Chapter 7

Dynamic Hedging Using Generated Genetic Programming Implied Volatility Models

Fathi Abid, Wafa Abdelmalek and Sana Ben Hamida

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/48148

1. Introduction

One challenge posed by financial markets is to correctly forecast the volatility of financial securities, which is a crucial variable in trading and risk management of derivative securities. Dynamic hedging is very sensitive to volatility forecast and good hedges require accurate estimate of volatility. Implied volatilities, generated from option markets, can be particularly useful in such contents as they are forward-looking measures of the market's expected volatility during the remaining life of an option [1, 2]. Since there is no explicit formula available to compute directly the implied volatility, the latter can be obtained by inverting the option pricing model. On the contrary, the genetic programming offers explicit formulas which can compute directly the implied volatility. This volatility forecasting method should be free of strong assumptions regarding underlying price dynamics and more flexible than parametric methods. This paper proposes a non parametric approach based on genetic programming to improve the accuracy of the implied volatility forecast and consequently the dynamic hedging.

Genetic Programming [3] is an optimization technique which extends the basic genetic algorithms [4] to process non-linear problem structure. In genetic programming, solutions are represented as tree structures that can vary in size and shape, rather than fixed length character strings as in genetic algorithms. This means that genetic programming can be used to perform optimization at a structural level. In the standard genetic programming, the entire population of function-trees is evaluated against the entire training data set, so the number of function-tree evaluations carried out per generation is directly proportional to both the population size and the size of the training set. Genetic programming can encounter the problem of managing training sets which are too large to fit into the memory of computers, and then the realization of predictors. In machine learning, the practiced solution to learn large data set is the application of resampling techniques, such as, bagging

[5], boosting [6] and arcing [7]. However, these techniques require that the entire data sets be stored in the main memory. When applied to large data sets, this approach could be impractical. In this paper, we proposed to split data into smaller subsets. First, the genetic programming is run separately on all training sub-samples. Such approach is called static training-subset selection method [8]; it might provide local solutions not adaptive to the entire enlarged data set. Alternatively, a dynamic training approach is developed. It allows genetic programming to learn simultaneously on all training sub-samples and it implies a new parameter added to the basic genetic programming algorithm which is the number of generations to change sample. This approach lightens the training task for the genetic programming and favors the discovery of solutions that are more robust across different learning data samples and seem to have better generalization ability. Comparison between generated models using static and dynamic selection methods reveals that, the dynamic approach improves the forecasting performance of the generated models using genetic programming. The best forecasting implied volatility models are selected according to total MSE criterion. They are used to compute hedge factors and implement dynamic hedging strategies. According to the average hedging errors, the genetic programming presented accurate hedging performance compared to that of Black-Scholes model.

The rest of the paper is organized as follows: section 2 provides background information regarding related works in forecasting volatility and dynamic hedging, section 3 describes research design and methodology used in this paper, section 4 reports experimental results and finally section 5 concludes.

2. Related works

Traditional parametric methods have limited success in estimating and forecasting volatility as they are dependent on restrictive assumptions and difficult to estimate. Several machine learning techniques have been recently used to overcome these difficulties such as artificial neural networks and evolutionary computation algorithms. In particular, genetic programming has been successfully applied to forecast financial time series [9,10].

This paper makes an initial attempt to test whether the hedger can benefit more by using generated genetic programming implied volatilities instead of Black-Scholes implied volatilities in conducting dynamic hedging strategies.

Changes in asset prices is not the only risk faced by market participants, instantaneous changes in market implied volatility can also bring a hedging portfolio significantly out of balance. Extensive research during the last two decades has demonstrated that the volatility of stocks is not constant over time [11]. The Autoregressive Conditional Heteroskedasticiy (ARCH) and the Generalized ARCH (GARCH) models are introduced [12,13] to describe the evolution of the asset price's volatility in discrete time. Econometric tests of these models clearly reject the hypothesis of constant volatility and find evidence of volatility clustering over time. In the financial literature, stochastic volatility models have been proposed to model these effects in a continuous-time setting [14-17]. Although these models improve the benchmark Black-Scholes model, they are complex because they require strong

assumptions and computational effort to estimate parameters and stochastic process. As mentioned in [18], traditional financial engineering methods based on parametric models such as the GARCH model family, seem to have difficulty to improve the accuracy in volatility forecasting due to their rigid as well as linear structure. Using its basic and flexible tree-structured representation, genetic programming is capable of solving non-linear problems. In the context of forecasting volatility, most of research papers have focused on forecasting historical volatility based on past returns in different markets. Using historical returns of Nikkei 225 and S&P500 indices, Chen and Yeh [19] have applied a recursive genetic programming approach to estimate volatility by simultaneously detecting and adapting to structural changes. Results have shown that the recursive genetic programming is a promising tool for the study of structural changes. Using high frequency foreign exchange USD-CHF and USD-JPY time series, Zumbach et al. [20] have compared the genetic programming forecasting accuracy to that of historical volatilities, the GARCH (1,1), FIGARCH and HARCH models. According to the root-mean squared errors, the generated genetic programming volatility models did consistently outperform the benchmarks. Similarly, Neely and Weller [21] have tested the forecasting performance of genetic programming for USD-DEM and USD-YEN daily exchange rates against that of GARCH (1,1) model and a related RiskMetrics volatility forecast over different time horizons, using various accuracy criteria. While the genetic programming rules did not usually match the GARCH (1,1) or RiskMetrics models' MSE or R^2, its performance on those measures was generally close. But, the genetic programming did consistently outperform the GARCH model on mean absolute error (MAE) and model error bias at all horizons. Overall, on some dimensions the genetic programming has produced significantly superior results. Applying a combination of theory and techniques such as wavelet transform, time series data mining, Markov chain based discrete stochastic optimization, and evolutionary algorithms genetic algorithms and genetic programming, Ma et al. [22,23] have proposed a systematic approach to address specifically non linearity problems in the forecast of financial indices using intraday data of S&P100 and S&P500 indices. As a result, accuracy of forecasting has reached an average of over 75% surpassing other publicly available results on the forecast of any financial index. Abdelmalek et al. [8] have extended the studies mentioned earlier by forecasting the implied volatility of Black-Scholes from the S&P500 index call options instead of historical volatility using a static training of genetic programming. The performance of generated genetic programming volatility forecasting models is compared between time series samples and moneyness-time to maturity classes. Using Total and out-of-sample mean squared errors (MSE) as forecasting performance measures, the time series model seems to be more accurate in forecasting implied volatility than moneyness-time to maturity models.

Option contracts prices are affected by new information and changes in expectations as much as they are by changes in the value of the underlying index. If traders have perfect foresight on forward volatility, then dynamic hedging would be essentially riskless. In practice, continuous hedging is impossible, but the convexity of option contract allows for adjustments in the exposure to higher-order sensitivities of the model, such as gamma, vega,

etc. Most of the existing literature on hedging a target contract using other exchange-traded options focuses on static strategies, motivated at least in part by the desire to avoid the high costs of frequent trading. The goal of static hedging is to construct a buy-and-hold portfolio of exchange traded claims that perfectly replicates the payoff of a given over-the-counter product [24,25]. The static hedging strategy does not require any rebalancing and therefore, it does not incur significant transaction costs. Unfortunately, the odds of coming up with a perfect static hedge for a given over-the-counter claim are small, given the limited number of exchange listed option contracts with sufficient trading volume. In other words, the static hedge can only be efficient if traded options are available with sufficiently similar maturity and moneyness as the over-the-counter product that has to be hedged. Under a stochastic volatility, a perfect hedge can in principle be constructed with a dynamically rebalanced portfolio consisting of the underlying and one additional option. In practice, the dynamic replication strategy for European options will only be perfect if all of the assumptions underlying the Black-Scholes formula hold. For general contingent claims on a stock, under market frictions, the delta might still be used as first-order approximation to set up a riskless portfolio. However, if the volatility of the underlying stock varies stochastically, then the delta hedging method might fail severely. A simple method to limit the volatility risk is to consider the volatility sensitivity vega of the contract. The portfolio will have to be rebalanced frequently to ensure delta-vega neutrality. With transaction costs, frequent rebalancing might result in considerable losses. In practice, investors can rebalance their portfolios only at discrete intervals of time to reduce transactions costs.

Non parametric hedging strategies as an alternative to the existing parametric model based-strategies, have been proposed [26,27]. Those studies estimated pricing formulas by nonparametric or semi-parametric statistical methods such as neural networks and kernel regression, and they measured their performance in terms of delta-hedging. Few researches have focused on the dynamic hedging using genetic programming, however. Chen et al. [28] have applied genetic programming to price and hedge S&P500 index options. By distinguishing the case in-the-money from the case out-of-the-money, the performance of genetic programming is compared with the Black-Scholes model in terms of hedging accuracy. Based on the post-sample performance, it is found that in approximately 20% of the 97 test paths, genetic programming has lower tracking error than the Black-Scholes formula.

Based on the literature survey, one can conclude that the genetic programming could be used to efficiently forecast volatility and implement accurate dynamic hedging strategies, which opens up an alternative path besides other data-based approaches.

3. Research design and methodology

Accurate volatility forecasting is an essential element in conducting good dynamic hedging strategies. The first thrust of this paper deals with generation of implied volatility from option markets using static and dynamic training of genetic programming, respectively. While the static training [8] is characterized by training the genetic programming independently on a single Sub-sample, the dynamic training allows the genetic

programming to train on the entire data sub samples simultaneously rather than just a single subset by changing the training Sub-sample during the run process. This permits to improve the robustness of genetic programming to generate general models adaptive to all training samples. The second thrust of this paper is to study the accuracy of the generated genetic programming implied volatility models in terms of dynamic hedging. Since the true volatility is unobservable, it is impossible to assess the accuracy of any particular model; forecasts can only be related to realized volatility. In this paper, we assume that the implied volatility is a reasonable proxy for realized volatility, to generate forecasting implied volatility models using genetic programming and then to analyze the implications of this predictability for hedging purposes.

Figure 1 illustrates the operational procedure to implement the proposed approach.

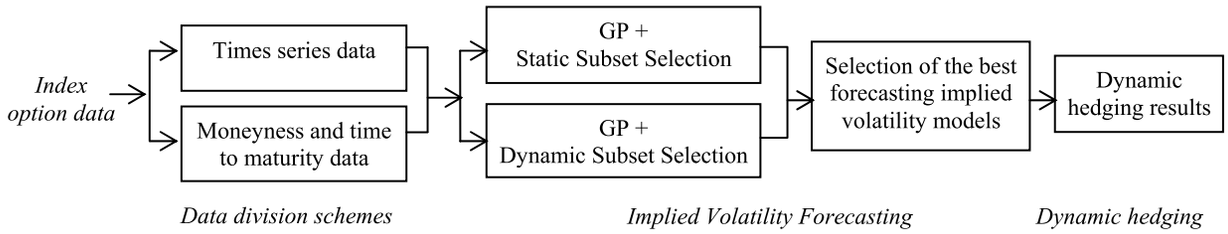


Figure 1. Description of the proposed approach's implementation

The operational procedure consists of the following steps: The first step is devoted for the data division schemes. The second step deals with the implementation of genetic programming[1] (GP), the application of training subset selection methods and the selection of the best forecasting implied volatility models. The last step is dedicated to dynamic hedging results.

