University of West Bohemia
Faculty of Applied Sciences
Department of Mathematics

Bachelor's thesis

Calibration of stochastic volatility models using quasi-evolutionary algorithms

Plzeň 2017 Tomáš Osvald

Místo této strany bude zadání práce.

Declaration

I hereby declare that this bachelor's thesis is completely my own work and that I used only the cited sources.

Plzeň, 9 July 2017

Tomáš Osvald

Abstract

In this thesis, we focus on calibration of stochastic volatility models using quasi-evolutionary algorithms. First we introduce evolutionary algorithms and types of initialization of a new population, that has an important impact on the algorithm. In methodology, we describe each step of the genetic algorithm, set the test functions and focus on stochastic volatility models. An implementation part of this thesis is also a modification of genetic algorithm in software Matlab. We compare quasi random and random initial population on real market data calibration problem.

Abstrakt

V této práci se zabýváme kalibrací modelů stochastické volatility pomocí kvazi-evolučních algoritmů. V úvodu nastíníme problém evolučních algoritmů a typy inicializačních populací, které mají velký vliv na výsledek algoritmu. V práci popisujeme jednotlivé kroky genetického algoritmu, stanovýme si testovací funkce a zabýváme se modely stochastické volatility. Součástí práce je i modifikace genetického algoritmu v programu Matlab. Zde porovnáváme použití náhodné a quasi náhodné inicializační populace při kalibraci na reálná tržní data.

Contents

1 Introduction

Evolutionary algorithms are optimizations algorithms, inspired by biological processes that allow populations of organisms to adapt to their environment. This theory was introduced by Charles Darwin (Ellegård (1958)) in the 19th century and it is still valid, but it was completed with further details.

In the mid 1960th were presented many of new Darwin's theory extensions. For example Holland (1992) introduced genetic algorithms, Fogel et al. (1966) and colleagues began experiments on evolutionary programing and Rechenberg (1973) started to work on evolutionary strategies. Their work brought bases of optimization methods suited for hard problems, where little is known about the underlying search space. For introduction to evolutionary computing Eiben – Smith (2003), or see also Fogel (1999) - 40 Years of Evolutionary Programing.

In this thesis, we will compare impact of random and quasi random numbers on searching global minimum of a d dimensional real valued function $f(x)$. We will use Evolutionary algorithms (EA), especially Genetic algorithm (GA) to search global extremes. The research is focused on using quasi random numbers in generating an initial population in order to improve a starting position in algorithm.

Algorithm

EA algorithm begins with initial population of randomly generated first individuals. After that, fitness value is evaluated for each member of population. Next step is regeneration, which will be done by selection of the best individuals and will set a new one for new generation by evolution rules (mutation and crossover). Then we evaluate fitness value for each new member form new population and replace last population with the new one. This procedure is repeated until the stopping conditions end the algorithm.

The aim of the thesis is to study Quasi-EA (QEA), i.e. EA that uses QRS. We will see that QRS are especiall crucial for initial population.

First we will make an introduction to initialization of the first population.

Population initialization techniques

Following Kazimipour et al. (2014) we can divide initialization of a new population in Evolutionary algorithm (EA) into 3 categories according to the techniques (Figure 1.1):

- Randomness,
- Compositionality,
- Generality.

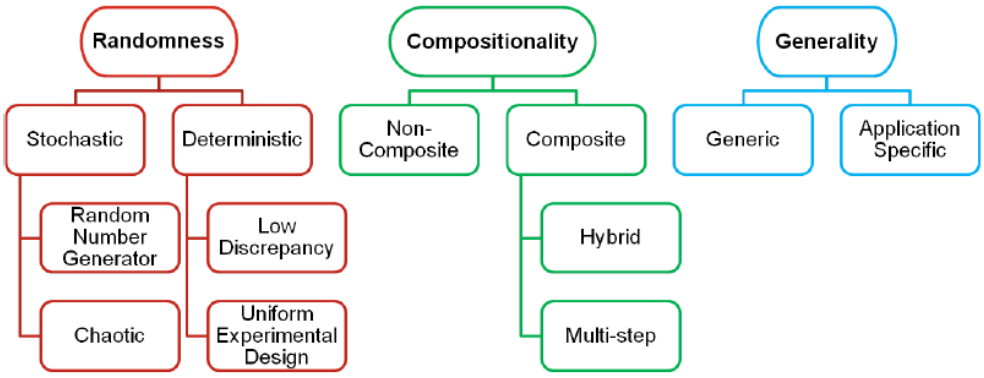


Figure 1.1: Initialization techniques.

Randomness

Random initial population is a truly random sequence of numbers. Truly random is defined as a sequence with unpredictability, irregularity and incompressibility.

We can also discriminate randomness initial techniques by attributes of resulted populations on:

- Stochastic,
- Deterministic.

Results, in stochastic techniques, depend on initial seeds. There is for example pseudo-random number generator (PRNG), or chaotic number generator (CNG). Deterministic techniques always generate the same populations. They are also called low-discrepancy techniques. It means that points with low-discrepancy are points with high level of uniformity. To this category belong quasi-random sequences and uniform experimental design.

Compositionality

Compositionality is defined as individual procedures involved in initialization techniques. We can divide initialization techniques into two groups:

- non-compositional,
- compositional

Non-compositional are techniques, that produce population in only one step. So we can mark all random (Stochastic, Deterministic) techniques as non-compositional.

On the other hand compositional techniques generate the population by more than one step. We divide this group in two subgroups:

- hybrid
- multi-step techniques

We can apply each component of hybrid techniques as a non-compositional separately. For example we can use CNG to generate the initial seed for PRNG. The hybrid techniques can be also used to give quasi random sequences (QRS) more randomness. It will cause that the new population has a uniformity of QRS and randomness of PRNG.

The second technique is multi-step, which consist of two or more initializations. Within the next steps they improve the population, which was generated.

The most used multi-step is opposition based learning. For the first population is generated a set of points by using any initializer technique, for example PRNG or CNG. We call it Original population. The Opposite population is generated in the second step by using simple heuristic rules. This new Opposite population has the same size as the Original population. As the last step, we make subset of both populations, which is based on fitness

value.

Generality

This selection is about variability of population initializer to be applied on many types of techniques.

- Generic
- Application Specific

The Generic techniques produce populations, which can be applied to all types of optimization problems. On the other hand, the application specific techniques are specially designed to specific real world problems.

Structure of the thesis

Structure of the thesis is as follows. In Chapter 2 we introduce quasi random initial population and we compare random and quasi random initial population. We discuss Genetic algorithm in steps, set a few test functions and we introduce stochastic volatility models.

In Chapter 3 we present results of GA on test functions and we compare quasi random and random creation function on real market data.

We conclude in Chapter 4.

2 Methodology

Let

$$f : \mathbb{R}^d \to \mathbb{R}$$

be a real function of d variables $(x_1, ..., x_d)$. The aim of the thesis is to study a simple minimization problem

$$\min_{x \in \mathbb{R}^d} f(x)$$

with respect to simple bounds (constraints)

$$l_k \leq x_k \leq u_k; \quad k = 1, ..., d,$$

where l_k are given lower bounds and u_k are given upper bounds.

2.1 Evolutionary algorithms

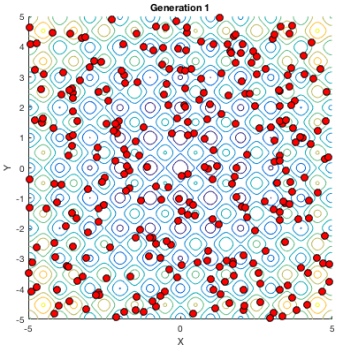
Evolutionary algorithm (EA) belongs to the evolutionary computing. It is a group of algorithms for global optimization, that use biological evolution processes. In contrast to local gradient based optimization algorithms, there are no restrictions or requiremets for function f. EAs can be successfully used if f is not smooth or even if f is discontinous.

