Portfolio Optimization using Genetic Algorithm and Monte Carlo Simulation

Xiyuan Zheng (251121253)
Yizhou Cai (251120680)
Ice, Wei Fan (250938491)

Department of Statistical and Actuarial Sciences
University of Western Ontario

April 20th, 2020

Abstract

The genetic algorithm (GA) is one of widely used search heuristic applied in various areas nowadays. It was inspired from Charles Darwin's theory of natural evolution and aims to find the fittest individual through the natural selection process. In this project, we combined the genetic algorithm with Monte Carlo simulations to address a portfolio optimization problem in a virtual equity market. The genetic algorithm was used as a core process to find an optimized asset allocation strategy, and the Monte Carlo simulations were used to obtain the optimal solution on average, study the behaviour of GA, and test the final performance. Based on the final results, the proposed methodology and its comparison groups were analyzed in terms of fitness score, running behaviour and time complexity.

Key words: Portfolio Optimization, Genetic Algorithm, Monte Carlo Simulation, search heuristic.

ACKNOWLEDGEMENTS

We would like first to express the most sincere gratitude and appreciation to Professor Yu whose guidance, support and patience from the initial and final level of this study term. We also owe our greatest gratitude to Professor Yu for his grant to form this project team so that we can work together, learn from each other and build up an effective team.

As a member of the team, we would like to thank each other for everyone's leadership, ownership and hard-working. We all learn a lot from others. Thank you, the team.

Table of Content

1 Introduction **5**

1.1 Background 5

1.2 How Genetic Algorithms Work 5

1.3 Why Genetic Algorithms 7

2 Methodology Proposals **8**

2.1 General Setup 8

2.2 Genetic Algorithm Setup: 11

Initial Population Phase 12

Fitness Evaluation Phase 12

Selection Phase 13

Crossover Phase 13

Mutation Phase 15

2.3 First Layer of Monte Carlos Simulation (Generation Production) 15

2.4 Second Layer of Monte Carlos Simulation 15

2.5 Comparison Groups 17

2.6 Final Performance Evaluation 17

3 Code & Results **18**

3.1 Analysis of the Structure of Optimized Portfolios 18

3.2 Running Behavior and Convergences 21

3.3 Final Evaluation Results Comparison 22

3.4 Time complexity 23

4 Conclusion **24**

4.1 Summary and Conclusion 24

4.2 Future Research and Recommendations 25

5 Bibliography **26**

6 Appendix **27**

6.1 Main code 27

6.2 Get plots 34

1 Introduction

1.1 Background

The Genetic algorithm (GA) is known as a problem-solving method (or heuristics) that simulates the process of natural evolution (Reeves, 2010). A genetic algorithm utilizes the concept of natural selection to determine the optimal solution for a problem, and it is commonly used as optimizers for tuning parameters to minimize or maximize the feedback measures.

In capital markets, many researchers had applied GAs to find the best combinations of parameters for trading strategies. Some published papers have shown the effectiveness of these methods, including "Genetic Algorithms: Genesis of Stock Evaluation" (Kanungo, 2004) and "The Applications of Genetic Algorithms in Stock Market Data Mining Optimization" by Lin (2004).

A common challenge for analysts in the financial market is that there are so many factors to be considered when they try to optimize their investment strategies. This situation could lead to over-complicated models and enormous computational time costs for finding the best solution. In a portfolio optimization problem, since there are so many stocks coming from various industries with different characteristics, to construct an optimal portfolio with limited capital can be very challenging. Even if the investor has a short list of alternative stocks pre-selected, due to the uncertainty in market movements, determining what portion to invest in each stock to achieve the optimal return is difficult. However, by combining the genetic algorithm and Monte Carlos simulations, an optimal asset allocation strategy that can be produced and survive under unpredictable market situations.

1.2 How Genetic Algorithms Work

Before we jump into the details of the project, let's recap the famous quote by Charles Darwin:

It is not the strongest of the species that survives, nor the most intelligent , but the one most responsive to change. ***-- Darwin, 1859***

The first genetic algorithm was introduced by John Holland (1975). But it was unavailable to apply with reasonable computation time until later in the 1990s -- when computer processors became much faster. In 1996, some applications in modeling natural evolutionary systems, including immune systems, were introduced in a published paper (Michalewicz,1996). Nowadays, the application of genetic algorithms in various areas and industries are discovered, such as artificial neural networks technology to model, understand, and optimize drug formulations (Landin, 2013), population-based sampling and fragment-based protein structure prediction (Simoncini, 2017), antimicrobial peptide designing and optimization (Kumar, 2019), topology optimization of trusses using genetic algorithm, force method and graph theory (Kaveh, 2003) and etc.

Essentially, there are five key steps exists in any genetic algorithms:

1. Population Generation Phase:

   A population is constituted by chromosomes, and a chromosome is constituted by genes. One gene can represent the value of one or one set of parameters, and one chromosome with fixed length n can represent a set of n genes in the optimization problem. Chromosomes will be generated randomly until they fill the population size m.

2. Fitness Evaluation Phase:

   All the chromosomes in the population will be evaluated based on a fitness function defined by the user. The fitness function measures how good a chromosome (a set of parameter values) performs in the optimization problem.

3. Selection Phase:

   Based on the fitness performance score from the fitness evaluation phase, a proportion of the population will be eliminated due to their poor performance.

4. Crossover Phase:

   Crossover phase represents the reproduction process in nature, where two chromosomes exchange their genes and produce new chromosomes into the population. This crossover process only happens among those chromosomes survived in the selection phase.

5. Mutation Phase:

   For a new chromosome generated in the crossover phase, there is a small possibility for its genes to randomly change to another value which is unrelated to its parent chromosomes. The mutation stage represents the biological mutations occurring in genetic information, and provides the possibility to jump out of a local optimum and to add diversity to the population.

By combining the surviving chromosomes and the offspring of those chromosomes, they together form a new generation with population size *m*. Repeating the phase 2 through phase 5 with the new generations multiple times, this process will result in increasingly favorable chromosomes (parameters) over time, Once the stopping criteria is met, which could be running time, fitness, number of generations or other criteria, the genetic algorithm terminates.

1.3 Why Genetic Algorithms

"Exhaustive search" is one of the typical optimizing algorithms used in the old days. It simply tries every possible combination in order to obtain the optimal solution. Using such a method usually can find a global optimum eventually. However, this approach is very inefficient, and it becomes not applicable when the number of parameter combinations is huge. That is why it is critical for the users of exhaustive search optimizers to limit the number of variables they use, or limit the number of values these variables can take.

In a different way, the genetic algorithm does not try out every possible combination. Instead, it takes samples and evaluates qualities of them. This kind of method reduces the computational time and guarantees a near optimal solution. As a result, compared to "exhaustive search", more variables could be utilized, and more or all values of a variable could be used. Optimization can still take a good deal of time if you give GAs a large number of variables, but it will be doing much more work given the same amount of time.

There are many alternative optimization algorithms available nowadays; however, most of them only choose one section of the searching space to search at one time. The strategy of genetic algorithms is to search multiple parts of the searching space simultaneously, which means they are less likely to get stuck in "local minima" compared to other optimizers.

There are a lot of successful applications of genetic algorithms in financial areas according to the publications. Multiple Discriminant Analysis (Frank, 1965) is a famous modeling technique in this area since the 1960s, in which the genetic algorithms have been used to induce rules for bankruptcy prediction. Another good example is the genetic algorithm used in solving the universal business problem - credit scoring (Fogarty, 2012). We believe the genetic algorithms appear to be particularly well-suited for financial optimization problem because of three reasons:

- Financial problems are payoff-driven

  A little bit increase in the expected return relative to the same amount of risk could lead to an ideal profit for an investor. With limited amounts of capital, Investors need optimizers such as genetic algorithms to help them allocate their assets efficiently.

