BACHELOR THESIS IN MATHEMATICS/APPLIED MATHEMATICS

The Java applet for pricing Asian options under Heston’s model using the new Ninomiya weak approximation scheme and quasi-Monte Carlo

by

Boyko Vasilev

Kandidatarbete i matematik / tillämpad matematik

MÄLARDALEN UNIVERSITY
UKK / TM
Box 883
SE-721 23 Västerås, Sweden
Bachelor Thesis in Mathematics/Applied Mathematics

Date:
2008-05-14

Project name:
The Java applet for pricing Asian options under Heston’s model using the new Ninomiya weak approximation scheme and quasi-Monte Carlo

Author(s):
Boyko Vasilev (boyko.vasilev@gmail.com)

Supervisor(s):
Anatoliy Malyarenko

Examiner:
Dimitri Silvestrov

Comprising:
15 ECTS credits (hp)
Abstract

This study is based on a new weak-approximation scheme for stochastic differential equations applied to the Heston stochastic volatility model. The scheme was published by Ninomiya and Ninomiya (2008) and is an extension of Kusuoka’s approximation scheme.

Ninomiya’s algorithm decomposes Kusuoka’s stochastic model into a set of ordinary differential equations with random coefficients and suggests several numerical optimisations for faster calculation.

The subject of this paper is a Java applet which calculates the price of an Asian option under the Heston model.
Acknowledgements

I would like to thank my supervisor, Anatoliy Malyarenko, for his guidance and help during my work on this thesis. I would also like to thank Michael Meyer from the Martingale project for his implementation of quasi-random generator.
Summary

The subject of this thesis work is building a Java Applet for calculating the price of Asian average strike call options. The price is calculated based on the Heston stochastic volatility model using a new weak approximation scheme by Ninomiya and Ninomiya (2008). The applet implements a quasi-Monte Carlo simulation method for faster convergence of the approximation algorithm. The user can specify the number of simulated paths in order to achieve the desired accuracy. The applet outputs the price of the option and a graphical representation of the change in price depending on the number of simulations. The applet was built with the Eclipse development environment and the Java 6 SDK. The free java library JFreeChart was used for plotting graphical output.
Contents

1 The Problem .................................................. 6
  1.1 The Black and Scholes Model ............................... 6
  1.2 Asian options ........................................... 6

2 Theoretical framework ........................................... 8
  2.1 Notation ................................................ 8
  2.2 The Heston model ......................................... 8
  2.3 The new Ninomiya weak approximation scheme .......... 9

3 Implementation of the algorithm .............................. 13
  3.1 Monte Carlo integration .................................. 13
  3.2 quasi-Monte Carlo integration ............................ 14
    3.2.1 Low-discrepancy sequences ......................... 15
    3.2.2 Normally distributed quasi-random sequences .... 16
  3.3 The Java applet ......................................... 18
    3.3.1 Input ............................................... 18
    3.3.2 Output ............................................ 19

4 Analysis ...................................................... 21
  4.1 Experiments .............................................. 21
  4.2 Convergence and integration error ....................... 25
  4.3 Accuracy of the algorithm ................................ 25

5 Conclusion .................................................... 26

A Source-code listing of Hestonwa.java ....................... 28

B Source-code listing for Derivs.java ......................... 46

C Source-code listing for NX.java ............................. 48
List of Figures

3.1 Integration of $f(x)$ over $[a,b]$ .................................................. 13
3.2 Scatter-plot of pseudo- and quasi-random sequences, N=100 ............... 15
3.3 Inverted uniform and native standard normal sequences, $N = 100$ ......... 17
3.4 The applet interface ......................................................... 18
3.5 Check for correctness of the correlation coefficient ................................ 19
3.6 Check for correct inputs .................................................... 19
3.7 Check for the inequality $2a\theta - \beta^2 > 0$ .................................. 19

4.1 Asian call option value depending on stock price, given $K = 1.05$, $r = 0.05$, $\sigma = 0.09$, $T = 1$, $\alpha = 2$, $\beta = 0.1$, $\theta = 0.09$, $\rho = 0$, time-steps = 48 and $N = 3000$ simulations. .......................................................... 21
4.2 Asian call option value depending on strike price, given $S = 1$, $r = 0.05$, $\sigma = 0.09$, $T = 1$, $\alpha = 2$, $\beta = 0.1$, $\theta = 0.09$, $\rho = 0$, time-steps = 48 and $N = 3000$ simulations. .......................................................... 22
4.3 Asian call option value depending on risk-free interest, given $S = 1$, $K = 1.05$, $r = 0.05$, $\sigma = 0.09$, $T = 1$, $\alpha = 2$, $\beta = 0.1$, $\theta = 0.09$, $\rho = 0$, time-steps = 48 and $N = 3000$ simulations. .......................................................... 22
4.4 Asian call option value depending on annualized volatility, given $S = 1$, $K = 1.05$, $r = 0.05$, $\sigma = 0.09$, $T = 1$, $\alpha = 2$, $\beta = 0.1$, $\theta = 0.09$, $\rho = 0$, time-steps = 48 and $N = 3000$ simulations. .......................................................... 22
4.5 Asian call option value depending on time to maturity, given $S = 1$, $K = 1.05$, $r = 0.05$, $\sigma = 0.09$, $T = 1$, $\alpha = 2$, $\beta = 0.1$, $\theta = 0.09$, $\rho = 0$, time-steps = 48 and $N = 3000$ simulations. .......................................................... 22
4.6 Asian call option value depending on speed of mean-reversion of the volatility, given $S = 1$, $K = 1.05$, $r = 0.05$, $\sigma = 0.09$, $T = 1$, $\alpha = 2$, $\beta = 0.1$, $\theta = 0.09$, $\rho = 0$, time-steps = 48 and $N = 3000$ simulations. .......................................................... 23
4.7 Asian call option value depending on the long-run mean value for volatility, given $S = 1$, $K = 1.05$, $r = 0.05$, $\sigma = 0.09$, $T = 1$, $\alpha = 2$, $\beta = 0.1$, $\rho = 0$, time-steps = 48 and $N = 3000$ simulations. .......................................................... 23
4.8 Asian call option value depending on the volatility of volatility, given $S = 1$, $K = 1.05$, $r = 0.05$, $\sigma = 0.09$, $T = 1$, $\alpha = 2$, $\beta = 0.1$, $\theta = 0.09$, $\rho = 0$, time-steps = 48 and $N = 3000$ simulations. .......................................................... 23
4.9 Asian call option value depending on correlation between stock price and volatility, given $S = 1$, $K = 1.05$, $r = 0.05$, $\sigma = 0.09$, $T = 1$, $\alpha = 2$, $\beta = 0.1$, $\theta = 0.09$, $\rho = 0$ and $N = 3000$ simulations. .......................................................... 24
4.10 Asian call option value depending on number of time-steps, given $S = 1$, $K = 1.05$, $r = 0.05$, $\sigma = 0.09$, $T = 1$, $\alpha = 2$, $\beta = 0.1$, $\theta = 0.09$, $\rho = 0$ and $N = 3000$ simulations. .......................................................... 24
Chapter 1

The Problem

1.1 The Black and Scholes Model

The highly-acknowledged Black and Scholes model was created in 1973 by Fischer Black and Myron Scholes. The model can price non-dividend paying options of European type and has been used widely for pricing securities since the 1970s. Despite its robustness and simple calculation, the model has proved empirically wrong. It does not fit the real market prices perfectly and gives only a rough approximation.

The reason for that is the way the model treats the volatility of the underlying asset. Even though the model assumes that the underlying’s price follows a geometrical Brownian motion (GBM), it uses implied volatility throughout the life of the security. This gives a good approximation for the value of at-the-money options, but is far from fair for in- and out-of-the money options. Careful examination of market data shows that Black and Scholes’ implied volatility for different strikes and maturities are not constant. Plotting the volatility surface from real market data shows that it forms a so called volatility smile, which explains why using one single implied volatility is not reliable. Karoui et al. (1998) shows that the original Black and Scholes formula is very inaccurate for pricing path-dependent stochastic volatility options.

1.2 Asian options

Asian options (also called average options) are becoming increasingly popular in times when instability reigns both the financial and commodity markets. Asian options are exotic options and their underlying variable is not the asset price at maturity, but the average price over a period of time. This type of option has lower volatility and is therefore cheaper than an European option. Average options are attractive to buyers of commodities and energy goods because these good are usually traded at average prices over a period of time. Average options also offer some protection against exchange-rate risks for exporters and multinational corporations. Average options are less vulnerable to price manipulations, because it is more difficult to manipulate the average price of the underlying, than to affect it only at the point of maturity. Average options were traded for the first time in 1987 in Tokyo, where Bankers Trust developed a pricing formula for options based on the average price of crude oil. Because it was firstly used in Asia, they were called Asian options. There are two basic types: average price option and average strike option. The payoff of an average price option is the difference between the average price \( S_A \) and strike price \( K \), as follows:
\[ Call_{avg.price} = \text{Max} \left( 0, (S_A - K) \right) \]
\[ Put_{avg.price} = \text{Max} \left( 0, (K - S_A) \right) \]

Average strike options, on the other hand, have a payoff based on the difference between the asset price at maturity \( S_T \) and its average price \( S_A \). That is:

\[ Call_{avg.strike} = \text{Max} \left( 0, (S_T - S_A) \right) \]
\[ Put_{avg.strike} = \text{Max} \left( 0, (S_A - S_T) \right) \]

Asian options also differ in the type of averaging function used to calculate the payoff. We can have arithmetic Asian options or geometric Asian options.

