MÄLARDALEN UNIVERSITY
SWEDEN

Department of Mathematics and Physics

MASTER THESIS IN MATHEMATICS/ APPLIED MATHEMATICS

Stochastic Volatility Models in Option Pricing

by

Michail Kalavrezos

Michael Wennermo

Magisterarbete i matematik/tillämpad matematik

Code: MDH.IMA.MAT.0093 (2007) 30 hp – AF

MÄLARDALEN UNIVERSITY
SWEDEN
Department of Mathematics and Physics

Master thesis in mathematics/applied mathematics

Date: 2007-12-17

Project name: Stochastic Volatility Models in Option Pricing

Authors: Michail Kalavrezos & Michael Wennermo

Supervisor: Senior Lecturer Jan Röman

Examiner: Professor Dmitrii Silvestrov

Comprising: 30 hp

Abstract

In this thesis we have created a computer program in Java language which calculates European call- and put options with four different models based on the article *The Pricing of Options on Assets with Stochastic Volatilities* by John Hull and Alan White. Two of the models use stochastic volatility as an input. The paper describes the foundations of stochastic volatility option pricing and compares the output of the models. The model which better estimates the real option price is dependent on further research of the model parameters involved.

Keywords: Option pricing, stochastic volatility models, Monte Carlo simulation, Java applet, variance reduction techniques

Acknowledgements

We would like to thank our supervisor Jan Röman for his valuable support and aid. Througout the process of our thesis his knowledge has been available and of great help for us.

Table of Contents

List of Figures

Section I

Introduction

There has been vast work on option pricing since the appearance of the celebrated Black and Scholes formula. The foundations [1] of all the techniques had been laid long time before by Charles Castelli, who in 1877 talked about the different purposes of options in his book titled *The Theory of Options in Stocks and Shares*. The first known analytical valuation for options was presented in 1897 by Louis Bachelier in his mathematics dissertation *Theorie de la Speculation*. The pitfalls in his work were that the process he chose generated negative security prices and the option prices were in some cases greater than the prices of the underlying assets. The next step on option pricing was conducted in 1955 by Paul Samuelson in his paper *Brownian Motion in the Stock Market*. Shortly after that Richard Kruizenga brought an extension to the same subject in his dissertation titled *Put and Call Options: A Theoretical and Market Analysis*. A more advanced model (at least in theory) was presented in 1962 by A. James Boness in his dissertation *A Theory and Measurement of Stock Option Value*. Eleven years later Fischer Black and Myron Scholes introduced their option pricing model. After this milestone in finance, numerous papers have examined the subject of option pricing [2] with and without the same assumptions. Cox and Ross (1976b) derived European option prices under various alternatives. Merton (1976) proposed a jump-diffusion model. He also dealt with option pricing under stochastic interest rate in 1973. The distributional hypothesis (normal) was also relaxed and models for pricing European options under different distributions appeared; Naik's (1993) regime-switching model, the implied binomial tree model of Derman and Kani (1994) and Rubinstein (1994). The assumption of constant volatility was also relaxed and models for pricing options under stochastic volatility appeared in Hull and White (1987) [8], Johnson and Shanno (1987), Scott (1987), Wiggins(1987), Stein and Stein (1991) and Heston (1993) to mention a few. Hull and White [8], hereafter referred to as H&W, provide a solution for the option pricing through a power series approximation technique which is compared with the Black and Scholes formula. H&W also provide two models with stochastic volatility generated by Monte Carlo simulation.

In this paper we apply all the models referred to in the H&W paper and we examine the outputs of the different models.

Our purpose is to:

- Create an application in a Java Applet where all the models are calculating the European call and put option prices of assets with no dividends.
- Test the effect of the parameters of the stochastic models on volatility and option price.

The reason we chose this particular paper is that; firstly it provides a good start in the study of stochastic volatility models and secondly it may provide a significant help in continuing with more advanced models in the future.

The paper is organized as follows; Section II provides the theoretical background on which the models rely. Section III contains a complete user's guide for the applet. Section IV is where the results of the tests are presented as well as comments on the results. Section V contains conclusions and the complete program is found in the appendix.

Section II

Option Pricing

Black-Scholes Model

The Black-Scholes formula is explained in this section since it provides a vital part of the solutions used in the H&W article. It is one of the most common option valuation models and was first developed for European options on non-dividend paying stocks [7]. Today there exist more complicated extensions that calculate prices of American options on stocks as well as other underlying assets.

The inputs required for the pricing of a call option on a non-dividend paying stock with the Black-Scholes formula are current stock price, strike price, interest rate, volatility and time to maturity. All these parameters are easily observed in the market with the exception of volatility.

Although the Black-Scholes model has been and still is a highly used option pricing framework, many of its assumptions may be disputed as to what extent they reflect the true market. The assumption which will be of most interest in this paper is that of constant versus stochastic volatility during the lifetime of the derivative. The Black-Scholes formula for the call price is:

$$C = S_t N(d_1) - Ke^{-r(T-t)} N(d_2) \quad \textbf{(Eq. 2.1)}$$

Where S_t is the stock price at time t, T is the maturity date, K is the strike price, $N(d_x)$ is the cumulative normal distribution and d_1, d_2 are as follows.

$$d_1 = \frac{\log\left(\frac{S_t}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)(T-t)}{\sqrt{\sigma^2(T-t)}}$$

$$d_2 = d_1 - \sqrt{\sigma^2(T-t)}$$

Where σ^2 is the variance of the continuously compounded return over the horizon [0, T]

Strong Law of Large Numbers

For a family of independent and identically distributed (IID) random variables $X_1, X_2, \ldots$, suppose that the mean $\mu = E[X_1]$ exists. Then

$$\lim_{n\to\infty} \frac{X_1 + X_2 + ... + X_n}{n} = \mu \quad \textbf{(Eq. 2.2)}$$

with probability one. The Law of Large numbers states that the sample mean of n numbers converges to the population mean almost surely as n tends to infinity [3].

Central Limit theorem

For a family of IID random variables $X_1, X_2, \ldots,$ with finite mean μ and finite variance $\sigma^2 > 0$, define

$$Z_n = \frac{X_1 + X_2 + ... + X_n - n\mu}{\sigma\sqrt{n}}, \quad n = 1, 2, \ldots$$

Then

$$\lim_{n\to\infty} P\{Z_n \le x\} = \Phi(x), \qquad x \in R$$

where $\Phi(x)$ is the standard normal function [3].

Monte Carlo simulation

In [3] we find a short description of the Monte Carlo idea which is reproduced here. For a family of IID random variables $X_1, X_2, \ldots$ and a real-valued function $h(x)$, the function $h(X)$ is a random variable too. Using equation 2.2 we get:

$$\lim_{n\to\infty} \frac{h(X_1) + h(X_2) + ... + h(X_n)}{n} = E[h(X_1)] \quad \textbf{(Eq. 2.3)}$$

with probability one if the expectation $E[X_1]$ exists. Supposing that the density function of the random variable X, $f(x)$, is known and $h(X)$ is a function of X then the expected value of $h(X)$ is given by:

$$E[h(X_1)] = \int_{-\infty}^{\infty} h(x) f(x) dx = I \qquad \textbf{(Eq. 2.4)}$$

provided that the integral I exists [4]. From equation 2.3 and 2.4 we see that the integral I can be approximated by the sample mean

$$I \approx \overline{Z}_n = \frac{h(X_1) + h(X_2) + ... + h(X_n)}{n}$$

for sufficiently large n. The 'sufficiently' term is explained with the help of the central limit theorem. By that theorem, Z_n is approximately normally distributed with mean $I = E[X_1]$ and variance $\frac{\sigma^2}{n}$ with $\sigma^2 = V[h(X_1)]$. By subtracting the mean and dividing with the standard deviation we create a standard normal variable $\frac{\overline{Z}_n - I}{\frac{\sigma}{\sqrt{n}}}$. Therefore the probability of the value I, being within the confidence interval of ε with probability α, can be rewritten as

$$P\left\{ -\frac{\varepsilon}{\frac{\sigma}{\sqrt{n}}} \le \frac{\overline{Z}_n - I}{\frac{\sigma}{\sqrt{n}}} \le \frac{\varepsilon}{\frac{\sigma}{\sqrt{n}}} \right\} = \alpha$$

The next steps are to denote the 100(1-α)-percentile of the standard normal distribution as χ_α and use the sample variance since the population variance is unknown. Finally the 'sufficiently' large n is given by

$$n = \left(\frac{\sigma \chi_{\frac{\alpha}{2}}}{\varepsilon} \right)^2$$

Variance reduction

The variance reduction techniques used by H&W are antithetic variable methods and control variate methods and we follow this order for the presentation of the two methods [3]. For the antithetic variable methods, the estimate of a variable X is the mean value of two variables X_1,

X_2 generated by Monte Carlo simulation. The estimate is equal to $(X_1 + X_2)/2$ and the variance is given by

$$V[\bar{X}] = \frac{1}{4}(V[X_1] + V[X_2] + 2C[X_1 X_2]$$

Where $C[X_1 X_2]$ is the covariance of X_1 with X_2. The estimate has smaller variance if the variables are negatively correlated and this is how the variance of the estimate is reduced. For the control variate methods two estimates X, whose mean (μ_X) is unknown, and Y whose mean (μ_Y) is known are obtained by the same simulation experiment. Let:

$$Z = X + a(Y - \mu_Y)$$

where a is a constant. Consequently:

$$E[Z] = E[X]$$
$$V[Z] = V[X] + \alpha^2 V[Y] + 2\alpha C[X, Y]$$

To minimize $V[Z]$, a must be equal to $-C[X, Y]/V[Y]$ and the variance of Z becomes

$$V[Z] = V[X] - \frac{(C[X, Y]))^2}{V[Y]}$$

The random variable Y is called a control variate for the estimation of $E[X]$.

Stochastic Processes

There are many assumptions that can be made regarding the nature of a random variable. A *Markov process* is a certain stochastic process that a variable may be assumed to follow. It states that the history of the variable is irrelevant and only the present value is used to predict the future. A *Wiener process* on the other hand, also known as a *Brownian motion*, is a particular case of a Markov process with a mean of zero and variance of 1.0 per year [6]. A variable z follows a Wiener process if:

1. The change Δz during a small time period Δt is $\Delta z = \varepsilon\sqrt{\Delta t}$

   where ε is normally distributed $\phi(0,1)$.

2. Δz for two different short intervals of time, Δt, are independent.

It follows that Δz is normal distributed with mean zero, standard deviation $\sqrt{\Delta t}$ and variance of Δt. Furthermore, for a larger time period from 0 to maturity T, $z(T)$ - $z(0)$ is normally distributed with mean zero, standard deviation $\sqrt{T}$ and variance T.

A *generalized Wiener process* for a variable s adds an expected drift rate μ and variability σ. We have:

$$ds = dt + dz$$

It can be shown that the variable s is normally distributed in any time interval T. The mean change of s is μT, standard deviation $\sigma\sqrt{T}$ and variance σ^2 T.

Stochastic Volatility Models

Stochastic volatility models treat the volatility of the underlying asset as a random process rather than a constant. Volatility measures the unexpected changes in the value of a financial asset in a certain time period. Most often it is calculated as the standard deviation, dispersion away from the mean. Since the magnitude of the fluctuations is unknown, volatility is used as a measure of the risk of a certain financial assets.

One problem arising with the assumptions of a model such as the Black-Scholes is volatility smile (or volatility skew in some markets). If we consider options on an underlying equity with different strike prices, then the volatilities implied by their market prices should be the same. They measure the risk for the same underlying asset. In many markets the implied volatilities often represent a “smile” or “skew” instead of a straight line. The “smile” is thus reflecting higher implied volatilities for deep in- or out of the money options and lower implied volatilities for at-the-money options (Figure 2.1).

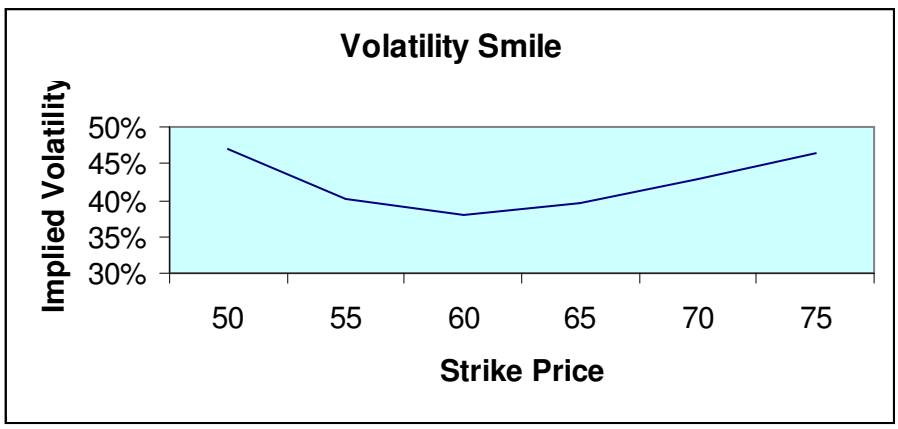

Figure 2.1 Volatility Smile

Similar patters are found by altering time-length to maturity when the market prices are used to find the implied volatilities. These patterns are very difficult to explain in a Black-Scholes world.

Both constant- and stochastic volatility models assume the stock price follows a stochastic process. The most widely used equation [8] for non-dividend paying stock price behaviour is:

$$dS = \mu S dt + \sigma S dz$$

Where S is the stock price at time t, the variable μ the drift or the expected rate of return and σ is the volatility of the stock price. The process for the stock price is known as a geometric Brownian motion. By studying the dynamics of historical prices we assume that the volatility follows a stochastic process [7]. In the case of stochastic volatility, the variance V (= σ^2) is replaced by a stochastic process

$$dV = \theta V dt + \xi V dW$$

Where the two processes, dz and dW are correlated with correlation ρ. There are several different models to describe the evolution of the volatility, such as the Heston model and the Garch model. Different stochastic volatility models take on different assumptions, parameters and simulations to better predict the volatility evolution.

In this thesis we have chosen to focus on John Hull and Alan White's article "The Pricing of Options on Assets with Stochastic Volatilities" from 1987. Their article is one of the first in solving option pricing with stochastic volatility.

The Hull & White Article

The article "The Pricing of Options on Assets with Stochastic Volatilities" produces solutions to the problem of pricing European call options on an underlying asset with stochastic volatility. The reason that this problem has not previously been solved is that there are no assets which are clearly perfectly correlated with the variance. The article produces one solution in series form and two numerical solutions with and without correlation between stock price and volatility.

Hull and White start by considering a derivative asset *f*. The price depends upon some stock price, *S*, and its instantaneous variance, $V = \sigma^2$, which obey the following stochastic processes:

$$dS = \phi S dt + \sigma S dw$$

$$dV = \mu V dt + \xi V dz$$

The drift term ϕ for the stock price may depend on *S*, σ and *t*, while the drift μ and the diffusion coefficient ξ for the variance may depend only on σ and *t*. The two processes, *dw* and *dz* are correlated with correlation ρ. H&W make the following assumptions:

1. *S* and σ^2 are the only two variables that affect the price of derivative *f*. Therefore the risk-free rate, *r*, must be constant or deterministic.
2. The volatility *V* is uncorrelated with the stock price *S*.
3. The volatility *V* is uncorrelated with aggregate consumption, or in other words that the volatility has zero systematic risk.

The expected return of a stock for example is not independent of risk preferences. Risk averse investors would ask for a higher expected return for increasing risk levels and risk seeking investors would ask for a lower expected return for increasing risk levels. The idea of risk-neutral valuation is the most important tool for the analysis of derivatives [6]. It guarantees that the variables involved in the valuation are not influenced by investor risk preference. Thus, in a world where all investors are risk-neutral, the expected return on a stock is the risk-free rate. With the three assumptions above the option value may be found using risk-neutral valuation and must be the present value of *f* at maturity. The price of a European call option is given by:

$$f\left(S_t, \sigma_t^2, t\right) = e^{-r(T-t)} \int f\left(S_T, \sigma_T^2, T\right) p\left(S_T \middle| S_t, \sigma_t^2\right) dS_T \qquad \textbf{(Eq. 2.5)}$$

Where σ_t is the instantaneous standard deviation at time *t* and $p\left(S_T \middle| S_t, \sigma_t^2\right)$ is the conditional distribution of S_T given the security price and variance at time *t*. This conditional distribution of S_T depends both on the process driving *S* and the process driving σ^2. To make it clear that in a risk-neutral world, the expected rate of return on S is the risk-free rate, the condition $E\left[S_T \middle| S_t\right] = S_t e^{r(T-t)}$ is given. The option price $f\left(S_T, \sigma_T^2, T\right)$ is $\max[0, S-K]$. Equation 2.5 is then greatly simplified and rewritten as:

$$f\left(S_t,\sigma_t^2,t\right)=\int\left[e^{-r(T-t)}\int f(S_T)g(S_T\left|\overline{V}\right.)dS_T\right]h(\overline{V}\left|\sigma_t^2\right.)d\overline{V} \qquad \textbf{(Eq. 2.6)}$$

The next step in the article shows with a lemma, that under the assumptions, the inner term can be rewritten as the Black-Scholes price for a call option on an underlying security with mean variance $\overline{V}$. See article for proof. The mean variance over the life of the derivative security is defined by a stochastic integral.

$$\overline{V}=\frac{1}{T}\int_0^T\sigma^2(t)dt$$

The option price may now be written as:

$$f\left(S_t,\sigma_t^2\right)=\int C(\overline{V})h(\overline{V}\left|\sigma_t^2\right.)d\overline{V} \qquad \textbf{(Eq. 2.7)}$$

Where $C(\overline{V})$ is the Black-Scholes price and *h* is the conditional density function of $\overline{V}$ given the instantaneous variance σ^2. Equation 2.7 states that the option price is the Black-Scholes price integrated over the distribution of the mean volatility. It is always true in a risk-neutral world when the stock price and volatility are instantaneously uncorrelated. H&W argue it may not be possible to find an analytic form for the distribution of $\overline{V}$ with the set of assumptions used for the volatility. The solution to resolve this issue is created in series form.

Series Solution

To find a solution to the distribution of $\overline{V}$, H&W calculate all the moments of $\overline{V}$ while keeping ξ and μ constant. The Black-Scholes option price with volatility $\overline{V}$ is then expanded in a Taylor series about its expected value. They show by using the moments for the distribution of $\overline{V}$, the Taylor series becomes:

$$f(S,\sigma^2)=C(\sigma^2)+\frac{1}{2}\frac{S\sqrt{T-t}N'(d_1)(d_1d_2-1)}{4\sigma^3}*\left[\frac{2\sigma^4(e^k-k-1)}{k^2}-\sigma^4\right]$$

$$+\frac{1}{6}\frac{S\sqrt{T-t}N'(d_1)\left[(d_1d_2-3)(d_1d_2-1)-(d_1^2+d_2^2)\right]}{8\sigma^5}$$

$$* \sigma^6 \left[\frac{e^{3k} - (9+18k)e^k + (8+24k+18k^2+6k^3)}{3k^3} \right] + \ldots$$

$$where\ k = \xi_3^2 (T-t)$$

(**Eq. 2.8** – Series Solution)

Stochastic Volatility

Hull and White continue with a model which uses Monte Carlo simulation to calculate the option price. We shall refer to this model as the stochastic volatility model (*SV*). Some of the assumptions necessary for the Series Solution are now relaxed. Stock price and volatility remain uncorrelated ($\rho = 0$), but ξ and μ may now depend on σ and t. This dependence allows the volatility to follow a mean-reverting process. A simple such process, where α, ξ and σ^* are constant, is:

$$\mu = \alpha(\sigma^* - \sigma)$$

The call price being the Black-Scholes price integrated over the distribution of $\overline{V}$ still holds in equation 2.7. Solving the following stochastic differential equation for $V(t)$:

$$\begin{cases} dV(t) = \mu V(t)dt + \xi V(t)dW \\ V(0) = v_0 \end{cases}$$

one can derive the formula used in the article for volatility generation. To solve this process for the volatility Itõ's lemma is used. The lemma is an important formula in financial analysis and states how to differentiate functions of certain stochastic processes. An Itô process for a variable x is a generalized Wiener process in which the parameters a (drift) and b (variance) are functions of the variable x and time t [6].

$$dx = a(x,t)dt + b(x,t)dz$$

The lemma shows that a function G of x and t follows the process:

$$dG = \left(\frac{\partial G}{\partial x} a + \frac{\partial G}{\partial t} + \frac{1}{2} \frac{\partial^2 G}{\partial x^2} b^2 \right) dt + \frac{\partial G}{\partial x} b dz$$

Where *dz* is the same Wiener process as in the first equation. Thus, by using Itõ's lemma and letting $X(t) = \ln\{V(t)\}$ we can solve for *X*:

$$dX = \frac{\partial X}{\partial t}dt + \frac{\partial X}{\partial V}dV + \frac{1}{2}\frac{\partial^2 X}{\partial V^2}(dV)^2$$

$$= \frac{1}{V}(\mu V dt + \xi V dW) - \frac{1}{2}\frac{1}{V^2}\xi^2 V^2 dt$$

$$= (\mu - \frac{1}{2}\xi^2)dt + \xi dW$$

$$\int_t^T dX dt = \int_t^T (\mu - \frac{1}{2}\xi^2)dt + \int_t^T \xi dW$$

$$X(T) - X(t) = (\mu - \frac{1}{2}\xi^2)(T-t) + \xi(W_T - W_t)$$

$$e^{\ln V(T)} = e^{\ln V(t) + (\mu - \frac{1}{2}\xi^2)(T-t) + \xi(W_T - W_t)}$$

$$V(T) = V(t) \times e^{(\mu - \frac{1}{2}\xi^2)(T-t) + \xi(W_T - W_t)}$$

The wiener process $W_T - W_t$ is $N(0, \sqrt{T-t})$ and by changing interval notation from $T - t$ to Δt and letting *z* be a standard normal variable we get:

$$V_i = V_{i-1} \times e^{(\mu - \frac{1}{2}\xi^2)\Delta t + \xi\sqrt{\Delta t}z}$$

Replacing *z* by v_i we arrive at the formula for the volatility:

$$V_i = V_{i-1} e^{\left[(\mu - \frac{\xi_1^2}{2})\Delta t + v_i \xi_1 \sqrt{\Delta t}\right]}$$

(**Eq. 2.9** – SV Model)

H&W state that an efficient way of performing Monte Carlo simulation is thus to divide the time interval *T* - *t* into *n* equal subintervals and use the independent standard normal variates v_i $(1 \le v_i \le n)$ to generate the variance at each time step. The initial volatility V_0 may be a market implied volatility for the stock in question or any other appropriate value of choice.

The arithmetic mean of the generated volatility values is used in the Black-Scholes formula to find a first option price p_1. A second option price p_2 is calculated with the same procedure but changing v_i to its antithetic standard normal variate - v_i. The mean value of p_1 and p_2 over a large amount of simulations gives the estimate of the option price.

Stochastic Stock Price and Volatility

The third solution in the article involves both a stochastic stock price as well as stochastic volatility and will be referred to as the *SSV* model. In this simulation the processes for *S* and *V* are correlated with correlation ρ. H&W also allow μ and ξ to depend on *S* as well as σ and *t*. The volatility continues to be uncorrelated with aggregate consumption so that risk-neutral valuation may be used.

To better understand how H&W may have reached the formulas for stock price- and volatility evolution, we solve the following set of differential equations:

$$\begin{cases} dS(t) = rS(t)dt + \sqrt{V_{t-1}}S(t)dW_1 \\ S(0) = s_0 \end{cases}$$

$$\begin{cases} dV(t) = \mu V(t)dt + \xi V(t)dW_2 \\ V(0) = v_0 \end{cases}$$

$$\text{Let } Y(t) = \ln\{S(t)\} \text{ and } X(t) = \ln\{V(t)\}$$

Solving for *S* with Itõ's lemma, we get:

$$dY = \frac{\partial Y}{\partial t}dt + \frac{\partial Y}{\partial S}dS + \frac{1}{2}\frac{\partial^2 Y}{\partial S^2}(dS)^2$$

$$= \frac{1}{S}(rSdt + \sqrt{V_{t-1}}SdW_1) - \frac{1}{2}\frac{1}{S^2}V_{t-1}S^2dt$$

$$= (r - \frac{1}{2}V_{t-1})dt + \sqrt{V_{t-1}}dW_1$$

For the volatility *V*:

$$dX = \frac{\partial X}{\partial t}dt + \frac{\partial X}{\partial V}dV + \frac{1}{2}\frac{\partial^2 X}{\partial V^2}(dV)^2$$

$$= \frac{1}{V}(\mu V dt + \xi V dW_2) - \frac{1}{2}\frac{1}{V^2}\xi^2 V^2 dt$$

$$= (\mu - \frac{1}{2}\xi^2)dt + \xi dW_2$$

We have:

$$dX = (\mu - \frac{1}{2}\xi^2)dt + \xi dW_2$$

$$dY = (r - \frac{1}{2}V_{t-1})dt + \sqrt{V_{t-1}}dW_1$$

Where dW_1 and dW_2 are correlated with $dW_1 dW_2 = \rho dt$ and ξ and $\sqrt{V_{t-1}}$ are the diffusion coefficients for the two Wiener processes. After performing a Cholesky transformation we can better see the role of the correlation [5]:

$$\begin{bmatrix} dX \\ dY \end{bmatrix} = \begin{bmatrix} (\mu - \frac{1}{2}\xi^2) \\ (r - \frac{1}{2}V_{t-1}) \end{bmatrix} dt + \begin{bmatrix} \sqrt{1-\rho^2}\xi & \rho\xi \\ 0 & \sqrt{V_{t-1}} \end{bmatrix} \begin{bmatrix} dZ_1 \\ dZ_2 \end{bmatrix}$$

Where $dZ_1 dZ_2 = 0$

Rewriting the matrix:

$$dX = (\mu - \frac{1}{2}\xi^2)dt + \sqrt{1-\rho^2}\xi dZ_1 + \rho\xi dZ_2$$

$$dY = (r - \frac{1}{2}V_{t-1})dt + \sqrt{V_{t-1}}dZ_2$$

Integration gives:

$$\int_t^T dXdt = \int_t^T (\mu - \frac{1}{2}\xi^2)dt + \int_t^T \sqrt{1-\rho^2}\,\xi dZ_1 + \int_t^T \rho\xi dZ_2$$

$$\int_t^T dYdt = \int_t^T (r - \frac{1}{2}V_{t-1})dt + \int_t^T \sqrt{V_{t-1}}\,dZ_2$$

$$X(T) - X(t) = (\mu - \frac{1}{2}\xi^2)(T-t) + \sqrt{1-\rho^2}\,\xi(Z_1^T - Z_1^t) + \rho\xi(Z_2^T - Z_2^t)$$

$$Y(T) - Y(t) = (r - \frac{1}{2}V_{t-1})(T-t) + \sqrt{V_{t-1}}\,(Z_2^T - Z_2^t)$$

$$e^{\ln X(T)} = e^{\ln X(t) + (\mu - \frac{1}{2}\xi^2)(T-t) + \sqrt{1-\rho^2}\,\xi(Z_1^T - Z_1^t) + \rho\xi(Z_2^T - Z_2^t)}$$

$$e^{\ln Y(T)} = e^{\ln Y(t) + (r - \frac{1}{2}V_{t-1})(T-t) + \sqrt{V_{t-1}}\,(Z_2^T - Z_2^t)}$$

$$V(T) = V(t) \times e^{(\mu - \frac{1}{2}\xi^2)(T-t) + \sqrt{1-\rho^2}\,\xi(Z_1^T - Z_1^t) + \rho\xi(Z_2^T - Z_2^t)}$$

$$S(T) = S(t) \times e^{(r - \frac{V_t}{2})(T-t) + \sqrt{V_{t-1}}\,(Z_2^T - Z_2^t)}$$

Rewriting we arrive at the formulas for stock price and volatility evolution:

$$S_i = S_{i-1} e^{(r - \frac{V_{i-1}}{2})\Delta t + u_i \sqrt{V_{i-1}\Delta t}}$$

(**Eq. 2.10** – SSV Model)

$$V_i = V_{i-1} e^{(\mu - \frac{1}{2}\xi_2^2)\Delta t + \rho u_i \xi_2 \sqrt{\Delta t} + \sqrt{1-\rho^2}\, v_i \xi_2 \sqrt{\Delta t}}$$

(**Eq. 2.11** – SSV Model)

S_0 and V_0 are initial values for the simulation. We divide the time interval T - t into n subintervals. The two independent normal variates, u_i and v_i, are used to generate the stock price S_i and variance V_i at time i in a risk-neutral world using the formulas in equations 2.10

and 2.11. The end price of the stock, S_n, is used to find a first option price estimate, p_1, by discounting:

$$e^{-r(T-t)} \times \max[S_n - K, 0] \quad \text{for a call}$$

$$e^{-r(T-t)} \times \max[K - S_n, 0] \quad \text{for a put}$$

Similarly, we find p_2 by using $-u_i$ instead of its antithetic standard normal variate u_i, p_3 by using u_i but replacing v_i with $-v_i$ and finally p_4 by using $-u_i$ and $-v_i$. Also two sample values of the Black-Scholes price q_1 and q_2 are calculated. These values are found simulating *S* using u_i and $-u_i$ respectively but keeping the volatility constant at v_0. In the article H&W provide two estimates of the pricing bias over a large amount of simulations with the following formulas:

$$\frac{p_1 + p_3 - 2q_1}{2} \text{ and } \frac{p_2 + p_4 - 2q_2}{2}$$

In our calculator the mean value of all six estimates is used for the value of the stock price at maturity and the option price.