- Genetic algorithms are inherently quantitative

  In the financial market, there are various quantitative indicators for investors to make decisions. A genetic algorithm could naturally use those quantitative indicators to construct its genes or fitness function and archive the desired balance among those parameters.

- Genetic algorithms allow a wide variety of extensions and constraints that cannot be provided in traditional methods

  Genetic algorithms have shown a remarkable balance between efficiency and efficacy. The unique way in the implementation of genetic algorithms, the search from a population, the blindness to auxiliary information, and the randomized operators, make it robust compared to many other algorithms in this field.

In this project, we will address a portfolio optimization problem using both genetic algorithms and Monte Carlo simulation. The genetic algorithm is used as a core process to find an optimized asset allocation strategy, while the Monte Carlo simulation is used to obtain the averaged optimal solution, study the behaviour of GA, and test the performance of the averaged optimal solution. Two comparison groups are also created to test if our proposed methodology solution works better (details in next section). All the code for this implementation is attached in Appendix. In Section 3, the running results are collected and presented with analysis and comparison accordingly. Our conclusion for this project and the plan for future research are discussed in Section 4.

2 Methodology Proposals

In order to apply the genetic algorithm to the portfolio optimization problem and investigate the general behaviour of the genetic algorithm, we combined GA and Monte Carlo simulation and designed the following methodology.

2.1 General Setup

In this portfolio optimization problem, we assume the following general investment parameters:

Risk-free rate (r) = 0% ;
Total Investment Horizon (T) = 20 (years) ;
Total time step number (k) = 251 * T (trading days) ;
Initial Wealth P(0) = 1.

In this project, we suppose a simple equity market which contains only nine stocks. In order to introduce more complexity and variations in our hypothetical equity market, we assumed that those nine stocks are combinations with three different types of model (Black-Scholes Model, Heston Model, and Cox-Ingersoll-Ross Model) and risks & return features (low-risk, medium-risk, and high-risk). The combination of those features are demonstrated in the following table:

	Low-Risk	Medium-Risk	High-Risk
Black-Scholes	Stock 1	Stock 4	Stock 7
Heston	Stock 2	Stock 5	Stock 8
CIR	Stock 3	Stock 6	Stock 9

For the ith Stock $S_i(t)$ (i = 1...9):

Initial Stock Price $S_i(0) = 1$;

For low-risk stock:

Expected return $\mu_i = 0.06$, and Volatility $\sigma_i = 0.05$;

For medium-risk stock:

Expected return $\mu_i = 0.07$, Volatility $\sigma_i = 0.06$;

For high-risk stock stock:

Expected return $\mu_i = 0.08$, Volatility $\sigma_i = 0.07$.

Using the parameters and models above, we constructed the stock paths as follows:

For stock 1, 4, 7, they follow the Black-Scholes Model :

$$dS_i(\mathrm{t}) = \mu_i S_i(t)\,dt + \sigma_i S_i(t)\,dW(t)$$

where $W(t)$ is the Brownian motion in the stock price ;

For stock 2, 5, 8, they follow the Heston Model :

$$dS_i(t) = \mu_i S_i(\mathrm{t})\,\mathrm{dt} + \sqrt{V_i(t)}\,S_i(\mathrm{t})\,dW_{is}(\mathrm{t})$$

$$dV_i(t) = \kappa(\theta - V_i(t))\,\mathrm{dt} + \xi\sqrt{V_i(t)}\,dW_{iv}(\mathrm{t}),$$

where

$V_i(0) = \sigma_i$,

$\kappa = 0.1$, which represents the strength of changes in volatility ,

$\theta = \sigma_i$, which represents the long term volatility ,

$\xi = 1$, which represents the volatility of volatility V_i ,

$W_{iv}(t)$ represents the Brownian motion in volatility, and it is correlated to the Brownian motion in the stock price $W_{is}(t)$ with instances correlation coefficient ρ_i

where

$\rho_i = 0.5, 0.4, 0.3$ for stock 2, 5, 8 respectively

$W_{iv}(t) = \rho_i W_{is}(t) + \sqrt{1 - \rho_i^2}\,W_{random}(t)$;

For stock 3, 6, 9, we assume their drift term μ_i follows the Cox-Ingersoll-Ross Model (CIR):

$$dS_i(\mathrm{t}) = \mu_i(t)\,S_i(\mathrm{t})\,dt + \sigma_i\,S_i(\mathrm{t})\,dW_{is}(\mathrm{t})$$

$$d\mu_i(t) = \kappa(\theta - \mu_i(t))\,\mathrm{dt} + \xi\sqrt{\mu_i(t)}\,dW_{i\mu}(\mathrm{t}),$$

where

$\mu_i(0) = \mu_i$,

$\kappa = 0.1$, which represents the strength of changes in expected return ,

$\theta = \mu_i$ which represents the long term expected return ,

$\xi = 1$ which represents the volatility of drift term μ_i ,

$W_{iv}(t)$ represents the Brownian motion in volatility, and it is correlated to the Brownian motion in the stock price $W_{is}(t)$ with instantaneous correlation coefficient ρ_i

where

$\rho_i = 0.5, 0.4, 0.3$ for stock 3, 6, 9 respectively

$W_{i\mu}(t) = \rho_i W_{is}(t) + \sqrt{1 - \rho_i^2}\, W_{random}(t)$.

All the stock price movement model will be used Euler approximation to simulate:

Assuming X(t) is a stochastic process whose stochastic differential equation is defined as

$$dX(t) = \mathrm{a(t, X(t))}\ dt + \mathrm{b(t, X(t))}\ dW(t) .$$

Then in each time step Δt_i, the stochastic process is defined as

$$X(t_{i+1}) = X(t_i) + a(t_i, X(t_i))\Delta t_i + b(t_i, X(t_i))\,\Delta W_i$$

where

$\Delta t_i = \frac{T}{k}$,

$\Delta W_i = W_{i+1} - W_i$.

2.2 Genetic Algorithm Setup:

Since there are 9 stocks that exist in this market in total, we designed the “chromosome” in this portfolio optimization problem to be an asset allocation strategy for the portfolio, and it is represented as an array of length 9 which represents the weight of each stock (shown in the Chart 1). The “genes” which constitute chromosomes are the weights for the nine stocks w_i .

w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9

Chart 1: A chromosome of the portfolio strategy population

At the same time, those genes have constraints such that

$$\sum_{i=1}^{9} w_i = 1$$

Therefore, the last gene w_9 , which represents the asset invested in stock 9 depends on the weight of the other stocks ($w_1 \ldots w_8$) , and it should be defined as $1 - \sum_{i=1}^{8} w_i$.

I. Initial Population Phase

Set Initial population size = 2000. For each chromosome, the first eight weights would be generated randomly and independently using $Normal(0, 1)$, and the last weight can be calculated through $1 - \sum_{i=1}^{8} w_i$.

II. Fitness Evaluation Phase

The fitness function for the ith strategy is defined as follows:

Fitness $(chromosome_i) = \frac{RoR}{Sd} - \lambda Maxdropdown + (1-\lambda)\ Lowerpoint - 0.01\ \left(\sum_{i=1}^{9} w_i \{ w_i < 0 \} \right)$, where

- *RoR (Rate of Return)* is defined as the annual return of the portfolio strategy. If the RoR is lower than 0, then the strategy will immediately get a fitness score of negative infinity. Since the fundamental goal of optimizing the portfolio is to maximize the return/profit, if a strategy brings a negative annual return over the investment horizon, then we should never consider it as a desired strategy.