3.1. Data division schemes

Data used in this study consist of daily prices for the European-style S&P 500 index calls and puts options traded on the Chicago Board of Options Exchange from 02 January to 29 August 2003. The data base include the time of the quote, the expiration date, the exercise price and the daily bid and ask quotes for call and put options. Similar information for the underlying S&P 500 index is also available on a daily basis. S&P500 index options are among the most actively traded financial derivatives in the world. The minimum tick for series trading below 3 is 1/16 and for all other series 1/8. Strike price intervals are 5 points, and 25 points for far months. The expiration months are three near term months followed by three additional months from the March quarterly cycle (March, June, September, and December). Following a standard practice, we used the average of an option's bid and ask price as a stand-in for the market value of the option. The risk free interest rate is approximated by using 3 month US Treasury bill rates. It is assumed that there are no transaction costs and no dividend.

1 GP system is built around the Evolving Object library, which is an ANSI-C++ evolutionary computation Framework (EO library).

To reduce the likelihood of errors, data screening procedures are used [29,30]. We apply four exclusion filters to construct the final option sample. First, as implied volatilities of short-term options are very sensitive to small errors in the option price and may convey liquidity-related biases, options with time to maturity less than 10 days are excluded. Second, options with low quotes are eliminated to mitigate the impact of price discreteness on option valuation. Third, deep-in-the-money and deep-out-of-the money option prices are also excluded due to the lack of trading volume. Finally, option prices not satisfying the arbitrage restriction [31], $C \geq S - Ke^{-r\tau}$, are not included.

The final sample contains 6670 daily option quotes, with at-the-money (ATM), in-the-money (ITM) and out-of-the money (OTM) options respectively taking up 37%, 34% and 29% of the total sample.

In this paper, two data division schemes are used. The full sample is sorted first, by time series (TS) and second by moneyness-time to maturity (MTM). For time series, data are divided into 10 successive samples (S_1, S_2…S_{10}), each contains 667 daily observations. The first nine samples are used as training sub-samples. For moneyness-time to maturity, data are divided into nine classes with respect to moneyness and time to maturity criteria. According to moneyness criterion: A call option is said out-of-the money (OTM) if $S/K < 0.98$; at-the-money (ATM) if $S/K \in [0.98, 1.03[$; and in-the-money (ITM) if $S/K \geq 1.03$. According to time to maturity criterion: A call option is Short Term (ST) if $\tau < 60$ days; Medium Term (MT) if $\tau \in [60, 180]$ days; and Long Term (LT) if $\tau > 180$ days. Each class C_i is divided on training set C_i^L and test set C_i^T, which produces respectively nine training and nine test MTM sub-classes. Figure 2 illustrates the two division schemes.

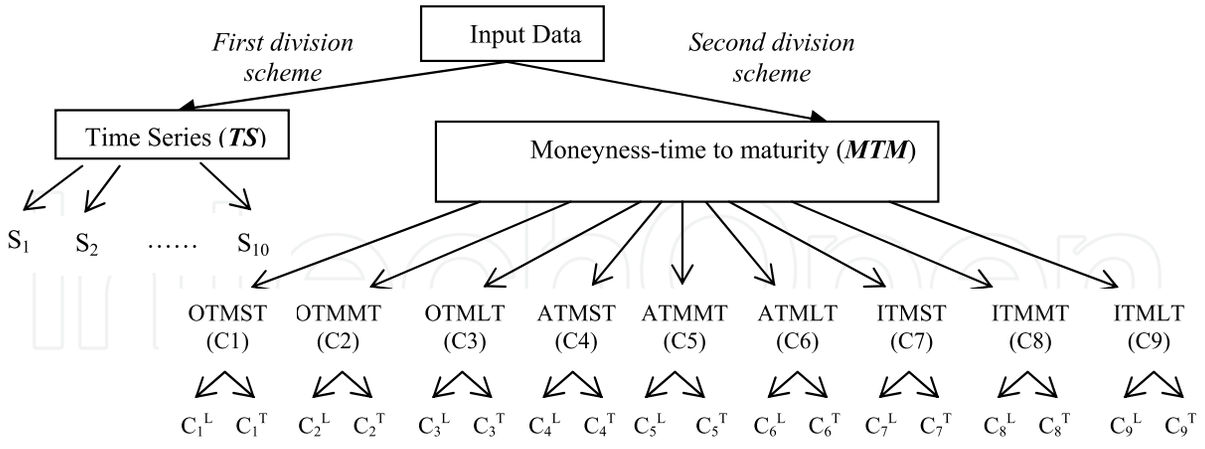


Figure 2. Data division schemes

3.2. Implied volatility forecasting using genetic programming:

This subsection describes the design of genetic programming and the experiments accomplished using the genetic programming method to forecast implied volatility. In the first experiment, the genetic programming is trained using static training-subset selection

method; in the second one, we used dynamic training-subset selection methods. We describe training and test samples used in these experiments.

3.2.1. The design of genetic programming:

Our genetic programming software is referred to as symbolic regression written in C++ language. It is designed to find a function that relates a set of inputs to an output without making any assumptions about the structure of that function. Symbolic regression was one of the earliest applications of genetic programming [3], and has continued to be widely studied [32-35]. The following pseudo code describes the genetic programming's algorithm structure used in this paper.

```
Initialize population
While (termination condition not satisfied) do
Begin
        Evaluate the performance of each individual according to the fitness criterion
        Until the offspring population is fully populated do
            - Select individuals in the population using the selection algorithm
            - Perform crossover and mutation operations on the selected individuals
            - Insert new individuals in the offspring population
    Replace the existing population by the new population
End while
Report the best solution found
End
```

Algorithm 1 Pseudo code of genetic programming

The genetic programming's algorithm structure consists of the following steps: nodes definition, initialization, fitness evaluation, selection, genetic operators (crossover and mutation) and termination condition.

Nodes Definition: The nodes in the tree structure of genetic programming can be classified into terminal (leaf) nodes and function (non-terminal) nodes. The terminal and function sets used are described in Table 1.

The terminal set includes the inputs variables, notably, the option price divided by strike price ($\frac{C}{K}$ for calls and $\frac{P}{K}$ for puts), the index price divided by strike price $\frac{S}{K}$ and time to maturity τ. The function set includes unary and binary nodes. Unary nodes consist of mathematical functions, notably, cosinus function (cos), sinus function (sin), log function (ln), exponential function (exp), square root function ($\sqrt{}$) and the normal cumulative distribution function (Φ). Binary nodes consist of the four basic mathematical operators, notably, addition (+), subtraction (-), multiplication ($\times$) and division (%). The basic division operation is protected against division by zero and the log and square root functions are protected against negative arguments.

Expression		Definition
Terminal Set	C/K	Call price / Strike price
	S/K	Index price / Strike price
	τ	Time to maturity
Function Set	+ (plus)	Addition
	- (minus)	Subtraction
	* (multiply)	Multiplication
	% (divide)	Protected division: x % y = 1 if y=0; x % y = x % y otherwise
	ln	Protected natural log: $\ln(x) = \ln(\lvert x \rvert)$
	Exp	Exponential function: $\exp(x) = e^x$
	Sqrt	Protected square root: $\sqrt{x} = \sqrt{\lvert x \rvert}$
	Ncdf	Normal cumulative distribution function Φ

Table 1. Terminal set and function set

Individuals are encoded as LISP S-expressions which can also be depicted as a parse tree. The search space for genetic programming is the space of all possible parse trees that can be recursively created from the terminal and function sets.

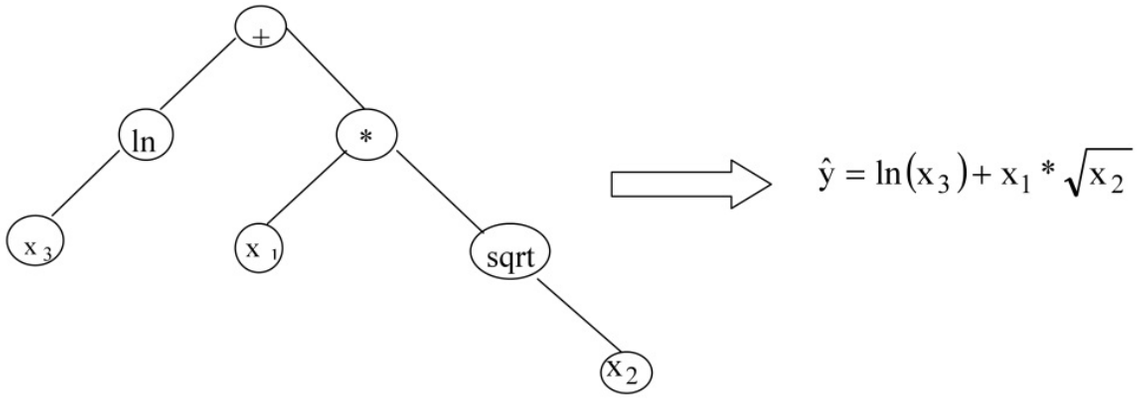


Figure 3. Example of a tree structure for GP and the corresponding functions

Initialization: The generated genetic programming volatility models are performed using a ramped half and half as initialization method [3]. This method involves generating an equal number of trees using a maximum initial depth that ranges from 2 to 6, as specified in Table 2. For each level of depth, 50% of the initial trees are generated via the full method and the

other 50% are generated via the grow method. In the full method, the initial trees have the property that every path from root to endpoint is of maximum depth. In the grow method, initial trees can be of various depths, subject to the constraint that they do not exceed the maximum depth.

Fitness function: The fitness function assigned to a particular individual in the population must reflect how closely the output of an individual program comes to the target function. In this paper, the Black-Scholes implied volatility σ_t^{BS} is used as target output. It is defined as the standard deviation which equates the Black-Scholes price C_{BS} [2] to the market option price C_t^* [36]:

$$\begin{aligned} &\exists! \quad \sigma_t^{BS}(K,T) \succ 0, \\ &C_{BS}\left(S_t,K,\tau,\sigma_t^{BS}(K,T)\right) = C_t^*(K,T) \end{aligned} \tag{1}$$

The generated genetic programming trees provide at each time t the forecast value $\hat{\sigma}_t$, and the fitness function used to measure the accuracy of forecast is the mean squared error (MSE) between the target (σ_t^{BS}) and forecasted ($\hat{\sigma}_t$) output volatility, computed as follows:

$$MSE = \frac{1}{N}\sum_{t=1}^{N}\left(\sigma_t^{BS} - \hat{\sigma}_t\right)^2 \tag{2}$$

Where, N is the number of data sample.

Selection: Based on the fitness criterion, the selection of the individuals for reproduction is done with the tournament selection algorithm. A group of individuals is selected from the population with a uniform random probability distribution. The fitness values of each member of this group are compared and the actual best is selected. The size of the group is given by the tournament size which is equal to 4, as indicated in Table 2.

