# An Adaptive Evolutionary Approach to Option Pricing via Genetic Programming

**N. K. Chidambaran, Chi-Wen Jevons Lee, and Joaquin R. Trigueros***

November 1998

Please do not quote without permission

---

* Chidambaran is visiting at NYU, on leave from Tulane. Lee holds joint appointments at Tulane and HKUST. Trigueros is at Tulane. We are grateful for the comments from participants at seminars at Tulane and HKUST, the 1998 FMA meetings, and the 1998 *Conference on Computational Intelligence for Financial Engineering*.

**Correspondence:**
N. K. Chidambaran, 44 West 4th Street, MEC 9-160, New York, NY 10012.
Phone: (212) 998-0318; Fax: (212) 995-4233; E-mail: chiddi@stern.nyu.edu

# An Adaptive Evolutionary Approach to Option Pricing via Genetic Programming


## Abstract

We propose a methodology of Genetic Programming to approximate the relationship between the option price, its contract terms and the properties of the underlying stock price. An important advantage of the Genetic Programming approach is that we can incorporate currently known formulas, such as the Black-Scholes model, in the search for the best approximation to the true pricing formula. Using Monte Carlo simulations, we show that the Genetic Programming model approximates the true solution better than the Black-Scholes model when stock prices follow a jump-diffusion process. We also show that the Genetic Programming model outperforms various other models in many different settings. Other advantages of the Genetic Programming approach include its robustness to changing environment, its low demand for data, and its computational speed. Since genetic programs are flexible, self-learning and self-improving, they are an ideal tool for practitioners.


The Black-Scholes model is a landmark in contingent claim pricing theory and has found wide acceptance in financial markets. The search for a better option pricing model continues, however, as the Black-Scholes model was derived under strict assumptions that do not hold in the real world and model prices exhibit systematic biases from observed option prices. While many extensions and alternative models have been suggested, none seems to be complete (Rubinstein, 1997). We propose a new methodology of Genetic Programming for better approximating the elusive relationship between the option price and its contract terms and properties of the underlying stock price. This method requires minimal assumptions and can easily adapt to changing and uncertain economic environments.

Many researchers have attempted to explain the systematic biases of the Black-Scholes model as an artifact of its assumptions.[1] The most often challenged assumption is the normality of stock returns.[2] Merton (1976) and Ball and Torous (1985) propose a Poisson jump-diffusion returns processes. French, Schwert and Stambaugh (1987) and Ballie & DeGennaro (1990)

[1] Black-Scholes pricing biases have been related to volatility, strike price, time to maturity, volume and leverage. Black and Scholes (1972) and Galai (1977) show that it is possible to make excess returns by buying "undervalued" (relative to the Black-Scholes price) and selling "overvalued" options. Mac Beth and Merville (1979) find that Black-Scholes overprices out-of-the-money and underprices in-the-money options. Rubinstein (1985) finds the same pricing biases as Mac Beth and Merville (1979) in the first half of his data set and the opposite biases in the second half.

[2] Normality of stock returns has been repeatedly rejected. Indeed, Kim and Kon (1994) have ranked candidates for return distributions and found normality to be the least likely. Their rankings are: 1) Intertemporal dependence models (ARCH, GARCH), 2) Student t, 3) Generalized mixture of normal distributions, 4) Poisson jump, and 5) Stationary normal.

advocate GARCH (Bollerslev 1986) processes.[3] While closed-form solutions for the option price cannot be obtained for all these models, pricing formulas can be obtained numerically.

The difficulty in finding an analytical closed-form parametric solution has also led to non-parametric approaches. Rubinstein (1997) suggests that we examine option data for the implied binomial tree to be used for pricing options. Chidambaran and Figlewski (1995) use a quasi-analytic approximation based on Monte Carlo simulation. Hutchinson, Lo and Poggio (1994) build a numerical pricing model using neural networks. Our approach, using Koza's (1992) Genetic Programming to develop an adaptive evolutionary model of option pricing, is also data driven and non-parametric. We show that this method is well suited to the task and offers some advantages over learning networks. In particular, it can operate on small data sets, circumventing the large data requirement of the neural network approach noted by Hutchinson, Lo, and Poggio (1994).

The philosophy underlying Genetic Programming is to replicate the stochastic process by which genetic traits evolve in offspring, through a random combination of the genes of the parents, in the biological world. A random selection of equations of the option contract terms and basic statistical properties of the underlying stock price will have among them some elements that will ultimately make up the true option pricing formula. By selectively breeding the equations, presumably these elements will be passed onto future generations of equations that can price options more accurately. The essence of our method is the selection of equation components, i.e. genetic traits, which parents pass on to the next generation. Since it is impossible to determine which element is the best ex-ante, our focus is on choosing parents that seem to be the fittest. The genes to be propagated to the next generation are thus selected on the basis of the pricing errors of the equations.

The method of selecting equations, with their embedded "genetic traits," to serve as parents for the next generation can have important implications for model efficiency. In this paper, we examine six alternative parent-selection methods: Best, Fitness, Fitness-overselection, Random, Tournament with 4 individuals and Tournament with 7 individuals. These methods vary in the degree to which they preserve randomness as new generations evolve. Finding the most efficient process for selecting the best among alternative genetic traits is an empirical issue. We find that the Fitness-overselection method seems to offer the best results for option pricing. We

[3] Heston (1993) finds that a pricing formula derived under the assumption that the returns process is log-gamma outperforms Black-Scholes.

also explore the effect of varying other model parameters, such as the properties of the data set required to train the genetic programs, on model efficiency.

An important advantage of the Genetic Programming approach over other numerical techniques is its ability to incorporate known approximate solution into the initial "gene pool" to be used in evolving future generations. In this paper, we include the Black-Scholes model in the initial gene pool. This approach can reach the true pricing model more efficiently as it begins the search from a locally optimum solution. We illustrate how this approach quickly *adapts* the Black-Scholes model to a jump-diffusion process, where the Black-Scholes assumption of returns normality does not hold and for pricing options in the real world. We find that the Genetic Programming formulas beats the Black-Scholes equation in 9 out of 10 runs when the underlying stock prices are generated by a jump-diffusion process and in 10 out of 10 runs when we apply the analysis to the S&P Index options. The method also outperforms the Black-Scholes model for four out of five stocks in our sample.

The paper proceeds as follows. In Section I, we introduce genetic programming and highlight its advantages over other non-parametric methods. In Section II, we assess the ability of Genetic Programming in learning the Black-Scholes model, given data that are simulated according to the assumptions of the Black-Scholes world. In Section III, we construct a non-Black-Scholes world and show how Genetic Programming can adapt the Black-Scholes model to its specifications. In Section IV, we show how Genetic Programming adapts Black-Scholes to the real world. In Section V, we conclude.

## I. Genetic Programming – A Brief Overview

Genetic Programming is a technique that applies the Darwinian theory of evolution to develop efficient computer programs.[4] In this section we describe the mechanics of the approach and the various ways to improve its efficiency.

[4] Genetic Programming is an offshoot of Genetic Algorithms. Genetic Algorithms have been used to successfully develop technical trading rules by Allen and Karlajainen (1993) for the S&P 500 index and by Neely, Weller, and Dittmar (1997) for foreign exchange markets. Genetic Programming has also been used in heterogeneous multi-agent economies by Marimon, McGrattan and Sargent (1990), in multi-agent financial markets by Lettau (1997), and in multi-agent games by Ho (1996).

**A. Basic Approach**

We use a variant of Genetic Programming called Genetic Regression, where the desired program is a function that relates a set of inputs such as share price, option exercise price, etc. to one output, the option price. The set of data on which the program operates to determine the relationship between input parameters and the options price is called *the training set*. The set of data on which the resulting formula is tested is called *the test set*. The procedure of the basic approach is described as follows.

- Given a problem to be solved and a training set of matched inputs and outputs, a set of possible formulas is randomly generated. These formulas are functions of some or all of the independent variables and randomly generated constants. The four trees in Figure 1 shows how a formula can be encoded as a tree of components. Each formula is an *individual* and the set of individuals is called the *population*. The size of the population is held constant and is a control variable for optimizing the modeling process.
- Every individual in the population is evaluated to test whether it can accurately price options in the training data set. We assign a fitness measure to select the surviving gene. A smaller mispricing for the training data set indicates a better fit.
- Based on a fitness measure, a subset of the population is selected to act as the parents for the next *generation* of the population of formulas.
- A pair of the parents generates a pair of offspring. Components of the parent formulas are *crossed* to generate offspring formulas. A random point is selected in each parent tree. The sub-trees below that random point are switched between the two parent formulas. This operation creates a new pair of individuals, the *offspring.* Figure 1 shows a pair of parents and their offspring after crossover. It is possible that no crossover is performed and the parents themselves are placed in the new population (a clone). The process of selection and crossover is repeated until the new generation is completely populated.
- The individuals in the new population are tested to gauge their performance in pricing options. The steps above are repeated for a pre-specified number of times, or *generations*. Evolutionary pressure in the form of fitness-related selection combined with the crossover operation eventually produces populations of highly fit individuals. We keep track of the best-fit individual found throughout this process and set it as the solution to the option pricing problem.

**Insert Figure 1 Here**

**B. Parent Selection Criteria**

The method of selecting parents for the next generation can affect the efficiency of genetic programs. We examine six different selection methods: Best, Fitness, Fitness-overselection, Random, Tournament with 4 individuals and Tournament with 7 individuals. These methods represent various attempts to preserve a degree of randomness in the evolutionary process.

In the Best method, individuals are ranked in terms of their fitness, ascending in the order of magnitude of their errors. The individuals with the smallest errors are thus picked to serve as parents of the next generation. In the Fitness method, individuals are selected randomly with a probability that is proportional to their fitness. In the Fitness-overselection method, 400 individuals are classified into two groups. Group 1 has 320 best-fit individuals and Group 2 has the remainder. Individuals are selected randomly with an 80% probability from Group 1 and a 20% probability from Group 2. In the Random method, the fitness of the individuals is completely ignored and parents are chosen at random from the existing population. Finally, in the Tournament method, n individuals are selected at random from the population and the best-fit individual is chosen to be a parent. We examine Tournament method with n=4 and n=7.

**C. Advantages of Genetic Programming**

An important advantage of Genetic Programming is its capability of incorporating a known analytical approximation to the solution into the program. In this paper, we include the Black-Scholes model as an initial parameter, i.e. part of the initial gene pool, for the algorithm. We can also include components of the formula instead of the entire formula. Since our method begins with a known approximation, it increases the probability of finding the true pricing formula and reduces computing time. Genetic Programming requires smaller training sets than Neural Networks which is a popular alternative adaptive learning algorithm (see Hutchinson, Lo, and Poggio (1994) and Koza (1992)). Since most options are thinly traded, Genetic Programming is an ideal tool for option pricing,

The methodology can also be made robust to changing environmental conditions and can operate on data sets generated over a range of possible conditions. We make the population robust by stochastically changing the training sets in the middle of the evolution. Only individuals with the desirable characteristics that are well adapted to changing environments will

survive. The problem of over-fitting, in particular, is easily resolved by this approach. Further, new formulas can evolve out of previously optimal solutions when the data set contains structural changes rather than requiring retraining from scratch like in learning networks. Since genetic programs are self-learning and self-improving, they are an ideal tool for practitioners.

**D. Convergence Characteristics of Genetic Algorithms and Programs**

Our implementation of the Genetic Programming is effectively a search over the space of functions that can be constructed from a user-defined set of base variables and operations. This space of functions is generally infinite. However, the Genetic Programming algorithms are aided by the fact that we *limit* the search space and that the search is a *parallel* search.

We control and limit the complexity of the problem by setting a maximum depth size of 17[5] for the trees used to represent formulas. The search space is, however, still very large and it is computationally inefficient to examine every possible tree. The implicit parallelism of Genetic Algorithms, however, ensures that the search is efficient. The central idea behind the parallelism of Genetic Algorithms is that each of the formula elements define *hyperplanes*, i.e. sub-regions of the search space. In the population of candidate formulas, all the elements are present, and the fitness of each formula is a function of how many of the elements of the true pricing formula is present in the individual being evaluated. All formulas that contain a particular element will have similar errors and an evaluation of the formulas in the population is a parallel search for the hyperplanes containing the elements that make up the true option pricing model. For example, the Black-Scholes formula is:

$$C = SN(d1) - Xe^{-r\tau}N(d2) \qquad (1)$$

where,

$$d1 = [\ln(S/X) + (r + \sigma^2/2)\tau]/\sigma\sqrt{\tau} \quad \text{and} \quad d2 = \text{d1} - \sigma\sqrt{\tau}$$

N(d1) and N(d2) are the cumulative standard normal values for d1 and d2, S is the current stock price, X is the exercise price, r is the risk free rate, $\tau$ is the option time to maturity and $\sigma$ is the volatility of the underlying stock. We can treat the formula to be the point at which the

[5] A 17 deep tree is a popular number used to limit the size of tree sizes Koza(1992). Practically, we chose the maximum depth size possible without running into excessive computer run times. Note that the Black-Scholes formula is represented by a tree of depth size 12. A depth size of 17, therefore, is large enough to accommodate complicated option pricing formulas and works in practice.

hyperplanes containing the term S N(d1) and -X e-$^{r\tau}$ N(d2) intersect. Searching over a randomly generated set of formulas is, therefore, a parallel search over a set of hyperplanes.

The true option pricing formula will consist of many different elements that form a set of hyperplanes and these is called its *schemata.* The individual sub-regions formed by the hyperplanes are the *schema*. If an individual equation contains elements that represents a superior region of the search space, it will generally be reflected as better fitness for the equation. This will increase the individual's chance to reproduce and pass on its schema to the next generation. When used to solve problems that involves a search for the sequence of elements that make up a gene, or any problem that involves a search for a sequence of numbers, Holland (75) and Koza (92) and show that the schemata of the Genetic Algorithm search process is extremely efficient and the algorithm converges. In this paper, we implicitly test whether such an approach will also work when searching for a closed-form option pricing model.

## II. Genetic Programming in a Black-Scholes World

In this section, we test the capacity of Genetic Programming to learn the Black-Scholes model, paralleling the study by Hutchinson, Lo, and Poggio (1994). Data to train the Genetic Programming is generated through Monte-Carlo simulation. For each data set, price paths of the underlying stock with initial value $S_0$ =50 are simulated for 504 days (24 months * 21 days/month). Stock returns are assumed to follow a diffusion process $dS(t)/S(t) = \mu dt + \sigma dW(t)$ with annual continuously compounded expected return $\mu$=0.10, standard deviation $\sigma$=0.20 and risk-free rate $r$=0.05. Stock price at time $t$ is calculated as:

$$S(t) = e^{\sum_{i=1}^{t} Z_t} \quad ; \quad t = 1,.., 504. \qquad (2)$$

We next generate a sample of call options for each stock price realization. CBOE rules (Hull (1993)) were used to create call options with varying strikes and maturity for each day of the simulated price path. Option prices are derived for each simulated option, using the Black-Scholes equation. We thus have a sample of simulated options data. We adopted many of the simplifications suggested by Hutchinson, Lo, and Poggio (1994) in generating the data sample, for example, we hold the annual volatility $\sigma$ and riskless rate $r$ constant throughout. Figure 2 shows a stock price path generated by Geometric Brownian motion and the distribution of its associated option prices.

**Insert Figure 2 Here**

Table I describes the specifications of the Genetic Programming model. We use the four basic mathematical operations, the log function, the exponential function, the square root function, and the cumulative Normal distribution. The basic division operation is protected against division by zero and the log and square root functions are protected against negative arguments. The current stock price, option exercise price, option intrinsic value, and option time-to-maturity are input parameters. The functional representation of a formula is assumed to be 17-step deep and allows sufficient variation in the complexity of the formulas without overwhelming memory requirements. For example, the Black-Scholes model can be represented in a 17-step tree using the operations and variables described above as shown in Appendix A. The formula population size is set to 25,000 and we run the program for a maximum of 51 generations.

