CHALMERS
UNIVERSITY OF TECHNOLOGY

Quadratic Volatility Models Applied to the Pricing of European Options

Master's thesis in Engineering Mathematics and Computational Science

GUSTAV LINDWALL

Department of Mathematical Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

Master's thesis 2018

Quadratic Volatility Models Applied to the Pricing of European Options

GUSTAV LINDWALL

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Division of Analysis and Probability Theory
Chalmers University of Technology
Gothenburg, Sweden 2018

Quadratic Volatility Models Applied to the Pricing of European Options
GUSTAV LINDWALL

© GUSTAV LINDWALL, 2018.

Supervisor: Docent Patrik Albin, Mathematical Sciences
Examiner: Docent Patrik Albin, Mathematical Sciences

Master's Thesis 2018
Department of Mathematical Sciences
Division of Analysis and Probability Theory
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2018

Abstract

In this thesis we derive a general framework for calibrating quadratic local volatility models in financial asset modelling. The method is first considered for constructed fictional data sets. Strengths and weaknesses of the method are studied thoroughly in this setting. We then apply our calibration method on stock market data, and use it to price European call options. The results of this are compared to actual option chains on the stocks in question as well as the cruder Black-Scholes prices for these stocks. We end the thesis with a discussion on further development of the quadratic volatility model.

Acknowledgements

I would like to extend my thanks to my advisor, Patrik Albin, for his expertise and guidance, without which this thesis would not have been possible. I would also like to thank friends and family for supporting me during the thesis work. Last but not least, I would like to thank my cats, Per and Irma, for keeping me company in the many long hours of writing from home.

Contents

Contents **9**

1 Introduction **1**

2 Theoretical prerequisities **2**
2.1 Stochastic calculus 2
2.2 Financial derivatives pricing 3

3 The quadratic volatility function **5**
3.1 Transformation of the SDE 5
3.2 Girsanov's theorem 6
3.3 Two real roots 6
3.4 Two complex roots 8
3.5 One real root 9

4 Parameter fitting for QV-models **10**
4.1 Particle swarm optimization 11
4.2 Single root configuration 11

5 Testing of the method on sample cases **12**
5.1 Geometrical properties of objective function depending on data 12
5.2 Test for two real roots, case 1 16
5.3 Test for two real roots, case 2 17
5.4 Test for two complex roots 18
5.5 Test for a single real root 19
5.6 Test conclusion 20

6 Option pricing using QV-models **22**

7 Application real data **23**

8 Conclusion **27**
8.1 Performance of parameter estimation program 27
8.2 Quality of real life applications 27
8.3 Possible expansions of the model and future work 27

Bibliography **29**

A Matlab code for parameter calibration **30**
A.1 Main.m 30
A.2 Optimization.m 31
A.3 GenerateSwarm.m 33
A.4 EvaluateIndividual.m 34
A.5 Constraints.m 35
A.6 PlotQuadraticVariation.m 36

A.7 FunctionString.m 36
A.8 OptimizationSingleRoot.m 37
A.9 GenerateSwarmSingleRoot.m 39
A.10 EvaluateIndividualSingleRoot.m 39
A.11 ConstraintsSingleRoot.m 40
A.12 PlotQuadraticVariationSingleRoot.m 40
A.13 OptimizationBlackScholes.m 41
A.14 PlotQuadraticVariationBS.m 41

B Code for pricing call option using PDE **42**

1 Introduction

For the past four decades, the mathematical modelling of stock markets have undergone rapid development. Individual stocks are modelled according to stochastic differential equations on the form

$$dX(t) = \mu(X(t), t)dt + \sigma(X(t), t)dB(t),$$

where μ is a function describing the deterministic time evolution of the stock, and σ is a function describing how the stock reacts to a driving noise. The most common and basic noise utilized is Brownian motion. The simplest model of a stock, as well as the back-bone of quantitative finance, is the Black-Scholes model

$$dX(t) = \mu_{BS}xdt + \sigma_{BS}xdB(t), \tag{1.1}$$

where μ_{BS} is a constant and $\sigma_{BS} > 0$ is a constant. The solution to this stochastic differential equation is the Geometric Brownian motion, given as

$$X(t) = e^{(\mu - \frac{\sigma^2}{2})t + \sigma B(t)}.$$

There are multiple problems with modelling stocks as Geometric Brownian motions, the chief issue being that it does not capture the volatile nature of stocks to full extent. To combat this, modifications to the Black-Scholes recipie have been proposed over the years. This paper is concerned with *local volatility modelling*, where the function σ only depends on $X(t)$ but takes on a more complex form than in (1.1). With this, researchers within the field of finance hope to more accurately model the random ups and downs of stock values. The particular model of our study is the *quadratic volatility* model, introduced by Lipton et al in 2001 [5], but most extensively studied by Andersen in 2008 [3]. His paper is of a theoretical character, were formulas for pricing European options given a quadratic volatility model are derived and thoroughly studied. However, calibration of this model to real world data is left out. Some papers on the subject has been written, for example Chibane et al in 2012 [6], but the performance of these calibration on the options market has as of 2017 not been studied thoroughly. In this thesis we will study the volatility function as proposed by Andersen,

$$\sigma(x) = \sigma_0((1-q)x_0 + qx + \frac{s}{2}(x - x_0)^2).$$

We will calibrate it to real world data using stochastic optimization and price European options using these parameters. Furthermore, we will explore shortcomings of the model and discuss modifications to better adapt it to market prices.

2 Theoretical prerequisities

The Black-Scholes framework of financial mathematics stands on a foundation of stochastic calculus. Before dealing with the actual problem of parameter estimation for the quadratic volatility model, we shall refresh a few key definitions and theorems from stochastic calculus. All of the results in this chapter can be found with proofs in [1].

2.1 Stochastic calculus

Stochastic calculus is the branch of mathematics pertaining the differentiation and integration of stochastic processes. Continuous-time stochastic processes possesses properties making standard analysis insufficient. This chapter will briefly cover the most important parts of stochastic calculus with applications to finance. Throughout this report, we assume that we have a filtered probability space $(\Omega, \mathcal{F}, \mathcal{F}_t, \mathbb{P})$ readily available, where the filtration $\mathcal{F}_t$ is generated by a Brownian motion $B(t)$.

Definition 2.1. (Brownian motion) Brownian motion, also known as a Wiener process, is a stochastic process satisfying the following properties:

- Independence of increments. $B(t) - B(s)$, for $t - s$, is independent of the past. This means that the σ-field $\mathcal{F}_s$ generated by the Brownian motion up until the time s is independent of the σ-field generated by the random variable $B(t) - B(s)$.
- Normal increments. The increment $B(t) - B(s)$ is a random variable with distribution $\mathcal{N}(0, t - s)$.
- Continuity of paths. $B(t)$, $t \geq 0$ are continuous functions of t.

Definition 2.2. (Martingale) A stochastic process $X(t)$ is called a martingale if

$$\mathbb{E}[X(t)|\mathcal{F}_s] = X(s), \qquad t > s.$$

Definition 2.3. (Quadratic variation) Let $f(t)$ be a right-continuous function of one real variable. The quadratic variation $[f, f](t)$ of $f(t)$ is defined as

$$[f, f](t) = \lim_{\delta_n \to 0} \sum_{i=1}^{n} |f(t_i^n) - f(t_{i-1}^n)|^2,$$

where the limit is taken over partitions $0 = t_0^n < t_1^n < \cdots < t_n^n = t$, with $\delta_n = \max_{1 \leq i \leq n}(t_i^n - t_{i-1}^n)$.

Definition 2.4. (Stochastic differential equation) Let $B(t)$, $t \geq 0$ be a Brownian motion process. An equation on the form

$$dX(t) = \mu(X(t), t)dt + \sigma(X(t), t)dB(t),$$

where functions $\mu(x, t)$ and $\sigma(x, t)$ are given and $X(t)$ is the unkown process is called a stochastic differential equation driven by Brownian motion. The functions $\mu(x, t)$ and $\sigma(x, t)$ are known as the *drift* and *diffusion* coefficients, respectively.

2.2 Financial derivatives pricing

We now let $X(t)$ note the value of a stock at time t and assume that $X(t)$ follows some model

$$dX(t) = \mu(X(t))dt + \sigma(X(t))dB(t), \tag{2.1}$$

In applications, we are interested in the price of a stock up to a time $T > 0$, meaning that in (2.1), the stochastic processes $X(t)$, μ and σ should be adapted to the filtrations $\mathcal{F}_t = \sigma(B(s), s \leq t)$ of the Borel σ-algebra. Intuitively, we say that any outfall of the stock price at the future time T can be *anticipated* by all possible trajectories of the Brownian motion $B(t)$ at the current time $t < T$ up until the time T, with better predictions being possible for t close to T. The initial value of the stochastic differential equation is a random variable X_0, which is also measurable with respect to $\mathcal{F}_t = \sigma(B(s), s \leq t)$.

Note that (2.1) is an abuse of notation; the correct way to interpret this equation is by integrating both sides. Thus, we have the formal definition

$$X(t) = X_0 + \int_0^t \mu(X(s))\mathrm{ds} + \int_0^t \sigma(X(s))dB(s).$$

The integral with respect to the Brownian motion process shall be percieved as an Itô-integral. In complement to this stochastic process representing the stock price, we also have the *risk-free asset* $\beta(t)$ governed by

$$\beta(t) = \beta(0)\exp(\int_0^t r(s)\mathrm{ds}), \qquad t \in [0, T].$$

Here, $r(t)$ is a positive stochastic process adapted to $\mathcal{F}_t = \sigma(B(s), s \leq t)$ that represents the *instantaneous interest rate* of the risk-free asset. For most applications, it is enough to assume that $r(t) = r$, a deterministic constant, as the maturity time of financial derivatives is usually not too large. Thus the interest rate can be assumed to not change by very much.

In our analysis, we are not interested in the stock-specific time-dependent dynamics governed by $\mu(X(t))$, but rather the behaviour of the stock in *risk-neutral probability measure*. The existence of such a measure is dependent on the *Novikov condition*, which we will now remind us of.

Lemma 2.1. *(The Novikov condition) Let $\{\theta(t)\}_{t\geq 0}$ be an Itô process satisfying*

$$\mathbb{E}\big[e^{\frac{1}{2}\int_0^T \theta^2(t)dt}\big].$$

Then the stochastic process

$$Z(t) = e^{-\int_0^t \theta(s)dB(s) - \frac{1}{2}\int_0^t \theta(t)^2 ds}$$

is a martingale relative to the filtration $\mathcal{F}_t = \sigma(B(s), s \leq t)$

Lemma 2.2. *(Equivalent martingale measure) The measure $\tilde{\mathbb{P}}(A)$ defined by*

$$\tilde{\mathbb{P}}(A) = \mathbb{E}[Z(T)\mathbb{I}_A]$$

is equivalent to the original probability measure $\mathbb{P}(A)$.

Theorem 2.3. *(Girsanov) Let $B(t)$ be a $\mathbb{P}$-Brownian motion and $\theta(t)$ be as in lemma 2.1. Under the probability measure $\tilde{\mathbb{P}}(A)$, the stochastic process*

$$\tilde{B}(t) = B(t) + \int_0^t \theta(s)ds$$

is a standard Wiener process.

Formally, we observe the stock in the risk-neutral measure by setting $\theta(t) = (r(t)X(t) - \mu(X(t)))/\sigma(X(t))$ and assume, for now, that this process satisfies the Novikov condition. With $\tilde{B}(t)$ being our driving Brownian motion, we are interested in a simplified stock model

$$dX(t) = rX(t)dt + \sigma(X(t))d\tilde{B}(t)$$

where $r(t) = r$, a constant. We will use this representation of $X(t)$ in the risk-neutral measure to derive equations that will give us fair pricing formulas for European call and put options.

Assume first that such functions indeed exists and depends on $X(t)$ and t. Let us define this function as

$$f(x,t) = \tilde{\mathbb{E}}(g(X(T)|X(t) = x),$$

the expected value of the pricing function $g(x)$ of the stock in the risk-neutral measure. We then perceive the price of our option as a stochastic process $\Pi(t) = f(X(t), t)$. A straightforward application of the towering property of expectations give us that the *discounted* price process

$$\Pi^*(t) = \exp(-\int_0^t rds)\Pi(t)$$

is a martingale in the risk-neutral measure. We call $D(t) = \beta^{-1}(t) = \exp(-\int_0^t rds)$ the discount process. We now use Itô's formula in two variables on $\Pi^*(t)$ and see that

$$\begin{aligned} d(\Pi^*(t)) &= D(t)d\Pi(t) + \Pi(t)dD(t) + dD(t)d\Pi(t) \\ &= D(t)\big[\partial_t f(x,t) + rx\partial_x f(x,t) + \frac{1}{2}\sigma^2(x)\partial_x^2 f(x,t) - rf(x,t)\big]_{\{X(t)=x\}}dt \\ &\quad + D(t)\sigma(X(t))\partial_x f(X(t),t)d\tilde{B}(t). \end{aligned}$$

Since this process is a known martingale, it must have drift zero. This means that the fair price function must satisfy the partial differential equation

$$\partial_t f(x,t) + rx\partial_x f(x,t) + \frac{\sigma^2(x)}{2}\partial_x^2 f(x,t) - rf(x,t) = 0$$

with terminal condition $f(x,T) = e^{-rT}g(x)$. A quick change of variable $\tau = T - t$ along with setting $u(x,\tau) = e^{r\tau}f(x,t)$ gives us a homogeneous PDE with an initial condition,

$$-\partial_\tau u(x,\tau) + rx\partial_x u(x,\tau) + \frac{\sigma^2(x)}{2}\partial_x^2 u(x,\tau) = 0, \qquad x > 0, \quad \tau \in [0,T], \tag{2.2}$$

$$u(0,\tau) = g(0), \tag{2.3}$$

$$u(x,0) = g(x). \tag{2.4}$$

Numerical solution methods for partial differential equations are abound. We thus price our asset by solving this equation.