Genetic operators: Crossover and mutation are the two basic operators which are applied to the selected individuals in order to generate new individuals for the next generation. As described in Figure 4, the subtree crossover creates new offspring trees from two selected parents by exchanging their sub-trees. As indicated in Table 2, the crossover operator is used to generate about 60% of the individuals in the population. The maximum tree size (measured by depth) allowed after the crossover is 17. This is a popular number used to limit the size of tree [3]. It is large enough to accommodate complicated formulas and works in practice.

---

[2] $C_{BS} = SN(d_1) - Ke^{-r\tau}N(d_2), d_1 = \dfrac{\ln\left(\dfrac{S}{K}\right) + \left(r + 0.5\sigma^2\right)\tau}{\sigma\sqrt{\tau}}, d_2 = d_1 - \sigma\sqrt{\tau}$.

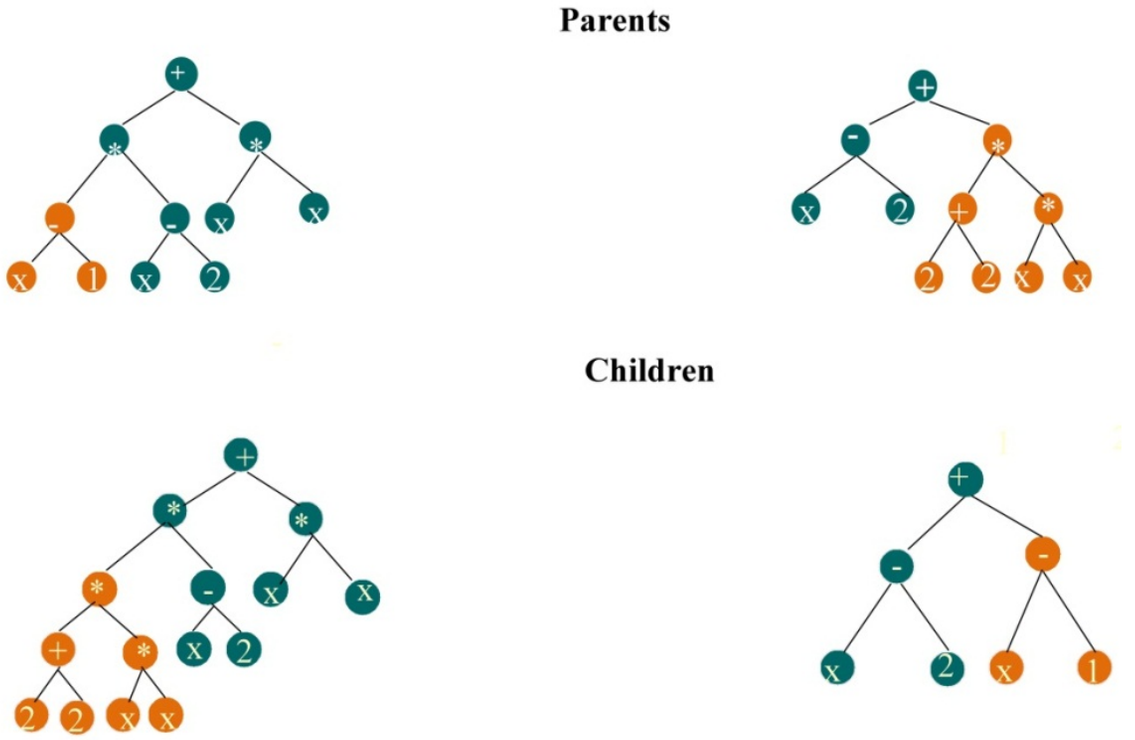


Figure 4. Example of subtree crossover

The mutation operator randomly changes a tree by randomly altering nodes or sub-trees to create a new offspring. Often multiple types of mutation are beneficially used simultaneously [37,38]. In this paper, three mutation operators are used simultaneously, they are described below:

Branch (or subtree) mutation operator randomly selects an internal node in the tree, and then it replaces the subtree rooted at that node with a new randomly-generated subtree [3].

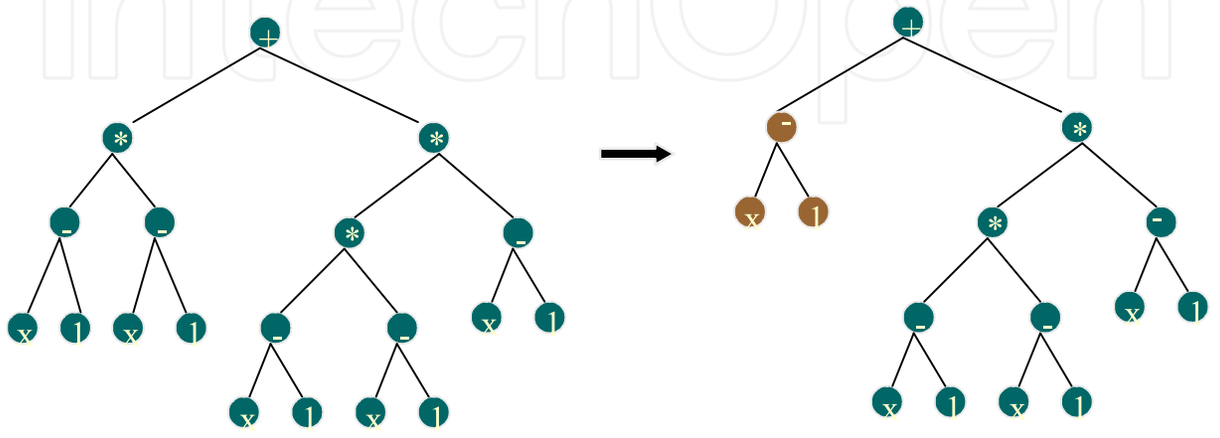


Figure 5. Example of subtree mutation

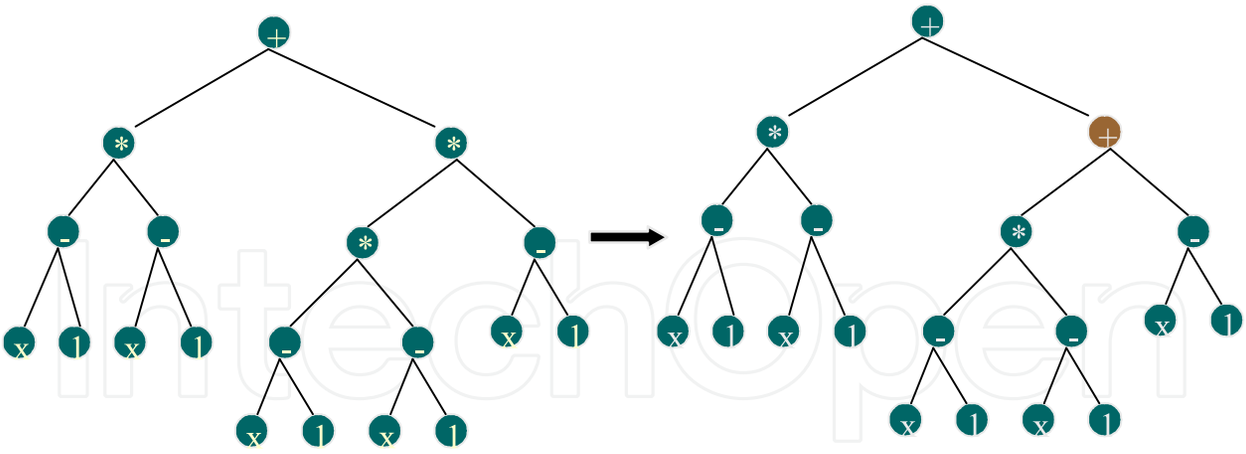


Figure 6. Example of point mutation

Point mutation operator consists of replacing a single node in a tree with another randomly-generated node of the same arity [39].

Expansion mutation operator randomly selects a terminal node in the tree, and then replaces it with a new randomly-generated subtree.

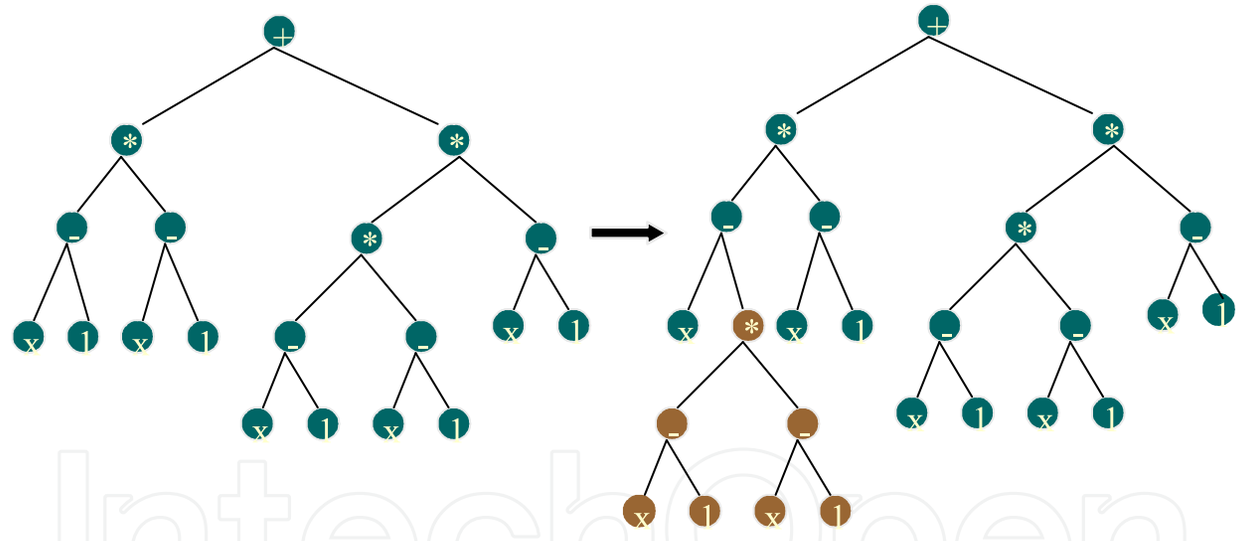


Figure 7. Example of expansion mutation

As indicated in Table 2, Branch mutation is applied with a rate of 20%; Point and Expansion mutations are applied with a rate of 10%, respectively.

Replacement: The method of replacing parents for the next generation is comma replacement strategy [40], which selects the best offspring to replace the parents. It assumes that offspring size is higher than parents' size. If μ is the population size and λ is the number of the new individuals (which can be larger than μ), the population is constructed using the best μ out of the λ new individuals.

Termination criterion: The stopping criterion is the maximum number of generations. It is fixed at 400 and 1000 for static and dynamic training- subset selection, respectively. In the dynamic

training- subset selection approach, the maximum number of generations is increased to allow the genetic programming to train on the maximum of samples simultaneously. The number of generations to change sample varied between 20 and 100 generations.

The implementation of genetic programming involves a series of trial and error experiments to determine the optimal set of genetic parameters which is listed in Table 2. By varying genetic parameters, each program is run ten times with ten different random seeds. The choice of the best genetic program is made according to the mean and median of Mean Squared Errors (MSE) for training and testing sets.