**Insert Table I Here**

It should be noted that the restrictions on Genetic Programming are far fewer than those required for Neural Networks. Only the variables needed for pricing options have to be specified. We need not make assumptions on the smoothness or complexity of the formulas beyond the maximum allowable depth (tree size) for representing a formula.

We implement ten trial runs of genetic programs. For each run, a 5% sub-sample of entire simulated options is used as the training set. The data set is stochastically changed in the middle of training run to prevent over-fitting. We find that evaluating the population formulas on such stochastic subsets of the data set resulted in reduced training times and better out-of-sample performance. Only robust formulas can survive the constantly changing environment and pass on their "traits" to the next generation.

The criterion for selecting the surviving formulas is a linear combination of the absolute pricing errors and the percentage pricing errors. We found that the formulas consistently made relatively small absolute errors when pricing out-of-the-money options and relatively large absolute errors when pricing in-the-money options. The pattern in the magnitudes of the percentage error was just the opposite. Linear combination of these two error measurements leads to a more efficient selection rule. In the classic Genetic Programming fashion, we define the fitness of a formula to be:

$$\frac{1}{1+\sum_{i=1}^{number\ of\ fitness\ cases}\varepsilon_i} \qquad (3)$$

where $\varepsilon_i$ is the training error for the *i*th case. This training error is defined as the sum of percentage and dollar errors if the Black-Scholes value was greater than \$0.01 and just the dollar error if the Black-Scholes value was less than \$0.01.[6]

The final formulas generated are complex functions of the parameters specified in Table I. For example, a formula generated by the fitness-overselection parent selection method is:

$$C(S,X,\tau) = \max(S-X,0) + \tau*(\tau+\log(X))*N\begin{pmatrix} -1.97548+3*Max(S-X,0)+2*(S/X) \\ 4*\tau+2*N(-0.98774+\log(S/X))+ \\ N(-0.98774+Max(S-X,0)+\log(S/X)) \\ +N(\log(S/X)*\log(X))+12*\log(S/X)+ \\ \log(S/X)*\log(X) \end{pmatrix} \quad (4)$$

where, S is the current stock price, X is the exercise price, $\tau$ is the time to maturity.

We measure the performance of Genetic Programming on an out-of-sample two-dimensional options grid of option maturities and strike prices. Tables II and appendix A present the absolute and percentage pricing errors respectively for the Genetic Programming formulas for each of the six different parent selection algorithms. Each cell in the table is the average pricing errors across ten different Genetic Programming formulas.

**Insert Table II**

In Table II, Figure 3, and appendices A and B, we observe a pattern of performance of the Genetic Programming model. First, the dollar pricing errors are small for short-maturity options as opposed to long-maturity options. However, the percentage pricing errors are just the opposite. Obviously this is because the magnitude of option prices vary substantially across option maturities. Second, we find that the errors vary across the option strike. Once again this is because option prices are very small for out-of-the money options and much higher for in-the-money options. Our fitness criterion balances the two effects by minimizing a combination of the absolute and percentage pricing errors. We found that this allows us to control the pricing errors for out-of-the money and in-the-money options without adversely affecting the errors for at-the-money options. Note also that the percentage pricing errors are very large for short-

[6] If, for example, the true price of an option is \$2.00 and one of our Genetic Programming formulas gives a price of \$2.20, then percentage error is small (10%) but dollar error is \$0.20, which is economically significant. If, on the other hand, the true price is \$0.10 and our formula gives a price of \$0.07, dollar error is small (\$0.03) but the price is off by 30%. Our error measure is then 30 (10% + \$0.20) in the first case and 33 (30% + \$0.03) in the second case.

maturity out-of-the-money options. This is because prices are very small for these options, and any error is magnified out of proportion. Absolute pricing errors are, however, less than 2 cents for these options.

**Insert Figure 3 Here**

While all the parent selection methods yield similar qualitative results, they vary widely in efficiency. The largest pricing error is for the 122-day maturity at-the-money option and varies from a minimum of $1.17 or 33.6% for the Fitness-overselection method to $2.34 for the Fitness method. The smallest absolute pricing error is for short-maturity options that are out-of-the money, e.g. the Fitness-overselection method has an error of $0.02 for a 10 day option that has a ratio of stock price to strike price of 0.93. The same option is priced with an error of $0.01 with the Tournament method (n=7) and an error of $0.06 for the Random method. The percentage pricing error for this option is, however, seemingly very high at 73.4% for the Fitness-overselection method and 285.5% for the Random method. Obviously this is because the option price itself is quite small and magnifies the percentage pricing error. While the errors are larger for longer maturity and in-the-money options, they are within an acceptable percentage range. The error is within a few pennies for short-maturity and out-of-the money options. Overall, we found the Fitness-overselection method and the Tournament method (n=7) providing the best results and that Genetic Programming gives a good numerical approximation to the Black-Scholes model.

## III. Performance Analysis in a Jump-Diffusion World

The Genetic Programming approach can incorporate any known analytical approximation into its algorithm. It is flexible to adapt to changing and unknown economic environments. In this section, we illustrate how the Genetic Programming model can adapt and outperform the Black-Scholes model in a jump-diffusion world described by Merton (1976). Since the closed form solution for the option prices in a jump-diffusion world is available, we can measure the pricing errors from the Genetic Programming model and the Black-Scholes model in such a world. The performance analysis highlights the salient features of the Genetic Programming approach to option pricing.

The jump-diffusion process is a combination of a Geometric Brownian diffusion process and a Poisson jump process and can be written as:

$$dS(t) / S(t) = (\mu - \lambda k)dt + \sigma dW(t) + dq \qquad (5)$$

where *dq* is the Poisson-lognormal jump process. The Poisson process determines when a jump occurs and jump size is lognormally distributed.

We simulate the price path of daily stock prices over a 24 month period with the initial price set at $S_0$ =50. Each month is assumed to have 21 trading days. The diffusion parameters $\mu$ (mean) and $\sigma$(standard deviation) were set at 10% and 20% respectively, and jump parameters *k*(jump size), $\lambda$ (jump rate), and $\delta$ (standard deviation of the log-jumps), were set at 0.02, 25 and 0.05 respectively.[7] These values are well within the range estimated by stock price data. Thus, 504 stock prices, S(t), are simulated using random daily returns $z_t \sim N((\mu-\sigma^2/2-k)/252, \sigma/252)$ and $n(t) \sim Poisson(\lambda t)$ jumps, each of magnitude $Y_j$ (where $\ln Y_j \sim N(\ln(1+k)-0.5\delta^2, \delta)$), for each t in {1…504}:

$$S(t) = S_0 e^{\sum_{i=1}^{t} z_i} Y(n(t)),\ t = 1,..,504 \qquad (6)$$

$$Y(0) = 1$$

where, $$Y(n(t)) = \prod_{j=1}^{n(t)} Y_j, n(t) > 0$$

We use CBOE rules to create call options from the simulated stock price path. Figure 4 illustrates a sample stock price path and its associated distribution of option prices in a jump-diffusion world.

**Insert Figure 4 Here**

Options are priced using Merton's (1976) jump diffusion formula given below, truncated at the point when the marginal contribution of additional terms is negligible.[8]

[7] This translates into 25 expected jumps per year, each inducing an expected percentage change of 2% on the stock price. The variance of the log-jumps is 0.05.

[8] Terms in the sum increase and then decrease in magnitude due to the distribution of the attached probabilities. All terms in the decreasing segment of the series whose marginal contribution was less than 0.00001% were dropped, as well as all terms beyond the 1000th in the sum.

$$F(S, X, r, \sigma, \tau, \lambda, k, \delta) = \sum_{n=0}^{\infty} \frac{e^{-\lambda' \tau}(\lambda' \tau)^n}{n!} f_n(S, \tau)$$

where, (7)

$$\lambda' = \lambda(1+k)$$
$$f_n = Black - Scholes(S, X, r_n, v_n, \tau)$$
$$r_n = r - \lambda k + \frac{n \ln(1+k)}{\tau}$$
$$v_n = \sigma^2 + \frac{n\delta^2}{\tau}$$
$$\tau = Time\ to\ maturity\ T\text{-}t$$

Panel A of Table IV presents the set of operations and variables used to develop the Genetic Programs. In addition to those used in the previous section where the objective was to learn the Black-Scholes model, we include the Black-Scholes option values as a component of the formula tree. This provides a good starting point for finding a solution and is a way in which we can adapt known analytical approximations to find a better approximation. We, however, correct for the volatility estimate that the investor would have calculated using a history of observed prices, which is a combination of the variances of the diffusion and jump processes. This reflects the approach of a naive investor who is unaware of the true nature of the underlying stock price process when using the Black-Scholes model to price option. The estimated call option value with the modified Black-Scholes model is, therefore, (Merton 1976):

$$C = BS(S, X, r, \sqrt{\sigma^2 + \lambda\delta^2}, \tau) \quad (8)$$

Panels B and C of Table III present the size of the training sets used and the algorithm training criteria. We implement an additional step in determining the size of the population and the number of generations. Since the Black-Scholes model is a very good approximation by itself for option pricing in a jump-diffusion world, we can use it as a benchmark to evaluate the effect of population size and number of generations on the efficiency of the genetic programs. Using an independent set of 25% of the options as a training set, we determine that a minimum population size of 5000 functions and 10 generations is needed to get a formula that outperforms the Black-Scholes model. We use these parameters on ten new 25% subsets of the options created to develop the genetic program for option pricing in a jump-diffusion world. We also train formulas on a smaller 5% subset of the total options data set. We, however, do not update any algorithm parameters.

**Insert Table III Here**

The formulas generated by the genetic program are adaptations of the Black-Scholes model. For example, one of the runs resulted in the formula,

$$C(S, X, \tau) = \sqrt{C_{Black-Scholes} * \left[0.11734 + \sqrt{0.95461 * C_{Black-Scholes} * (C_{Black-Scholes} + \tau)}\right]} \quad (9)$$

where, $C_{Black\text{-}Scholes}$ is the Black-Scholes formula, $\tau$ is the time to maturity, and N(.) is the cumulative normal distribution.

We examine the performance of our genetic program on ten out-of-sample test sets of option data. Table IV shows details of the absolute and percentage pricing errors for each of the six parent selection methods. The Fitness-overselection method gives the lowest pricing errors and beats the modified Black-Scholes model in each of the ten out-of-sample tests. The next best performance is the Tournament method with n=7. Clearly, the Genetic Programming model based on Fitness-overselection outperforms the Black-Scholes model in out-of-sample tests. The only measure in which the original Black-Scholes model ever beats the Genetic Programming formula is in the training-set sum of percentage errors, and this occurs for *only* 1 out of the 10 Genetic Programming formulas. However, the error is large enough to blow up the average percentage error. We attribute this fluke to our decision to ignore during training all percentage errors for options worth less than $0.01. Note that the Genetic Programming formula performs better than the Black-Scholes model for each of the 10 out-of-sample test sets.

**Insert Table IV Here**

We also address an important criticism usually leveled at complex numerical methodologies. Can the method perform any better than a simple linear regression model? While linear regression of the options price on variables such as the options strike and current stock price can result in an equation that gives small errors within the sample, it is obvious that out-of-the sample option values will be priced with larger errors. It is, however, a useful benchmark. We, therefore, run single-stage and two-stage linear regressions with and without Black-Scholes model as an independent variable. The two-stage model represents separate equations for in-the-money and out-of-the-money options.

Table V presents the pricing errors for the Genetic Programming formulas, the Black-Scholes equation, and for the linear models. The absolute and average errors for the Genetic Programming formulas are the average pricing error for all options which is again averaged across the ten Genetic Program runs. Results are presented for all six parent-selection methods

considered. The modified Black-Scholes equation has the largest errors compared to all other models. The linear models give very good results when we include the naïve Black-Scholes model as an independent variable with the two-stage linear model giving the lowest errors. Among the Genetic Programming formulas, the Fitness-overselection parent selection method provides the smallest absolute pricing error and one of the smaller percentage pricing errors. The magnitudes are comparable to the two-stage linear model with the Black-Scholes as an independent variable.

**Insert Table V Here**

The linear models that have the Black-Scholes model as an independent variable, however, have one major draw back -- the partial derivatives of the pricing equation are equal to the Black-Scholes partial derivatives with a constant adjustment term. The true test of any option pricing model is its performance in hedging and the constant adjustment to the option price sensitivity with respect to the stock price, the option *delta*, will not work in practice. Genetic Programming allows general adjustments to the option pricing model and is not subject to this problem. Note that if the linear model is indeed the best model, the Genetic Program theoretically should be able to find it and no generality is lost.

We test the hedging effectiveness of the Genetic Program formula by constructing a hedge portfolio of the option, stock, and a riskless bond. The amount of stocks in the portfolio is chosen as usual to be the *delta* amount, where delta is determined by taking the first partial of the Genetic Programming formula with respect to the stock price. We estimate the performance of the hedge over ten samples of 100 paths for options of varying maturities and strike prices. The hedging performance in each path is calculated to be the deviation from zero in the portfolio value. We also similarly determine the hedging performance of the Black-Scholes model over the same 100 paths.

**Insert Tables VI Here**

Table VI reports the hedging effectiveness of the Genetic Program (with Fitness-overselection) by calculating the fraction of the 100 price paths in which it does better than Black-Scholes in hedging the option. Data is presented for option maturities of 1, 3, and 6 months and for strike prices of 40, 50, and 60 assuming an initial stock price of 50. The first number in each column is the average fraction of time Genetic Programming beats the Black-Scholes model over ten data sets. The next three numbers present statistics of the distribution of this measure by giving the standard error, minimum, and maximum across ten test sets. Genetic

Programming formulas do especially well for in-the-money options. Overall, the Genetic Program beats the Black-Scholes model in over 50% of the cases.

**Insert Tables VII Here**

We further evaluate the performance of the Genetic Programming formula by comparing its pricing errors with that of the Black-Scholes model and Neural Networks[9] for options of various maturities and moneyness. The details of the Neural Networks we use are reported in Appendix E. Of the various normalization and initialization schemes considered, results are reported for the best neural network, i.e. the one that gives the lowest average absolute pricing errors.

Table VII and appendix C report the absolute pricing errors and the percentage pricing errors for the Genetic Programming formula developed with Fitness-overselection, for the Black-Scholes model, and for the best Neural Network, on an out-of-sample two-dimensional grid. Each cell in the table represents the average across ten out-of-sample test data sets and the value in each cell is the average over 5 options. The Genetic Programming formula tends to do better with in-the-money and short-maturity options whereas the Black-Scholes model seems to perform relatively better with out-of-the money and long-term options. This result is consistent with the notion that the jump term influences prices of short-maturity in-the-money options more, relative to long-term and out-of-the-money options. We found that Genetic Programming beats Neural Networks in all cases.

Figure 5 and appendix D plot the absolute pricing errors and percentage pricing errors for the Black-Scholes equation and the Genetic Programming formulas. An interesting observation is that the large percentage pricing errors observed for Genetic Programming formula approximations to the Black-Scholes model as seen in Figure 3 are not observed in this case. This is because the Black-Scholes model itself is incorporated into the Genetic Programming formulas, as shown in the equation above, which improves the performance of the Genetic Programming formulas in this region. The deviations for the low-priced, short-maturity, out-of-the-money options are thus lower.

**Insert Figure 5 Here**

[9] We chose Neural Networks for benchmarking our results as it is the closest to genetic programming in its philosophy and is also a data-driven non-parametric methodology. Other option pricing methods such as GARCH models require assumptions of the underlying process and parameter values.