2.1.1 Terminology

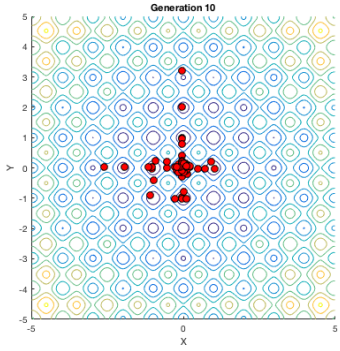
We have to define a few terms, that we will use in the description of EA optimization techniques.

Fitness function is a function that will be optimized. The main purpose of EA is to find a global minimum of fitness function.

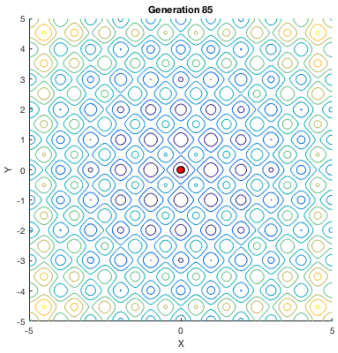
An **individual** is a vector $[x_1, ..., x_d] \in \mathbb{R}^d$ where fitness function can be evaluated, we usually call this value a **score**. When we talk about the best fitness value, it means the lowest fitness value from one population. **Population** is a finite set of individuals. For example, if we have a function with 2 variables and a size of population is 50, the whole population can be represented by a matrix 50×2.



(a) Generation 1



(b) Generation 10



(c) Last generation

Figure 2.1: Iteration process

The first (**initial**) population is of special importance. There exist several initiation techniques that we introduce above. In the interation process of every EA, a new popilation is generated from the previous population by evolution rules. One step in this iteration process is called a **generation**. In each consecutive generation, a new population should have all individuals closer to minimum of the function. We can see the generation process in Figure 2.1.

We can see, that **diversity** (average distance among scores of individuals in population)

$$\sigma_N = \sqrt{\frac{\sum_{i=1}^{N}(f_i^k - \overline{f}^k)^2}{N-1}},$$

where

$$\overline{f}^k = \frac{1}{N}\sum_{i=1}^{N} f_i^k$$

is higher in the first generation, than diversity in the next generation. It

means that the diversity is getting lower with each generation.

For evolutionary proceses, we have to choose **parents** among individuals in each population, which will create individuals for the new population. These new individuals are called **children**.

Quasi random initial population

The main intention of low discrepancy sequence is to create more uniformly distributed individuals than the standard random number sequences. We apply the low discrepancy sequence generator instead of PRNG.

One way how to measure uniformity is to define the discrepancy number for N points $X_1, X_2, \ldots, X_N$ in $[0,1]^d$ as

$$D_N = \sup_{R \in \mathcal{R}} \mid \frac{1}{N} \sum \overrightarrow{1}_{\{X_{n \in R}\}} - \mathrm{vol}(R) \mid,$$

where R is a rectangles such $[a_1; a_2] \times \cdots \times [a_d; b_d]$ in $[0,1]^d$ and $0 \leq a_n \leq b_n \leq 1$ and $\mathcal{R}$ is a set of all rectangles. Volume of R: $\mathrm{vol}(R) = \prod_{n=1}^{d}(b_n - a_n)$.

This formula calculates the discrepancy of the set of points (X_i), where $i = 1, \ldots, N$. Discrepancy measures the uniformity between the points (X_i). We find the rectangle R, where the points are the most different from our uniformity expectation.

Since D_N can be algorithmically difficult to compute, we define also the star discrepancy

$$D_N^* = \sup_{R \in \mathcal{A}} \mid \frac{1}{N} \sum \overrightarrow{1}_{\{X_{n \in R}\}} - \mathrm{vol}(R) \mid .$$

The only difference from the definition of D_N is such that the supremum is taken over the set $\mathcal{A}$ that contains all rectangles with one corner at the origin, i.e. where all $a_n \equiv 0, n = 1, \ldots, d$.

Lemma 2.1. *For all N and d:*

$$D_N^* \leq D_N \leq 2^d D_N^*.$$

Proof. The left-hand bound is obvious since $\mathcal{A} \subset \mathcal{R}$ The right-hand bound follows form the fact that every $R \in \mathcal{R}$ can be composed via unions, intersections and complements of no more than 2^d anchored rectangles, i.e.

rectangles in $\mathcal{A}$.

□

Sequences with low D_N^* are called **low discrepancy sequences**. One good example is a Halton sequence. It is an extension of one dimensional **Van der Corput sequence**, that is formed by a binary expansion of the integer i and it reflects the digits around the decimal point. Let $\phi_b(n)$ be a radical inverse function in base b :

$$\phi_b(n) = \sum_{k=0}^{+\infty} a_k b^{-k-1},$$

where $n = \sum_{k=0}^{+\infty} a_k b^k$ and a_k are the digits in the base b expansion of n.

The **Halton sequence** extend Van der Corput sequence to multiple dimensions. Let p_m be the m-th smallest prime, then the $n-$th point X_n of the d-dimensional Halton sequence is

$$X_n = (\phi_{p_1}(n), \phi_{p_2}(n), ..., \phi_{p_d}(n)).$$

Halton sequence is extensible. That means, it is not necessary to choose the length of sequence at the beginning.

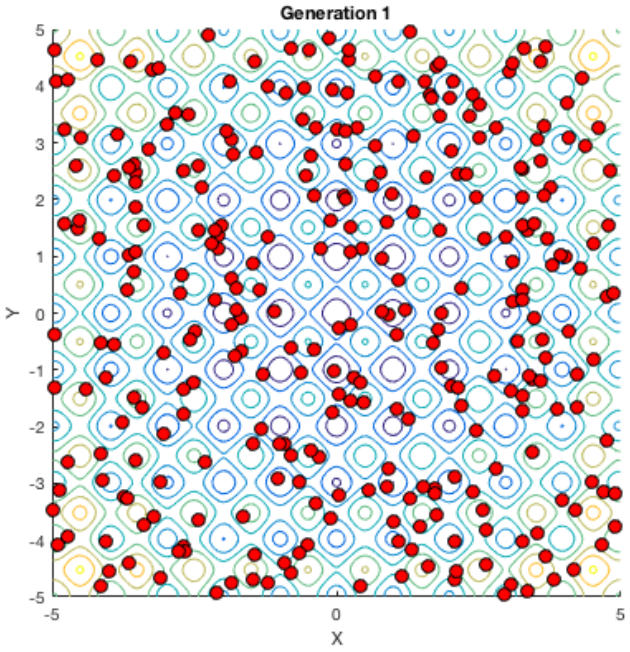
2.2 Genetic algorithm

Genetic algorithm (GA) is a process inspired by nature. It is based on biological process. There are mainly used operators of mutation, selection and crossover.

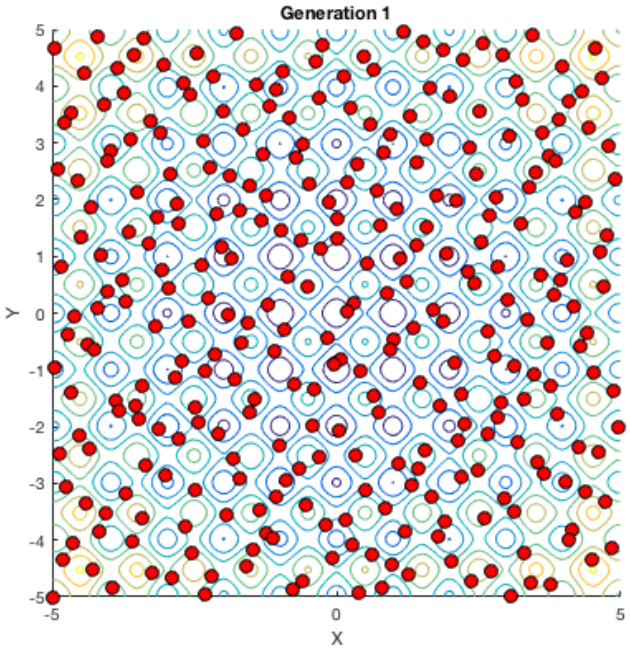
GA is a subgroup of EA. The main preference of GA is due to generations of high quality solutions for optimizations.