Section III

User's guide

Before we start with the user's guide we provide a table with the notation for the parameters appearing on the applet.

Notation:

Mathematical	Explanation	Java Applet
	SV Model	
α_1	Constant parameter used for the estimation of the drift coefficient of the variance V	`Alpha1`
σ_1*	Constant parameter used for the estimation of the drift coefficient of the variance V	`Sigma*1`
ξ_1	Diffusion coefficient for the stochastic process of the variance V	`Ksi1`
n	Number of intervals that the time to maturity is divided into	`Intervals1`
-	Number of simulations for each interval	`Simulations1`
	SSV Model	
α_2	Constant parameter used for the estimation of the drift coefficient of the variance V	`Alpha2`
σ_2*	Constant parameter used for the estimation of the drift coefficient of the variance V	`Sigma*2`
ξ_2	Diffusion coefficient for the stochastic process of the variance V	`Ksi2`
ρ	Correlation coefficient between stock price and its variance V	`Rho`
n	Number of intervals that the time to maturity is divided into	`Intervals2`
-	Number of simulations for each interval	`Simulations2`
	PS Model	
ξ_3	Diffusion coefficient for the stochastic process of the variance V	`Ksi3`

Description of the applet

There are two panels in the applet; the input panel and the output panel. The input panel is shown in figure 3.1.

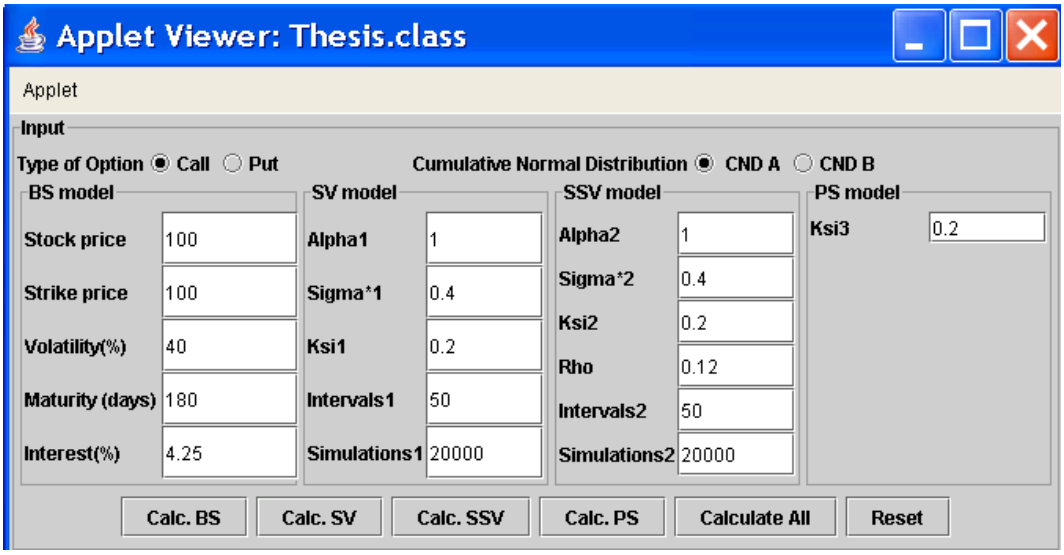

Figure 3.1 The input panel.

The first two choices are the type of the option in question and the method for the calculation of the cumulative normal distribution. The user may click on either one of the two radio buttons to the left of the top row of the input panel in order to choose between a 'Call' or a 'Put' option. The second choice depends on the time constraint of the user. The first method, 'CND A' (which was created by us using standard formulas) is slightly more accurate and more time consuming than 'CND B' (which was granted to us by our supervisor).

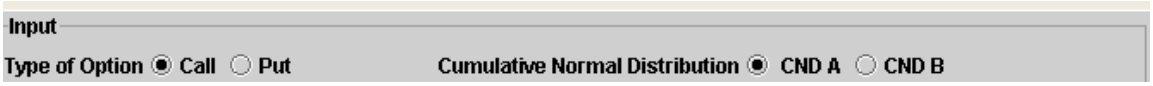

Figure 3.2 'Call' option price to be calculated with 'CND A' method.

The main part of the input panel is divided into four parts each of them consisting of the input parameters required for each of the models. The parts from left to right are:

1. The Black-Scholes model labelled 'BS model'
2. The Stochastic Volatility model labelled 'SV model'
3. The Stochastic Price Volatility model labelled 'SSV model'
4. The Series Solution model labelled 'PS model' (Power Series)

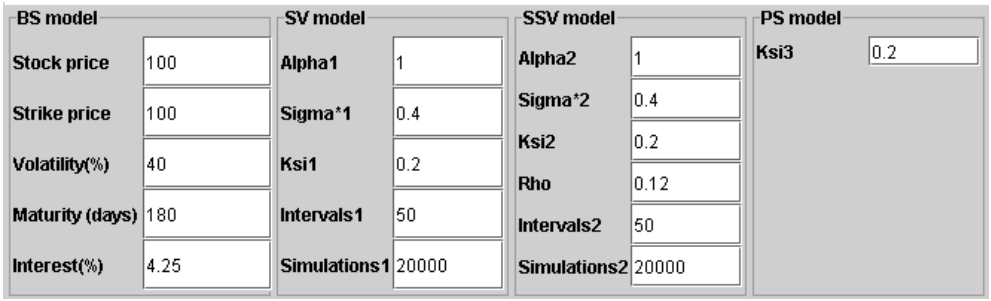

Figure 3.3 The main part of the input panel

After the type of the option and the cumulative normal distribution method has been chosen the user needs to insert the specific parameter values for the models:

1. Stock price: the price of the underlying stock.
   Acceptable value: $0 <$ *Stock price*
2. Strike price: the price for the underlying stock at maturity over (under) which a call (put) has positive value.
   Acceptable value: $0 <$ *Strike price*
3. Volatility (%): the standard deviation of the underlying stock , expressed in percentage points.
   Acceptable value: $0 <$ *Volatility*
4. Maturity (days): the time remaining to the expiration of the option, expressed in days.
   Acceptable value: $0 <$ *Maturity*
5. Interest(%): the risk free interest rate, expressed in percentage points.
   Acceptable value: $0 <$*Interest*
6. Alpha1: constant parameter for the 'SV model', used for the estimation of the drift coefficient of the variance when the variance follows a mean reverting process.
   Acceptable value: $0 <$ *Alpha1*
7. Sigma*1: constant parameter for the 'SV model', used for the estimation of the drift coefficient of the variance when the variance follows a mean reverting process.
   Acceptable value: $0 <$ *Sigma*1*
8. Ksi1: constant parameter for the 'SV model', used for the estimation of the diffusion coefficient of the variance.
   Acceptable value: $0 <$ *Ksi1*
9. Intervals1: number of time intervals used for the estimation of the variance in 'SV model'.
   Acceptable value: positive integer
10. Simulations1: number of simulations for each calculation involving random number in 'SV model'.

Acceptable value: positive integer

11. Alpha2: constant parameter for the 'SSV model', used for the estimation of the drift coefficient of the variance when the variance follows a mean reverting process.
    Acceptable value: $0 < Alpha2$
12. Sigma*2: constant parameter for the 'SSV model', used for the estimation of the drift coefficient of the variance when the variance follows a mean reverting process.
    Acceptable value: $0 < Sigma*2$
13. Ksi2: constant parameter for the 'SSV model', used for the estimation of the drift coefficient of the variance.
    Acceptable value: $0 < Ksi2$
14. Rho: a correlation coefficient between the stock price and volatility used in the 'SSV model'.
    Acceptable value: $-1 \leq Rho \leq 1$
15. Intervals2: number of time intervals used for the estimation of the variance in 'SSV model'
    Acceptable value: positive integer
16. Simulations2: number of simulations for each calculation involving random number in 'SSV model'.
    Acceptable value: positive integer
17. Ksi3: constant parameter for the 'PS model'.
    Acceptable value: $0 < Ksi3$

The lower part of the input panel consists of six buttons that perform an action when pressed. These actions from left to right are:

1. Calculate the price using the Black-Scholes model labelled 'Calc. BS '.
2. Calculate the price using the Stochastic Volatility model labelled 'Calc. SV '.
3. Calculate the price using the Stochastic Price Volatility model labelled 'Calc. SSV '.
4. Calculate the price using the Power Series model labelled 'Calc. PS '.
5. Calculate the price using all four models labelled 'Calculate All '.
6. Set the input parameters to their default values labelled 'Reset'.

Figure 3.4 The lower part of the input panel where the orders for calculation are given

Once the user has pressed one of the above buhttons the order will be executed and the result will be shown in the output panel, which is displayed in the lower part of the applet.

Figure 3.5 The output panel

In case no choice is made about the type of the option, or no values are inserted in the text fields, the applet will execute with the default values. To set all the fields to the default values the user has to click on the button with labelled 'Reset'. If the 'Calculate All' has been pressed the program will be executed and all the results will appear in the output panel. The output panel displays the price of the option using four different methods.

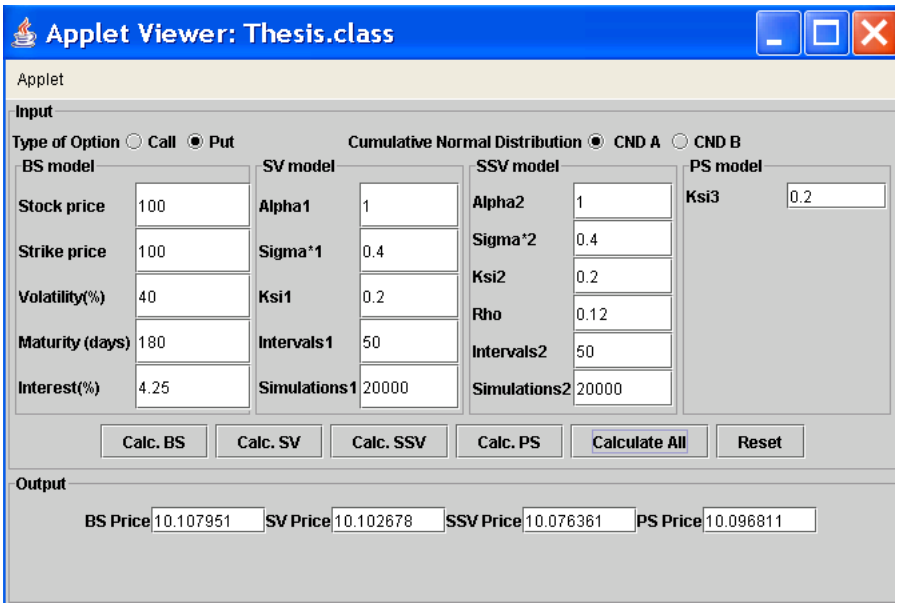

Figure 3.6 Input and output panels after the execution of the program

Inserting non-acceptable parameter values

In case an unacceptable value is inserted as a parameter, the user is informed about the wrong entry by a pop-up window that appears showing where the problem has occurred. The user needs to close this window before the execution of the program can continue. The window closes after the user has clicked 'OK' or exit ('x'). This process resets the value of the text

field to the last acceptable value inserted. In figure 3.7 the volatility has been set to a negative value (-40). The information window has popped up and the program execution will not continue until the user has closed the window. This process ensures the user is aware of the mistake.

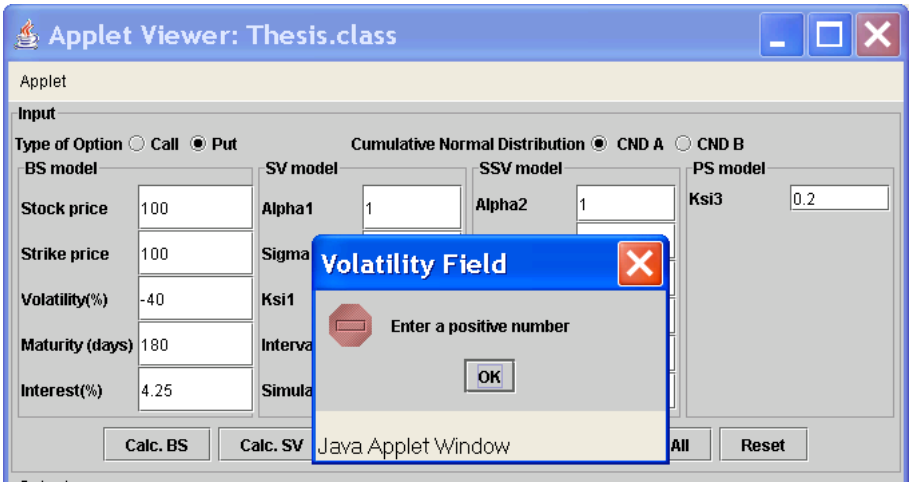

Figure 3.7 Window informing about a wrong entry for the 'Volatility'

In figure 3.8 the number of 'Intervals1' has been set to a non-integer number (50.5). The information window informs the user of the location and type of mistake.

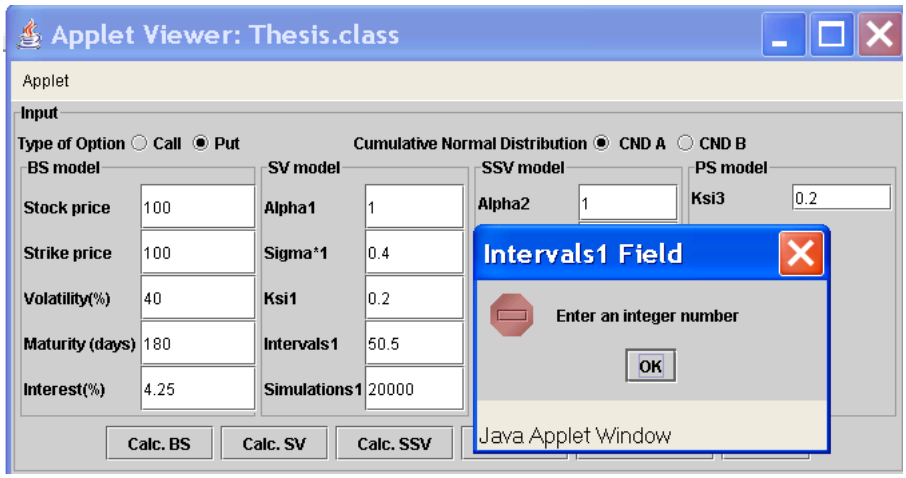

Figure 3.8 Window informing about a wrong entry for the 'Intervals1'

Section IV

Data Generation

Parameter Analysis

An analysis of the effect of the parameters involved in the option calculator is done in this section. When applicable, both the effect on both the volatility as well as the call option price is presented. In this section volatility refers to the standard deviation of the stock price. Unless otherwise stated in the figure legend or description, the following set of standard parameter values is used in each section:

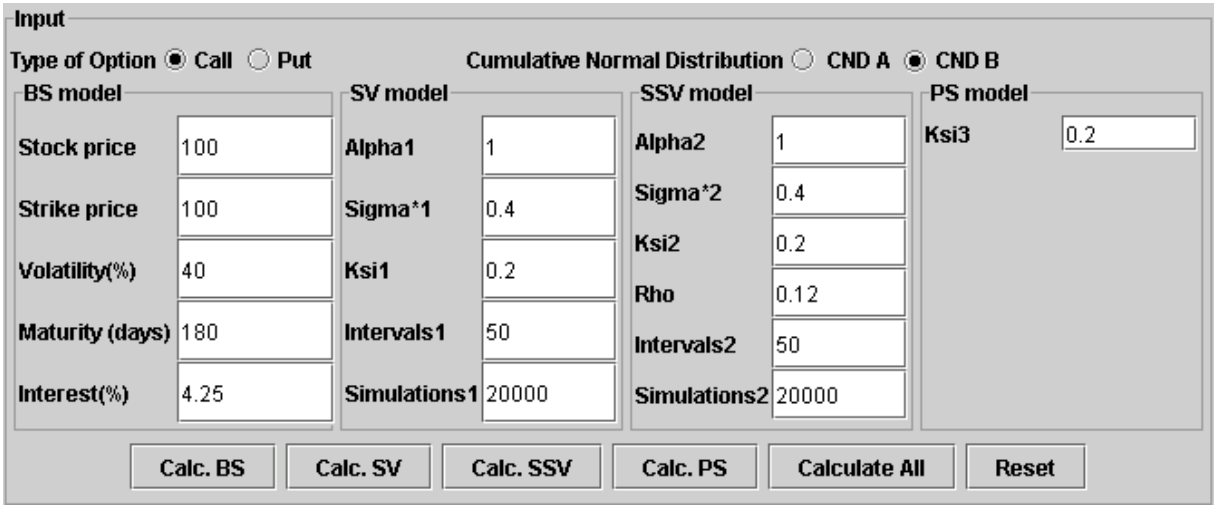

Figure 4.1 Standard parameter values

Black-Scholes Model

The Black-Scholes option price calculator involves only the five usual parameters: stock price, strike price, time to maturity, volatility and interest rate. Changing any of these values will affect the option price but a detailed analysis of Black-Scholes will not be given in this paper.

Series Solution

The Series Solution (#4 - '*PS model*') calculates the option price using a closed form solution. The only model-specific parameter in altering the option price is *ksi3*.

Ksi3

Increasing the value for *ksi3* reduces the Series Solution call price rapidly as can be seen in figure 4.2.

Figure 4.2 Effect of *Ksi3* on Series Solution call price

SV model

The stochastic volatility found using the SV model:

$$V_i = V_{i-1} e^{\left[(\mu - \frac{\xi_1^2}{2})\Delta t + v_i \xi_1 \sqrt{\Delta t}\right]}$$

is directly affected by the parameters inserted. Each parameter's influence on the volatility and call price is presented.

*Sigma*1 and Alpha1*

The parameters *sigma*1* and *alpha1* are constant values in the mean-reverting process the volatility is assumed to depend on:

$$\mu = \alpha_1(\sigma_1^* - \sigma)$$

By altering either of the two parameters the value of μ (the drift coefficient of the stochastic differential equation) is changing. For values of σ_1^* greater (smaller) than σ the volatility becomes larger (smaller) (Figure 4.4). If the starting volatility σ and the parameter σ_1^* are set equal then μ is zero. Parameter α_1 only becomes important if there is a difference between the two volatilities and works as a multiplier of this difference. A larger μ normally results in a larger volatility found by the stochastic process and thus a larger option price when used in the Black-Scholes formula (Figure 4.3).

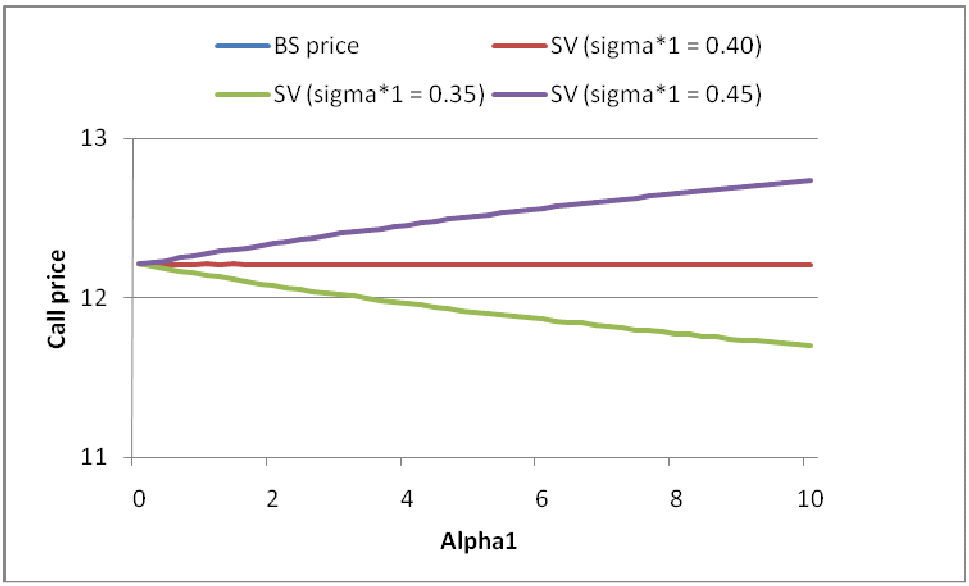

Figure 4.3 Effect of *Alpha1* on call price in SV model. BS price is covered by SV price due to the insignificant difference in call price

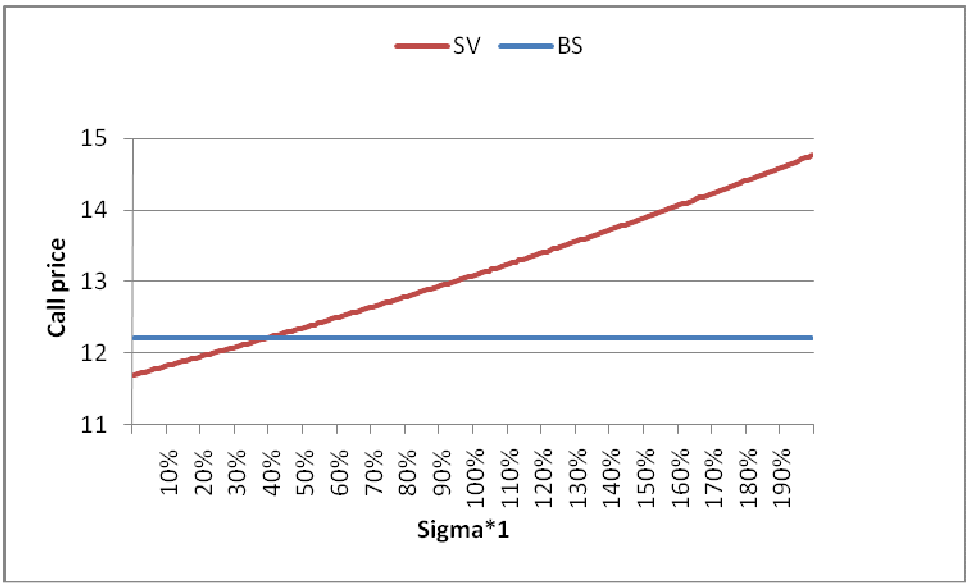

Figure 4.4 Effect of *Sigma*1* on call price in SV model with σ=40%

The parameters σ_1^*- and α_1's effect on the volatility follow a very similar pattern as can be seen in Figure 4.5 and 4.6.

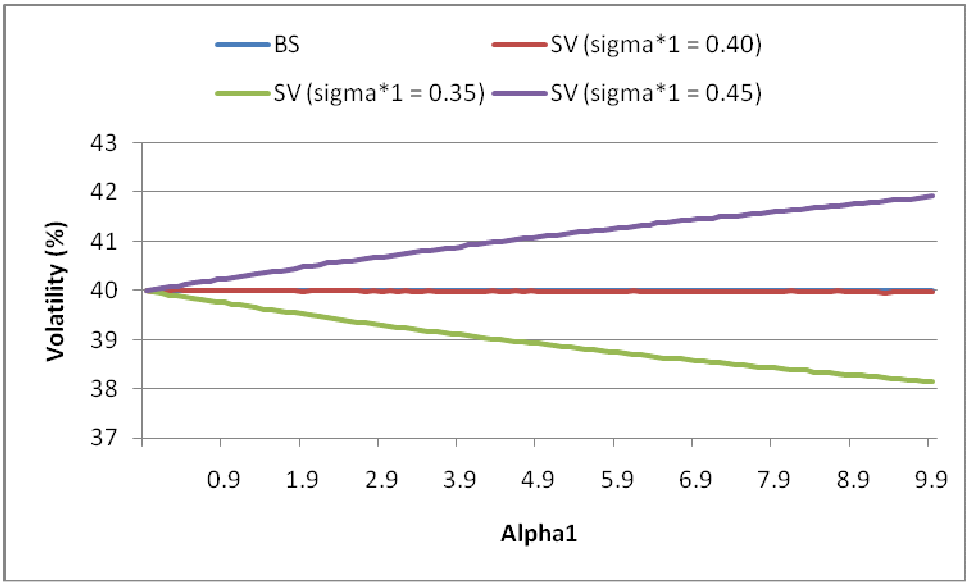

Figure 4.5 Effect of *Alpha1* on volatility in SV model

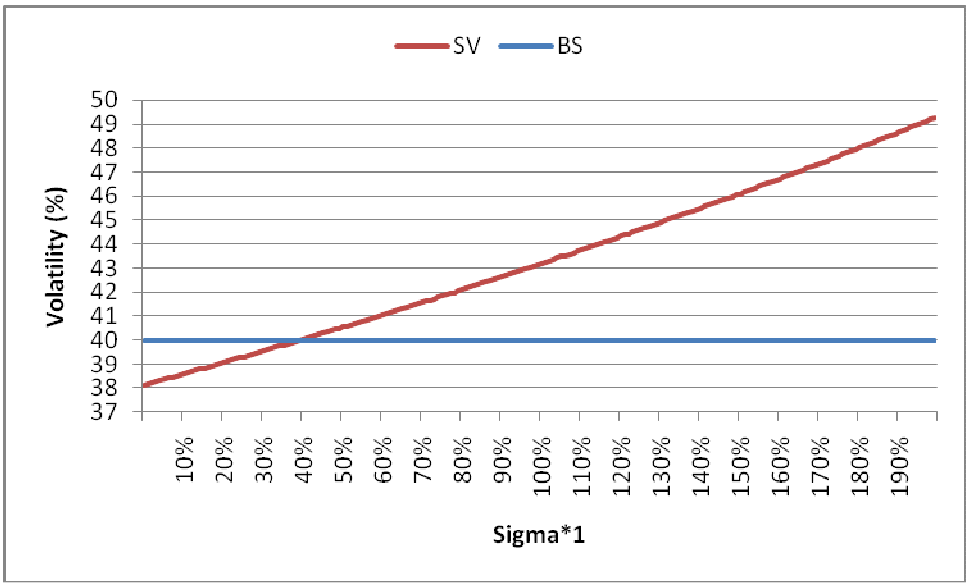

Figure 4.6 Effect of *Sigma*1* on volatility in SV model

Ksi1

The parameter ξ_1 appears twice in the exponential part of the formula for the volatility in the SV model (Eq. 2.9). In Figures 4.7 and 4.8 we can see how the option price and volatility respectively are affected by changing ξ_1.