\[ S_A(\text{arithmetic}) = \frac{1}{n} \left[ \sum_{i=1}^{n} S_i \right] = \frac{s_1 + s_2 + s_3 + \ldots + s_n}{n} \]
\[ S_A(\text{geometric}) = \left( \prod_{i=1}^{n} S_i \right)^{\frac{1}{n}} = \sqrt[n]{s_1s_2\ldots s_n} \]

I chose to apply the Ninomiya and Ninomiya (2008) implementation of Heston’s model on pricing Asian arithmetic average-strike options, because they are used widely for covering currency exposures and protect against fluctuations of commodity prices.
Chapter 2

Theoretical framework

2.1 Notation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t$</td>
<td>Current time</td>
</tr>
<tr>
<td>$S(t)$</td>
<td>The stock price at time $t$</td>
</tr>
<tr>
<td>$\sigma^2(t)$</td>
<td>The annualised volatility of the underlying stock at time $t$</td>
</tr>
<tr>
<td>$r$</td>
<td>Risk-free interest rate</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>The speed of volatility mean reversion</td>
</tr>
<tr>
<td>$\theta$</td>
<td>The long run mean of the volatility process</td>
</tr>
<tr>
<td>$\beta$</td>
<td>The volatility of the volatility process</td>
</tr>
<tr>
<td>$\rho$</td>
<td>The correlation coefficient between $S(t)$ and $\sigma^2(t)$</td>
</tr>
<tr>
<td>$B_1(t), B_2(t)$</td>
<td>Two independent Brownian motions</td>
</tr>
<tr>
<td>$K$</td>
<td>The strike price of an Asian call option</td>
</tr>
<tr>
<td>$T$</td>
<td>The expiration date of the option contract</td>
</tr>
</tbody>
</table>

2.2 The Heston model

In his paper Heston (1993) proposed a stochastic-volatility model which extends the traditional Black–Scholes model. Heston introduced settings like the non-lognormal distribution of the asset returns, the assets volatility as a stochastic process and the mean-reverting volatility.

Zhang and Jinghong Shu (2003) have shown that Heston’s model can reduce pricing errors by 25% on average compared to Black and Scholes model. Higher accuracy is exhibited especially in pricing middle-term and long-term options, and is explained by adding a random process for the volatility.

This paper describes the implementation of the Heston model which is a two-factor stochastic volatility(SV) model. SV models assume that the underlying asset’s price follows a stochastic process, whose volatility itself is a stochastic process, unlike the single value for volatility in the BS model. This approach overcomes the implied volatility problem from BS model and gives much better approximation for in- and out-of-the-money options.

According to Heston’s model the price of the underlying asset follows a diffusion process with drift $\mu$ and volatility $\sigma$.

$$dS(t) = \mu S(t) \, dt + S(t) \sigma(t) \, dB^1(t), \quad \text{where} \quad S(0) = S_0. \quad (2.1)$$

The volatility, on the other hand, follows a second diffusion process. It is similar to the Cox, Ingersoll and Ross model for the evolution of interest rates. It is non-negative and
Pricing Asian options under Heston’s model using the new Ninomiya weak approximation scheme

reverts to a long-term mean value $\theta$ at rate of reversion $\theta$. The volatility of the volatility process itself is $\beta$.

$$d\sigma^2(t) = \alpha(\theta - \sigma(t))\,dt + \beta\sigma(t)\,dB^2(t), \quad \text{where} \quad \sigma^2(0) = \sigma_0^2. \quad (2.2)$$

Let $B^1(t)$ and $B^2(t)$ be two Brownian motions with correlation $\rho$, while $dB^1(t)$ and $dB^2(t)$ are their respective stochastic differentials.

$$\text{Cov}[dB^1(t), dB^2(t)] = \rho\,dt. \quad (2.3)$$

We can express the correlation relationship (2.3) into equation (2.2), and we yield the following stochastic differential equation for the volatility diffusion process:

$$d\sigma^2(t) = \alpha(\theta - \sigma(t))\,dt + \beta\sigma(t)(\rho\,dB^1(t) + \sqrt{1 - \rho^2}\,dB^2(t)). \quad (2.4)$$

We can summarize equations (2.1) and (2.4) as Heston’s model.

$$\begin{align*}
\frac{dS(t)}{S(t)} &= \mu S(t)\,dt + S(t)\sigma(t)\,dB^1(t), \quad \text{where} \quad S(0) = S_0, \\
\frac{d\sigma^2(t)}{\sigma^2(t)} &= \alpha(\theta - \sigma(t))\,dt + \beta\sigma(t)(\rho\,dB^1(t) + \sqrt{1 - \rho^2}\,dB^2(t)), \quad \text{where} \quad \sigma^2(0) = \sigma_0^2.
\end{align*}$$

2.3 The new Ninomiya weak approximation scheme

A new high-order scheme was developed by Kusuoka et al. (1998, 2005), for the purpose of doing very fast approximations of diffusion processes. In a paper published in 2008 Ninomiya S. and Ninomiya M. extended the scheme and proposed a new implementation method. They used the notion of Lie algebra brackets and suggested that the stochastic approximation operator by Kusuoka can be considered a composition of the solutions of ordinary differential equations (ODEs). The set of ODEs can be approximated using the Runge–Kutta method for ODEs. This implementation is extremely fast and efficient because of the use of Runge–Kutta approximation scheme and the suggested Romberg extrapolation which boosts the performance even further.

In this paper I apply the new SDE weak-approximation scheme to pricing Asian options of European type (exercise only at maturity). The price process for the asset is modelled according to the Heston two-factor stochastic volatility model.

Without touching on details around the derivation of the scheme, we can outline it in a few basic steps. Firstly we define the pricing process:

$$\begin{align*}
Y_1(t, x) &= x_1 + \int_0^t \mu Y_1(s, x)\,ds + \int_0^t Y_1(s, x)\sqrt{Y_2(s, x)}\,dB^1(s), \\
Y_2(t, x) &= x_2 + \int_0^t \alpha(\theta - Y_2(s, x))\,ds + \int_0^t \beta \sqrt{Y(s, x)}\left(\rho\,dB^1(s) + \sqrt{1 - \rho^2}\,dB^2(s)\right),
\end{align*}$$

where $Y_1$ describes the asset price movement, and $Y_2$ is the diffusion process for the volatility. Here $(x_1, x_2) \in (\mathbb{R}_{>0})^2$ is the initial value for price and probability respectively at time 0. $(B^1(t), B^2(t))$ is a two-dimensional Brownian motion with correlation $\rho$, $(-1 < \rho < 1)$.

We want to calculate the value of an Asian average-strike option. Since it is a path-dependent option and its payoff at maturity is $\max(Y_3(T, x)/T - K, 0)$, we introduce a third process $Y_3(t, x)$:

$$Y_3(t, x) = \int_0^t Y_1(s, x)\,ds.$$
The first step in the algorithm is to express the processes from Heston’s model in Itô’s integral form:

\[
S(t) = S_0 + \int_0^t rS(s)ds + \int_0^t S(s)\sigma(s)dB^1(s),
\]

\[
\sigma^2(t) = \sigma_0^2 + \int_0^t \alpha(\theta - \sigma(s))ds + \int_0^t \beta\sigma(s)\rho dB^1(s) + \sqrt{1 - \rho^2}dB^2(s),
\]

\[
I(t) = 0 + \int_0^t S(s)ds.
\]

Let \( f(S(T), \sigma^2(T), I(T)) = \max\{I(T)/T - K, 0\} \) be the payoff of the Asian call option at maturity. Therefore the price of the option is \( D \times C = D \times Ef(S(T), \sigma^2(T), I(T)) \) where \( D \) is the appropriate discount factor over the time to maturity. The discount factor under continuous compounding is given by:

\[
D = \exp^{-rT}
\]

Then we introduce the following notation in order to transform the model to Stratonovich form:

\[
B^0(s) = s,
\]

\[
Y(t) = (S(t), \sigma^2(t), I(t))',
\]

and rewrite the Itô integral form of the Heston model as

\[
Y(t) = Y(0) + \sum_{i=0}^{2} \int_0^t \tilde{V}_i(Y(s)) dB^i(s), \tag{2.5}
\]

where \( \tilde{V}_0, \tilde{V}_1, \tilde{V}_2 : \mathbb{R}^3 \mapsto \mathbb{R}^3 \) are the following functions:

\[
\tilde{V}_0((y_1, y_2, y_3)') = (ry_1, \alpha(\theta - y_2), y_1)',
\]

\[
\tilde{V}_1((y_1, y_2, y_3)') = (y_1\sqrt{y_2}, \rho\beta\sqrt{y_2}, 0)',
\]

\[
\tilde{V}_2((y_1, y_2, y_3)') = (0, \beta\sqrt{(1 - \rho^2)y_2}, 0)'.
\]

Now we can step into transforming (2.5) and (2.6) in Stratonovich form.

\[
Y(t) = Y(0) + \sum_{i=0}^{2} \int_0^t V_i(Y(s)) dB^i(s), \tag{2.7}
\]

where \( V_0, V_1, V_2 : \mathbb{R}^3 \mapsto \mathbb{R}^3 \) are the following functions.

\[
V_0((y_1, y_2, y_3)') = (y_1(r - y_2/2 - \rho\beta/4), \alpha(\theta - y_2) - \beta^2/4, y_1)',
\]

\[
V_1((y_1, y_2, y_3)') = (y_1\sqrt{y_2}, \rho\beta\sqrt{y_2}, 0)' ,
\]

\[
V_2((y_1, y_2, y_3)') = (0, \beta\sqrt{(1 - \rho^2)y_2}, 0)'.
\]