To take advantage of Genetic Programming's ability to learn with small training sets (Koza 1992) and reduce computational time, we tested its performance using random samples of 5% and 25% of the options generated in the simulation. We found that the training formulas with the smaller data sets resulted in only a minimal reduction in out-of-sample performance. Our tests dramatically support the notion that Genetic Programming needs only small training sets in order to arrive at a good solution.

## IV. Applications in the Real world

In this section, we apply Genetic Programming to price real-world options data. Call options data for the S&P 500 Index and 5 different stocks were obtained from the Berkeley Options Data Base (BODB). BODB's data is time stamped to the nearest second and ensures a good match between the values of an option and its underlying asset. Raw BODB records are screened as follows. We do not include records from the first 2,500 seconds after 8:30 am or in the last 2,500 seconds before 3:00 pm and required at least 300 seconds within a 1% deviation for the underlying index/equity price. We also reject data when the option bid-ask spread is more than ¼ or 5% of the option value. The first restriction eliminates artificial pricing that may occur due to the structure of the market at the beginning and the end of the day. The second restriction is to allow the options market to adjust to changes in underlying asset value.[10] The third gives us a tighter handle on the option's equilibrium price.

Option prices are set to be the average of the bid and ask prices. We calculate the risk-free rate between two calendar dates using the Nelson-Siegel-Bliss term structure model. The option's time to maturity is set to be the number of trading days between the trade date and the expiration date of the option. We use a two step approach to develop the Genetic Program for real-world options. We first determine the optimal set of algorithm parameters using a training and validation step. We vary algorithm parameters for the genetic programs when training them on a subset of options price data. In each iteration, we test the performance of the genetic program on a validation data set of options prices from a later date. The algorithm parameters that give the best results are then used in the next step where the genetic program is developed on a separate training data set and then tested on an out-of-sample test set of options prices from a later date. This is the training/test step.

[10] Stephan and Whaley (1990) find that stock price changes lead option price changes. Finucane (1991) finds similar results for S&P 100 index options.

Ten Genetic Programming formulas were developed using ten training/validation data sets and ten training/test data sets. The training/validation data sets were created by randomly sampling April 3-4, 1995 screened S&P 500 Index Option data.[11] These data sets are used to evaluate algorithm parameters adopted to implement the genetic program. The training/test data sets are separately created from screened April 6-10, 1995 screened S&P 500 Index Options data. All out-of-sample validation and test data occurred later in time than the training data. Training sets contained a mere 50 points each and training times do not exceed 3 minutes per formula.

**Insert Table VIII Here**

The sets of operations, functions and variables allowed in our formulas are those used in the jump-diffusion world, augmented by the risk-free rate and historical volatility. They are presented in Table VIII. As in Hutchinson, Lo, and Poggio (1994), we estimate the S&P 500 Index volatility by computing the standard deviation of the 60 most recent continuously compounded daily S&P 500 returns using 3.00pm (CST) prices. We adjusted for dividends by subtracting the present value of actual dividends between the record date $t_0$ and the option maturity date T:

$$S(t_0)^* = S(t_0) - \sum_{t=t_0}^{T} D(t)e^{-r(t_0,t)(t-t_0)} \tag{10}$$

The formulas generated by the genetic program for the index options were adaptations of the Black-Scholes model. For example, one of the runs resulted in the formula,

$$C(S, X, \tau) = C_{Black\ -\ Scholes} + 3\tau \tag{11}$$

where, $C_{\text{Black-Scholes}}$ is the Black-Scholes formula and $\tau$ is the time to maturity.

Table IX presents the average absolute pricing error for the 10 Genetic Programming formulas, the Black-Scholes model, and the best neural network trained on the data. The difference in pricing errors is also presented. The out-of-sample performance of these Genetic Programming adaptations of Black-Scholes model is remarkable: 9 of the10 Genetic Programming formulas beat the Black-Scholes model in both the average absolute pricing errors and the average percentage pricing errors.

**Insert Table IX Here**

---

[11] We picked April 3 at random. The data sampled for parameter search spanned April 3, 9:11 a.m. to April 4, 9:16 a.m.

Since the Black-Scholes model is a component of our Genetic Programming model and the Black-Scholes model appears in every resulting formula, one can view our model as an adaptation of the Black-Scholes model to the trading environment. The idea is similar to the control variate method for controlling the variance of errors in Monte Carlo simulation.

For testing the performance of Genetic Programming in pricing equity options, we choose five stocks that had options volume of at least 1500 contracts and which never paid cash dividends. The stocks are: Best Buy Company Inc., Broderbund Software Inc., CompUsa Inc., Digital Equipment Corporation, and Novellus Systems Inc. For each stock, we develop ten Genetic Programming formulas. The training/validation data sets are constructed using BODB records for April 3-4, 1995. The training/test data sets are constructed from options traded during the period April 6-13, 1995.

The formulas generated by the genetic program for the equity options were also adaptations of the Black-Scholes model. In most cases, the formulas were of the form,

$$C(S, X, \tau) = C_{Black-Scholes} + \text{Constant} * \tau \qquad (12)$$

where, $C_{\text{Black-Scholes}}$ is the Black-Scholes formula and $\tau$ is the time to maturity. The constant takes values from 1 to 4 depending on the stock underlying the option that is being priced.

Table IX also presents the average absolute pricing errors for the ten Genetic Programming formulas, the Black Scholes formula, and for the best neural networks. When there is no difference between the errors for the Genetic Programming formula and the Black-Scholes model, it indicates that the genetic program converged on the Black-Scholes model.

Except for Best Buy, the Genetic Programming method produced formulas that outperform the Black-Scholes model on average, though the results are not as strong as the case of S&P Index options. We attribute the results for Best Buy to the fact that the data set used for the parameter search is much smaller than the data sets used for the other four stocks. The resulting training and validation sets were thus not independent enough to yield an insight on satisfactory parameters.

The advantages of Genetic Programming over neural networks when dealing with small data sets is, however, dramatically illustrated. Equity options are more thinly traded and there is less data available to train neural networks and genetic programs in comparison to the S&P 500 index option. Neural Network pricing errors are lower than those of the Genetic Programming model for the S&P 500 index option. On the other hand, Neural Network pricing errors are a

higher than that of the Genetic Programming model for four out five equity options. In the remaining case (Best Buy), the magnitude of the errors of the Genetic Programming model is only marginally higher than those for the Black-Scholes model and Neural Networks.

Note that for Broderbund and DEC, a majority of the genetic programs converge on the Black-Scholes model as the best possible pricing formula. This highlights the advantage of the Genetic Programming approach -- it can easily converge on existing known models, if they are indeed the best solutions. By including known analytical solutions in the parameter set, we thus increase the efficiency of Genetic Programming by using it to improve on existing solutions.

## V. Conclusion

In this paper we have developed a procedure to apply the principles of Genetic Programming to option pricing. Our results, from controlled simulations and real world data, are strongly encouraging and suggest that the Genetic Programming approach works well in practice. The Genetic Programming method has many advantages over other numerical techniques. First, it is a non-parametric data driven approach and requires minimal assumptions. We thus avoid the problems associated with making specific assumptions regarding the stock price process. Many researchers attribute the systematic biases in Black-Scholes prices to the assumption that returns are normally distributed and have developed extensions by considering other stock price processes. However, no single model explains all of the Black-Scholes biases and closed form solutions are elusive (Rubinstein (1997)). The Genetic Programming method uses options price data and extracts the implied pricing equation directly.

We show that Genetic Programming formulas beat the Black-Scholes model in 10 out of 10 cases in a simulation study where the underlying stock prices were generated using a jump diffusion process. They work almost as well in pricing S&P Index options with genetic programs beating the Black-Scholes model in 9 out of 10 cases. The results for five equity options are not as strong, perhaps because we use very small data sets, and genetic programs beat or match the Black-Scholes model for 4 of the 5 stocks considered.

Second, the Genetic Programming method requires less data than other numerical techniques such as Neural Networks (Hutchinson, Lo, and Poggio (1994)). We show this by simulation studies that use smaller subsets of the data and by using both genetic programs and neural networks to price relatively thinly traded equity options. Indeed, in some cases the programs are run on as few as 50 data points. We show that Genetic Programs have much better

results than Neural Networks for four of the five equity options we consider, all of which have small data sets. The time required to train and develop the genetic programming formulas is also relatively short.

Third, the Genetic Programming method can incorporate known analytical approximations in the solution method. For example, we use the Black-Scholes model as a parameter in the genetic program to build the option pricing model. The final solution can then be considered to be an adaptation of the Black-Scholes model to conditions that violate the underlying assumptions. The flexibility in adding terms to the parameter set used to develop the functional approximation can also be used to examine whether factors beyond those used in this study, for example, trading volume, skewness and kurtosis of returns, and inflation, are relevant to option pricing. The self-learning and self-improving feature also makes the method robust to changes in the economic environment. Finally, since the Genetic Programming method is fast, self-learning, and self-improving, it is an ideal too for practitioners.

**REFERENCES**


Allen, F. and Karjalainen, R., 1998 "Using Genetic Algorithms to find technical trading rules," *Journal of Financial Economics*, Forthcoming.

Amin, L. A. and Jarrow, R.A., 1992, "Pricing options on risky assets in a stochastic interest rate economy." *Mathematical Finance*, Vol. 2.

Ball, C.A. and Torous, W.N., 1985 "On jumps in common stock prices and their impact on call option pricing." *Journal of Finance*, Vol. 40 (March).

Ballie R. and DeGennaro, R., 1990 "Stock returns and volatility." *Journal of Financial and Quantitative Analysis*, Vol. 25 (June).

Black, F. and Scholes, M., 1972 "The valuation of option contracts and a test of market efficiency." *Journal of Finance*, Vol. 27 (May).

Black, F. and Scholes, M., 1973 "The pricing of options and corporate liabilities." *Journal of Political Economy*, Vol. 81.

Bollerslev T., 1986, "Generalized Autoregressive conditional Heteroskedasticity." *Journal of Econometrics*, Vol. 31 (April).

Chance, D. M., 1986,"Empirical tests of the pricing of index call options," *Advances in Futures and Options Research*, Vol. 1.

Chidambaran, N. K. and S. Figlewski, 1995, "Streamlining Monte Carlo Simulation with the Quasi-Analytic Method: Analysis of a Path-Dependent Option Strategy," *Journal of Derivatives*, Winter.

Fama, E.F., 1965, "The behavior of stock market prices." *Journal of Business*, Vol. 38 (January).

French, K. R., Schwert, G.W., and Stambaugh, R.F., 1987, "Expected stock returns and volatility." *Journal of Financial Economics*, Vol. 19 (September).

Galai, D., 1977, "Tests of market efficiency of the Chicago Board of Options Exchange." *Journal of Business*, Vol. 50.

Heston, S., 1993, "Invisible parameters in option prices." *Journal of Finance*, Vol. 48 (July)

Ho, T. H., 1996, "Finite automata play repeated prisoner's dilemma with information processing costs." *Journal of Economic Dynamics and Control*, Vol. 20 (January-March)

Holland, J. H. 1975, *Adaptation in natural and artificial systems*, The University of Michigan Press, Ann Arbor.

Hull, J., 1993, *Options, Futures, and Other Derivative Securities*, 2nd Ed., (Prentice-Hall, Englewood Cliffs, New Jersey).

Hutchinson, J., Lo A., and Poggio, T., 1994, "A Nonparametric approach to the Pricing and Hedging of Derivative Securities Via Learning Networks," *Journal of Finance*, Vol. 49. (June).

Kim, D. and Kon, S.J., 1994, "Alternative models for the conditional heteroscedasticity of stock returns." *The Journal of Business*, Vol. 67 (October).

Koza, J. R., 1992, *Genetic Programming*, (MIT Press, Cambridge, Massachusetts).

Lettau, M., 1997, "Explaining the facts with adaptive agents." *Journal of Economic Dynamics and Control*, Vol. 21.

Macbeth, J. D. and Merville, L. J., 1979, "An empirical estimation of the Black-Scholes call option pricing model." *Journal of Finance*, Vol. 34 (December).

Macbeth, J. D. and Merville, L. J., 1980, "Tests of the Black-Scholes and Cox call option valuation models" *Journal of Finance*, Vol. 35 (May).

Marimon, R., McGrattan, E., Sargent, T.J., 1990, "Money as a medium of exchange in an economy with artificially intelligent agents." *Journal of Economic Dynamics and Control*, Vol. 14.

Merton, R.C., 1973, "Theory of rational option pricing." *Bell Journal of Economics and Management Science*, Spring.

Merton, R.C., 1976, "Option pricing when underlying stock returns are discontinuous." *Journal of Financial Economics*, Vol. 3 (January-March).

Neely, C., P. Weller, and R. Dittmar, 1997, "Is Technical Analysis in the Foreign Exchange Market Profitable? A Genetic Programming Approach," *Journal of Financial and Quantitative Analysis*, Vol. 32(4), pp.405-426.

Rubinstein, M., 1985,"Nonparametric Tests of Alternative Option Pricing Models." *Journal of Finance*, Vol. 40 (June).

Rubinstein, M., 1997, "Implied Binomial Trees", *Journal of Finance*, Vol. 49.

Stephan, J. A., and Whaley, R. E., 1990, "Intraday price change and trading volume relations in the stock and options markets." *Journal of Finance*, Vol. 45 (March).

Trigueros, J. 1997, "A Nonparametric Approach to Pricing and Hedging Derivative Securities Via Genetic Regression," *Proceedings of the Conference on Computational Intelligence for Financial Engineering*, March.

**Table I**
**Genetic Programming Model Specification in a Black-Scholes World**

**Panel A: Training Variables**

| Name | Source | Definition |
|---|---|---|
| S | Option Contract | Stock price |
| X | Option Contract | Exercise price |
| S/X | Part of Black-Scholes | Degree of option moneyness |
| □ | Option Contract | Time to maturity (in years) |
| max(S-X) | Boundary Condition | Option intrinsic value Max (S-X,0) |
| + | Standard arithmetic | Addition |
| - | Standard arithmetic | Subtraction |
| * | Standard arithmetic | Multiplication |
| % | Standard arithmetic | Protected Division: x%y = 1 if y = 0<br>= x/y otherwise |
| Exp | Black-Scholes component | Exponent: $\exp(x) = e^x$ |
| Plog | Black-Scholes component | Protected Natural log: plog(x) = ln(\|x\|) |
| Psqrt | Black-Scholes component | Protected Square root: psqrt(x) = sqrt(\|x\|) |
| Ncdf | Black-Scholes component | Normal Cumulative Distribution Function |

**Panel B: Size of Training Sets**

For each training set (i.e. a Genetic Programming option pricing formula), the price path of a stock with starting value $S_0$=50 was simulated through 24 21-day months. Options were created according to CBOE rules and valued using the Black Scholes formula. Each training set consisted of the daily values of these options. Formula populations were exposed to dynamically sampled subsets of these training sets.

| Training Set | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Data Points | 6037 | 5897 | 5609 | 6069 | 5847 | 6633 | 5980 | 5615 | 5975 | 6741 |

**Panel C: Genetic Algorithm Training Parameters**

The following Genetic Programming algorithm parameters were used for the training step in the Black-Scholes world.

| | |
|---|---|
| Fitness Criterion | Sum of absolute dollar errors and percentage errors |
| Population Size | 25,000 |
| Number of Generations | 51 |

**Table II**
**Performance of Genetic Programming in a Black-Scholes World**
**-- Measured by Mean Absolute Pricing Errors--**

We generate the underlying stock price to follow Geometric Brownian motion. The model specification for Genetic Programming are specified in Table I. Pricing errors are presented for six parent-selection algorithms to evaluate the efficiency of alternate methods for generating new populations from the previous generation of formulas. Each cell in the table is the average value across 10 Genetic Programming formulas generated for each of the alternate methods. For each programming formula, the error is calculated by taking the average pricing error over five options. Rows in the table represent days-to-maturity and columns represent the degree-of-moneyness, S/X.