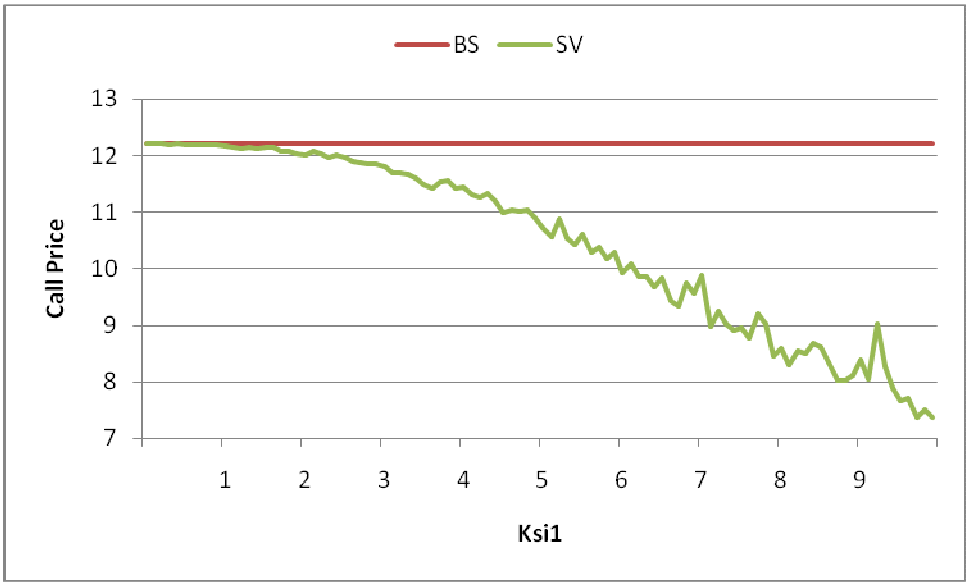

Figure 4.7 Effect of *Ksi1* on call price in SV model

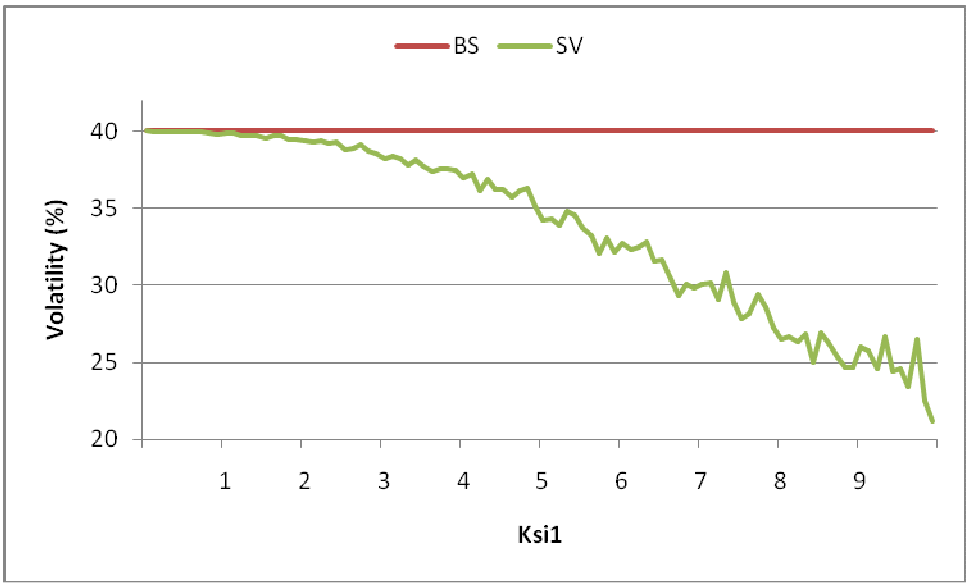

Figure 4.8 Effect of *Ksi1* on volatility in SV model

Intervals1

The parameter *intervals1* refers to number of subintervals that the time to maturity is divided into. The volatility is generated at each interval as many times as the simulation parameter is set to. The arithmetic mean of all the volatilities found until maturity in the SV model is used in the Black-Scholes formula. Even with a small number of intervals the volatility generated has a small difference with that of a larger number of intervals. In Figures 4.9 and 4.10 we see only a minor fluctuation due to changes in the number of intervals in the SV model.

Figure 4.9 Effect of *Intervals1* on the call price in SV model

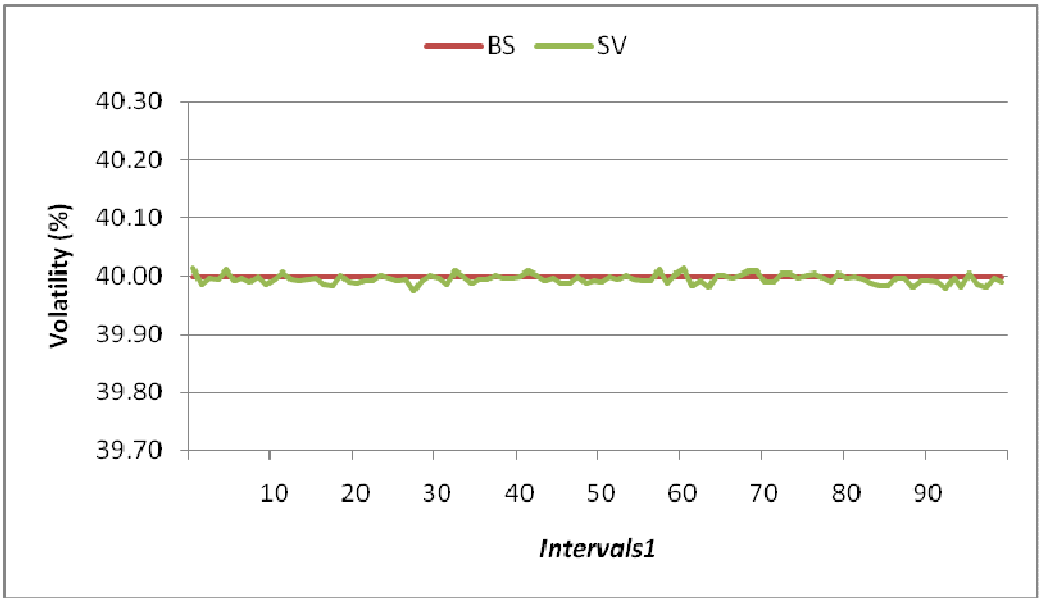

Figure 4.10 Effect of *Intervals1* on the volatility in SV model

Simulations1

The more times we simulate the stochastic volatility the less the fluctuation of the option price found. With less than one hundred simulations the volatility will have a wide range span. Since the SV model uses the arithmetic mean of the volatility as an input into the Black-Scholes formula, a large number of simulations results in an option price close to the BS price. In graph 4.11 we can see the price of a call option using different number of simulations for the volatility. Once we reach approximately one thousand simulations the option price and volatility respectively converge to within a small fluctuation range. A small fluctuation will always be present due to the stochastic nature of the process.

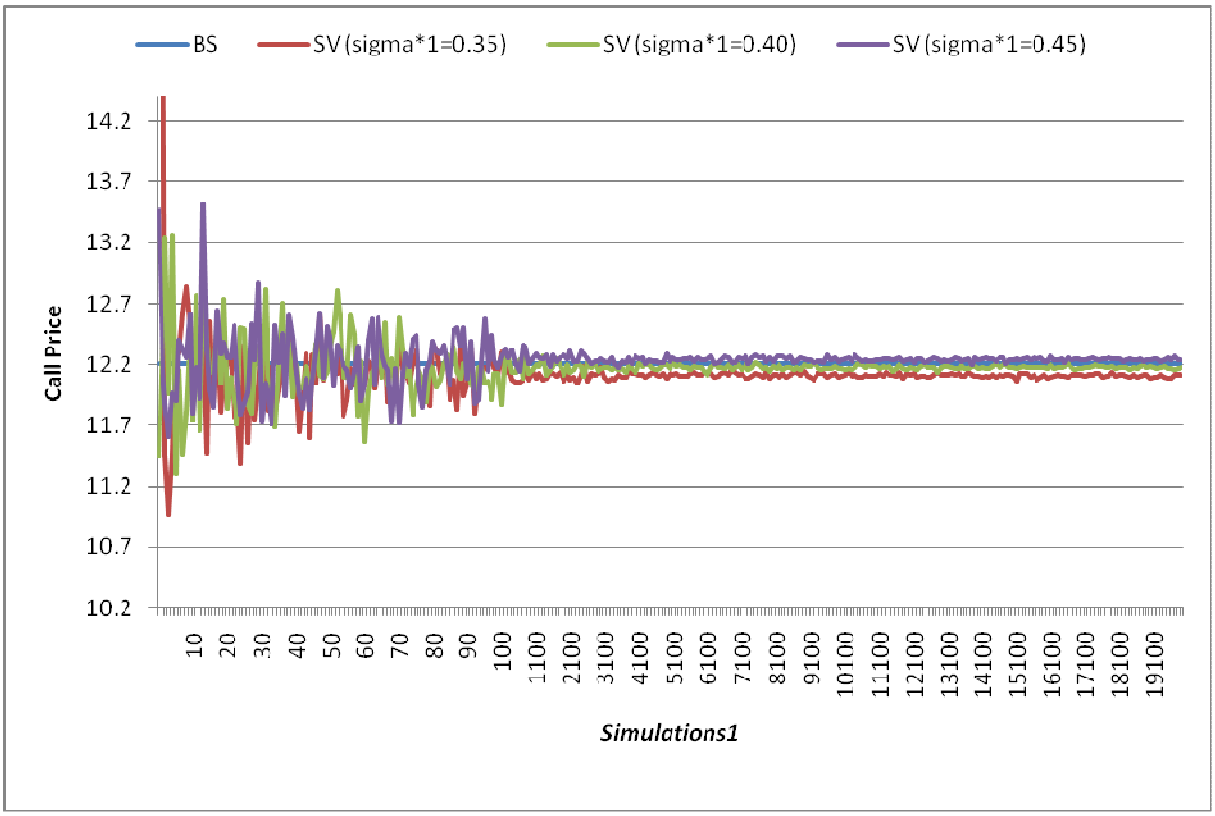

Figure 4.11 Effect of *Simulations1* on call price in SV model

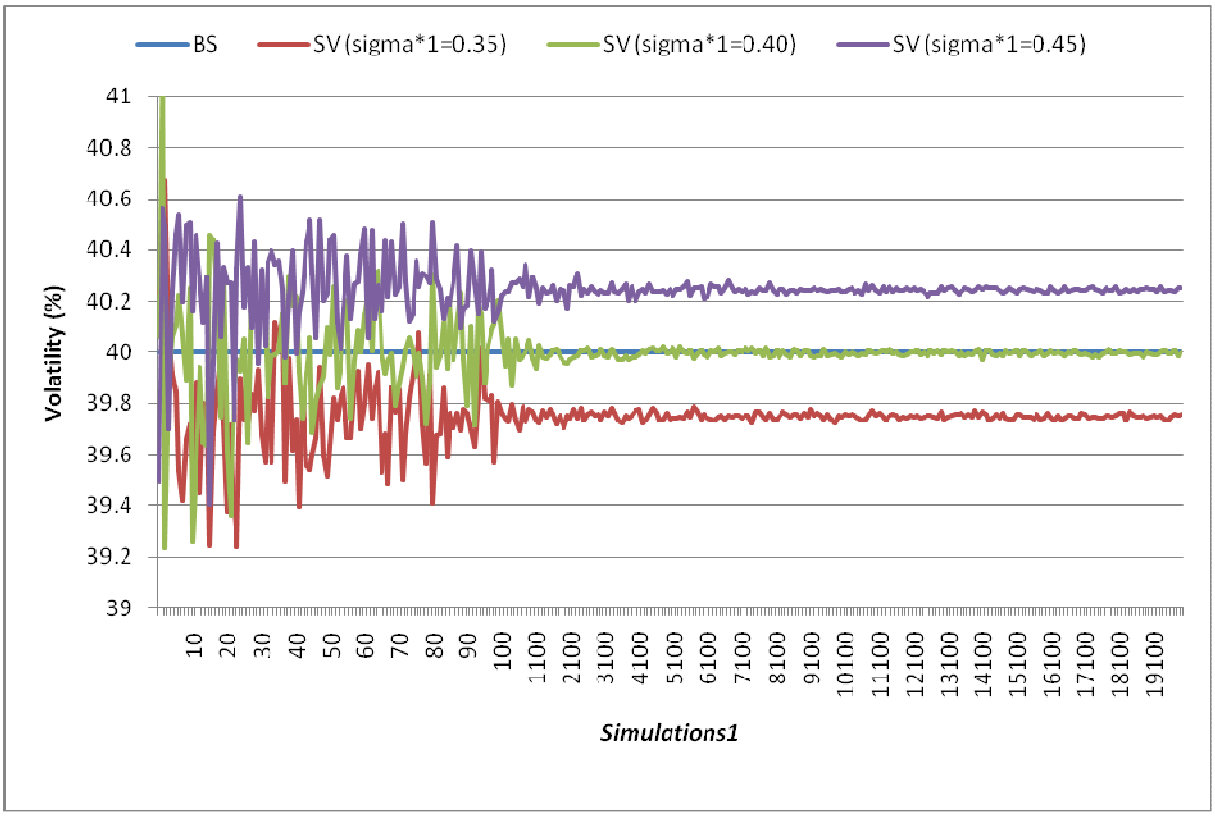

Figure 4.12 Effect of *Simulations1* on the volatility in SV model

SSV model

The SSV model contains both a stochastic volatility and a stochastic stock price. The volatility generated in each time step is used to calculate the stochastic stock price and thus directly affects the option price. The SSV model only makes use of the volatility and stock price generated at maturity (T). The effect of each parameter's influence on the volatility and call price is presented.

*Sigma*2 and Alpha2*

The parameters *sigma*2* and *alpha2* in the SSV model are constant values in the mean-reverting process the volatility is assumed to depend on. By altering either of the two variables the value of μ is changing.

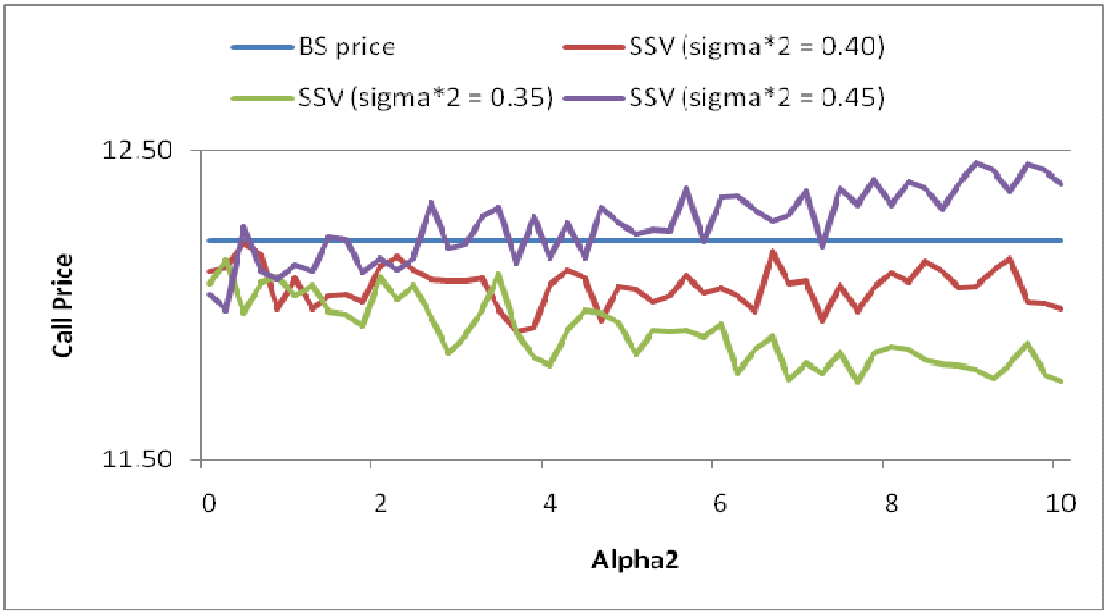

Figure 4.13 Effect of *Alpha2* on the call price in the SSV model

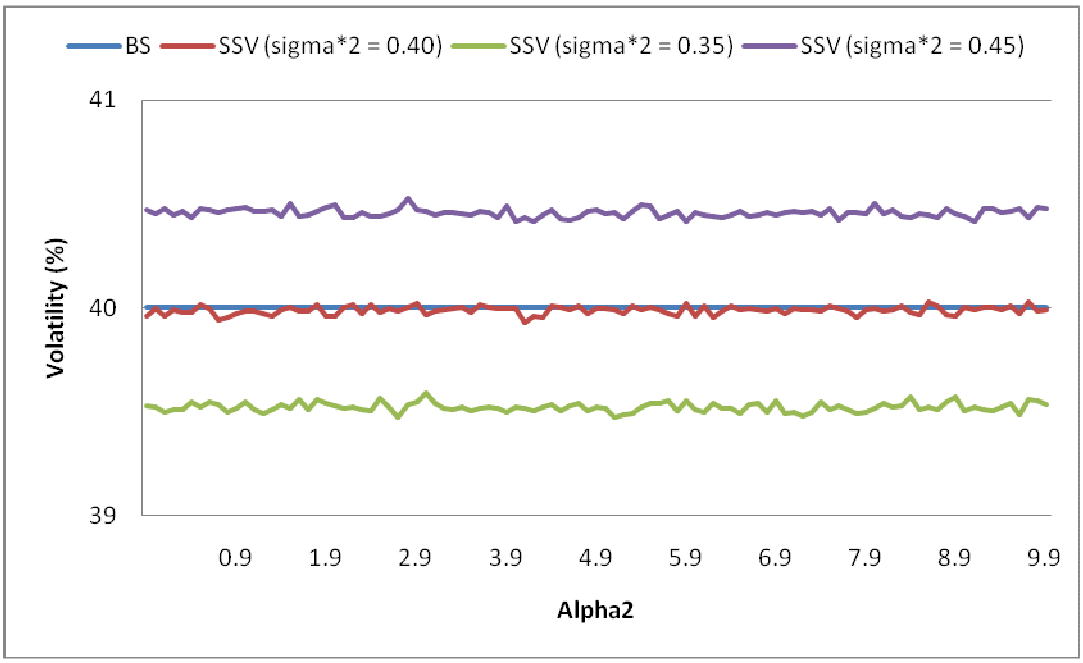

Figure 4.14 Effect of *Alpha2* on the volatility in the SSV model

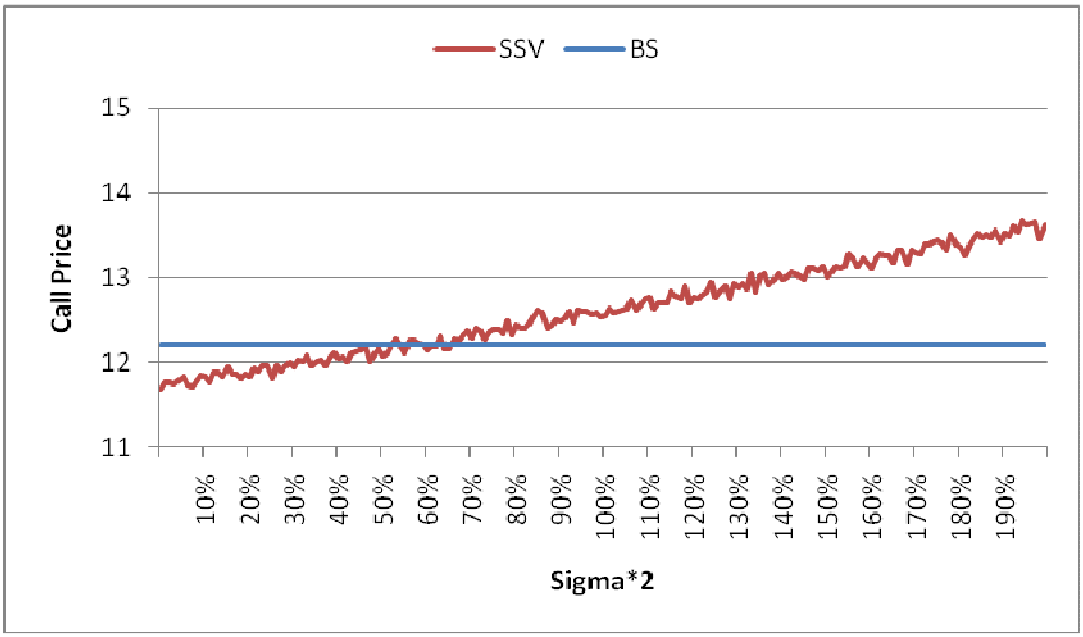

Figure 4.15 Effect of *Sigma*2* on the call price in the SSV model

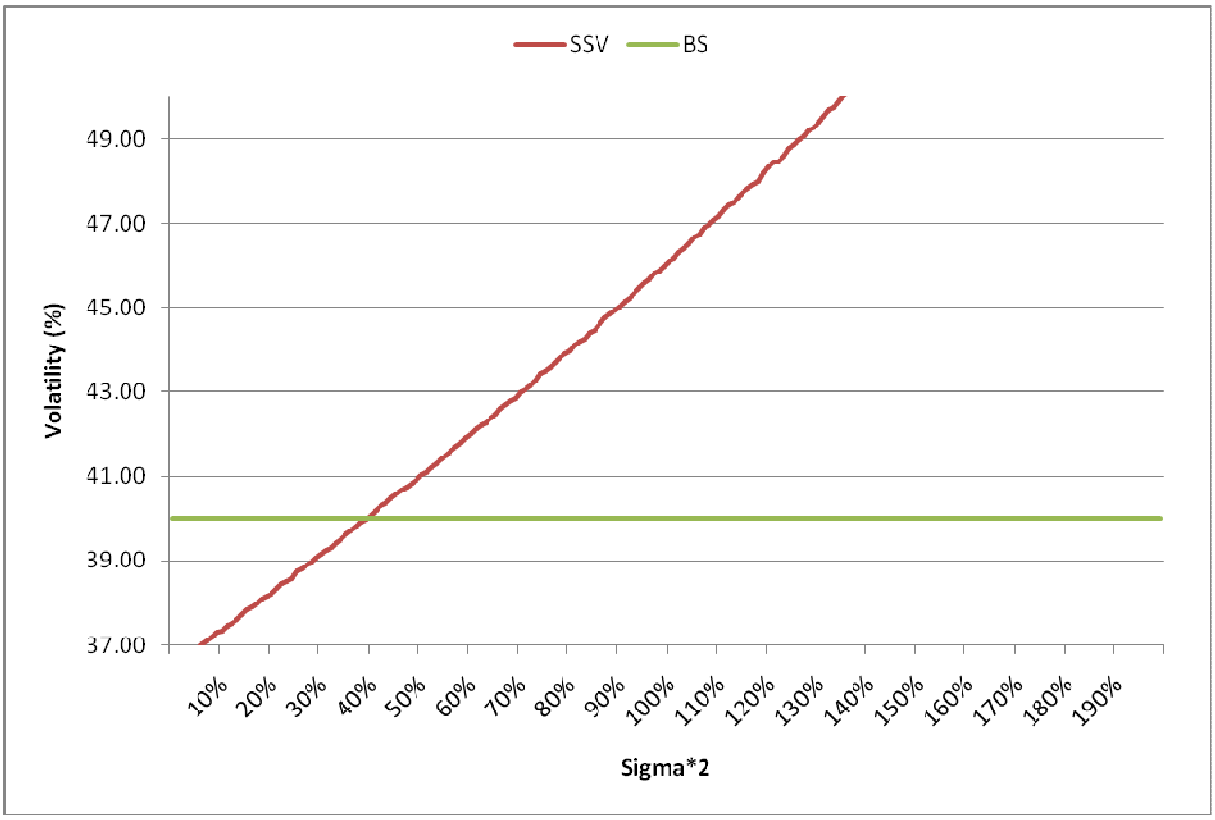

Figure 4.16 Effect of *Sigma*2* on the volatility in the SSV model

Ksi2

The parameter ξ_2 is involved three times in the formula for the volatility in the SSV model (Eq. 2.11). In Figures 4.17 and 4.18 we can see how the option price and volatility respectively are affected by changing ξ_2.

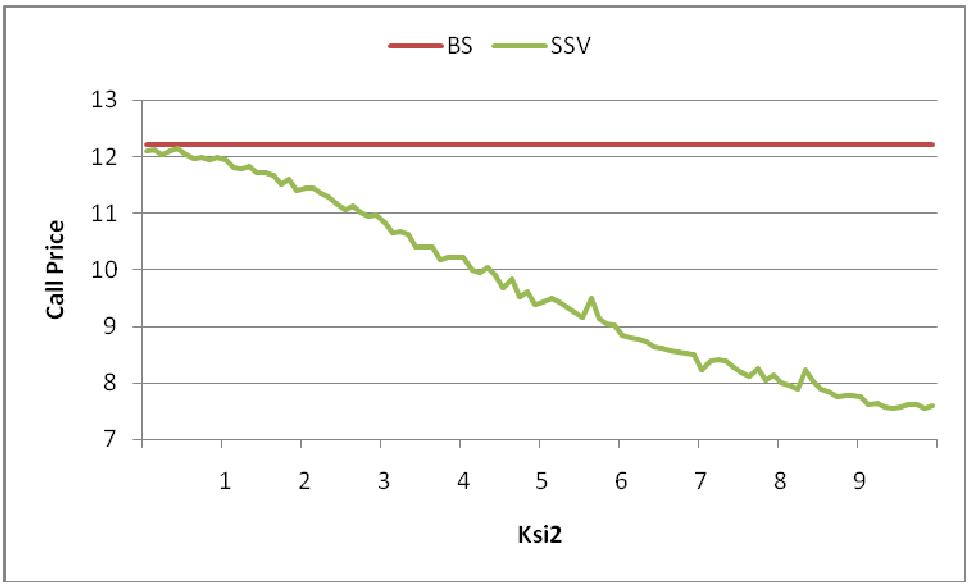

Figure 4.17 Effect of *Ksi2* on call price in SSV model

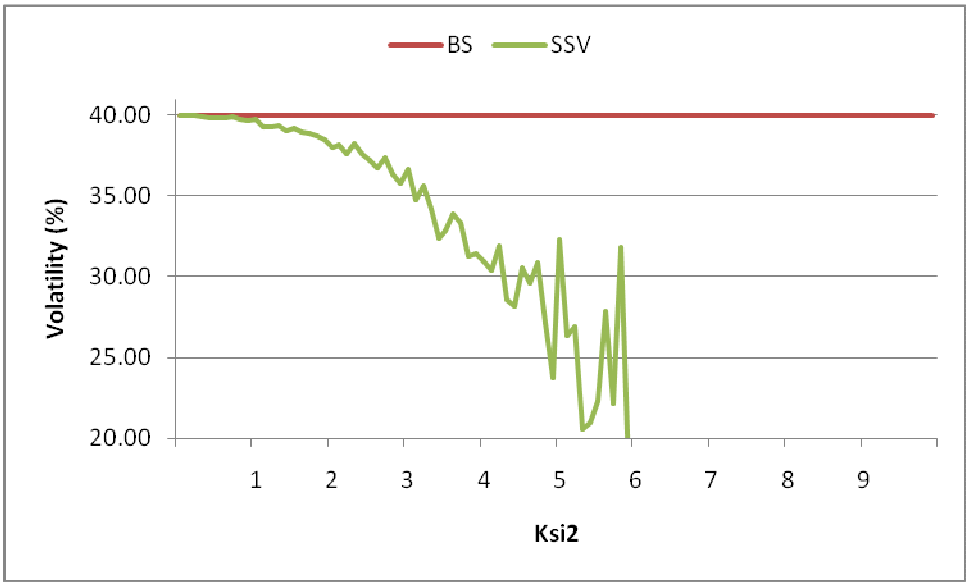

Figure 4.18 Effect of *Ksi2* on volatility in SSV model

Rho

The parameter ρ is only involved in the equation for the stochastic volatility. It represents the correlation between the variance and the stock price. No certain pattern was discovered for call price or volatility when ρ varied.