Population size:	100
Offspring size:	200
Maximum number of generations for static method:	400
Maximum number of generations for dynamic method:	1000
Generations' number to change sample	20-100
Maximum depth of new individual:	6
Maximum depth of the tree:	17
Tournament size:	4
Crossover probability:	60%
Mutation probability:	40%
Branch mutation:	20%
Point mutation:	10%
Expansion mutation:	10%

Table 2. Summary of genetic programming parameters

3.2.2. Dynamic training-subset selection method

As data are divided in several sub-samples, the genetic programming is trained, first, independently on each sub-sample relative to each data division scheme (algorithm 1). This approach is called static training-subset selection method [8]. Second, the genetic programming is trained simultaneously on the entire data sub-samples relative to each data division scheme, rather than just a single subset by changing the training sub-sample during the run process. This approach is called dynamic training-subset selection method. The main goal of this method is to make genetic programming adaptive to all training samples and able to generate general models and solutions that are more robust across different learning data samples. In the context of evolutionary algorithms, there are at least three approaches for defining the frequency of resampling [41]. The first approach called “individual-wise” consists of extracting a new sample of data instances from the training set for each individual of the population. As a result, different individuals will probably be evaluated on different data samples, which cast some doubts on the fairness of the selection procedure of the evolutionary algorithm. The second approach called “run-wise” consists of extracting a single fixed sample of data instances from the training set used to evaluate the fitness of all individuals throughout the evolutionary run, which will probably reduce significantly the robustness and predictive accuracy of the evolutionary algorithm. The third approach called

"generation-wise" consists of extracting a single fixed sample of data instances from the training set at each generation, and all individuals of that generation will have their fitness evaluated on that data sample. This method avoids the disadvantages of the two previous approaches, and as such seems more effective. In particular, an individual will only survive for several generations if it has a good predictive accuracy across different data samples. The dynamic approach proposed in this study differs from the three previous approaches as it doesn't extract a fixed sample of data instances from the training set, but selects it from the whole sub-samples data which are already built up and use it to evaluate the fitness of all individuals when the generations' number to change sample is reached. In this paper, we proposed four dynamic training-subset selection methods: *Random Subset Selection* method (*RSS*), *Sequential Subset Selection* method (*SSS*), *Adaptive-Sequential Subset Selection* method (ASSS) and *Adaptive-Random Subset Selection* method (*ARSS*). The *RSS* and *SSS* allow the genetic programming to learn on all training samples in turn (*SSS*) or randomly (*RSS*). However, with these methods, there is no certainty that genetic programming will focus on the samples which are difficult to learn. Then, the *ASSS* and the *ARSS,* which are variants of *the adaptive subset selection (ASS),* are introduced to focus the genetic programming's attention onto the difficult samples i.e. having the greatest MSE and then to improve the learning algorithm.

Dynamic subset selection is easily added to the basic GP algorithm with no additional computational cost according to the static subset selection.

Let S be the set of training samples $S_{i\ (i=1\ldots k)}$, where k is the total number of samples. A selection probability P (S_i) is allocated to each sample S_i from S. The training sample S_i is changed each g generations (g is the number of generations to change sample) according to this selection probability and the dynamic training-subset selection method used. Once a new training sample is selected, the best individuals are used as population for the next training samples. This procedure is repeated until the maximum number of generations is reached. This permits genetic programming to adapt its generating process to changing data in response to feedback from the fitness function which is the mean squared error computed as in static approach. By the end of the evolution, only individuals with the desirable characteristics that are well adapted to the environmental changes will survive.

a. Random training-Subset Selection method (RSS):

It selects randomly the training samples with replacement. At each g generations, all the samples from S have the same probability to be selected as the current training sample: $P(S_i) = 1/k$, $1 \leq i \leq k$. This method differs from that proposed by Gathercole and Ross [42] as random selection concerns training samples which are already constructed according to data division scheme, rather than data instances.

As selection of training samples is random, the performance of the current population changes with the training sample used for evolving the genetic program. Figure 8 illustrates an example of the best fitness (MSE) curve along evolution using RSS method. With the sample change, the MSE may increase, but it is improved during the following generations, the time that the population adapts itself to the new environment.

Figure 8 shows that some training samples could be duplicated, but some others could be eliminated.

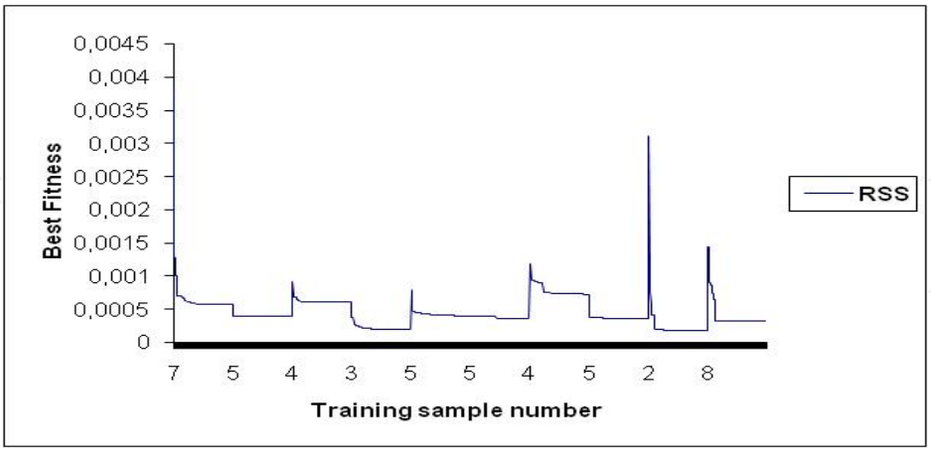


Figure 8. Example of fitness curve of the best individuals generated by genetic programming using RSS method for time series samples

b. Sequential training-Subset Selection method (SSS)

It selects all the training samples in the order. If, at generation g-1, the current training sample is S_i, then at generation g: P (S_j) = 1, with j= i+1 if i<k, or j=1 if i=k.

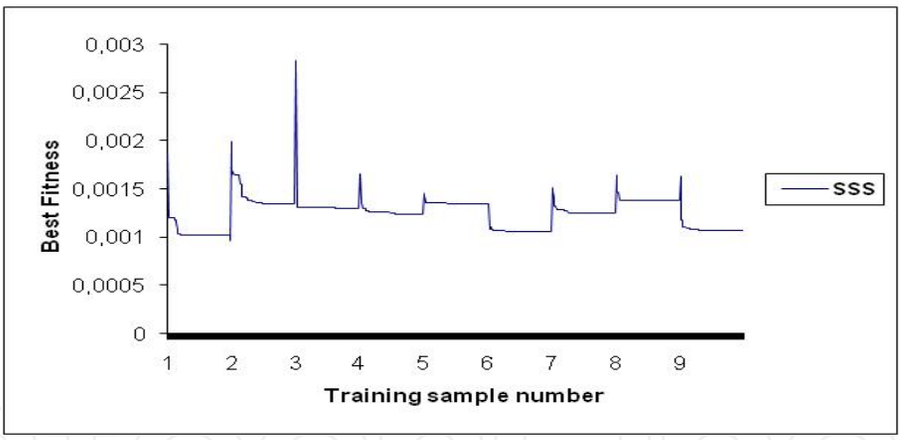


Figure 9. Example of curve fitness of the best individuals generated by genetic programming using SSS method for moneyness-time to maturity classes

As illustrated in Figure 9, all the learning subsets are used during the evolution in an iterative way.

c. Adaptive training-Subset Selection method (ASS):

Instead of selecting a training subset data in a random or sequential way, one can use an adaptive approach to dynamically select difficult training subsets data which are frequently misclassified. This approach is inspired from the dynamic subset selection method proposed by Gathercole and Ross [42] which is based on the idea of dynamically selecting instances, not training samples, which are difficult and/or have not been selected for several

generations. Selection is made according to a weight computed proportionally to the sample's average fitness. Each g generations, the weights are updated as follows:

$$W(S_i) = \frac{\sum_{t=1}^{g}\sum_{j=1}^{M} f(X_j)}{M * g} \tag{3}$$

Where, M is the size of S_i ($X_j \in S_i$), g is the number of generations to change sample, and $f(X_j)$ is the MSE of the individual X_j .

At each g generations, training samples are re-ordered, so that the most difficult training samples, which have higher weights, will be moved to the beginning of the ordered training list, and the easiest training samples, which have smaller weights, will be moved to the end of the ordered training list.

1. Adaptive-Sequential training-Subset Selection method (ASSS):

It uses the following procedure (step 1 to step 3):

Step 1. Let the first generation t be set to 0. Each training sample is assigned an equal weight, i.e., $W(S_i) = 1$ for $1 \le i \le k$.

Step 2. The probability $P(S_i)$ that a training sample S_i is selected to be included in the training set and evolve genetic programming is determined using the Roulette wheel selection scheme.

$$P(S_i) = \frac{w(S_i)}{\sum w(S_i)}$$

Where, the summation is over all training samples.

Moreover, the probability $P(S_i)$ is positively related to the fitness of the parse tree generated relative to the corresponding training sample.

$$P(S_i) = \frac{f(S_i)}{\sum f(S_i)}$$

Where, $f(S_i)$ is the average fitness of individuals relative to the training sample.

Compute a fitness function which is the mean squared error for each individual in the training sample and then the average fitness. Update the weights: $W(S_i) = \frac{\sum_{t=1}^{g}\sum_{j=1}^{M} f(X_j)}{M * g}$

Step 3. t=t+g. If t<T (T is the total number of generations), then go to step 2.

As illustrated in Figure 10, selection of training samples is made in the order for the first t generations using the SSS method. Some training samples could be duplicated to improve

the genetic programming learning. Later, samples are selected for the next run according to the adaptive approach based on the re-ordering procedure.

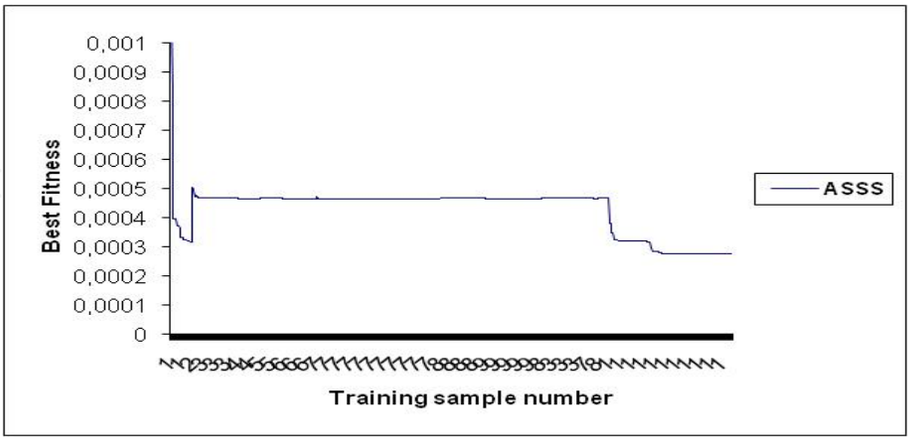


Figure 10. Example of curve fitness of the best individuals generated by genetic programming using ASSS method for time series samples

2. Adaptive-Random training-Subset Selection method (ARSS):

The ARSS method uses the same procedure as the ASSS method, except that the initial weights are generated randomly in the start of running, rather than initialized with a constant: For t=0, $W(S_i) = \tilde{P}_i, \tilde{P}_i \in [0,1], 1 \le i \le k$. Then, for the few first generations, samples are selected using *RSS* method. After, the selection of samples is made using the adaptive approach based on the re-ordering procedure.