3 The quadratic volatility function

A stock $X(t)$ is in the risk neutral measure described by the stochastic differential equation

$$dX(t) = r(t)X(t)dt + \sigma(X(t))dB(t).$$

We will work with a general interest-rate model $r(t)$ in this treatise. The volatility function $\sigma(X(t))$ will be defined as

$$\sigma(x) = \sigma_0((1-q)x_0 + qx + \frac{s}{2x_0}(x-x_0)^2) \tag{3.1}$$

Here σ_0 is analouge to the fixed volatility in the Black-Scholes model, q is a "skew" parameter and s is a measure of the convexity of $\sigma(x)$. x_0 is the value of the stock at time $t = 0$. In order to model a tangible random phenomena, we are to impose some restrictions on the root configuration on (3.1). First we note that the roots of (3.1) are located at

$$x_{1,2} = \frac{x_0}{s}\big((s-q) \pm \sqrt{q^2 - 2s}\big).$$

We can thus conclude that the polynomial have two distinct real roots if $q^2 > 2s$, one single real root if $q^2 = 2s$ and no real root if $q^2 < 2s$. For the sake of simplicity, we will from here on out define the lower and upper roots as:

$$u := \frac{x_0}{s}\big((s-q) + \sqrt{q^2 - 2s}\big)$$
$$l := \frac{x_0}{s}\big((s-q) - \sqrt{q^2 - 2s}\big)$$

3.1 Transformation of the SDE

In order to fit the parameters σ_0, s and q to real data we need to first introduce the function $\Sigma(x)$, given as

$$\begin{aligned}\Sigma(x) &= \int \frac{dx}{\sigma_0((1-q)x_0 + qx + \frac{s}{2x_0}(x-x_0)^2)} \\ &= \frac{2}{\sigma_0\sqrt{2s-q^2}} \arctan\Big(\frac{qx_0 + s(x-x_0)}{x_0\sqrt{2s-q^2}}\Big)\end{aligned}$$

We now consider a transformed stochastic process $Y(t) = \Sigma(X(t))$. From Itô's formula we get that $Y(t)$ is satisfied by the SDE

$$dY(t) = \Sigma'(X(t))dX(t) + \frac{1}{2}\Sigma''(X(t))d[X,X](t) \tag{3.2}$$
$$= \frac{1}{\sigma(X(t))}\big(r(t)X(t)dt + \sigma(X(t))dB(t)\big) + \frac{1}{2}\big(\frac{1}{\sigma(X(t))}\big)'\sigma^2(X(t))dt \tag{3.3}$$
$$= \big(\frac{r(t)X(t)}{\sigma(X(t))} - \frac{\sigma'(X(t))}{2}\big)dt + dB(t) \tag{3.4}$$
$$= R_\Sigma(Y(t))dt + dB(t) \tag{3.5}$$

This means that the diffusive part of $Y(t)$ is simply a Brownian motion, and that if the drift process $R_\Sigma(Y(t))$ satisfies the Novikov condition, we can regard $Y(t)$ as a Brownian motion under the new measure as specified in Girsanov's theorem. We can then use the fact that under equivalent measure changes, the quadratic variation of a process does not change. If such a change of measure is possible, it is certain that $[Y,Y](t) = t$.

3.2 Girsanov's theorem

We will now study the process $R_\Sigma(Y(t))$ with greater care. First, we express $X(t)$ as the inverse process of $Y(t)$, i.e $X(t) = \Sigma^{-1}(Y(t))$. Tedious but standard calculations leads to that

$$\Sigma^{-1}(x) = \frac{x_0}{s}\big(\sqrt{2s-q^2}\tan\big(\frac{\sigma_0\sqrt{2s-q^2}}{2}x\big) + (s-q)\big).$$

This in turns leads us to

$$X(t) = \frac{x_0}{s}\big(\sqrt{2s-q^2}\tan\big(\frac{\sigma_0\sqrt{2s-q^2}}{2}Y(t)\big) + (s-q)\big).$$

$\tan(x)$ is a horrible function which is garantueed to explode in finite time. However, in real-life applications we note that since $\sigma_0\sqrt{2s-q^2}Y(t) < \pi$, this will never happen. The analysis can thus be fully carried out with the expression

$$R_\Sigma(Y(t)) = \frac{r(t)X(t)}{\sigma(X(t))} + \sigma'(X(t)) := \theta(t).$$

We note that the only possible pitfall here is in the case of real roots, and that is if the stock $X(t)$ would hit any of the roots of $\sigma(x)$. For all other cases, as long as $r(t)X(t)$ behaves even remotely nicely, the Novikov condition is satisfied. There exist a measure as in Lemma 2.2 in which $Y(t)$ is a Brownian motion. We stress again that this is because $\sigma_0\sqrt{2s-q^2}Y(t) < \pi$ in our application.

3.3 Two real roots

As seen in the analysis above, the case of real roots presents a problem that is absent in the case of complex roots. Not only can crossing the roots lead to a violation of the Novikov condition. Essentially, the quadratic volatility model allows us to reach negative volatility unless the parameters σ_0, s and q are such that for a stock time series $\mathbf{x}$ we have:

1. $u \leq \min(\mathbf{x})$,
2. $l \leq \min(\mathbf{x}) < \max(\mathbf{x}) \leq u$.

These are the exact same restrictions as we arrived at when studying $\theta(t)$. We also note that this is not a problem if the polynomial (3.1) would have complex roots.

The two cases 1. and 2. correspond to the two possible profiles of $\sigma(x)$. 1. will correspond to a "smiling" volatility curve, where the feasible set of stock values will be $X(t) \in [u, \infty]$. 2. will correspond to a "frowning" volatility curve, and admit stock values in the range $X(t) \in [l, u]$. These two possible profiles for two real roots are exemplified in Figure 3.1.

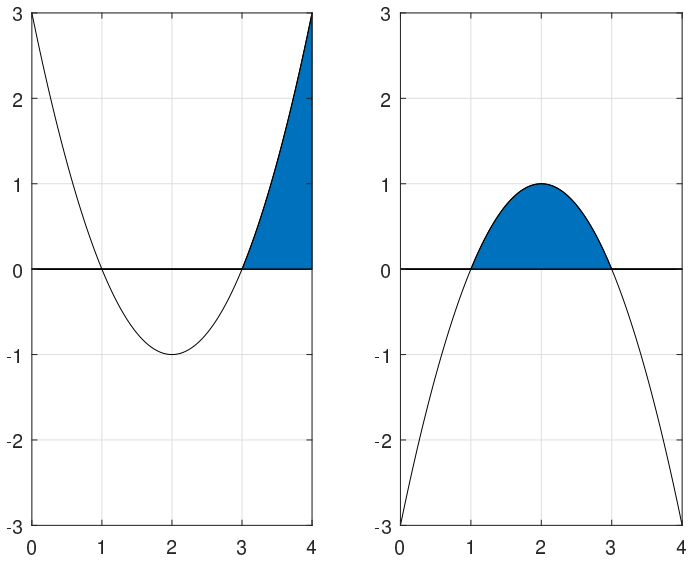


Figure 3.1: The possible profiles for $\sigma(x)$ with two distinct real roots. The admissable stock values are marked with blue. Here $l = 1$, $u = 3$ in both cases.

However, in a realistic model we should allow the stock value to reach zero. This is also important for future applications regarding pricing the stock. Thus we should impose further restrictions on the parameters s and q in the case of real roots.

In case one, we only accept stock values higher than u. In order for this to work for low stock values, we must have $u < 0$. We can write this in terms of s and q: if $q^2 - 2s > 0$ and $s > 0$, then $(s-q)+\sqrt{q^2-2s} < 0$ must hold. This only allows a tiny part of the (s, q)-plane to be admissable solutions, visualized in Figure (ref goes here).

In case two, we note that $l < 0$ must hold. We must also have that $u > 0$, since otherwise no stock values would be admissable. Put in terms of s and q: if $q^2 - 2s > 0$ and $s < 0$, then $(s - q) - \sqrt{q^2 - 2s} < 0$ and $(s - q) + \sqrt{q^2 - 2s} > 0$ must hold.

We finish this treatise by adressing what happens to $\Sigma(x)$ when $q^2 - 2s > 0$. We note that whenever this is the case, we have

$$\Sigma(x) = \frac{-2i}{\sigma_0\sqrt{q^2-2s}} \arctan\Big(\frac{-i(qx_0 + s(x - x_0))}{x_0\sqrt{q^2-2s}}\Big)$$

Now we use the identity $\operatorname{arctanh}(z) = -i \arctan(iz)$ and thus we have that

$$\Sigma(x) = \frac{2}{\sigma_0\sqrt{q^2-2s}} \operatorname{arctanh}\Big(\frac{-(qx_0 + s(x - x_0))}{x_0\sqrt{q^2-2s}}\Big).$$

Let us rewrite this as a function of something more familiar. By definition we have that

$$\operatorname{arctanh}(\mathrm{x}) = \frac{1}{2}\big(\log(1 + x) - \log(1 - x)\big), \quad x \in (-1, 1).$$

We note that an imaginary argument get added for $x \notin (-1,1)$, which is undesirable. Keeping $x \in (-1,1)$ let us stay on the *principal branch* of this multivalued function. We have, with $\sqrt{q^2-2s} = \eta$,

$$\begin{aligned}
\Sigma(x) &= \frac{1}{\sigma_0\eta}\Big(\log\Big(1 - \frac{qx_0 + s(x-x_0)}{x_0\eta}\Big) - \log\Big(1 + \frac{qx_0 + s(x-x_0)}{x_0\eta}\Big)\Big) \\
&= \frac{1}{\sigma_0\eta}\Big(\log\Big(\frac{x_0(\eta - q + s) - sx}{x_0\eta}\Big) - \log\Big(\frac{x_0(\eta + q - s) + sx}{x_0\eta}\Big)\Big) \\
&= \frac{1}{\sigma_0\eta}\Big(\log\Big(\frac{x_0(\eta - q + s)}{s} - x\Big) - \log\Big(\frac{x_0(\eta + q - s)}{s} + x\Big)\Big) \\
&= \frac{1}{\sigma_0\eta}\Big(\log(u-x) - \log(x-l)\Big) \\
&= \frac{1}{\sigma_0\eta}\Big(\log\Big(\frac{u-x}{l-x}\Big) - \pi i\Big).
\end{aligned}$$

We note that in order to keep this expression real, we need to be on the non-principal branch of the logarithmic function. This will happen whenever $x \in (l,u)$. To show this, let $0 < y < u - l$ and set $x = l + y$. We get

$$\begin{aligned}
\Sigma(l+y) &= \frac{1}{\sigma_0\eta}\Big(\log\Big(\frac{u-l-y}{-y}\Big) - \pi i\Big) \\
&= \frac{1}{\sigma_0\eta}\Big(\log\Big(\frac{(l-u)+y}{y}\Big) - \pi i\Big) \\
&= \frac{1}{\sigma_0\eta}\log\Big(\frac{(u-l)-y}{y}\Big)
\end{aligned}$$

which is real for all allowed y. We see that this corresponds exactly to case 2. For case 1 we will need a more careful approach. Consider the function

$$\tilde{\Sigma}(x) = \frac{1}{\sigma_0\sqrt{q^2-2s}}\log\Big(\frac{x - \frac{x_0}{s}((s-q) + \sqrt{q^2-2s})}{x - \frac{x_0}{s}((s-q) - \sqrt{q^2-2s})}\Big).$$

We note that $\tilde{\Sigma}(X(t)) = Y(t)$. This follows from the easily verified fact that $\tilde{\Sigma}'(x) = \sigma(x)$ and we carry out the same calculations as in equation (3.2)-(3.5). The chief difference between $\tilde{\Sigma}(x)$ and $\Sigma(x)$ is a complex modulus of $2\pi i$. $\tilde{\Sigma}(x)$ stays on the principal branch of the logarithm for $x \in)l,u($. Disregarding this, we have that $\Re(\Sigma(x)) = \Re(\tilde{\Sigma}(x))$ for all $x \in \mathbb{R}$. We thus redefine $\Sigma(x)$ as

$$\Sigma(x) = \Re\left(\frac{2}{\sigma_0\sqrt{2s-q^2}}\arctan\Big(\frac{qx_0 + s(x-x_0)}{x_0\sqrt{2s-q^2}}\Big)\right)$$

and note that this is the same as rotating to the principal branch of the complex logarithm in the case of real roots. We have thus achieved what we set out to find - a real-valued function that transforms $X(t)$ into a Brownian motion with drift.