Optimization

The main intention of an optimization task is to get the best solution from the initial population by genetic operations. Each individuals have a set of properties which are used to differentiate how they will be altered. Usually, there we consider individuals $X_i \in \mathbb{R}^d$ with real valued scores $f(X_i)$.



(a) Random generation initial population



(b) Quasi random initial population

Figure 2.2: Comparation of quasi random and random initial population

Initial population of GA is usually random generation of individuals,

which is modified to better solution in each iteration (generation). Fitness value of all functions is evaluated in each generation and works like a main indicator. Individuals are modified by recombinations or they randomly mutate to get a new better generation. The new generation is used as the next iteration. This process is applied until the time, when the algorithm will be terminated by stopping criteria, which can be for example exceeding maximum number of iterations, or getting acceptable fitness values.

2.2.1 Algorithm

Step 1:
Initialize the population $P_0 \in \mathbb{R}^{d\times N}$ of N individuals $x_i \in \mathbb{R}^d$,$i = 1, ..., N$. Define the elite proportion $e \in [0, 1]$, crossover proportion $c \in [0, 1]$: $e + c < 1$. Algorithm then creates the sequence of new populations $P_k, k = 1, 2, \ldots, k_{\max}$.

Step 2:
Let $\lceil x \rceil$ denote the upper integer part of x (ceil) $\lceil x \rceil = \min\{z \in \mathbb{R}; z \geq x\}$. $N_e = \lceil eN \rceil$ be the number of elite individuals, $N_c = \lceil c(N - N_e) \rceil$ be the number of crossover individuals and $N_m = N - N_e - N_c$ be the number of mutated individuals. Number of parents will be denoted by $N_p = 2N_c + N_m$.

Step 3:
Elite kids: form the ordered population $P_k : x_1^k, \ldots, x_N^k$, where $f(x_1^{(k)}) \leq f(x_2^{(k)}) \leq ... \leq x_N^{(k)}$, select the elite kids $a^{(k)} = [x_1^{(k)}, \ldots, x_{N_e}^{(k)}]$.

Step 4:
Crossover kids: Let $X_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,d}]$ and $Y_i = [y_{i,1}, y_{i,2}, \ldots, y_{i,d}]$ be two parents; $X_i, Y_i \in P_k$. We get, by random combination of these parents, a new child $b^{(k)} = [x_1^{(k)}, \ldots, x_{N_c}^{(k)}]$.

Step 5:
Mutate kids: Substitute the rest of population P_k by randomly generated kids $c^{(k)} = [x_1^{(k)}, \ldots, x_{N_m}^{(k)}]$, where $l_k \leq x_j^{(k)} \leq u_k$.

Step 6:
New population $P_{k+1} = [a^{(k)}; b^{(k)}, c^{(k)}]$.

Step 7:

Termination: if a tereminal condition is met, then repeat form step 3.

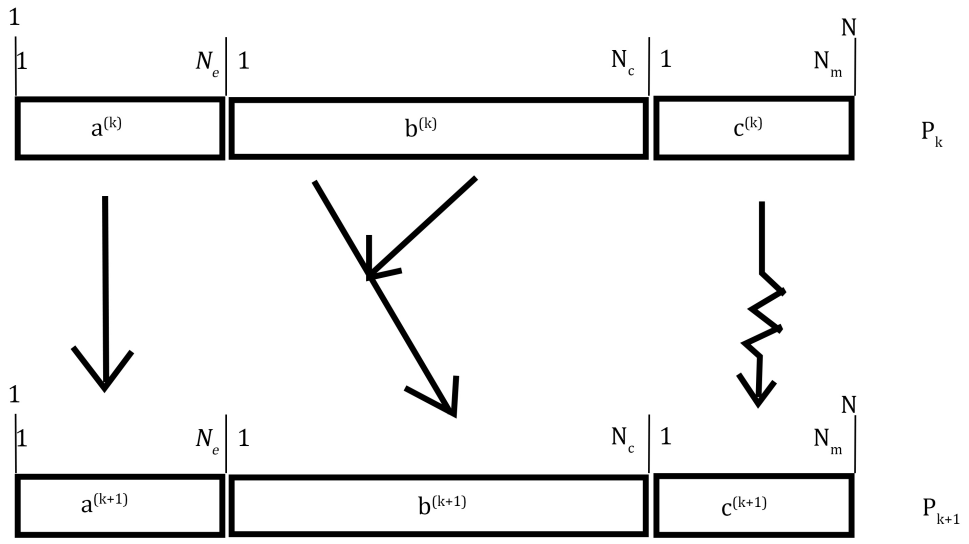


Figure 2.3: Population

In the following paragraphs, we will describe each step in more details.

Initialization (Step 1)

We will compare two types of initialization. Initial population is usually generated randomly, but we can also use quasi random numbers, which have better conditions for the initial population. **Population size** depends on the solved problem, it is given and constant in all generations. It can contain thousands of individuals.

Creating next generation (Step 2)

GA has a specific procedure of generating a new generation. There are 3 techniques to produce children:

- Elite
- Crossover
- Mutation

The ratio of genetic operators is always set in the algorithm. For example elite 20 %, crossover 60 % and mutation 20 %. It means 20 % of individuals will be selected, 60 % will be recombined by crossover and 20 % will mutate.

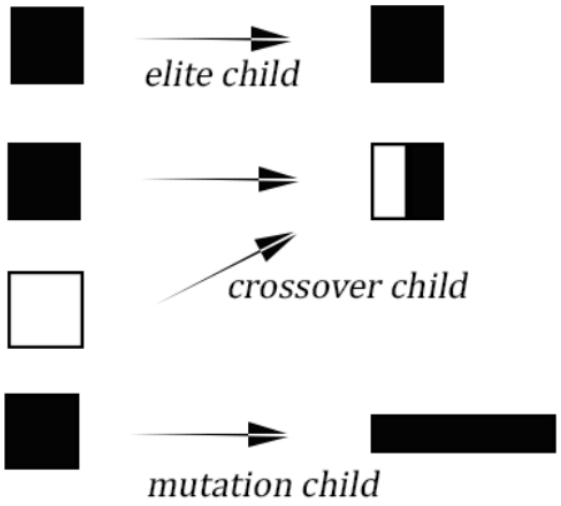


Figure 2.4: GA operators (MathWorks (2017))

Elite (Step 3)

For generating a new generation it is necessary to find the best individuals by using fitness-based process. Each individual is measured by fitness function and the best solutions are selected as the elite kids. Some methods calculate and rate fitness of all solutions, the others use only random samples of solutions and rate them to reach better calculation time.

After selection elite kids, only a few best individuals from the initial generation are in the next generation. To form a new generation, crossover and mutation will be used.

Crossover (Step 4)

One of genetic operations is crossover, that recombines parents' genotypes to produce new children. Crossover function combines two parents' vector in every step.

Selection function specify how the genetic algorithm chooses parents for the next generation. For example roulette selection chooses parents by sim-

ulating a roulette wheel, in which the area of the section of the wheel corresponding to an individual is proportional to the individual's expectation. The algorithm uses a random number to select one of the sections with a probability equal to its area. Parents are selected and used for crossover. This is used in matlab like the function for selection `selectionroulette` function.