- *Sd (Standard Deviation)* represents the standard deviation of annual rate of return after applying the portfolio strategy. Notice that the ratio between *RoR* and *Sd* represents the Sharpe Ratio of the ith portfolio strategy (ith chromosome) under our assumption where risk-free interest rate is 0. The higher the Sharpe ratio, the better the portfolio strategy.

- *Maxdropdown* represents the averaged largest portfolio value dropdown in daily scale within one year. The lower the *Maxdropdown,* the better the portfolio strategy.

- *Lowerpoint* represents the averaged lowest portfolio value in percentage of the beginning portfolio within one year (this is a negative number).

- λ & ($1 - \lambda$) represent weight coefficients for *Maxdropdown* and *Lowerpoint* ranged in [0, 1]. These weight coefficients represent how adverse the inverter is on *Maxdropdown* and *Lowerpoint.* In this project, λ takes a value of 0.3.

- $\sum_{i=1}^{9} w_i \{ w_i < 0 \}$ represents the penalty term for implementing a shorting strategy. One particular reason for adding this penalty term is that we want to prevent the situation of an unlimited shorting strategy where a portfolio short stocks with low expected returns and purchasing the stocks with the highest expected return unlimitedly. Therefore, we encourage the algorithm to choose strategies with long positions in those stocks. At the same time, we also want to preserve the independence among the initial weights (w_1 *to* w_8). As a result, this constraint was not implemented in the initial population phase.

This fitness evaluation function will be applied to the entire population of the chromosomes.

III. Selection Phase

In the selection phase, the entire population will be sorted from the highest to lowest fitness score. Then the population is divided into the top 30% (upper class) and the bottom 70% (lower class). The lower class will be eliminated due to their poor fitness scores, while the upper class will survive and reproduce their next generations in the following crossover phase and mutation phase.

IV. Crossover Phase

For each pair of father chromosome and mother chromosome who survived in the selection phase:
A random integer *a* will be generated from Uniform distribution (1, 7), then the child chromosome’s weights were generated as follows:

- w_1 *to* w_a is copied from the father chromosome,
- w_{a+1} *to* w_8 is copied from the mother chromosome,
- w_9 is calculated through $1 - \sum_{i=1}^{8} w_i$.

Among the 30% of the survived population in the selection phase, we consider the top 2% chromosome (of the population) as the “elite class” which will preserve their original weights(children will keep complete genes from their father and mother without crossover). For the rest of the upper class (28% of the initial population), we consider them as “normal class”. They will be replaced by the child produced by mating every two pairs of them. As for the rest of the 70% of the population, they will be filled with the children of the entire upper class, which were produced by randomly selecting father chromosomes and mother chromosomes from the upper class (See Chart 2 below).

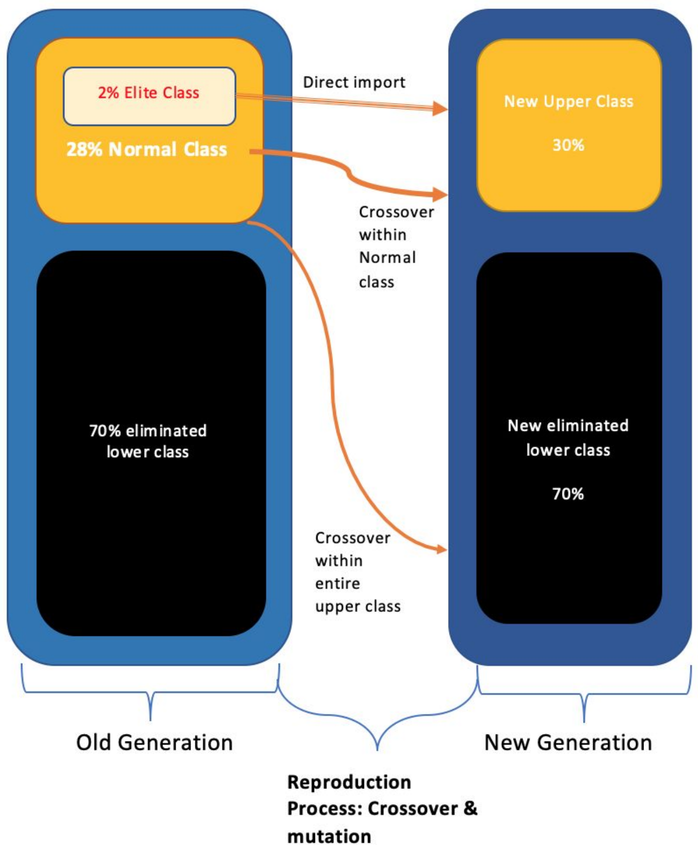

2% Elite Class
28% Normal Class
70% eliminated lower class
Direct import
Crossover within Normal class
Crossover within entire upper class
New Upper Class 30%
New eliminated lower class 70%
Old Generation
New Generation
Reproduction Process: Crossover & mutation

Chart 2: Reproduction Process

Notice that this process could lead to a situation where the father chromosome and mother chromosome are exactly the same. We allowed this situation to happen because this situation sometimes does happen in nature (e.g. asexual reproduction)

V. Mutation Phase

Set mutation rate = 1%. For the newly generated chromosomes, gene 1 to gene 8 (weight for stock 1 to stock 8) in the chromosome has a 1% chance for mutation. Once the mutation happens, the original w_i will be mutated as

$$w_i = w_i(1 + \zeta), \qquad i = 1, 2 \ldots 8$$

where $\zeta \sim Normal(0, \Phi)$.

Here we choose $\Phi = 0.1$. Finally, the weight for 9th stock is adjusted through $1 - \sum_{i=1}^{8} w_i$ according to the portfolio constraint $\sum_{i=1}^{9} w_i = 1$.

2.3 First Layer of Monte Carlos Simulation (Generation Production)

One last important step for genetic algorithms is to keep generating new populations of chromosomes and repeat from the fitness evaluation phase through the mutation phase multiple times. In this project, we set 300 generations as one trail of the genetic algorithm. In order to get an unbiased optimal chromosome through the genetic algorithm described above, after each new generation being produced from the old generation, a set of new paths for the nine stocks will be simulated according to their stochastic differential equations using Euler approximation. The new generation of chromosomes' fitness score will be evaluated based on their performance on the new paths of the stocks. In this way, the best chromosome produced from the last generation should be the optimal asset allocation strategy which produces the desired results under different market situations (the most fitted one).

2.4 Second Layer of Monte Carlos Simulation

In this project, 1000 trails (for computation time-wise) will be repeated so that we can study the behaviour of our genetic algorithm. Through this layer of the Monte Carlos simulation, 1000 best fitted optimal asset allocation strategies can be obtained. The final optimal strategy would be the average of these 1000 best chromosomes, that is,

$$MC\ Best\ Chromosome\ =\ \frac{1}{1000}\sum_{j=1}^{1000} Best\ Chromosome_j\ .$$

To have a complete view of proposed genetic algorithm, we have included the pseudocode as follows:

Main Code

```
For j = 1:MCMax
    Initialize the first generation of population
    Generate stock price paths
    Get the fitness and sort the individuals by fitness
    For i = 1: MaxGeneration
        Generate new stock paths
        Get New Generation:
            Sort the individuals
            Crossover
                •Keep the elite class unchanged
                •Swap gene of chromosome of normal class
                •Eliminate the last class and fulfill the niches with children of elite
                 class and  normal class
            Mutation: every gene point has 1% probability to mutate
        Restore the best individual’s gene and its fitness
    End for
    Restore the best individual of last generation and restart an independent MC loop
End for
```

How to get fitness

```
By weight and stock path, get the portfolio value path
Calculate the rate of return, sd, mean of max dropdown and mean of lower point
Get the fitness by determined formula, suppose the value = F
If (go bankruptcy at any time):
    Return fitness = -Inf
Else:
    Return fitness = F
End if
```

2.5 Comparison Groups

To compare the performance of the optimal asset allocation strategy (MC Best Chromosome) produced by the Monte Carlos genetic algorithm we designed above, two comparison groups were designed to compare its performance.