3.4 Two complex roots

Here instead we impose that $\sigma(x)$ must be positive for all x, something that is only true if $s > 0$. A smiling second-degree polynomial with complex roots will be positive everywhere. Any combination of σ_0, q and s fullfilling $q^2 - 2s < 0$, $s > 0$ will thus do.

3.5 One real root

This case stands out from the two other. First we note that one real root corresponds to $q^2 = 2s$, and thus $\Sigma(x)$ is singular for this parameter choice. This means that another transformation must be chosen for this case if we are to use a similar method as in the other cases. Setting $s = q^2/2$ the polynomial becomes

$$\sigma(x) = \frac{\sigma_0 q^2}{4x_0}\Big(x - x_0\big(\frac{q-2}{q}\big)\Big)^2$$

and the corresponding transformation function is

$$\Sigma(x) = \int \frac{4x_0}{\sigma_0 q^2\big(x - x_0\big(\frac{q-2}{q}\big)\big)^2}dx = -\frac{4x_0}{\sigma_0 q^2\Big(x - x_0\big(\frac{q-2}{q}\big)\Big)} \tag{3.6}$$

Let us by $c = x_0(q-2)/q$ denote the single root of the polynomial. Redoing the calculations of (3.2) − (3.5), we arrive at

$$dY(t) = \Big(\frac{r(t)X(t)}{\sigma(X(t))} - \frac{\sigma'(X(t))}{2}\Big)dt + dB(t)$$

and can once again conclude that as long as $X(t) > c$ a change of measure is possible by Girsanov's theorem. We also must have that $q \leq 2$ in order to keep the stock price positive. There are no other restrictions on the parameters, as we must have that both x_0 and σ are greater than zero by default. The risks of having to handle complex arguments associated with the two real roots is not present either.

4 Parameter fitting for QV-models

We now present the optimization problem that is the centerpiece of this report. We have shown that the process $Y(t)$ can be made a Brownian motion by a change of probability measure. We will invoke this measure, and infer that the quadratic variation of of $Y(t)$ must be that of a Brownian motion, $[B, B](t) = t$. If we are to fit a quadratic volatility to some data set $\mathbf{x}$, we are tasked with the following problem:

Let $\mathbf{x}_t$ be a time series, and assume that $\mathbf{x}_t$ are discrete samples from a stochastic process

$$X(t) = X_0 + \int_0^t \sigma(X(s))dB(s)$$

where $\sigma(x) = \sigma_0((1-q)x_0 + qx + \frac{s}{2x_0}(x-x_0)^2)$. Let us further define a function

$$\Sigma(x) = \Re\left(\frac{2}{\sigma_0\sqrt{2s-q^2}}\arctan\left(\frac{qx_0 + s(x-x_0)}{x_0\sqrt{2s-q^2}}\right)\right)$$

and set $Y(t) = \Sigma(X(t))$. Find σ_0, s and q such that we minimize the quantity

$$|[Y,Y](t) - t|^2.$$

This problem comes with a plethora of case-sensitive restrictions. In one line, we can write the restrictions for case 1, two real roots and a smiling polynomial, as

$$h_1(\sigma_0, s, q) = \mathbb{I}_{(q^2-2s>0)}\mathbb{I}_{(s>0)}\left((s-q) + \sqrt{q^2-2s}\right) \leq 0.$$

We can formulate the second case, two real roots and a frowning polynomial, as the conditions

$$h_2(\sigma_0, s, q) = \mathbb{I}_{(q^2-2s>0)}\mathbb{I}_{(s<0)}\left((s-q) - \sqrt{q^2-2s}\right) \leq 0,$$
$$h_3(\sigma_0, s, q) = \mathbb{I}_{(q^2-2s>0)}\mathbb{I}_{(s<0)}\left((q-s) - \sqrt{q^2-2s}\right) \leq 0.$$

In order to keep the volatility function positive for the case of two imaginary roots, we add the restriction

$$h_4(\sigma_0, s, q) = -\mathbb{I}_{(q^2-2s<0)}s \leq 0.$$

In all cases, we have the restriction that

$$h_5(\sigma_0, s, q) = -\sigma_0 \leq 0.$$

This leads us to the final formulation

$$\begin{aligned} \underset{\sigma_0,s,q}{\text{minimize}} \quad & |[Y,Y](t) - t|^2 \\ \text{subject to} \quad & \\ & h_1(\sigma_0, s, q) \leq 0, \\ & h_2(\sigma_0, s, q) \leq 0, \\ & h_3(\sigma_0, s, q) \leq 0, \\ & h_4(\sigma_0, s, q) \leq 0, \\ & h_5(\sigma_0, s, q) \leq 0. \end{aligned} \tag{4.1}$$

4.1 Particle swarm optimization

It turned out that classical gradient-descent based optimization methods were an ill fit for the optimization problem (4.1). The geometry of the objective function is alternately jagged and singular, alternately very flat. If one would start a gradient descent in the domain of complex roots solutions ($2s > q^2$) but the actual optimum would be in the domain of real roots solution ($2s < q^2$), there would be no way of passing over the one real root parabola $s = q^2/2$, as the objective function is singular here. To solve this problem we will utilize a stochastic optimization method known as particle swarm optimization.

In particle swarm optimization, one randomly generates a swarm of N points in the (σ_0, s, q)-space, and gives them all a randomly velocity in a random direction. They will evaluate the objective function value at their current location, inform one another of which particle that did the best, and then set off in a direction influenced by three inputs:

1. Their current velocity,
2. a weighted shift towards their personal best ever,
3. a weighted shift towards the best point ever achieved by any particle.

When 500 generations passes by and no new best ever solution has been found, we consider the current best an optimum. Alternatively, we terminate after 10,000 evaluations. The strength of using this method here is that each individual particle is unaware of the geometry of the problem, and is only aware of triplets (σ_0, s, q) that have performed well. All N particles will thus after a few generations swarm around well-performing points, effectively functioning as a search party. However we must implement the constraints $h_i(\sigma_0, s, q)$, $i \in (1, \ldots, 5)$ in some new manner. The answer to this is to utilize a penalty method. In a penalty method, we instead solve the unconstrained problem

$$\underset{\sigma_0,s,q}{\text{minimize}} \qquad |[Y,Y](t) - t|^2 + \mu_k \sum_{i=1}^{5} g_i(\sigma_0, s, q). \tag{4.2}$$

Here $g_i(\sigma_0, s, q) = \max\{0, h_i(\sigma_0, q, s)\}^2$, and μ_k is the penalty term. The goal is to first set $\mu_1 = 1$, fairly low, and then increase it by a factor of ten for a few generations. In the next generation, we still remember the last optimum and set it as the current best in the particle swarm implementation. As $\mu_k \to \infty$, we will find the optimum with regard to the restrictions.

4.2 Single root configuration

The chief difference from the two roots case is that $Y(t) = \Sigma(X(t))$ is given by (3.6) and we have fewer, different constraints. With $h_1(\sigma_0, q) = q - 2$ and $h_2(\sigma_0, q) = -\sigma_0$, particle swarm optimization will be used to solve the problem

$$\underset{\sigma_0,q}{\text{minimize}} \qquad |[Y,Y](t) - t|^2 + \mu_k \sum_{i=1}^{2} g_i(\sigma_0, s, q).$$

For a detailed description of particle swarm optimization, see [2].

5 Testing of the method on sample cases

To ensure ourselves that good results can and will be achieved by solving (4.1) for some dataset X, we will test the optimization algorithm on four sample cases. In all of these cases, we fix $x_0 = 55$, $\sigma_0 = 0.25$ and vary s and q so that different root configurations will be featured and tested. The test cases will be generated according to the *forward Euler-Mayurama* scheme of the stock model. Here, we will omit the drift part as it does not have any effect on the quadratic variation. With $\sigma(x)$ as in (3.1) we generate our test cases according to the following scheme

$$X_0 = x_0, \tag{5.1}$$
$$X_n = X_{n-1} + \sqrt{\Delta t}\sigma(X_{n-1})Z_{n-1}. \tag{5.2}$$

Here Δt is a time step corresponding to one trading day as a fraction of year, and Z_n are $\mathcal{N}(0,1)$-distributed random variables. We will generate the equivalence of two years of financial data, with ten time steps per day. We will then look at the imagined closing values of each day. The reason behind raising the resolution of the simulation is to combat numerical instability. In order for the Euler-Maruyama scheme to be unconditionally stable, the function $\sigma(x)$ must be Lipschitz continuous, which is not the case with quadratic volatility. The risk of exploding numerical values of the simulated stock X_n decreases drastically with small time steps, however.

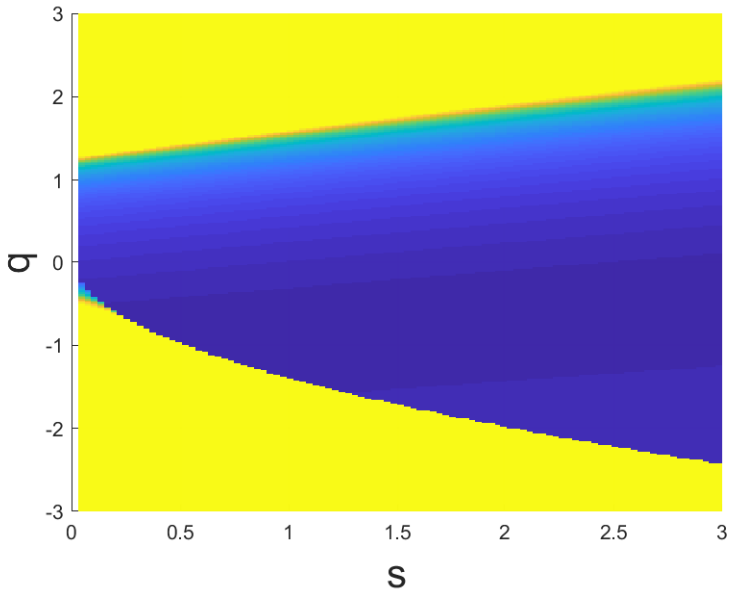
5.1 Geometrical properties of objective function depending on data

Before we conclude this chapter on implementation we shall study the archetypical geometry of $|[Y,Y](t) - t|^2$, and how it depends on the resolution of the data. Here we assume that the stochastic process $X(t)$ follows a quadratic volatility model with

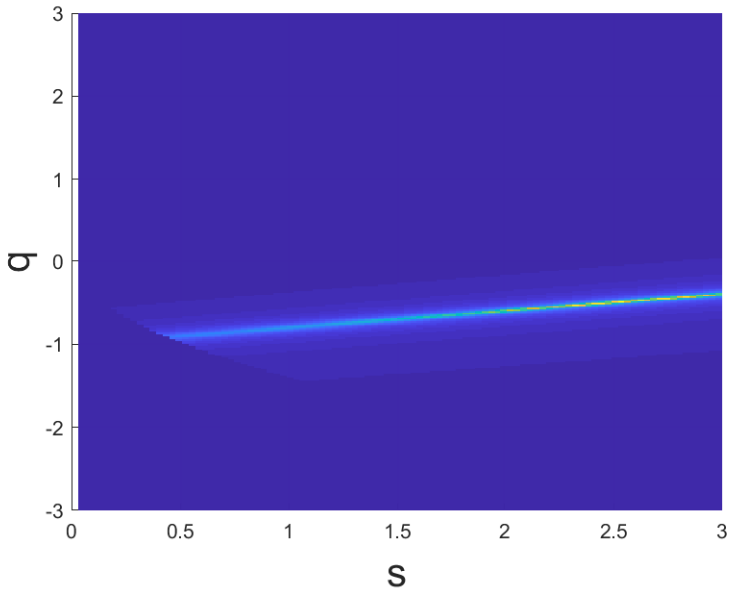
$$\sigma_0 = 0.25, \quad s = 2.5, \quad q = -0.5, \quad x_0 = 55. \tag{5.3}$$

This corresponds to a two complex roots scenario.

First we use the Euler-Mayurama scheme (5.2) to generate the equivalence of ten observations per day for 506 trading days, roughly two trading years. For the sake of simplicity we assume the observations to be equally spaced, and thus $\Delta t = 1/2530$.



(a)



(b)

Figure 5.1: $|[Y,Y](t)-t|^2+\mu_1\sum_{i=1}^5 g_i(\sigma_0,s,q)$ and $\left(|[Y,Y](t)-t|^2+\mu_1\sum_{i=1}^5 g_i(\sigma_0,s,q)\right)^{-1}$.