There exist many types of crossover functions to produce new children. For example crossover function creates a random binary vector and combines two parents' vectors according to the binary vector. On positions, where are on binary vector ones, there will be applied the genes from first parent and where are zeros, there will be applied the genes form the second parent.

Example: if two parents have vectors:

$$X_1 = [1; 2; 3; 4; 5; 6; 7; 8]; Y_2 = [9; 10; 11; 12; 13; 14; 15; 16]$$

and random generated binary vector is

$$[0, 0, 1, 0, 1, 0, 1, 1]$$

then child vector will be:

$$b = [9; 10; 3; 12; 5; 14; 7; 8]$$

This function is used in matlab as the default crossover function for problems without linear constraint. It is called `crossoverscatterd`.

Mutation (Step 5)

The second genetic operations is mutation. This operation is based on making small random changes in individuals to get mutated children for new population.

For example, the mutation could be provided by mutationadaptfeasible function (default mutation function in matlab). This function is constrained and randomly generates directions as a follow up to previous successful and un-successful generations. Mutation function choses the direction, that satisfies the set bounds and constraints.

Termination (Step 6)

Termination conditions stop the generating process when one of them is reached. Usually in an algorithm setting there are more than one stopping condition. We will consider:

- Fitness limit - a solution is found when the score for the best point is less then fitness limit,
- Generations - maximum number of generation K is reached $K = k_{max}$,
- Stall generations - the algorithm stops, when there is no change in the best score over the StallGenLimit number of generations.

2.3 Test functions for optimization

Example 2.1. There exist many useful functions for testing the optimization algorithms. One of them is the Rastrigin's function. Formula of the function is

$$f(x) = Ad + \sum_{i=1}^{d} [x_i^2 - A\cos(2\pi x_i)],$$

where $A = 10$. This function has many local minimas and maximas, but it has a clear global minimum $f(0, 0) = 0$. In Figure 2.5, there is a graph of Rastrigin's fuction in the region $[-5, 5] \times [-5, 5]$ for $d = 2$.

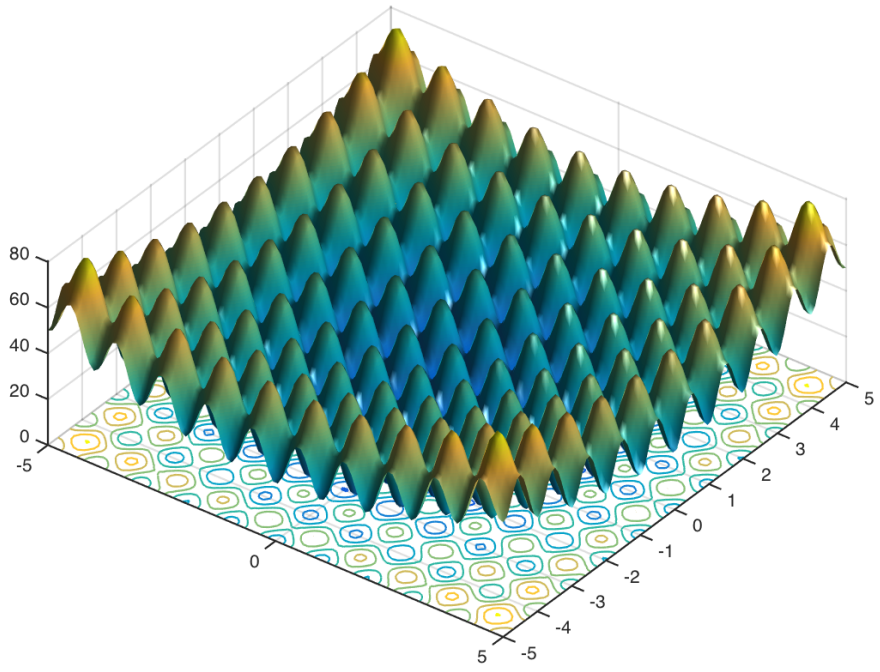


Figure 2.5: Rastrigin's function in 2D.

Example 2.2. Another example is the Ackley's function

$$f(x,y) = -20\exp[-0.2\sqrt{0.5(x^2+y^2)}] - \exp[0.5(\cos 2\pi x + \cos 2\pi y)] + e + 20.$$

It has also many local minimas and maximas but there exists only one global minimum $f(0,0) = 0$. A graph of Ackley's function in the region $[-2,2] \times [-2,2]$ is in Figure 2.6.

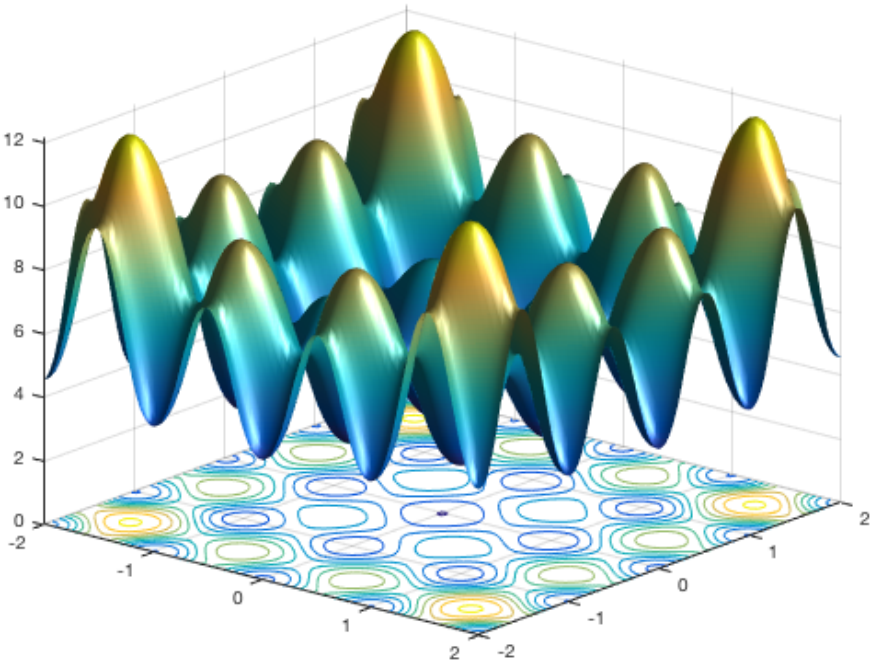


Figure 2.6: Ackley's function in 2D.

Example 2.3. To test the behaviour of the algorithm for functions with multiple global minimums, we consider

$$f(x,y) = \sin(x) * \cos(y).$$

In Figure 2.7, there is a graph of fuction with multiple same global minimas and maximas in the region $[-5,5] \times [-5,5]$ for $d = 2$.