To calculate this, we used formula (6.1.3) from Øksendal (2003). In our notation this formula has the form

\[
V_0^i = V_0^i - \frac{1}{2} \sum_{i=1}^{3} \sum_{k=1}^{3} \frac{\partial V_0^j}{\partial y_k} V_i^k,
\]

\[
V_i^j = V_i^j, \quad i \geq 1.
\]
After we expressed Heston’s model in Stratonovich form as it appears in (2.7), we can introduce the new weak approximation scheme. In Theorem 1.2 from Ninomiya and Ninomiya (2008) we set \( u = \frac{3}{4} \). Then we have \( c_1 = -\frac{1}{2}, c_2 = \frac{3}{2}, R_{11} = \frac{3}{4}, R_{22} = \frac{11}{4}, R_{12} = -\frac{5}{4} \). Let \( \xi_1, \xi_2, \xi_3, \) and \( \xi_4 \) be four independent standard normal random variables. Put

\[
S^1_j = \frac{\sqrt{3}}{2} \xi_1, \quad S^2_j = -\frac{5\sqrt{3}}{6} \xi_1 + \frac{\sqrt{6}}{3} \xi_2, \quad S^3_j = \frac{\sqrt{3}}{2} \xi_3, \quad S^4_j = -\frac{5\sqrt{3}}{6} \xi_3 + \frac{\sqrt{6}}{3} \xi_4.
\]

Then \( E[S^i_j] = 0, E[S^i_j S^j_l] = R_{ij} \delta_{il} \), as desired.

The next step is to calculate the functions \( W_{1,s} = \Phi \Psi_s(Z_1) \) and \( W_{2,s} = \Phi \Psi_s(Z_2) \), \( s > 0 \) which are needed for Corollary 1.1 from Ninomiya and Ninomiya (2008). By definition of \( Z_j \), we have

\[
Z_1 = -\frac{1}{2} v_0 + \frac{\sqrt{3}}{2} \xi_1 v_1 + \frac{\sqrt{3}}{2} \xi_3 v_2, \\
Z_2 = \frac{3}{2} v_0 - \frac{5\sqrt{3}}{6} \xi_1 v_1 + \frac{\sqrt{6}}{3} \xi_2 v_1 - \frac{5\sqrt{3}}{6} \xi_3 v_2 + \frac{\sqrt{6}}{3} \xi_4 v_2,
\]

where \( v_0, v_1, \) and \( v_2 \) are the letters of an alphabet. By definition of \( \Psi_s \), all terms with \( v_0 \) in the right hand side must be multiplied by \( s \), while all the other terms must be multiplied by \( \sqrt{s} \). So,

\[
\Psi_s(Z_1) = -\frac{1}{2} sv_0 + \frac{\sqrt{3}}{2} \xi_1 \sqrt{s} v_1 + \frac{\sqrt{3}}{2} \xi_3 \sqrt{s} v_2, \\
\Psi_s(Z_2) = \frac{3}{2} sv_0 - \frac{5\sqrt{3}}{6} \xi_1 \sqrt{s} v_1 + \frac{\sqrt{6}}{3} \xi_2 \sqrt{s} v_1 - \frac{5\sqrt{3}}{6} \xi_3 \sqrt{s} v_2 + \frac{\sqrt{6}}{3} \xi_4 \sqrt{s} v_2.
\]

By definition of \( \Phi \), the letters must be changed to the corresponding functions. So,

\[
W_{1,s} = -\frac{1}{2} s v_0 + \frac{\sqrt{3}}{2} \xi_1 \sqrt{s} v_1 + \frac{\sqrt{3}}{2} \xi_3 \sqrt{s} v_2, \\
W_{2,s} = \frac{3}{2} s v_0 - \frac{5\sqrt{3}}{6} \xi_1 \sqrt{s} v_1 + \frac{\sqrt{6}}{3} \xi_2 \sqrt{s} v_1 - \frac{5\sqrt{3}}{6} \xi_3 \sqrt{s} v_2 + \frac{\sqrt{6}}{3} \xi_4 \sqrt{s} v_2.
\]

Now let’s consider the following two systems of ordinary differential equations (ODE):

\[
\frac{dy(t)}{dt} = W_{j,s}(y(t)), \quad y(0) = y_0, \quad j = 1, 2, \quad s > 0.
\]

These systems are the ODE-valued random variables in the sense of Ninomiya and Ninomiya (2008), because the definition of \( W_{1,s} \) involves random variables. Let \( a_{jk} \) be an \( M \times M \) matrix with \( a_{jk} = 0 \) for \( i \leq k \), and let \( b_i \) be a vector of length \( M \) satisfying the \( m \)th order conditions (46) from Ninomiya and Ninomiya (2008). The \( M \)-stage Runge–Kutta method of order \( m \) for the above mentioned systems is written as follows.

\[
y^{(i)} = y_0 + s \sum_{k=1}^M a_{ik} W_{j,s}(y^{(k)}), \quad 1 \leq i \leq M,
\]

\[
g(W_{j,s})(y_0) = y_0 + s \sum_{i=1}^M b_i W_{j,s}(y^{(i)}).
\]

In this case we have \( M = 9 \) and \( m = 7 \). The \( a \) and \( b \) coefficients for the seventh-order Runge-Kutta approximation scheme are taken from (Butcher, 2003) as follows:
Let $n$ be a positive integer. Define recursively the second order scheme as the following sequence of random vectors:

$$Y^{(n)}_0 = Y(0),$$
$$Y^{(n)}_{(k+1)T/n} = g(W_{1,1}) \circ g(W_{2,1}) (Y^{(n)}_{kT/n}), \quad 0 \leq k \leq n - 1,$$

and define the third order scheme by the Romberg extrapolation:

$$X^{(2n)}_T = \frac{4}{3} Y^{(2n)}_T - \frac{1}{3} Y^{(n)}_T.$$

Ninomiya and Ninomiya (2008) proved that the average of $X^{(2n)}_T$ approximates system (2.7). In particular,

$$\lim_{n \to \infty} E f(X^{(2n)}_T) = E f(Y(T)).$$

To calculate $E f(X^{(2n)}_T)$, we use quasi-Monte Carlo simulation.
Chapter 3

Implementation of the algorithm

3.1 Monte Carlo integration

Monte Carlo integration is a numerical method for approximating the value of a definite integral. Unlike other numerical methods which use a regular grid, Monte Carlo chooses randomly the points at which the integrand is calculated.

Let’s illustrate the method with a simple example. If you want to calculate a one-dimensional integral of \( f(x) \) on the region \([a, b]\), this means that you are interested in the area of the graph captured between \( a, b, f(x) \) and the x-axis.

![Graph of f(x) over [a, b]](image)

Figure 3.1: Integration of \( f(x) \) over \([a, b]\)

Using numerical methods we can approximate this area with the sum:

\[
I = \int_{a}^{b} f(x) \, dx = \frac{(b - a)}{N} \sum_{i=1}^{N} f(x_i).
\]

Under Monte Carlo integration we sample \( N \) uniformly-distributed random points on the interval \([a, b]\) and calculate the value of the integrand at these points. Then we average the values from all sample points in order to find the area under the graph. The same approach can be used for calculating multidimensional integrals. The expression for Monte Carlo approximation of the multidimensional integral over the \( n \)-dimensional unit hypercube is given by:
\[
\int_0^1 \cdots \int_0^1 g(x_1, \ldots, x_n) dx_1, \ldots, dx_n \approx \frac{1}{N} \sum_{i=1}^N g(\Gamma_i)
\]

Monte Carlo simulation has proved itself very useful for approximating the solutions of problems in the areas of mathematics, physics, biology and finance. Simulation can be considered a case of integral valuation. In order to calculate an expected value we need to integrate the probability density function of the set of outcomes. This method for simulation requires vectors of random numbers as an input. These vectors can be generated by a computer using deterministic methods, i.e. pseudo-random number generators.

Monte Carlo simulation has become so popular because of its characteristics:

- Simplicity of implementation and speed of calculation. In order to simulate very complex processes we need only to sample a set of random points \(x\) and to calculate the integrand \(f(x)\) at these points
- The inputs for the simulation (random vectors) can be generated extremely fast with modern congruential algorithms.
- The ability to approximate integrals in high dimensions without losing accuracy. The standard error does not depend on the number of dimensions, which is the main problem of other methods for numerical integration.

On the other hand, Monte Carlo has certain limitations, which make it far from perfect.

- This simulation method converges quite slowly. The integration error from MC is proportional to \(N^{-\frac{1}{2}}\). In order to decrease the integration error 10 times, we need to increase the sample size 100 times. This makes Monte Carlo simulation computationally intensive.
- The result from this method highly depends on the initial seed and the random vectors
- The simulation outcome is statistical by nature, i.e. there are only probabilistic error bounds for the estimate.

### 3.2 quasi-Monte Carlo integration

Quasi-Monte Carlo(QMC) integration is based on the same idea as the original Monte Carlo method. QMC, however, does not use pseudo-random numbers as an input, but sequences of quasi-random numbers with more uniform behaviour. These sequences are computer generated in a deterministic manner in order to minimize the integration error. QMC integration, unlike crude Monte Carlo, is heavily dependent on the dimensions of integration. For MC the integration error is proportional to \(N^{-\frac{1}{2}}\), while for QMC that is \((\ln N)^s/N\) (where \(s\) is the number of dimensions). This implies that QMC integration can achieve the same integration error with considerably less sample points than crude MC. Another advantage of QMC over traditional MC integration is the nature of the error. Because quasi-Monte Carlo is based on strictly deterministic samples, we get an exact prediction for the error bound.