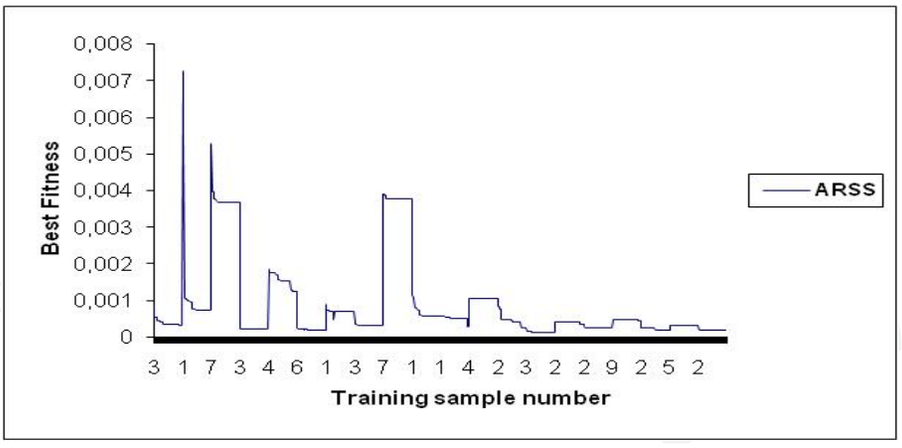


Figure 11. Example of curve fitness of the best individuals generated by genetic programming using ARSS method for moneyness-time to maturity classes

3.2.2. Training and test samples

Different forecasting genetic programming volatility models are estimated from the training set and judged upon their performance on the test set. Table 3 summarizes the training and test data samples used for static and dynamic training-subset selection methods, respectively.

In static training-subset selection approach, first, the genetic program is trained separately on each of the first nine TS sub-samples (S_1,…, S_9) using ten different seeds and is tested on the subset data from the immediately following date (S_2,…, S_{10}). Second, using the same genetic parameters and random seeds applied for TS data, the genetic programming is trained separately on each of the first nine MTM sub-classes (C_1^L,…, C_9^L) and is tested on the second nine MTM sub-classes (C_1^T,…, C_9^T).

Subset Selection	**Learning data sample**	**Test data sample**
Static Subset Selection	$S_i \in$ TS samples (S_1, …, S_9) *(1 subset for a run)*	The successive TS sample S_j, j=i+1
	$C_i^L \in$ MTM training samples (C_1^L, …, C_9^L) *(1 subset for a run)*	The corresponding MTM test samples C_i^T
Dynamic Subset Selection (RSS/SSS/ASSS/ ARSS)	TS samples S_1, …, S_9 *(9 subsets for a run)*	The last subset in TS samples set (S_{10})
	MTM samples C_1^L, …, C_9^L *(9 subsets for a run)*	The nine MTM test samples ($C_1^T + C_2^T$ …+ C_9^T)
	TS samples + MTM samples (S_1, …, S_9 ; C_1^L, …, C_9^L) (*18 subsets for a run)*	The last TS sample with the nine MTM test samples ($S_{10} + C_1^T + C_2^T$ …+ C_9^T)

Table 3. Definition of training and test data samples for static and dynamic training-subset selection methods

In dynamic training-subset selection approach, first, the genetic program is trained on the first nine TS sub-samples simultaneously (S_1,…, S_9) using ten different seeds and it is tested only on the tenth sub-sample data (S_{10}). Second, the genetic programming is trained on the first nine MTM sub-classes simultaneously (C_1^L,…, C_9^L) and it is tested on the second nine MTM sub-classes regrouped in one test sample data ($C_1^T + C_2^T$ …+ C_9^T). Third, the genetic programming is trained on both the nine TS sub-samples and the nine MTM sub-classes simultaneously (S_1, …, S_9 ; C_1^L, …, C_9^L) and it is tested on one test sample data composed of the TS and MTM test data ($S_{10} + C_1^T + C_2^T$ …+ C_9^T).

Based on the training and test MSE, the best generated genetic programming volatility models relative to static and dynamic training-subset selection methods respectively are selected. These models are then compared with each other according to the MSE total and the best ones are used to implement the dynamic hedging strategies as described in the following section.

3.3. Dynamic hedging

To assess the accuracy of selected generated genetic programming volatility models in hedging with respect to Black-Scholes model, three dynamic hedging strategies are employed, notably, delta-neutral, delta-gamma neutral and delta-vega neutral strategies.

For delta hedging, at date zero, a delta hedge portfolio consisting of a short position in one call (or put) option and a long (short) position in the underlying index is formed. At any time t, the value of the delta hedge portfolio $\mathrm{P}(t)$ is given by:

$$P(t) = V(t) + \Delta_V(t)S(t) + \beta(t) \tag{4}$$

Where, $P(t)$, $V(t)$, $S(t)$, $\Delta_V(t)$ and $\beta(t)$ denote the values of the portfolio, hedging option (call or put), underlying, delta hedge factor and bond (money market account) respectively.

The portfolio is assumed self-financed, so the initial value of the hedge portfolio at the beginning of the hedge horizon is zero:

$$\mathrm{P}(0) = V(0) + \Delta_V(0)S(0) + \beta(0) = 0 \tag{5}$$

$$\Rightarrow \beta(0) = -(V(0) + S(0)\Delta_V(0)) \tag{6}$$

A dynamic trading strategy is performed in underlying and bond to hedge the option during the hedge horizon. The portfolio rebalancing takes place at intervals of length δt during the hedge horizon $\left[0, \tau\right]$, $o < \tau \le T$, where T is the maturity of the option. At each rebalancing time t_i, the hedge factor $\Delta_v(t_i)$ is recomputed and the money market account is adjusted:

$$\beta(t_i) = e^{r\delta t}\beta(t_{i-1}) - S(t_i)(\Delta_V(t_i) - \Delta_V(t_{i-1})) \tag{7}$$

The delta hedge error is defined as the absolute value of the delta hedge portfolio at the end of the hedge horizon of the option, $\left|P(\tau)\right|$.

For delta-gamma hedging, a new position in a traded option is required. Then, the delta-gamma hedge portfolio is formed with:

$$P(t) = V(t) + x(t)S(t) + y(t)V_1(t) + B(t) \tag{8}$$

Where, $V_1(t)$ is the value of an additional option which depends on the same underlying, with the same maturity but different strike price than the hedging option $V(t)$. $x(t)$ and $y(t)$ are the proportions of the underlying and the additional option respectively. They are chosen such that the portfolio $\mathrm{P}(t)$ is both delta and gamma neutral:

$$\begin{cases} \textit{Delta}\text{ neutral:}\Delta_V(t) + x(t) + y(t)\Delta_{V1}(t) = 0 \\ \textit{Gamma}\text{ neutral:}\Gamma_V(t) + y(t)\Gamma_{V_1}(t) = 0 \end{cases} \tag{9}$$

$$\Rightarrow \begin{cases} y(t) = \dfrac{-\Gamma_V(t)}{\Gamma_{V_1}(t)} \\ x(t) = -\Delta_V(t) - y(t)\Delta_{V1}(t) \end{cases} \tag{10}$$

Where, the values of $\Delta_V(t)$ and $\Gamma_V(t)$ are the delta and gamma factors for the option $V(t)$; the values $\Delta_{V_1}(t)$ and $\Gamma_{V_1}(t)$ are the delta and gamma factors for the option $V_1(t)$.

At the beginning of the hedge horizon, the value of the hedge portfolio is zero:

$$P(0) = V(0) + x(0)S(0) + y(0)V_1(0) + B(0) = 0 \tag{11}$$

$$\Rightarrow B(0) = -(V(0) + x(0)S(0) + y(0)V_1(0)) \tag{12}$$

At each rebalancing time t_i, both delta and gamma hedge factors are recomputed and the money market account is adjusted:

$$B(t_i) = e^{r\delta t}B(t_{i-1}) - (x(t_i) - x(t_{i-1}))S(t_i) - (y(t_i) - y(t_{i-1}))V_1(t_i) \tag{13}$$

The delta-gamma hedge error is defined as the absolute value of the delta-gamma hedge portfolio at the end of the hedge horizon of the option, $\left|P(\tau)\right|$.

For delta-vega hedging, a new position in a traded option is required as in the delta-gamma hedging. The proportions of the underlying $x(t)$ and the additional option $y(t)$ are chosen such that the portfolio $\mathrm{P}(t)$ is both delta and vega neutral:

$$\begin{cases} Delta \text{ neutral}: \Delta_V(t) + x(t) + y(t)\Delta_{V1}(t) = 0 \\ V\text{ega neutral}: \vartheta_V(t) + y(t)\vartheta_{V_1}(t) = 0 \end{cases} \tag{14}$$

$$\Rightarrow \begin{cases} y(t) = \dfrac{-\vartheta_V(t)}{\vartheta_{V_1}(t)} \\ x(t) = -\Delta_V(t) - y(t)\Delta_{V1}(t) \end{cases} \tag{15}$$

Where, $\vartheta_V(t)$ and $\vartheta_{V_1}(t)$ are the vega factors for the options $V(t)$ and $V_1(t)$ respectively.

As in delta-gamma hedging, at each rebalancing time t_i, both delta and vega hedge factors are recomputed and the money market account is adjusted. The delta-vega hedge error is defined as the absolute value of the delta-vega hedge portfolio at the end of the hedge horizon of the option, $\left|P(\tau)\right|$.

35 option contracts are used as hedging options and 35 other contracts which depend on the same underlying, with the same maturity but different strike prices are used as additional options. Contracts used to implement the hedging strategies are divided according to moneyness and time to maturity criteria, which produces nine classes.

The delta, gamma and vega hedge factors are computed using the Black-Scholes formula by taking the derivative of the option value with respect to index price, the derivative of delta with respect to index price and the derivative of the option value with respect to volatility respectively. For the genetic programming models, the hedge ratios are computed using the same formulas replacing the Black-Scholes implied volatilities with the generated genetic programming volatilities. Two rebalancing frequencies are considered: 1-day and 7 days revision.

The average hedging error is used as performance measure. For a particular moneyness-time to maturity class, the tracking error is given by:

$$\begin{cases} \varepsilon_M = \dfrac{\sum_{i=1}^{n} \varepsilon_i(\tau)}{n} \\ \varepsilon_i = e^{-rT} \times \dfrac{\left|P_i(\tau)\right|}{N \times V(0)} \end{cases} \tag{16}$$

Where, n is the number of options corresponding to a particular moneyness-time to maturity class and $\varepsilon_i(\tau)$ is the present value of the absolute hedge error of the portfolio $\left|P(\tau)\right|$ over the observation path N (as a function of rebalancing frequency), divided by the initial option price $V(0)$.