Comparison Group 1: Best Chromosome Generated from a Single Long Generation Chain

In this comparison group, instead of using the second layer of Monte Carlos simulation, this comparison group's result is obtained from keeping reproducing the new generations for 3000 times in a single trial. The final best chromosome is the top 1 chromosome survived in the last generation.

Comparison Group 2: Equal Weighted Chromosome

This strategy is one of the common strategies that exist in the real-world market to diversify away individual risk, and it can be interpreted as the investor passively investing in the market index instead of trying to find the optimal asset allocation strategy actively.

2.6 Final Performance Evaluation

The final performance of the three sets of the best chromosome (MC Best Chromosome, Long Generation Chromosome, Equal Weight Chromosome) will be evaluated through independent sets of Stock Market paths. In this project, we generated a set of 50,000 independents market paths and a set of 500,000 independents market paths. Those sets of chromosomes / portfolio strategies would be applied on those sets of market paths, and then final fitness performance of those strategies would be obtained by taking the average of fitness scores of those chromosomes applying on those market paths, that is,

$$MC\ GA\ Performance \;=\; \frac{1}{n} \sum_{i=1}^{n} fitness\ (\text{MC Best Chromosome})\ ,$$

$$\textit{Long GA Performance} \quad = \frac{1}{n}\sum_{i=1}^{n} \textit{fitness}\ (\text{Long Generation Chromosome})\ ,$$

$$\textit{Equal Weight Performance} \quad = \frac{1}{n}\sum_{i=1}^{n} \textit{fitness}\ (\text{Equal Weight Chromosome})$$

where *n* represents the number of different market paths.

3 Code & Results

Since Matlab is more efficient than R in terms of matrices computation and parallel computation, we implemented the proposed methodology in section 2 with Matlab (detailed Matlab code is attached in Appendix).

3.1 Analysis of the Structure of Optimized Portfolios

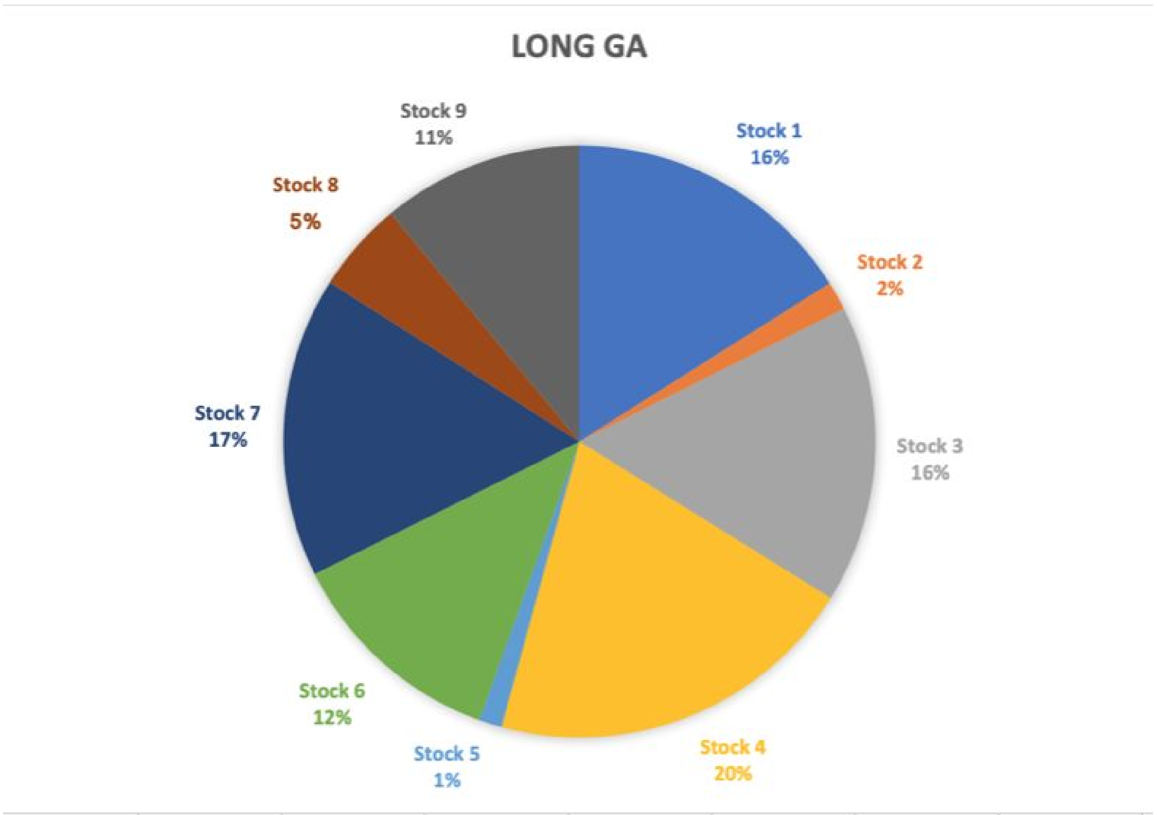
"**Table 1: Weight Distributions**" and "**Chart 3: Pie Charts of Weight Distribution**" below demonstrate the optimal asset allocation strategies generated from "Long GA" and "MC GA Mean" respectively, where

- "Long GA" represents the genetic algorithm which has only one trail but with 3,000 generations (Comparison Group 1)

- "MC GA Mean" represents the proposed genetic algorithm which has 1000 trails, and each trail contains 300 generations. The final chromosome was obtained from taking the average of the best chromosome from the 1000 trials.

Stock	Long GA	MC GA Mean
1	16.03%	19.60%
2	1.57%	1.24%
3	16.27%	13.90%
4	20.38%	18.61%
5	1.24%	1.08%
6	12.12%	13.91%
7	16.43%	16.53%
8	4.97%	4.91%
9	10.99%	10.22%

