22th January 2024

M2 IRFA

Project C++ : Genetic Algorithm

Made By Naly Razafiarifera Andrianjanahary, Karl Galle, Rayan Sassi and David Lion Cerf.

UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

Introduction

Genetic algorithms (GAs) represent a class of optimization algorithms based on principles inspired by evolutionary biology. These approaches draw on the natural selection process of genetics to solve complex optimization problems. This type of algorithm is used in several fields such as routing and scheduling problems, machine learning, traffic engineering, etc. However, in our case, we will focus on the field of finance, particularly in optimizing financial portfolios.

Portfolio optimization aims to determine the optimal allocation of financial assets according to the chosen fitness criterion (maximize the Sharpe ratio, minimize variance, maximize returns, etc.). This task often faces complex challenges due to the diversity and large number of assets in the market, risk management constraints, and financial market fluctuations.

Genetic algorithms are particularly effective in this context by simulating evolutionary processes such as reproduction, natural selection, and mutation. Potential solutions, represented as genomes, undergo genetic transformations over generations to converge towards optimal portfolio configurations.

In this framework, our study explores the application of genetic algorithms to portfolio optimization. Using evolutionary concepts, these algorithms offer a flexible and powerful approach to search for effective solutions, adapted to the dynamic conditions of financial markets. We will examine how these algorithms can be adapted and integrated into the financial decision-making process to create robust and high-performing portfolios. Finally, we will determine the best way to generate new portfolios based on a given evaluation criterion.

Structure of the code

The code is structured into several parts, each playing a specific role in the portfolio optimization process.

1. Dataset

To test our maximization algorithm, we used a dataset comprising of daily returns of 10 stocks over 254 days. The dataset is stored in a .txt file, where each column corresponds to the daily returns of an individual stock. The data originates from the top 10 stocks in the SBF120 index based on Market Capitalization as of December 2019.

Stock 1	Stock 2	Stock 3	Stock 4	Stock 5	Stock 6	Stock 7	Stock 8	Stock 9	Stock 10
-0.038097118	-0.008573572	-0.012024048	-0.009814324	-0.035166924	-0.054865938	-0.028475712	-0.005884610	-0.009485095	-0.018378614
0.030781859	0.024970273	0.014198783	0.009107956	0.048146780	0.043866562	0.029741379	0.048513705	0.010487916	0.027844845
-0.001791758	-0.002320186	-0.012750000	-0.019909743	-0.018327068	-0.002516356	-0.011720385	-0.001472754	-0.001805054	-0.005605324
0.031312325	-0.002114165	0.006077488	0.007854821	0.036979416	0.035570131	0.016306650	0.005039331	0.005424955	0.001409488
0.011603172	0.013347458	-0.008054367	-0.003896802	0.034968263	0.023386114	0.023546572	-0.002690473	0.004496403	-0.013602438
-0.009176066	-0.004808697	0.005582340	0.000674491	-0.000780553	-0.034991669	-0.002646580	0.010790926	0.003581021	-0.010936187
-0.003279954	-0.017121849	-0.001009336	-0.000674036	-0.013726147	0.001233350	0.013268014	0.003760767	0.001338091	-0.007212400

Representation of the dataset

The first part of the code consists of reading the data contained in the .txt file and storing it in a matrix of type MatrixXd, which will serve as the basis for subsequent calculations. The Eigen library is used for matrix manipulation operations, thus offering optimal performance in the matrix calculations necessary for portfolio optimization.

2. Portfolio class

The Portfolio class represents an investment portfolio. It encapsulates data related to the portfolio, including the return matrix, covariance matrix, stock weights, and

methods necessary for various calculations. It will be particularly useful for the creation of new individuals.

- The main features of the Portfolio class include:
- Calculation of covariance between stocks.
- Normalization of portfolio weights.
- Calculation of the portfolio's standard deviation
- Calculation of expected returns.
- Calculation of the Sharpe ratio as a measure of portfolio performance.
- Mutation operations of portfolio weights.