2.4 Stochastic volatility models

Stochastic volatility (SV) models have the main contribution to finance. They were developed to modify the Black & Scholes model for option pricing, that brings to Scholes the Nobel memorial price for economics sciences

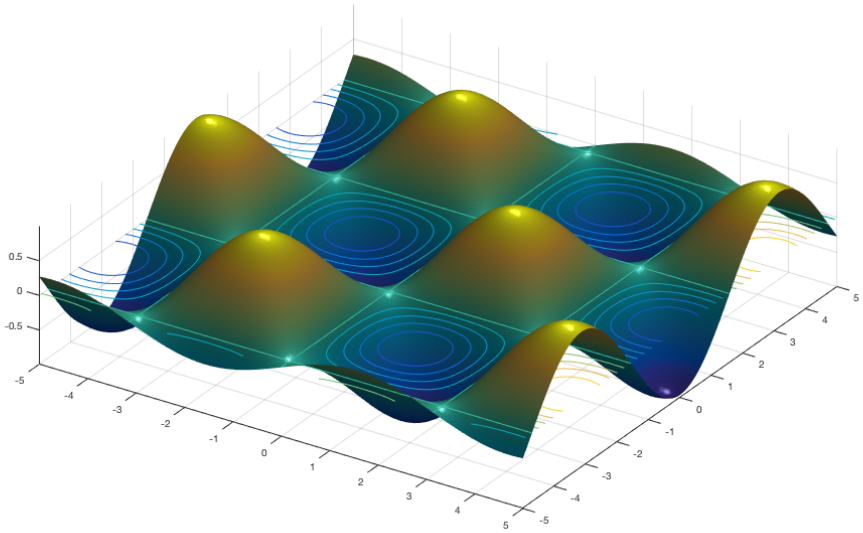


Figure 2.7: Function with the same global extremes.

in 1997. In Black & Scholes model, it is assumed that the modelled volatility of the underlying security is constant. In SV models, volatility is modelled as a stochastic process.

One of the models, that allows calibration to real market data, is the Heston model Heston (1993). We have the risk-neutral stock price model:

$$\begin{aligned} dS_t &= rS_t dt + \sqrt{v_t} S_t d\widetilde{W}_t^s, \\ dv_t &= k(\theta - v_t)dt + \sigma\sqrt{v_t} d\widetilde{W}_t^v, \\ d\widetilde{W}_t^s d\widetilde{W}_t^v &= \rho dt, \end{aligned}$$

with initial conditions $S_0 \geq 0$ and $v_0 \geq 0$. S_t is the price of the underlying asset at the time t, instant variance v_t at the time t, risk free rate r, long term average price variance θ, the rate k which v_t reverse to θ and σ is the volatility of the volatility. $(\widetilde{W}^s, \widetilde{W}^{v\cdot})$ is a two dimensional Wiener process under the risk of a neutral measure $\widetilde{P}$ with instant correlation ρ.

A price of the European call option with strike price K, time to maturity τ can be calculated as

$$V = S - Ke^{-r\tau}\frac{1}{\pi}\int_{0+i/2}^{\infty+i/2} e^{-ikX}\frac{\hat{F}(k,v,\tau)}{k^2 - ik}\mathrm{d}k, \tag{2.1}$$

where $X = \ln(S/K) + r\tau$ and

$$\hat{F}(k, v, \tau) = \exp\left(\frac{2\kappa\theta}{\sigma^2}\left[q\, g - \ln\left(\frac{1 - he^{-\xi q}}{1 - h}\right)\right] + \right.$$
$$\left. + vg\left(\frac{1 - e^{-\xi q}}{1 - he^{-\xi q}}\right)\right),$$

where

$$g = \frac{b - \xi}{2}, \quad h = \frac{b - \xi}{b + \xi}, \quad q = \frac{\sigma^2\tau}{2},$$
$$\xi = \sqrt{b^2 + \frac{4(k^2 - ik)}{\sigma^2}},$$
$$b = \frac{2}{\sigma^2}\Big(ik\rho\sigma + \kappa\Big).$$

The main intention of calibrating model to real market data is to minimize the pricing errors between the real market price and model price. This error can be measured by nonlinear least square method

$$\min G(\Theta),$$

$$G(\Theta) = \sum_{i=1}^{N} w_i |v_i^{\Theta}(\tau_i, K_i) - v_i^*(\tau_i, K_i)|^2,$$

where N is a number of option prices, w_i is a weight, v_i^* is the real market price of option and v_i^{Θ} is the calculated price by the model, where Θ is a vector of model parameters, here $\Theta = (v_0, \kappa, \theta, \sigma, \xi)$.

The function G may have more than one global minimum and it is not clear to decide, if a unique minimum can be found by gradient based methods. There is a high probability, that the method ends up in a local minimum. According to Mrázek et al. (2016) a suitable combination of a global and a local optimizer is the most convenient way of calibration. Global optimizers are required in order to find a suitable initial guess for gradient-based local optimizers. In this thesis, we will focus on the global optimization part of the calibration. For this purpose we will use GA and its quasi modification.

The main intention is to compare the real market results with a calibration of Heston model for our real data. We will need to define 5 parameters to calibrate the model. Vector of model parameters is defined:

$$\Theta = (v_0, \kappa, \theta, \sigma, \xi),$$

where v_0 is initial volatility, κ is a mean reversion rate, θ is an average volatility, σ is a volatility of volatility and ξ is a corelation koeficient.

We measure two types of errors, as a criterion for the performance evaluation of the optimizing methods. The **average of absolute relative errors** across all strikes and maturities

$$AARE(\Theta) = \frac{1}{N} \sum_{i=1}^{N} \frac{|v_i^{\Theta} - v_i^*|}{v_i^*}$$

and **maximum absolute value of relative errors** (for $i = 1, ..., N$)

$$MARE(\Theta) = \max_i \frac{|v_i^{\Theta} - v_i^*|}{v_i^*}.$$

We also have to define weights w_i, that set the biggest weight to the most liquid quotes on the market. There exist many types of weight function. For example:

$$w_i = \frac{|\delta_i|^{-1}}{\sum_{j=1}^{N} |\delta_j|^{-1}},$$

$$w_i = \frac{\delta_i^{-2}}{\sum_{j=1}^{N} \delta_j^{-2}},$$

$$w_i = \frac{1}{N},$$

where δ_i is the bid ask spread.

3 Results

We test modification of GA with quasi random initial population on test functions.

3.1 Test function

The algorithm has found global minimum in all our test function examples. In Figure 3.1 there is Rastrigin's function (Example 2.1). In Figure 3.2 we can see results for the Ackley's function (Example 2.2) and in Figure 3.3 there is the function with multiple same global minimums (Example 2.3). In first two examples we can see that all individuals converge to global minimum. But in the last example, there are more than one same global minimums, we can see that the algorithm finds a different minimums in the first steps, but later, the optimizer choses only one global minimum. The primary aim of the algorithm was find one global minimum, not all of them.

3.2 Stochastic volatility model

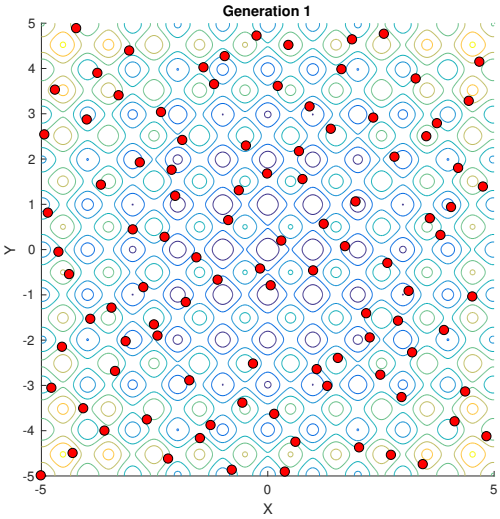
Data description

We consider the following real market data set for comparison of the EA and QEA: 97 ODAX call options traded on March 18, 2013 ranging from 86.5 percent to 112.0 percent moneyness across 5 maturities from ca. 13.5 weeks to 1.75 years. This data were gained form Bloomberg's Option Monitor. They consist of call contracts on the Deutsche Boerse AG German Stock Index (DAX).

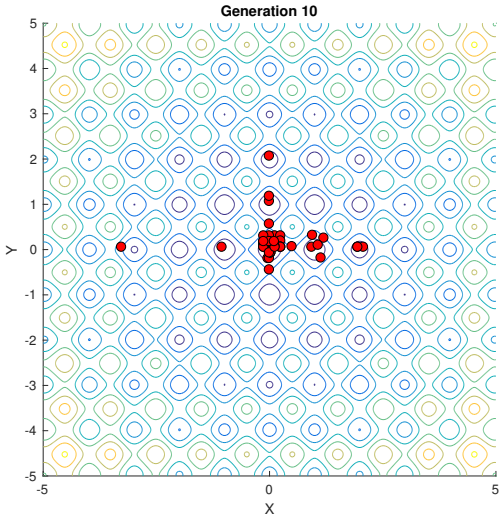
Illustration of our real market data is shown in Figure3.4.