Table 1: Weights Distributions



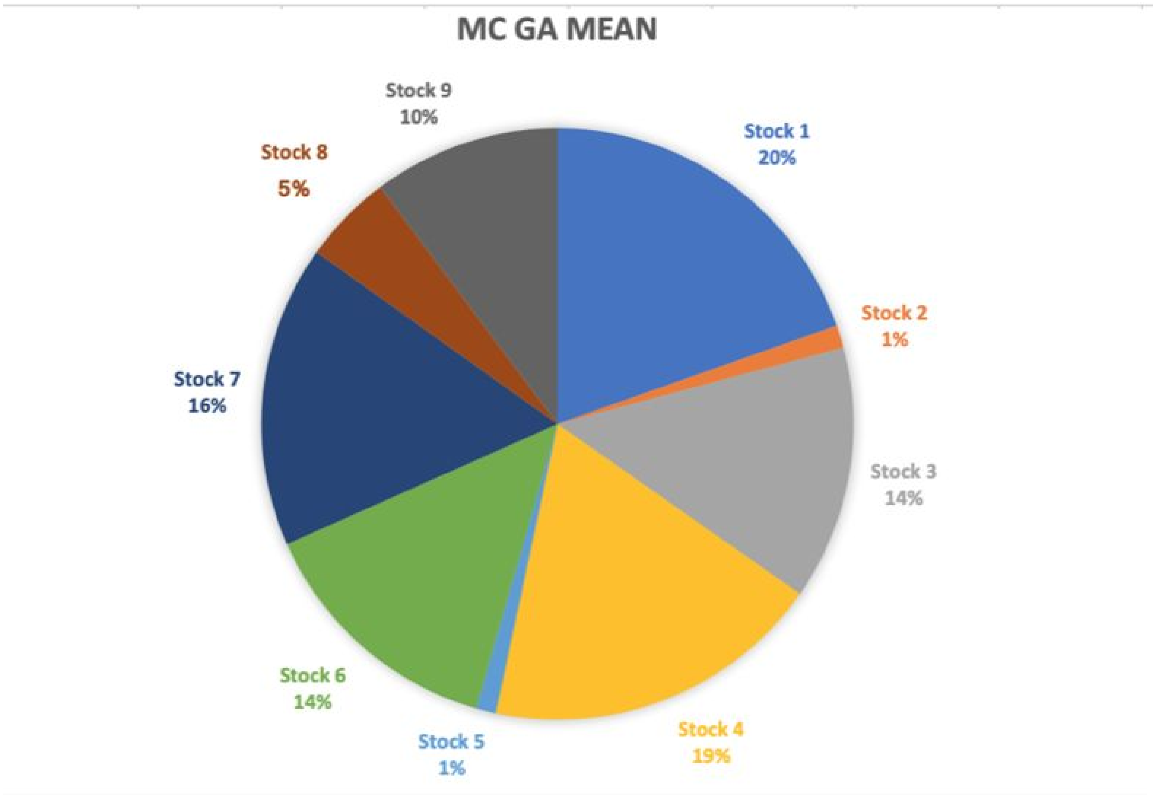


Chart 3: Pie Charts of Weight Distribution

From the above table and charts, we can see that the optimal portfolio strategy generated from "Long GA" and "MC GA Mean" are very similar, which may imply that as long as the length of the generation chain is long enough, the optimal solution achieved in "Long GA" could approach to the optimal result obtained in "MC GA Mean". To verify this hypothesis, we need to further investigate their running behaviours.

3.2 Running Behavior and Convergences

In the following Chart 4: Running Behaviour of Genetic Algorithms, the nine dashed lines are the optimal weights for each stock obtained from our proposed "MC GA Mean", and the nine solid curves represent the optimal weights generated from "Long GA" throughout the generation length.

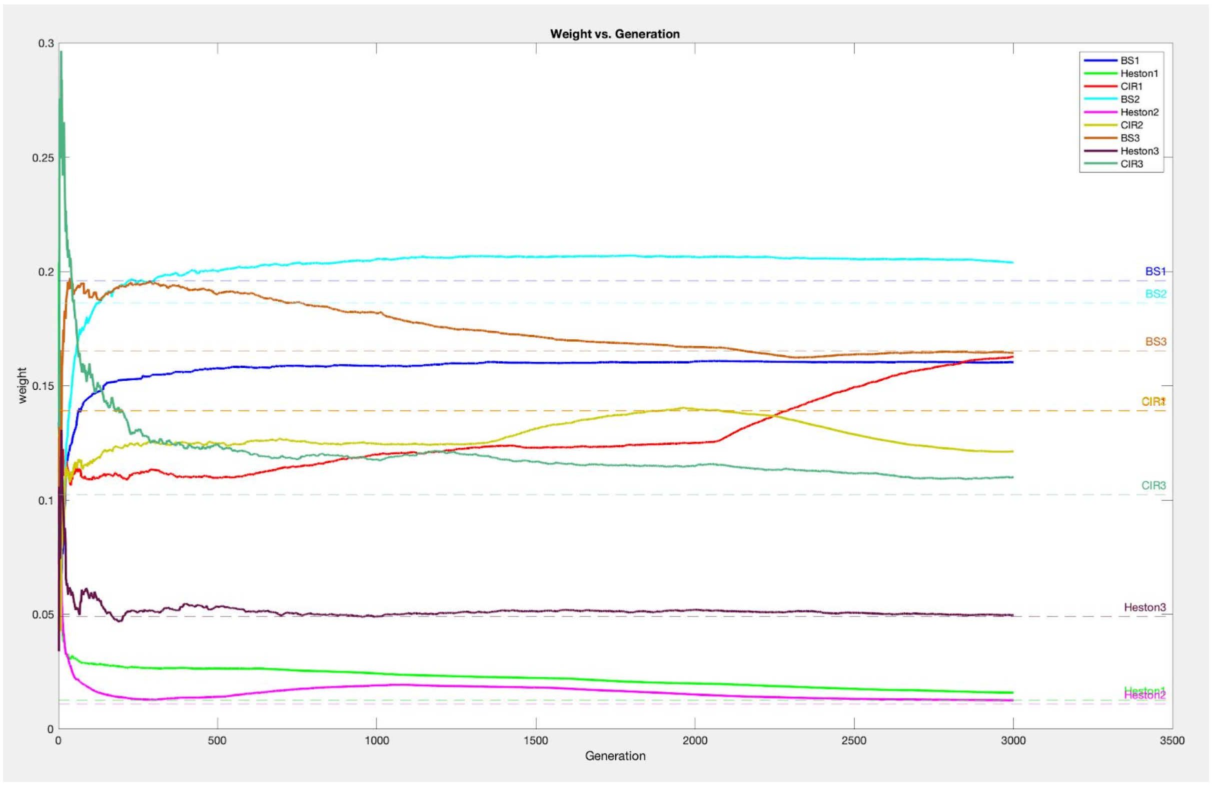


Chart 4: Running Behaviour of Genetic Algorithms

From the chart above, we can see that both "Long GA" and "MC GA Mean" put more weights on the stocks that follow the Black-Scholes (BS) model because those stocks are relatively less volatile which leads to higher Sharpe ratios. At the same time, they both put medium weights on CIR stocks and least weights on Heston stocks respectively.

Notice that as the generation length increases in the "Long GA", the optimal weights allocated in Heston stocks from "Long GA" converges to those optimal weights generated from "MC GA Mean". However, the optimal weights allocated in the BS stocks and CIR stocks from "Long GA" do not have a convergent movement trend towards those optimal weights generated from "MC GA Mean". One of the reasons for such a result is that the fitness function is very flat around the optimum weight distribution for CIR stocks and BS stocks. This resulted "Long GA" only being able to find a near-best solution rather than the best one as "MC GA Mean". This is a piece of evidence which rejects our hypothesis that the solution of "Long GA" converges to the solution of "MC GA Mean" as generation length increases.

3.3 Final Evaluation Results Comparison

Applying the optimal weights generated from "Long GA" , "MC GA Mean", and equal weight strategy to the two sets of independent market movement simulations proposed in Section 2, we obatined the following results :

Method	Average Fitness Score
Equal Weight	0.0338
"Long GA"	0.0423
"MC GA Mean"	0.0424

Table 2: The Average Fitness Scores in 50,000 Simulations

Method	Average Fitness Score
Equal Weight	0.0337
"Long GA"	0.0424
"MC GA Mean"	0.0425

Table 3: The Average Fitness Scores in 500,000 Simulations

From above Table 2 and Table 3, we can see the final evaluation results are almost identical under the two sets of simulation: strategies generated from GA algorithms outperformed the equal weight strategy, while “MC GA Mean” slightly outperformed the “Long GA” by 0.0001. Based on these results, we can also conclude that the fitness evaluation results converge after 50,000 times of market movement simulation.