3. Fitness function

The fitness function is the function used to evaluate a portfolio's performance. Thus, one can say that portfolio A is better than portfolio B if fitness(A) > fitness(B).

a. Sharpe ratio

There are many evaluation functions, but in our case, as a first step we preferred to choose the function that calculates the Sharpe ratio. Indeed, we considered it a solid approach for portfolio optimization, as it takes into account risk-adjusted returns.

$$Sharpe\,ratio = (Rp - Rf) / \sigma p$$

where:
Rp is the return of the portfolio
Rf is the risk free rate
σp is the volatility of the portfolio

```
double fitness(){

    // Sharpe Ratio
    double _return = this-> mean_returns();
    this->std_dev();
    return (_return - RISK_FREE_RATE) / portfolio_std;
}
```

The optimization problem therefore comes down to calculating the optimal weights for a portfolio with the highest sharpe ratio. Moreover, here, we chose a risk-free rate of 0% as it was a period where the interest rate was really low or negative.

b. Volatility

Secondly, to confirm or refute our results with the previous fitness function, we will use another function. The volatility of the portfolio returns.

```
    // compute the standard deviation of the portfolio
    double std() {
        this->cov();
        // We return the opposite of the standard deviation to turn it into a maximization problem
        return - sqrt(weights.transpose() * covMatrix * weights) * sqrt(254);
}
```

Our optimisation problem therefore comes down to calculating the optimal weights for a portfolio with the lowest variance.

4. Crossover functions

A crossover function is used to combine the genetic information of two parents to create new offspring. There are several methods of crossover, but the general idea is to split the genes of the parents at a certain point and exchange segments between the parents to form offspring.

For our study, we created 3 crossover functions and will compare the results obtained after presenting the code structure.

a. NaiveCrossover

```
Portfolio NaiveCrossover(Portfolio parent1, Portfolio parent2){

    Portfolio child;
    int size = parent1.getWeights().size();
    int half_size = size / 2;

    // Initialize an empty vector of size 10
    VectorXd child_weights(size);

    // Child receives one half from parent1 and the other half from parent2
    child_weights.head(half_size) = parent1.getWeights().head(half_size);
    child_weights.tail(half_size) = parent2.getWeights().tail(half_size);

    child.setWeights(child_weights);

    return child;

}
```

This code implements a "Naive" crossover function because it creates a child that takes half of the genes from parent 1 and the other half from parent 2.

b. Blendcrossover

```
Portfolio blendCrossover(Portfolio parent1, Portfolio parent2, double alpha = 0.7) {

    if (parent1.getWeights().size() != parent2.getWeights().size()) {
        cerr << "Error: Parents have different number of weights." << endl;
        exit(1);
    }


    Portfolio child;

    // If parent1 has better performance than parent2 the child inherits more weights from it.
    if (parent1.fitness() >= parent2.fitness()) {

    VectorXd child_weights = alpha * parent1.getWeights() + (1.0 - alpha) * parent2.getWeights();
    child.setWeights(child_weights);
    }

    VectorXd child_weights = alpha * parent2.getWeights() + (1.0 - alpha) * parent1.getWeights();
    child.setWeights(child_weights);

    return child;
}
```

This function takes a different approach. The blendCrossover function in this context combines the weights of the two parents using an alpha weighting where the sum is done for each element of the vector. If the performance of the first parent is better or equal to that of the second parent, the child's weights are obtained by weighing parent1's weighs more heavily. Otherwise, the child's weights are obtained by weighting parent2's weighs more heavily. The weighting is controlled by the alpha parameter. In the figure above, alpha equals 0.7.

c. TwoPointCrossover

```
Portfolio TwoPointCrossover(Portfolio parent1, Portfolio parent2) {
    Portfolio child;

    int size = parent1.getWeights().size();

    VectorXd child_weights(size);

    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<int> distribution(0, size - 1);

    int crossover_point1 = distribution(gen);
    int crossover_point2 = distribution(gen);

    // Ensure crossover_point1 is less than crossover_point2
    if (crossover_point1 > crossover_point2) {
        std::swap(crossover_point1, crossover_point2);
    }

    // Copy genetic material from parents to child
    child_weights.segment(0, crossover_point1) = parent1.getWeights().segment(0, crossover_point1);
    child_weights.segment(crossover_point1, crossover_point2 - crossover_point1 + 1) = parent2.getWeights().segment(crossover_point1, crossover_point2 - crossover_point1 + 1);
    child_weights.segment(crossover_point2 + 1, size - crossover_point2 - 1) = parent1.getWeights().segment(crossover_point2 + 1, size - crossover_point2 - 1);

    child.setWeights(child_weights);

    return child;
}
```

In this function, two crossover points are randomly chosen among the weights of the parents. Then, a check is made to ensure that the two crossover points are different, thus guaranteeing genetic diversity in the offspring. The portions of weights between the two crossover points are exchanged between the parents, creating the child's weights.