Parent Selection Criteria : **Best**
Individuals with the smallest pricing errors are selected to be in the new population.

| | | **Moneyness S/X** | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **0.875** | **0.9** | **0.925** | **0.95** | **0.975** | **1** | **1.025** | **1.05** | **1.075** | **1.1** | **1.125** | **1.15** |
| **MATURITY** | **5** | 0.068 | 0.046 | 0.034 | 0.034 | 0.101 | 0.479 | 0.112 | 0.038 | 0.056 | 0.070 | 0.083 | 0.097 |
| | **10** | 0.063 | 0.040 | 0.025 | 0.039 | 0.203 | 0.655 | 0.242 | 0.047 | 0.051 | 0.075 | 0.091 | 0.104 |
| | **30** | 0.079 | 0.070 | 0.068 | 0.180 | 0.493 | 1.137 | 0.543 | 0.257 | 0.092 | 0.075 | 0.115 | 0.145 |
| | **45** | 0.153 | 0.103 | 0.118 | 0.269 | 0.603 | 1.376 | 0.656 | 0.369 | 0.173 | 0.110 | 0.135 | 0.176 |
| | **60** | 0.208 | 0.138 | 0.172 | 0.325 | 0.652 | 1.551 | 0.718 | 0.443 | 0.244 | 0.160 | 0.170 | 0.211 |
| | **90** | 0.320 | 0.239 | 0.257 | 0.386 | 0.679 | 1.806 | 0.768 | 0.534 | 0.356 | 0.268 | 0.269 | 0.310 |

Parent Selection Criteria : **Fitness**
Individuals are chosen randomly with a probability that is inversely proportional to their pricing errors.

| | | **Moneyness S/X** | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **0.875** | **0.9** | **0.925** | **0.95** | **0.975** | **1** | **1.025** | **1.05** | **1.075** | **1.1** | **1.125** | **1.15** |
| **MATURITY** | **5** | 0.038 | 0.032 | 0.026 | 0.015 | 0.133 | 0.565 | 0.139 | 0.030 | 0.030 | 0.038 | 0.046 | 0.054 |
| | **10** | 0.059 | 0.052 | 0.037 | 0.054 | 0.282 | 0.782 | 0.290 | 0.091 | 0.030 | 0.038 | 0.046 | 0.054 |
| | **30** | 0.162 | 0.107 | 0.129 | 0.303 | 0.708 | 1.298 | 0.688 | 0.397 | 0.208 | 0.118 | 0.088 | 0.091 |
| | **45** | 0.209 | 0.155 | 0.238 | 0.474 | 0.935 | 1.555 | 0.882 | 0.581 | 0.357 | 0.228 | 0.163 | 0.138 |
| | **60** | 0.242 | 0.221 | 0.347 | 0.626 | 1.119 | 1.761 | 1.033 | 0.733 | 0.492 | 0.339 | 0.252 | 0.204 |
| | **90** | 0.342 | 0.368 | 0.553 | 0.882 | 1.410 | 2.083 | 1.265 | 0.975 | 0.722 | 0.545 | 0.429 | 0.354 |

Parent Selection Criteria : **Fitness-overselection**
Individuals are divided into two groups. Group 1 has the top 320 individual with the smallest pricing errors. The remainders are placed in Group 2. Individuals are then chosen randomly with a higher probability assigned to Group 1. In our implementation, the probability of selection was 80% for Group 1 individuals and 20% for Group 2 individuals.

| | | **Moneyness S/X** | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **0.875** | **0.9** | **0.925** | **0.95** | **0.975** | **1** | **1.025** | **1.05** | **1.075** | **1.1** | **1.125** | **1.15** |
| **MATURITY** | **5** | 0.094 | 0.06 | 0.028 | 0.019 | 0.081 | 0.414 | 0.13 | 0.129 | 0.136 | 0.129 | 0.129 | 0.141 |
| | **10** | 0.064 | 0.03 | 0.015 | 0.042 | 0.175 | 0.586 | 0.15 | 0.106 | 0.137 | 0.142 | 0.141 | 0.149 |
| | **30** | 0.049 | 0.071 | 0.070 | 0.161 | 0.432 | 0.922 | 0.399 | 0.201 | 0.127 | 0.145 | 0.172 | 0.188 |
| | **45** | 0.119 | 0.102 | 0.109 | 0.233 | 0.537 | 1.038 | 0.544 | 0.313 | 0.188 | 0.171 | 0.194 | 0.216 |
| | **60** | 0.162 | 0.128 | 0.144 | 0.283 | 0.604 | 1.103 | 0.645 | 0.403 | 0.257 | 0.222 | 0.236 | 0.259 |
| | **90** | 0.246 | 0.201 | 0.214 | 0.345 | 0.672 | 1.159 | 0.759 | 0.530 | 0.383 | 0.350 | 0.361 | 0.382 |

**Table II (continued)**
**Performance of Genetic Programming in a Black-Scholes World**
**-- Measured by Mean Absolute Pricing Errors--**

Parent Selection Criteria : **Random**
Individuals are chosen randomly and their fitness errors are ignored.

| | | **Moneyness S/X** | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **0.875** | **0.9** | **0.925** | **0.95** | **0.975** | **1** | **1.025** | **1.05** | **1.075** | **1.1** | **1.125** | **1.15** |
| **MATURITY** | **5** | 0.069 | 0.061 | 0.053 | 0.046 | 0.139 | 0.570 | 0.166 | 0.048 | 0.042 | 0.045 | 0.047 | 0.050 |
| | **10** | 0.077 | 0.068 | 0.061 | 0.068 | 0.287 | 0.785 | 0.336 | 0.124 | 0.060 | 0.054 | 0.056 | 0.060 |
| | **30** | 0.159 | 0.133 | 0.116 | 0.309 | 0.717 | 1.304 | 0.819 | 0.490 | 0.284 | 0.177 | 0.138 | 0.125 |
| | **45** | 0.215 | 0.164 | 0.197 | 0.472 | 0.940 | 1.557 | 1.069 | 0.711 | 0.461 | 0.305 | 0.225 | 0.188 |
| | **60** | 0.259 | 0.181 | 0.276 | 0.604 | 1.112 | 1.750 | 1.263 | 0.890 | 0.614 | 0.425 | 0.311 | 0.251 |
| | **90** | 0.336 | 0.228 | 0.397 | 0.793 | 1.352 | 2.020 | 1.540 | 1.153 | 0.848 | 0.621 | 0.463 | 0.363 |

Parent Selection Criteria : **Tournament, n=4**
Four individuals are first chosen randomly from the population. The best of the four individuals is then selected for the next generation.

| | | **Moneyness S/X** | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **0.875** | **0.9** | **0.925** | **0.95** | **0.975** | **1** | **1.025** | **1.05** | **1.075** | **1.1** | **1.125** | **1.15** |
| **MATURITY** | **5** | 0.035 | 0.023 | 0.016 | 0.030 | 0.088 | 0.455 | 0.194 | 0.171 | 0.127 | 0.103 | 0.106 | 0.123 |
| | **10** | 0.033 | 0.022 | 0.017 | 0.046 | 0.164 | 0.625 | 0.167 | 0.144 | 0.137 | 0.123 | 0.125 | 0.138 |
| | **30** | 0.039 | 0.033 | 0.049 | 0.165 | 0.401 | 1.068 | 0.297 | 0.161 | 0.132 | 0.150 | 0.174 | 0.191 |
| | **45** | 0.080 | 0.068 | 0.089 | 0.234 | 0.494 | 1.242 | 0.387 | 0.236 | 0.169 | 0.172 | 0.197 | 0.225 |
| | **60** | 0.131 | 0.111 | 0.122 | 0.274 | 0.544 | 1.344 | 0.449 | 0.301 | 0.221 | 0.204 | 0.228 | 0.265 |
| | **90** | 0.258 | 0.218 | 0.192 | 0.308 | 0.574 | 1.445 | 0.524 | 0.404 | 0.318 | 0.287 | 0.307 | 0.356 |

Parent Selection Criteria : **Tournament, n=7**
Seven individuals are first chosen randomly from the population. The best of the seven individuals is then selected for the next generation.

| | | **Moneyness S/X** | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **0.875** | **0.9** | **0.925** | **0.95** | **0.975** | **1** | **1.025** | **1.05** | **1.075** | **1.1** | **1.125** | **1.15** |
| **MATURITY** | **5** | 0.030 | 0.015 | 0.012 | 0.008 | 0.064 | 0.441 | 0.126 | 0.092 | 0.067 | 0.043 | 0.038 | 0.039 |
| | **10** | 0.023 | 0.013 | 0.009 | 0.026 | 0.132 | 0.589 | 0.175 | 0.099 | 0.075 | 0.052 | 0.044 | 0.044 |
| | **30** | 0.023 | 0.032 | 0.055 | 0.115 | 0.256 | 0.969 | 0.287 | 0.203 | 0.133 | 0.088 | 0.070 | 0.071 |
| | **45** | 0.057 | 0.060 | 0.105 | 0.161 | 0.286 | 1.160 | 0.327 | 0.282 | 0.213 | 0.142 | 0.103 | 0.095 |
| | **60** | 0.090 | 0.089 | 0.153 | 0.200 | 0.305 | 1.310 | 0.350 | 0.343 | 0.286 | 0.211 | 0.150 | 0.125 |
| | **90** | 0.157 | 0.142 | 0.238 | 0.278 | 0.322 | 1.525 | 0.379 | 0.429 | 0.399 | 0.347 | 0.267 | 0.208 |

**Table III**
**Genetic Programming Model Specification in a Jump-Diffusion World**

**Panel A: Training Variables**

| Name | Source | Definition |
|---|---|---|
| S | Option Contract | Stock price |
| X | Option Contract | Exercise price |
| S/X | Black-Scholes Parameter | Degree of option moneyness |
| □ | Option Contract | Time to maturity (in years) |
| Max(S-X) | Boundary Condition | Option intrinsic value Max (S-X,0) |
| Black-Scholes | Naïve investor's valuation | Black Scholes value of option |
| + | Standard arithmetic | Addition |
| - | Standard arithmetic | Subtraction |
| * | Standard arithmetic | Multiplication |
| % | Standard arithmetic | Protected Division: x%y = 1 , if y = 0<br>= x/y , otherwise |
| Exp | Black-Scholes component | Exponent: $\exp(x) = e^x$ |
| Plog | Black-Scholes component | Protected Natural log: plog(x) = ln(\|x\|) |
| Psqrt | Black-Scholes component | Protected Square root: psqrt(x) = sqrt(\|x\|) |
| Ncdf | Black-Scholes component | Normal Cumulative Distribution Function |

**Panel B: Size of Training Sets**

For each training set (option pricing formula), the price path of a stock with beginning value $S_0 = 50$ was simulated through 24 21-day months. Options were created according to CBOE rules and valued using the Black Scholes formula. Each training set consisted of the daily values of these options.

| Training Set | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Data Points | 311 | 350 | 364 | 308 | 288 | 420 | 318 | 387 | 409 | 319 |

**Panel C: Training Parameters**

Genetic Programming algorithm training parameters used in the non-Black-Scholes world where stock prices are generated by a jump diffusion process.

| | |
|---|---|
| Fitness Criterion | Sum of absolute dollar and percentage errors |
| Population Size | 5,000 |
| Number of Generations | 10 |

**Table IV**
**Performances of the Genetic Programming Model and the Black-Scholes Model in a Jump-Diffusion World**

We generate the underlying stock price as a jump-diffusion process. The model specifications for Genetic Programming are specified in Table IV. Pricing errors are presented for six Genetic Programming algorithms that use alternate methods for generating new populations from the previous generation. Each cell in the table presents the average pricing-error over the entire sample of options generated in each sample set.

**Parent Selection Criteria: Best**

| | | Mean Absolute Error | | | Mean Percentage Error | | |
|---|---|---|---|---|---|---|---|
| Data Set | Average Size | Genetic Programming | Black-Scholes | Best Neural Network | Genetic Programming | Black-Scholes | Best Neural Network |
| Training Average | 347.4 | 0.0626 | 0.0875 | 0.5473 | 10.0% | 16.42% | 34221130.0% |
| GP 1 | 1080 | 0.0675 | 0.0888 | 0.2042 | 4.3% | 7.5% | 20.4% |
| GP 2 | 1080 | 0.0789 | 0.0888 | 0.0971 | 4.9% | 7.5% | 9.7% |
| GP 3 | 1080 | 0.0635 | 0.0888 | 0.2771 | 3.9% | 7.5% | 27.7% |
| GP 4 | 1080 | 0.0639 | 0.0888 | 0.0927 | 3.9% | 7.5% | 9.3% |
| GP 5 | 1080 | 0.0637 | 0.0888 | 0.0863 | 2.5% | 7.5% | 8.6% |
| GP 6 | 1080 | 0.0641 | 0.0888 | 0.1320 | 4.1% | 7.5% | 13.2% |
| GP 7 | 1080 | 0.0487 | 0.0888 | 0.0845 | 3.7% | 7.5% | 8.4% |
| GP 8 | 1080 | 0.0700 | 0.0888 | 0.6365 | 5.1% | 7.5% | 63.7% |
| GP 9 | 1080 | 0.0745 | 0.0888 | 0.1142 | 5.4% | 7.5% | 11.4% |
| GP 10 | 1080 | 0.0600 | 0.0888 | 0.0897 | 2.4% | 7.5% | 9.0% |
| Test Average | 1080 | 0.0655 | 0.0888 | 0.1814 | 4.0% | 7.47% | 17.1% |

**Parent Selection Criteria: Fitness**

| | | Mean Absolute Error | | | Mean Percentage Error | | |
|---|---|---|---|---|---|---|---|
| Data Set | Average Size | Genetic Programming | Black-Scholes | Best Neural Network | Genetic Programming | Black-Scholes | Best Neural Network |
| Training Average | 347.4 | 0.03753 | 0.08752 | 0.5473 | 1192610.0% | 16.4% | 34221130.0% |
| GP 1 | 1080 | 0.05401 | 0.0888 | 0.2042 | 3.2% | 7.5% | 20.4% |
| GP 2 | 1080 | 0.0400 | 0.0888 | 0.0971 | 3.1% | 7.5% | 9.7% |
| GP 3 | 1080 | 0.02415 | 0.0888 | 0.2771 | 2.5% | 7.5% | 27.7% |
| GP 4 | 1080 | 0.04673 | 0.0888 | 0.0927 | 2.9% | 7.5% | 9.3% |
| GP 5 | 1080 | 0.03602 | 0.0888 | 0.0863 | 2.2% | 7.5% | 8.6% |
| GP 6 | 1080 | 0.04183 | 0.0888 | 0.1320 | 2.9% | 7.5% | 13.2% |
| GP 7 | 1080 | 0.03476 | 0.0888 | 0.0845 | 2.3% | 7.5% | 8.4% |
| GP 8 | 1080 | 0.04558 | 0.0888 | 0.6365 | 2.8% | 7.5% | 63.7% |
| GP 9 | 1080 | 0.03499 | 0.0888 | 0.1142 | 2.4% | 7.5% | 11.4% |
| GP 10 | 1080 | 0.0346 | 0.0888 | 0.0897 | 2.8% | 7.5% | 9.0% |
| Test Average | 1080 | 0.03927 | 0.0888 | 0.1814 | 2.7% | 7.5% | 17.1% |