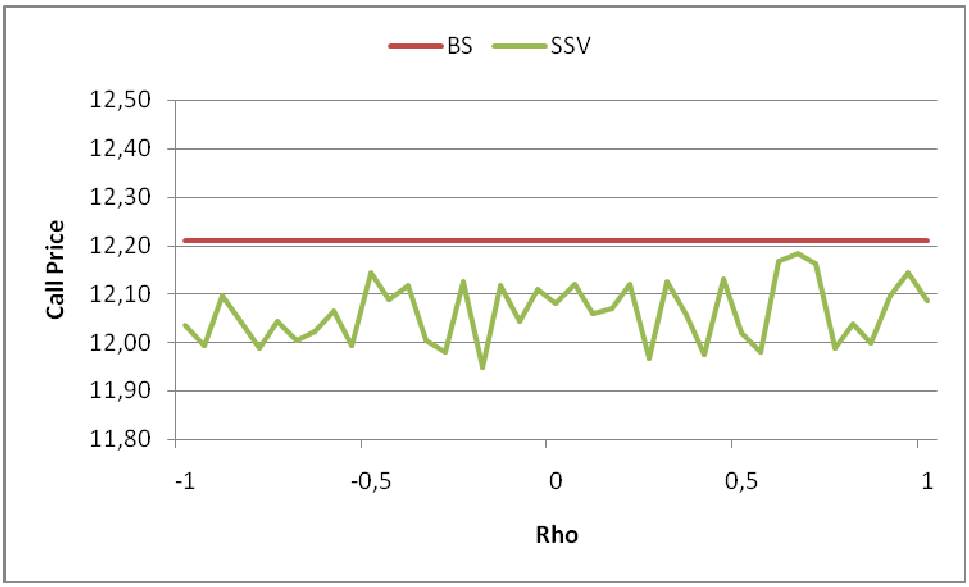

Figure 4.19 Effect of *Rho* on the call price in the SSV model

Figure 4.20 Effect of *Rho* on the volatility in the SSV model

Intervals2

The parameter *intervals2* refers to how many times the time to maturity is split up. The volatility is generated at each interval as many times as the simulation parameter is set equal to. The volatility used in the SSV model to calculate the option price is the volatility found at maturity. With a small number of intervals the call price is quite different from the Black-Scholes price.

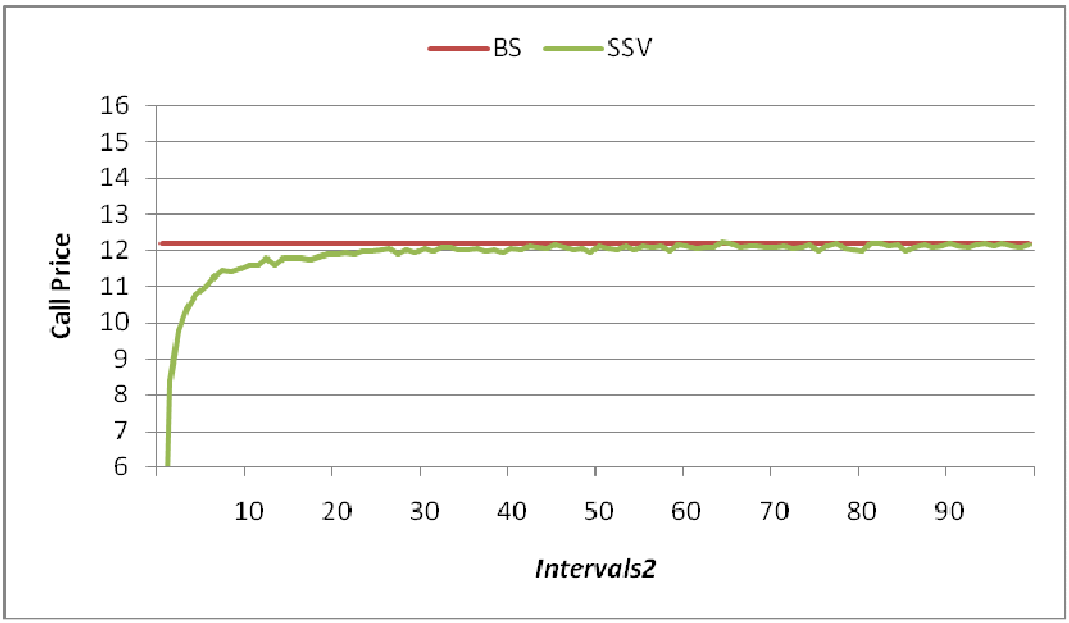

Figure 4.21 Effect of *Intervals2* on the call price in SSV model

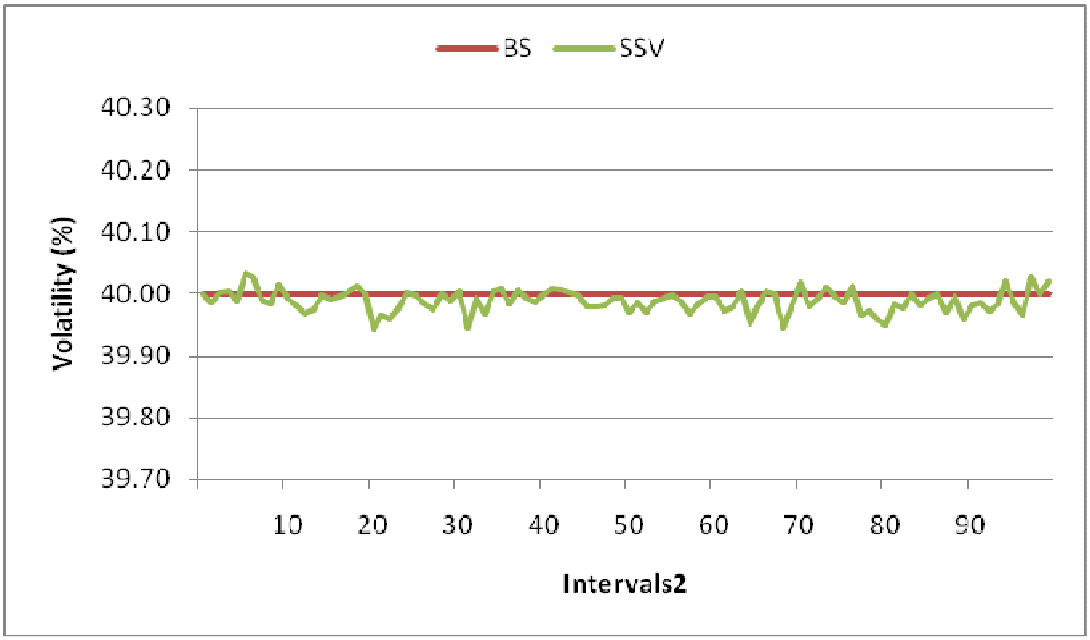

Figure 4.22 Effect of *Intervals2* on the volatility in SSV model

Simulations

The behavior of volatility and call price due to changes in simulations in the SSV model is very similar with the patterns found in the SV model. The more times we simulate the stochastic volatility the less the fluctuation of the volatility and call price is found. The range of the fluctuations becomes very small once we reach around one thousand simulations.

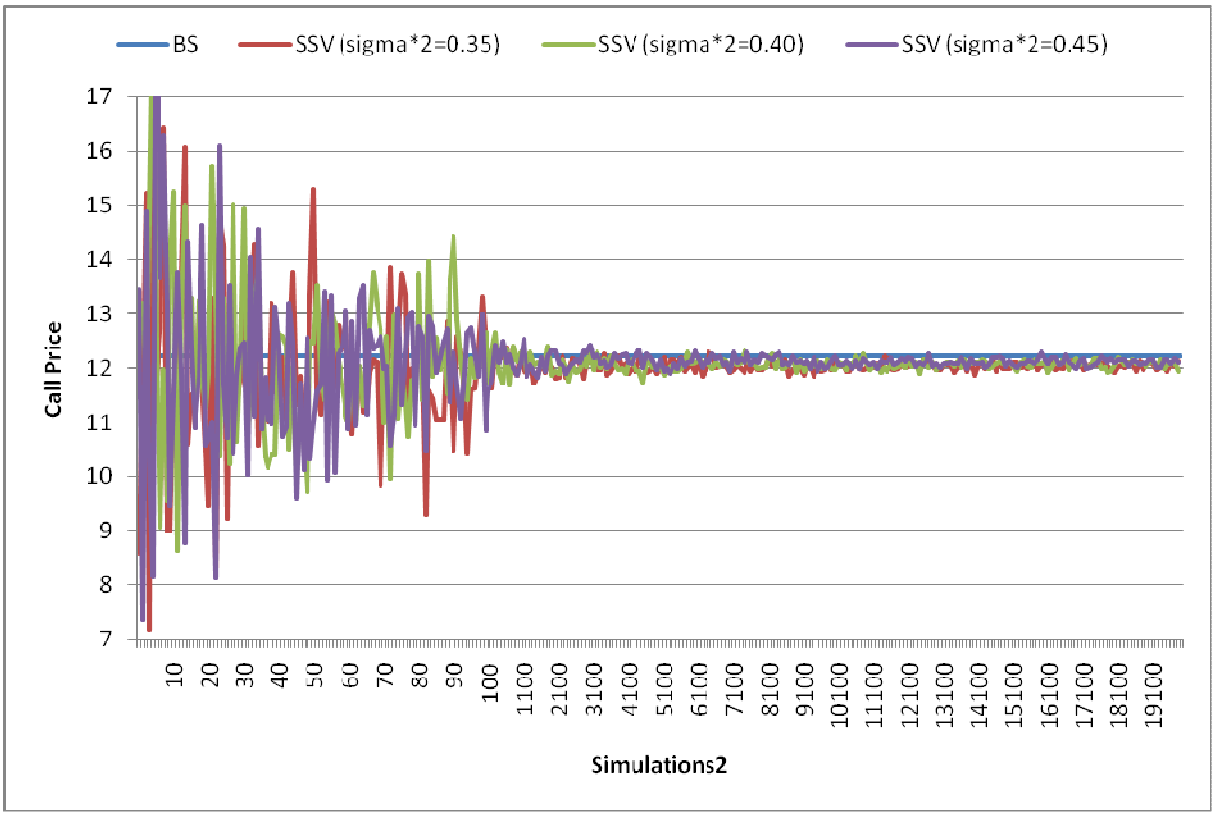

Figure 4.23 Effect of *Simulations2* on call price in the SSV model

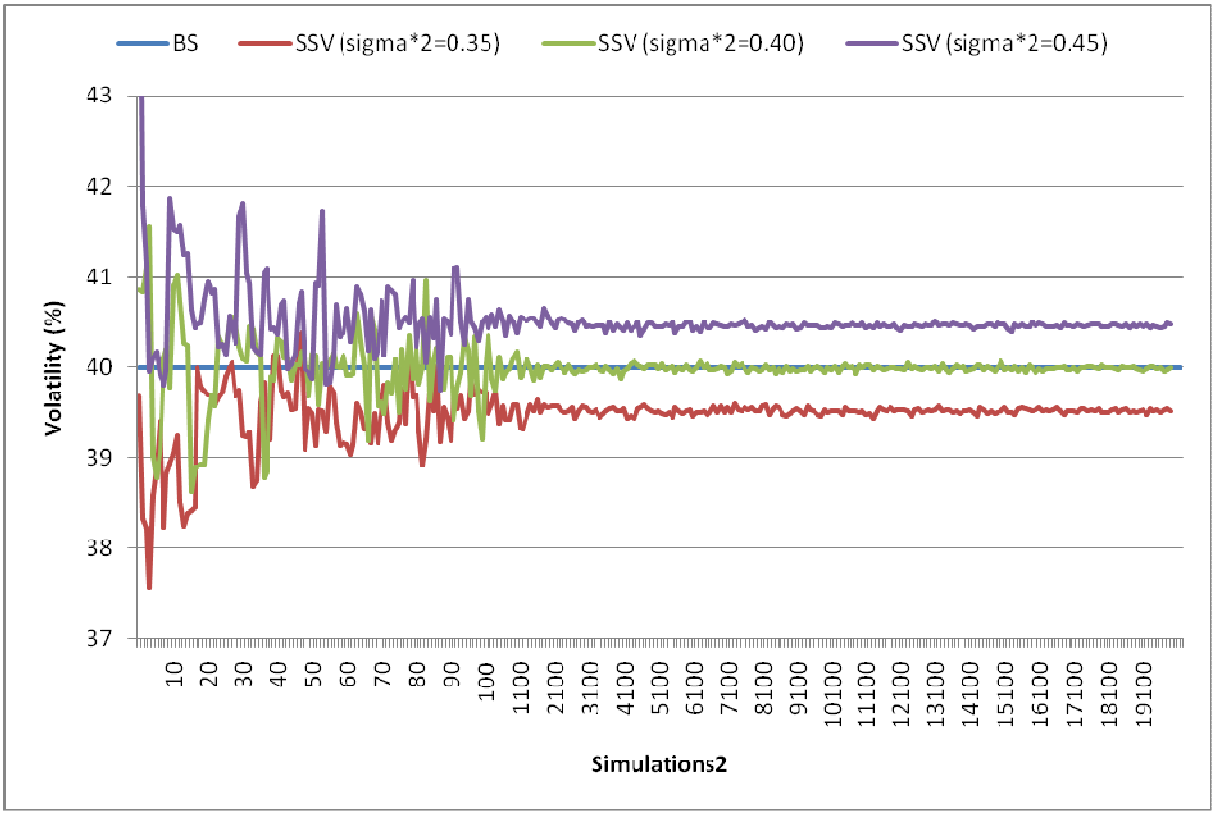

Figure 4.24 Effect of *Simulations2* on the volatility in the SSV model

Comparison of Output

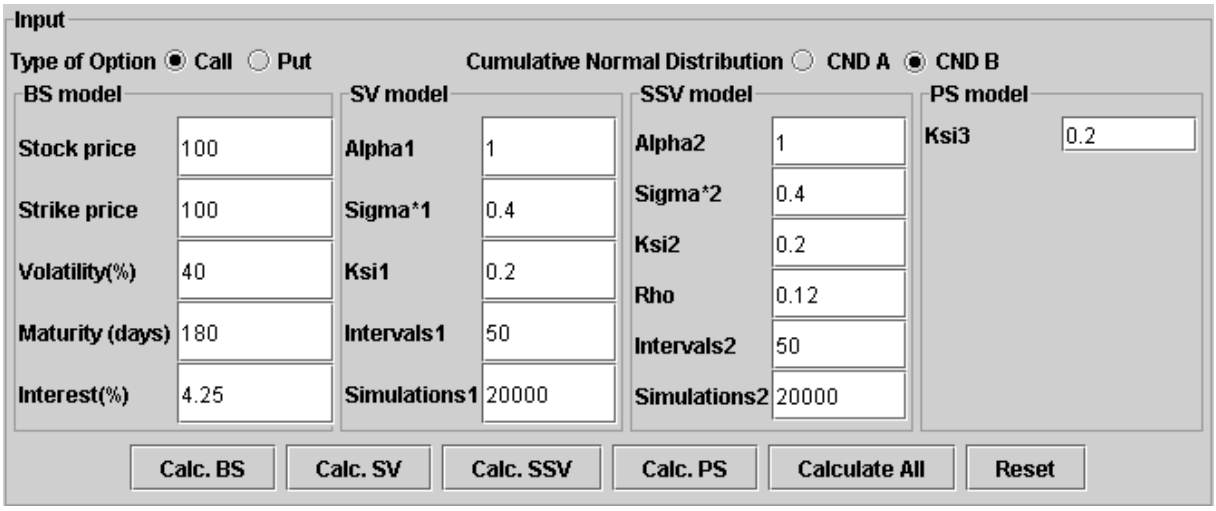

Figure 4.25 Values for the first comparison of model prices

The output of the four models is displayed in figure 4.26 using the same parameter values as in the previous analysis (Figure 4.25). The Black Scholes, depicted in blue, is set as the option price to which the other models are compared with. The Series Solution, *SV* and *SSV* are thus shown as the differences from Black Scholes. This is done by subtracting the BS price from the other three models' call prices. Due to the small difference in the output it would be impossible to clearly depict any patterns. To make these patterns visible, the difference between the model price and the Black Scholes price is exaggerated by a multiple (example ((*SV-BS*)*25 + *BS*). In figure 4.26 this difference is multiplied twenty-five times. The model prices are then shown as the ratio between stockprice and strikeprice changes from 0.75 to 1.25.

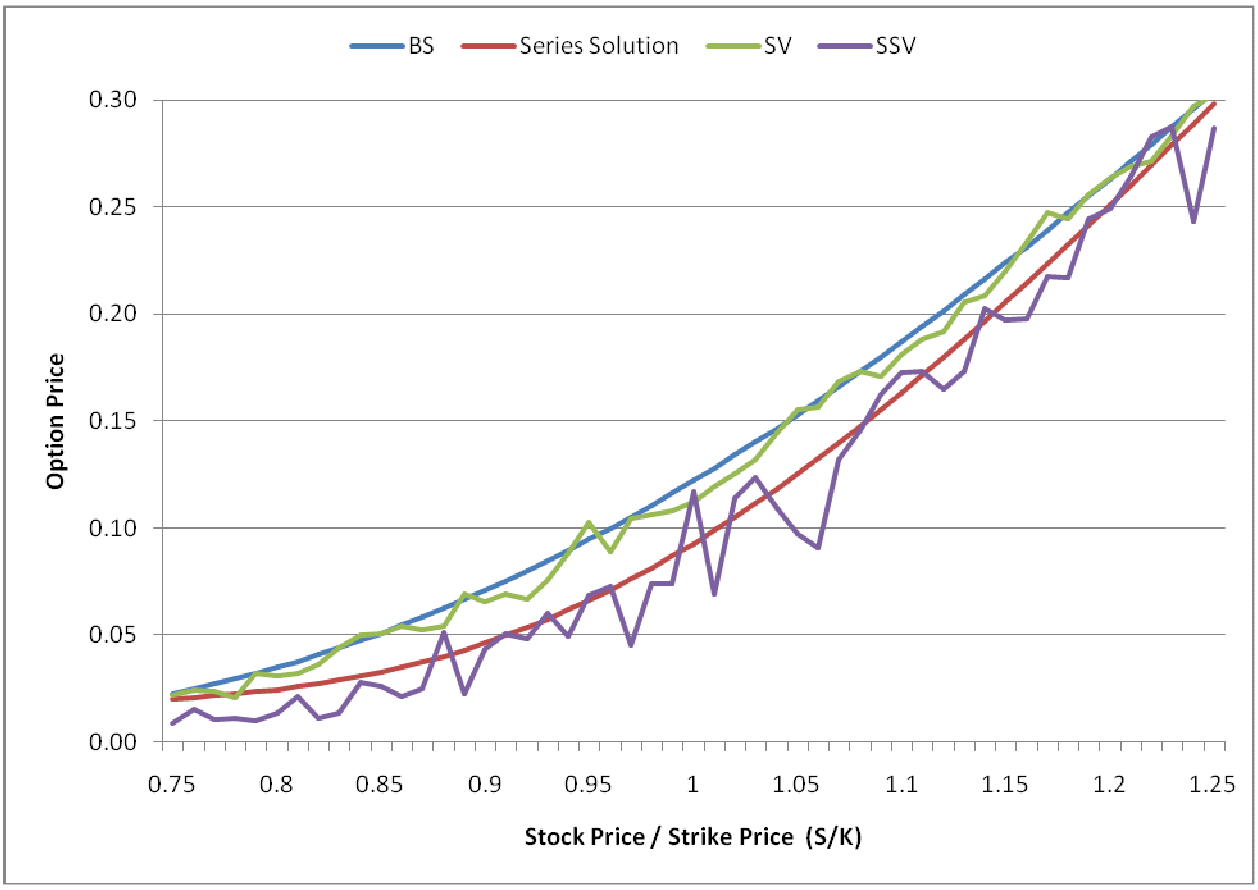

Figure 4.26 Option price difference exaggerated 25 times

With a slight lower volatility (15%), *ksi1*, *ksi2* and *ksi3* increased to one and the interest rate set to zero, the differences in call price become more obvious (figure 4.27). Black Scholes overprice near- and at the money call options while it produces a lower price for deep out-of-the money options. This is true compared with the Series Solution and SSV. The pattern for the SV difference, if any, is very small. The reason for this may be that the SV model generates a volatility that is in fact used in the Black Scholes formula.

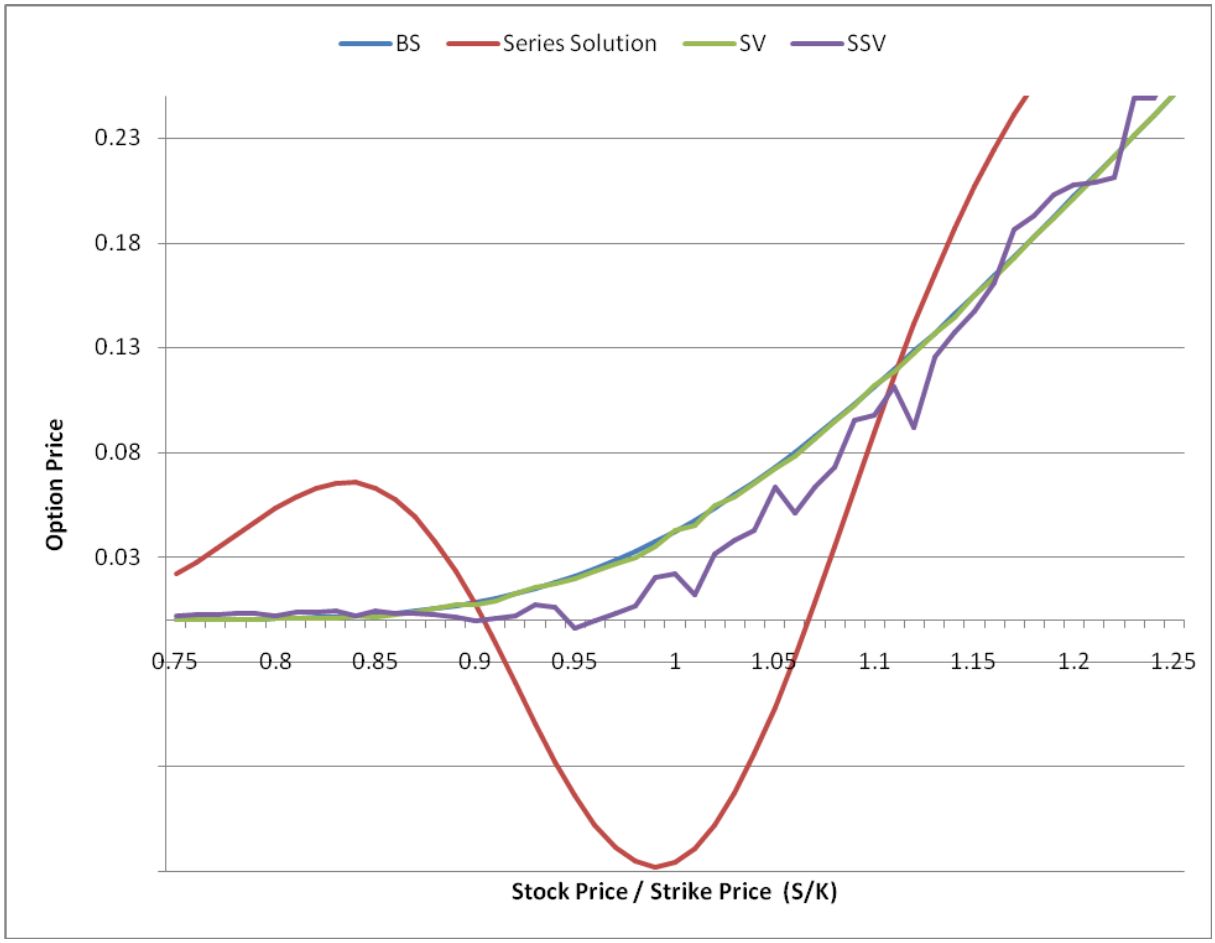

Figure 4.27 Option price difference exaggerated 25 times with new parameter values

If we allow the SV difference from BS to be multiplied 250 times (Figure 4.28), we see that SV also follows the same pattern. The SSV price difference is now multiplied 50 times to improve the visibility of the differences.