The quasi-random sequences used for QMC integration are also called low-discrepancy sequences. Discrepancy measures how much a given sequence deviates from the ideal sequence which is evenly distributed in predefined geometric subset. Therefore low-discrepancy sequences fill up the unit hypercube much more evenly than a pseudo-random sequence and avoid gaps and clusterings in space.
Pricing Asian options under Heston’s model using the new Ninomiya weak approximation scheme

3.2.1 Low-discrepancy sequences

There are various algorithms for generating low-discrepancy sequences. Some of them are Halton, Faure, Sobol sequences. Halton is the most basic multi-dimensional quasi-random sequence. It consists of s one-dimensional sequences created with the van der Corput method using different base for every dimension. As the number of dimensions increases, it takes longer time to fill up the hypercube, which makes Halton’s quasi-random generator rather slow.

The Sobol sequence generator uses 2 as a base for all dimensions, which makes it extremely fast even in high dimensions. This generator produces sequences with very good uniformity properties, and has been preferred by many researches as the best-performing algorithm. However, it requires a predefined sets of directional numbers and is not easy to implement. That is why I chose another algorithm for the purposes of my applet.

The Niederreiter-Xing random number generator was developed initially by Harold Niederreiter, and then was further developed by him and Chaoping Xing. This algorithm is based on the theory of $(t,s)$-sequences using the digital method. Other generators based on the digital method are Faure and Sobol. For my applet I have used Michael Meyer’s implementation of the Niederreiter-Xing algorithm. I have borrowed his java class from Boyko Vasilev.

For the pseudo-random sequence I have used Java’s built-in congruential random generator and have supplied a very good seed. You can see that this sequence does not fill-up the grid evenly. The quasi-random sequence was created with Niederreiter-Xing generator. Please notice, that the sequences on Figure 3.2 are not the same as the one used in the actual simulation. I have presented 2 uniformly-distributed sequences for better visual perception of the reader. In the simulation process I use a standard normal sequence obtained through Beasley-Springer-Moro inversion from a Niederreiter-Xing uniform sequence.

For the pseudo-random sequence I have used Java’s built-in congruential random generator and have supplied a very good seed. You can see that this sequence does not fill-up the grid evenly. The quasi-random sequence was created with Niederreiter-Xing generator. Please notice, that the sequences on Figure 3.2 are not the same as the one used in the actual simulation. I have presented 2 uniformly-distributed sequences for better visual perception of the reader. In the simulation process I use a standard normal sequence obtained through Beasley-Springer-Moro inversion from a Niederreiter-Xing uniform sequence.

Figure 3.2: Scatter-plot of pseudo- and quasi-random sequences, N=100
The sequence in a) does not fill up the unit space evenly, while the one in b) inhibits the grid quite uniformly.

(a) 2-dimensional pseudo-random sequence generated
(b) the first 2 dimensions from a 4-dimensional Niederreiter-Xing sequence

3.2.1 Low-discrepancy sequences

There are various algorithms for generating low-discrepancy sequences. Some of them are Halton, Faure, Sobol sequences. Halton is the most basic multi-dimensional quasi-random sequence. It consists of s one-dimensional sequences created with the van der Corput method using different base for every dimension. As the number of dimensions increases, it takes longer time to fill up the hypercube, which makes Halton’s quasi-random generator rather slow.

The Sobol sequence generator uses 2 as a base for all dimensions, which makes it extremely fast even in high dimensions. This generator produces sequences with very good uniformity properties, and has been preferred by many researches as the best-performing algorithm. However, it requires a predefined sets of directional numbers and is not easy to implement. That is why I chose another algorithm for the purposes of my applet.

The Niederreiter-Xing random number generator was developed initially by Harold Niederreiter, and then was further developed by him and Chaoping Xing. This algorithm is based on the theory of $(t,s)$-sequences using the digital method. Other generators based on the digital method are Faure and Sobol. For my applet I have used Michael Meyer’s implementation of the Niederreiter-Xing algorithm. I have borrowed his java class from Boyko Vasilev.

For the pseudo-random sequence I have used Java’s built-in congruential random generator and have supplied a very good seed. You can see that this sequence does not fill-up the grid evenly. The quasi-random sequence was created with Niederreiter-Xing generator. Please notice, that the sequences on Figure 3.2 are not the same as the one used in the actual simulation. I have presented 2 uniformly-distributed sequences for better visual perception of the reader. In the simulation process I use a standard normal sequence obtained through Beasley-Springer-Moro inversion from a Niederreiter-Xing uniform sequence.

Figure 3.2: Scatter-plot of pseudo- and quasi-random sequences, N=100
The sequence in a) does not fill up the unit space evenly, while the one in b) inhibits the grid quite uniformly.
the **martingale** package and have subsequently licensed the derivative work under the GNU General Public License. This implementation is of base 2 for all dimensions, uses the Gray code counter and bit-wise operations, which makes the algorithm extremely fast. I have also used a generator matrix by Gottlieb Pirsic for generating a 4-dimensional low-discrepancy sequence.

### 3.2.2 Normally distributed quasi-random sequences

The low-discrepancy generators described in the previous section produce a sequence that fills up the multidimensional unit cube evenly. Each dimension is a uniformly distributed sequence. For the purposes of financial simulations, however, we want the corresponding quasi-random numbers for the standard normal distribution. The cumulative function for the standard normal distribution is given by:

\[
Y(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{t^2}{2}} dt
\]

Therefore we need to find the inverse cumulative distribution function and to obtain a standard normal sample from our uniform sample. The most-used method for inversion is the so-called Box-Muller algorithm. It uses a pair of uniformly distributed samples to obtain one standard normal sample. Unfortunately we cannot use it, because the Box-Muller inversion is not suitable for low-discrepancy sequences because it scrambles the sequence’s uniformity (Moro, 1995). Instead, we will use the Beasley-Springer-Moro (BSM) method. It was developed by Beasley and Springer (1977) and optimized for low-discrepancy sequences by (Moro, 1995).

Moro divided the domain of the uniform sample \( U \) in 2 regions.

- The central region of the distribution \( |U| \leq 0.42 \) is modelled according to the Beasley-Springer algorithm:

\[
\Phi^{-1}(x) = U \left( \frac{\sum_{i=0}^{3} a_i U^{2i}}{\sum_{i=0}^{4} b_i U^{2i}} \right)
\]

<table>
<thead>
<tr>
<th>( i )</th>
<th>( a_i )</th>
<th>( b_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2.50662823884</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>-18.61500062529</td>
<td>-8.47351093090</td>
</tr>
<tr>
<td>2</td>
<td>41.39119773534</td>
<td>23.08336743743</td>
</tr>
<tr>
<td>3</td>
<td>-25.44106049637</td>
<td>-21.06224101826</td>
</tr>
<tr>
<td>4</td>
<td>3.13082909833</td>
<td></td>
</tr>
</tbody>
</table>

- For the tails of the distribution \( |U| > 0.42 \) he used truncated Chebyshev series:

\[
\Phi^{-1}(x) = \begin{cases} 
\sum_{i=0}^{8} c_i T_i(z) - \frac{c_0}{2}, & \text{for } U > 0 \\
\frac{c_0}{2} - \sum_{i=0}^{8} c_i T_i(z), & \text{for } U \leq 0
\end{cases}
\]

\[
z = k_1 \left[ 2 \ln(-\ln(0.5 - |U|)) - k_2 \right]
\]
I have tested the inversion algorithm and evaluated the quality of the obtained standard normal sequence. Then I compared this sequence with one uniform pseudo-random sequence inverted with the same method and one standard normal sequence that has not been inverted. As you can see on Figure 3.3, the inverted Niederreiter-Xing sequence is distributed in a radially-balanced way around the point $(0, 0)$, which is the mean in 2 dimensions.

On the table below you can also see that the sequence on the figure a) has the best characteristics among all three. It is therefore closest to the theoretically expected standard normal sequence.

![Figure 3.3: Inverted uniform and native standard normal sequences, $N = 100$](image)

<table>
<thead>
<tr>
<th>$i$</th>
<th>$a_i$</th>
<th>$b_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.3374754822726147</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0.9761690190917186</td>
<td>0.4179886424926431</td>
</tr>
<tr>
<td>2</td>
<td>0.160797914918209</td>
<td>4.2454686881376569</td>
</tr>
<tr>
<td>3</td>
<td>0.0276438810333863</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.0038405729373609</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.0003951896511919</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.0000321767881768</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0.0000002888167364</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.0000003960315187</td>
<td></td>
</tr>
</tbody>
</table>

![Table 3.1: Performance of standard normal sequences, $N = 1000$](image)
3.3 The Java applet

In this chapter I will present the Java applet for pricing Asian options. There are 3 distinct panel areas: the input panel, the control panel and the output panel. On pressing the Calculate button the applet starts calculating the option price, and the bar below the Reset button shows the progress of the quasi-Monte Carlo algorithm. When all simulations are complete, the option price will appear in the bottom of the Output panel, together with a graph of the price during the simulation process. The Reset button will clear the last option price from the Output panel and will reset all input variables in the Input and Controls panels to their default values. The progress bar will also be reset to 0.