**Table IV (continued)**

**Parent Selection Criteria: Fitness-overselection**

| | | Mean Absolute Error | | | Mean Percentage Error | | |
|---|---|---|---|---|---|---|---|
| Data Set | Average Size | Genetic Programming | Black-Scholes | Best Neural Network | Genetic Programming | Black-Scholes | Best Neural Network |
| Training Average | 347.4 | 0.0375 | 0.0875 | 0.5473 | 1,192,618% | 16.42% | 34221130.0% |
| GP 1 | 1080 | 0.0540 | 0.0888 | 0.2042 | 3.20% | 7.47% | 20.4% |
| GP 2 | 1080 | 0.0400 | 0.0888 | 0.0971 | 3.12% | 7.47% | 9.7% |
| GP 3 | 1080 | 0.0242 | 0.0888 | 0.2771 | 2.48% | 7.47% | 27.7% |
| GP 4 | 1080 | 0.0467 | 0.0888 | 0.0927 | 2.86% | 7.47% | 9.3% |
| GP 5 | 1080 | 0.0360 | 0.0888 | 0.0863 | 2.23% | 7.47% | 8.6% |
| GP 6 | 1080 | 0.0418 | 0.0888 | 0.1320 | 2.90% | 7.47% | 13.2% |
| GP 7 | 1080 | 0.0348 | 0.0888 | 0.0845 | 2.34% | 7.47% | 8.4% |
| GP 8 | 1080 | 0.0456 | 0.0888 | 0.6365 | 2.81% | 7.47% | 63.7% |
| GP 9 | 1080 | 0.0350 | 0.0888 | 0.1142 | 2.45% | 7.47% | 11.4% |
| GP 10 | 1080 | 0.0346 | 0.0888 | 0.0897 | 2.79% | 7.47% | 9.0% |
| Test Average | 1080 | 0.0393 | 0.0888 | 0.1814 | 2.72% | 7.47% | 17.1% |

**Parent Selection Criteria: Random**

| | | Mean Absolute Error | | | Mean Percentage Error | | |
|---|---|---|---|---|---|---|---|
| Data Set | Average Size | Genetic Programming | Black-Scholes | Best Neural Network | Genetic Programming | Black-Scholes | Best Neural Network |
| Training Average | 347.4 | 0.0709 | 0.0875 | 0.5473 | 47903300.0% | 16.42% | 34221130.0% |
| GP 1 | 1080 | 0.0716 | 0.0888 | 0.2042 | 5.2% | 7.5% | 20.4% |
| GP 2 | 1080 | 0.0796 | 0.0888 | 0.0971 | 6.1% | 7.5% | 9.7% |
| GP 3 | 1080 | 0.0854 | 0.0888 | 0.2771 | 7.3% | 7.5% | 27.7% |
| GP 4 | 1080 | 0.0700 | 0.0888 | 0.0927 | 5.1% | 7.5% | 9.3% |
| GP 5 | 1080 | 0.0676 | 0.0888 | 0.0863 | 5.2% | 7.5% | 8.6% |
| GP 6 | 1080 | 0.0506 | 0.0888 | 0.1320 | 3.8% | 7.5% | 13.2% |
| GP 7 | 1080 | 0.0716 | 0.0888 | 0.0845 | 5.2% | 7.5% | 8.4% |
| GP 8 | 1080 | 0.0506 | 0.0888 | 0.6365 | 3.8% | 7.5% | 63.7% |
| GP 9 | 1080 | 0.0709 | 0.0888 | 0.1142 | 5.4% | 7.5% | 11.4% |
| GP 10 | 1080 | 0.0862 | 0.0888 | 0.0897 | 3.5% | 7.5% | 9.0% |
| Test Average | 1080 | 0.0704 | 0.0888 | 0.1814 | 5.1% | 7.47% | 17.1% |

**Table IV (continued)**

**Parent Selection Criteria: Tournament, Size = 4**

| | | Mean Absolute Error | | | Mean Percentage Error | | |
|---|---|---|---|---|---|---|---|
| Data Set | Average Size | Genetic Programming | Black-Scholes | Best Neural Network | Genetic Programming | Black-Scholes | Best Neural Network |
| Training Average | 347.4 | 0.0527 | 0.0875 | 0.5473 | 8406.4% | 16.42% | 34221130.0% |
| GP 1 | 1080 | 0.0646 | 0.0888 | 0.2042 | 3.9% | 7.5% | 20.4% |
| GP 2 | 1080 | 0.0515 | 0.0888 | 0.0971 | 4.6% | 7.5% | 9.7% |
| GP 3 | 1080 | 0.0564 | 0.0888 | 0.2771 | 3.3% | 7.5% | 27.7% |
| GP 4 | 1080 | 0.0651 | 0.0888 | 0.0927 | 4.0% | 7.5% | 9.3% |
| GP 5 | 1080 | 0.0474 | 0.0888 | 0.0863 | 3.6% | 7.5% | 8.6% |
| GP 6 | 1080 | 0.0545 | 0.0888 | 0.1320 | 2.9% | 7.5% | 13.2% |
| GP 7 | 1080 | 0.0637 | 0.0888 | 0.0845 | 3.8% | 7.5% | 8.4% |
| GP 8 | 1080 | 0.0351 | 0.0888 | 0.6365 | 3.3% | 7.5% | 63.7% |
| GP 9 | 1080 | 0.0574 | 0.0888 | 0.1142 | 5.5% | 7.5% | 11.4% |
| GP 10 | 1080 | 0.0387 | 0.0888 | 0.0897 | 3.2% | 7.5% | 9.0% |
| Test Average | 1080 | 0.0534 | 0.0888 | 0.1814 | 3.8% | 7.47% | 17.1% |

**Parent Selection Criteria: Tournament, Size=7**

| | | Mean Absolute Error | | | Mean Percentage Error | | |
|---|---|---|---|---|---|---|---|
| Data Set | Average Size | Genetic Programming | Black-Scholes | Best Neural Network | Genetic Programming | Black-Scholes | Best Neural Network |
| Training Average | 347.4 | 0.0451 | 0.0875 | 0.5473 | 29234.6% | 16.42% | 34221130.0% |
| GP 1 | 1080 | 0.0578 | 0.0888 | 0.2042 | 3.8% | 7.5% | 20.4% |
| GP 2 | 1080 | 0.0504 | 0.0888 | 0.0971 | 4.4% | 7.5% | 9.7% |
| GP 3 | 1080 | 0.0316‘ | 0.0888 | 0.2771 | 2.8% | 7.5% | 27.7% |
| GP 4 | 1080 | 0.0521 | 0.0888 | 0.0927 | 2.7% | 7.5% | 9.3% |
| GP 5 | 1080 | 0.0514 | 0.0888 | 0.0863 | 2.6% | 7.5% | 8.6% |
| GP 6 | 1080 | 0.0448 | 0.0888 | 0.1320 | 2.6% | 7.5% | 13.2% |
| GP 7 | 1080 | 0.0510 | 0.0888 | 0.0845 | 2.9% | 7.5% | 8.4% |
| GP 8 | 1080 | 0.0436 | 0.0888 | 0.6365 | 3.8% | 7.5% | 63.7% |
| GP 9 | 1080 | 0.0413 | 0.0888 | 0.1142 | 3.7% | 7.5% | 11.4% |
| GP 10 | 1080 | 0.0396 | 0.0888 | 0.0897 | 2.6% | 7.5% | 9.0% |
| Test Average | 1080 | 0.0464 | 0.0888 | 0.1814 | 3.2% | 7.47% | 17.1% |

**Table V**
**Performance of Genetic Programming Model, Black-Scholes Model, and Linear Models in a Jump-Diffusion World**

Pricing errors are presented for six Genetic Programming formulas using alternate methods for generating new populations from the previous generation and for four linear models that are a function of the initial stock price, exercise price, and time to maturity. Each cell in the table presents the average pricing errors over ten sets of stock and option prices and for the entire sample of options generated in each set. Parameter values used to generate stock price and options data and the Genetic Programming parameters are given in Table IV.

| **Average absolute pricing error** | | | | | | |
|---|---|---|---|---|---|---|
| | **Genetic Programming** | **Black-Scholes** | **One stage linear with Black-Scholes** | **One stage linear without Black-Scholes** | **Two stage linear with Black-Scholes** | **Two stage linear without Black-Scholes** |
| **Best** | 0.065481 | 0.088808 | 0.035018 | 0.935721 | 0.015809 | 0.404870 |
| **Fitness** | 0.051669 | 0.088808 | 0.035018 | 0.935721 | 0.015809 | 0.404870 |
| **Fitness-Overselection** | 0.039272 | 0.088808 | 0.035018 | 0.935721 | 0.015809 | 0.404870 |
| **Random** | 0.070398 | 0.088808 | 0.035018 | 0.935721 | 0.015809 | 0.404870 |
| **Tournament, Size = 4** | 0.053441 | 0.088808 | 0.035018 | 0.935721 | 0.015809 | 0.404870 |
| **Tournament, Size = 7** | 0.046351 | 0.088808 | 0.035018 | 0.935721 | 0.015809 | 0.404870 |

| **Average percentage pricing error** | | | | | | |
|---|---|---|---|---|---|---|
| | **Genetic Programming** | **Black-Scholes** | **One stage linear with Black-Scholes** | **One stage linear without Black-Scholes** | **Two stage linear with Black-Scholes** | **Two stage linear without Black-Scholes** |
| **Best** | 4.04% | 7.47% | 3.36% | 86.98% | 2.47% | 34.66% |
| **Fitness** | 3.81% | 7.47% | 3.36% | 86.98% | 2.47% | 34.66% |
| **Fitness-overselection** | 2.72% | 7.47% | 3.36% | 86.98% | 2.47% | 34.66% |
| **Random** | 5.07% | 7.47% | 3.36% | 86.98% | 2.47% | 34.66% |
| **Tournament, Size = 4** | 3.81% | 7.47% | 3.36% | 86.98% | 2.47% | 34.66% |
| **Tournament, Size = 7** | 3.20% | 7.47% | 3.36% | 86.98% | 2.47% | 34.66% |

**Table VI**
**Comparing Hedging Errors**
**for the Genetic Programming Model and the Black-Scholes Model**

This table presents the fraction of 100 price paths over which hedged option portfolios formed with option deltas calculated using Genetic Programming formulas gives a lower error as compared to portfolios where the option deltas are calculated using the Black-Scholes equation. The first number in each column is the mean value across ten sets of 100 price paths. The other numbers present the standard error and the minimum and maximum values over the ten test sets. The initial stock price in all cases is $S_0 = 50$. Hedging errors are calculated for five option strikes (X = 40, 45, 50, 55, and 60) and 3 option maturities (1 month, 3 month, and 6 month). The model specifications for Genetic Programming are given n Table IV.

| | X=40 | X=45 | X=50 | X=55 | X=60 |
|---|---|---|---|---|---|
| | | | | | |
| Time to maturity = 1 month | | | | | |
| Mean | 0.344 | 0.505 | 0.545 | 0.622 | 0.714 |
| (SE) | (0.071) | (0.060) | (0.030) | (0.015) | (0.015) |
| Minimum | 0.080 | 0.040 | 0.310 | 0.540 | 0.670 |
| Maximum | 0.820 | 0.800 | 0.630 | 0.710 | 0.830 |
| | | | | | |
| Time to maturity = 3 months | | | | | |
| Mean | 0.437 | 0.456 | 0.531 | 0.526 | 0.536 |
| (SE) | (0.046) | (0.047) | (0.043) | (0.019) | (0.019) |
| Minimum | 0.150 | 0.140 | 0.190 | 0.430 | 0.450 |
| Maximum | 0.620 | 0.680 | 0.650 | 0.610 | 0.640 |
| | | | | | |
| Time to maturity = 6 months | | | | | |
| Mean | 0.389 | 0.477 | 0.555 | 0.470 | 0.435 |
| (SE) | (0.071) | (0.055) | (0.050) | (0.023) | (0.021) |
| Minimum | 0.060 | 0.130 | 0.220 | 0.320 | 0.300 |
| Maximum | 0.640 | 0.730 | 0.740 | 0.560 | 0.510 |
| | | | | | |
| Overall Mean | 0.503 | | | | |

**Table VII**
**Performance Comparison of Genetic Programming. the Black-Scholes Model, and Neural Networks in a Jump Diffusion World Measured by Mean Absolute Pricing Errors**

The numbers in each cell are the average pricing errors from the models across 10 test sets for the Genetic Programming formulas, the Black-Scholes model, and Neural Networks. For each test set, the error value for each cell is calculated by taking the average pricing errors over five options. Rows in the table represent days-to-maturity and columns represent the degree-of-moneyness, S/X. We shade the areas where Genetic Programming improves Black-Scholes. Genetic Programming is better than neural networks in all cases. The model specification for Genetic Programming is given in Table IV.

| | | | **Moneyness S/X** | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | **0.875** | **0.9** | **0.925** | **0.95** | **0.975** | **1** | **1.025** | **1.05** | **1.075** | **1.1** | **1.125** | **1.15** |
| | | BS= | 0.02 | 0.05 | 0.08 | 0.08 | 0.04 | 0.03 | 0.06 | 0.03 | 0.00 | 0.01 | 0.01 | 0.00 |
| **M** | **5** | GP= | 0.01 | 0.03 | 0.04 | 0.02 | 0.05 | 0.12 | 0.11 | 0.06 | 0.03 | 0.01 | 0.01 | 0.01 |
| | | NN= | 0.23 | 0.23 | 0.21 | 0.18 | 0.14 | 0.12 | 0.16 | 0.21 | 0.21 | 0.23 | 0.26 | 0.26 |
| **A** | | BS= | 0.05 | 0.08 | 0.10 | 0.09 | 0.05 | 0.00 | 0.03 | 0.04 | 0.03 | 0.01 | 0.00 | 0.00 |
| | **10** | GP= | 0.02 | 0.03 | 0.02 | 0.02 | 0.05 | 0.08 | 0.08 | 0.07 | 0.05 | 0.03 | 0.02 | 0.02 |
| **T** | | NN= | 0.19 | 0.17 | 0.12 | 0.08 | 0.07 | 0.12 | 0.18 | 0.22 | 0.21 | 0.22 | 0.25 | 0.26 |
| | | BS= | 0.12 | 0.13 | 0.13 | 0.12 | 0.10 | 0.07 | 0.05 | 0.02 | 0.01 | 0.00 | 0.01 | 0.01 |
| **U** | **30** | GP= | 0.03 | 0.03 | 0.03 | 0.03 | 0.02 | 0.02 | 0.02 | 0.03 | 0.04 | 0.05 | 0.05 | 0.04 |
| | | NN= | 0.05 | 0.09 | 0.14 | 0.17 | 0.17 | 0.17 | 0.18 | 0.19 | 0.20 | 0.21 | 0.24 | 0.26 |
| **R** | | BS= | 0.14 | 0.15 | 0.15 | 0.14 | 0.12 | 0.10 | 0.08 | 0.06 | 0.04 | 0.03 | 0.01 | 0.01 |
| | **45** | GP= | 0.03 | 0.04 | 0.04 | 0.04 | 0.03 | 0.03 | 0.02 | 0.02 | 0.03 | 0.03 | 0.04 | 0.04 |
| **I** | | NN= | 0.12 | 0.17 | 0.19 | 0.18 | 0.16 | 0.16 | 0.17 | 0.18 | 0.20 | 0.22 | 0.23 | 0.26 |
| | | BS= | 0.16 | 0.16 | 0.16 | 0.15 | 0.14 | 0.12 | 0.11 | 0.09 | 0.07 | 0.05 | 0.04 | 0.03 |
| **T** | **60** | GP= | 0.04 | 0.04 | 0.05 | 0.05 | 0.04 | 0.03 | 0.03 | 0.03 | 0.03 | 0.03 | 0.04 | 0.04 |
| | | NN= | 0.17 | 0.19 | 0.18 | 0.16 | 0.15 | 0.15 | 0.17 | 0.19 | 0.20 | 0.21 | 0.23 | 0.26 |
| **Y** | | BS= | 0.19 | 0.19 | 0.19 | 0.18 | 0.17 | 0.16 | 0.15 | 0.13 | 0.11 | 0.10 | 0.08 | 0.07 |
| | **90** | GP= | 0.06 | 0.06 | 0.06 | 0.06 | 0.06 | 0.05 | 0.05 | 0.05 | 0.05 | 0.05 | 0.05 | 0.06 |
| | | NN= | 0.16 | 0.14 | 0.13 | 0.14 | 0.16 | 0.18 | 0.19 | 0.20 | 0.20 | 0.21 | 0.24 | 0.28 |