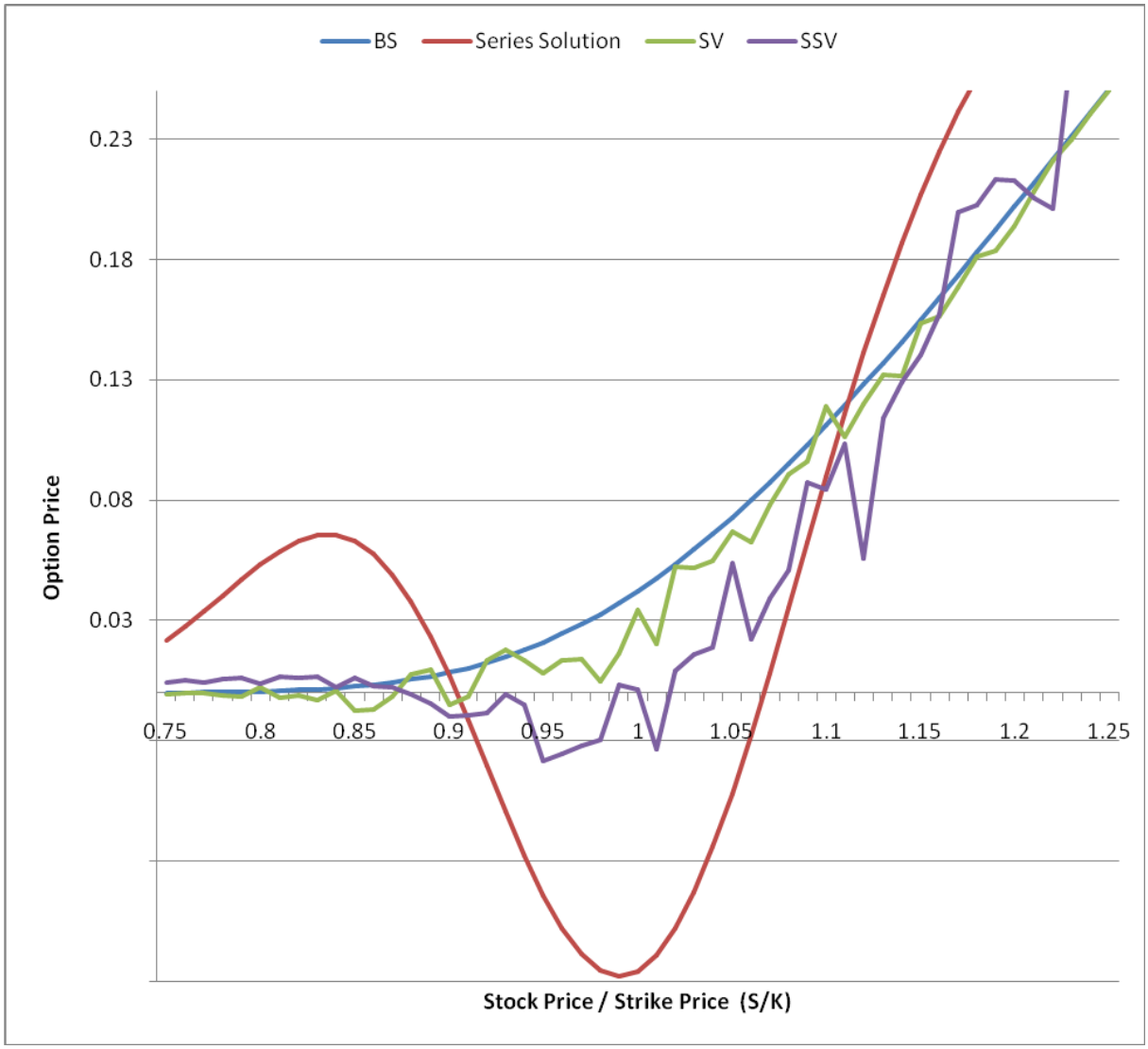

Figure 4.28 Option price difference multiplied 25 (Series Sol.), 250 (SV) and 50 (SSV) times

By increasing *ksi1*, *ksi2* and *ksi3* to two the price difference becomes even more evident (Figure 4.29). The Series Solution is showing much larger difference with the Black Scholes price and has its difference only enhanced by a multiple of three.

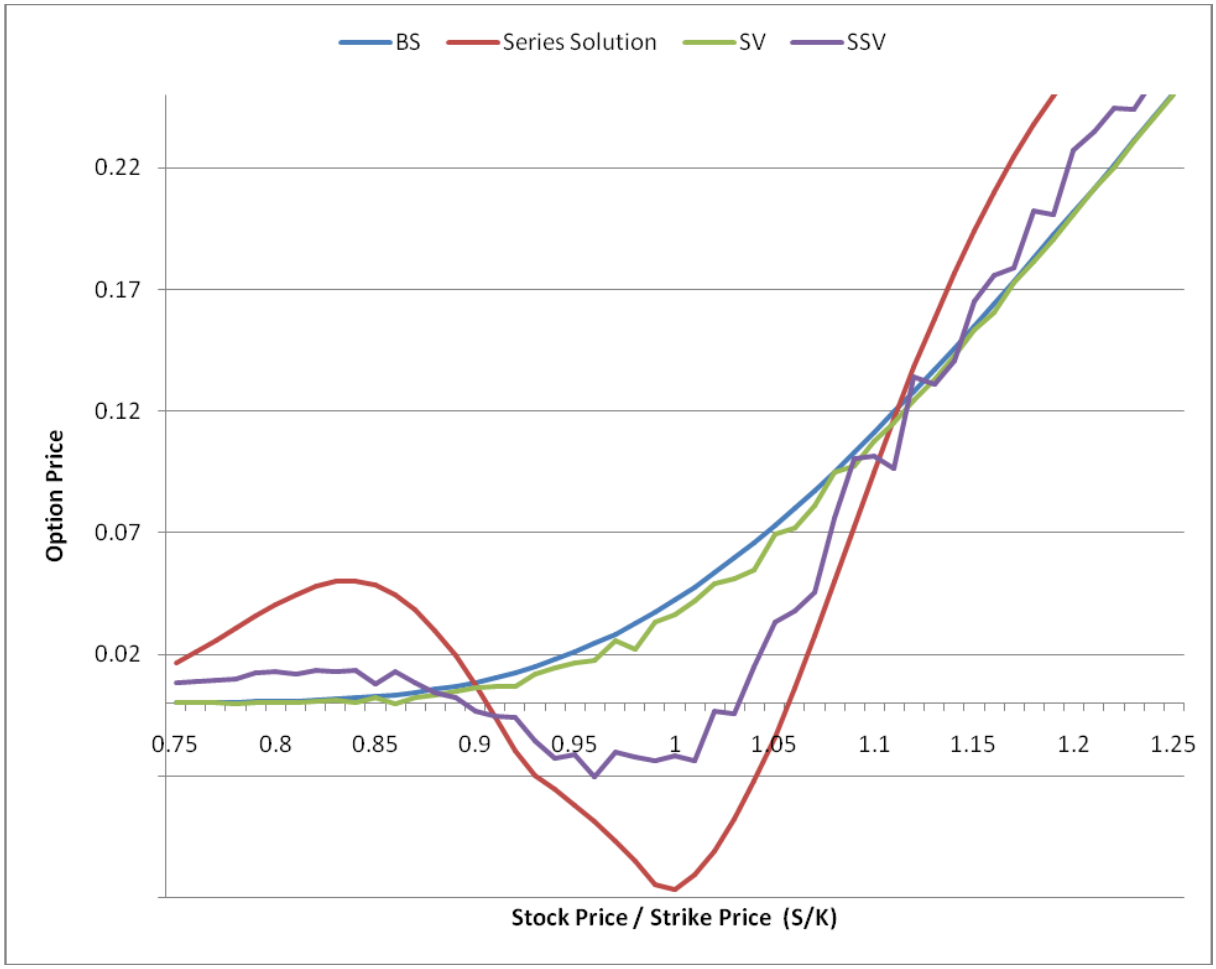

Figure 4.29 Option price difference with *ksi1*,*ksi2*,*ksi3*=2, multiplied 25 (Series Sol.), 250 (SV) and 50 (SSV) times

The same pattern seems to exist for the stochastic models, SV and SSV, as the time to maturity is increased from 180- to 480 days. It is much more obvious in SSV than SV (Figures 4.30 and 4.31).

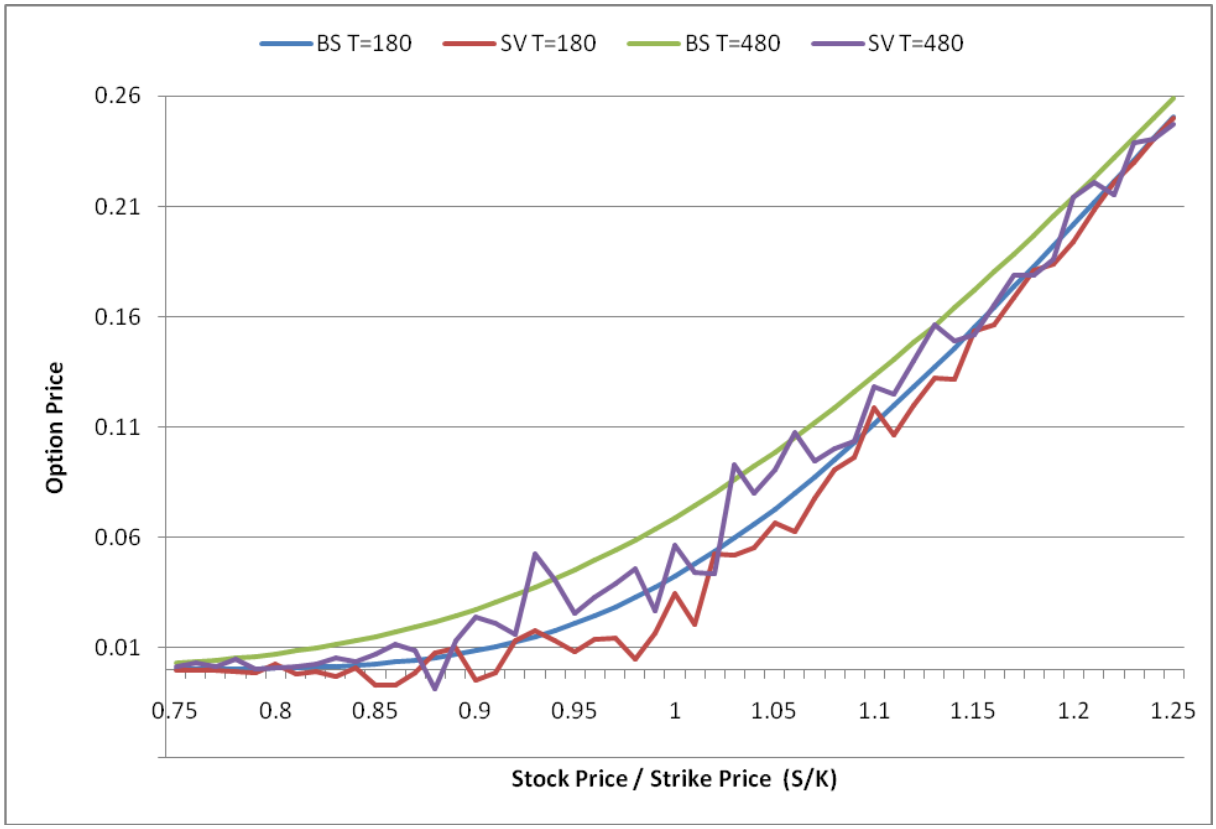

Figure 4.30 Option price difference for SV with T = 180 & 480 multiplied 25 times

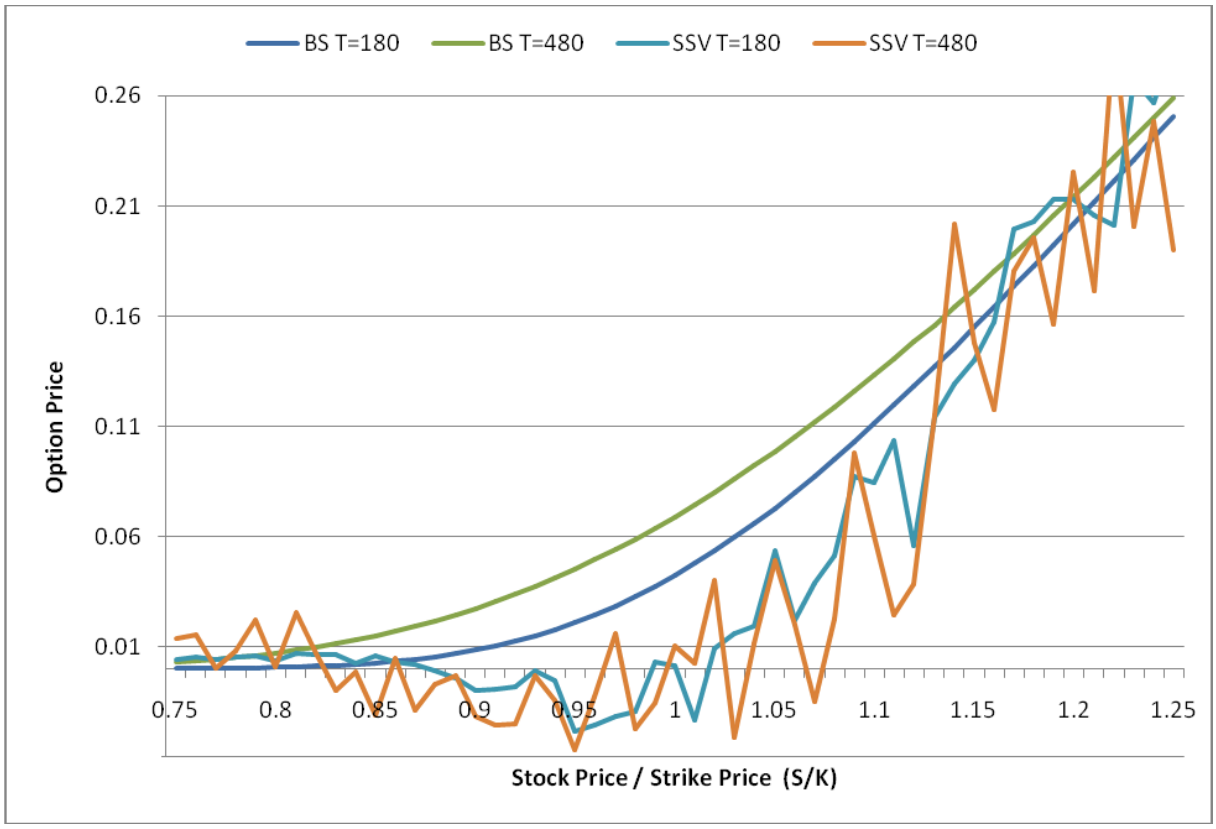

Figure 4.31 Option price difference for SSV with T = 180 & 480 multiplied 25 times

Section V

Conclusions

This master thesis has dealt with:

1. Creating an application in a Java Applet where all the models are calculating the European call and put option prices with no dividends.
2. Testing the effect of the parameters of the stochastic models on volatility and option price.

The attempt to create a program calculating option prices with stochastic volatility based on one of the first articles dealing with stochastic option prices has been successfully completed.

Regarding the effect of the parameters; the parameters that cause the largest difference in prices are ξ (the diffusion coefficient of the volatility) and σ^* (a constant term in the mean-reverting process describing μ (the drift coefficient of the volatility)). A value of ξ up to 1 results in option prices that do not differ largely from each other. The difference between σ^* and σ (the diffusion coefficient of the asset price) forces the volatility in *SV* and *SSV* to be larger or smaller depending on the magnitude of the difference. With σ^* and σ set equal, the difference between the option price of Black-Scholes and the three models is the smallest. The effect of the number of *intervals* (into which the time to maturity is divided) is not as important as the amount of Monte Carlo *simulations* for both the stochastic models (*SV* and *SSV*). When the number of simulations exceeds 1000 the option price given converges with the option price of the two non-stochastic models. With certain parameter values[1] we find a pattern of the Black Scholes model overpricing at-the-money options and underpricing in- and out-of-the money options. The pattern becomes more evident as ξ and time-to-maturity increased. This is true regarding all three models compared with Black Scholes and is in agreement with Hull & White's findings.

The Java source code can be extended to include option prices for American call options with no dividends. Additions to the applet can easily include other option pricing models as well. The Monte Carlo simulations of the stochastic processes are done within a

[1] S=100, K=100, σ=40%, T=180d, r=4.25%, $\alpha_1, \alpha_2 = 1$, $\sigma_1^*, \sigma_2^* = 40\%$, $\xi_1, \xi_2, \xi_3 = 0.2$, $\rho = 0.12$
intervals1,intervals2=50, simulations1,simulations2=20,000

reasonable timeframe. The option price calculator provides a result for all four models (approximately 3 million calculations depending on simulations and intervals) on average within five seconds. The time needed is much greater when a very accurate cumulate probability distribution value (CND-A) is required. If a good approximation of the probability distribution (CND-B) is sufficient then the time needed is greatly reduced.

The actual parameter values are difficult to determine, especially *rho* (the correlation coefficient between stock price and volatility). In our paper we explored the parameter values and partly relied on the Hull & White article. With more research (gathering data from the market about the stock price and the option price) a decision of the correct parameter values can be made. This would enable the user to select the model estimating the option price accurately.

References

[1] http://bradley.bradley.edu/~arr/bsm/pg03.html last visited 10:00, 06/10/2007

[2] http://www.biz.uiowa.edu/faculty/dbates/papers/survey.pdf last visited 10:00, 06/10/2007

[3] M. Kijima, *Stochastic Processes with Applications to Finance*, Chapman & Hall/CRC, Florida, 2003

[4] D. Wackerly, W. Mendenhall III, R. L. Scheaffer, *Mathematical Statistics with Applications*, 6th edition, Duxbury, California, 2002

[5] Ren-Raw Chen, San-Lin Chung and Tyler T.L. Yang, *An Easy Derivatives Pricing Algorithm Under A Dynamically Complete Multi-Asset*, November 2000.

[6] John C. Hull, *Options, Futures, And Other Derivatives*, 6th Edition, Pearson Education, New Jersey, 2006. pp 265, 269, 293

[7] Jan R.M Röman, *Analytical Finance I*, Department of Mathematics and Physics/ Mälardalen University, Västerås, August 2006. pp 101,103.

[8] John Hull and Alan White, *The Pricing of Options on Assets with Stochastic Volatilities,* The Journal of Finance, Vol. 42, No. 2 (June, 1987), pp. 281-300.

Appendix

The written code for the option calculator is found in the appendix. Each class is divided in sections A through G.

Part A – HTML File

```
<HTML>
    <HEAD>
        <TITLE>MyApplet Example1</TITLE>
    </HEAD>
    <BODY>
        <H1>Master thesis: Stochastic volatility models in option pricing</H1>
        <HR>
        <P>
            <APPLET CODE="Thesis.class" WIDTH="580" HEIGHT="350">
            </APPLET>
        </P>
        <HR>
    </BODY>
</HTML>
```

Part B – Black Scholes

```
/**
 * Stochastic Volatility Models in Option Pricing
 *
 * Copyright (c) 2007 Mälardalen University
 * Högskoleplan Box 883, 721 23 Västerås, Sweden.
 * All Rights Reserved.
 *
 * The copyright to the computer program(s) herein
 * is the property of Mälardalen University.
 * The program(s) may be used and/or copied only with
 * the written permission of Mälardalen University
 * or in accordance with the terms and conditions
 * stipulated in the agreement/contract under which
 * the program(s) have been supplied.
 *
 * Description: Option price calculator
 * @version 1.0  Dec 17th 2007
 * @authors: Michail Kalavrezos, Michael Wennermo
 * Email: michail_kalavrezos@yahoo.se, mikewennermo@yahoo.se
 */


import java.util.*;

public class BlackScholesFormula {


     public BlackScholesFormula() {
     }

   public  double BlackScholesCallA(double S, double K, double sigma,double
                                                              T, double r) {

      double d1 = (Math.log(S/K) + ((r/100) + Math.pow((sigma/100), 2)/2) *
      ((double)T/360)) / ((sigma/100) * Math.sqrt((double)T/360));
      double d2 = d1 - ((sigma/100) * Math.sqrt((double)T/360));
      double p = 0;
      p = S* new NormalDistribution().getProbability(d1);
      double d=Math.max( p - K * Math.exp(-(r/100) * (T/360)) *
      new NormalDistribution().getProbability(d2),0);

      return d;
    }

    public  double BlackScholesCallB(double S, double K, double
                                      sigma,double T,  double r) {

      double d1 = (Math.log(S/K) + ((r/100) + Math.pow(( sigma/100), 2)/2)
      * ((double)T/360)) / ((sigma/100) * Math.sqrt((double)T/360));
      double d2 = d1 - ( (sigma/100) * Math.sqrt((double)T/360));
      double p = 0;
      p = S* new NormalDistributionCND().CND(d1);
      double d =Math.max( p - K * Math.exp(-(r/100) * (T/360)) *  new
      NormalDistributionCND().CND(d2),0);
```

```
        return d;
    }


     public double BlackScholesPutA(double S, double K, double sigma,double
                                                          T,  double r) {

       double d1 = (Math.log(S/K) + ((r/100) + Math.pow(( sigma/100), 2)/2)
       * ((double)T/360)) / ((sigma/100) * Math.sqrt((double)T/360));
       double d2 = d1 - ( (sigma/100) * Math.sqrt((double)T/360));
       double p = 0;
       double norm1=new NormalDistribution().getProbability(-d1);
       double norm2=new NormalDistribution().getProbability(-d2);

       p = - S*norm1;
       p = Math.max((p +  K * Math.exp(-(r/100) * (T/360))  * norm2),0);

       return p;
     }

      public double BlackScholesPutB(double S, double K, double
                                     sigma,double T,  double r) {

       double d1 = (Math.log(S/K) + ((r/100) + Math.pow(( sigma/100), 2)/2)
       * ((double)T/360)) / ((sigma/100) * Math.sqrt((double)T/360));
       double d2 = d1 - ( (sigma/100) * Math.sqrt((double)T/360));
       double p = 0;
       double norm1=new NormalDistributionCND().CND(-d1);
       double norm2=new NormalDistributionCND().CND(-d2);

       p = - S*norm1;
       p = Math.max((p +  K * Math.exp(-(r/100) * (T/360))  * norm2),0);

       return p;
     }

}
```

Part C – Normal Distribution

CND-A

```
/**
 * Stochastic Volatility Models in Option Pricing
 *
 * Copyright (c) 2007 Mälardalen University
 * Högskoleplan Box 883, 721 23 Västerås, Sweden.
 * All Rights Reserved.
 *
 * The copyright to the computer program(s) herein
 * is the property of Mälardalen University.
 * The program(s) may be used and/or copied only with
 * the written permission of Mälardalen University
 * or in accordance with the terms and conditions
 * stipulated in the agreement/contract under which
 * the program(s) have been supplied.
 *
 * Description: Option price calculator
 * @version 1.0  Dec 17th 2007
 * @authors: Michail Kalavrezos, Michael Wennermo
 * Email: michail_kalavrezos@yahoo.se, mikewennermo@yahoo.se
 */


public class NormalDistribution{

   public  NormalDistribution(){
   }

   //d1= the first argument on BS
   //d2= the first argument on BS
   //create an array of doubles

   //this is a method that finds the cumulative probability
   //for a standard normal variable

   public double getProbability(double u){

     double[] points= new double[200000];
     double[] points1= new double[200000];
     double[] area=new double[200000] ;
     int k=0;
     int r=0;
     double t=-100.00;
     points[0]=0.00;
     points1[0]=0.00;
     for ( k=1;k<200000;k++){

       points[k]=(Math.exp(-Math.pow((t+0.001*k),2)/2));
       points[k]=points[k]/Math.sqrt(2*Math.PI);
```

```
        area[k]=(Math.abs(points[k]+points[k-1])/2)*0.001;


        points1[k]=points1[k-1]+ area[k];


      }

      double p=Math.exp(-0)/Math.sqrt(2*Math.PI);
      double distance=Math.abs(-100-u);
      //System.out.println(distance);
      search:
      for ( k=1;k<200000;k++){
            if (Math.abs(-100-u+0.001*k)<distance){
                  distance=Math.abs(-100-u+0.001*k);
                  //System.out.println(distance);
            }
            else if ( Math.abs(-100-u+0.001*k)>=distance ){
                  r=k-1;
                  break search;
            }
      }

      return points1[r];

   }
}
```

CND-B

```
/**
 * Stochastic Volatility Models in Option Pricing
 *
 * Copyright (c) 2007 Mälardalen University
 * Högskoleplan Box 883, 721 23 Västerås, Sweden.
 * All Rights Reserved.
 *
 * The copyright to the computer program(s) herein
 * is the property of Mälardalen University.
 * The program(s) may be used and/or copied only with
 * the written permission of Mälardalen University
 * or in accordance with the terms and conditions
 * stipulated in the agreement/contract under which
 * the program(s) have been supplied.
 *
 * Description: Option price calculator
 * @version 1.0  Dec 17th 2007
 * @authors: Michail Kalavrezos, Michael Wennermo
 * Email: michail_kalavrezos@yahoo.se, mikewennermo@yahoo.se
 */

// Calculates the Normal distribution with |e(x)| < 1.0e-8

public class NormalDistributionCND {

  public NormalDistributionCND(){
  }

  public double CND(double x){

  double sgn;
  double x2, q0, q1, q2;

  if (x < 0.0) {
    x = -x;
    sgn = -1.0;
  }
  else if (x > 0.0) {
    sgn = 1.0;
  }
  else /* x == 0.0 */
    return 0.5;

  if (x > 20.0) {
    if (sgn < 0)
      return 0.0;
    else
      return 1.0;
  }

  x *= 0.70710678118654746;     // 1/sqrt(2)
  x2 = x*x;
```

```
  if (x < 0.46875) {
    q1 = 3209.37758913846947
      +x2*(377.485237685302021
            +x2*(113.864154151050156
                  +x2*(3.16112374387056560
                        +x2*0.185777706184603153)));

    q2 = 2844.23683343917062
      +x2*(1282.61652607737228
            +x2*(244.024637934444173
                  +x2*(23.6012909523441209
                        +x2)));
    return 0.5*(1.0+sgn*x*q1/q2);
  }

  else if (x < 4.0) {
    q1 = 1230.33935479799725
      +x*(2051.07837782607147
           +x*(1712.04761263407058
                +x*(881.952221241769090
                     +x*(298.635138197400131
                          +x*(66.1191906371416295
                               +x*(8.88314979438837594
                                    +x*(0.564188496988670089
                                         +x*2.15311535474403846e-8)))))));

    q2 = 1230.33935480374942
      +x*(3439.36767414372164
           +x*(4362.61909014324716
                +x*(3290.79923573345963
                     +x*(1621.38957456669019
                          +x*(537.181101862009858
                               +x*(117.693950891312499
                                    +x*(15.7449261107098347
                                         +x)))))));
    return 0.5*(1.0+sgn*(1.0-Math.exp(-x2)*q1/q2));
  }

else {
    q0 = 1.0/x2;
    q1 = 6.58749161529837803e-4
      +q0*(0.0160837851487422766
            +q0*(0.125781726111229246
                  +q0*(0.360344899949804439
                        +q0*(0.305326634961232344
                              +q0*0.0163153871373020978))));

    q2 = 2.33520497626869185e-3
      +q0*(0.0605183413124413191
            +q0*(0.527905102951428412
                  +q0*(1.87295284992346047
                        +q0*(2.5685201922898242+q0))));
    return 0.5*(1.0+sgn*(1.0-Math.exp(-x2)/x*(0.56418958354775628-
            q0*q1/q2)));
  }
 }
}
```