![Applet Viewer: Hestonwa.class](image)

**Figure 3.4: The applet interface**

### 3.3.1 Input

The input panel contains fields for entering strike price, stock price at time zero, risk-free interest rate and maturity. The last four fields in the Input panel are specific for the Heston model. According to this model the volatility of the asset price is determined by a mean-reverting stochastic process described in equation (2.2). There $\alpha$ is the speed of volatility mean reversion, $\theta$ is the long-run mean of the volatility and $\beta$ is the volatility of the volatility.
process. The last parameter in this panel $\rho$ is the correlation of the two Brownian motions in equations (2.1) and (2.2).

In the Controls panel there are 2 additional parameters, 2 buttons and a progress bar. The number of simulations variable is crucial for the accuracy of the quasi-Monte Carlo algorithm. More simulation cycles will yield smaller integration error, and the expected option price will get closer and closer to the true value. The second variable in this panel is the number of time-steps, which sets the number of averaging days during the life of the option. For example, if time to maturity is 1 year and the user inputs 365 time-steps, then the payoff of the option will be based on the average of 365 daily prices.

![Correlation between S and $\sigma^2$]

**Figure 3.5:** Check for correctness of the correlation coefficient

![Initial stock price and Number of QMC simulations]

(a) Message for a negative input parameter  (b) Message for a non-integer input parameter

**Figure 3.6:** Check for correct inputs

As Ninomiya and Ninomiya (2008) points out, $\alpha$, $\theta$ should be positive coefficients, and $2\alpha\theta - \beta^2 > 0$ should hold to ensure that the stochastic differential equation has a unique solution. If this inequality is not fulfilled, the user will see the following error message after pressing the Calculate button.

![Invalid Inputs]

**Figure 3.7:** Check for the inequality $2\alpha\theta - \beta^2 > 0$

### 3.3.2 Output

In the output panel of the applet there is a graph that demonstrates the convergence of the quasi-Monte Carlo method. On the Y-axis is the option price, and on the X-axis is the num-
number of simulations performed to obtain this price. The user can change the number of simulations in the Controls panel in order to achieve the desired accuracy for the option price. The user can see graphically the speed of convergence of the quasi-Monte Carlo method in order to choose the optimal number of simulations for the desired accuracy.
Chapter 4

Analysis

4.1 Experiments

In this section I will present some experiments I did with the Java applet. I tried to investigate the relationship between the price of the Asian call option and the input parameters of the Heston model. I plotted the option price over a range for each variable, keeping all other inputs constant. In this way we can see the net effect of this variable on the price.

Figure 4.1: Asian call option value depending on stock price, given $K = 1.05$, $r = 0.05$, $\sigma = 0.09$, $T = 1$, $\kappa = 2$, $\beta = 0.1$, $\theta = 0.09$, $\rho = 0$, time-steps = 48 and $N = 3000$ simulations.
Figure 4.2: Asian call option value depending on strike price, given $S = 1$, $r = 0.05$, $\sigma = 0.09$, $T = 1$, $\alpha = 2$, $\beta = 0.1$, $\theta = 0.09$, $\rho = 0$, time-steps = 48 and $N = 3000$ simulations.

Figure 4.3: Asian call option value depending on risk-free interest, given $S = 1$, $K = 1.05$, $\sigma = 0.09$, $T = 1$, $\alpha = 2$, $\beta = 0.1$, $\theta = 0.09$, $\rho = 0$, time-steps = 48 and $N = 3000$ simulations.

Figure 4.4: Asian call option value depending on annualized volatility, given $S = 1$, $K = 1.05$, $r = 0.05$, $T = 1$, $\alpha = 2$, $\beta = 0.1$, $\theta = 0.09$, $\rho = 0$, time-steps = 48 and $N = 3000$ simulations.
Figure 4.5: Asian call option value depending on time to maturity, given $S = 1$, $K = 1.05$, $r = 0.05$, $\sigma = 0.09$, $\alpha = 2$, $\beta = 0.1$, $\theta = 0.09$, $\rho = 0$, time-steps = 48 and $N = 3000$ simulations.

Figure 4.6: Asian call option value depending on speed of mean-reversion of the volatility, given $S = 1$, $K = 1.05$, $r = 0.05$, $\sigma = 0.09$, $\beta = 0.1$, $\theta = 0.09$, $\rho = 0$, time-steps = 48 and $N = 3000$ simulations.

Figure 4.7: Asian call option value depending on the long-run mean value for volatility, given $S = 1$, $K = 1.05$, $r = 0.05$, $\sigma = 0.09$, $T = 1$, $\alpha = 2$, $\beta = 0.1$, $\rho = 0$, time-steps = 48 and $N = 3000$ simulations.
Figure 4.8: Asian call option value depending on the volatility of volatility, given \( S = 1, K = 1.05, r = 0.05, \sigma = 0.09, T = 1, \alpha = 2, \theta = 0.09, \rho = 0 \), time-steps = 48 and \( N = 3000 \) simulations.

Figure 4.9: Asian call option value depending on correlation between stock price and volatility, given \( S = 1, K = 1.05, r = 0.05, \sigma = 0.09, T = 1, \alpha = 2, \beta = 0.1, \theta = 0.09 \), time-steps = 48 and \( N = 3000 \) simulations.

Figure 4.10: Asian call option value depending on number of time-steps, given \( S = 1, K = 1.05, r = 0.05, \sigma = 0.09, T = 1, \alpha = 2, \beta = 0.1, \theta = 0.09, \rho = 0 \) and \( N = 3000 \) simulations.
4.2 Convergence and integration error

In order to evaluate the efficiency and speed of convergence of the algorithm, I measured CPU-times and compared the resulting prices. All input parameters are fixed to the default values \((S = 1, K = 1.05, r = 0.05, \sigma = 0.09, T = 1, \alpha = 2, \beta = 0.1, \theta = 0.09, \rho = 0, \text{time-steps } = 48)\), while increasing the number of simulations. The table below shows the CPU time used for calculation, the price of the option, and the absolute error. Notice that I have used the result from \(N=800000\) simulations as a control value to measure error. The tests were performed on Intel Core2 Duo processor 2.66GHz.

<table>
<thead>
<tr>
<th>Simulations</th>
<th>CPU-time(sec)</th>
<th>Option Price</th>
<th>Error*</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2 \times 10^2)</td>
<td>0.2</td>
<td>0.087305008807</td>
<td>0.00043424469</td>
</tr>
<tr>
<td>(2 \times 10^3)</td>
<td>1.96</td>
<td>0.086914090857</td>
<td>0.00000432674</td>
</tr>
<tr>
<td>(2 \times 10^4)</td>
<td>18.9</td>
<td>0.086874999062</td>
<td>0.000004234945</td>
</tr>
<tr>
<td>(2 \times 10^5)</td>
<td>183.6</td>
<td>0.086871089883</td>
<td>0.000000325766</td>
</tr>
<tr>
<td>(8 \times 10^5)</td>
<td>721</td>
<td>0.086870764117</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4.1: Performance of the applet

A quick look at the first and the last column of the table can give us an approximate measure of the speed of convergence. When the number of simulations \(N\) is increased 10 times, the absolute error decreases 10 times. Therefore we can state that the integration error of the algorithm is proportional to \(N^{-1}\). This is a much better measure than the crude-Monte Carlo integration error proportional to \(N^{-\frac{1}{2}}\) (see Section 3.1).

4.3 Accuracy of the algorithm

In the previous section I described the accuracy of the algorithm in terms of convergence and integration error. Unfortunately I did not find another Option price calculator for Asian options under the Heston model, so I can not compare the correctness of the produced prices. This remains to be a subject of future investigations. It is important to point out that the Heston stochastic volatility method produces more accurate results than other implied volatility methods because of its volatility mean-reverting process.
Chapter 5

Conclusion

The purpose of this thesis work, creating an applet for pricing Asian arithmetic call options under the Heston stochastic volatility model using Ninomiya’s weak approximation scheme, was fully achieved. The applet implements the new approximation scheme for stochastic differential equations described in (Ninomiya and Ninomiya, 2008) and successfully simulates the price of an Asian option. For the simulation algorithm I used the quasi-Monte Carlo method together with a low-discrepancy sequence generated by Niederreiter-Xing algorithm. The partial differential equations for the weak-approximation scheme were calculated by a 7-th order Runge-Kutta method, and Romberg extrapolation was used to boost the method to order three.

In Chapter 4 I demonstrated how the price of an Asian option changes as we change each of the input parameters for the Heston model. I also analysed the speed of calculation and the rate of convergence for the quasi-Monte Carlo method.