3.4 Time complexity

Although the final evaluation scores for “Long GA” and “MC GA Mean” are very close, the computation time required for those two genetic algorithms are quite different.

With parameters in those two GAs are fixed, the time required for “Long GA” approximately equals to a constant value. Therefore, the total time consumption for our algorithms (both GAs and Monte Carlo simulation) will be determined by the number of generations in the GAs and the number of Monte Carlo simulations.

If we define:

> N - the number of generations used in our GAs algorithm;
>
> M - the maximum number of Monte Carlo simulations.

Then the time complexity for our algorithm (both GAs and Monte Carlo simulation) is $O(M*N)$, and the time complexity for the “Long GA” is $O(N)$. In the following table, the running time for two different methods in this project are collected and presented.

Method	Actual Time Elapsed	Time complexity
MC GA Mean	39 hours	$O(M*N)$
Long GA	40 min	$O(N)$

Table 4: Time Consumption and Complexity

The differences in the running time are significant. By checking the time complexity, we can see that when $M > N$, the time complexity of MC Fitness $O(M*N) > O(N^2)$. When M is very small (but cannot be close to 0), the time complexity for the two models becomes similar.

Hence, the value of M has a significant impact on time complexity for the "MC GA Mean". In our case, $M = 1000 = ⅓ N$ where $N = 3000$. Therefore, the time complexity of MC Fitness becomes $O(M*N) = O(⅓\ N^2) = O(N^2)$. This explains why it took 39 hours for "MC GA Mean" but only 40 minutes for "Long GA" to run.

4 Conclusion

4.1 Summary and Conclusion

After comparing the results from our proposed genetic algorithm with its comparison groups, we achieved a promising result: by combining genetic algorithms and Monte Carlo simulation, it can generate an optimized portfolio investment strategy that maximizes the desired features (defined by fitness score) by the investors. However, due to its code complexity, it can take a huge amount of time to obtain the optimized portfolio strategy if the Monte Carlo simulation number is large.

In addition, throughout the comparison process, we found that using a genetic algorithm with a long generation chain could achieve a close-to-optimal result. Although its fitness score may not be as good as that from our proposed algorithm (a little bit lower), it could save much more computation time. Hence, under the situation where computational power is limited, a single genetic algorithm trail with a long generation chain would be the better optimizer to choose.

In this project, the portfolio optimization problem is based on several assumptions which simplified the problem. For example, we assumed that the stock market only has nine stocks; we set the risk-free rate to zero; and we did not include VaR in the fitness function. Therefore, there is still a lot of room for further improvement for making the proposed genetic algorithm more applicable in the real financial market. One of our concerns about the proposed Monte Carlos genetic algorithm is that: with more realistic factors added into the model in the future, we wonder if our algorithm still can produce the optimal portfolio strategy in an efficient time manner. This concern would be addressed in research in the future.

4.2 Future Research and Recommendations

For further research, we plan to test our proposed Monte Carlos genetic algorithm under the real financial setups. Instead of assuming a virtual equity market, we could set the S&P 500 index as the stock pool to choose potential stocks. At the same time, because the risk-free rate is not 0 in real life, we should add risk-free assets as investable assets in our portfolio. These setups would allow us to check if the solutio from our algorithm works well in real life. In addition to that, since the complexity of the market model increases with those setups, the efficiency and time complexity should be investigated as well.

Another area we could improve in the future is the comparison group. In this project, we picked a simple alternative investment strategy -- equal weighted portfolio. However, there are many other portfolio optimization algorithms such as mean-variance optimization that exist in the financial market. We should choose another more advanced alternative portfolio optimization algorithm as the comparison group, and see if the optimal strategy generated from our GA can outperform the optimal solution from the comparison group.

5 Bibliography

1. Reeves, C. R. (2010). Genetic algorithms. In *Handbook of metaheuristics* (pp. 109-139). Springer, Boston, MA.
2. Kanungo, R. P. (2004). Genetic algorithms: Genesis of stock evaluation. *Economics WPA Working Paper*, (0404007).
3. Lin, L., Cao, L., Wang, J., & Zhang, C. (2004). The applications of genetic algorithms in stock market data mining optimisation. *Management Information Systems*.
4. Holland, J. H. (1975). Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.
5. Michalewicz, Z., & Schoenauer, M. (1996). Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary computation*, *4*(1), 1-32.
6. Landin, M., & Rowe, R. C. (2013). Artificial neural networks technology to model, understand, and optimize drug formulations. In *Formulation Tools for Pharmaceutical Development* (pp. 7-37). Woodhead Publishing.
7. Simoncini, D., Schiex, T., & Zhang, K. Y. (2017). Balancing exploration and exploitation in population-based sampling improves fragment-based de novo protein structure prediction. *Proteins: Structure, Function, and Bioinformatics*, *85*(5), 852-858.
8. Kumar, N., Sood, D., Tomar, R., & Chandra, R. (2019). Antimicrobial Peptide Designing and Optimization Employing Large-Scale Flexibility Analysis of Protein-Peptide Fragments. *ACS omega*.
9. Kaveh, A., & Kalatjari, V. (2003). Topology optimization of trusses using genetic algorithm, force method and graph theory. *International Journal for Numerical Methods in Engineering*, *58*(5), 771-791.
10. Frank, R. E., Massy, W. F., & Morrison, D. G. (1965). Bias in multiple discriminant analysis. *Journal of Marketing Research*, *2*(3), 250-258.
11. Fogarty, D. J. (2012). Using genetic algorithms for credit scoring system maintenance functions. *International Journal of Artificial Intelligence & Applications*, *3*(6), 1.
12. Drake, A. E., & Marks, R. E. (2002). Genetic algorithms in economics and finance: Forecasting stock market prices and foreign exchange—A review. In *Genetic algorithms and genetic programming in computational finance* (pp. 29-54). Springer, Boston, MA.