4. Result analysis and empirical findings

4.1. Selection of the best genetic programming-implied volatility forecasting models

Selection of the best generated genetic programming volatility model, relative to each training set, for TS, MTM, and both TS and MTM classifications, is made according to the training and test MSE. For static training-subset selection method, nine generated genetic programming volatility models are selected for TS (M1S1…M9S9) and similarly nine generated genetic programming volatility models are selected for MTM classification (M1C1…M9C9). The performance of these models is compared according to the MSE Total, computed using the same formula as the basic MSE for the enlarged data sample.

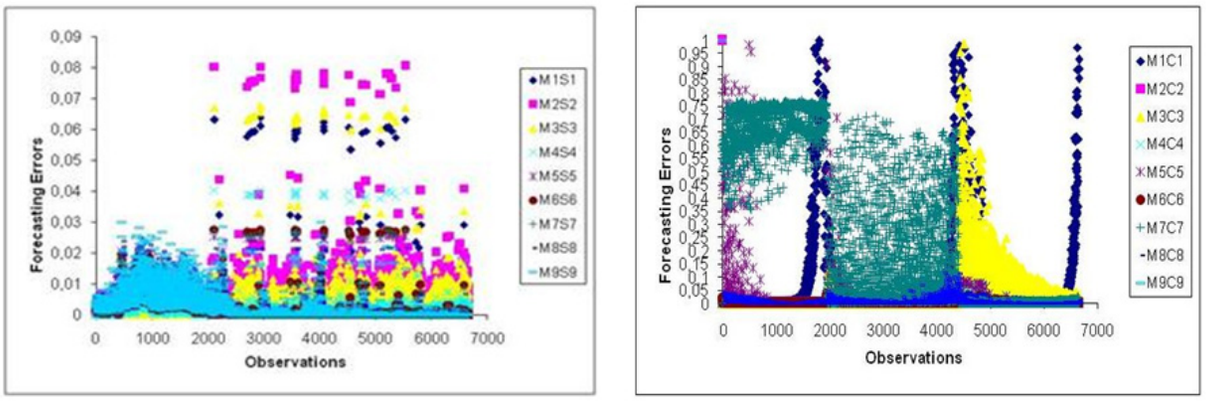
Table 4 reports the MSE total and the standard deviation (in parentheses) of the generated genetic programming volatility models, using static training-subset selection method, relative to the TS samples and the MTM classes.

TS Models	MSE Total	MTM Models	MSE Total
M1S1	0,002723 (0,004278)	M1C1	2,566 (20,606)
M2S2	0,005068 (0,006213)	M2C2	0,006921 (0,032209)
M3S3	0,003382 (0,004993)	M3C3	0,030349 (0,076196)
M4S4	**0,001444 (0,002727)**	**M4C4**	**0,001710 (0,004624)**
M5S5	0,002012 (0,003502)	M5C5	1,427142 (33,365115)
M6S6	0,001996 (0,003443)	**M6C6**	**0,002357 (0,004096)**
M7S7	0,001901 (0,003317)	M7C7	0,261867 (0,303256)
M8S8	0,002454 (0,004005)	M8C8	0,004318 (0,008479)
M9S9	0,002419 (0,004095)	M9C9	0,002940 (0,010490)

Table 4. Performance of the generated genetic programming volatility models using static training-subset selection method, according to MSE total for the TS samples and the MTM classes

Table 4 shows that, the generated genetic programming volatility models M4S4, M4C4 and M6C6 present the smallest MSE on the enlarged sample for TS and MTM samples respectively. Comparison between these models reveals that the TS model M4S4 seems to be more performing than MTM models M4C4 and M6C6 for the enlarged sample. Furthermore, results show that the performance of TS models is more uniform than that of MTM models. MTM models are not able to fit appropriately the entire data sample as well as the TS models as they have large Total MSE. Indeed, the MSE total exceed 1 with some MTM classes, however it does not reach 0.006 for all TS samples. Figure 12 describes the evolution's pattern of the squared errors given by TS models and MTM models for all observations in the enlarged data sample. Some extreme MSE values for MTM data are not shown in this figure.

It appears throughout Figure 12 that, the TS models are adaptive not only to training samples, but also to the enlarged sample. In contrast, the MTM models such as M1C1 are adaptive to training classes, but not all to the enlarged sample. A first plausible explanation of these unsatisfied results is an insufficient search intensity inducing difficulty to obtain general model suitable for the entire benchmark input data. To enhance exploration intensity during learning and thus improve the genetic programming performance, we introduced to the evolution procedure the dynamic subset selection, which aims to obtain a general model that can be adaptive to both TS and MTM classes simultaneously.



(a) MSE pattern for TS samples

(b) MSE pattern for MTM classes

Figure 12. Evolution of the squared errors for total sample of the best generated GP volatility models, using static training-subset selection method, relative to TS samples(a) and MTM classes (b).

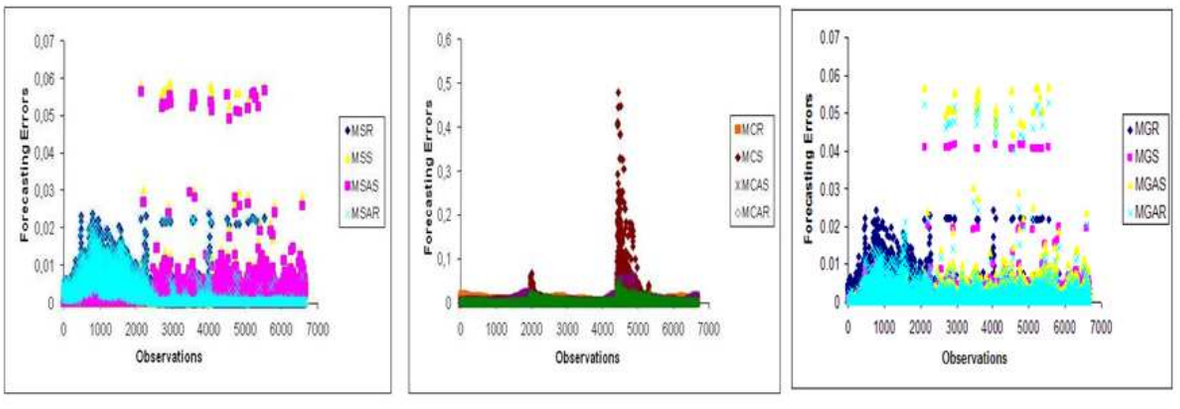
For dynamic training-subset selection methods (RSS, SSS, ASSS and ARSS), four generated genetic programming volatility models are selected for TS classification (MSR, MSS, MSAS and MSAR). Similarly, four generated genetic programming volatility models are selected for MTM classification (MCR, MCS, MCAS and MCAR) and four generated genetic programming volatility models are selected for global classification, both TS and MTM classes (MGR, MGS, MGAS and MGAR). Table 5 reports the best generated genetic programming volatility models, using dynamic training-subset selection, relative to TS samples, MTM classes and both TS and MTM data.

TS Models	MSE Total	MTM Models	MSE Total	Global Models	MSE Total
MSR	0.002367 (0.003934)	MCR	0.002427 (0.003777)	MGR	0.002034 (0.003501)
MSS	**0.002076 (0.004044)**	MCS	0.007315 (0.025811)	MGS	0.002492 (0.003013)
MSAS	0.002594 (0.003796)	MCAS	0.002831 (0.004662)	MGAS	0.001999 (0.003587)
MSAR	0.002232 (0.003782)	**MCAR**	**0.001424 (0.003527)**	**MGAR**	**0.001599 (0.003590)**

Table 5. Performance of the generated genetic programming volatility models, using dynamic training-subset selection method, according to MSE total for the TS samples, the MTM classes and both TS and MTM samples

Based on the MSE total as performance criterion, the generated genetic programming volatility models MSS, MCAR and MGAR are selected. They seem to be more accurate in forecasting implied volatility than the other models because they have the smallest MSE in enlarged sample. However, the MTM model MCAR and the global model MGAR outperform the TS model MSS. Figure 13 describes the evolution's pattern of the squared errors for these generated volatility models.

Figure 13 shows that almost all models relative to each data's group are performing on the enlarged sample and present forecasting errors which are small and very closed. Forecasting errors are higher for the MTM classes than for the TS samples and both TS and MTM samples. Comparison between models generated using static training-subset selection method (Figure 12) and dynamic training-subset selection methods (Figure 13) respectively, reveals that the amplitude of forecasting errors relative to TS and MTM classes respectively is lower for the models generated using dynamic training-subset selection methods than for the models generated using static training-subset selection method. Actually, the quality of the generated genetic programming forecasting models has been improved with the dynamic training, in particular for MTM classes.



(a) MSE pattern for TS samples

(b) MSE pattern for MTM classes

(c) MSE pattern for TS+MTM

Figure 13. Evolution of the squared errors for total sample of the best generated GP volatility models, using dynamic training-subset selection methods, relative to TS samples (a), MTM classes (b) and both TS and MTM samples (c).

The best generated genetic programming volatility models selected, relative to dynamic training-subset selection method, are compared to the best generated genetic programming volatility model, relative to static training-subset selection method. Results are reported in Table 6.

Models	**MSE total**
M4S4	0,001444 (0,002727)
MCAR	0.001424 (0.003527)
MGAR	**0.001599 (0.003590)**

Table 6. Comparison between best models generated by static and dynamic selection methods for call options

Comparison between models reveals that the best models generated respectively by static (M4S4) and dynamic selection methods (MCAR and MGAR) present total MSE small and very close. While the generated genetic programming volatility models M4S4 and MCAR have total MSE smaller than the MGAR model, the latest seems to be more accurate in forecasting implied volatility than the other models. This can be explained by the fact that, on one hand, the difference between forecasting errors is small, and on the other hand, the MGAR model is more general than MCAR and M4S4 models because it is adaptive to all TS and MTM classes simultaneously. In fact, the MGAR model, generated using ARSS method, is trained on all TS and MTM classes simultaneously. Whereas, the MCAR model, generated using ARSS method, is trained only on MTM classes simultaneously; and the M4S4 model, generated using static training-subset selection method, is trained separately on each subset of TS.

As the adaptive-random training subset selection method is considered the best one to generate implied volatility model for call options, it is applied to put options. The decoding of volatility forecasting formulas generated for call and put options as well as their forecasting errors are reported in Table 7.

A detailed examination of the formulas in Table 7 shows that the implied volatilities generated by genetic programming are function of all the inputs used, namely the option price divided by strike price ($\frac{C}{K}$ for calls and $\frac{P}{K}$ for puts), the index price divided by strike price $\frac{S}{K}$ and time to maturity τ. The implied volatilities generated for calls and puts cannot be negative since they are computed using the square root and the normal cumulative distribution functions as the root nodes. Furthermore, the performance of models is uniform as they present near MSE on the enlarged sample.

4.2. Dynamic hedging results:

The performance of the best genetic programming forecasting models is compared to the Black-Scholes model in delta, gamma and vega hedging strategies. Table 8 reports the

average hedging errors for call options using Black-Scholes (BS) and genetic programming (GP) models, at the 1-day and 7-days rebalancing frequencies. Values in bold correspond to the GP hedging errors which are less than the BS ones.