However there is no general conclusion for the accuracy of the pricing algorithm. This remains to be investigated further in order to compare the results from the new approximation scheme with other methods incorporating the Heston diffusion processes.
Bibliography


Appendix A

Source-code listing of Hestonwa.java

```java
import java.awt.*;
import java.awt.event.*;
import java.text.*;
import javax.swing.*;
import javax.swing.border.TitledBorder;
import org.jfree.chart.plot.*;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.JFreeChart;
import org.jfree.data.xy.XYSeries;
import org.jfree.data.xy.XYSeriesCollection;
import org.jfree.chart.ChartPanel;

/*
 * Java Applet for pricing Asian options
 * under the Heston stochastic volatility model
 * by Ninomiya weak approximation
 * Version 0.98c
 * 21.04.2008
 * 
 * Author: Boyko Vasilev (boyko.vasilev@gmail.com)
 */

public class Hestonwa extends JApplet implements ActionListener,
FocusListener
{

  private static final long serialVersionUID = -791580869497094349L;
  // variables
  // panels
```
private JPanel mainPanel = null;
private JPanel inputPanel = null;
private JPanel labelPanel = null;
private JPanel dataPanel = null;
private JPanel controlPanel = null;
private JPanel outputPanel = null;
private ChartPanel graphPanel1 = null;
// buttons
private JButton calculateButton = null;
private JButton resetButton = null;
// text fields
private JTextFields strike_KField = null;
private JTextFields stockprice_Field = null;
private JTextFields interest_Field = null;
private JTextFields volatility_Field = null;
private JTextFields maturity_Field = null;
private JTextFields alfa_Field = null;
private JTextFields beta_Field = null;
private JTextFields theta_Field = null;
private JTextFields rho_Field = null;
private JTextFields sim_Field = null;
private JTextFields dim_Field = null;
private JTextFields price_Field = null;
// String constants
private final String CALCULATE = "Calculate";
private final String INPUT = "Input";
private final String RESET = "Reset";
// Texts of labels
private final String STRIKE_K_LABEL = "Strike price";
private final String STOCK_LABEL = "Stock price";
private final String INTEREST_LABEL = "Interest rate";
private final String VOLATILITY_LABEL = "Volatility";
private final String MATURITY_LABEL = "Maturity";
private final String ALFA_LABEL = "Mean reversion coef. (\"+\'\u03B1\'+\")";
private final String BETA_LABEL = "Volatility of Vol. (\"+\'\u03B2\'+\")";
private final String THETA_LABEL = "Long−term average Vol. (\"+\'\u03B8\'+\")";
private final String RHO_LABEL = "Correlation (\"+\'\u03C1\'+\")";
private final String SIM_LABEL = "Simulations";
private final String DIM_LABEL = "Time−steps";
private final String GRAPH1_TITLE = "Simulated option price";
private final String Y1_LABEL = "Option Value";
private final String CONTROL = "Controls";
private final String PRICE_LABEL = "Simulated option price";
// Tooltips
private final String STRIKE_K_TOOLTIP = "The price at which one is prepare to excercise the option";
private final String STOCKPRICE_TOOLTIP = "Initial stock price";
private final String INTEREST_TOOLTIP = "Anual risk free interest rate";
private final String VOLATILITY_TOOLTIP = "Annualized volatility of the stock price";
private final String MATURITY_TOOLTIP = "Maturity in years";
private final String ALFA_TOOLTIP = "Mean reversion coef. (" + 'α' + " )";
private final String BETA_TOOLTIP = "Volatility of the volatility process";
private final String THETA_TOOLTIP = "Long-run mean of the volatility process";
private final String RHO_TOOLTIP = "Correlation";
private final String SIM_TOOLTIP = "Number of QMC simulations";
private final String DIM_TOOLTIP = "Time-discretization";
private final String PRICE_TOOLTIP = "quasi-Monte-Carlo simulated price";
// Error messages
private final String NOT_A_NUMBER = "Enter a number";
private final String NOT_INTEGER = "Enter an integer number";
private final String NON_POSITIVE = "Enter a positive number";
private final String NON_UNIT = "Enter a number in the interval [−1;1]";
// numerical constants
private final double STRIKE_K = 1.05;
private final double STOCKPRICE = 1;
private final double INTEREST = 0.05;
private final double VOLATILITY = 0.09;
private final double MATURITY = 1;
private final double ALFA = 2;
private final double BETA = 0.1;
private final double THETA = 0.09;
private final double RHO = 0;
private final int SIM = 48; // number of simulations/trajectories
private final int DIM = 48; // time-discretization
// numerical variables
private double strike_K = STRIKE_K;
private double stockprice = STOCKPRICE;
private double interest = INTEREST;
private double volatility = VOLATILITY;
private double maturity = MATURITY;
private double alfa = ALFA;
private double beta = BETA;
private double theta = THETA;
private double rho = RHO;
private int sim = SIM;
private int dim = DIM;
private double MC;
public int mc;
private GridBagLayout gbl = new GridBagLayout();
private JProgressBar progressBar = null;
private DecimalFormat numberFormatter = null;
private double[][][] Y = new double[9][3];
private double[][][] W = new double[9][3];
private double[] yOld = new double[3];
private double[] yNew = new double[3];
141 public void init() {
    // get content pane
    Container contentPane = getContentPane();

    // create main panel
    mainPanel = new JPanel();
    // set box layout
    mainPanel.setLayout(new BoxLayout(mainPanel, BoxLayout.Y_AXIS));
    // add main panel to content pane
    contentPane.add(mainPanel);

    // Set proper number format
    DecimalFormatSymbols symbols = new DecimalFormatSymbols();
    symbols.setDecimalSeparator('.');
    numberFormatter = new DecimalFormat("#.###########", symbols);

    // create input panel
    inputPanel = new JPanel(new GridLayout(0, 3));
    inputPanel.setBorder(new TitledBorder(INPUT));

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

    // create labelPanel
    labelPanel = new JPanel(new GridLayout(0, 1));
    inputPanel.add(labelPanel);

    // create data panel
    dataPanel = new JPanel(new GridLayout(0, 1));
    // add it to input panel
    inputPanel.add(dataPanel);

    // create Control panel inside InputPanel
    controlPanel = new JPanel(new GridLayout(7, 2));
    controlPanel.setBorder(new TitledBorder(CONTROL));
    inputPanel.add(controlPanel);

    // create output panel
    outputPanel = new JPanel(gbl);
    outputPanel.setBorder(new TitledBorder("Output"));

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

    // add labels
    JLabel label = new JLabel(STRIKE_K_LABEL);
    labelPanel.add(label);

    // create Strike field

Pricing Asian options under Heston’s model using the new Ninomiya weak approximation scheme

strike_KField = new JTextField();
   // add tooltip
strike_KField.setToolTipText(STRIKE_K_TOOLTIP);
   // set value
strike_KField.setText(numberFormatter.format(STRIKE_K));
   // add focus listener
strike_KField.addFocusListener(this);
   // add it to data panel
dataPanel.add(strike_KField);

   // create stock label
label = new JLabel(STOCK_LABEL);
labelPanel.add(label);
   // create stock field
stockprice_Field = new JTextField();
   // add tooltip
stockprice_Field.setToolTipText(STOCKPRICE_TOOLTIP);
   // set value
stockprice_Field.setText(numberFormatter.format(STOCKPRICE));
   // add focus listener
stockprice_Field.addFocusListener(this);
   // add it to data panel
dataPanel.add(stockprice_Field);

   // create interest label
label = new JLabel(INTEREST_LABEL);
labelPanel.add(label);
   // create stock field
interest_Field = new JTextField();
   // add tooltip
interest_Field.setToolTipText(INTEREST_TOOLTIP);
   // set value
interest_Field.setText(numberFormatter.format(INTEREST));
   // add focus listener
interest_Field.addFocusListener(this);
   // add it to data panel
dataPanel.add(interest_Field);

   // create volatility label
label = new JLabel(VOLATILITY_LABEL);
labelPanel.add(label);
   // create volatility field
volatility_Field = new JTextField();
   // add tooltip
volatility_Field.setToolTipText(VOLATILITY_TOOLTIP);
   // set value
volatility_Field.setText(numberFormatter.format(VOLATILITY));
   // add focus listener
volatility_Field.addFocusListener(this);
// add it to data panel
dataPanel.add(volatility_Field);

// create maturity label
label = new JLabel(MATURITY_LABEL);
labelPanel.add(label);

// create maturity field
maturity_Field = new JTextField();
// add tooltip
maturity_Field.setToolTipText(MATURE_TOOLTIP);
// set value
maturity_Field.setText(numberFormatter.format(MATURITY));
// add focus listener
maturity_Field.addFocusListener(this);

// add it to data panel
dataPanel.add(maturity_Field);

// create alfa label
label = new JLabel(ALFA_LABEL);
labelPanel.add(label);

// create alfa field
alfa_Field = new JTextField();
// add tooltip
alfa_Field.setToolTipText(ALFA_TOOLTIP);
// set value
alfa_Field.setText(numberFormatter.format(ALFA));
// add focus listener
alfa_Field.addFocusListener(this);

// add it to data panel
dataPanel.add(alfa_Field);

// create beta label
label = new JLabel(BETA_LABEL);
labelPanel.add(label);

// create beta field
beta_Field = new JTextField();
// add tooltip
beta_Field.setToolTipText(BETA_TOOLTIP);
// set value
beta_Field.setText(numberFormatter.format(BETA));
// add focus listener
beta_Field.addFocusListener(this);

// add it to data panel
dataPanel.add(beta_Field);

// create theta label
label = new JLabel(THETA_LABEL);
labelPanel.add(label);

// create theta field
theta_Field = new JTextField();
// add tooltip
theta_Field.setToolTipText(THETA_TOOLTIP);
// set value
theta_Field.setText(numberFormatter.format(THETA));
// add focus listener
theta_Field.addFocusListener(this);

// add it to data panel
dataPanel.add(theta_Field);

// create rho label
label = new JLabel(RHO_LABEL);
labelPanel.add(label);

// create rho field
rho_Field = new JTextField();
// add tooltip
rho_Field.setToolTipText(RHO_TOOLTIP);
// set value
rho_Field.setText(numberFormatter.format(RHO));
// add focus listener
rho_Field.addFocusListener(this);

// add it to data panel
dataPanel.add(rho_Field);

// create sim label
label = new JLabel(SIM_LABEL);
controlPanel.add(label);

// create sim field
sim_Field = new JTextField();
// add tooltip
sim_Field.setToolTipText(SIM_TOOLTIP);
// set value
sim_Field.setText(numberFormatter.format(SIM));
// add focus listener
sim_Field.addFocusListener(this);