Part D – Power Series

```
/**
 * Stochastic Volatility Models in Option Pricing
 *
 * Copyright (c) 2007 Mälardalen University
 * Högskoleplan Box 883, 721 23 Västerås, Sweden.
 * All Rights Reserved.
 *
 * The copyright to the computer program(s) herein
 * is the property of Mälardalen University.
 * The program(s) may be used and/or copied only with
 * the written permission of Mälardalen University
 * or in accordance with the terms and conditions
 * stipulated in the agreement/contract under which
 * the program(s) have been supplied.
 *
 * Description: Option price calculator
 * @version 1.0  Dec 17th 2007
 * @authors: Michail Kalavrezos, Michael Wennermo
 * Email: michail_kalavrezos@yahoo.se, mikewennermo@yahoo.se
 */


import java.util.Random;

public class PowerSeries {

   public PowerSeries() {
   }

   public double calculatePSA(double S, double K, double sigma, double
                              time, double interestrate, double ksi3) {

   // These next 3 lines ease the work when using sigma, T and r
   double sigmap = sigma/100;
   double T = time/360;
   double r = interestrate/100;

   // d1 and d2
   double d1 = (Math.log(S/K) + (r + Math.pow(sigmap, 2)/2) * T) / (sigmap
   * Math.sqrt(T));
   double d2 = d1 - (sigmap * Math.sqrt(T));
   double c,k,x,n,part1,part2,part3,part4,part5,part6,ps = 0;

   // Regular Black Scholes and k = ksi^2(T-t)
   double d = S* new NormalDistribution().getProbability(d1);
   c = Math.max(d - K * Math.exp(-r * T) *  new
   NormalDistribution().getProbability(d2),0);
   k = Math.pow(ksi3, 2) * T;
   x = (Math.exp(-d1*d1/2)) / (1/(Math.sqrt(2*Math.PI)));

   // calculating formula (9)
```

```
    part1 = 0.5* (S*(double)Math.sqrt(T)*x*(d1*d2-
             1)/(4*Math.pow((sigmap),3)));              // 1st term in eq
    part2 = (2*Math.pow(sigmap,4)*(Math.exp(k)-k-1))/(Math.pow(k,2)) -
             Math.pow(sigmap,4);                        // 2nd term in eq
    part3 = (double)(1/6) * S*Math.sqrt(T)*x*((d1*d2-3)*(d1*d2-1)-
             (Math.pow(d1,2)+Math.pow(d2,2)));
    part4 = 8*Math.pow(sigmap,5);
    part5 = Math.pow(sigmap,6);
    part6 = (Math.exp(3*k)-
             (9+18*k)*Math.exp(k)+(8+24*k+18*Math.pow(k,2)+6*Math.pow(k,3)))
             / (3*Math.pow(k,3));

    ps = c + part1*part2 + (part3/part4)*part5*part6;

    ps=Math.max(ps,0);

        return ps;
    }

    public double calculatePSB(double S, double K, double sigma, double
                           time, double interestrate, double ksi3) {

    double sigmap = sigma/100;
    double T = time/360;
    double r = interestrate/100;

    double d1 = (Math.log(S/K) + (r + Math.pow(sigmap, 2)/2) * T) / (sigmap
                * Math.sqrt(T));
    double d2 = d1 - (sigmap * Math.sqrt(T));
    double c,k,x,n,part1,part2,part3,part4,part5,part6,ps = 0;

    double d = S* new NormalDistributionCND().CND(d1);
    c = Math.max(d - K * Math.exp(-r * T) *  new
          NormalDistributionCND().CND(d2),0);
    k = Math.pow(ksi3, 2) * T;
    x = (Math.exp(-d1*d1/2)) / (1/(Math.sqrt(2*Math.PI)));

    part1 = 0.5* (S*(double)Math.sqrt(T)*x*(d1*d2-
             1)/(4*Math.pow((sigmap),3)));
    part2 = (2*Math.pow(sigmap,4)*(Math.exp(k)-k-1))/(Math.pow(k,2)) -
             Math.pow(sigmap,4);
    part3 = (double)(1/6) * S*Math.sqrt(T)*x*((d1*d2-3)*(d1*d2-1)-
             (Math.pow(d1,2)+Math.pow(d2,2)));
    part4 = 8*Math.pow(sigmap,5);
    part5 = Math.pow(sigmap,6);
    part6 = (Math.exp(3*k)-
(9+18*k)*Math.exp(k)+(8+24*k+18*Math.pow(k,2)+6*Math.pow(k,3))) /
(3*Math.pow(k,3));

    ps = c + part1*part2 + (part3/part4)*part5*part6;

    ps=Math.max(ps,0);

        return ps;
    }

}
```

Part E – SV Model

```
/**
 * Stochastic Volatility Models in Option Pricing
 *
 * Copyright (c) 2007 Mälardalen University
 * Högskoleplan Box 883, 721 23 Västerås, Sweden.
 * All Rights Reserved.
 *
 * The copyright to the computer program(s) herein
 * is the property of Mälardalen University.
 * The program(s) may be used and/or copied only with
 * the written permission of Mälardalen University
 * or in accordance with the terms and conditions
 * stipulated in the agreement/contract under which
 * the program(s) have been supplied.
 *
 * Description: Option price calculator
 * @version 1.0  Dec 17th 2007
 * @authors: Michail Kalavrezos, Michael Wennermo
 * Email: michail_kalavrezos@yahoo.se, mikewennermo@yahoo.se
 */


import java.util.Random;

public class StochasticVolatilityFile1{

  double vione;
  double vitwo;
  public Random generator1=new Random();
  public Random generator2=new Random();
  public Random generator3=new Random();

  public StochasticVolatilityFile1(){
  }


    public double simulateVolatility1(double alpha,double ksi,
                                      double sigmastar,int
                                       numberOfSubintervals,
                                      int numberOfIterations,double sigma,
                                      double timeToMaturity){

double deltati=timeToMaturity/(360*numberOfSubintervals);
double sigmap = sigma/100;
double vol=0.0;
double[] vi=new double[numberOfSubintervals+1];
//here we create an array of doubles representing Vi in each of the
//time steps (input - intervals1)
vi[0]=Math.pow(sigmap,2);

for(int s=1;s<numberOfIterations+1;s++){
   ouble sumvi=0.0;
```

```
   sigmap = sigma/100;
      for (int d=1;d<numberOfSubintervals+1;d++){
      // vi[d] is equal to the vol in the last time step
         vi[d] =vi[d-1]*Math.exp((alpha*(sigmastar-sigmap)-
         (Math.pow(ksi,2)/2))*deltati+
         generator1.nextGaussian()*ksi*Math.sqrt(deltati));
         sumvi= sumvi+vi[d];
         sigmap=Math.sqrt(vi[d]);
      }
   vol =sumvi/numberOfSubintervals;
   vione= vione+vol;
}

vol=Math.sqrt(vione/numberOfIterations)*100;
return vol;
    }

public double simulateVolatility2(double alpha,double ksi,
                                  double sigmastar,int
                                  numberOfSubintervals,
                                  int numberOfIterations,double sigma,
                                  double timeToMaturity){

double deltati=timeToMaturity/(360*numberOfSubintervals);
double sigmap = sigma/100;
double vol=0.0;
double[] vi=new double[numberOfSubintervals+1];
vi[0]=Math.pow(sigmap,2);
   for(int s=1;s<numberOfIterations+1;s++){
      double sumvi=0.0;
      sigmap = sigma/100;
         for (int d=1;d<numberOfSubintervals+1;d++){

vi[d] =vi[d-1]*Math.exp((alpha*(sigmastar-sigmap)-
(Math.pow(ksi,2)/2))*deltati-
generator2.nextGaussian()*ksi*Math.sqrt(deltati));
sumvi= sumvi+vi[d];
sigmap=Math.sqrt(vi[d]) ;
}
vol =sumvi/numberOfSubintervals;
vione= vione+vol;
}

vol=Math.sqrt(vione/numberOfIterations)*100;
return vol;
}


public double simulateVolatility3(double alpha,double ksi,
                                  double sigmastar,int
                                  numberOfSubintervals,
                                  int numberOfIterations,double sigma,
                                  double timeToMaturity){

double deltati=timeToMaturity/(360*numberOfSubintervals);
                                  double sigmap = sigma/100;
                                  double vol=0.0;
                                  double[] vi=new
double[numberOfSubintervals+1];
vi[0]=Math.pow(sigmap,2);
```

```
for (int d=1;d<numberOfSubintervals+1;d++){
                                                double sim=0.0;
                                                double sumsim=0.0;
for(int s=1;s<numberOfIterations+1;s++){
sim=vi[d-1]*Math.exp((alpha*(sigmastar-sigmap)-
(Math.pow(ksi,2)/2))*deltati-
generator3.nextGaussian()*ksi*Math.sqrt(deltati));
sumsim= sumsim+sim;
}
vi[d]=sumsim/numberOfIterations;
sigmap=Math.sqrt(vi[d]);
vitwo= vitwo+vi[d];
}

vol=Math.sqrt(vitwo/numberOfSubintervals)*100;
return vol;
    }

}
```

Part F – SSV Model

```
/**
 * Stochastic Volatility Models in Option Pricing
 *
 * Copyright (c) 2007 Mälardalen University
 * Högskoleplan Box 883, 721 23 Västerås, Sweden.
 * All Rights Reserved.
 *
 * The copyright to the computer program(s) herein
 * is the property of Mälardalen University.
 * The program(s) may be used and/or copied only with
 * the written permission of Mälardalen University
 * or in accordance with the terms and conditions
 * stipulated in the agreement/contract under which
 * the program(s) have been supplied.
 *
 * Description: Option price calculator
 * @version 1.0  Dec 17th 2007
 * @authors: Michail Kalavrezos, Michael Wennermo
 * Email: michail_kalavrezos@yahoo.se, mikewennermo@yahoo.se
 */


import java.util.Random;

public class StochasticPriceVolatility{

  public Random generator1=new Random();
  public Random generator2=new Random();

  //one constructor
  public  StochasticPriceVolatility(){
  }
  double p1=0.0;
  double p1estimation=0.0;
  double p2=0.0;
  double p2estimation=0.0;
  double p3=0.0;
  double p3estimation=0.0;
  double p4=0.0;
  double p4estimation=0.0;
  double q1=0.0;
  double q1estimation=0.0;
  double q2=0.0;
  double q2estimation=0.0;

  double v1=0.0;//
  double v1estimation=0.0;//

  public double getP1Call(double alpha,
                      double sigmastar,
                      double ksi,
```

```
                        double rho,
                        int numberOfSubintervals,
                        int numberOfIterations,
                        int timeToMaturity,
                        double sigma,
                        double stockprice,
                        double interestrate,
                        double strikeprice){

                           double deltati=(double)
timeToMaturity/(360*numberOfSubintervals);
                           double[] vi1=new double[numberOfSubintervals+1];
                           vi1[0]=Math.pow(sigma/100,2);
                           double[] price1=new
double[numberOfSubintervals+1];
                           double pone=0.0;
                           for (int sim=1;sim<numberOfIterations+1;sim++){
                                price1[0]=stockprice;
                                double onePathPrice=0;
                                double onePathPrice1=0;//
                                for (int d=1;d<numberOfSubintervals+1;d++){
                                     price1[d] =price1[d-
1]*Math.exp(((interestrate*0.01)- (vi1[d-1]/2))*deltati+

generator1.nextGaussian()*Math.sqrt(vi1[d-1]*deltati));

                                     vi1[d]=vi1[d-
1]*Math.exp(((alpha*(sigmastar-Math.sqrt(vi1[d-1]))-

(Math.pow(ksi,2)/2))*deltati)+

rho*generator1.nextGaussian()*ksi*Math.sqrt(deltati)+

Math.sqrt(deltati*(1-rho*rho))*

generator2.nextGaussian()*ksi);

onePathPrice=  price1[numberOfSubintervals-1];
onePathPrice1=  vi1[numberOfSubintervals-1];
                                }

 p1=Math.exp(-(interestrate/100)*((double)timeToMaturity/360))*
                                   Math.max(onePathPrice-strikeprice,0);

pone=pone +p1;
v1=v1+onePathPrice1;

   }

p1estimation  =((double) pone/numberOfIterations);
v1estimation  =((double) v1/numberOfIterations);
return   p1estimation;

  }

public double getP1Put(double alpha,
                        double sigmastar,
                        double ksi,
                        double rho,
                        int numberOfSubintervals,
                        int numberOfIterations,
```

```
                        int timeToMaturity,
                        double sigma,
                        double stockprice,
                        double interestrate,
                        double strikeprice){

                           double deltati=(double)
timeToMaturity/(360*numberOfSubintervals);
                           double[] vi1=new double[numberOfSubintervals+1];
                           vi1[0]=Math.pow(sigma/100,2);
                           double[] price1=new
double[numberOfSubintervals+1];
                           double pone=0.0;
                           for (int sim=1;sim<numberOfIterations+1;sim++){
                                price1[0]=stockprice;
                                double onePathPrice=0;
                                for (int d=1;d<numberOfSubintervals+1;d++){
                                     price1[d] =price1[d-
1]*Math.exp(((interestrate*0.01)- (vi1[d-1]/2))*deltati+

generator1.nextGaussian()*Math.sqrt(vi1[d-1]*deltati));

                                     vi1[d] =vi1[d-
1]*Math.exp(((alpha*(sigmastar-Math.sqrt(vi1[d-1]))-

(Math.pow(ksi,2)/2))*deltati)+

rho*generator1.nextGaussian()*ksi*Math.sqrt(deltati)+

Math.sqrt(deltati*(1-rho*rho))* generator2.nextGaussian()*ksi);

onePathPrice=  price1[numberOfSubintervals-1];
       }

p1=Math.exp(-(interestrate/100)*((double)timeToMaturity/360))*
                                  Math.max(strikeprice-onePathPrice,0);
pone=pone +p1;

   }

p1estimation  =((double) pone/numberOfIterations);
return   p1estimation;

  }


  public double getP2Call(double alpha,
                        double sigmastar,
                        double ksi,
                        double rho,
                        int numberOfSubintervals,
                        int numberOfIterations,
                        int timeToMaturity,
                        double sigma,
                        double stockprice,
                        double interestrate,
                        double strikeprice){

double deltati=(double) timeToMaturity/(360*numberOfSubintervals);
double[] vi2=new double[numberOfSubintervals+1];
vi2[0]=Math.pow(sigma/100,2);
```

```
double[] price2=new double[numberOfSubintervals+1];
double ptwo=0.0;
   for (int sim=1;sim<numberOfIterations+1;sim++){
                           price2[0]=stockprice;
                           double onePathPrice=0;
          for (int d=1;d<numberOfSubintervals+1;d++){
                price2[d] =price2[d-1]*Math.exp(((interestrate*0.01)-
               (vi2[d-1]/2))*deltati-
               generator1.nextGaussian()*Math.sqrt(vi2[d-1]*deltati));

         vi2[d]=vi2[d-1]*Math.exp(((alpha*(sigmastar-Math.sqrt(vi2[d-1]))-
                (Math.pow(ksi,2)/2))*deltati)-
               rho*generator1.nextGaussian()*ksi*Math.sqrt(deltati)+
               Math.sqrt(deltati*(1-rho*rho))*
                generator2.nextGaussian()*ksi);

onePathPrice=  price2[numberOfSubintervals-1];
                           }

p2=Math.exp(-(interestrate/100)*((double)timeToMaturity/360))*
        Math.max(onePathPrice-strikeprice,0);
        ptwo=ptwo +p2;
          }
p2estimation  =((double) ptwo/numberOfIterations);
return   p2estimation;

  }

   public double getP2Put(double alpha,
                      double sigmastar,
                      double ksi,
                      double rho,
                      int numberOfSubintervals,
                      int numberOfIterations,
                      int timeToMaturity,
                      double sigma,
                      double stockprice,
                      double interestrate,
                      double strikeprice){

double deltati=(double) timeToMaturity/(360*numberOfSubintervals);
                        double[] vi2=new double[numberOfSubintervals+1];
                        vi2[0]=Math.pow(sigma/100,2);
                        double[] price2=new double[numberOfSubintervals+1];
                        double ptwo=0.0;
                        for (int sim=1;sim<numberOfIterations+1;sim++){
price2[0]=stockprice;
double onePathPrice=0;
for (int d=1;d<numberOfSubintervals+1;d++){
      price2[d] =price2[d-1]*Math.exp(((interestrate*0.01)- (vi2[d-
      1]/2))*deltati-
      generator1.nextGaussian()*Math.sqrt(vi2[d-1]*deltati));

vi2[d]=vi2[d-1]*Math.exp(((alpha*(sigmastar-Math.sqrt(vi2[d-1]))-
        (Math.pow(ksi,2)/2))*deltati)-
       rho*generator1.nextGaussian()*ksi*Math.sqrt(deltati)+
        Math.sqrt(deltati*(1-rho*rho))*
         generator2.nextGaussian()*ksi);

onePathPrice=  price2[numberOfSubintervals-1];
          }
```

```
p2=Math.exp(-(interestrate/100)*((double)timeToMaturity/360))*
           Math.max(strikeprice-onePathPrice,0);
 ptwo=ptwo +p2;
        }
p2estimation  =((double) ptwo/numberOfIterations);
return   p2estimation;

  }

public double getP3Call(double alpha,
                        double sigmastar,
                        double ksi,
                        double rho,
                        int numberOfSubintervals,
                        int numberOfIterations,
                        int timeToMaturity,
                        double sigma,
                        double stockprice,
                        double interestrate,
                        double strikeprice){

double deltati=(double) timeToMaturity/(360*numberOfSubintervals);
double[] vi3 = new double[numberOfSubintervals+1];
vi3[0] = Math.pow(sigma/100,2);
double[] price3=new double[numberOfSubintervals+1];
double pthree=0.0;
    for (int sim=1;sim<numberOfIterations+1;sim++){
       price3[0]=stockprice;
       double onePathPrice=0;
            for (int d=1;d<numberOfSubintervals+1;d++){
                 price3[d] =price3[d-1]*Math.exp(((interestrate*0.01)-
                           (vi3[d-1]/2))*deltati+

generator1.nextGaussian()*Math.sqrt(vi3[d-1]*deltati));

vi3[d]=vi3[d-1]*Math.exp(((alpha*(sigmastar-Math.sqrt(vi3[d-1]))-
            (Math.pow(ksi,2)/2))*deltati)+
            rho*generator1.nextGaussian()*ksi*Math.sqrt(deltati)+
            Math.sqrt(deltati*(1-rho*rho))*
            (-generator2.nextGaussian())*ksi);

onePathPrice= price3[numberOfSubintervals-1];
            }
p3=Math.exp(-(interestrate/100)*((double)timeToMaturity/360))*
Math.max(onePathPrice-strikeprice,0);
pthree=pthree +p3;
        }
p3estimation  =((double) pthree/numberOfIterations);
return   p3estimation;

  }
    public double getP3Put(double alpha,
                        double sigmastar,
                        double ksi,
                        double rho,
                        int numberOfSubintervals,
                        int numberOfIterations,
                        int timeToMaturity,
                        double sigma,
                        double stockprice,
```

```
                    double interestrate,
                    double strikeprice){

double deltati=(double) timeToMaturity/(360*numberOfSubintervals);
double[] vi3 = new double[numberOfSubintervals+1];
vi3[0] = Math.pow(sigma/100,2);
double[] price3=new double[numberOfSubintervals+1];
double pthree=0.0;
     for (int sim=1;sim<numberOfIterations+1;sim++){
         price3[0]=stockprice;
         double onePathPrice=0;
             for (int d=1;d<numberOfSubintervals+1;d++){
                                  price3[d] =price3[d-
1]*Math.exp(((interestrate*0.01)- (vi3[d-1]/2))*deltati+
generator1.nextGaussian()*Math.sqrt(vi3[d-1]*deltati));
vi3[d]=vi3[d-1]*Math.exp(((alpha*(sigmastar-Math.sqrt(vi3[d-1]))-
(Math.pow(ksi,2)/2))*deltati)+
rho*generator1.nextGaussian()*ksi*Math.sqrt(deltati)+
Math.sqrt(deltati*(1-rho*rho))* (-generator2.nextGaussian())*ksi);

onePathPrice=  price3[numberOfSubintervals-1];
         }
p3=Math.exp(-(interestrate/100)*((double)timeToMaturity/360))*
       Math.max(strikeprice-onePathPrice,0);
       pthree=pthree +p3;
                        }
p3estimation  =((double) pthree/numberOfIterations);

return   p3estimation;

  }

  public double getP4Call(double alpha,
                    double sigmastar,
                    double ksi,
                    double rho,
                    int numberOfSubintervals,
                    int numberOfIterations,
                    int timeToMaturity,
                    double sigma,
                    double stockprice,
                    double interestrate,
                    double strikeprice){

double deltati=(double) timeToMaturity/(360*numberOfSubintervals);
double[] vi4=new double[numberOfSubintervals+1];
vi4[0]=Math.pow(sigma/100,2);
double[] price4=new double[numberOfSubintervals+1];
double pfour=0.0;
      for (int sim=1;sim<numberOfIterations+1;sim++){
          price4[0]=stockprice;
          double onePathPrice=0;
               for (int d=1;d<numberOfSubintervals+1;d++){
                                 price4[d] =price4[d-
1]*Math.exp(((interestrate*0.01)- (vi4[d-1]/2))*deltati-
generator1.nextGaussian()*Math.sqrt(vi4[d-1]*deltati));
vi4[d]=vi4[d-1]*Math.exp(((alpha*(sigmastar-Math.sqrt(vi4[d-1]))-
        (Math.pow(ksi,2)/2))*deltati)-
         rho*generator1.nextGaussian()*ksi*Math.sqrt(deltati)+
         Math.sqrt(deltati*(1-rho*rho))*(- generator2.nextGaussian())*ksi);
```

```
onePathPrice= price4[numberOfSubintervals-1];
                  }
p4=Math.exp(-(interestrate/100)*((double)timeToMaturity/360))*
      Math.max(onePathPrice-strikeprice,0);
pfour=pfour +p4;
                         }
p4estimation  =((double) pfour/numberOfIterations);
return   p4estimation;

  }

   public double getP4Put(double alpha,
                       double sigmastar,
                       double ksi,
                       double rho,
                       int numberOfSubintervals,
                       int numberOfIterations,
                       int timeToMaturity,
                       double sigma,
                       double stockprice,
                       double interestrate,
                       double strikeprice){

double deltati=(double) timeToMaturity/(360*numberOfSubintervals);
double[] vi4=new double[numberOfSubintervals+1];
vi4[0]=Math.pow(sigma/100,2);
double[] price4=new double[numberOfSubintervals+1];
double pfour=0.0;
          for (int sim=1;sim<numberOfIterations+1;sim++){
               price4[0]=stockprice;
               double onePathPrice=0;
                   for (int d=1;d<numberOfSubintervals+1;d++){
                     price4[d] =price4[d-1]*Math.exp(((interestrate*0.01)-
(vi4[d-1]/2))*deltati-
generator1.nextGaussian()*Math.sqrt(vi4[d-1]*deltati));
vi4[d] =vi4[d-1]*Math.exp(((alpha*(sigmastar-Math.sqrt(vi4[d-1]))-
(Math.pow(ksi,2)/2))*deltati)-
rho*generator1.nextGaussian()*ksi*Math.sqrt(deltati)+
Math.sqrt(deltati*(1-rho*rho))*(- generator2.nextGaussian())*ksi);

onePathPrice= price4[numberOfSubintervals-1];
                                  }
p4=Math.exp(-(interestrate/100)*((double)timeToMaturity/360))*
Math.max(strikeprice-onePathPrice,0);
pfour=pfour +p4;
                         }
p4estimation  =((double) pfour/numberOfIterations);
return   p4estimation;

  }

  public double getQ1Call(
                       int numberOfSubintervals,
                       int numberOfIterations,
                       int timeToMaturity,
                       double sigma,
                       double stockprice,
                       double interestrate,
                       double strikeprice){

double deltati=(double) timeToMaturity/(360*numberOfSubintervals);
```

```
double vol=Math.pow(sigma/100,2);
double[] priceq1=new double[numberOfSubintervals+1];
                        double qone=0.0;
                        for (int sim=1;sim<numberOfIterations+1;sim++){
                             priceq1[0]=stockprice;
                             double onePathPrice=0;
                             for (int d=1;d<numberOfSubintervals+1;d++){
                                  priceq1[d] =priceq1[d-
1]*Math.exp(((interestrate*0.01)- (vol/2))*deltati+
generator1.nextGaussian()*Math.sqrt(vol*deltati));
onePathPrice= priceq1[numberOfSubintervals-1];
                                  }
q1=Math.exp(-(interestrate/100)*((double)timeToMaturity/360))*
    Math.max(onePathPrice-strikeprice,0);
qone=qone +q1;
   }
q1estimation  =((double) qone/numberOfIterations);
return   q1estimation;
  }

   public double getQ1Put(
                       int numberOfSubintervals,
                       int numberOfIterations,
                       int timeToMaturity,
                       double sigma,
                       double stockprice,
                       double interestrate,
                       double strikeprice){

double deltati=(double) timeToMaturity/(360*numberOfSubintervals);
double vol=Math.pow(sigma/100,2);
double[] priceq1=new double[numberOfSubintervals+1];
                        double qone=0.0;
                        for (int sim=1;sim<numberOfIterations+1;sim++){
                             priceq1[0]=stockprice;
                             double onePathPrice=0;
                             for (int d=1;d<numberOfSubintervals+1;d++){
                                  priceq1[d] =priceq1[d-
1]*Math.exp(((interestrate*0.01)- (vol/2))*deltati+
generator1.nextGaussian()*Math.sqrt(vol*deltati));
onePathPrice=  priceq1[numberOfSubintervals-1];
                                  }
q1=Math.exp(-(interestrate/100)*((double)timeToMaturity/360))*
Math.max(strikeprice-onePathPrice,0);
qone=qone +q1;
                        }
q1estimation  =((double) qone/numberOfIterations);
return   q1estimation;
  }

  public double getQ2Call(
                       int numberOfSubintervals,
                       int numberOfIterations,
                       int timeToMaturity,
                       double sigma,
                       double stockprice,
                       double interestrate,
                       double strikeprice){

double deltati=(double) timeToMaturity/(360*numberOfSubintervals);
double vol=Math.pow(sigma/100,2);
```

```
double[] priceq2=new double[numberOfSubintervals+1];
double qtwo=0.0;
                        for (int sim=1;sim<numberOfIterations+1;sim++){
                              priceq2[0]=stockprice;
                              double onePathPrice=0;
                              for (int d=1;d<numberOfSubintervals+1;d++){
                                   priceq2[d] =priceq2[d-
1]*Math.exp(((interestrate*0.01)- (vol/2))*deltati-
generator1.nextGaussian()*Math.sqrt(vol*deltati));
onePathPrice= priceq2[numberOfSubintervals-1];
                              }
q2=Math.exp(-(interestrate/100)*((double)timeToMaturity/360))*
   Math.max(onePathPrice-strikeprice,0);
qtwo=qtwo +q2;
   }
q2estimation  =((double) qtwo/numberOfIterations);
return   q2estimation;

  }

  public double getQ2Put(
                       int numberOfSubintervals,
                       int numberOfIterations,
                       int timeToMaturity,
                       double sigma,
                       double stockprice,
                       double interestrate,
                       double strikeprice){

double deltati=(double) timeToMaturity/(360*numberOfSubintervals);
double vol=Math.pow(sigma/100,2);
double[] priceq2=new double[numberOfSubintervals+1];
                        double qtwo=0.0;
                        for (int sim=1;sim<numberOfIterations+1;sim++){
                              priceq2[0]=stockprice;
                              double onePathPrice=0;
                              for (int d=1;d<numberOfSubintervals+1;d++){
priceq2[d]=priceq2[d-1]*Math.exp(((interestrate*0.01)- (vol/2))*deltati-
generator1.nextGaussian()*Math.sqrt(vol*deltati));
onePathPrice= priceq2[numberOfSubintervals-1];
                              }
q2=Math.exp(-(interestrate/100)*((double)timeToMaturity/360))*
Math.max(strikeprice-onePathPrice,0);
qtwo=qtwo +q2;
}
q2estimation  =((double) qtwo/numberOfIterations);
return   q2estimation;

  }

}
```