**Table VIII**
**Real World Applications: Genetic Programming Model Specification for S&P500 Index Option and Five Equity Options**

| Name | Source | Definition |
|---|---|---|
| S | Option Contract | Stock price |
| X | Option Contract | Exercise price |
| S/X | Black-Scholes parameter | Degree of option moneyness |
| R | Black-Scholes parameter | Risk-free rate |
| □ | Black-Scholes parameter | Annualized historical volatility |
| □ | Option Contract | Time to maturity (in years) |
| Max(S-X) | Boundary Condition | Option intrinsic value Max (S-X,0) |
| Black-Scholes | Naïve investor's valuation | Black Scholes value of option |
| + | Standard arithmetic | Addition |
| - | Standard arithmetic | Subtraction |
| * | Standard arithmetic | Multiplication |
| % | Standard arithmetic | Protected Division: x%y = 1 , if y = 0<br>= x/y , otherwise |
| Exp | Black-Scholes component | Exponent: $\exp(x) = e^x$ |
| Plog | Black-Scholes component | Protected Natural log: plog(x) = ln(\|x\|) |
| Psqrt | Black-Scholes component | Protected Square root: psqrt(x) = sqrt(\|x\|) |
| Ncdf | Black-Scholes component | Normal Cumulative Distribution Function |

**Table IX**
**Performance in the Real World**

This table shows the mean absolute pricing errors for ten Genetic Programming Formulas, the Black-Scholes model, and Neural Networks, on ten out-of-sample data sets of the S&P 500 Index (SPX) option and five equity options. Each formula came from a separate training set and was evaluated on a separate test set. The parameter search was performed using April 3-4 data to find algorithm parameters that give formulas with good out-of-sample performance. Ten training sets with these parameters were used to train an equal number of formulas for each underlying asset. Ten additional test sets were used per underlying asset to test their out-of-sample performance. All training and test sets for SPX came from April 6-10 BODB data and those for five equity options came from April 6-13. Each cell contains the average error over an out-of-sample data set. The model specifications for Genetic Programming are given in Table IX.

| | **SP 500 Training Set Size=50, Test Set Size=50** | | | **Best Buy Training Set Size=200, Test Set Size=50** | | |
|---|---|---|---|---|---|---|
| Formula/ Test Set | Genetic Programming Mean Absolute Pricing Errors | Black-Scholes Mean Absolute Pricing Errors | Neural Networks Mean Absolute Pricing Error | Genetic Programming Mean Absolute Pricing Errors | Black-Scholes Mean Absolute Pricing Errors | Neural Networks Mean Absolute Pricing Error |
| 1 | 1.955458 | 3.230137 | 1.23707 | 0.092718 | 0.118118 | 0.05825 |
| 2 | 3.032258 | 4.673598 | 1.17730 | 0.060467 | 0.085869 | 0.06334 |
| 3 | 2.361709 | 3.324834 | 1.20080 | 0.115019 | 0.079415 | 0.07712 |
| 4 | 1.517703 | 3.288438 | 1.04564 | 0.099566 | 0.101912 | 0.05830 |
| 5 | 2.480248 | 3.544208 | 2.05279 | 0.126194 | 0.086824 | 0.07433 |
| 6 | 2.021971 | 3.011138 | 0.78043 | 0.104570 | 0.089499 | 0.06210 |
| 7 | 2.335770 | 3.180231 | 1.69375 | 0.101157 | 0.114478 | 0.05273 |
| 8 | 2.176582 | 3.315890 | 1.21113 | 0.118906 | 0.078582 | 0.07014 |
| 9 | 1.630518 | 3.284605 | 1.07199 | 0.085496 | 0.101072 | 0.06011 |
| 10 | 3.080306 | 4.180943 | 0.93764 | 0.118083 | 0.077998 | 0.07660 |
| Average | 2.259 | 3.503 | 1.24085 | 0.102218 | 0.093377 | 0.06530 |

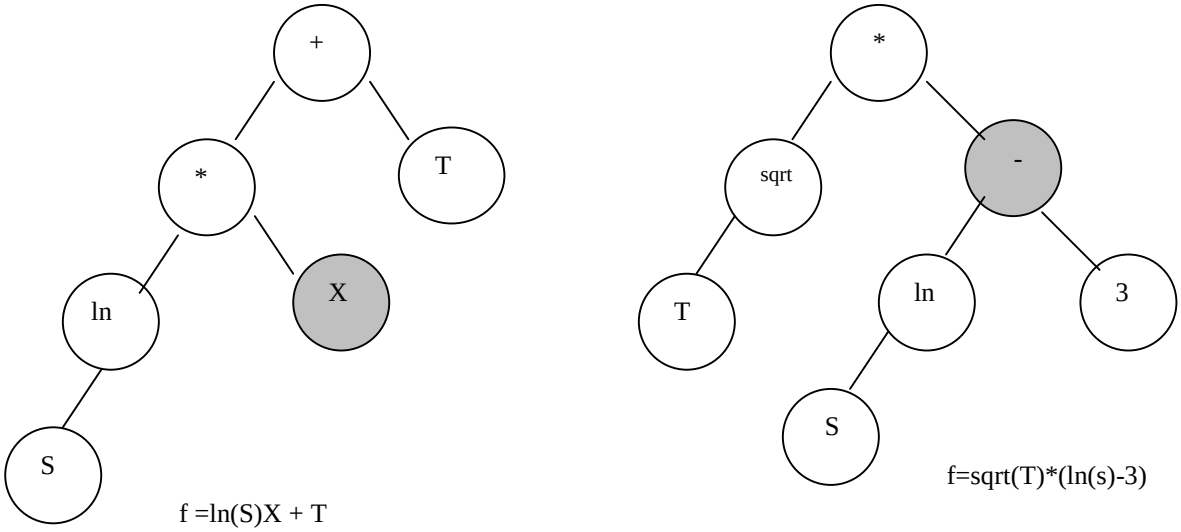
| | **Broderbund Training Set Size=50, Test Set Size=50** | | | **Comp USA Training Set Size=200, Test Set Size=50** | | |
|---|---|---|---|---|---|---|
| Formula/ Test Set | Genetic Programming Mean Absolute Pricing Errors | Black-Scholes Mean Absolute Pricing Errors | Neural Networks Mean Absolute Pricing Error | Genetic Programming Mean Absolute Pricing Errors | Black-Scholes Mean Absolute Pricing Errors | Neural Networks Mean Absolute Pricing Error |
| 1 | 0.134556 | 0.134556 | 0.74069 | 0.147540 | 0.189146 | 0.16404 |
| 2 | 0.125486 | 0.125486 | 0.60994 | 0.191933 | 0.185619 | 0.13962 |
| 3 | 0.107404 | 0.107404 | 0.56951 | 0.214485 | 0.148687 | 0.27743 |
| 4 | 0.141839 | 0.141452 | 0.59754 | 0.162178 | 0.186945 | 0.13950 |
| 5 | 0.144841 | 0.144841 | 0.68213 | 0.159190 | 0.165545 | 0.16200 |
| 6 | 0.127919 | 0.127919 | 0.69162 | 0.213317 | 0.184263 | 0.26520 |
| 7 | 0.125268 | 0.125268 | 0.61138 | 0.166541 | 0.185641 | 0.12111 |
| 8 | 0.121064 | 0.121064 | 0.56095 | 0.150210 | 0.188550 | 0.15825 |
| 9 | 0.147464 | 0.147463 | 0.72522 | 0.179620 | 0.192222 | 0.14528 |
| 10 | 0.107356 | 0.115799 | 0.63489 | 0.151901 | 0.186986 | 0.17235 |
| Average | 0.128320 | 0.129126 | 0.64239 | 0.173692 | 0.181361 | 0.17448 |

**Table IX**
**Performance in the Real World (continued)**

| | **DEC Training Set Size=50, Test Set Size=50** | | | **Novellus Training Set Size=50, Test Set Size=50** | | |
|---|---|---|---|---|---|---|
| Formula/ Test Set | Genetic Programming Mean Absolute Pricing Errors | Black-Scholes Mean Absolute Pricing Errors | Neural Networks Mean Absolute Pricing Error | Genetic Programming Mean Absolute Pricing Errors | Black-Scholes Mean Absolute Pricing Errors | Neural Networks Mean Absolute Pricing Error |
| 1 | 0.159522 | 0.159522 | 0.43014 | 0.142276 | 0.253066 | 0.47557 |
| 2 | 0.146230 | 0.146230 | 0.41401 | 0.197596 | 0.284001 | 0.52269 |
| 3 | 0.143515 | 0.143515 | 0.45442 | 0.224893 | 0.284992 | 0.45737 |
| 4 | 0.114799 | 0.114799 | 0.40556 | 0.181735 | 0.269153 | 0.50346 |
| 5 | 0.087957 | 0.122264 | 0.43355 | 0.188179 | 0.277145 | 0.44845 |
| 6 | 0.125400 | 0.125400 | 0.40758 | 0.204846 | 0.234419 | 0.43888 |
| 7 | 0.123594 | 0.123594 | 0.43016 | 0.166795 | 0.321690 | 0.45069 |
| 8 | 0.128593 | 0.128593 | 0.33289 | 0.200639 | 0.197881 | 0.58192 |
| 9 | 0.145408 | 0.145408 | 0.43425 | 0.195321 | 0.309613 | 0.58591 |
| 10 | 0.106262 | 0.106262 | 0.41054 | 0.161740 | 0.297320 | 0.44203 |
| Average | 0.128129 | 0.131559 | 0.41531 | 0.186402 | 0.272928 | 0.49070 |

**Figure 1**. **Parents and Offspring**
Randomly chosen crossover points are shaded.

**Parents**




**Offspring**

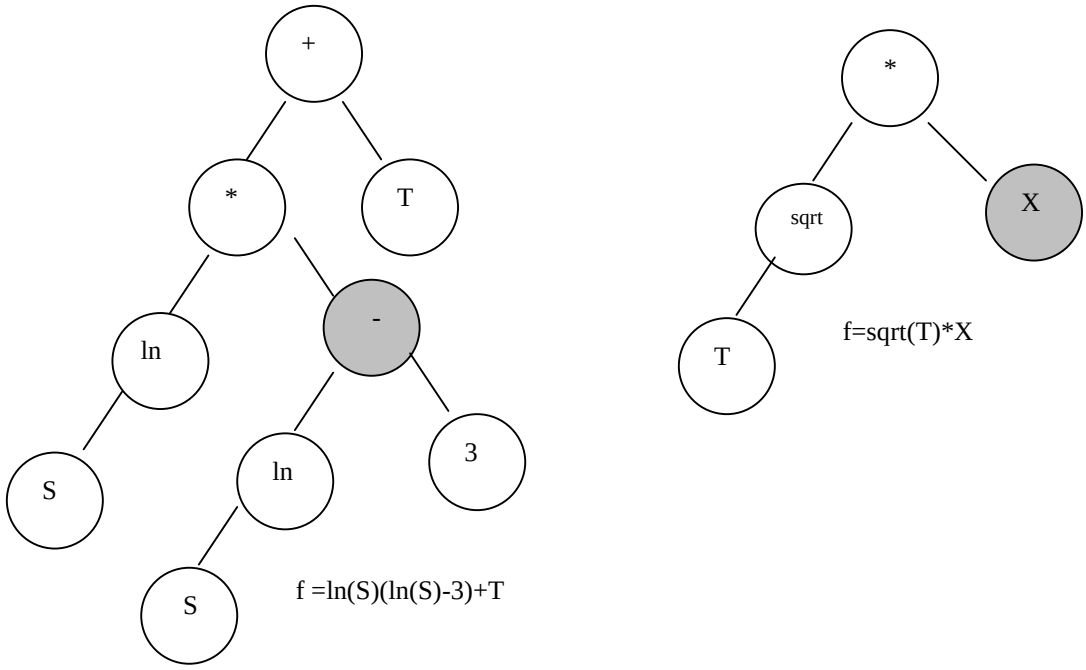
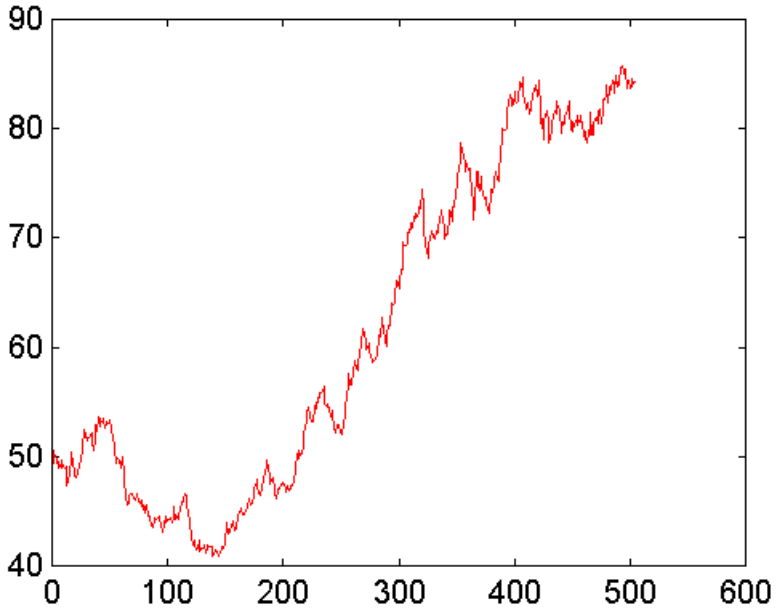


**Figure 2. The Black-Scholes world.** In the Black-Scholes world, the underlying stock prices follow Geometric Brownian motion.

***A. A sample stock price path.***




***B. The distribution of option prices derived from the above stock price path in a Black-Scholes world.***

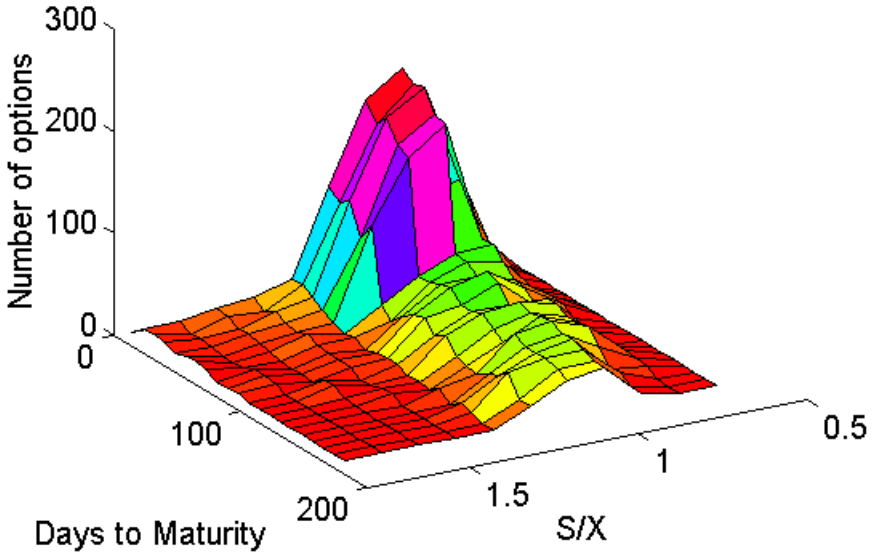


**Figure 3. The performance of Genetic Programming in a Black-Scholes World.** The performance is measured by mean absolute errors. This figure illustrates pricing errors generated by Genetic Programming.

