        theta,sigma,rho,v0]
x0 = [6.5482   0.0731 2.3012 -0.4176 0.1838];
lb = [0 0 0 -1 0];
ub = [20 1 5 0 1];

options = optimset('MaxFunEvals',20000);
%sets the max no. of iteration to 20000 so that termination
%doesn't take place early.

tic;
Calibration = lsqnonlin(@HestonDifferences,x0,lb,ub);
toc;

Solution = [(Calibration(1)+Calibration(3)^2)/ ...
               (2*Calibration(2)), Calibration(2:5)];



function ret = HestonDifferences(input)

global NoOfOptions;
global OptionData;
global NoOfIterations;
global PriceDifference;

NoOfIterations = NoOfIterations + 1;
%counts the no of iterations run to calibrate model

for i = 1:NoOfOptions
    PriceDifference(i) = (OptionData(i,5)-HestonCallQuad( ...
    (input(1)+input(3)^2)/(2*input(2)),input(2), ...
    input(3),input(4),input(5), ...
    OptionData(i,1),OptionData(i,2),OptionData(i,3), ...
    OptionData(i,4)))/sqrt((abs(OptionData(i,6)- ...
                                       OptionData(i,7))));
    %input matrix = [kappa theta sigma rho v0]
end

ret = PriceDifference';
```

As mentioned before, the file `OptionData.m` is a text file containing the market data for calibration. Vanilla calls written on Anglo American shares that trade on the LSE have been used. For illustrative purposes, `OptionData` contains the following information:

r-q	Term	Spot	Strike	Mid Price	Bid	Offer
2.2685%	0.126027	1544.50	1000.00	559.00	553.00	565.00
2.2685%	0.126027	1544.50	1050.00	509.50	503.50	515.50
2.2685%	0.126027	1544.50	1100.00	460.00	454.00	466.00
2.2685%	0.126027	1544.50	1150.00	411.00	405.00	417.00
2.2685%	0.126027	1544.50	1200.00	362.50	356.50	368.50
2.2342%	0.375342	1544.50	1200.00	386.00	378.50	393.50
2.2685%	0.126027	1544.50	1250.00	315.00	309.00	321.00
2.2342%	0.375342	1544.50	1250.00	345.50	338.00	353.00
2.2685%	0.126027	1544.50	1300.00	269.50	263.50	275.50
2.2342%	0.375342	1544.50	1300.00	300.50	293.00	308.00
2.2685%	0.126027	1544.50	1350.00	223.00	217.00	229.00
2.2342%	0.375342	1544.50	1350.00	259.00	251.50	266.50
2.1947%	0.627397	1544.50	1350.00	281.00	272.00	290.00
2.2685%	0.126027	1544.50	1400.00	179.00	176.00	182.00
2.2342%	0.375342	1544.50	1400.00	221.00	213.50	228.50
2.1947%	0.627397	1544.50	1400.00	244.00	235.00	253.00
2.2685%	0.126027	1544.50	1450.00	140.00	136.00	144.00
2.2342%	0.375342	1544.50	1450.00	180.00	174.00	186.00
2.1947%	0.627397	1544.50	1450.00	207.50	198.50	216.50
2.2685%	0.126027	1544.50	1500.00	105.00	102.00	108.00
2.2342%	0.375342	1544.50	1500.00	149.50	145.00	154.00
2.1947%	0.627397	1544.50	1500.00	173.00	166.00	180.00
2.2685%	0.126027	1544.50	1600.00	56.50	51.50	61.50
2.2342%	0.375342	1544.50	1600.00	96.00	92.00	100.00
2.1947%	0.627397	1544.50	1600.00	121.00	114.00	128.00
2.2685%	0.126027	1544.50	1700.00	23.50	20.50	26.50
2.2342%	0.375342	1544.50	1700.00	57.25	51.00	63.50
2.1947%	0.627397	1544.50	1700.00	81.50	77.00	86.00
2.2685%	0.126027	1544.50	1800.00	10.00	7.00	13.00
2.2342%	0.375342	1544.50	1800.00	32.50	28.00	37.00
2.1947%	0.627397	1544.50	1800.00	50.50	44.50	56.50
2.2685%	0.126027	1544.50	1900.00	4.50	2.00	7.00
2.2342%	0.375342	1544.50	1900.00	18.25	14.50	22.00
2.1947%	0.627397	1544.50	1900.00	35.50	29.50	41.50

A.4.2 Adaptive Simulated Annealing (ASA) in MATLAB

The scheme that controls the ‘acceptance’ of new solutions is so simple that the cost of implementing ASA is purely dependent on the computational efficiency of evaluating the objective function. In our case, calculation of the objective function for a given set of parameters entails the evaluation of a large number of options. This makes ASA very computationally demanding and time consuming. The following script, `HestonASACalibration`, calibrates the model using ASA. It uses the function `HestonCostFunc`

```
clear;
global OptionData;
global NoOfOptions;
global NoOfIterations;
global PriceDifference;
global ObjectiveFunc;

NoOfIterations = 0;

load OptionData.m ;
%OptionData = [r,T,S0,K,Option Value,bid,offer]

Size = size(OptionData);
NoOfOptions = Size(1);