Model calibration

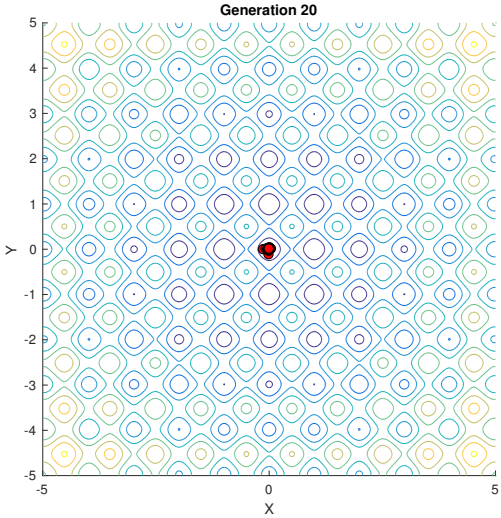
We try to search global minimum of $G(\Theta)$ that uses real market data. First we use a global optimization that is provided by GA and the second step is



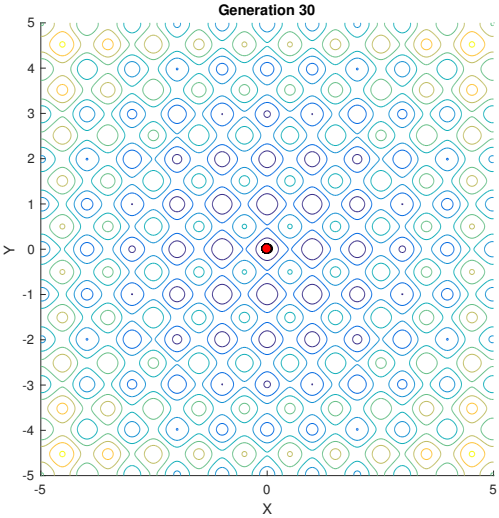
(a) Rastrigin's function in the first generation



(b) Rastrigin's function in the 10^{th} generation

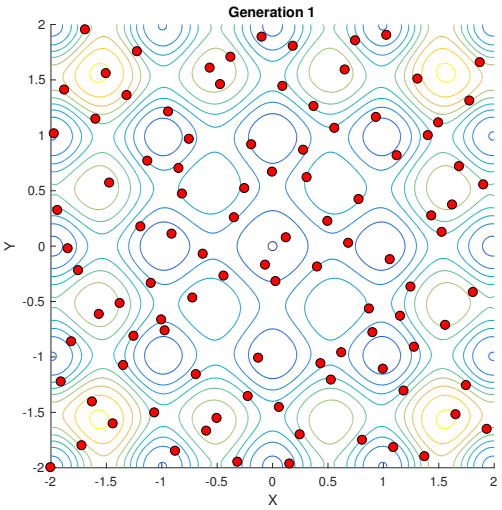


(c) Rastrigin's function in the 20^{th} generation

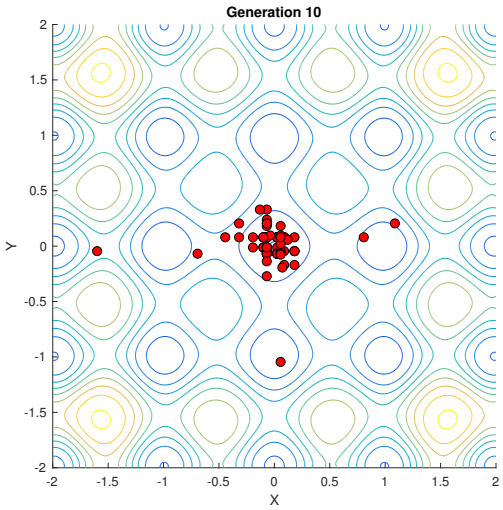


(d) Rastrigin's function in the 30^{th} generation

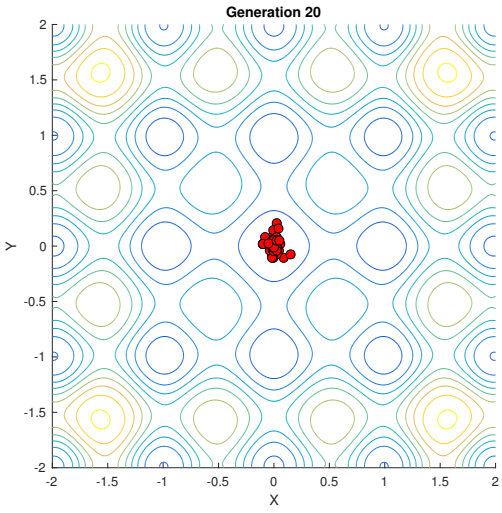
Figure 3.1: Rastrigin's function (Example 2.1)



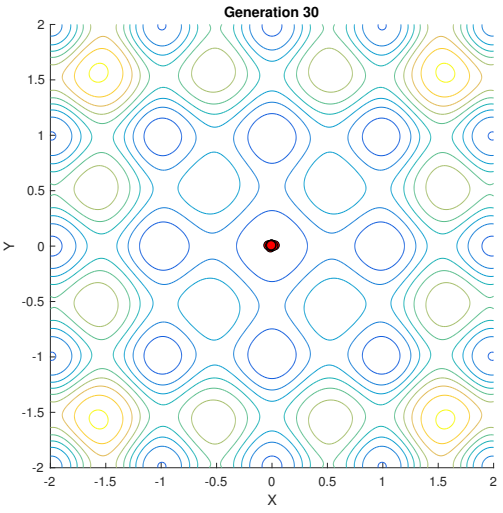
(a) Ackley's function in the first generation



(b) Ackley's function in the 10^{th} generation

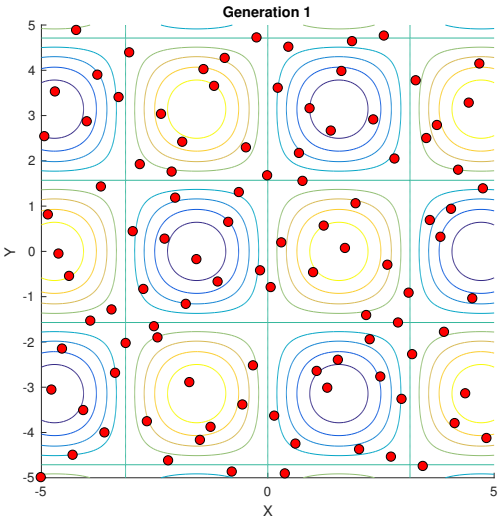


(c) Ackley's function in the 20^{th} generation

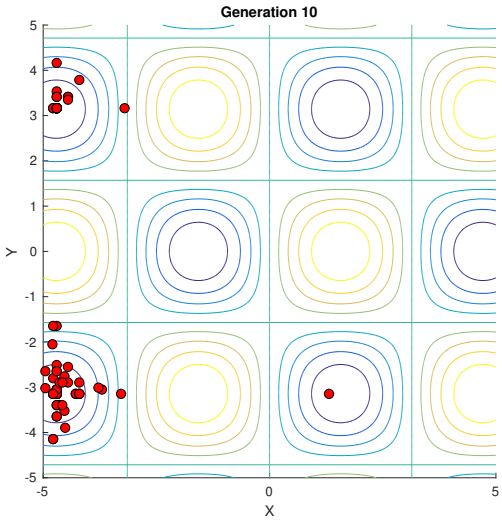


(d) Ackley's function in the 30^{th} generation

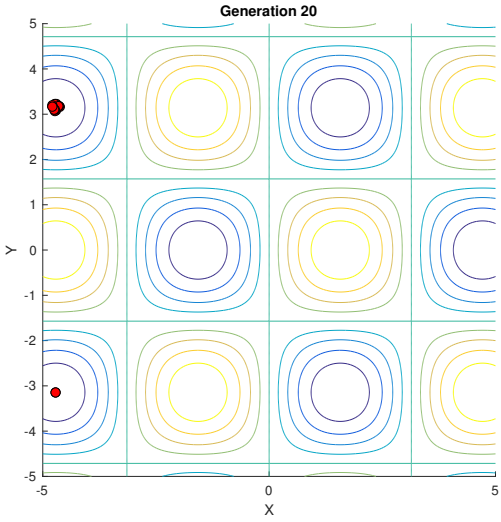
Figure 3.2: Ackley's function (Example 2.2)