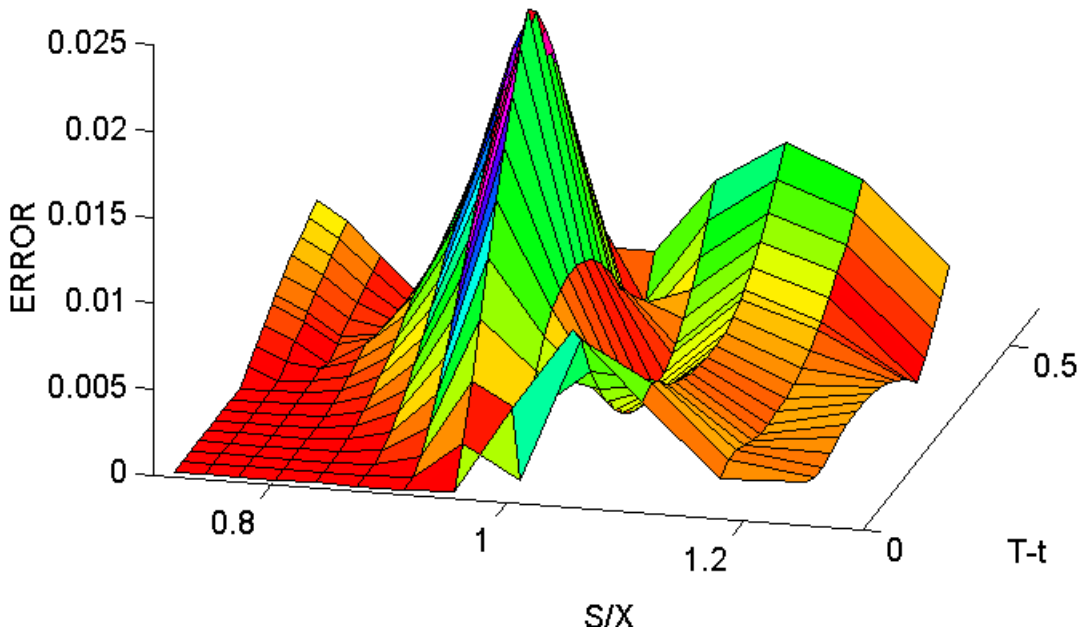
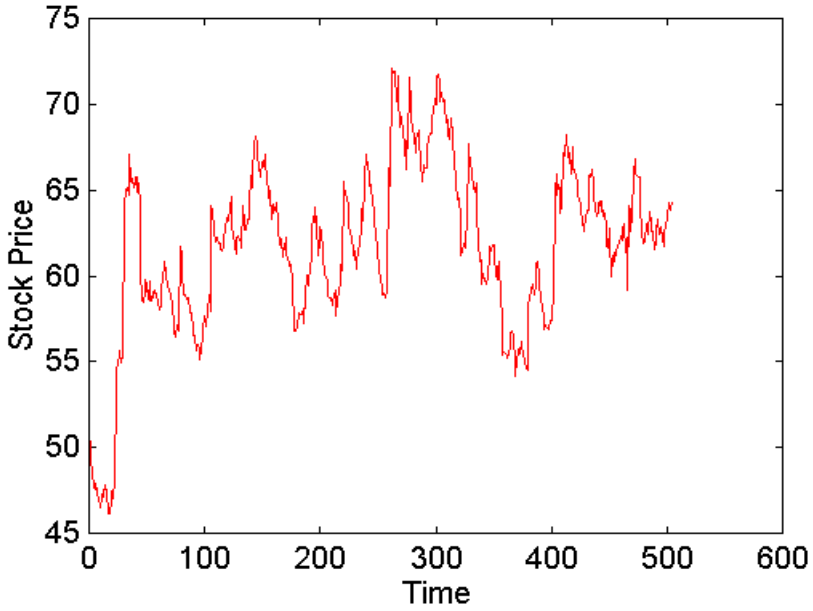


**Figure 4. The jump-diffusion world.**
***A. A sample stock price path.***




***B. The distribution of option prices derived from the above stock price path in a jump-diffusion world.***

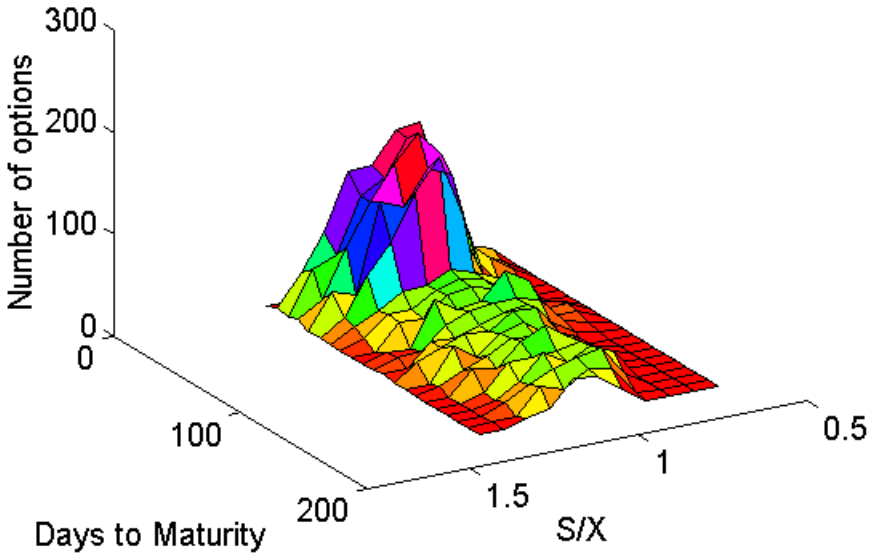
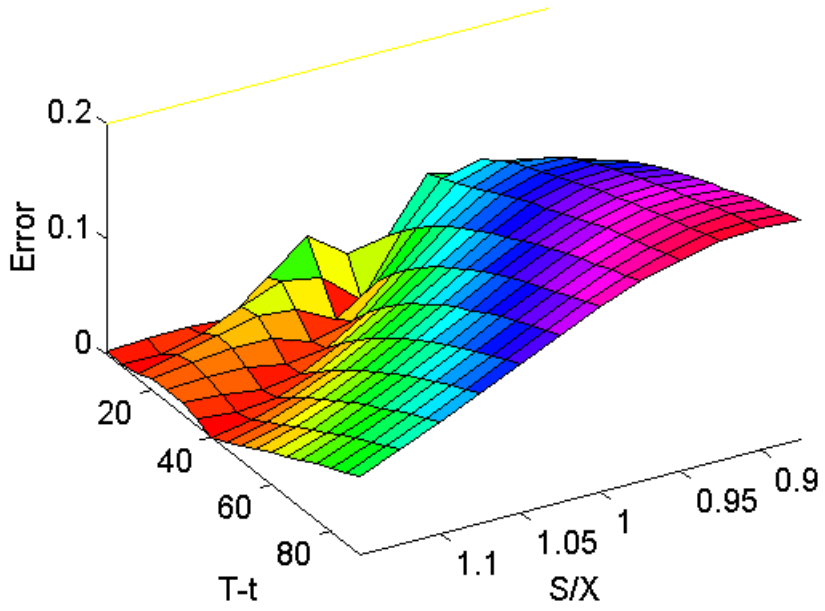
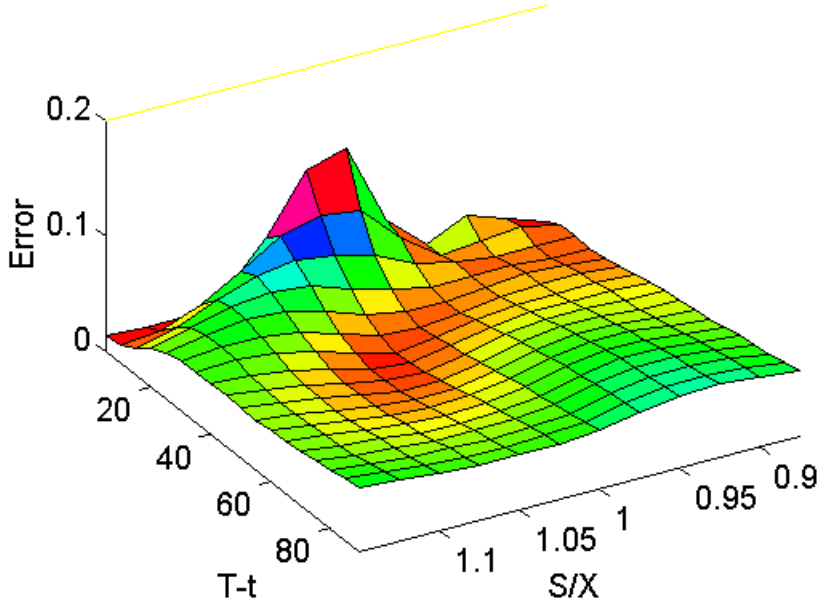


**Figure 5. Performances of the Black-Scholes model and Genetic Programming in a jump-diffusion world.** We use the Black-Scholes model and Genetic Programming to estimate option prices when the underlying asset prices follow a jump-diffusion process described in Figure 4.

***A. Absolute pricing errors of the Black-Scholes Model in a jump-diffusion world.***




***B. Absolute pricing errors made by the Genetic Programming in a jump-diffusion world.***



## Appendix A

### Performance of Genetic Programming in a Black-Scholes World
### --Measured by Mean Percentage Pricing Errors--

We generate the underlying stock prices as Geometric Brownian motion. The model specifications for Genetic Programming are specified in Table I. Pricing errors are presented for six parent selection algorithms to evaluate the efficiency of alternate methods for generating new populations from the previous generation of formulas. Each cell in the table is the average value across 10 Genetic Programming formulas generated for each of the alternate methods. For each Genetic Programming formula, the error is calculated by taking the average pricing error over five options. Rows in the table represent days-to-maturity and columns represent the degree-of-moneyness, S/X.

Parent Selection Criteria : **Best**
Individuals with the smallest pricing errors are selected to be in the new population.

| | | **Moneyness S/X** | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **0.875** | **0.9** | **0.925** | **0.95** | **0.975** | **1** | **1.025** | **1.05** | **1.075** | **1.1** | **1.125** | **1.15** |
| **MATURITY** | **5** | 820.0% | 820.0% | 728.4% | 171.6% | 67.9% | 81.5% | 7.7% | 1.5% | 1.5% | 1.4% | 1.4% | 1.3% |
| | **10** | 731.7% | 728.0% | 123.7% | 38.9% | 59.1% | 77.5% | 14.5% | 1.7% | 1.4% | 1.5% | 1.5% | 1.4% |
| | **30** | 188.7% | 67.4% | 26.9% | 33.9% | 52.0% | 74.0% | 23.2% | 7.7% | 2.1% | 1.5% | 1.9% | 2.0% |
| | **45** | 129.7% | 43.5% | 24.9% | 32.2% | 45.8% | 71.5% | 23.9% | 9.9% | 3.6% | 2.0% | 2.1% | 2.3% |
| | **60** | 93.4% | 35.2% | 24.6% | 29.0% | 39.5% | 68.4% | 23.1% | 10.8% | 4.8% | 2.7% | 2.6% | 2.7% |
| | **90** | 68.3% | 33.3% | 23.1% | 23.5% | 30.3% | 63.1% | 20.5% | 11.4% | 6.3% | 4.2% | 3.7% | 3.8% |

Parent Selection Criteria : **Fitness**
Individuals are chosen randomly with a probability that is inversely proportional to their pricing errors.

| | | **Moneyness S/X** | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **0.875** | **0.9** | **0.925** | **0.95** | **0.975** | **1** | **1.025** | **1.05** | **1.075** | **1.1** | **1.125** | **1.15** |
| **MATURITY** | **5** | 900.0% | 900.0% | 900.0% | 75.7% | 88.8% | 96.3% | 9.6% | 1.2% | 0.8% | 0.8% | 0.7% | 0.7% |
| | **10** | 900.0% | 841.1% | 184.9% | 52.9% | 82.7% | 92.4% | 17.4% | 3.3% | 0.8% | 0.8% | 0.7% | 0.7% |
| | **30** | 371.2% | 101.2% | 51.1% | 58.3% | 75.6% | 84.8% | 29.5% | 12.1% | 4.7% | 2.1% | 1.3% | 1.2% |
| | **45** | 177.0% | 62.4% | 50.3% | 58.0% | 72.2% | 81.0% | 32.4% | 16.0% | 7.6% | 3.9% | 2.4% | 1.7% |
| | **60** | 108.1% | 53.9% | 49.9% | 57.2% | 69.4% | 78.0% | 33.7% | 18.4% | 9.9% | 5.6% | 3.5% | 2.5% |
| | **90** | 71.7% | 48.6% | 49.1% | 55.2% | 64.8% | 73.0% | 34.5% | 21.4% | 13.0% | 8.3% | 5.6% | 4.1% |

**Appendix A (continued)**

Parent Selection Criteria : **Fitness-overselection**

Individuals are divided into two groups. Group 1 has the top 320 individual with the smallest pricing errors. The remainders are placed in Group 2. Individuals are then chosen randomly with a higher probability assigned to Group 1. In our implementation, the probability of selection was 80% for Group 1 individuals and 20% for Group 2 individuals.

| | | **Moneyness S/X** | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **0.875** | **0.9** | **0.925** | **0.95** | **0.975** | **1** | **1.025** | **1.05** | **1.075** | **1.1** | **1.125** | **1.15** |
| **MATURITY** | **5** | 820.0% | 787.6% | 696.2% | 95.6% | 53.9% | 69.9% | 9.5% | 5.3% | 3.7% | 2.7% | 2.2% | 1.9% |
| | **10** | 808.6% | 706.2% | 73.4% | 41.9% | 50.7% | 68.8% | 9.2% | 4.1% | 3.7% | 2.9% | 2.3% | 2.0% |
| | **30** | 123.4% | 66.8% | 27.9% | 30.4% | 45.4% | 59.7% | 16.8% | 6.1% | 3.1% | 2.9% | 2.8% | 2.5% |
| | **45** | 101.0% | 42.4% | 23.2% | 28.0% | 40.8% | 53.5% | 19.6% | 8.4% | 4.1% | 3.1% | 3.0% | 2.9% |
| | **60** | 73.3% | 32.5% | 20.9% | 25.3% | 36.7% | 48.2% | 20.6% | 9.9% | 5.1% | 3.8% | 3.5% | 3.3% |
| | **90** | 53.0% | 28.1% | 19.5% | 21.2% | 30.3% | 40.0% | 20.2% | 11.3% | 6.9% | 5.4% | 4.9% | 4.6% |

Parent Selection Criteria : **Random**

Individuals are chosen randomly and their fitness errors are ignored.