Option	**LISP Expression**	**Formula**	**MSE Total**
Call	sqrt((X0/(multiply(X,((multiply(X1,plus(X1,X2))*X1)*X1))*X1)))	$\sigma_{GP} = \sqrt{\frac{\frac{C}{K}}{\left(\frac{S}{K}\right)^6 + \left(\frac{S}{K}\right)^5 * \tau}}$	0.001599
Put	ncdf (sin ((cos (sin (minus (minus (-(cos (sin(X2))), ln(X0)), ln(X0))))-exp(X1))))	$\sigma_{GP} = \Phi\left(\sin\left(\cos\left(\sin\begin{pmatrix}-\cos(\sin(\tau)) \\ -2*\ln\left(\frac{P}{K}\right)\end{pmatrix}\right)\right) - \exp\left(\frac{S}{K}\right)\right)$	0.001539

Table 7. Performance of the best generated genetic programming volatility models for call and put options and their decoding formulas $\left(X_0 = \frac{C}{K} or \frac{P}{K}, X_1 = \frac{S}{K}, X_2 = \tau\right)$

Results in Table 8 show that the delta hedging performance improves for out-of-the money call options at longer maturities, for at-the-money call options at medium maturities and for in-the money call options at shorter maturities, regardless of the model used at daily hedge revision frequency. The best delta hedging performance is achieved using in-the-money short term call options for all MTM classes, regardless of the option model used.

The delta-gamma hedging performance improves for all moneyness classes of call options at longer maturities, regardless of the model used at daily hedge frequency (except in-the-money call options using the genetic programming model). The best delta-gamma hedging performance is achieved, for BS model, using at-the-money long term call options for all MTM classes. However, the best delta-gamma hedging performance is achieved, for genetic programming model, using in-the-money short term call options for all MTM classes.

The delta-vega hedging performance improves for out-of-the money and in-the-money call options at longer maturities and for at-the-money call options at shorter maturities, regarding BS model at daily hedge revision frequency. However, the delta-vega hedging performance improves for out-of-the money call options at shorter maturities, for at-the-money call options at medium maturities and for in-the money call options at longer maturities, regarding genetic programming model at daily hedge revision frequency. The best delta-vega hedging performance is achieved, for BS model, using out-of-the-money long term call options for all moneyness and time to maturity classes. However, the best delta-gamma hedging performance is achieved, for genetic programming model, using at-the-money medium term call options for all MTM classes.

The percentage of cases where the hedging error of the genetic programming model is less than the BS hedging error is around 59%. In particular, the performance of genetic

programming model is better than the BS model on in-the-money call options class. Further, the total of hedging errors relative to genetic programming model is about 21 percent slightly lower than 19 percent relative to BS model. Table 9 displays the average hedge errors for put options using BS and genetic programming models, at the 1-day and 7-days rebalancing frequencies. Values in bold correspond to the genetic programming hedging errors which are less than the BS ones.

			Rebalancing Frequency					
			1-day			**7- days**		
S/K	**Hedging strategy**	**Model**	**<60**	**60-180**	**>=180**	**<60**	**60-180**	**>=180**
<0.98	**Delta hedging**	BS	0,013119	0,001279	0,000678	0,057546	0,010187	0,005607
		GP	**0,009669**	**0,001081**	**0,000662**	**0,053777**	**0,009585**	**0,005594**
	Gamma hedging	BS	0,000596	0,000732	0,000061	0,003026	0,007357	0,000429
		GP	0,000892	0,002040	0,000075	0,003855	**0,001359**	**0,000153**
	Vega hedging	BS	0,000575	0,000050	0,000039	0,000525	0,000226	0,000099
		GP	**0,000473**	0,002035	0,004518	0,000617	0,004642	0,040071
0.98-1.03	**Delta hedging**	BS	0,002508	0,000717	0,000730	0,019623	0,005416	0,002283
		GP	**0,002506**	**0,0007**	0,001725	0,020	**0,0054**	**0,0022**
	Gamma hedging	BS	0,000069	0,000018	0,000006	0,000329	0,000169	0,000027
		GP	0,000377	0,000040	0,000029	0,000727	**0,000155**	0,000059
	Vega hedging	BS	0,000066	0,000373	0,003294	0,000527	0,023500	0,031375
		GP	0,000281	**0,000013**	**0,000207**	0,001102	**0,000147**	**0,000134**
>=1.03	**Delta hedging**	BS	0,000185	0,000906	0,001004	0,001602	0,006340	0,006401
		GP	**0,000184**	**0,000905**	**0,001**	**0,000840**	**0,005789**	**0,0064**
	Gamma hedging	BS	0,000323	0,000047	0,000028	0,001546	0,000386	0,000157
		GP	**0,000028**	0,000057	0,000036	**0,000227**	0,000429	0,000175
	Vega hedging	BS	0,000362	0,000060	0,000052	0,001757	0,002015	0,000247
		GP	**0,000067**	**0,000057**	**0,00005**	**0,000831**	**0,000864**	**0,000186**

Table 8. Average hedge errors of dynamic hedging strategies relative to BS and GP models for call options

Results in Table 9 show that the delta-gamma hedging performance improves for all moneyness classes of put options (except in-the-money put options) at longer maturities, regarding BS model at daily hedge frequency. However, the delta-gamma hedging performance improves for in-the money put options and at-the-money put options at medium maturities and for out-of-the money put options at longer maturities, regarding genetic programming model at daily hedge revision frequency. The best delta-gamma hedging performance is achieved, for BS model, using at-the-money long term put options

for all MTM classes. However, the best delta-gamma hedging performance is achieved, for genetic programming model, using out-of-the-money long term put options for all MTM classes.

			Rebalancing Frequency					
			1-day			7- days		
S/K	**Hedging strategy**	**Model**	**<60**	**60-180**	**>=180**	**<60**	**60-180**	**>=180**
<0.98	**Delta hedging**	BS	0,007259	0,002212	0,001189	0,015453	0,013715	0,007740
		GP	0,064397	0,002270	0,001256	0,016872	0,013933	0,007815
	Gamma hedging	BS	0,000107	0,000043	0,000705	0,000383	0,000253	0,013169
		GP	0,000177	0,000351	**0,000676**	0,000990	0,000324	**0,009201**
	Vega hedging	BS	0,000051	0,000715	0,000612	0,000174	0,002995	0,008527
		GP	0,002800	**0,000345**	0,000625	0,018351	**0,000184**	0,008979
0.98-1.03	**Delta hedging**	BS	0,007331	0,002267	0,001196	0,170619	0,009875	0,004265
		GP	**0,0073**	**0,002219**	**0,001185**	**0,170316**	**0,009715**	**0,004260**
	Gamma hedging	BS	0,003750	0,000049	0,000027	0,032725	0,000119	0,000119
		GP	**0,003491**	**0,000031**	**0,000024**	**0,029792**	**0,000113**	**0,000103**
	Vega hedging	BS	0,035183	0,000052	0,000044	0,037082	0,000329	0,000043
		GP	**0,004343**	**0,000038**	**0,000043**	**0,037045**	**0,000190**	**0,000041**
>=1.03	**Delta hedging**	BS	0,007680	0,004469	0,000555	0,037186	0,017322	0,011739
		GP	**0,006641**	**0,004404**	**0,0005**	**0,037184**	**0,017076**	**0,011733**
	Gamma hedging	BS	0,000262	0,000204	0,000079	0,001196	0,001319	0,000369
		GP	0,000548	0,000287	0,000166	0,002034	0,001323	0,001059
	Vega hedging	BS	0,000232	0,000108	0,000025	0,000488	0,000644	0,000270
		GP	0,000312	**0,000080**	**0,00002**	0,001047	0,001186	**0,000244**

Table 9. Average hedge errors of dynamic hedging strategies relative to BS and GP models for put options

The delta-vega hedging performance improves for BS using at-the-money and out-of-the-money put options at longer maturities and in-the-money put options at shorter maturities, at daily hedge revision frequency. However, the delta-vega hedging performance improves for all moneyness classes of put options (except in-the-money put options) at longer maturities, regarding genetic programming model at daily hedge frequency. The best delta-vega hedging performance is achieved, for BS model, using out-of-the-money long term put options for all MTM classes. However, the best delta-vega hedging performance is achieved, for genetic programming model, using at-the-money long term put options for all MTM classes.

The percentage of cases where the hedging error of the genetic programming model is less than the BS hedging error is around 57%. In particular, the performance of genetic

programming model is better than the BS model on at-the-money put options class. But, the total of hedging errors relative to genetic programming model is about 50 percent slightly higher than 46 percent relative to BS model.

In summary, the genetic programming model is more accurate in all hedging strategies than the BS model, for in-the-money call options and at-the-money put options. The performance of genetic programming is pronounced essentially in terms of delta hedging for call and put options. The percentage of cases where the delta hedging error of the genetic programming model is less than the BS delta hedging error is 100% for out-of-the money and in-the-money call options as well as for at-the-money and out-of-the-money put options. The percentage of cases where the delta-vega hedging error of the genetic programming model is less than the BS delta-vega hedging error is 100% for in-the-money call options as well as for at-the-money put options. The percentage of cases where the delta-gamma hedging error of the genetic programming model is less than the BS delta-gamma hedging error is 100% for at-the-money put options.

Furthermore, results exhibit that as the rebalancing frequency changes from 1-day to 7-days revision, as the hedging errors increase and vice versa. The option value is a nonlinear function of the underlying, therefore, hedging is instantaneous and hedging with discrete rebalancing gives rise to error. Frequent rebalancing can be impractical due to transactions costs. In the literature, consequences of discrete time hedging have been considered usually in conjunction with the existence of transaction costs, that's why hedgers would like to trade at least frequently as possible. Pioneered by Leland [43], asymptotic approaches are used as well [44-46]. For most MTM classes, delta-gamma and delta-vega hedging strategies are shown to perform better in dynamic hedging when compared with delta hedging strategy, regardless of the model used. The delta-gamma strategy enables the performance of a discrete rebalanced hedging to be improved. The delta-vega strategy corrects partly for the risk of a randomly changing volatility.

5. Conclusion

This paper is concerned with improving the dynamic hedging accuracy using generated genetic programming implied volatilities. Firstly, genetic programming is used to predict implied volatility from index option prices. Dynamic training-subset selection methods are applied to improve the robustness of genetic programming to generate general forecasting implied volatility models relative to static training-subset selection method. Secondly, the implied volatilities derived are used in dynamic hedging strategies and the performance of genetic programming is compared to that of Black-Scholes in terms of delta, gamma and vega hedging.

Results show that the dynamic training of genetic programming yields better results than those obtained from static training with fixed samples, especially when applied on time series and moneyness-time to maturity samples simultaneously. Based on the MSE total as performance criterion, three generated genetic programming volatility models are selected M4S4, MCAR and MGAR. However, the MGAR seems to be more accurate in forecasting

implied volatility than MCAR and M4S4 models because it is more general and adaptive to all time series and moneyness-time to maturity classes simultaneously.