%input sequence in initial vectors [kappa,theta,sigma,rho,v0]
x0 = [0.030854841   0.999999922 0.248415019 -1  0.08977599];
lb = [0 0 0 -1 0];
ub = [10 1 5 0 1];

asamin('set','test_in_cost_func',0)

tic;
[fstar, xstar, grad, hessian, state] = asamin('minimize',...
                    'HestonCostFunc',x0',lb',ub',-1*ones(5,1));
toc;



function [cost , flag] = HestonCostFunc(input)
global NoOfOptions;
global OptionData;
global NoOfIterations;
global PriceDifference;
global ObjectiveFunc;

%input matrix = [kappa theta sigma rho v0]

NoOfIterations = NoOfIterations + 1;

if (2*input(1)*input(2)<=input(3)^2)  %test for constraint
   flag = 0;   %flag = 0 if contraint is violated, else = 1
   cost = 0
else

   for i = 1:NoOfOptions
       PriceDifference(i) = (OptionData(i,5)-HestonCallQuad(...
       input(1),input(2),input(3),input(4),input(5), ...
       OptionData(i,1),OptionData(i,2),OptionData(i,3),...
       OptionData(i,4)))/sqrt((abs(OptionData(i,6)- ...
                                             OptionData(i,7))));
   end

   cost = sum(PriceDifference.^2)
   ObjectiveFunc(NoOfIterations) = cost; %stores the path of
   flag = 1;                             %the optimiser
end
```

References

Bakshi, G., Cao, C. & Chen, Z. (1997), 'Empirical performance of alternative option pricing models', *Journal of Finance* **52**, 20032049.

Black, F. & Scholes, M. (1973), 'Valuation of options and corporate liabilities', *Journal of Political Economy* **81**, 637–654.

Carr, P. & Madan, D. B. (1999), 'Option evaluation the using fast fourier transform', *Journal of Computational Finance* **2**(4), 61–73.

Cheney, W. & Kincaid, D. (1999), *Numerical Mathematics and Computing*, 4 edn, Brooks/Cole Publishing Company.

Chiarella, C., Craddock, M. & El-Hassan, N. (2000), The calibration of stock option pricing models using inverse problem methodology, Research Paper Series 39, Quantitative Finance Research Centre, University of Technology, Sydney. http://ideas.repec.org/p/uts/rpaper/39.html.

Coleman, T. F. & Li, Y. (1994), 'On the convergence of reflective newton methods for large-scale nonlinear minimization subject to bounds', *Mathematical Programming*.

Coleman, T. F. & Li, Y. (1996), 'An interior, trust region approach for nonlinear minimization subject to bounds', *SIAM Journal on Optimization* **6**, 418–445.

Cont, R. (2001), 'Empirical properties of asset returns: stylized facts and statistical issues', *Quantitative Finance* **1**, 223–236.

Cont, R. (2005), 'Recovering volatilty from option prices by evolutionary optimisation', *Research Paper Series*.

Cox, J. C., Ingersoll, J. E. & Ross, S. A. (1985), 'A theory of the term structure of interest rates', *Econometrica* **53**, 385–407.

Frost, R. & Heineman, P. (1997), Simulated annealing: A heuristic for parallel stochastic optimization., *in* 'PDPTA', pp. 1595–.

Gander, W. & Gautschi, W. (1998), Adaptve quadrature - revisited, Technical report, Departement Informatik, ETH Zürich.

Gatheral, J. (2004), ‘Lecture 1: Stochastic volatility and local volatility’, *Case Studies in Financial Modelling Notes, Courant Institute of Mathematical Sciences*.

Heston, S. L. (1993), ‘A closed-form solution for options with stochastic volatility with applications to bonds and currency options’, *The Review of Financial Studies* **6**(2), 327–343.

Hong, G. (2004), ‘Forward smile and derivative pricing’.

Ingber, A. L. (1993), ‘Simulated annealing: Practice versus theory’, *Journal of Mathematical Computational Modelling* **18**(11), 29–57.

Ingber, A. L. (1995), ‘Adaptive simulated annealing (asa): Lessons learned’, *Control and Cybernetics*.

Johnson, S. & Lee, H. (2004), ‘Capturing the smile’, *Risk Magazine* **March**, 89–93.

Kirkpatrick, S., Jr., C. D. G. & Vecchi, M. P. (1983), ‘Optimization by simulated annealing’, *Science* **220**(4598), 671–680.

Majmin, L. (2005), Local and stochastic volatility models: An investigation into the pricing of exotic equity options, Master’s thesis, University of the Witwatersrand, South Africa.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A. & Teller, E. (1953), ‘Equation of state calculations by fast computing machines’, *Journal of Chemisry and Physics* **21**(6), 1087–1092.

Mikhailov, S. & Nögel, U. (2003), ‘Hestons stochastic volatility model implementation, calibration and some extensions’, *Wilmott*.

Moins, S. (2002), Implementation of a simulated annealing algorithm for MATLAB, Technical report, Linköping Institute of Technology.

Shu, J. & Zhang, J. E. (2004), ‘Pricing s&p 500 index options under stochastic volatility with the indirect inference method’, *Journal of Derivatives Accounting* **1**(2), 1–16.

Xu, J. (2003), Pricing and hedging under stochastic volatility, Master’s thesis, Peking University, China.