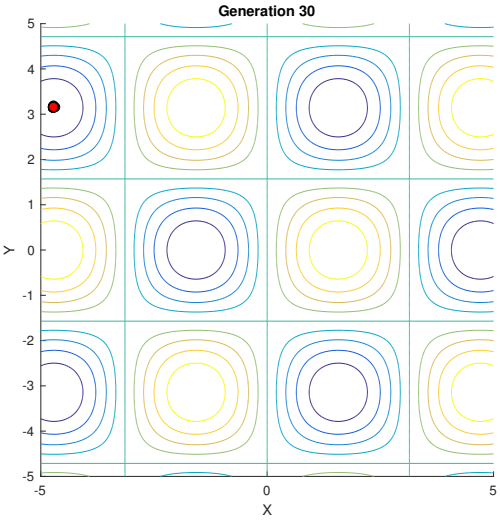
(a) Function with the same multiple global minimums in the first generation



(b) Function with the same multiple global minimums in the 10^{th} generation



(c) Function with the same multiple global minimums in the 20^{th} generation



(d) Function with the same multiple global minimums in the 30^{th} generation

Figure 3.3: Function with the same multiple global minimums (Example 2.3)

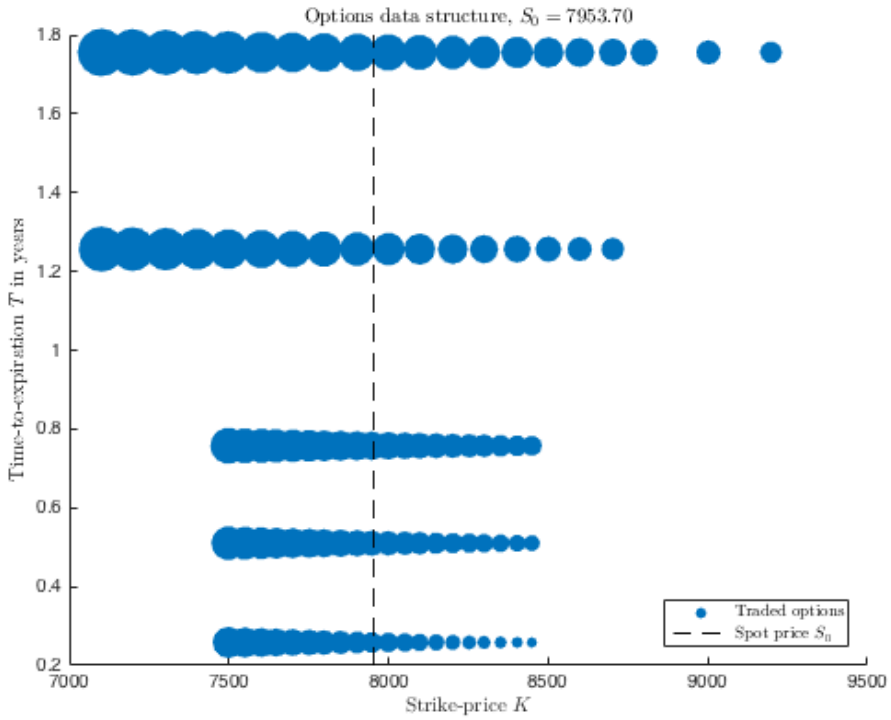


Figure 3.4: Structure of our real market data, where the center of each point corresponds to the strike/maturity pair of the traded contract. The diameter of every point is proportionate to the option premium.

a local optimization. We focus especially on the global optimization part.

We compare two types of optimization of our real market data. In Figure 3.5 there are results with randomly generated initial population. In the second Figure 3.6 there are results of optimization with quasi random initial population.

According to Mrázek – Pospíšil (2017), it's sufficient to consider only 10 generations in the GA part of the calibration. In Figures 3.5 and 3.6 there are results of our optimization. In the first column there are numbers of generation, in the second column are numbers of fitness function evaluations in each generation, in the third column there are the best scores from the generation, in the fourth column there are the means of scores in each generation and in the last column there are numbers of the stalled generations.

Global optimizer ends after 10 generations in both examples. The local optimizer always find minimum in a few additional steps in relatively short

time.

```
Loaded QRNG: Halton (Dimensions: 5, Leap: 0, Skip: 8623, Scramble: RR2, State: 851)
RUN 1 (ODAX-2013-03-18.txt) - start (19-Jun-2017 15:46:00)

running GA

                         Best              Mean   Stall
Generation    f-count    f(x)              f(x)   Generations
    1           100       176.18085       14785.913        0
    2           150       48.067388       8320.0924        0
    3           200       20.342888       6203.4475        0
    4           250       20.137936       4455.4796        0
    5           300       19.541498       2952.9149        0
    6           350       19.541498       1919.8621        1
    7           400       19.541498       670.96868        2
    8           450        19.22375       708.86283        0
    9           500        19.22375       233.22385        1
   10           550       19.063645       186.28168        0
Optimization terminated: maximum number of generations exceeded.
running LSQ (after GA completed)

                                        Norm of      First-order
 Iteration  Func-count     f(x)          step          optimality
     0          6          19.0636                     1.17e+03
     1         12          8.18027       0.698524           505
     2         18           2.9505       0.465011          92.4
     3         24          1.82664        2.25243           515
     4         30          1.01704     0.00153446          5.76
     5         36          1.01704        2.66543          5.76
     6         42         0.825494       0.666357            39
     7         48         0.678631        1.33271           186
     8         54         0.545055    0.000700499         0.861
     9         60         0.545055        1.02924         0.861
    10         66         0.506404       0.257309          10.3
    11         72          0.46189       0.514618          45.5
    12         78         0.438785       0.343594          43.1
    13         84         0.421822       0.183364          6.13
    14         90         0.386822       0.918647          70.1
    15         96         0.350085       0.102341          3.73
    16        102         0.350085        0.99564          3.73
    17        108         0.324347        0.24891          11.5
    18        114         0.290328       0.466378          67.7
    19        120         0.242226      0.0925961          8.23
    20        126         0.242226       0.442566          8.23
    21        132         0.221494       0.110641          6.62
    22        138         0.183765       0.221283          31.3
    23        144         0.149734       0.142609          26.7
    24        150         0.117053       0.131125          21.1
    25        156        0.0895527       0.126192          20.8
    26        162        0.0677253      0.0957295            15
    27        168        0.0530991      0.0919656          14.8
    28        174        0.0443358      0.0637983          8.42
    29        180        0.0405617      0.0541852          5.88
    30        186        0.0394869      0.0279594          1.74
    31        192        0.0393757      0.0118006         0.321
    32        198         0.039374    0.000817649       0.00556
    33        204         0.039374    3.35029e-05         4e-05

Local minimum possible.

lsqnonlin stopped because the final change in the sum of squares relative to
its initial value is less than the selected value of the function tolerance.
```

Figure 3.5: Calibration model on our real market data using quasi random initial population.