The main conclusion concerns the importance of implied volatility forecasting in conducting hedging strategies. Genetic programming forecasting volatility makes hedge performances higher than those obtained in the Black-Scholes world. The best genetic programming hedging performance is achieved for in-the-money call options and at-the-money put options in all hedging strategies. The percentage of cases where the hedging error of the genetic programming model is less than the Black-Scholes hedging error is around 59% for calls and 57% for puts. The performance of genetic programming is pronounced essentially in terms of delta hedging for call and put options. The percentage of cases where the delta hedging error of the genetic programming model is less than the Black-Scholes delta hedging error is 100% for out-of-the money and in-the-money call options as well as for at-the-money and out-of-the-money put options. The percentage of cases where the delta-vega hedging error of the genetic programming model is less than the Black-Scholes delta-vega hedging error is 100% for in-the-money call options as well as for at-the-money put options. The percentage of cases where the delta-gamma hedging error of the genetic programming model is less than the Black-Scholes delta-gamma hedging error is 100% for at-the-money put options.

Finally, improving the accuracy of implied volatility forecasting using genetic programming can lead to well hedged options portfolios relative to the conventional parametric models.

Our results suggest some interesting issues for further investigation. First, the genetic programming can be used to hedge options contracts using implied volatility of other models than Black-Scholes model, notably stochastic volatility models and models with jump, as a proxy for genetic programming volatility forecasting. Further, the hedge factors can be computed numerically not analytically. Second, this work can be reexamined using data from individual stock options, American style index options, options on futures, currency and commodity options. Third, as the genetic programming can incorporate known analytical approximations in the solution method, parametric models such as GARCH models can be used as a parameter in the genetic programming to build the forecasting volatility model and the hedging strategies. Finally, the genetic programming can be extended to allow for dynamic parameter choices including the form and the rates of genetic operators, the form and pressure of selection mechanism, the form of replacement strategy and the size of population. This dynamic genetic programming method can improve the performance without extra calculation costs. We believe these extensions are of interest for application and will be object of our future works.

Author details

Fathi Abid and Wafa Abdelmalek
Research Unit MODESFI, Faculty of Economics and Business, Sfax, Tunisia

Sana Ben Hamida
Research Laboratory SOIE (ISG Tunis), Paris West University, Nanterre, France

6. References

[1] Blair B.J, Poon S, Taylor S.J (2001) Forecasting S&P100 Volatility: The Incremental Information Content of Implied Volatilities and High Frequency Index Returns. Journal of Econometrics.105: 5-26.

[2] Busch T, Christensen B.J, Nielsen M.Ø (2007) The Role of Implied Volatility in Forecasting Future Realized Volatility and Jumps in Foreign Exchange, Stock, and Bond Markets. CREATES Research Paper 2007-9. Aarhus School of Business, University of Copenhagen. pp.1-39.

[3] Koza J.R (1992) Genetic programming: on the Programming of Computers by means of Natural Selection. Cambridge, Massachusetts: the MIT Press. 819 p.

[4] Holland J.H (1975) Adaptation in Natural and Artificial Systems. Ann Arbor: University of Michigan Press.

[5] Breiman L (1996) Bagging Predictors. Machine Learning. 2:123-140.

[6] Freund Y, Schapire R (1996) Experiments with a New Boosting Algorithm. In Proceedings of the 13th International Conference on Machine Learning. Morgan Kauffman Publishers. pp. 148-156.

[7] Breiman L (1998) Arcing Classifiers. Annals of Statistics. 26: 801-849.

[8] Abdelmalek W, Ben Hamida S, Abid F (2009) Selecting the Best Forecasting-Implied Volatility Model using Genetic programming. Journal of Applied Mathematics and Decision Sciences (Special Issue: Intelligent Computational Methods for Financial Engineering). Hindawi Publishing Corporation. Available: http:// www.hindawi.com/journals/jamds/2009/179230.html

[9] Tsang E, Yung P, Li J (2004) EDDIE-Automation, a Decision Support Tool for Financial Forecasting. Decision Support Systems. 37: 559–565.Available: http://sci2s.ugr.es/keel/pdf/specific/.../ science2_4.pdf

[10] Kaboudan M (2005) Extended Daily Exchange Rates Forecasts using Wavelet Temporal Resolutions. New Mathematics and Natural Computing. 1: 79-107. Available: http://www.mendeley.com/.../extended-daily-... - États-Unis

[11] Bollerslev T, Chou R.Y, Kroner K.F (1992) ARCH Modelling in Finance: a Review of the Theory and Empirical Evidence. Journal of Econometrics. 52: 55-59.

[12] Engle R.F (1982) Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of U.K. Inflation. Econometrica. 50: 987-1008.

[13] Bollerslev T (1986) Generalized Autoregressive Conditional Heteroscedasticity. Journal of Econometrics. 31: 307-327.

[14] Hull J, White A (1987) The Pricing of Options on Assets with Stochastic Volatilities. Journal of Finance. 42: 218-300.

[15] Scott L (1987) Option Pricing When the Variance Changes Randomly: Theory, Estimation and an Application. Journal of Financial and Quantitative Analysis. 22: 419-438. Available: http:// www.globalriskguard.com/resources/.../der6.pdf

[16] Wiggins J (1987) Option Values under Stochastic Volatility: Theory and Empirical Evidence. Journal of Financial Economics. 19: 351-372.

[17] Heston S.L (1993) A Closed-Form Solution for Options with Stochastic Volatility. Review of Financial Studies. 6: 327-344.

[18] Ma I, Wong T, Sankar T, Siu R (2004) Volatility Forecasts of the S&P100 by Evolutionary Programming in a Modified Time Series Data Mining Framework. In: Jamshidi M, editor. Proceedings of the World Automation Congress (WAC2004). 17: 567-572.

[19] Chen S.H, Yeh C.H (1997) Using Genetic programming to Model Volatility in Financial Time Series. In: Koza J.R, Deb K, Dorigo M, Fogel D.B, Garzon M, Iba H, Riolo R.L, editors. Genetic programming 1997, Proceedings of the Second Annual Conference. Morgan Kaufmann Publishers. pp. 58-63.

[20] Zumbach G, Pictet O.V, Masutti O (2002) Genetic programming with Syntactic Restrictions Applied to Financial Volatility Forecasting. In: Kontoghioghes E.J, Rustem B, Siokos S, editors. Computational Methods in Decision-Making, Economics and Finance. Kluwer Academic Publishers. pp. 557-581.

[21] Neely C.J, Weller P.A (2002) Using a Genetic Program to Predict Exchange Rate Volatility. In: Chen S.H, editor. Genetic Algorithms and Genetic programming in Computational Finance, Chapter 13. Kluwer Academic Publishers. pp. 263-279.

[22] Ma I, Wong T, Sanker T (2006) An Engineering Approach to Forecast Volatility of Financial Indices. International Journal of Computational Intelligence. 3: 23-35.

[23] Ma I, Wong T, Sanker T (2007) Volatility Forecasting using Time Series Data Mining and Evolutionary Computation Techniques. In Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO 07). ACM New York Press.

[24] Derman E, Ergener D, Kani I (1995) Static Options Replication. The Journal of Derivatives. 2: 78-95.

[25] Carr P, Ellis K, Gupta V (1998) Static Hedging of Exotic Options. Journal of Finance. 53: 1165-1190.

[26] Hutchinson J.M, Lo A.W, Poggio T (1994) A NonParametric Approach to Pricing and Hedging Derivative Securities via Learning Network. Journal of Finance. 49: 851-889.

[27] Aït-Sahalia Y, Lo A (1998) Nonparametric Estimation for State-Price Densities Implicit in Financial Asset Prices. The Journal of Finance. 53: 499-547.

[28] Chen S.H, Lee W.C, Yeh C.H (1999) Hedging Derivative Securities with Genetic Programming. International Journal of Intelligent Systems in Accounting, Finance and Management. 4: 237-251.

[29] Harvey C.R, Whaley R.E (1991) S&P 100 Index Option Volatility. Journal of Finance. 46: 1551-1561.

[30] Harvey C.R, Whaley R.E (1992) Market Volatility Prediction and the Efficiency of the S&P100 Index Option Market. Journal of Financial Economics. 31: 43-73.

[31] Merton R.C (1973) Theory of Rational Option Pricing. Bell Journal of Economics and Management Science. 4: 141-183.

[32] Cai W, Pacheco-Vega A, Sen M, Yang K.T (2006) Heat Transfer Correlations by Symbolic Regression. International Journal of Heat and Mass Transfer. 49: 4352-4359.

[33] Gustafson S, Burke E.K, Krasnogor N (2005) On Improving Genetic programming for Symbolic Regression. In Proceedings of the IEEE Congress on Evolutionary Computation. 1: 912-919.

[34] Keijzer M (2004) Scaled Symbolic Regression. Genetic programming and Evolvable Machines. 5: 259-269.

[35] Lew T.L, Spencer A.B, Scarpa F, Worden K (2006) Identification of Response Surface Models Using Genetic programming. Mechanical Systems and Signal Processing. 20: 1819-1831.

[36] Black F, Scholes M. (1973) The Pricing of Options and Corporate Liabilities. Journal of Political Economy. 81: 637-659.

[37] Kraft D. H, Petry F. E, Buckles W. P, Sadasivan T (1994) The Use of Genetic Programming to Build Queries for Information Retrieval. In Proceedings of the 1994 IEEE World Congress on Computational Intelligence. IEEE Press. pp. 468–473.

[38] Angeline P. J (1996) An Investigation into the Sensitivity of Genetic Programming to the Frequency of Leaf Selection during Subtree Crossover. In: Koza J. R et al., editors. Genetic Programming 1996: Proceedings of the First Annual Conference. MIT Press. pp. 21–29. Available: www.natural-selection.com/Library/1996/gp96.zip.

[39] McKay B, Willis M.J, Barton G.W (1995) Using a Tree Structural Genetic Algorithm to Perform Symbolic Regression. In First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA). 414: 487-492.

[40] Schwefel H.P (1995) Numerical Optimization of Computer Models. John Wiley & Sons, New York.

[41] Cavaretta M.J, Chellapilla K. (1999) Data Mining Using Genetic Programming: The Implications of Parsimony on Generalization Error. In Proceedings of the 1999 Congress on Evolutionary Computation (CEC' 99). IEEE Press. pp. 1330-1337.

[42] Gathercole C, Ross P (1994) Dynamic Training Subset Selection for Supervised Learning in Genetic Programming. Parallel Problem Solving from Nature III. 866 of LNCS: 312-321.

[43] Leland H.E. (1985) Option Pricing and Replication with Transaction Costs. Journal of Finance. 40: 1283-1301.

[44] Kabanov Y.M, Safarian M.M (1997) On Leland Strategy of Option Pricing with Transaction Costs. Finance Stochastic. 1: 239-250.

[45] Ahn H, Dalay M, Grannan E, Swindle G (1998) Option Replication with Transactions Costs: General Diffusion Limits. Ann. Appl. Prob. 8: 676-707.

[46] Grandits P, Schachinger W (2001) Leland's Approach to Option Pricing: The Evolution of Discontinuity. Math Finance. 11: 347-355.

IntechOpen