6 Appendix

6.1 Main code

```
%initialize parameters
Population = 2000;
%2 percent elites whose gene will not change
EliteRatio = 0.02;
% 70 percent of the population will be eliminated
%also means 28 percent of the population is normal
EliminationRatio = 0.7;
%every gene in chromosome have 1% probability to mutate
MutationRatio = 0.01;
%number of gene in the chromosome
ChromosomeDim = 9;
%extent of mutation
disper = 0.1;
%number of max generation
MaxIter = 300;
%number of MC loop
M = 1000;
%matrix to store result
MCWeight = zeros(ChromosomeDim, M);

%parameters for generating stock prices
T = 20;
%one year has 241 trading days
k = 251 * T;
S0 = 1;
sigma = 0.05;
mu = 0.06;

parfor l = 1:M

        %generate first stock price
        St = ones(k + 1, 9);
        % GenerateBS(T, k, sigma, mu, S0)
        % GenerateHeston(T, k, mu, kappa, theta, volvol, S0, rho, v0)
        % GenerateCIR(T, k, sigma, kappa, theta, voldrift, S0, rho, mu0)
        %S1 S4 S7 are BS, S2 S5 S8 are Heston, S3 S6 S9 are CIR-BS
        for j = 0:2
        St(:, 3 * j + 1) = GenerateBS(T, k, sigma + 0.01 * j, mu + 0.01 * j, 1);
        St(:, 3 * j + 2) = GenerateHeston(T, k, mu + 0.01 * j, 0.1, sigma + 0.01 * j, 1, S0, 0.5 -
 0.1 * j, sigma + 0.01 * j);
        St(:, 3 * j + 3) = GenerateCIR(T, k, sigma + 0.01 * j, 0.1, mu + 0.01 * j, 1, S0, 0.5 - 0.1
 * j, mu + 0.01 * j);
        end

        BestStr = zeros(9, MaxIter + 1);
        %initialize the chromosome
        Chromosome = InitChromo(ChromosomeDim, disper, Population);
        %use generated chromosome get the first fitness
        InitFitness = GetFitness(Chromosome, St);
        %sort individuals by fitness
        [InitFitness, idx] = sortrows(InitFitness, 'descend');
        Chromosome = Chromosome(idx);
        %get the most fitted individual
        BestStr(:, 1) = Chromosome{1};

        for i = 1:MaxIter
        %generate new stock price
        St = ones(k + 1, 9);
        % GenerateBS(T, k, sigma, mu, S0)
        % GenerateHeston(T, k, mu, kappa, theta, volvol, S0, rho, v0)
        % GenerateCIR(T, k, sigma, kappa, theta, voldrift, S0, rho, mu0)
        for j = 0:2
       St(:, 3 * j + 1) = GenerateBS(T, k, sigma + 0.01 * j, mu + 0.01 * j, 1);
        St(:, 3 * j + 2) = GenerateHeston(T, k, mu + 0.01 * j, 0.1, sigma + 0.01 * j, 1, S0, 0.5 -
 0.1 * j, sigma + 0.01 * j);
        St(:, 3 * j + 3) = GenerateCIR(T, k, sigma + 0.01 * j, 0.1, mu + 0.01 * j, 1, S0, 0.5 - 0.1
 * j, mu + 0.01 * j);
        end
        %get new generation
        [Fitness, Chromosome] = GetNewGeneration(Chromosome, Population, EliteRatio,
EliminationRatio, MutationRatio, St);
        %get most fitted individual in new generation
        BestStr(:, i + 1) = Chromosome{1};
        %display results
        if mod(i, 150) == 0 || i == 0
        disp(['for ', num2str(i), '-th loop, Max fitness is ', num2str(Fitness(1)), ', Mean Fitness
is ', num2str(nanmean(Fitness))])
        disp('Best fitted weight is')
        disp(Chromosome{1}')
        end
        end

        %plots
        for i = 1:9
        plot(BestStr(i, :), 'o')
        hold on
        end
        hold off
        MCWeight(:, l) = Chromosome{1};
        disp(l)
end



function [Fitness, New_Chromosome] = GetNewGeneration(Chromosome, Population,
EliteRatio, EliminationRatio, MutationRatio, St)

Fitness = GetFitness(Chromosome, St);

%sort by fitness
[Fitness, idx] = sortrows(Fitness, 'descend');
New_Chromosome = Chromosome(idx);

%determine the number of elite and normal individual
Start = floor(EliteRatio * Population);
Stop = Population - floor(EliminationRatio * Population);

%keep elite gene unchanged
%the normal individuals crossover with themselves
for i = Start:2:Stop
        [New_Chromosome{i}, New_Chromosome{i + 1}] = Crossover(New_Chromosome{i},
New_Chromosome{i + 1}, MutationRatio);
end

%remove all of the bad gene
%use crossover of left gene to fulfill their niches
%it is possible for a individual to crossover itself, such as parthenogenesis
for i = (Stop + 2):Population
        New_Chromosome{i} = Crossover(New_Chromosome{randi(Stop)},
New_Chromosome{randi(Stop)}, MutationRatio);
end

end

function chromosome = InitChromo(InputSize, sigma, Population)
        %only generate the first n-1 dimension, because sum(weight) = 1
        parfor i = 1:Population
        Matrix = randn(InputSize - 1, 1) * sigma;
        temp = 1 - sum(Matrix);
        chromosome{i} = [Matrix; temp];
        end
end

function New_Chromosome = Mutation(Chromosome, MutationRatio)

%when th i-th element of id is 1, the the i-th gene will mutate
id = binornd(1,MutationRatio, [length(Chromosome) - 1, 1]);

%if mutated, the new weight is St = St * (1 + N), N is a random number
Chromosome(1:end - 1) = Chromosome(1:end - 1) .* ( 1 + randn(length(Chromosome) - 1,
1).* id);

New_Chromosome = [Chromosome(1:end - 1); 1 - sum(Chromosome)];

end

function [Child1, Child2] = SwapChromosome(Father, Mother, point)
        Child1 = Father;
        Child2 = Mother;
        %swap the tail of father and mother
        Child1(1 : point) = Mother(1 : point);
        Child2(1 : point) = Father(1 : point);
end

function [Child1, Child2, point] = Crossover(Father, Mother, MutationRatio)
        RoundedFather = Father(1 : end - 1);
        RoundedMother = Mother(1 : end - 1);

        %choose the point to swap chromosome
        point = randi(length(RoundedFather) - 1);

        [Child1, Child2] = SwapChromosome(RoundedFather, RoundedMother, point);

        %mutation
        Child1 = Mutation(Child1, MutationRatio);
        Child2 = Mutation(Child2, MutationRatio);

        %determine the last dimension of weight
        temp = 1 - sum(Child1);
        Child1 = [Child1; temp];
        temp = 1 - sum(Child2);
        Child2 = [Child2; temp];
end

function Fitness = GetFitness(Chromosome, St)

        %determine parameters
        penalty = 0.01;
        w = 0.3;

        for i = 1:length(Chromosome)
        %get portfolio value vs time
        Portfolio =  St * Chromosome{i};
        %take rate of return, sd, max dropdown and lower point as our
        %indexs
        [RoR, SD, MaxDropdown, lowerpoint] = GetPara(1, Portfolio);

        if RoR > 0
        %we set risk free rate = 0 so RoR / SD is sharpe ratio
```

```
            %if there is a shorted stock, there is 1% penalty in fitness
            Fitness(i) = RoR / SD - w * MaxDropdown + (1 - w) * lowerpoint - penalty *
sum(Chromosome{i} < 0);

            else

            Fitness(i) = RoR * SD - w * MaxDropdown + (1 - w) * lowerpoint - penalty *
sum(Chromosome{i} < 0);
        end
            end

            %for multiply of matrix
            Fitness = Fitness';

end

%we use Euler's scheme to generate stock price
function St = GenerateBS(T, k, sigma, mu, S0)

dt = T / k;
dWt = randn(k, 1) * sqrt(dt);
St = S0 * ones(1, 1 + k);

for i = 1:k
    St(i + 1) = St(i) + St(i) * mu * dt + St(i) * sigma *dWt(i);
end

end

function St = GenerateHeston(T, k, mu, kappa, theta, volvol, S0, rho, v0)

dt = T / k;
dWst = randn(k, 1) * sqrt(dt);
dWvt = sqrt(1 - rho^2) * randn(k, 1) * sqrt(dt) + rho * dWst;

St = S0 * ones(1, 1 + k);
vt = v0 * ones(1, 1 + k);

for i = 1:k
```

```
        vt(i + 1) = max(0, vt(i) + kappa * (theta - vt(i)) * dt + volvol * sqrt(vt(i)) * dWvt(i));
        St(i + 1) = St(i) + St(i) * mu * dt + St(i) * sqrt(vt(i)) *dWst(i);
end

end

function St = GenerateCIR(T, k, sigma, kappa, theta, voldrift, S0, rho, mu0)

dt = T / k;
dWst = randn(k, 1) * sqrt(dt);
dWmut = sqrt(1 - rho^2) * randn(k, 1) * sqrt(dt) + rho * dWst;

St = S0 * ones(1, 1 + k);
mut = mu0 * ones(1, 1 + k);

for i = 1:k
        mut(i + 1) = max(0, mut(i) + kappa * (theta - mut(i)) * dt + voldrift * sqrt(mut(i)) *
dWmut(i));
        St(i + 1) = St(i) + St(i) * mut(i) * dt + St(i) * sigma *dWst(i);
end

end

function [RoR, SD, MaxDropdown, lowerpoint] = GetPara(T, St)

%use log return
logSt = log(St);
RoR = (logSt(end) - logSt(1)) / T;
SD = sqrt(var(logSt) * length(St) / T);

for i = 1:T
        yrdrop = - min(diff(logSt((251 * (i-1) + 1):251 * i)));
        MaxDropdown(i) = max(0, yrdrop);
        yrlower = min(logSt((251 * (i-1) + 1):251 * i) - logSt(251 * (i-1) + 1));
        lowerpoint(i) = min(0, yrlower);
end

%mean of every year's data
MaxDropdown = mean(MaxDropdown);
```

```
lowerpoint = mean(lowerpoint);

%if the portfolio value is negative, it goes to bankruptcy (too high leverage)
if sum(St < 0) > 0
        RoR = -inf; SD = 1; MaxDropdown = 0; lowerpoint = 0;
end

end
```