We see in Figure 5.1(a) clearly which areas that violate the constraints - the yellow areas outside the parabola $s = q^2/2$ is the penalty function being activated. To improve readability of this figure, we simple set $|[Y,Y](t) - t|^2 + \mu_1 \sum_{i=1}^5 g_i(\sigma_0, s, q) = 10$ whenever a constraint is violated. However, we also see that the objective function is seemingly very flat and featureless in the feasible set. To get a grip on where a global minima can be found,

we instead study $(|[Y,Y](t)-t|^2+\mu_k\sum_{i=1}^5 g_i(\sigma_0,s,q))^{-1}$, visualized in 5.1(b). Here, a thin strip of combinations of s and q that result in a small error becomes clearly visible.

When studying the same geometry as in 5.1(b) from a different angle, we note that ridge is jagged with many local minima. The proposed global minima $s=2.5$, $q=-0.5$ is close to the peak of the ridge. The value of the objective function at this point is around 0.00223, indicating that the simulation faithfully follows a possible trajectory of $X(t)$. When the optimization program was run for this particular $X(t)$ generated by the parameters (5.3), it settled for the parameters $\sigma_0=0.2495$, $s=2.5062$, $q=-0.5109$, with an ojbective value 0.00209. We thus see that we can faithfully reconstruct the underlying process when the data is of a decent resolution.

It is time to study what the method can make out of lower-resolution data. For this purpose we will study the same trajectory as earlier, but only utilize every tenth data point, representing the end of day values of our fictional stock. We can se the resulting geometry in Figure 5.3. The ridge is now almost flat and not at all jagged. The objective at $s=2.5$, $q=-0.5$ is 0.0223. When the optimization program was run on this data set, it settled for parameters $\sigma_0=0.3067$, $s=5.7544$, $q=0.9951$, resulting in an objective of 0.0193. This indicates that for low resolution data, it is difficult to reconstruct a constructed data set, as there are many candidate solutions of equal or better quality. We finish with studying the profile of $\left(|[Y,Y](t)-t|^2+\mu_1\sum_{i=1}^5 g_i(\sigma_0,s,q)\right)^{-1}$ under the assumption that $\sigma_0=0.3067$ in Figure 5.4. Now the ridge has a clear top. This indicates that if one can accurately determine σ_0, s and q are easy to determine as well. However, this set was to small to capture the intended σ_0.

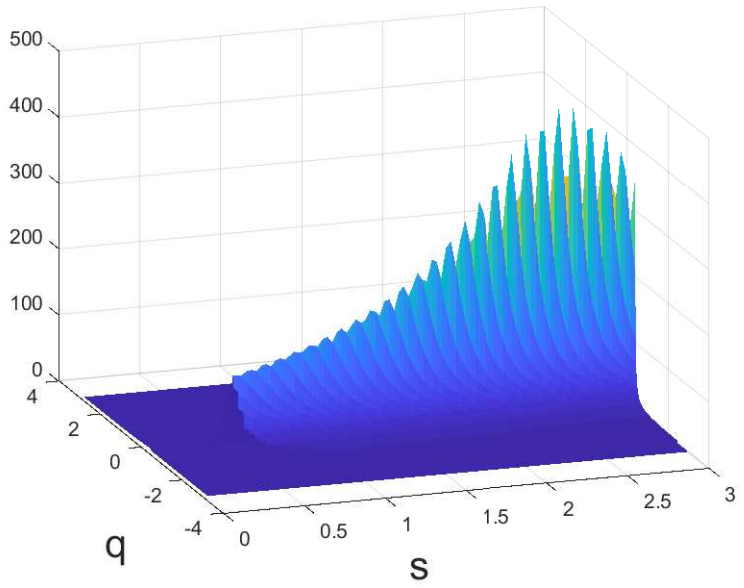


Figure 5.2: $\left(|[Y,Y](t)-t|^2+\mu_1\sum_{i=1}^5 g_i(\sigma_0,s,q)\right)^{-1}$, high resolution. Note the fluctuations.

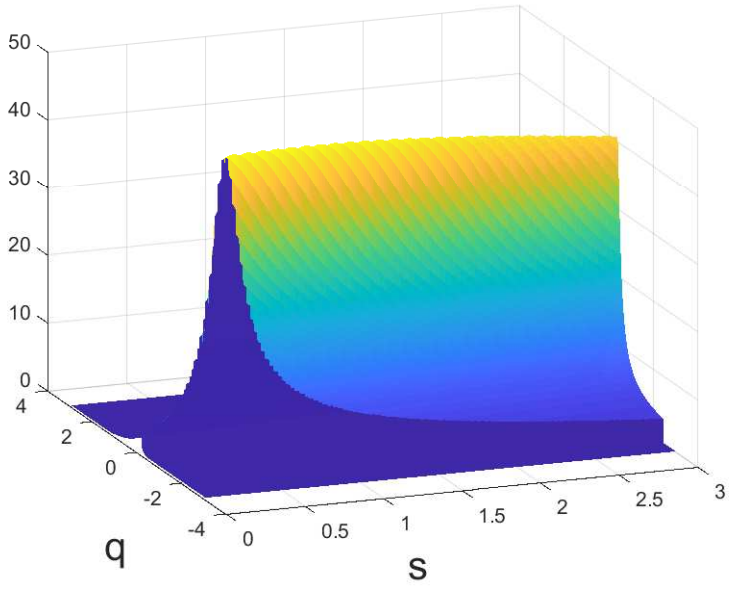


Figure 5.3: $\left(|[Y,Y](t) - t|^2 + \mu_1 \sum_{i=1}^{5} g_i(\sigma_0, s, q)\right)^{-1}$, low resolution. Markedly different geometry compared to Figure 5.2.

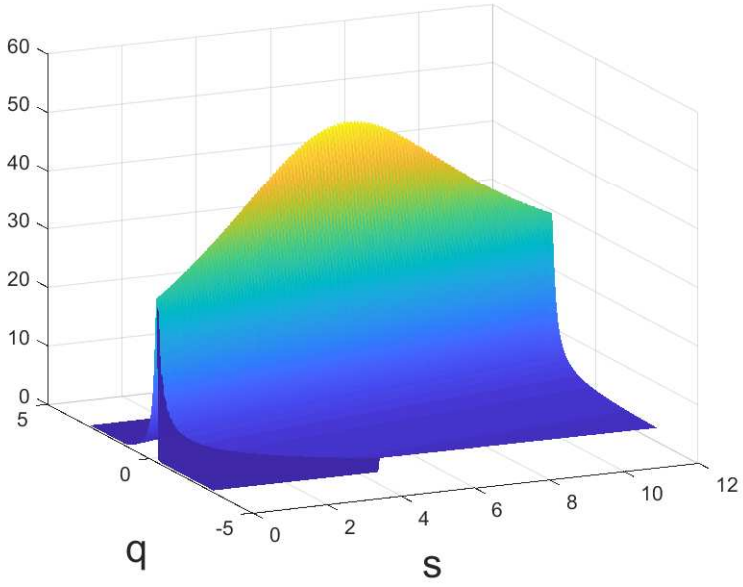


Figure 5.4: $\left(|[Y,Y](t) - t|^2 + \mu_1 \sum_{i=1}^{5} g_i(\sigma_0, s, q)\right)^{-1}$, low resolution. Under the false assumption that $\sigma_0 = 0.3067$. Clear top visible.

5.2 Test for two real roots, case 1

Here we set

$$\sigma_0 = 0.25, \quad s = 1, \quad q = \frac{3}{2}, \quad x_0 = 55. \tag{5.4}$$

giving us roots $l = -55$ and $u = 0$. We note that the true parameters are captured fairly well for the high resolution data. For low resolution data a good fit is achieved, but with completely wrong parameters. This indicates that tighter samples are necessary in order to unambiguously determine what the underlying process is.

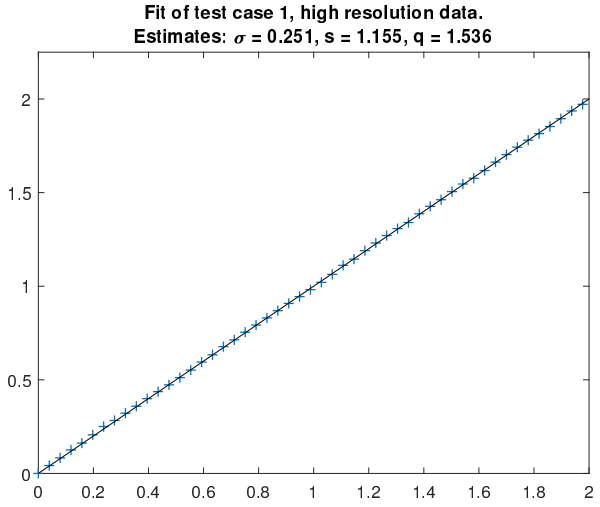


Figure 5.5

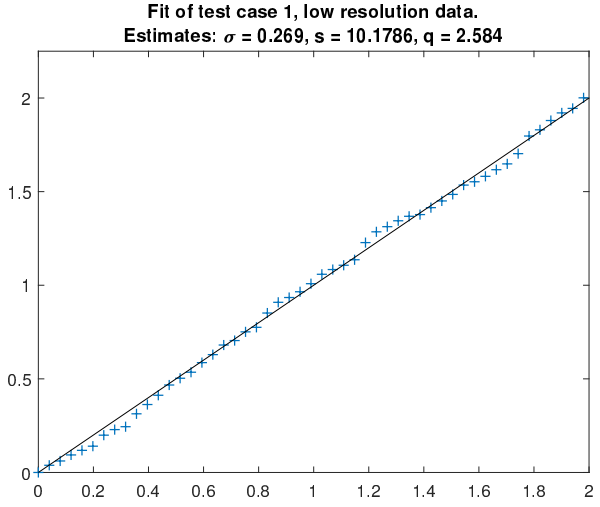


Figure 5.6

5.3 Test for two real roots, case 2

Here we set

$$\sigma_0 = 0.25, \quad s = -2, \quad q = 0, \quad x_0 = 55. \tag{5.5}$$

giving us roots $l = 0$ and $u = 110$. As for the first test case, the high resolution data estimates the parameters fairly. This time, we also managed to capture the parameters somewhat with the low resolution data.

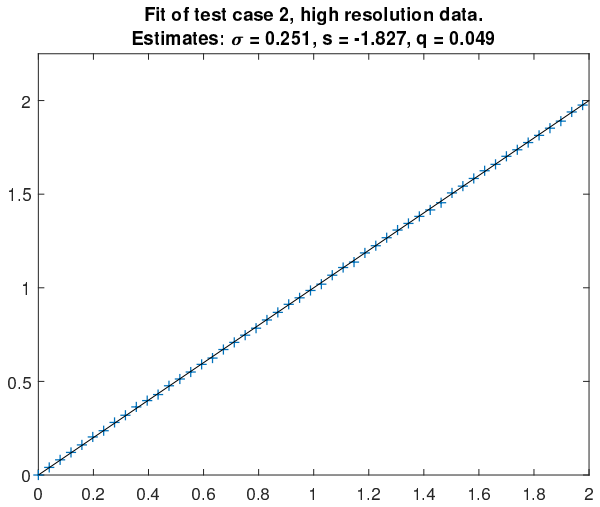


Figure 5.7

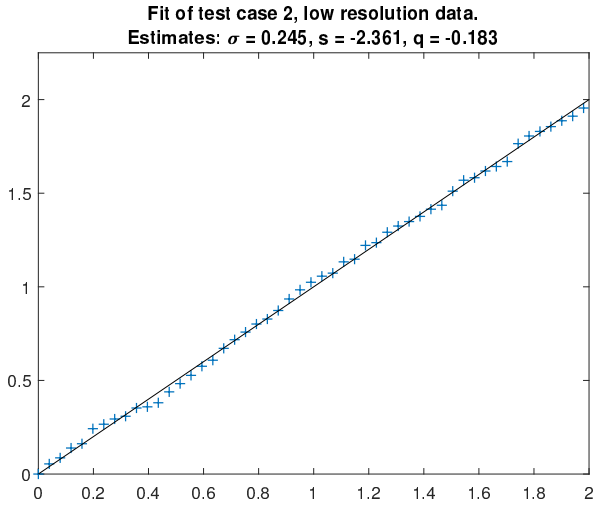


Figure 5.8

5.4 Test for two complex roots

Here we set

$$\sigma_0 = 0.25, \quad s = 2, \quad q = 0, \quad x_0 = 55. \tag{5.6}$$

giving us roots for $x = 55(1 \pm i)$. The results are similar to those of real roots, case 1. The higher resolution data does not capture the parameters perfectly, but it does a much better job than the lower resolution data.