Part G – Thesis (main file)

```
/**
 * Stochastic Volatility Models in Option Pricing
 *
 * Copyright (c) 2007 Mälardalen University
 * Högskoleplan Box 883, 721 23 Västerås, Sweden.
 * All Rights Reserved.
 *
 * The copyright to the computer program(s) herein
 * is the property of Mälardalen University.
 * The program(s) may be used and/or copied only with
 * the written permission of Mälardalen University
 * or in accordance with the terms and conditions
 * stipulated in the agreement/contract under which
 * the program(s) have been supplied.
 *
 * Description: Option price calculator
 * @version 1.0  Dec 17th 2007
 * @authors: Michail Kalavrezos, Michael Wennermo
 * Email: michail_kalavrezos@yahoo.se, mikewennermo@yahoo.se
 */


import java.awt.*;
import java.text.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.table.*;
import javax.swing.border.*;

import org.jfree.chart.*;
import org.jfree.chart.axis.*;
import org.jfree.chart.plot.*;
import org.jfree.data.xy.*;
import org.jfree.data.statistics.*;


public class Thesis extends JApplet implements FocusListener,ActionListener

{
   // class variables

   // panels
   private JPanel mainPanel = null;
   private JPanel inputPanel = null;
   private JPanel stochasticPanel = null;

   private JPanel bsPanel = null;
   private JPanel bsstochPanel = null;
   private JPanel svstochdataPanel = null;
   private JPanel powerPanel = null;
```

```
private JPanel outputPanel = null;
private JPanel output1Panel = null;

private JPanel buttonPanel = null;
private JPanel button1Panel = null;

private JPanel dataPanel = null;
private JPanel data1Panel = null;
private JPanel data2Panel = null;
private JPanel data3Panel = null;
private JPanel data4Panel = null;

private JPanel button2Panel = null;

private JPanel button3Panel = null;
private JPanel button4Panel = null;

// private ChartPanel graphPanel = null;

// button groups
private ButtonGroup fileGroup = null;
private ButtonGroup methodGroup = null;
private ButtonGroup graphGroup = null;
private ButtonGroup typeGroup = null;
private ButtonGroup normalDistributionGroup = null;

// radio buttons
private JRadioButton callButton = null;
private JRadioButton putButton = null;
private JRadioButton normalDistributionAButton = null;
private JRadioButton normalDistributionBButton = null;

// buttons
private JButton calculateButton = null;
private JButton calculate1Button = null;
private JButton calculate2Button = null;
private JButton calculate3Button = null;
private JButton calculate4Button = null;
private JButton resetButton = null;

// string constants
private final String CALL = "Call";
private final String PUT = "Put";
private final String normalDistributionA = " CND A";
private final String normalDistributionB = "CND B";

private final String CALCULATE = "Calculate All";
private final String CALCULATE1 = "Calc. BS";
private final String CALCULATE2 = "Calc. SV";
private final String CALCULATE3 = "Calc. SSV";
private final String CALCULATE4 = "Calc. PS";

private final String RESET = "Reset";
private final String INPUT = "Input";
private final String OUTPUT = "Output";
private final String ERROR = "Error";


// texts of labels
```

```
private final String SIMULATIONS1_LABEL = " Simulations1";
private JTextField simulations1Field = null;

private final String SIMULATIONS2_LABEL = " Simulations2";
private JTextField simulations2Field = null;

private final String RHO_LABEL = "Rho";
private JTextField rhoField = null;

//THE EXTRA

private final String ALPHAONE_LABEL = "Alpha One";
private JTextField alphaoneField = null;

private final String ALPHATWO_LABEL = "Alpha Two";
private JTextField alphatwoField = null;

private final String ALPHATHREE_LABEL = "Alpha Three";
private JTextField alphathreeField = null;

private final String SIGMASTAR1_LABEL = "Sigma Star1";
private JTextField sigmastar1Field = null;

private final String SIGMASTAR2_LABEL = "Sigma Star2";
private JTextField sigmastar2Field = null;


private final String KSI1_LABEL = " Ksi1";
private JTextField ksi1Field = null;

private final String KSI2_LABEL = " Ksi2";
private JTextField ksi2Field = null;

private final String KSI3_LABEL = " Ksi3";
private JTextField ksi3Field = null;

private final String STOCKPRICE_LABEL = "Stock price";
private JTextField stockpriceField = null;

private final String STRIKEPRICE_LABEL = "Strike price";
private JTextField strikepriceField = null;

private final String VOLATILITY_LABEL = "Volatility(%)";
private JTextField volatilityField = null;

private final String TIMETOMATURITY_LABEL = "Maturity (days)";
private JTextField timetomaturityField = null;

private final String INTERESTRATE_LABEL = "Interest(%)";
private JTextField interestrateField = null;

private final String INTERVALS1_LABEL = "Intervals1";
private JTextField intervals1Field = null;

private final String INTERVALS2_LABEL = "Intervals2";
private JTextField intervals2Field = null;


private final String BSPRICE_LABEL = "BS Price";
private JTextField bspriceField = null;
```

```
    private final String SVPRICE_LABEL = "SV Price";
    private JTextField svpriceField = null;

    private final String SSVPRICE_LABEL = "SSV Price";
    private JTextField ssvpriceField = null;

    private final String PSPRICE_LABEL = "PS Price";
    private JTextField pspriceField = null;


    private final String TYPEOFOPTION_LABEL = "Type of Option";
    private final String NORMALDISTRIBUTION_LABEL = "
    Cumulative Normal Distribution";


    // Tooltips
    private final String SIMULATIONS_TOOLTIP ="The Recommended value is
    >20000";
    private final String STOCKPRICE_TOOLTIP = "Current price of the
    underlying";
    private final String STRIKEPRICE_TOOLTIP = "Exercise  price";
    private final String VOLATILITY_TOOLTIP = "Insert the volatility";
    private final String TIMETOMATURITY_TOOLTIP = "Time expressed in days";
    private final String INTERESTRATE_TOOLTIP = "Interest rate expressed in
    %";
    private final String ALPHAONE_TOOLTIP = "Estimated or observed value of
    alpha one";
    private final String ALPHATWO_TOOLTIP = "Estimated or observed value of
    alpha two";
    private final String BSPRICE_TOOLTIP = "DO NOT INSERT ANY VALUE HERE";
    private final String SVPRICE_TOOLTIP = "DO NOT INSERT ANY VALUE HERE";
    private final String SSVPRICE_TOOLTIP = "DO NOT INSERT ANY VALUE HERE";
    private final String PSPRICE_TOOLTIP = "DO NOT INSERT ANY VALUE HERE";
    private final String INTERVALS1_TOOLTIP = "Integer number of time
    intervals ";
    private final String INTERVALS2_TOOLTIP = "Integer number of time
    intervals ";
    private final String SIMULATIONS1_TOOLTIP = "Integer number of
    simulations ";
    private final String SIMULATIONS2_TOOLTIP = "Integer number of
    simulations ";
    private final String RHO_TOOLTIP = "Estimated or observed value of rho";
    private final String SIGMASTAR1_TOOLTIP = "Integer number of time
    intervals ";
    private final String SIGMASTAR2_TOOLTIP = "Integer number of time
    intervals ";
    private final String KSI1_TOOLTIP = "Integer number of time intervals ";
    private final String KSI2_TOOLTIP = "Integer number of time intervals ";
    private final String KSI3_TOOLTIP = "Integer number of time intervals ";

//Error messages
private final String NOT_A_NUMBER = " Enter a number";
private final String NOT_INTEGER = " Enter an integer number";
private final String NOT_DOUBLE = " Enter a double number";
private final String NOT_POSITIVE = "Enter a positive number";
private final String NOT_ACCEPTED = "Enter a number between -1 AND 1";
private final String CAUTION  = "CAUTION";

    // numerical constants

    private static double STOCKPRICE = 100.0;
```

```
private static double STRIKEPRICE = 100.0;
private static double VOLATILITY = 40;
private static int TIMETOMATURITY = 180;
private static double INTERESTRATE =4.25;

private static double ALPHAONE = 1;
private static double ALPHATWO = 1;

private static double KSI1 = 0.20;
private static double KSI2 = 0.20;
private static double KSI3 = 0.20;

private static double RHO = 0.12;

private static double SIGMASTAR1 = 0.40;
private static double SIGMASTAR2 = 0.40;

private static int INTERVALS1=50;
private static int INTERVALS2=50;

private static int SIMULATIONS1=20000;
private static int SIMULATIONS2=20000;

private static double BSPRICE = 0.0;
private static double SVPRICE = 0.0;
private static double SSVPRICE = 0.0;
private static double PSPRICE = 0.0;

// numerical variables

private static double stockprice = STOCKPRICE;
private static double strikeprice = STRIKEPRICE;
private static double volatility = VOLATILITY;
private static int timetomaturity = TIMETOMATURITY;
private double interestrate = INTERESTRATE;

private double alphaone = ALPHAONE;
private double alphatwo = ALPHATWO;

private double sigmastar1=SIGMASTAR1;
private double sigmastar2=SIGMASTAR2;

private static int simulations1 = SIMULATIONS1;
private static int simulations2 = SIMULATIONS2;

private static int intervals1 = INTERVALS1;
private static int intervals2 = INTERVALS2;

private double rho=RHO;

private double ksi1= KSI1;
private double ksi2= KSI2;
private double ksi3= KSI3;

private static double bsprice = BSPRICE;
private static double svprice = SVPRICE;
private static double ssvprice = SSVPRICE;
private static double psprice = PSPRICE;


// number formatters
```

```
private DecimalFormat numberFormatter = null;
private DecimalFormat numberFormatter1 = null;

// class methods

// initialising
public void init () {

   // Initialise formatter
   DecimalFormatSymbols symbols = new DecimalFormatSymbols();
   symbols.setDecimalSeparator('.');
   numberFormatter = new DecimalFormat("##.######",symbols);
   numberFormatter1 = new DecimalFormat("##.###",symbols);

   // get content pane
   Container contentPane = getContentPane();

   // create main panel
   mainPanel =new JPanel(new BorderLayout());

   // set box layout
   //mainPanel.setBorder(new TitledBorder(CAPPRICING));

   //mainPanel.setLayout(new GridLayout(0,1));
   mainPanel.setLayout(new BoxLayout(mainPanel,BoxLayout.Y_AXIS));

   // add main panel to content pane
   contentPane.add(mainPanel);

   // create input panel
   inputPanel = new JPanel(new BorderLayout());
   inputPanel.setPreferredSize(new Dimension(550,250));
   inputPanel.setBorder(new TitledBorder(INPUT));

   // add it to the main panel
   mainPanel.add(inputPanel);

   button2Panel = new JPanel();
   button2Panel.setLayout(new BoxLayout(button2Panel,BoxLayout.X_AXIS));
   button2Panel.setPreferredSize(new Dimension(550,20));
   inputPanel.add(button2Panel,BorderLayout.PAGE_START);

   button3Panel = new JPanel();
   button3Panel.setLayout(new BoxLayout(button3Panel,BoxLayout.X_AXIS));
   button3Panel.setPreferredSize(new Dimension(220,20));
   button2Panel.add(button3Panel);

   JLabel label= new JLabel(TYPEOFOPTION_LABEL);
   //forwardratefileField = new JTextField();
   button3Panel.add(label);

   // create button group
   typeGroup = new ButtonGroup();

   // create  oneday button
   callButton = new JRadioButton(CALL);
   typeGroup.add(callButton);

   // add action listener
   callButton.addActionListener(this);
   callButton.setSelected(true);
```

```
// add it to button panel
button3Panel.add(callButton);



// create  put button
putButton = new JRadioButton(PUT);
typeGroup.add(putButton);
// add action listener
putButton.addActionListener(this);
// add it to button panel
button3Panel.add(putButton);

label= new JLabel(NORMALDISTRIBUTION_LABEL);
button3Panel.add(label);


normalDistributionGroup= new ButtonGroup();
normalDistributionAButton = new JRadioButton(normalDistributionA );
normalDistributionGroup.add(normalDistributionAButton);
// add action listener
normalDistributionAButton.addActionListener(this);
// add it to button panel
button3Panel.add(normalDistributionAButton);

normalDistributionBButton = new JRadioButton(normalDistributionB );
normalDistributionGroup.add(normalDistributionBButton);
// add action listener
normalDistributionBButton.addActionListener(this);
normalDistributionBButton.setSelected(true);
// add it to button panel
button3Panel.add(normalDistributionBButton);


// create button 1 panel
button1Panel = new JPanel();
// add it to the input panel
inputPanel.add(button1Panel,BorderLayout.PAGE_END);
//button2Panel.add(button1Panel);

// create button group
methodGroup = new ButtonGroup();

// create calculate button
calculate1Button = new JButton(CALCULATE1);
// add action listener
calculate1Button.addActionListener(this);
// add it to button panel
button1Panel.add(calculate1Button);

// create calculate button
calculate2Button = new JButton(CALCULATE2);
// add action listener
calculate2Button.addActionListener(this);
// add it to button panel
button1Panel.add(calculate2Button);

// create calculate button
calculate3Button = new JButton(CALCULATE3);
// add action listener
calculate3Button.addActionListener(this);
```

```
// add it to button panel
button1Panel.add(calculate3Button);

// create calculate button
calculate4Button = new JButton(CALCULATE4);
// add action listener
calculate4Button.addActionListener(this);
// add it to button panel
button1Panel.add(calculate4Button);

// create calculate button
calculateButton = new JButton(CALCULATE);
// add action listener
calculateButton.addActionListener(this);
// add it to button panel
button1Panel.add(calculateButton);

//create reset button
resetButton = new JButton(RESET);
//add action listener
resetButton.addActionListener(this);
// add it to button panel
button1Panel.add(resetButton);

// create bs panel
bsPanel = new JPanel(new BorderLayout());
bsPanel.setPreferredSize(new Dimension(150,100));
bsPanel.setBorder(new TitledBorder("BS model"));

// add it to the main panel
inputPanel.add(bsPanel,BorderLayout.LINE_START);

// create data panel for the inputs
dataPanel = new JPanel(new GridLayout(0,2));
dataPanel.setPreferredSize(new Dimension(140,100));
// add it to input panel
bsPanel.add(dataPanel,BorderLayout.LINE_START);

// create labels,create text field,add focus listener and then add
// the labels and text field to data panel

label = new JLabel(STOCKPRICE_LABEL);
stockpriceField = new JTextField();
dataPanel.add(label);
stockpriceField.addFocusListener(this);
dataPanel.add(stockpriceField);

label = new JLabel(STRIKEPRICE_LABEL);
strikepriceField = new JTextField();
dataPanel.add(label);
strikepriceField.addFocusListener(this);
dataPanel.add(strikepriceField);

label= new JLabel(VOLATILITY_LABEL);
volatilityField = new JTextField();
dataPanel.add(label);
volatilityField.addFocusListener(this);
dataPanel.add(volatilityField);

label= new JLabel(TIMETOMATURITY_LABEL);
timetomaturityField = new JTextField();
```

```
dataPanel.add(label);
timetomaturityField.addFocusListener(this);
dataPanel.add(timetomaturityField);

label = new JLabel(INTERESTRATE_LABEL);
interestrateField = new JTextField();
dataPanel.add(label);
interestrateField.addFocusListener(this);
dataPanel.add(interestrateField);


// create stochastic  panel
stochasticPanel =new JPanel(new BorderLayout());

stochasticPanel.setLayout(new GridLayout(1,0));
// add main panel to content pane
inputPanel.add(stochasticPanel,BorderLayout.LINE_END);

bsstochPanel = new JPanel(new BorderLayout());
bsstochPanel.setPreferredSize(new Dimension(100,100));
bsstochPanel.setBorder(new TitledBorder("SV model"));

// add it to the main panel
stochasticPanel.add(bsstochPanel,BorderLayout.LINE_START);

data2Panel = new JPanel(new GridLayout(0,2));
data2Panel.setPreferredSize(new Dimension(100,100));
// add it to input panel
bsstochPanel.add(data2Panel,BorderLayout.CENTER);

label = new JLabel(ALPHAONE_LABEL);
alphaoneField = new JTextField();
data2Panel.add(label);
alphaoneField.addFocusListener(this);
data2Panel.add(alphaoneField);

label = new JLabel(SIGMASTAR1_LABEL);
sigmastar1Field = new JTextField();
data2Panel.add(label);
sigmastar1Field.addFocusListener(this);
data2Panel.add(sigmastar1Field);

label = new JLabel(KSI1_LABEL);
ksi1Field = new JTextField();
data2Panel.add(label);
ksi1Field.addFocusListener(this);
data2Panel.add(ksi1Field);

label = new JLabel(INTERVALS1_LABEL);
intervals1Field = new JTextField();
data2Panel.add(label);
intervals1Field.addFocusListener(this);
data2Panel.add(intervals1Field);

 label = new JLabel(SIMULATIONS1_LABEL);
simulations1Field = new JTextField();
data2Panel.add(label);
simulations1Field.addFocusListener(this);
data2Panel.add(simulations1Field);

svstochdataPanel = new JPanel(new BorderLayout());
```

```
svstochdataPanel.setPreferredSize(new Dimension(100,100));
svstochdataPanel.setBorder(new TitledBorder("SSV model"));

// add it to the main panel
stochasticPanel.add(svstochdataPanel,BorderLayout.CENTER);

data3Panel = new JPanel(new GridLayout(0,2));
data3Panel.setPreferredSize(new Dimension(100,100));
// add it to input panel
svstochdataPanel.add(data3Panel,BorderLayout.CENTER);


label = new JLabel(ALPHATWO_LABEL);
alphatwoField = new JTextField();
data3Panel.add(label);
alphatwoField.addFocusListener(this);
data3Panel.add(alphatwoField);

label = new JLabel(SIGMASTAR2_LABEL);
sigmastar2Field = new JTextField();
data3Panel.add(label);
sigmastar2Field.addFocusListener(this);
data3Panel.add(sigmastar2Field);

label = new JLabel(KSI2_LABEL);
ksi2Field = new JTextField();
data3Panel.add(label);
ksi2Field.addFocusListener(this);
data3Panel.add(ksi2Field);

label = new JLabel(RHO_LABEL);
rhoField = new JTextField();
data3Panel.add(label);
rhoField.addFocusListener(this);
data3Panel.add(rhoField);

label = new JLabel(INTERVALS2_LABEL);
intervals2Field = new JTextField();
data3Panel.add(label);
intervals2Field.addFocusListener(this);
data3Panel.add(intervals2Field);

 label = new JLabel(SIMULATIONS2_LABEL);
simulations2Field = new JTextField();
data3Panel.add(label);
simulations2Field.addFocusListener(this);
data3Panel.add(simulations2Field);

//NEW

powerPanel = new JPanel(new BorderLayout());
powerPanel.setPreferredSize(new Dimension(140,100));
powerPanel.setBorder(new TitledBorder("PS model"));

// add it to the main panel
stochasticPanel.add(powerPanel,BorderLayout.LINE_END);


data4Panel = new JPanel(new GridLayout(0,2));
data4Panel.setPreferredSize(new Dimension(80,20));
// add it to input panel
```

```
powerPanel.add(data4Panel,BorderLayout.NORTH);


// create labels,create text field,add focus listener and then add
// the labels and text field to data panel

label = new JLabel(KSI3_LABEL);
ksi3Field = new JTextField();
data4Panel.add(label);
ksi3Field.addFocusListener(this);
data4Panel.add(ksi3Field);

//create data panel
data1Panel = new JPanel(new BorderLayout());
data1Panel.setPreferredSize(new Dimension(150,100));
// add it to input panel
inputPanel.add(data1Panel,BorderLayout.EAST);

// create output panel
outputPanel = new JPanel();
outputPanel.setBorder(new TitledBorder(OUTPUT));
outputPanel.setPreferredSize(new Dimension(550,100));

// add it to the main panel
mainPanel.add(outputPanel);

// create output1 panel
output1Panel = new JPanel();
output1Panel.setLayout(new BoxLayout(output1Panel,BoxLayout.X_AXIS));
output1Panel.setPreferredSize(new Dimension(550,20));
// add it to the output panel
outputPanel.add(output1Panel,BorderLayout.NORTH);

// add label and field to output1 panel
label = new JLabel(BSPRICE_LABEL);
bspriceField = new JTextField();
output1Panel.add(label);
output1Panel.add(bspriceField);

label = new JLabel(SVPRICE_LABEL);
svpriceField = new JTextField();
output1Panel.add(label);
output1Panel.add(svpriceField);

label = new JLabel(SSVPRICE_LABEL);
ssvpriceField = new JTextField();
output1Panel.add(label);
output1Panel.add(ssvpriceField);

label = new JLabel(PSPRICE_LABEL);
pspriceField = new JTextField();
output1Panel.add(label);
output1Panel.add(pspriceField);


// add tooltip
stockpriceField.setToolTipText(STOCKPRICE_TOOLTIP);
strikepriceField.setToolTipText(STRIKEPRICE_TOOLTIP);
volatilityField.setToolTipText(VOLATILITY_TOOLTIP);
timetomaturityField.setToolTipText(TIMETOMATURITY_TOOLTIP);
interestrateField.setToolTipText(INTERESTRATE_TOOLTIP);
```

```
        alphaoneField.setToolTipText(ALPHAONE_TOOLTIP);
        alphatwoField.setToolTipText(ALPHATWO_TOOLTIP);

        sigmastar1Field.setToolTipText(SIGMASTAR1_TOOLTIP);
        sigmastar2Field.setToolTipText(SIGMASTAR2_TOOLTIP);

        simulations1Field.setToolTipText(SIMULATIONS1_TOOLTIP);
        simulations2Field.setToolTipText(SIMULATIONS2_TOOLTIP);
        rhoField.setToolTipText(RHO_TOOLTIP);
        intervals1Field.setToolTipText(INTERVALS1_TOOLTIP);
        intervals2Field.setToolTipText(INTERVALS2_TOOLTIP);

        ksi1Field.setToolTipText(KSI1_TOOLTIP);
        ksi2Field.setToolTipText(KSI2_TOOLTIP);
        ksi3Field.setToolTipText(KSI3_TOOLTIP);

        bspriceField.setToolTipText(BSPRICE_TOOLTIP);
        svpriceField.setToolTipText(SVPRICE_TOOLTIP);
        ssvpriceField.setToolTipText(SSVPRICE_TOOLTIP);
        pspriceField.setToolTipText(PSPRICE_TOOLTIP);


        //set value
        stockpriceField.setText(numberFormatter.format(STOCKPRICE));
        strikepriceField.setText(numberFormatter.format(STRIKEPRICE));
        volatilityField.setText(numberFormatter.format(VOLATILITY));

        timetomaturityField.setText(numberFormatter.format(TIMETOMATURITY));
        interestrateField.setText(numberFormatter.format(INTERESTRATE));

        alphaoneField.setText(numberFormatter.format(ALPHAONE));
        alphatwoField.setText(numberFormatter.format(ALPHATWO));

        rhoField.setText(numberFormatter.format(RHO));

        sigmastar1Field.setText(numberFormatter.format(SIGMASTAR1));
        sigmastar2Field.setText(numberFormatter.format(SIGMASTAR2));

        ksi1Field.setText(numberFormatter.format(KSI1));
        ksi2Field.setText(numberFormatter.format(KSI2));
        ksi3Field.setText(numberFormatter.format(KSI3));

        simulations1Field.setText(numberFormatter.format(SIMULATIONS1));
        simulations2Field.setText(numberFormatter.format(SIMULATIONS2));

        intervals1Field.setText(numberFormatter.format(INTERVALS1));
        intervals2Field.setText(numberFormatter.format(INTERVALS2));
        bspriceField.setText(numberFormatter.format(BSPRICE));
        svpriceField.setText(numberFormatter.format(SVPRICE));
        ssvpriceField.setText(numberFormatter.format(SSVPRICE));
        pspriceField.setText(numberFormatter.format(PSPRICE));

    }


// method of Action listener
public void actionPerformed(ActionEvent e){

        //determine,who called action listener
        Object source = e.getSource();
```

```
            if(source == resetButton){

            //reset all TextFields and variables to the initial values

             stockprice = STOCKPRICE;
             stockpriceField.setText(numberFormatter.format(STOCKPRICE));

             strikeprice= STRIKEPRICE;
             strikepriceField.setText(numberFormatter.format(STRIKEPRICE));

             volatility=VOLATILITY;
             volatilityField.setText(numberFormatter.format(VOLATILITY));

             timetomaturity = TIMETOMATURITY;

timetomaturityField.setText(numberFormatter.format(TIMETOMATURITY));

             interestrate = INTERESTRATE;
             interestrateField.setText(numberFormatter.format(INTERESTRATE));

             alphaone= ALPHAONE;
             alphaoneField.setText(numberFormatter.format(ALPHAONE));

             alphatwo= ALPHATWO;
             alphatwoField.setText(numberFormatter.format(ALPHATWO));

             ksi1= KSI1;
             ksi1Field.setText(numberFormatter.format(KSI1));

             ksi2= KSI2;
             ksi2Field.setText(numberFormatter.format(KSI2));

             ksi3= KSI3;
             ksi3Field.setText(numberFormatter.format(KSI3));

             sigmastar1= SIGMASTAR1;
             sigmastar1Field.setText(numberFormatter.format(SIGMASTAR1));

             sigmastar2= SIGMASTAR2;
             sigmastar2Field.setText(numberFormatter.format(SIGMASTAR2));

             rho = RHO;
             rhoField.setText(numberFormatter.format(RHO));

             intervals1=INTERVALS1;
             intervals1Field.setText(numberFormatter.format(INTERVALS1));

             intervals2=INTERVALS2;
             intervals2Field.setText(numberFormatter.format(INTERVALS2));

             simulations1=SIMULATIONS1;
             simulations1Field.setText(numberFormatter.format(SIMULATIONS1));

             simulations2=SIMULATIONS2;
             simulations2Field.setText(numberFormatter.format(SIMULATIONS2));

             bsprice = BSPRICE;
             bspriceField.setText(numberFormatter.format(BSPRICE));

             svprice = SVPRICE;
             svpriceField.setText(numberFormatter.format(SVPRICE));
```

```
            ssvprice = SSVPRICE;
            ssvpriceField.setText(numberFormatter.format(SSVPRICE));

            psprice = PSPRICE;
            pspriceField.setText(numberFormatter.format(PSPRICE));

           }

if (source == calculate1Button) {

    if (callButton.isSelected()){

      if(normalDistributionAButton.isSelected()){

           double bscall = new
BlackScholesFormula().BlackScholesCallA(stockprice,strikeprice,volatility,
                                                  timetomaturity,interestrate);
   bspriceField.setText(numberFormatter.format(bscall));
       }

      else {

         for(stockprice=0.75; stockprice<1.26; stockprice=stockprice+0.01){

         double bscall = new
BlackScholesFormula().BlackScholesCallB(stockprice,strikeprice,volatility,
         timetomaturity,interestrate);
bspriceField.setText(numberFormatter.format(bscall));
// to check -System.out.println(bscall);
          }
       }
      }

   else {

      if(normalDistributionAButton.isSelected()){

            double bsput = new
BlackScholesFormula().BlackScholesPutA(stockprice,strikeprice,volatility,
timetomaturity,interestrate);

bspriceField.setText(numberFormatter.format(bsput));
            }

      else {

double bsput = new
BlackScholesFormula().BlackScholesPutB(stockprice,strikeprice,volatility,
timetomaturity,interestrate);

bspriceField.setText(numberFormatter.format(bsput));
            }

   }
}

if (source == calculate2Button) {

   if (callButton.isSelected()){
```

```
      if(normalDistributionAButton.isSelected()){

      double stvol1=new
StochasticVolatilityFile1().simulateVolatility1(alphaone,ksi1,sigmastar1,
intervals1,simulations1,volatility,timetomaturity);

double svcall1=new BlackScholesFormula().BlackScholesCallA(stockprice,
      strikeprice,stvol1,timetomaturity,interestrate);

double stvol2=new
StochasticVolatilityFile1().simulateVolatility2(alphaone,ksi1,
sigmastar1, intervals1,simulations1,volatility,timetomaturity);

double svcall2=new BlackScholesFormula().BlackScholesCallA(stockprice,
      strikeprice,stvol2,timetomaturity,interestrate);

double svcall=(svcall1+svcall2)/2;
svpriceField.setText(numberFormatter.format(svcall));

      }

      // call with Normal-B
      else {

      for(stockprice=0.75; stockprice<1.26; stockprice=stockprice+0.01){

      double stvol1=new
StochasticVolatilityFile1().simulateVolatility1(alphaone,ksi1,sigmastar1,

intervals1,simulations1,volatility,timetomaturity);

      double svcall1=new
BlackScholesFormula().BlackScholesCallB(stockprice,
      strikeprice, stvol1, timetomaturity, interestrate);

      double stvol2=new
StochasticVolatilityFile1().simulateVolatility2(alphaone,ksi1,sigmastar1,
intervals1,simulations1,volatility,timetomaturity);

      double svcall2=new
BlackScholesFormula().BlackScholesCallB(stockprice, strikeprice, stvol2,
timetomaturity, interestrate);

   double svcall=(svcall1+svcall2)/2;
   svpriceField.setText(numberFormatter.format(svcall));
   // System.out.println(svcall);
   }
      }
   }

   else{

      if(normalDistributionAButton.isSelected()){

double stvol1=new
StochasticVolatilityFile1().simulateVolatility1(alphaone,ksi1,
sigmastar1,intervals1,simulations1,volatility,timetomaturity);

double svput1=new BlackScholesFormula().BlackScholesPutA(stockprice,
strikeprice, stvol1, timetomaturity,  interestrate);
```

```
double stvol2=new
StochasticVolatilityFile1().simulateVolatility2(alphaone,ksi1,
sigmastar1,intervals1,simulations1,volatility,timetomaturity);

double svput2=new BlackScholesFormula().BlackScholesPutA(stockprice,
      strikeprice, stvol2, timetomaturity, interestrate);

double svput=(svput1+svput2)/2;
svpriceField.setText(numberFormatter.format(svput));
      }

      else {

      double stvol1=new
StochasticVolatilityFile1().simulateVolatility1(alphaone,ksi1,
sigmastar1,intervals1,simulations1,volatility,timetomaturity);

      double svput1=new BlackScholesFormula().BlackScholesPutB(stockprice,
      strikeprice,stvol1,timetomaturity, interestrate);

      double stvol2=new
StochasticVolatilityFile1().simulateVolatility2(alphaone,ksi1,
sigmastar1,intervals1,simulations1,volatility,timetomaturity);

      double svput2=new BlackScholesFormula().BlackScholesPutB(stockprice,

strikeprice, stvol2, timetomaturity, interestrate);

      double svput=(svput1+svput2)/2;
      svpriceField.setText(numberFormatter.format(svput));
      }
   }
}


if (source == calculate3Button) {

   if (callButton.isSelected()){

      for(stockprice=0.75; stockprice<1.26; stockprice=stockprice+0.01){

      double ssv1=new StochasticPriceVolatility().getP1Call(alphatwo,
                                                            sigmastar2,
                                                            ksi2,
                                                            rho,
                                                            intervals2,
                                                            simulations2,
                                                            timetomaturity,
                                                            volatility,
                                                            stockprice,
                                                            interestrate,
                                                            strikeprice );
      double ssv2=new StochasticPriceVolatility().getP2Call(alphatwo,
                                                            sigmastar2,
                                                            ksi2,
                                                            rho,
                                                            intervals2,
                                                            simulations2,
                                                            timetomaturity,
                                                            volatility,
                                                            stockprice,
```

```
                                                        interestrate,
                                                        strikeprice );
      double ssv3=new StochasticPriceVolatility().getP3Call(alphatwo,
                                                        sigmastar2,
                                                        ksi2,
                                                        rho,
                                                        intervals2,
                                                        simulations2,
                                                        timetomaturity,
                                                        volatility,
                                                        stockprice,
                                                        interestrate,
                                                        strikeprice );

      double ssv4=new StochasticPriceVolatility().getP4Call(alphatwo,
                                                       sigmastar2,
                                                        ksi2,
                                                        rho,
                                                        intervals2,
                                                        simulations2,
                                                        timetomaturity,
                                                        volatility,
                                                        stockprice,
                                                        interestrate,
                                                        strikeprice );
      double ssv5=new StochasticPriceVolatility().getQ1Call(intervals2,
                                                          simulations2,
                                                        timetomaturity,
                                                            volatility,
                                                           stockprice,
                                                        interestrate,
                                                        strikeprice);

      double ssv6=new StochasticPriceVolatility().getQ2Call(intervals2,
                                                          simulations2,
                                                        timetomaturity,
                                                            volatility,
                                                           stockprice,
                                                          interestrate,
                                                           strikeprice);



ssvpriceField.setText(numberFormatter.format((ssv1+ssv2+ssv3+ssv4+ssv5+ssv6
)/6));

// System.out.println((ssv1+ssv2+ssv3+ssv4+ssv5+ssv6)/6);
         }
      }

   else  {

      double ssv1=new StochasticPriceVolatility().getP1Put(alphatwo,
                                                        sigmastar2,
                                                        ksi2,
                                                        rho,
                                                        intervals2,
                                                        simulations2,