// add it to data panel
controlPanel.add(sim_Field);

// create dim label
label = new JLabel(DIM_LABEL);
controlPanel.add(label);

// create dim field
dim_Field = new JTextField();
// add tooltip
Pricing Asian options under Heston’s model using the new Ninomiya weak approximation scheme

```java
dim_Field.setToolTipText(DIM_TOOLTIP);
// set value
dim_Field.setText(numberFormatter.format(DIM));
// add focus listener
dim_Field.addFocusListener(this);

// add it to data panel
controlPanel.add(dim_Field);

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

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

GridBagConstraints c = new GridBagConstraints();

// create progress bar
progressBar = new JProgressBar();
progressBar.setStringPainted(true);
progressBar.setValue(0);
progressBar.setIndeterminate(false);
// add it to progress panel
c.fill = GridBagConstraints.HORIZONTAL;
c.gridwidth = 3;
controlPanel.add(progressBar, c);

JFreeChart chart;

// GridBagConstraints c = new GridBagConstraints();
c.fill = GridBagConstraints.BOTH;

// create panel for Jfreechart
graphPanel1 = new ChartPanel(null);
c.ipady = 0; // reset to default
c.weightx = 1.0; // request any extra horizontal space
c.weighty = 0.92; // request any extra vertical space
c.anchor = GridBagConstraints.PAGE_END; // bottom of space
c.insets = new Insets(0, 0, 0, 0); // top padding
c.gridx = 0;
c.gridy = 0;
c.gridwidth = 2;
outputPanel.add(graphPanel1, c);

// create price label
JLabel label2 = new JLabel(PRICE_LABEL);
```

Boyko Vasilev  
Mälardalen University
Pricing Asian options under Heston’s model using the new Ninomiya weak approximation scheme

```java
        c.gridwidth = 1;
        c.gridx = 0;
        c.gridy = 1;
        c.weighty = 0.08;
        label2.setSize(0, 20);
        outputPanel.add(label2, c);

        // create price field
        price_Field = new JTextField();
        // add tooltip
        price_Field.setToolTipText(PRICE_TOOLTIP);
        // set value
        price_Field.setText(numberFormatter.format(0));
        price_Field.setEditable(false);
        // add it to data panel
        c.gridx = 1;
        c.gridy = 1;
        c.weighty = 0.08;
        outputPanel.add(price_Field, c);
        outputPanel.validate();
    }

    public void focusGained(FocusEvent e) {}

    public void focusLost(FocusEvent e) {
        Object source = e.getSource();
        // if strike K field
        if (source == strike_KField) {
            // check strike_K
            strike_K = readPositive(strike_KField, strike_K,
                                    strike_KField.getToolTipText());

            return;
        }
        // If stock price field
        if (source == stockprice_Field) {
            // check stockprice
            stockprice = readPositive(stockprice_Field, stockprice,
                                       stockprice_Field.getToolTipText());

            return;
        }
        // If interest rate field
        if (source == interest_Field) {
            // check interest
            interest = readPositive(interest_Field, interest,
                                     interest_Field.getToolTipText());

            return;
        }
    }
```

Boyko Vasilev

Mälardalen University
Pricing Asian options under Heston’s model using the new Ninomiya weak approximation scheme

```java
// If volatility field
if (source == volatility_Field) {
    // check volatility
    volatility = readPositive(volatility_Field,
                              volatility,
                              volatility_Field.getToolTipText());
    return;
}

// If maturity field
if (source == maturity_Field) {
    // check maturity
    maturity = readPositive(maturity_Field,
                            maturity,
                            maturity_Field.getToolTipText());
    return;
}

// If alfa field
if (source == alfa_Field) {
    alfa = readPositive(alfa_Field,
                        alfa,
                        alfa_Field.getToolTipText());
    return;
}

// If beta field
if (source == beta_Field) {
    beta = readDouble(beta_Field,
                       beta,
                       beta_Field.getToolTipText());
    return;
}

// If theta field
if (source == theta_Field) {
    theta = readPositive(theta_Field,
                          theta,
                          theta_Field.getToolTipText());
    return;
}

// If rho field
if (source == rho_Field) {
    rho = readUnit(rho_Field,
                   rho,
                   rho_Field.getToolTipText());
    return;
}

// If SIM field
if (source == sim_Field) {
    sim = readInt(sim_Field,
                  sim,
                  sim_Field.getToolTipText());
    return;
}
```
Pricing Asian options under Heston’s model using the new Ninomiya weak approximation scheme

```java
    sim_field.getToolTipText();
    return;
}
// If DIM field
if (source == dim_field) {
    dim = readInt(dim_field,
                  dim,
                  dim_field.getToolTipText());
    return;
}
}

public void actionPerformed(ActionEvent e) {
    // determine, who called action listener
    Object source = e.getSource();
    // If Calculate button
    if (source == calculateButton) {
        // verify condition for unique solution of SDEs and
correlation
        if ((2 * alpha * theta - beta * beta > 0) &&
            Math.abs(rho) <= 1) {
            XYSeriesCollection dataset = new XYSeriesCollection();
            XYSeries priceSeries = new XYSeries("Price");
            dataset.removeAllSeries();
            priceSeries.clear();

            // Prepare progress bar
            progressBar.setMaximum(100);
            progressBar.setValue(0);

            long before = System.currentTimeMillis();
            double t;
            int pb;
            MC = 0.0;
            fSingle = new Derivs(maturity/dim, interest, rho, beta, alpha, theta);
            fDouble = new Derivs(maturity/(2*dim), interest, rho, beta, alpha,
                                 theta);
            nxs = new NX(4); // init Niederreiter–Xing generator in 4
dimensions
            for (mc=0; mc<=sim; mc++) {
                t = GetX2n();
                MC = MC + t;
                nxs.restart();
                pb = mc*100 / sim; // refresh progressbar and add points to chart
                                 // only 100 times for all simulations
                if (progressBar.getValue() != pb) {
                    progressBar.setValue(pb);
                    progressBar.paint(progressBar.getGraphics());
                }
            }
```
if (mc != 0) priceSeries.add(mc, MC / mc);
}
MC = MC / sim;
priceField.setText(numberFormatter.format(MC));

JFreeChart chart;
dataset.addSeries(priceSeries);

chart = ChartFactory.createXYLineChart(GRAPH1_TITLE, "Iterations", Y1_LABEL, dataset, PlotOrientation.VERTICAL, true, false, false);

graphPanel1.setChart(chart);
}
else {
    if ((2 * alpha * theta - beta * beta <= 0)) JOptionPane.showMessageDialog(null, "Please observe 2\u03B1\u03C1-\u03B2^2>0 in order to ensure unique solution to Heston’s SDEs", "Invalid inputs", JOptionPane.ERROR_MESSAGE);
    if (Math.abs(rho) > 1) JOptionPane.showMessageDialog(null, "Correlation\u03C1 should be in the interval [-1;1]", "Invalid inputs", JOptionPane.ERROR_MESSAGE);
}