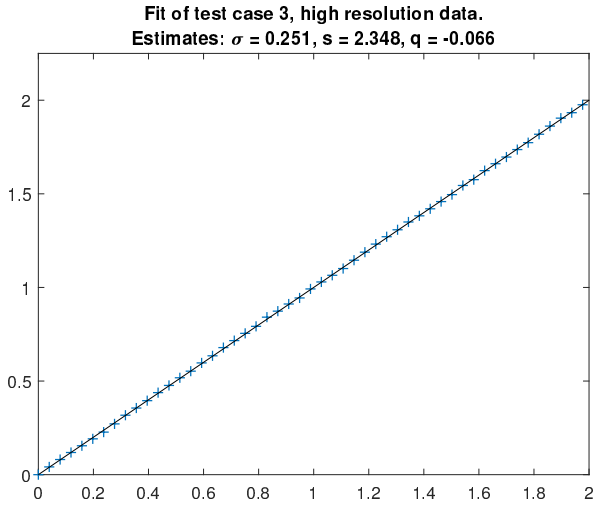


Figure 5.9

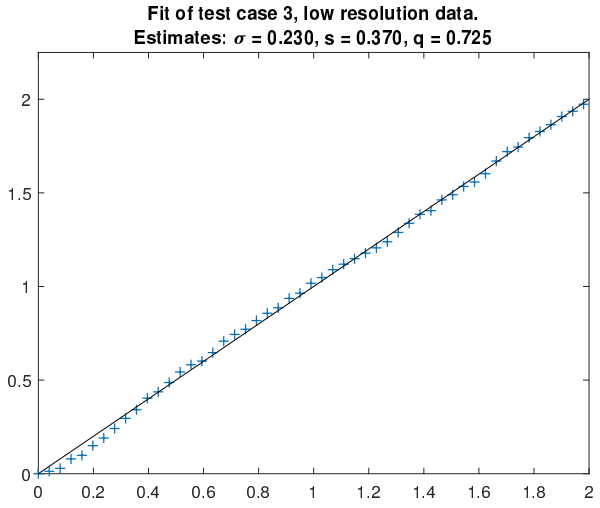


Figure 5.10

5.5 Test for a single real root

Here we set

$$\sigma_0 = 0.25, \quad s = 2, \quad q = 2, \quad x_0 = 55. \tag{5.7}$$

giving us a double root at $x = 0$. Here we manage to determine the underlying parameters almost perfectly for both the high and the low resolution data sets. In practice, this problem has one dimension less than the other two, meaning that there is less risk for finding a local minima.

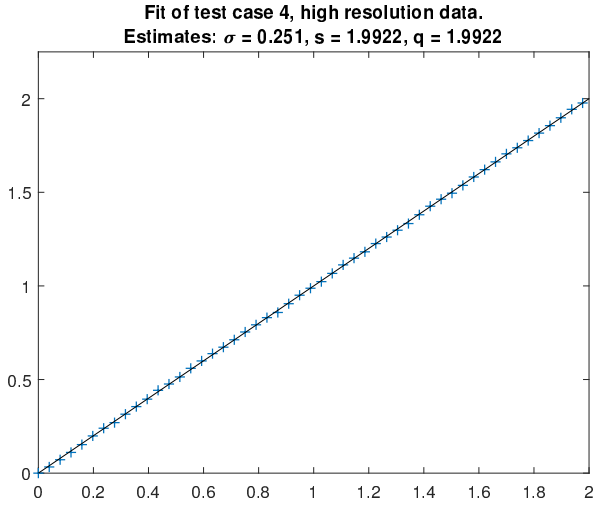


Figure 5.11

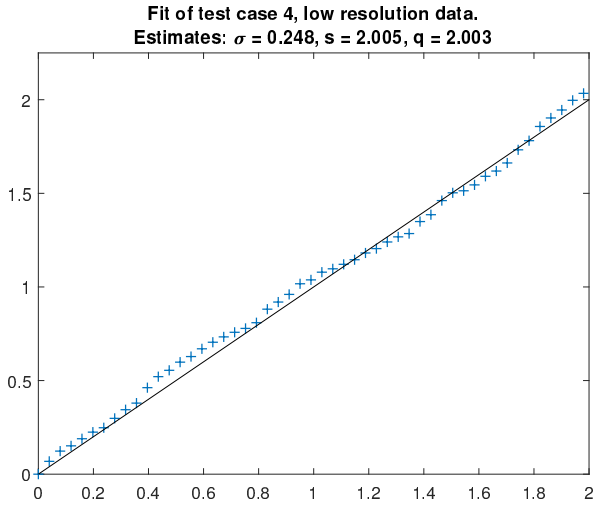


Figure 5.12

5.6 Test conclusion

For higher resolution data, the method can with some accuracy describe the underlying parameters of the random process For lower resolution data, there appear to be many contesting solutions that are good enough, and provide a decent fit. As a final test case, the parameter set $\sigma_0 = 0.20$, $s = 2$, $q = 1$ was used to generate a 2^{23} data points sample (roughly 100 samples every second for two years), starting from $x_0 = 20$. We then look at increasingly large samples from this set, and estimate the parameters. The results are visualized in Figure 5.13.

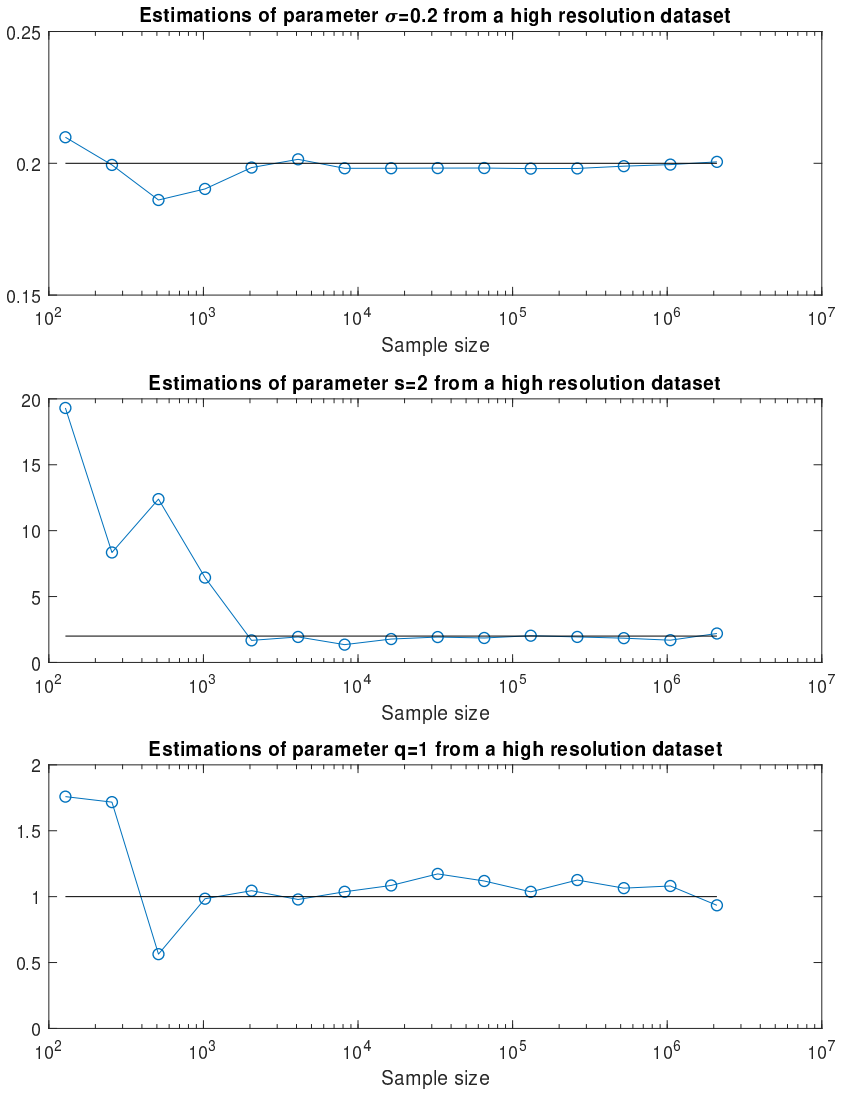


Figure 5.13

To complement Figure 5.13, we also include plots of the actual proposed polynomials as given by the parameters. Here the target polynomial is in black, and the polynomial as proposed by the model is marked by blue stars. We see that after the sample consisting out of 2^{10} samples we get somewhat accurate results around x_0, and the polynomial is reconstructed to very little error after 2^{11} samples. Smaller samples tend to overestimate the steepness of the polynomial.

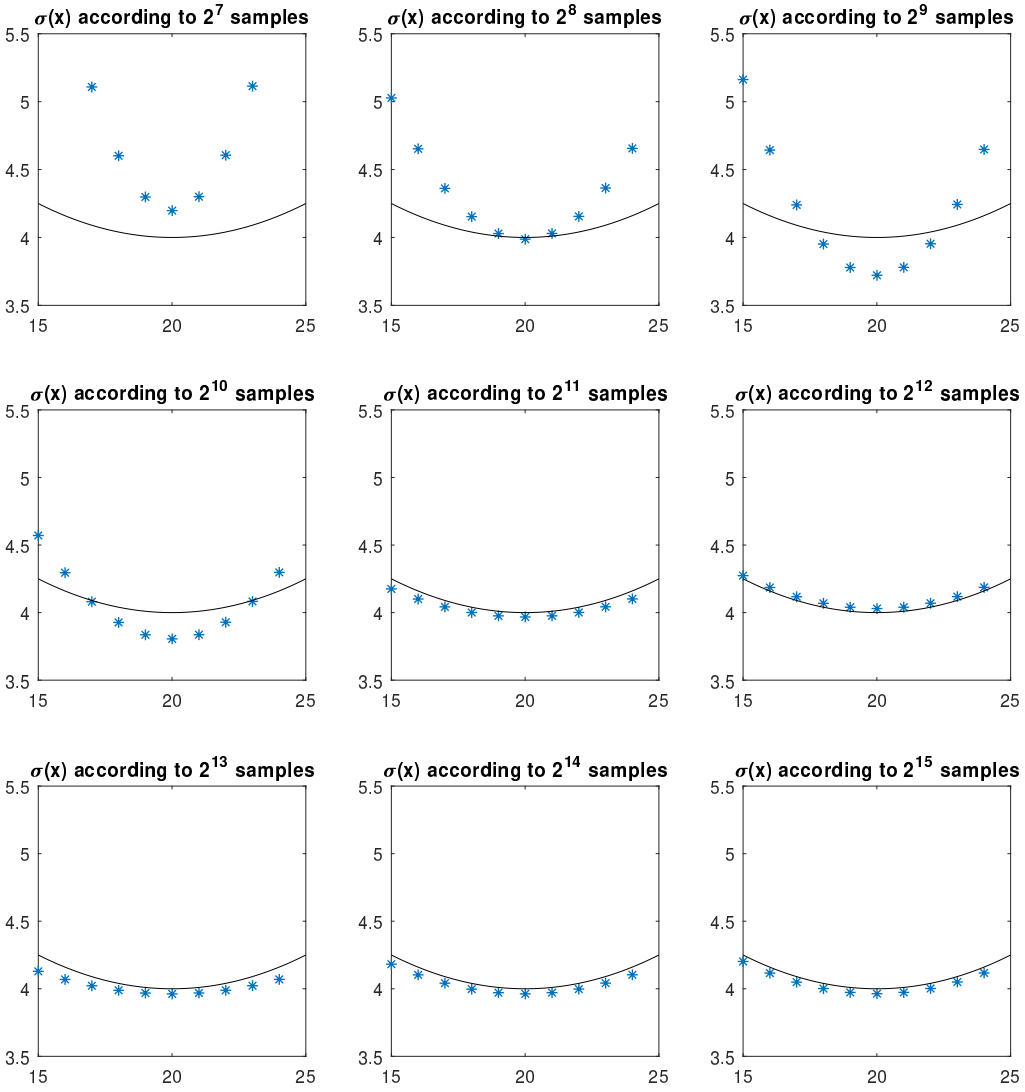


Figure 5.14: Gradual increases in sample resolution with the resulting estimated polynomials marked by blue stars. The true polynomial is given by the black curve.