                                                        timetomaturity,
                                                        volatility,
                                                        stockprice,
```

```
                                                        interestrate,
                                                        strikeprice );
        double ssv2=new StochasticPriceVolatility().getP2Put(alphatwo,
                                                        sigmastar2,
                                                        ksi2,
                                                        rho,
                                                        intervals2,
                                                        simulations2,
                                                        timetomaturity,
                                                        volatility,
                                                        stockprice,
                                                        interestrate,
                                                        strikeprice );
        double ssv3=new StochasticPriceVolatility().getP3Put(alphatwo,
                                                        sigmastar2,
                                                        ksi2,
                                                        rho,
                                                        intervals2,
                                                        simulations2,
                                                        timetomaturity,
                                                        volatility,
                                                        stockprice,
                                                        interestrate,
                                                        strikeprice );

        double ssv4=new StochasticPriceVolatility().getP4Put(alphatwo,
                                                      sigmastar2,
                                                        ksi2,
                                                        rho,
                                                        intervals2,
                                                        simulations2,
                                                        timetomaturity,
                                                        volatility,
                                                        stockprice,
                                                        interestrate,
                                                        strikeprice );
        double ssv5=new StochasticPriceVolatility(). getQ1Put(intervals2,
                                                          simulations2,
                                                        timetomaturity,
                                                            volatility,
                                                            stockprice,
                                                          interestrate,
                                                           strikeprice);

        double ssv6=new StochasticPriceVolatility(). getQ2Put(intervals2,
                                                          simulations2,
                                                        timetomaturity,
                                                            volatility,
                                                            stockprice,
                                                          interestrate,
                                                           strikeprice);



ssvpriceField.setText(numberFormatter.format((ssv1+ssv2+ssv3+ssv4+ssv5+s
sv6)/6));
        }
 }

if (source == calculate4Button) {
```

```
   if (callButton.isSelected()){

      if(normalDistributionAButton.isSelected()){

double bscall3 = new
PowerSeries().calculatePSA(stockprice,strikeprice,volatility,
      timetomaturity,interestrate,ksi3);

pspriceField.setText(numberFormatter.format(bscall3));
      }

      else {

      for(stockprice=0.75; stockprice<1.26; stockprice=stockprice+0.01){

double bscall3 = new
PowerSeries().calculatePSB(stockprice,strikeprice,volatility,
      timetomaturity,interestrate,ksi3);

pspriceField.setText(numberFormatter.format(bscall3));
// to check - System.out.println(bscall3);
       }
      }
   }

   else {

      if(normalDistributionAButton.isSelected()){

      double bscall3 = new
PowerSeries().calculatePSA(stockprice,strikeprice,volatility,
      timetomaturity,interestrate,ksi3);
double putPrice=(strikeprice*Math.exp(- interestrate/100*
timetomaturity/360)+(bscall3)-stockprice);

pspriceField.setText(numberFormatter.format(putPrice));
      }

      else {

      double bscall3 = new
PowerSeries().calculatePSB(stockprice,strikeprice,volatility,
      timetomaturity,interestrate,ksi3);
double putPrice=(strikeprice*Math.exp(- interestrate/100*
timetomaturity/360)+(bscall3)-stockprice);

pspriceField.setText(numberFormatter.format(putPrice));
      }
   }
}


// CALCULATE ALL BUTTON
if (source == calculateButton) {

   if (callButton.isSelected()){

      if(normalDistributionAButton.isSelected()){

      // BS Model
```

```
      double bscall = new
BlackScholesFormula().BlackScholesCallA(stockprice,strikeprice,volatility,

timetomaturity,interestrate);

      bspriceField.setText(numberFormatter.format(bscall));

      // SV Model
      double stvol1=new
StochasticVolatilityFile1().simulateVolatility1(alphaone,ksi1,
sigmastar1, intervals1,  simulations1, volatility, timetomaturity);

      double svcall1=new
BlackScholesFormula().BlackScholesCallA(stockprice, strikeprice, stvol1,
timetomaturity, interestrate);

       double stvol2=new
StochasticVolatilityFile1().simulateVolatility2(alphaone,ksi1,
sigmastar1, intervals1, simulations1, volatility, timetomaturity);

       double svcall2=new
BlackScholesFormula().BlackScholesCallA(stockprice,
      strikeprice, stvol2, timetomaturity, interestrate);

      double svcall=(svcall1+svcall2)/2 ;
      svpriceField.setText(numberFormatter.format(svcall));

      // SSV
      double ssv1=new StochasticPriceVolatility().getP1Call(alphatwo,
                                                            sigmastar2,
                                                            ksi2,
                                                            rho,
                                                            intervals2,
                                                            simulations2,
                                                            timetomaturity,
                                                            volatility,
                                                            stockprice,
                                                            interestrate,
                                                            strikeprice );
      double ssv2=new StochasticPriceVolatility().getP2Call(alphatwo,
                                                            sigmastar2,
                                                            ksi2,
                                                            rho,
                                                            intervals2,
                                                            simulations2,
                                                            timetomaturity,
                                                            volatility,
                                                            stockprice,
                                                            interestrate,
                                                            strikeprice );
      double ssv3=new StochasticPriceVolatility().getP3Call(alphatwo,
                                                            sigmastar2,
                                                            ksi2,
                                                            rho,
                                                            intervals2,
                                                            simulations2,
                                                            timetomaturity,
                                                            volatility,
                                                            stockprice,
                                                            interestrate,
                                                            strikeprice );
```

```
      double ssv4=new StochasticPriceVolatility().getP4Call(alphatwo,
                                                            sigmastar2,
                                                            ksi2,
                                                            rho,
                                                            intervals2,
                                                            simulations2,
                                                            timetomaturity,
                                                            volatility,
                                                            stockprice,
                                                            interestrate,
                                                            strikeprice );
      double ssv5=new StochasticPriceVolatility(). getQ1Call(intervals2,
                                                              simulations2,
                                                              timetomaturity,
                                                              volatility,
                                                                stockprice,
                                                              interestrate,
                                                               strikeprice);
      double ssv6=new StochasticPriceVolatility(). getQ2Call(intervals2,
                                                            simulations2,
                                                            timetomaturity,
                                                              volatility,
                                                              stockprice,
                                                            interestrate,
                                                             strikeprice);


   ssvpriceField.setText(numberFormatter.format((ssv1+ssv2+ssv3+ssv4+ssv5+s
sv6)/6));

      // Series Solution
      double bscall3 = new
PowerSeries().calculatePSA(stockprice,strikeprice,volatility,
      timetomaturity,interestrate,ksi3);

pspriceField.setText(numberFormatter.format(bscall3));
      }

      // normaldistribution B
      else {

      // BS
      double bscall = new
BlackScholesFormula().BlackScholesCallB(stockprice,strikeprice,volatility,
      timetomaturity,interestrate);

bspriceField.setText(numberFormatter.format(bscall));

      // SV
      double stvol1=new
StochasticVolatilityFile1().simulateVolatility1(alphaone,ksi1,
sigmastar1, intervals1, simulations1, volatility, timetomaturity);
      double svcall1=new
BlackScholesFormula().BlackScholesCallB(stockprice, strikeprice, stvol1,
timetomaturity, interestrate);

       double stvol2=new
StochasticVolatilityFile1().simulateVolatility2(alphaone,ksi1, sigmastar1,
intervals1, simulations1, volatility, timetomaturity);
```

```
       double svcall2=new
BlackScholesFormula().BlackScholesCallB(stockprice, strikeprice,
stvol2, timetomaturity, interestrate);

      double svcall=(svcall1+svcall2)/2 ;
      svpriceField.setText(numberFormatter.format(svcall));

      // SSV
      double ssv1=new StochasticPriceVolatility().getP1Call(alphatwo,
                                                        sigmastar2,
                                                        ksi2,
                                                        rho,
                                                        intervals2,
                                                        simulations2,
                                                        timetomaturity,
                                                        volatility,
                                                        stockprice,
                                                        interestrate,
                                                        strikeprice );
      double ssv2=new StochasticPriceVolatility().getP2Call(alphatwo,
                                                        sigmastar2,
                                                        ksi2,
                                                        rho,
                                                        intervals2,
                                                        simulations2,
                                                        timetomaturity,
                                                        volatility,
                                                        stockprice,
                                                        interestrate,
                                                        strikeprice );
      double ssv3=new StochasticPriceVolatility().getP3Call(alphatwo,
                                                        sigmastar2,
                                                        ksi2,
                                                        rho,
                                                        intervals2,
                                                        simulations2,
                                                        timetomaturity,
                                                        volatility,
                                                        stockprice,
                                                        interestrate,
                                                        strikeprice );

      double ssv4=new StochasticPriceVolatility().getP4Call(alphatwo,
                                                        sigmastar2,
                                                        ksi2,
                                                        rho,
                                                        intervals2,
                                                        simulations2,
                                                        timetomaturity,
                                                        volatility,
                                                        stockprice,
                                                        interestrate,
                                                        strikeprice );
      double ssv5=new StochasticPriceVolatility(). getQ1Call(intervals2,
                                                          simulations2,
                                                          timetomaturity,
                                                          volatility,
                                                            stockprice,
                                                          interestrate,
                                                           strikeprice);
      double ssv6=new StochasticPriceVolatility(). getQ2Call(intervals2,
```

```
                                                        simulations2,
                                                        timetomaturity,
                                                          volatility,
                                                          stockprice,
                                                        interestrate,
                                                         strikeprice);


   ssvpriceField.setText(numberFormatter.format((ssv1+ssv2+ssv3+ssv4+ssv5+s
sv6)/6));

      // Series Solution
      double bscall3 = new
PowerSeries().calculatePSB(stockprice,strikeprice,volatility,
timetomaturity,interestrate,ksi3);

pspriceField.setText(numberFormatter.format(bscall3));
      }
   }

   // if put button is selected
   else{

      if(normalDistributionAButton.isSelected()){


      // BS
      double bsput = new
BlackScholesFormula().BlackScholesPutA(stockprice,strikeprice,volatility,
timetomaturity,interestrate);

bspriceField.setText(numberFormatter.format(bsput));

      // SV
      double stvol1=new
StochasticVolatilityFile1().simulateVolatility1(alphaone,ksi1,
sigmastar1, intervals1, simulations1, volatility,timetomaturity);

   double svput1=new BlackScholesFormula().BlackScholesPutA(stockprice,
   strikeprice, stvol1, timetomaturity,  interestrate);


   double stvol2=new
   StochasticVolatilityFile1().simulateVolatility2(alphaone,ksi1,
   sigmastar1, intervals1, simulations1, volatility, timetomaturity);

   double svput2=new BlackScholesFormula().BlackScholesPutA(stockprice,
   strikeprice, stvol2, timetomaturity, interestrate);


   double svput=(svput1+svput2)/2 ;
   svpriceField.setText(numberFormatter.format(svput));

      // SSV
      double ssv1=new StochasticPriceVolatility().getP1Put(alphatwo,
                                                        sigmastar2,
                                                        ksi2,
                                                        rho,
                                                        intervals2,
                                                        simulations2,
                                                        timetomaturity,
```

```
                                                        volatility,
                                                        stockprice,
                                                        interestrate,
                                                        strikeprice );
      double ssv2=new StochasticPriceVolatility().getP2Put(alphatwo,
                                                        sigmastar2,
                                                        ksi2,
                                                        rho,
                                                        intervals2,
                                                        simulations2,
                                                        timetomaturity,
                                                        volatility,
                                                        stockprice,
                                                        interestrate,
                                                        strikeprice );
      double ssv3=new StochasticPriceVolatility().getP3Put(alphatwo,
                                                        sigmastar2,
                                                        ksi2,
                                                        rho,
                                                        intervals2,
                                                        simulations2,
                                                        timetomaturity,
                                                        volatility,
                                                        stockprice,
                                                        interestrate,
                                                        strikeprice );

      double ssv4=new StochasticPriceVolatility().getP4Put(alphatwo,
                                                        sigmastar2,
                                                        ksi2,
                                                        rho,
                                                        intervals2,
                                                        simulations2,
                                                        timetomaturity,
                                                        volatility,
                                                        stockprice,
                                                        interestrate,
                                                        strikeprice );
      double ssv5=new StochasticPriceVolatility(). getQ1Put(intervals2,
                                                          simulations2,
                                                          timetomaturity,
                                                          volatility,
                                        stockprice,
                                                          interestrate,
                                                           strikeprice);
      double ssv6=new StochasticPriceVolatility(). getQ2Put(intervals2,
                                                          simulations2,
                                                          timetomaturity,
                                                            volatility,
                                       stockprice,
                                                          interestrate,
                                                           strikeprice);


   ssvpriceField.setText(numberFormatter.format((ssv1+ssv2+ssv3+ssv4+ssv5+s
sv6)/6));

      // Series Solution
      double bscall3 = new
PowerSeries().calculatePSA(stockprice,strikeprice,volatility,
timetomaturity,interestrate,ksi3);
```

```
      double putPrice=(strikeprice*Math.exp(- interestrate/100*
timetomaturity/360)+(bscall3)-stockprice);

pspriceField.setText(numberFormatter.format(putPrice));
      }

      // is normaldistribution B is used (put)
      else {

      // BS
      double bsput = new
BlackScholesFormula().BlackScholesPutB(stockprice,strikeprice,volatility,
            timetomaturity,interestrate);

bspriceField.setText(numberFormatter.format(bsput));

      // SV
      double stvol1=new
StochasticVolatilityFile1().simulateVolatility1(alphaone,ksi1,
sigmastar1, intervals1, simulations1, volatility, timetomaturity);


      double svput1=new BlackScholesFormula().BlackScholesPutB(stockprice,
               strikeprice, stvol1, timetomaturity, interestrate);

       double stvol2=new
StochasticVolatilityFile1().simulateVolatility2(alphaone,ksi1,
sigmastar1, intervals1, simulations1, volatility, timetomaturity);

       double svput2=new BlackScholesFormula().BlackScholesPutB(stockprice,
strikeprice, stvol2, timetomaturity, interestrate);


      double svput=(svput1+svput2)/2 ;
      svpriceField.setText(numberFormatter.format(svput));

      // SSV
      double ssv1=new StochasticPriceVolatility().getP1Put(alphatwo,
                                                           sigmastar2,
                                                           ksi2,
                                                           rho,
                                                           intervals2,
                                                           simulations2,
                                                           timetomaturity,
                                                           volatility,
                                                           stockprice,
                                                           interestrate,
                                                           strikeprice );
      double ssv2=new StochasticPriceVolatility().getP2Put(alphatwo,
                                                           sigmastar2,
                                                           ksi2,
                                                           rho,
                                                           intervals2,
                                                           simulations2,
                                                           timetomaturity,
                                                           volatility,
                                                           stockprice,
                                                           interestrate,
                                                           strikeprice );
      double ssv3=new StochasticPriceVolatility().getP3Put(alphatwo,
```

```
                                                        sigmastar2,
                                                        ksi2,
                                                        rho,
                                                        intervals2,
                                                        simulations2,
                                                        timetomaturity,
                                                        volatility,
                                                        stockprice,
                                                        interestrate,
                                                        strikeprice );

      double ssv4=new StochasticPriceVolatility().getP4Put(alphatwo,
                                                        sigmastar2,
                                                        ksi2,
                                                        rho,
                                                        intervals2,
                                                        simulations2,
                                                        timetomaturity,
                                                        volatility,
                                                        stockprice,
                                                        interestrate,
                                                        strikeprice );
      double ssv5=new StochasticPriceVolatility(). getQ1Put(intervals2,
                                                          simulations2,
                                                          timetomaturity,
                                                          volatility,
                                                           stockprice,
                                                          interestrate,
                                                           strikeprice);
        double ssv6=new StochasticPriceVolatility(). getQ2Put(intervals2,
                                                          simulations2,
                                                          timetomaturity,
                                                            volatility,
                                                           stockprice,
                                                          interestrate,
                                                           strikeprice);


   ssvpriceField.setText(numberFormatter.format((ssv1+ssv2+ssv3+ssv4+ssv5+s
sv6)/6));

      // Series Solution
      double bscall3 = new
PowerSeries().calculatePSB(stockprice,strikeprice,volatility,
timetomaturity,interestrate,ksi3);

      double putPrice=(strikeprice*Math.exp(- interestrate/100*
timetomaturity/360)+(bscall3)-stockprice);

pspriceField.setText(numberFormatter.format(putPrice));
      }
   }
 } // closing calculate all
} // closing actionlistener

    //if focus is gained, do nothing
    public void focusGained(FocusEvent e){

    }
    //if focus is lost, do something
    public void focusLost(FocusEvent e){
```

```
//find the source which called focus lost
Object source = e.getSource();



 //if the source is timetomaturity
if (source == timetomaturityField){
    timetomaturity=readInt( timetomaturityField,
                                    timetomaturity,
                               "Time to maturity");
   return;
}


//if the source is the intervals1
if (source == intervals1Field){
    intervals1=readInt(intervals1Field,
                            intervals1,
                       "Intervals1 Field");
   return;
}

 //if the source is the intervals2
if (source == intervals2Field){
    intervals2=readInt(intervals2Field,
                            intervals2,
                       "Intervals2Field");
   return;
}

 //if the source is the simulations1
if (source == simulations1Field){
    simulations1=readInt(simulations1Field,
                              simulations1,
                       "Simulations1 Field");
   return;
}

 //if the source is the simulations2
if (source == simulations2Field){
    simulations2=readInt(simulations2Field,
                              simulations2,
                       "Simulations2 Field");
   return;
}


//if the source is the stockprice
if (source == stockpriceField){
   stockprice=readPositive(stockpriceField,
                         stockprice,
                         "Stock price");
   return;
}

//if the source is  strikeprice
if (source == strikepriceField){
   strikeprice=readPositive(strikepriceField,
                            strikeprice,
                         "Strike price");
   return;
```

```
}
//if the source is volatility
if (source == volatilityField){
   volatility=readPositive(volatilityField,
                           volatility,
                           "Volatility Field");
   return;
}

 //if the source is interest rate
if (source == interestrateField){
   interestrate=readPositive(interestrateField,
                           interestrate,
                           "Interest rate");
   return;
}

 //if the source is  alphaone
if (source == alphaoneField){
   alphaone=readPositive(alphaoneField,
              alphaone,
              "Alpha one field");
   return;
}


 //if the source is   alphatwo
if (source == alphatwoField){
   alphatwo=readPositive(alphatwoField,
              alphatwo,
              "Alpha two field");
   return;
}

 //if the source is  sigmastar1
if (source == sigmastar1Field){
    sigmastar1=readPositive(sigmastar1Field,
              sigmastar1,
              "Sigma star1 field");
   return;
}

//if the source is  sigmastar2
if (source == sigmastar2Field){
    sigmastar2=readPositive(sigmastar2Field,
              sigmastar2,
              "Sigma star2 field");
   return;
}

  //if the source is  ksi1
if (source == ksi1Field){
    ksi1=readPositive(ksi1Field,
              ksi1,
              "Ksi1 field");
   return;
}

 //if the source is  ksi2
if (source == ksi2Field){
    ksi2=readPositive(ksi2Field,
```

```
                ksi2,
                "Ksi2 field");
      return;
   }

    //if the source is  ksi3
   if (source == ksi3Field){
       ksi3=readPositive(ksi3Field,
                ksi3,
                "Ksi3 field");
      return;
   }

    //if the source is rho
   if (source == rhoField){
       rho=readPositive1(rhoField,
                rho,
                "Rho field");
      return;
   }


}
//read positive double numbers
private double readPositive(JTextField field,
                            double oldValue,
                            String title)
{
   boolean isOK = true;
   double newValue = 1;
   try{ //test input
      newValue = Double.parseDouble(field.getText());
   }
   catch (NumberFormatException e){//Error message
      JOptionPane.showMessageDialog(null,
                                    NOT_A_NUMBER,
                                    title,
                                    JOptionPane.ERROR_MESSAGE);
      isOK = false;
   }
   if (newValue <=0){//ERROR message
      JOptionPane.showMessageDialog(null,
                                    NOT_POSITIVE,
                                    title,
                                    JOptionPane.ERROR_MESSAGE);
      isOK = false;
   }
   if (isOK){
      return newValue;
   }
   else {
      field.setText(numberFormatter.format(oldValue));
      return oldValue;
   }
}

    //read rho -1 t0 1
private double readPositive1(JTextField field,
                            double oldValue,
                            String title)
{
```

```
    boolean isOK = true;
    double newValue = 1;
    try{ //test input
       newValue = Double.parseDouble(field.getText());
    }
    catch (NumberFormatException e){//Error message
       JOptionPane.showMessageDialog(null,
                                     NOT_A_NUMBER,
                                     title,
                                     JOptionPane.ERROR_MESSAGE);
       isOK = false;
    }
    if (newValue <-1 ^ newValue >1){//ERROR message
       JOptionPane.showMessageDialog(null,
                                     NOT_ACCEPTED,
                                     title,
                                     JOptionPane.ERROR_MESSAGE);
       isOK = false;
    }
    if (isOK){
       return newValue;
    }
    else {
       field.setText(numberFormatter.format(oldValue));
       return oldValue;
    }
}

     //read double numbers
private double readDouble(JTextField field,
                          double oldValue,
                          String title){
    boolean isOK = true;
    double newValue = 1;
    try {// test input
       newValue = Double.parseDouble(field.getText());
    }
    catch (NumberFormatException e){// ERROR message
       JOptionPane.showMessageDialog(null,
                                     NOT_A_NUMBER,
                                     title,
                                     JOptionPane.ERROR_MESSAGE);
       isOK = false;
    }
    if (isOK) {
       return newValue;
    }
    else {
       field.setText(numberFormatter.format(oldValue));
       return oldValue;
    }
}

//Read integer numbers
private int readInt(JTextField field,
                    int oldValue,
                    String title) {
    boolean isOK = true;
    int newValue = 1;
    try { // test input
       newValue = Integer.parseInt(field.getText());
```

```
    }
    catch (NumberFormatException e){// ERROR message
       JOptionPane.showMessageDialog(null,
                                     NOT_INTEGER,
                                     title,
                                     JOptionPane.ERROR_MESSAGE);
       isOK = false;
    }
    if (newValue <=0) {//ERROR message
       JOptionPane.showMessageDialog(null,
                                     NOT_POSITIVE,
                                     title,
                                     JOptionPane.ERROR_MESSAGE);
       isOK = false;
    }
    if (isOK) {
       return newValue;
    }
    else {
       field.setText(numberFormatter.format(oldValue));
       return oldValue;
    }
}

}
```