The main contrast between our two techniques is in the first two gen-

```
Loaded QRNG: Halton (Dimensions: 5, Leap: 0, Skip: 8623, Scramble: RR2, State: 751)
RUN 1 (ODAX-2013-03-18.txt) - start (19-Jun-2017 15:44:07)

running GA

                        Best              Mean   Stall
Generation    f-count   f(x)               f(x)   Generations
    1           100       40.892562       10752.294        0
    2           150       32.857768       7871.4501        0
    3           200       31.986298       4655.9775        0
    4           250       31.986298       3409.7357        1
    5           300       31.661548       3165.0168        0
    6           350       31.661548       1979.1272        1
    7           400       31.661548       2738.8956        2
    8           450       31.221656       2093.5783        0
    9           500       31.054038       2720.1455        0
   10           550        30.78757       491.39209        0
Optimization terminated: maximum number of generations exceeded.
running LSQ (after GA completed)

                                        Norm of      First-order
 Iteration  Func-count     f(x)          step          optimality
     0          6          30.7876                     1.87e+03
     1         12          5.88526        2.65971           28.7
     2         18           2.3231        0.85711            128
     3         24           2.3231        5.97436            128
     4         30          1.67496        1.49359            109
     5         36          1.67496        3.36664            109
     6         42          1.29736       0.746795           53.5
     7         48         0.974567        1.49359            305
     8         54         0.526378       0.422189           83.7
     9         60         0.484075       0.511668            119
    10         66         0.408514        0.12133           11.4
    11         72         0.385357       0.616031           60.3
    12         78         0.355071       0.113276           2.89
    13         84         0.355071        1.13654           2.89
    14         90         0.326654       0.284136             14
    15         96         0.287837       0.403597           49.6
    16        102         0.251402       0.124995           10.9
    17        108         0.234089       0.385564           81.5
    18        114         0.159106      0.0611759           8.24
    19        120         0.137015      0.0963911           8.54
    20        126         0.112245       0.192782           39.4
    21        132        0.0790459      0.0663938             11
    22        138          0.06298       0.132234           26.6
    23        144        0.0479573      0.0488586           6.21
    24        150        0.0424932       0.084917           12.9
    25        156        0.0397268       0.023962           1.69
    26        162        0.0393911       0.024126            1.1
    27        168         0.039374     0.00159863           0.02
    28        174         0.039374    6.72516e-05       0.000211

Local minimum possible.
```

Figure 3.6: Calibration model on our real market data using random initial population.

erations. When we use the quasi random initial population (Figure 3.5), there is a big jump from the first population to the second one. It happens because of a good distribution of individuals by the creation function.

The second results (Figure 3.6), where the initial population is generated by random generator, go fluently but slowly in the direction to minimum.

We tested each process 1000 times. The relative improvement from quasi random initial population to second generation was 38%. We also tested the improvement from random initial population to the second generation. There was only 32% change. We also tested the improvement form initial population to 3^{rd} generation. There was change of 60% by quasi random and 56% by random initial population.

4 Conclusion

The main intention of this thesis was to compare quasi random and random numbers in evolutionary algorithms. We introduced evolutionary algorithms, focused on initial population generation and introduced the global optimization problem that we solved using quasi-evolutionary algorithm.

We focused on quasi random initial population, that has a significant influence on searching global minimums, because of more uniformly distributed individuals (we used quasi-random sequences with low discrepancy).

We set and commented each step of the genetic algorithm, set 3 test functions and introduced the methodology how to calibrate stochastic volatility models to real market data.

Implementation part of this thesis was written in Matlab. Inspired by the Matlab `ga`, we implemented own function `myga` so that it can work also with the quasi random initial population. The code was tested using 3 test functions. We also modified this code for calibration of stochastic volatility models to real market data.

Using Matlab, we compared quasi random initial population and random initial population. We found out, that quasi random initial population improves the technique of algorithm and terminates in shorter time.

We applied this algorithm to real market data calibration, specifically we considered 97 ODAX call options traded on March 18, 2013. This data were gained form Bloomberg's Option Monitor. Using quasi random sequences in initial population generation we slightly improved the results obtained by Mrázek et al. (2016).

A Appendix

A.1 Attachments

On CD in the attachment there are all implemented matlab codes.

Matlab codes	
calibration_myga_test.m	Calibration problem with my GA for 1000 runs
calibration_myga.m	Calibration problem with my GA
generation_plots.m	Figure of generation
hestonError.m	Error in Heston model calibration
HestonLewis.m	Function evaluation by Heston Lewis formula
hestonUtility.m	Heston calibration - utility function
meanf.m	Robust mean function
myCreationFcn.m	Simple implementation of creation function
myCrossoverFcn.m	Simple implementation of crossover function
myga.m	Simple implementation of genetic algorithm
mygademo.m	Initiator of myga.m
mygaplot.m	Plot function for ga.m
mygaplotbest.m	Plot function with highlight of the best individuals
myMutationFcn.m	Simple implementation of mutation function
mypeaks.m	My own test function
myQuasiCreationFcn.m	Simple implementation of creation function that uses quasi random numbers
mySelectionFcn.m	Simple implementation of selection function
option_structure.m	Options for figure
run.m	Initiator of calibration problem
userFcn.m	My own test function with multiple global minimas

Bibliography

Eiben, A. E. – Smith, J. E. *Introduction to Evolutionary Computing*. Springer Berlin Heidelberg, 2003. doi: 10.1007/978-3-662-05094-1. Available from: http://dx.doi.org/10.1007/978-3-662-05094-1.

Ellegård, A. *Darwin and the general reader: the reception of Darwin's theory of evolution in the British periodical press, 1859-1872*. University of Chicago Press, 1958.

Fogel, L. J. *Intelligence Through Simulated Evolution: Forty Years of Evolutionary Programming*. John Wiley & Sons, Inc., 1999. ISBN 0-471-33250-X.

Fogel, L. – Owens, A. – Walsh, M. *Artificial Intelligence Through Simulated Evolution*. John Wiley & Sons, 1966. Available from: https://books.google.cz/books?id=75RQAAAAMAAJ.

Heston, S. L. A closed-form solution for options with stochastic volatility with applications to bond and currency options. *Rev. Financ. Stud.* 1993, 6, 2, p. 327–343. ISSN 0893-9454. doi: 10.1093/rfs/6.2.327.

Holland, J. H. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

Kazimipour, B. – Li, X. – Qin, A. K. A review of population initialization techniques for evolutionary algorithms. In *2014 IEEE Congress on Evolutionary Computation (CEC)*, p. 2585–2592. IEEE, 2014. doi: 10.1109/CEC.2014.6900618. ISBN 978-1-4799-1488-3.

MathWorks. *How the Genetic Algorithm Works* [online]. 2017. GA in Matlab. Available from: https://www.mathworks.com/help/gads/how-the-genetic-algorithm-works.html.

Mrázek, M. – Pospíšil, J. Calibration and Simulation of Heston Model. *Open Math.* 2017, 15, 1, p. 679–704. ISSN 2391-5455. doi: 10.1515/math-2017-0058.

Mrázek, M. – Pospíšil, J. – Sobotka, T. On calibration of stochastic and fractional stochastic volatility models. *European J. Oper. Res.* 2016, 254, 3, p. 1036–1046. ISSN 0377-2217. doi: 10.1016/j.ejor.2016.04.033.

Rechenberg, I. *Evolutionsstrategie: optimierung technischer systeme nach prinzipien der biologischen evolution*. Frommann-Holzboog, 1973.