6 Option pricing using QV-models

The value of a European call option at expiry is $(X(T)-K)_+$, and the value of a put option is $(K-X(T))_+$, where K is the strike price and $X(T)$ is the underlying stock value at the expiry time T. Remembering (2.2), for European call options we are tasked with solving the initial-value problem

$$-\partial_\tau c(x,\tau) + rx\partial_x c(x,\tau) + \frac{\sigma^2(x)}{2}\partial_x^2 c(x,\tau) = 0, \qquad x>0, \quad \tau\in[0,T], \tag{6.1}$$

$$c(0,t) = 0, \tag{6.2}$$

$$c(x,0) = (x-K)_+. \tag{6.3}$$

For put options the corresponding PDE is

$$-\partial_\tau p(x,\tau) + rx\partial_x p(x,\tau) + \frac{\sigma^2(x)}{2}\partial_x^2 p(x,\tau) = 0, \qquad x>0, \quad \tau\in[0,T],$$

$$p(0,\tau) = Ke^{r\tau},$$

$$p(x,0) = (K-x)_+.$$

One of the main strengths of the Black-Scholes model $\sigma(x)=\sigma_{BS}x$ is that this PDE has a closed-form solution as the log-normal cumulative probability distribution. There is no close form solution in the case of quadratic volatility, and we will have to resort to numerical methods. However, in order to use the more popular numerical methods for PDE:s, we're going to have to truncate to some compact subinterval of $\mathbb{R}$ in order to solve this equation for x. We choose the interval $I=[0,\max\{6x_0,u\}]$, where x_0 is the initial value of the stock we aim to price options for. We thus deal with parabolic initial-boundary condition problems, with an added boundary condition

$$c(\max\{6x_0,u\},\tau) = \max\{6x_0,u\} - Ke^{r\tau},$$
$$p(\max\{6x_0,u\},\tau) = 0.$$

This is assuming that we do not choose strike prices as large as $6x_0$. These imposed boundary conditions make sure that the solution stays continuous along the boundary of the space-time domain $[0,\max\{6x_0,u\}]\times[0,T]$, and serves as an estimate of $\lim_{x\to\infty} c(x,\tau)$, $\lim_{x\to\infty} p(x,\tau)$. Remembering the *put-call parity*

$$c(x,\tau) - p(x,\tau) = x - Ke^{-r\tau}$$

we realize that solving the PDE for only one case is sufficient for determining both prices. We thus focus on solving the PDE corresponding to call options. In order to do so, we use the Matlab function `pdepe.m`, which uses a continuous second-order Galerkin finite element approximation for the space variable and uses the stiff ODE-solver `ode15.m` for time-discretization.

7 Application real data

In this section we present the results of solving the penalty method problem (4.2) when the time series $\mathbf{x}_t$ is an actual stock history. We will consider four different stocks, collected from the Nasdaq stock exchange. We will observe the last two years of end-of-day values, corresponding to 506 data points. From Section 4.3 we have indications that such a low resolution on the time series risk giving us inaccurate estimates, especially of s and q. Figure 5.13 indicate that a sample larger than 1000 is required to reach acceptable accuracy.

The stocks that are up for consideration are Apple, Microsoft, Netflix and Tesla. The two first companies generally enjoys low stock volatility, while the two last have underwent rapid growth in the last few years, with some downfalls as well. These thus represent higher-volatility cases, in which we expect quadratic volatility to perform significantly better than pure Black Scholes. The stock profiles alongside the fit of a quadratic variation model can be seen in figure 7.1-7.4. For reference, the fit according to a Black-Scholes model is included for reference, i.e the case $\sigma(x) = \sigma_{BS}x$.

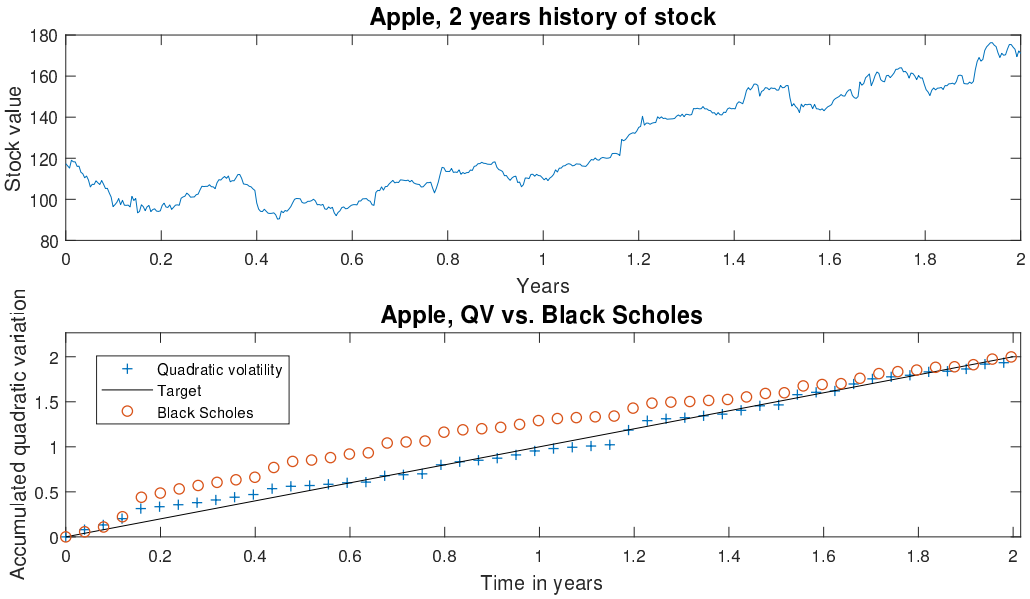


Figure 7.1: Apple stock data from 2015-12-01 to 2017-12-01 [7]. Quadratic volatility fit: $\sigma_0 = 0.1771$, $s = 16.0776$, $q = -1.1303$. Black Scholes volatility: $\sigma_{BS} = 0.2173$.

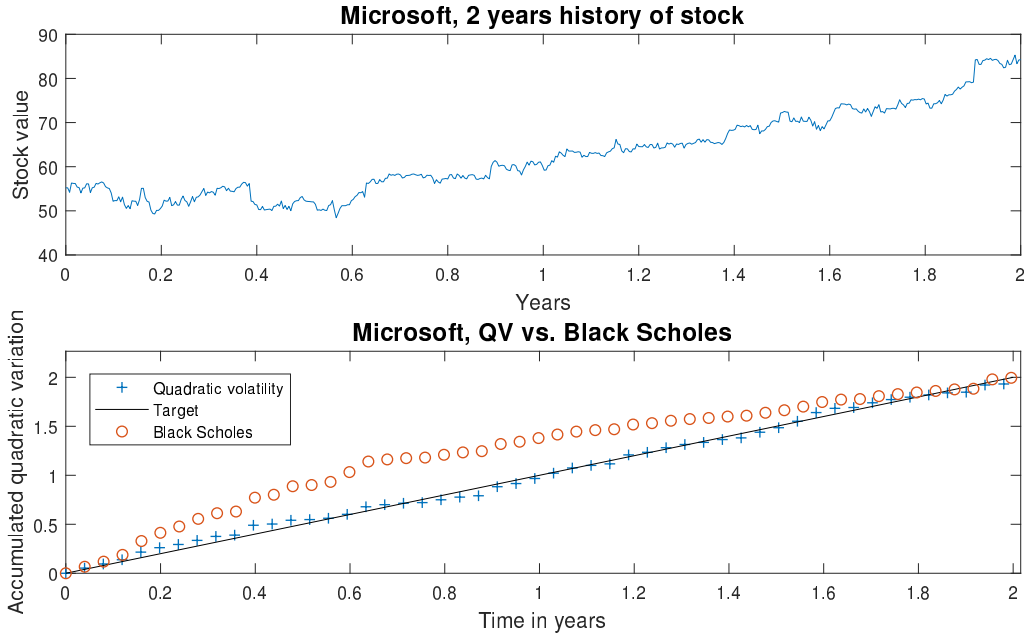


Figure 7.2: Microsoft stock data from 2015-12-01 to 2017-12-01 [8]. Quadratic volatility fit: $\sigma_0 = 0.2321$, $s = 15.9339$, $q = -2.7437$. Black Scholes volatility: $\sigma_{BS} = 0.2089$.

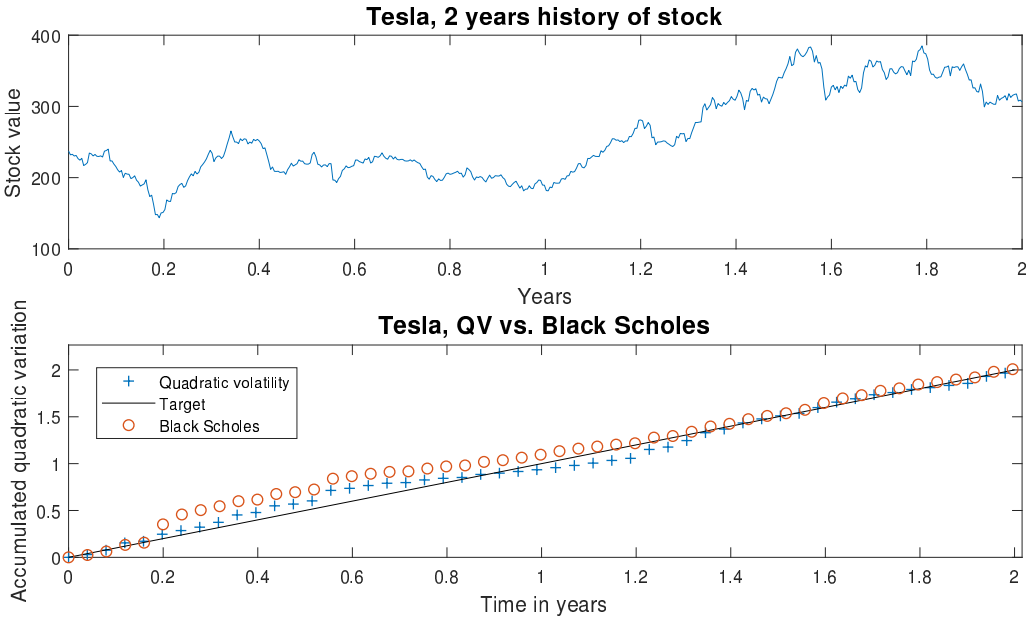


Figure 7.3: Tesla stock data from 2015-12-01 to 2017-12-01 [9]. Quadratic volatility fit: $\sigma_0 = 0.3179$, $s = 8.1029$, $q = -0.1312$. Black Scholes volatility: $\sigma_{BS} = 0.3715$.

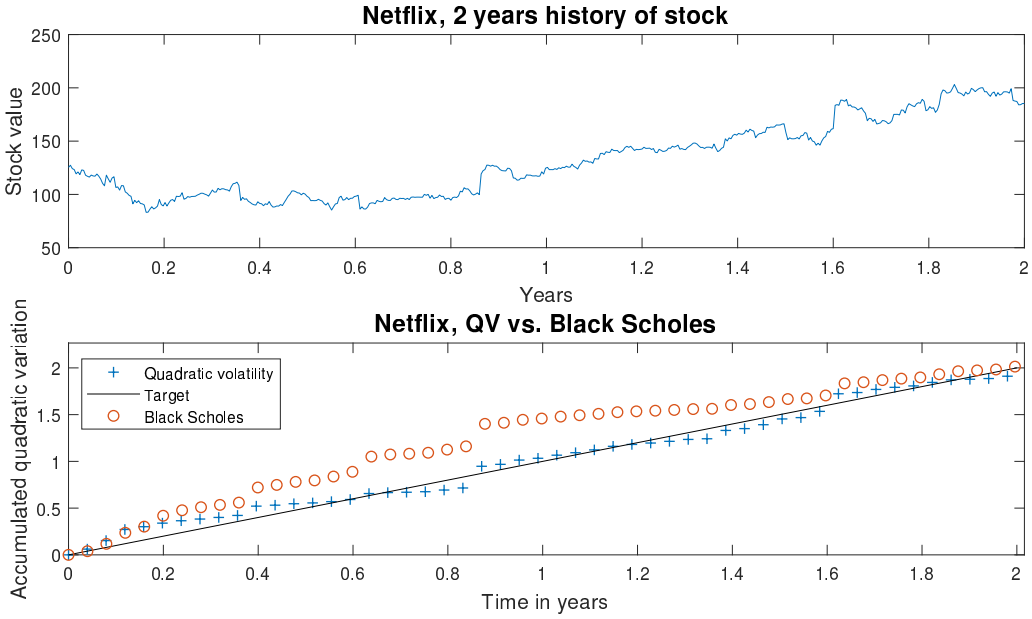


Figure 7.4: Netflix stock data from 2015-12-01 to 2017-12-01 [10]. Quadratic volatility fit: $\sigma_0 = 0.2829$, $s = 10.9509$, $q = -0.6675$. Black Scholes volatility: $\sigma_{BS} = 0.3753$.