Published with MATLAB® R2019a

6.2 Get plots

```
%'.mat' files need to save by hand
load('SingleSimulation.mat');
load('MCWeight300.mat');
% MCWeightshort = load('MCWeight.mat');
% MCWeightshort = MCWeightshort.MCWeight;
%number of MC loop
MaxIter = 5000;
T = 20;
k = 251 * T;
S0 = 1;
sigma = 0.05;
mu = 0.06;

%get time mean of 1 loop with many generation
Weight = BestStr;
for i = 1:3001
        Weight(:, i) = mean(BestStr(:, 1:i), 2);
end
```

```
% get the mean weight of MC
MCMean = mean(MCWeight, 2);
% MCMeanShort = mean(MCWeightshort, 2);

Legend = {'BS1', 'Heston1', 'CIR1', 'BS2', 'Heston2', 'CIR2', 'BS3', 'Heston3', 'CIR3'};
colour = {'blue', 'green', 'red', 'cyan', [1, 0, 1], [0.8, 0.8, 0], [0.8, 0.4, 0.1], [0.4, 0.1, 0.3], [0.3,
0.7, 0.5]};

for i = 1:9
plot(Weight(i,:), 'linewidth' , 2, 'color', colour{i})
hold on
end

for i = 1:9
        yline(MCMean(i), '--', Legend{i}, 'color', colour{i})
end

hold off
legend('BS1', 'Heston1', 'CIR1', 'BS2', 'Heston2', 'CIR2', 'BS3', 'Heston3', 'CIR3')

% get the weight of different method
EqualWeight = 1/9 * ones(9, 1);
BestWeight = Weight(:, end);
MCWeight = MCMean;

%initialize the matrix to restore fitness simulation
EqualFitness = zeros(1, MaxIter);
MCFitness = zeros(1, MaxIter);
BestFitness = zeros(1, MaxIter);

%MC to get fitness of different weight
parfor i = 1: MaxIter
        St = ones(k + 1, 9);
        % GenerateBS(T, k, sigma, mu, S0)
        % GenerateHeston(T, k, mu, kappa, theta, volvol, S0, rho, v0)
        % GenerateCIR(T, k, sigma, kappa, theta, voldrift, S0, rho, mu0)
        for j = 0:2
        St(:, 3 * j + 1) = GenerateBS(T, k, sigma + 0.01 * j, mu + 0.01 * j, 1);
        St(:, 3 * j + 2) = GenerateHeston(T, k, mu + 0.01 * j, 0.1, sigma + 0.01 * j, 1, S0, 0.5 -
 0.1 * j, sigma + 0.01 * j);
        St(:, 3 * j + 3) = GenerateCIR(T, k, sigma + 0.01 * j, 0.1, mu + 0.01 * j, 1, S0, 0.5 - 0.1
 * j, mu + 0.01 * j);
        end
        EqualFitness(i) = GetFitness(EqualWeight, St);
        BestFitness(i) = GetFitness(BestWeight, St);
        MCFitness(i) = GetFitness(MCWeight, St);
        if mod(i, 5000) == 0
        disp(i)

        end
end

%display results
mean(EqualFitness)
mean(BestFitness)
mean(MCFitness)

function Fitness = GetFitness(Chromosome, St)

        penalty = 0.01;
        w = 0.3;

        Portfolio =  St * Chromosome;
        [RoR, SD, MaxDropdown, lowerpoint] = GetPara(1, Portfolio);

        if RoR > 0
        Fitness = RoR / SD - w * MaxDropdown + (1 - w) * lowerpoint - penalty *
sum(Chromosome < 0);
        else
        Fitness = RoR * SD - w * MaxDropdown + (1 - w) * lowerpoint - penalty *
sum(Chromosome < 0);
        end

        Fitness = Fitness';

end

function St = GenerateBS(T, k, sigma, mu, S0)
```

```
dt = T / k;
dWt = randn(k, 1) * sqrt(dt);
St = S0 * ones(1, 1 + k);

for i = 1:k
        St(i + 1) = St(i) + St(i) * mu * dt + St(i) * sigma *dWt(i);
end

end

function St = GenerateHeston(T, k, mu, kappa, theta, volvol, S0, rho, v0)

dt = T / k;
dWst = randn(k, 1) * sqrt(dt);
dWvt = sqrt(1 - rho^2) * randn(k, 1) * sqrt(dt) + rho * dWst;

St = S0 * ones(1, 1 + k);
vt = v0 * ones(1, 1 + k);

for i = 1:k
        vt(i + 1) = max(0, vt(i) + kappa * (theta - vt(i)) * dt + volvol * sqrt(vt(i)) * dWvt(i));
        St(i + 1) = St(i) + St(i) * mu * dt + St(i) * sqrt(vt(i)) *dWst(i);
end

end

function St = GenerateCIR(T, k, sigma, kappa, theta, voldrift, S0, rho, mu0)

dt = T / k;
dWst = randn(k, 1) * sqrt(dt);
dWmut = sqrt(1 - rho^2) * randn(k, 1) * sqrt(dt) + rho * dWst;

St = S0 * ones(1, 1 + k);
mut = mu0 * ones(1, 1 + k);

for i = 1:k
        mut(i + 1) = max(0, mut(i) + kappa * (theta - mut(i)) * dt + voldrift * sqrt(mut(i)) *
dWmut(i));
        St(i + 1) = St(i) + St(i) * mut(i) * dt + St(i) * sigma *dWst(i);
end
```

```
end

function [RoR, SD, MaxDropdown, lowerpoint] = GetPara(T, St)

logSt = log(St);
RoR = (logSt(end) - logSt(1)) / T;
SD = sqrt(var(logSt) * length(St) / T);

for i = 1:T
        yrdrop = - min(diff(logSt((251 * (i-1) + 1):251 * i)));
        MaxDropdown(i) = max(0, yrdrop);

        yrlower = min(logSt((251 * (i-1) + 1):251 * i) - logSt(251 * (i-1) + 1));
        lowerpoint(i) = min(0, yrlower);
end

MaxDropdown = mean(MaxDropdown);
lowerpoint = mean(lowerpoint);

if sum(St < 0) > 0
        RoR = -inf; SD = 1; MaxDropdown = 0; lowerpoint = 0;
end

end
```

Published with MATLAB® R2019a