| | | **Moneyness S/X** | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **0.875** | **0.9** | **0.925** | **0.95** | **0.975** | **1** | **1.025** | **1.05** | **1.075** | **1.1** | **1.125** | **1.15** |
| **MATURITY** | **5** | 820.0% | 820.0% | 732.0% | 219.7% | 92.8% | 96.9% | 11.4% | 1.9% | 1.1% | 0.9% | 0.8% | 0.7% |
| | **10** | 820.0% | 711.3% | 294.0% | 67.4% | 84.2% | 92.8% | 20.2% | 4.5% | 1.5% | 1.1% | 0.9% | 0.8% |
| | **30** | 384.8% | 125.4% | 45.9% | 59.2% | 76.5% | 85.2% | 35.4% | 15.0% | 6.5% | 3.2% | 2.1% | 1.6% |
| | **45** | 180.4% | 67.1% | 41.1% | 57.5% | 72.6% | 81.2% | 39.6% | 19.7% | 9.9% | 5.3% | 3.3% | 2.3% |
| | **60** | 115.6% | 45.1% | 39.1% | 54.9% | 68.9% | 77.5% | 41.6% | 22.6% | 12.4% | 7.0% | 4.3% | 3.0% |
| | **90** | 71.4% | 30.7% | 34.5% | 49.1% | 62.1% | 70.8% | 42.2% | 25.4% | 15.3% | 9.4% | 6.0% | 4.1% |

Parent Selection Criteria : **Tournament, n=4**

Four individuals are first chosen randomly from the population. The best of the four individuals is then selected for the next generation.

| | | **Moneyness S/X** | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **0.875** | **0.9** | **0.925** | **0.95** | **0.975** | **1** | **1.025** | **1.05** | **1.075** | **1.1** | **1.125** | **1.15** |
| **MATURITY** | **5** | 740.0% | 655.3% | 571.7% | 151.2% | 58.8% | 77.9% | 14.0% | 7.0% | 3.5% | 2.1% | 1.7% | 1.6% |
| | **10** | 579.6% | 365.7% | 82.5% | 45.6% | 48.0% | 74.0% | 10.4% | 5.7% | 3.7% | 2.5% | 2.0% | 1.8% |
| | **30** | 99.7% | 32.1% | 19.6% | 31.5% | 42.3% | 69.8% | 12.4% | 4.9% | 3.2% | 2.9% | 2.7% | 2.5% |
| | **45** | 68.4% | 29.0% | 18.9% | 28.2% | 37.5% | 64.8% | 13.7% | 6.3% | 3.7% | 3.2% | 3.0% | 2.9% |
| | **60** | 59.9% | 28.8% | 17.9% | 24.5% | 33.0% | 59.5% | 14.1% | 7.3% | 4.5% | 3.5% | 3.3% | 3.3% |
| | **90** | 55.8% | 30.9% | 18.0% | 18.8% | 25.6% | 50.5% | 13.6% | 8.5% | 5.7% | 4.4% | 4.1% | 4.2% |

Parent Selection Criteria : **Tournament, n=7**

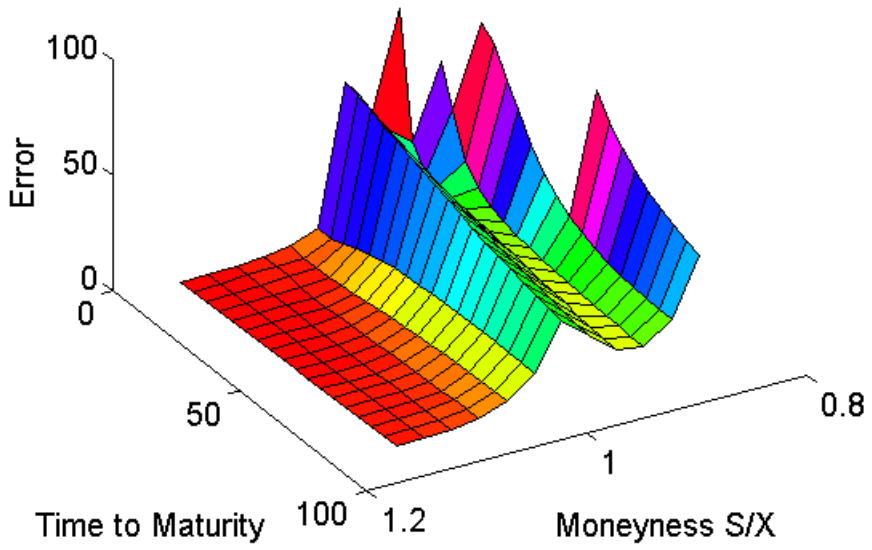
Seven individuals are first chosen randomly from the population. The best of the seven individuals is then selected for the next generation.

| | | Moneyness S/X | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **0.875** | **0.9** | **0.925** | **0.95** | **0.975** | **1** | **1.025** | **1.05** | **1.075** | **1.1** | **1.125** | **1.15** |
| **MATURITY** | **5** | 740.0% | 739.1% | 644.6% | 39.9% | 42.1% | 75.1% | 8.8% | 3.7% | 1.8% | 0.9% | 0.6% | 0.6% |
| | **10** | 746.0% | 469.0% | 45.7% | 25.6% | 38.4% | 69.5% | 10.5% | 3.8% | 2.1% | 1.1% | 0.7% | 0.6% |
| | **30** | 59.4% | 31.0% | 22.5% | 22.3% | 27.1% | 63.2% | 11.9% | 6.1% | 3.1% | 1.7% | 1.1% | 1.0% |
| | **45** | 49.0% | 25.0% | 23.0% | 20.0% | 21.6% | 60.4% | 11.4% | 7.5% | 4.5% | 2.5% | 1.6% | 1.2% |
| | **60** | 41.1% | 22.6% | 22.7% | 18.5% | 18.4% | 57.8% | 10.9% | 8.4% | 5.7% | 3.5% | 2.1% | 1.5% |
| | **90** | 33.8% | 19.8% | 21.8% | 17.9% | 14.7% | 53.2% | 10.0% | 9.2% | 7.1% | 5.3% | 3.5% | 2.4% |

**Appendix B**

**The performance of Genetic Programming in a Black-Scholes World.**

The performance is measured by mean absolute percentage errors. This figure illustrates pricing errors generated by Genetic Programming. Errors were capped at 100% for the purposes of this figure and the large percentage errors observed for short maturity out-of-the-money options are omitted.



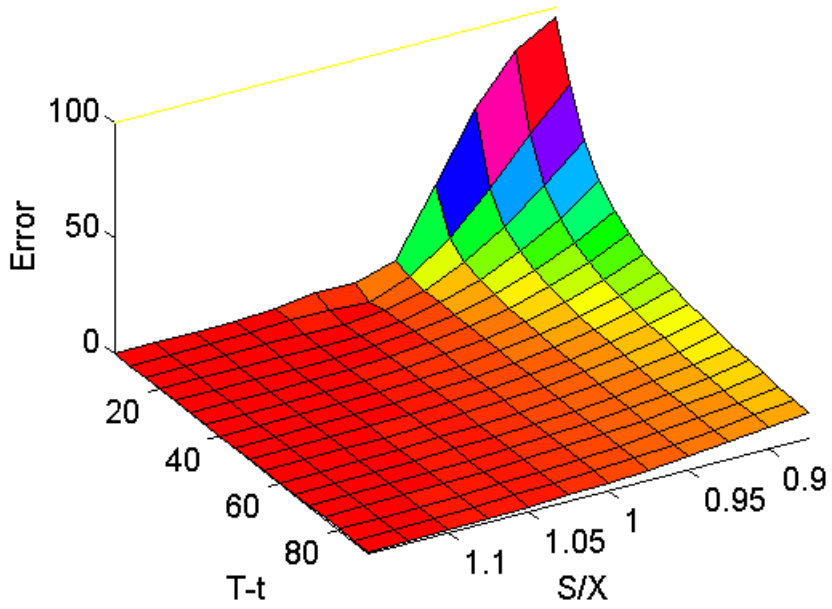
**Appendix C**
**Performance Comparison of Genetic Programming. the Black-Scholes Model, and Neural Networks in a Jump Diffusion World Measured by Absolute Percentage Pricing Errors**

The numbers in each cell are the average pricing errors from the models across 10 test sets for the Genetic Programming formulas, the Black-Scholes model, and Neural Networks. For each test set, the error value for each cell is calculated by taking the average pricing errors over five options. Rows in the table represent days-to-maturity and columns represent the degree-of-moneyness, S/X. We shade the areas where Genetic Programming improves Black-Scholes. Genetic Programming is better than neural networks in all cases. The model specification for Genetic Programming is given in Table IV.

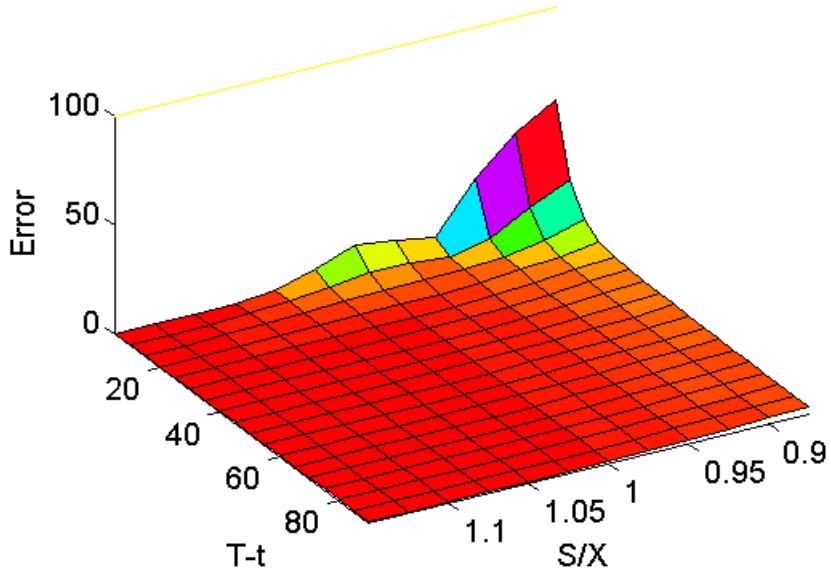
| | | **0.875** | **0.9** | **0.925** | **0.95** | **0.975** | **1** | **1.025** | **1.05** | **1.075** | **1.1** | **1.125** | **1.15** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BS= | 95.6% | 85.7% | 65.7% | 36.5% | 8.7% | 3.7% | 3.5% | 1.1% | 0.1% | 0.1% | 0.1% | 0.0% |
| **5** | GP= | 58.0% | 47.1% | 30.9% | 8.5% | 11.3% | 13.9% | 6.9% | 2.4% | 0.7% | 0.3% | 0.2% | 0.2% |
| | NN= | 862.6% | 443.7% | 189.5% | 77.2% | 30.0% | 12.7% | 9.5% | 7.7% | 5.4% | 4.4% | 4.1% | 3.5% |
| | BS= | 71.8% | 54.6% | 35.8% | 18.8% | 6.5% | 0.3% | 1.5% | 1.3% | 0.6% | 0.2% | 0.0% | 0.0% |
| **10** | GP= | 25.8% | 18.1% | 8.3% | 4.5% | 5.8% | 6.1% | 4.2% | 2.5% | 1.3% | 0.6% | 0.3% | 0.2% |
| | NN= | 256.9% | 115.6% | 45.0% | 15.9% | 8.6% | 8.7% | 8.5% | 7.3% | 5.2% | 4.2% | 3.9% | 3.5% |
| | BS= | 27.3% | 19.7% | 13.5% | 8.8% | 5.3% | 3.0% | 1.5% | 0.6% | 0.2% | 0.1% | 0.1% | 0.1% |
| **30** | GP= | 5.6% | 4.3% | 3.0% | 1.9% | 1.2% | 0.8% | 0.6% | 0.8% | 0.9% | 0.8% | 0.7% | 0.5% |
| | NN= | 10.8% | 13.4% | 14.9% | 13.0% | 9.4% | 7.0% | 5.7% | 4.8% | 4.1% | 3.6% | 3.4% | 3.3% |
| | BS= | 19.0% | 14.2% | 10.4% | 7.3% | 5.0% | 3.4% | 2.1% | 1.3% | 0.8% | 0.4% | 0.2% | 0.1% |
| **45** | GP= | 4.5% | 3.3% | 2.5% | 1.9% | 1.3% | 0.8% | 0.5% | 0.5% | 0.5% | 0.6% | 0.5% | 0.5% |
| | NN= | 15.9% | 16.0% | 13.6% | 9.9% | 6.9% | 5.3% | 4.5% | 4.0% | 3.7% | 3.4% | 3.1% | 3.1% |
| | BS= | 15.1% | 11.7% | 8.9% | 6.6% | 4.9% | 3.5% | 2.5% | 1.7% | 1.2% | 0.8% | 0.5% | 0.3% |
| **60** | GP= | 3.8% | 3.0% | 2.5% | 2.0% | 1.4% | 1.0% | 0.7% | 0.5% | 0.5% | 0.5% | 0.5% | 0.5% |
| | NN= | 15.8% | 13.5% | 10.1% | 7.0% | 5.1% | 4.3% | 4.0% | 3.7% | 3.4% | 3.1% | 2.9% | 3.0% |
| | BS= | 11.4% | 9.2% | 7.4% | 5.9% | 4.6% | 3.6% | 2.8% | 2.2% | 1.7% | 1.3% | 0.9% | 0.7% |
| **90** | GP= | 3.4% | 2.9% | 2.4% | 1.9% | 1.5% | 1.1% | 0.9% | 0.8% | 0.7% | 0.7% | 0.6% | 0.6% |
| | NN= | 9.5% | 6.8% | 4.9% | 4.4% | 4.4% | 4.2% | 3.8% | 3.4% | 2.9% | 2.7% | 2.8% | 2.9% |

**Appendix D**

**Percentage pricing errors made by the Black-Scholes Model in a jump-diffusion world.**




**Percentage pricing errors made by Genetic Programming in a jump-diffusion world.**



## Appendix E
## Neural networks used in the study

Following Hutchinson, Lo, and Poggio (1994), we trained a one-hidden layer multilayer-perceptron networks (MLPs) to price options on our data set. Our networks also use the same topology. Using their notation, the networks may be represented by the following equation:

$$f(\vec{x}) = h(\sum_{i=1}^{n} \delta_i h(\beta_{oi} + \vec{\beta}'_{1i} \quad \vec{x}) + \delta_o)$$

where n is the number of hidden layers, $\delta_i, \beta_{i,j}$ are weights, $\vec{x}$ is the vector of inputs, and *h* is the logistic function

$$h(u) = \frac{1}{1+e^{-u}}$$

We used four data normalization schemes and two weight initialization schemes. The number of hidden nodes, training cycles, and training method, are as specified by Hutchinson, Lo, and Poggio (1994). Table E.1 shows the network training parameters.

**Table E.1**
**Neural Network Training Parameters**

| | |
|---|---|
| Input Variables | Same as GP input variables, normalized to lie in [0,1] |
| Output Variable | Call price, normalized to lie in [0,1] |
| Training cycles | 10,0000 |
| Training Method | Gradient descent with momentum. |
| Initialization | 1) Random $\beta_{i,j}$, $\delta_i$. Weights were chosen by neural networks package (Matlab) according to ranges in training data values.<br>2) Unit weight to Black-Scholes, zero to other variables ($\beta_{i,j}$, $\delta_i$ = 0 $\forall$ i, j , except for $\beta_{11}$, $\delta_1$, which are set equal to (This method almost always gave the best results). |
| Normalization of inputs and outputs | 1) Divide each input by the corresponding value in normalization vector.<br>2) Divide all inputs by maximum value in the normalization vector.<br>3) Normalize as in (1) and then apply logistic function *h*.<br>4) Normalize as in (2) and then apply logistic function *h*. |

After training each network we evaluated it on an out-of-sample data set. The results presented in the paper are those of the network with the **smallest mean absolute error** on the out-of-sample data set.