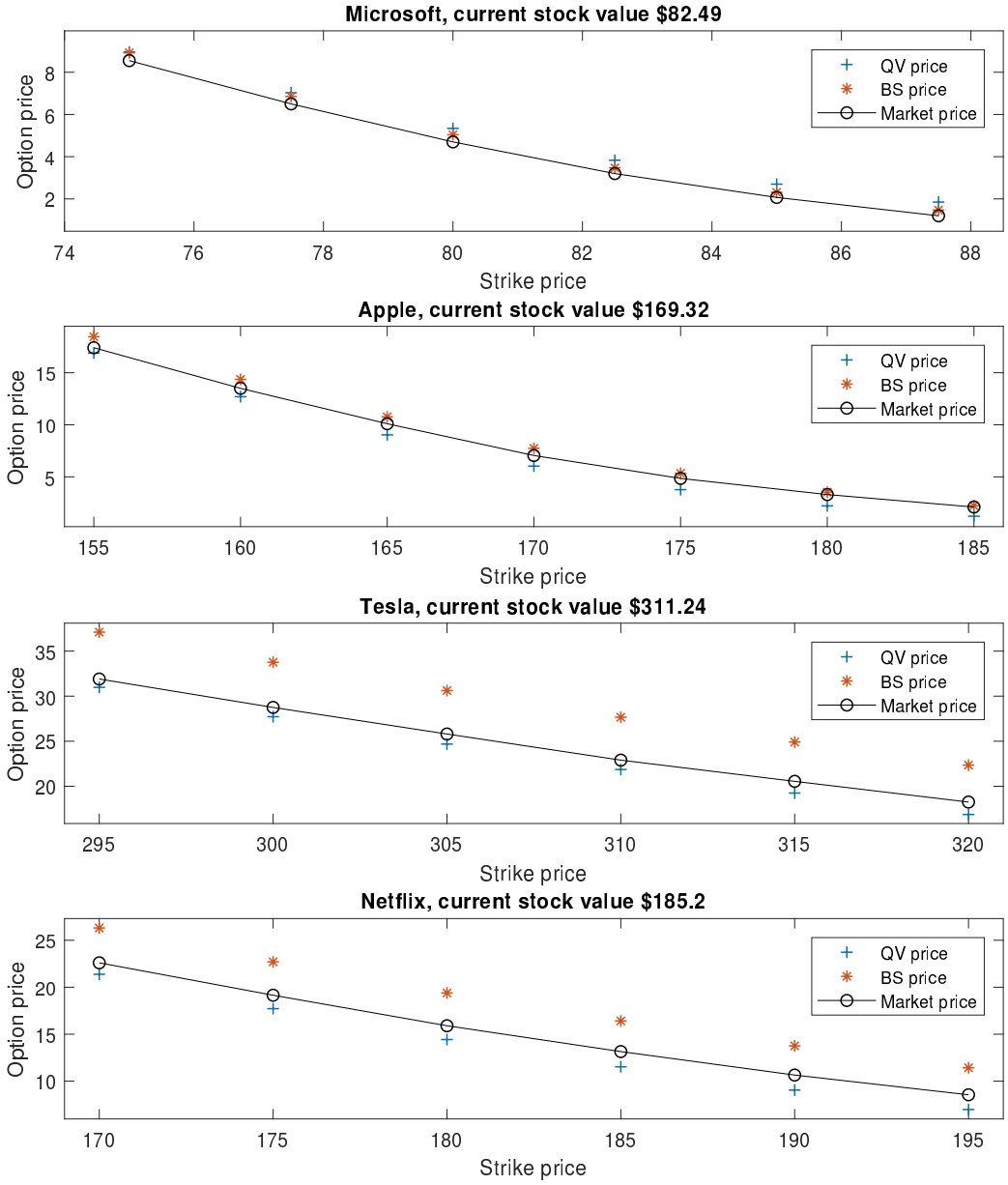


Figure 7.5: Prices on call options derived from solving (6.1) with parameters calibrated from the stock data. Black line is actual prices for options expiring 2018-02-16, taken from [7]-[10] on 2017-12-08.

8 Conclusion

8.1 Performance of parameter estimation program

It turned out the transformation-optimization method of parameter calibration produce very good results for high resolution data. We also note from the test cases that the even when the estimated polynomial might seem to be off target, it remains acceptably close to the target function $\sigma(x)$ for x in the neighbourhood of x_0. For smaller samples the main issue seem to be overestimation of the steepness of the polynomial. We see this in Figure 5.14, and it is indicated by that s is estimated to be quite large in all real-life samples. In conclusion, one would need tighter samples than end-of-day values in order to achieve the best possible results when applying this method to real data. However, for strike prices close to x_0, we get fair estimates.

8.2 Quality of real life applications

It is greatly evident from Figure 7.1-7.4 that even with a low resolution sample, quadratic volatility describes real life data to greater accuracy than just modelling the stock as a Geometric Brownian motion. Since we have concluded that low-resolution estimates of $\sigma(x)$ behaves well locally, we also get decent and very realistic prices on European call options when using quadratic volatility. For the more volatile stocks, Tesla and Netflix, quadratic volatility prices were much closer than the prices given by Black Scholes. Thus looking into our local volatility model might be a good idea if historic stock data indicate that $\sigma_{BS} > 0.3$. For the less volatile stocks, quadratic volatility and Black Scholes were an even match and they both estimated market values well.

8.3 Possible expansions of the model and future work

In real-life applications of stochastic calculus for pricing options, more advanced models than the still very much simplified one we use here are considered. Common methods include using *stochastic interest rate models* for pricing of long-term contracts (we assume constant interest rates) and *stochastic* volatility models for more fine-tuned volatility calibration. Our current quadratic volatility model fall under the category of *local* volatility models. Putting

$$L(x) = (1-q)x_0 + qx + \frac{s}{2x_0}(x-x_0)^2$$

we propose an extended model

$$\begin{aligned} dX(t) &= r(t)X(t)dt &+ \sqrt{\nu(t)}L(X(t))dB_1(t) \\ dr(t) &= \alpha_R(r(t),t)dt &+ \beta_R(r(t),t)dB_2(t) \\ d\nu(t) &= \alpha_\nu(\nu(t),t)dt &+ \beta_\nu(\nu(t),t)dB_3(t) \end{aligned}$$

where $\{B_i\}_{i=1,2,3}$ are three Brownian motions that may or may not be correlated. This is a quadratic volatility version of a standard stochastic interest rate-volatility model. $\alpha_{R,\nu}$, $\beta_{R,\nu}$ are adapted processes describing the dynamics of $r(t)$ and $\nu(t) = \sigma_0^2(t)$. We see in Figure 7.5 that in applications to option pricing, the added parameters of quadratic volatility does indeed provide additional accuracy, at least in the sense of using market prices as a bench

mark. Thus it is reasonable to assume that using a quadratic local volatility function in conjunction with sophisticated stochastic models of interest rates and volatility would provide even further accuracy. A problem that arises here is the excessive amount of parameters; let us briefly consider the popular Vasciek model of interest rates and the GARCH model of volatility. This would result in the system of equations

$$\begin{aligned} dX(t) &= r(t)X(t)dt + \sqrt{\nu(t)}L(X(t))dB_1(t) \\ dr(t) &= a(b - r(t))dt + cdB_2(t) \\ d\nu(t) &= \alpha(\beta - \nu(t))dt + \gamma\nu(t)dB_3(t) \end{aligned}$$

This gives us an addition of the parameters a, b, c describing the interest rate dynamics, and α, β, γ describing the volatility. Using a similar approach as in this report, one would have a far more complicated quadratic volatility structure, and the calibration have up to eight degrees freedom. This is without considering the possible correlation structure of the driving Brownian motions. My hypothesis is that very high-resolution data would be necessary to accurately calibrate these parameters. A different approach to a similar model can be found in [4], where parameter averaging methods are used to derive analytical pricing methods. This article inferes that there is indeed much power in local-stochastic volatility models, but that calibration routines is a delicate problem for future research.

Bibliography

[1] Klebaner F.C., *Introduction to Stochastic Calculus with Applications*, Imperial College Press, 3rd edition, 2012.

[2] Wahde M., *Biologically Inspired Optimization Methods, an Introduction*, WITPress, 2008.

[3] Andersen L., *Option Pricing with Quadratic Volatility: a Revisit*, Bank of America Securities, 2008.

[4] Andersen L., Hutchings N., *Parameter averaging of Quadratic SDEs with Stochastic Volatility*, Banc of America Securities, 2008.

[5] Albanese C., Campolieti G., Carr P., Lipton A., *Black-Scholes goes Hypergeometric*, RISK.NET 2001.

[6] Chibane, M., Law, D., *A Quadratic Volatility Cheyette Model*, Shinsei Bank Limited, 2012.

[7] Apple stock and option chain data. Retrieved from `http://www.nasdaq.com/aapl` on 2017-12-08.

[8] Microsoft stock and option chain data. Retrieved from `http://www.nasdaq.com/msft` on 2017-12-08.

[9] Tesla stock and option chain data. Retrieved from `http://www.nasdaq.com/tsla` on 2017-12-08.

[10] Netflix stock and option chain data. Retrieved from `http://www.nasdaq.com/nflc` on 2017-12-08.

A Matlab code for parameter calibration

A.1 Main.m

```
clear all
close all
clc

% stock takes on the values {Apple, Netflix, Microsoft, Tesla}
% Data gathered from Nasdaq.com on 2017-12-08
stock = 'Tesla';
X = flip(Data(stock));
subplot(2,1,1)
plot(linspace(0,2,length(X)),X)
stock = strcat('\fontsize{14}',stock);
xlabel('\fontsize{12}Years')
ylabel('\fontsize{12}Stock value')
title(strcat(stock,'\fontsize{14}, 2 years history of stock'))

%% Optimize according to two different roots
tic
[sigma,q,s,L2error] = Optimization(X);
discriminant = (q^2-2*s);
polynomial = FunctionString(sigma,q,s,X(1))
subplot(2,1,2)
PlotQuadraticVariation(sigma,q,s,X)
toc

%% Optimize according to a double root
[sigma,q,L2error] = OptimizationSingleRoot(X);
s = abs(q)^2/2;
polynomial = FunctionString(sigma,q,s,X(1))
PlotQuadraticVariationSingleRoot(sigma,q,X)

%% Fit a standard Black Scholes for reference
subplot(2,1,2)
[sigmaBS,L2error] = OptimizationBlackScholes(X);
PlotQuadraticVariationBS(sigmaBS,X)
xlabel('\fontsize{12}Time in years')
ylabel('\fontsize{12}Accumulated quadratic variation')
legend('Quadratic volatility','Target','Black Scholes')
title(strcat(stock,'\fontsize{14}, QV vs. Black Scholes'))
```

A.2 Optimization.m

```
function [sigma,q,s,fitnessValue] = Optimization(X)
n                = length(X);
populationSize   = 100;
alpha            = 0.1;
beta             = 0.9999;
dt               = 0.1;
c1               = 2;
c2               = 2;
bestEver         = 1000000000000000;
fitness          = zeros(populationSize,1);
bestPerformance  = 1000000000000000*ones(populationSize,1);
bestPlaceEver    = [100*rand 100*rand 100*rand];
maxSpeed         = 0.5;
inertiaMin       = 0.3;

for k = 1:5
    bestPlace              = zeros(populationSize,3);
    [swarm,swarmVelocity] = generateSwarm(populationSize,alpha,dt,X
       (1));
    sinceNice = 0;
    iteration = 0;
    inertia   = 15;
    while sinceNice < 500 && iteration < 10000

        for iFitness=1:populationSize
            fitness(iFitness) = EvaluateIndividual(swarm(iFitness,:),
               X,n,k);

        end

        for iBest=1:populationSize
            if fitness(iBest) < bestPerformance(iBest)
                bestPerformance(iBest) = fitness(iBest);
                bestPlace(iBest,:)     = swarm(iBest,:);
            end
        end

        [bestThisGeneration,bestTemp] = min(bestPerformance);


        if bestThisGeneration < bestEver
            temple         = bestEver;
            TempTemp       = bestThisGeneration;
            bestEver       = bestThisGeneration;
            bestPlaceEver  = swarm(bestTemp,:);
            sinceNice      = 0;
            %if abs(temple-TempTemp)<10^(-4)
            %    [swarm,swarmVelocity] = generateSwarm(populationSize
               ,alpha,dt,X(1));
```

```
47              %end
48
49          end
50
51          for iUp=1:populationSize
52              swarmVelocity(iUp,:) = inertia*swarmVelocity(iUp,:)...
53                  + c1/dt*rand*(bestPlace(iUp,:)-swarm(iUp,:))...
54                  + c2/dt*rand*(bestPlaceEver-swarm(iUp,:));
55              speed = sqrt(swarmVelocity(iUp,1)^2+swarmVelocity(iUp,2)
                    ^2);
56
57              if speed > maxSpeed
58                  swarmVelocity(iUp,:) = maxSpeed*swarmVelocity(iUp,:)/
                        speed;
59              end
60
61              swarm(iUp,:) = swarm(iUp,:) + swarmVelocity(iUp,:)*dt;
62
63
64          end
65
66          inertia   = inertia*beta^(iteration)+inertiaMin;
67          iteration = iteration + 1;
68          sinceNice = sinceNice + 1;
69
70      end
71  end
72  localOptimum = bestPlaceEver;
73  fitnessValue = bestEver;
74  sigma        = localOptimum(1);
75  q            = localOptimum(2);
76  s            = localOptimum(3);
77
78  end
```

A.3 GenerateSwarm.m

```
function [swarm,swarmVelocities] = generateSwarm(populationSize,alpha
    ,deltaT,x0)

    swarm           = zeros(populationSize,3);
    swarmVelocities = zeros(populationSize,3);

    for i=1:populationSize
        swarm(i,1)           = rand*1;
        swarm(i,2)           = (rand*x0/32-x0/16)*0.5;
        swarm(i,3)           = (rand*x0/32-x0/16)*0.5;

        swarmVelocities(i,1) = alpha/deltaT*(-1/2+rand);
        swarmVelocities(i,2) = alpha/deltaT*(-1/2+rand);
        swarmVelocities(i,3) = alpha/deltaT*(-1/2+rand);
    end

end
```