// If Reset button
if (source == resetButton) {
    // reset all TextFields and variables to the initial values
    strike_K = Strike_K;
    strike_KField.setText(numberFormatter.format(Strike_K));
    stockprice = STOCKPRICE;
    stockpriceField.setText(numberFormatter.format(STOCKPRICE));
    interest = INTEREST;
    interestField.setText(numberFormatter.format(INTEREST));
    volatility = VOLATILITY;
    volatilityField.setText(numberFormatter.format(VOLATILITY));
    maturity = MATURITY;
    maturityField.setText(numberFormatter.format(MATURITY));
    alpha = ALFA;
    alphaField.setText(numberFormatter.format(ALFA));
}
Pricing Asian options under Heston’s model using the new Ninomiya weak approximation scheme

```java
beta = BETA;
beta_Field.setText(numberFormatter.format(BETA));
theta = THETA;
theta_Field.setText(numberFormatter.format(THETA));
rho = RHO;
rho_Field.setText(numberFormatter.format(RHO));
sim = SIM;
sim_Field.setText(numberFormatter.format(SIM));
dim = DIM;
dim_Field.setText(numberFormatter.format(DIM));
price_Field.setText(numberFormatter.format(0));
progressBar.setValue(0);
return;
}

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

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;
}

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,
            NON_POSITIVE,
            title,
            JOptionPane.ERROR_MESSAGE);
        isOK = false;
    }
    if (isOK) {
        return newValue;
    } else {
        field.setText(numberFormatter.format(oldValue));
        return oldValue;
    }
}

// Reads integer numbers
private int readInt(JTextField field,
    int oldValue,
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,
    NON_POSITIVE,
    title,
    JOptionPane.ERROR_MESSAGE);
    isOK = false;
}

if (isOK) {
    return newValue;
} else {
    field.setText(numberFormatter.format(oldValue));
    return oldValue;
}

double moro_inv(double u)
{
    /* returns the inverse of cumulative normal distribution function */
    Reference> The Full Monte, by Boris Moro, RISK 1995(2)*/
    double a[]={2.5066282823884,
                -18.61500062529,
                41.39119773534,
                -25.44106049637};
    double b[]={-8.47351093090,
                 23.08336743743,
                 -21.06224101826,
                 3.13082909833};
    double c[]={0.3374754822726147,
                 0.9761690190917186,
                 0.1607979714918209,
                 0.0276438810333863,
                 0.0038405729373609,
                 0.000321767881768,
                 0.0000002888167364,
                 0.0000003960315187};
    double x, r;
x = u - 0.5;
    if (Math.abs(x) < 0.42) {

Pricing Asian options under Heston’s model using the new Ninomiya weak approximation scheme

r = x * x;
r = x * (((a[3] * r + a[2]) * r + a[1]) * r + a[0]) /
(((b[3] * r + b[2]) * r + b[1]) * r + b[0]) * r + 1.0;
}
else {
    if (x < 0.0) r = u;
    else r = 1 - u;
    r = Math.log(-1 * (Math.log(r)));
        r * (c[7] + r * c[8]))))))));
    if (u < 0.5) r = -r;
}
return 0.0 + r * 1.0;
}

public double GetX2n() { // simulate 1 trajectory and return option price
    yOld[0] = stockprice; // initialize Yn[0]=y0
    yOld[1] = volatility;
    yOld[2] = 0;
    double result = 0;
    for (int i = 0; i < dim; i++) {
        double[] newXi = nxs.nextPoint();
        for (int j = 0; j < 4; j++) {
            newXi[j] = moro_inv(newXi[j]);
        }
        fSingle.setXi(newXi);
        rk7(yOld, 0, 1.0, fSingle, 2, yNew);
        System.arraycopy(yNew, 0, yOld, 0, 3);
        rk7(yOld, 0, 1.0, fSingle, 1, yNew);
        System.arraycopy(yNew, 0, yOld, 0, 3);
    }
    result = yOld[2];
    yOld[0] = stockprice; // initialize Yn[0]=y0
    yOld[1] = volatility;
    yOld[2] = 0;
    for (int i = 0; i < 2 * dim; i++) {
        double[] xiNew = nxs.nextPoint();
        for (int j = 0; j < 4; j++) {
            xiNew[j] = moro_inv(xiNew[j]);
        }
        fDouble.setXi(xiNew);
        rk7(yOld, 0, 1.0, fDouble, 2, yNew);
        System.arraycopy(yNew, 0, yOld, 0, 3);
        rk7(yOld, 0, 1.0, fDouble, 1, yNew);
        System.arraycopy(yNew, 0, yOld, 0, 3);
    }
    result = 4.0 * result / 3.0 - yOld[2] / 3.0; // Romberg extrapolation
    result = Math.max(result / maturity - strike_K, 0) * (Math.exp(-interest *
        maturity));
    return result;
}
public void rk7(double[] y,
    double t,
    double h,
    Derivs f,
    int index,
    double[] yOut) {

    // Given values for the variables y[N−1] known at t, use the seventh-order Runge–Kutta
    // method to advance the solution over an interval h and return the incremented
    // variables as yOut[N−1].
    final int M = 9;
    // Runge–Kutta 7th order coefficients
    final double[][] A = {
        {},
        {1.0/6.0},
        {0, 1.0/3.0},
        {1.0/8.0, 0, 3.0/8.0},
        {148.0/1331.0, 0, 150.0/1331.0, −56.0/1331.0},
        {−404.0/243.0, 0, −170.0/27.0, 4024.0/1701.0,
         10648.0/1701.0},
        {2466.0/2401.0, 0, 1242.0/343.0, −19176.0/16807.0,
         −51909.0/16807.0, 1053.0/2401.0},
        {5.0/154.0, 0, 96.0/539.0, −1815.0/20384.0,
         49.0/1144.0},
        {−113.0/32.0, 0, −195.0/22.0, 32.0/7.0,
         29403.0/3584.0, −729.0/512.0, 1029.0/1408.0, 21.0/16.0}},
    final double[] B =
        {0, 0.0, 32.0/105.0, 1771561.0/6289920.0, 243.0/1560.0, 16807.0/74880.0, 77.0/17.0};

    int n = y.length;
    for (int i=0; i<M; i++) {
        System.arraycopy(y, 0, Y[i], 0, n);
        for (int j=0; j<i; j++) {
            for (int k=0; k<n; k++) {
                Y[i][k] += h*A[i][j]*W[j][k];
            }
        }
        if (index == 1) {
            f.getW1(Y[i],W[i]);
        } else {
            f.getW2(Y[i],W[i]);
        }
    }
    System.arraycopy(y, 0, yOut, 0, n);
    for (int i=0; i<M; i++) {
        for (int k=0; k<n; k++) {
            yOut[k] += h*B[i]*W[i][k];
        }
    }
}
Pricing Asian options under Heston’s model using the new Ninomiya weak approximation scheme

Boyko Vasilev

Mälardalen University
Appendix B

Source-code listing for Derivs.java

```java
public class Derivs {
    private double s = 0;
    private double r = 0;
    private double rho = 0;
    private double beta = 0;
    private double alpha = 0;
    private double theta = 0;

    private double sqrtS = 0;

    private final int D = 2;
    private final int N = 2;

    private final double C1 = Math.sqrt(3.0) / 2.0;
    private final double C2 = 5.0 * Math.sqrt(3.0) / 6.0;
    private final double C3 = Math.sqrt(6.0) / 3.0;

    private double[] v0 = new double[D + 1];
    private double[] v1 = new double[D + 1];
    private double[] v2 = new double[D + 1];

    // contains D*N independent Gaussian random variables
    private double[] xi = null;

    public Derivs(double s,
                   double r,
                   double rho,
                   double beta,
                   double alpha,
                   double theta) {
        this.s = s;
        this.r = r;
        this.rho = rho;
        this.beta = beta;
        this.alpha = alpha;
        this.theta = theta;

        sqrtS = Math.sqrt(s);

        v1[2] = 0;
    }
```
Pricing Asian options under Heston’s model using the new Ninomiya weak approximation scheme

```java
public void setXi(double[] newXi) {
    xi = newXi;
}

private void getV(double[] y) {
    // Initial values
    v0[0] = r;
    v0[0] -= y[1]/2;
    v0[0] -= rho*beta / 4.0;
    v0[0] *= y[0];

    v0[1] = theta-y[1];
    v0[1] *= alpha;
    v0[1] -= beta*beta / 4.0;

    v0[2] = y[0];

    double s = Math.sqrt(y[1]);

    v1[0] = y[0]*s;
    v1[1] = rho*beta*s;

    v2[1] = beta*Math.sqrt(1.0-rho*rho)*s;
}

public void getW1(double[] y, double[] w1) {
    getV(y);
    for (int i=0; i<D+1; i++) {
        w1[i] = -s*v0[i]/2.0;
        w1[i] += C1*xi[0]*sqrtS*v1[i];
        w1[i] += C1*xi[2]*sqrtS*v2[i];
    }
}

public void getW2(double[] y, double[] w2) {
    getV(y);
    for (int i=0; i<D+1; i++) {
        w2[i] = 3.0*s*v0[i]/2.0;
        w2[i] -= C2*xi[0]*sqrtS*v1[i];
        w2[i] += C3*xi[1]*sqrtS*v1[i];
        w2[i] -= C2*xi[2]*sqrtS*v2[i];
        w2[i] += C3*xi[3]*sqrtS*v2[i];
    }
}
```

Boyko Vasilev

Mälardalen University
Appendix C

Source-code listing for NX.java

class NX{
   /* WARRANTY NOTICE AND COPYRIGHT
   This program is free software; you can redistribute it and/or
   modify it under the terms of the GNU General Public License
   as published by the Free Software Foundation; either version 2
   of the License, or (at your option) any later version.

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

   You should have received a copy of the GNU General Public License
   along with this program; if not, write to the Free Software
   Foundation, Inc., 59 Temple Place – Suite 330, Boston, MA
   02111−1307, USA.

   Copyright (C) Michael J. Meyer

   matmj@mindspring.com
   spyqqqdia@yahoo.com

   */

   /*
   * NX.java
   *
   * Created on May 23, 2002, 3:59 PM
   */

   static final int M=30; // we are using 30 bit integers
   static final int N=1073741842; // 2^30

   int [] x_int; // current vector of NX integers
   int nxdim; // dimension
   int index; // index of current point in sequence

   double [] x; // current uniform point
   double [] z; // current quasi normal transform of x
Pricing Asian options under Heston’s model using the new Ninomiya weak approximation scheme

/** genMats[j] is the column encoding of the generator matrix C(j) for
coordinate j in the current dimension: for 0<=k<M
genMats[j][k] is the kth column col_k of C(j) encoded as a
decimal integer with binary digits the entries of col_k increasing in
significance from the bottom up. */

int[][] genMats =
{
    // C(0), columns:
    
    // C(1), columns:
    
    // C(2), columns:
    
    // C(3), columns:

    public NX(int nxdim) { // construct NX low discrepancy sequence in
dimension dim.

        this.nxdim=nxdim;
        index=1;
        x=new double[nxdim];
        z=new double[nxdim];

        // array of current nx-integers
        x_int=new int[nxdim];
        for(int j=0;j<nxdim;j++)x_int[j]=genMats[j][0];
    } // end constructor
```java
public void restart() // NX points restart
{
    index = 1;
    // return the integer vector to the initial state
    for (int j = 0; j < nxdim; j++)
        x_int[j] = genMats[j][0];
}

/**
 * The next nx point in the unit cube [0,1]^dim.
 */
public double[] nextPoint()
{
    // find the position k of the rightmost zero bit of index
    // 0<=k<M
    int k = 0, n = index;
    while (n % 2 == 1)
    {
        n = n >> 1;
        k ++;
    }
    for (int j = 0; j < nxdim; j++)
    {
        x_int[j] ^= genMats[j][k];
        x[j] = ((double)x_int[j]) / N;
    }
    index ++;
    return x;
}
```