A.4 EvaluateIndividual.m

```
function [f] = EvaluateIndividual(swarm,X,n,k)
    X0    = X(1);
    sigma = swarm(1);
    q     = swarm(2);
    s     = swarm(3);
    xdata = zeros(n-1,2);
    ydata = cumsum(ones(n-1,1))/253;

        for i=1:n-1
            xdata(i,1)=X(i+1);
            xdata(i,2)=X(i);
        end

        quadraticVariation = zeros(1,n-1);
        for iTime=1:n-1
            part1 = sigma*sqrt(2*s-q^2);
            part2 = q*X0+s*(xdata(iTime,1)-X0);
            part3 = X0*sqrt(2*s-q^2);
            part4 = q*X0+s*(xdata(iTime,2)-X0);
            quadraticVariation(iTime+1) = quadraticVariation(iTime)
                ...
                                        + real(2/(part1)*((atan(part2
                                          /part3)))-...
                                          2/(part1)*((atan(part4/
                                             part3))))^2;
        end

        e = 0;
        for i=1:n-1
            errorI = (quadraticVariation(i)-ydata(i)).^2;
            e      = e + errorI;
        end

        constraints = Constraints(sigma,s,q);
        f   = sqrt(e/n)*253/n;
        f = f + (10^k)*constraints;
end
```

A.5 Constraints.m

```
function constraints = Constraints(sigma,s,q)
    case1 = 0;
    case2 = 0;
    case3 = 0;

    if (q^2 > 2*s) && (s > 0)
        case1 = 1;
    end

    if (q^2 > 2*s) && (s < 0)
        case2 = 1;
    end

    if (q^2 < 2*s)
        case3 = 1;
    end

    h1 = case1*((s-q)+sqrt(q^2-2*s));
    h2 = case2*((s-q)-sqrt(q^2-2*s));
    h3 = case2*((q-s)-sqrt(q^2-2*s));
    h4 = -case3*s;
    h5 = -sigma;

    g1 = max(h1,0)^2;
    g2 = max(h2,0)^2;
    g3 = max(h3,0)^2;
    g4 = max(h4,0)^2;
    g5 = max(h5,0)^2;


    constraints = g1+g2+g3+g4+g5;

end
```

A.6 PlotQuadraticVariation.m

```
function PlotQuadraticVariation(sigma,q,s,X)
    X0 = X(1);
    n  = length(X);
    Y = real(2/(sigma*sqrt(2*s-q^2))*atan((q*X0+s*(X-X0))/(X0*sqrt(2*
       s-q^2))));


    QY = zeros(length(X),1);
    for i=2:n
        QY(i)  = QY(i-1) + (Y(i) - Y(i-1))^2;
    end
    time = linspace(0,1,length(QY))*n/253;
    plot(time(1:10:end),QY(1:10:end), '+')
    hold on
    plot(time,time,'black')
    axis([0 time(end) 0 time(end)+0.25])
end
```

A.7 FunctionString.m

```
function polynomial = FunctionString(sigma,q,s,x0)
    a = num2str(sigma*x0*((1-q)+s/2));
    b = num2str(sigma*(q-s));
    c = num2str(sigma*s/(2*x0));
    polynomial = strcat(a,"+(",b,"x)+(",c,")x^2");
end
```

A.8 OptimizationSingleRoot.m

```
function [sigma,q,fitnessValue] = OptimizationSingleRoot(X)
n               = length(X);
populationSize  = 100;
alpha           = 0.1;
beta            = 0.9999;
dt              = 0.1;
c1              = 2;
c2              = 2;
bestEver        = 1000;
fitness         = zeros(populationSize,1);
bestPerformance = 1000*ones(populationSize,1);
bestPlaceEver   = [1 1];
maxSpeed        = 1;
inertiaMin      = 0.3;

for k = 1:5
    bestPlace             = zeros(populationSize,2);
    [swarm,swarmVelocity] = generateSwarmSingleRoot(populationSize,
       alpha,dt);
    sinceNice = 0;
    iteration = 0;
    inertia   = 15;
    while sinceNice < 500 && iteration < 100000

        for iFitness=1:populationSize

            fitness(iFitness) = EvaluateIndividualSingleRoot(swarm(
               iFitness,:),X,n,k);
        end

        for iBest=1:populationSize
            if fitness(iBest) < bestPerformance(iBest)
                bestPerformance(iBest) = fitness(iBest);
                bestPlace(iBest,:)     = swarm(iBest,:);
            end
        end

        [bestThisGeneration,bestTemp] = min(bestPerformance);

        if bestThisGeneration < bestEver
            bestEver      = bestThisGeneration;
            bestPlaceEver = swarm(bestTemp,:);
            sinceNice     = 0;
        end

        for iUp=1:populationSize
            swarmVelocity(iUp,:) = inertia*swarmVelocity(iUp,:)...
                + c1/dt*rand*(bestPlace(iUp,:)-swarm(iUp,:))...
                + c2/dt*rand*(bestPlaceEver-swarm(iUp,:));
```

```
48              speed = sqrt(swarmVelocity(iUp,1)^2+swarmVelocity(iUp,2)
                    ^2);
49
50              if speed > maxSpeed
51                  swarmVelocity(iUp,:) = maxSpeed*swarmVelocity(iUp,:)/
                        speed;
52              end
53
54              swarm(iUp,:) = swarm(iUp,:) + swarmVelocity(iUp,:)*dt;
55
56
57          end
58
59          inertia   = inertia*beta^(iteration)+inertiaMin;
60          iteration = iteration + 1;
61          sinceNice = sinceNice + 1;
62      end
63  end
64  localOptimum = bestPlaceEver;
65  fitnessValue = min(fitness);
66  sigma        = localOptimum(1);
67  q            = localOptimum(2);
68  fitnessValue
69  end
```

A.9 GenerateSwarmSingleRoot.m

```
function [swarm,swarmVelocities] = generateSwarm(populationSize,alpha
    ,deltaT)

    swarm           = zeros(populationSize,2);
    swarmVelocities = zeros(populationSize,2);

    for i=1:populationSize
        swarm(i,1)           = rand*1;
        swarm(i,2)           = rand*20-10;

        swarmVelocities(i,1) = alpha/deltaT*(-1+2*rand);
        swarmVelocities(i,2) = alpha/deltaT*(-1/2+rand);
    end

end
```

A.10 EvaluateIndividualSingleRoot.m

```
function f = EvaluateIndividualSingleRoot(swarm,X,n,k)
    X0    = X(1);
    sigma = swarm(1);
    q     = swarm(2);
    xdata = zeros(n-1,2);
    ydata = cumsum(ones(n-1,1))/253;

        for i=1:n-1
            xdata(i,1)=X(i+1);
            xdata(i,2)=X(i);
        end

        quadraticVariation = zeros(1,n-1);
        for iTime=1:n-1
            part1 = -4*X0/(sigma*q^2);
            part2 = 1/(xdata(iTime,1)-X0*((q-2)/q));
            part3 = 1/(xdata(iTime,2)-X0*((q-2)/q));
            quadraticVariation(iTime+1) = quadraticVariation(iTime)
                ...
                                        + (part1*(part2-part3))^2;
        end

        e = 0;
        for i=1:n-1
            errorI = (quadraticVariation(i)-ydata(i)).^2;
            e      = e + errorI;
        end

        constraints = ConstraintsSingeRoot(sigma,q);
        f   = sqrt(e/n)*253/n;
        f = f + (10^k)*constraints;
end
```

A.11 ConstraintsSingleRoot.m

```
function constraints = ConstraintsSingeRoot(sigma,q)
    h1 = -sigma;
    h2 = q-2;

    g1 = max(h1,0)^2;
    g2 = max(h2,0)^2;

    constraints = g1+g2;




end
```

A.12 PlotQuadraticVariationSingleRoot.m

```
function PlotQuadraticVariationSingleRoot(sigma,q,X)
    X0 = X(1);
    n  = length(X);
    Y = -4*X0/(sigma*q^2)./(X-X0*((q-2)/(q)));


    QY = zeros(length(X),1);
    for i=2:n
        QY(i)  = QY(i-1) + (Y(i) - Y(i-1))^2;
    end
    time = linspace(0,1,length(QY))*n/253;
    plot(time(1:10:end),QY(1:10:end), '+')
    hold on
    plot(time,time,'black')
    axis([0 time(end) 0 time(end)+0.25])
end
```

A.13 OptimizationBlackScholes.m

```
function [sigma,L2error] = OptimizationBlackScholes(X)
    n = length(X);
    xdata = zeros(n-1,2);
    for i=1:n-1
            xdata(i,1)=X(i+1);
            xdata(i,2)=X(i);
    end

    quadraticVariation = zeros(1,n-1);
            for iTime=1:n-1
                quadraticVariation(iTime+1) = quadraticVariation(
                   iTime)...
                                    +  (log(xdata(iTime,1)/xdata(iTime
                                       ,2)))^2;
            end

    fun = @(x)(1/x(1).^2*quadraticVariation(end)-n/251).^2;
    lb = [0];
    ub = [1000];
    A = [];
    b = [];
    Aeq = [];
    beq = [];
    x0 = [0.5];
    [x,fval] = fmincon(fun,x0,A,b,Aeq,beq,lb,ub);

    sigma   = x;
    L2error = sqrt(fval);


end
```

A.14 PlotQuadraticVariationBS.m

```
function PlotQuadraticVariationBS(sigma,X)
    hold on
    n  = length(X);
    Y = log(X)/sigma;

    QY = zeros(length(X),1);
    for i=2:n
        QY(i)  = QY(i-1) + (Y(i) - Y(i-1))^2;
    end
    time = linspace(0,1,length(QY))*n/251;
     plot(time(1:10:end),QY(1:10:end), 'o')
    axis([0 time(end) 0 time(end)+0.25])
end
```

B Code for pricing call option using PDE

Note that the parameters σ, s, q, x_0, r and σ_{BS} must be manually inserted in the code below, due to limitations in the Matlab function `pdepe.m`.

```
function PriceCallOption

close all
x0 = 65;
tradingDays = 48;
strikePrice = 55;
sPString    = num2str(strikePrice);
dt          = 1/251;
h           = 1/16;

m = 0;
x = linspace(0,6*x0,6*x0/h);
t = linspace(0,tradingDays*dt,tradingDays);

sol   = pdepe(m,@pdex1pde,@pdex1ic,@pdex1bc,x,t);
solBS = pdepe(m,@pdex2pde,@pdex1ic,@pdex1bc,x,t);
% Extract the first solution component as u.
u   = sol(:,:,1);
uBS = solBS(:,:,1);



for time=1:tradingDays
    clf
    plot(x,u(time,:))
    hold on
    plot(x,uBS(time,:))
    axis([0.8*strikePrice 1.2*strikePrice 0 45])
    title(strcat('Strike price is $',sPString))
    xlabel('Stock Value')
    ylabel('Call option price')
    drawnow
    pause(0.01)
    sol = sol(end,:);
end
QVPrice = num2str(sol(end,round(x0/h)));
BSPrice = num2str(uBS(end,round(x0/h)));
PricingInfo = strcat('Strike: $',sPString,'. QV price is $',QVPrice,'
    . BS price is $',BSPrice)
% --------------------------------------------------------------
function [c,f,s] = pdex1pde(x,t,u,DuDx)
sigma = 0.25;
s     = 2.0;
q     = 00;
r = 0.05;
x0 = 65;
volatility = sigma*((1-q)*x0+q*x0+(s/(2*x0))*(x-x0).^2);
```

```
47 interest   = r*x;
48
49 c = 1;
50 f = volatility.^2/2*DuDx+interest*u;
51 s = 0;
52 % -----------------------------------------------------------------
53 function [c,f,s] = pdex2pde(x,t,u,DuDx)
54 sigma = 0.23;
55 s     = 0.37;
56 q     = 0.725;
57 r = 0.05;
58 x0 = 65;
59 volatility = sigma*((1-q)*x0+q*x0+(s/(2*x0))*(x-x0).^2);
60 interest   = r*x;
61
62 c = 1;
63 f = volatility.^2/2*DuDx+interest*u;
64 s = 0;
65 % -----------------------------------------------------------------
66 function u0 = pdex1ic(x)
67 strikePrice = 55;
68 u0          = max((x-strikePrice),0);
69 % -----------------------------------------------------------------
70 function [pl,ql,pr,qr] = pdex1bc(xl,ul,xr,ur,t)
71 x0 = 65;
72 r           = 0.05;
73 strikePrice = 55;
74 dt          = 1/251;
75
76 pl = ul;
77 ql = 0;
78 pr = ur-max((6*x0-strikePrice),0)-strikePrice*exp(r*(t));
79 qr = 0;
```