A new Portfolio object representing the child is created, and its weights are set according to the result of the crossover. In summary, this two-point crossover function combines the genetic information of two parental portfolios to create a new investment portfolio, exchanging portions of weights between two specific points. This method aims to introduce genetic variability into the population of individuals generated by the genetic algorithm.

5. Genetic algorithm

The main loop of the genetic algorithm is responsible for the evolution of the portfolio population over several generations, here's how it works.

The program begins by defining certain numerical parameters, including the population size (POPULATION_SIZE), the total number of generations (NUM_GENERATIONS), the mutation rate (MUTATION_RATE), and the elitism percentage (ELITISM_PERCENTAGE). These parameters influence the functioning of the algorithm.

```
int main() {
    const int POPULATION_SIZE = 100;
    const int NUM_GENERATIONS = 100;
    const double MUTATION_RATE = 0.01;
    //Percentage of the best portfolios to keep
    const double ELITISM_PERCENTAGE = 0.1;
```

The process starts by initializing a population of 100 portfolios, each characterized by random weights assigned to different financial assets. The main loop represents successive generations, evaluating the performance of each portfolio and identifying the best at regular intervals (every 10 generations). Detailed information about the best portfolio such as its Sharpe ratio and its weightings as well as the average Sharpe ratio of its generation are displayed.

The creation of each new generation involves preserving an elite consisting of the top 10% portfolios from the previous generation. The remaining 90% of the new generation are generated through random crossover and mutation. The crossover is performed by randomly choosing two parents from the population, while mutation can occur with a probability of 1%. This percentage of one percent was chosen in regards with mutation rate in the DNA where it is generally around 10-4 and 10-11. Introducing a mutation rate too high might interfere with the stability of a solution leading to performance decrease.

This loop repeats for 100 generations, ultimately providing an evolved population of 100 portfolios that should represent potentially more effective solutions for the investment portfolio optimization problem.

Outputs analysis

1. Maximizing the Sharpe ratio

Thus, depending on the chosen crossover function, we obtain different optimal portfolios. Therefore, we can compare the average Sharpe ratio as well as that of the best portfolio of each generation according to the crossover function used.

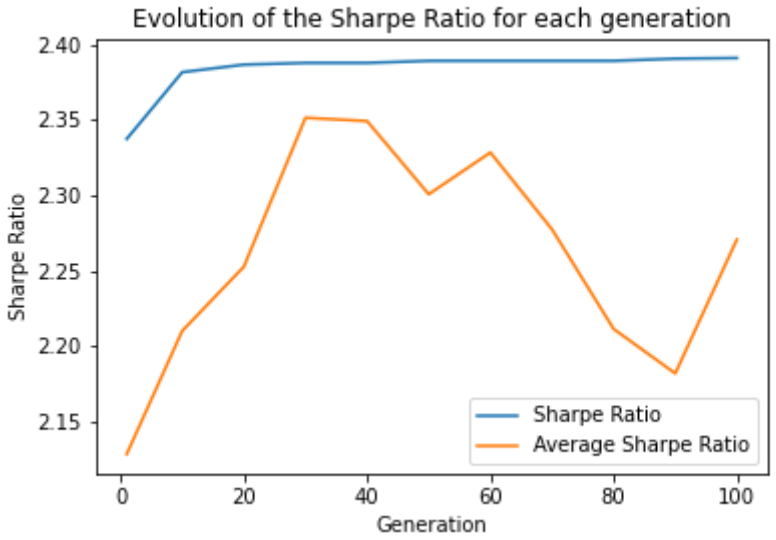
a. NaiveCrossover

```
Generation 1 - Best Portfolio:
Weights:
  0.138197
0.00510163
   0.15692
 0.0250322
 0.0796469
  0.105915
  0.138533
  0.133333
 0.0787299
  0.138591
Sharpe Ratio of the best portfolio: 2.3375
Average Sharpe Ratio: 2.1286
```

```
Generation 50 - Best Portfolio:
Weights:
 0.292187
0.0267601
0.0518194
0.0240428
 0.161396
0.0165479
 0.119232
0.0801049
0.0931993
  0.13471
Sharpe Ratio of the best portfolio: 2.38924
Average Sharpe Ratio: 2.30084
```

```
Generation 100 - Best Portfolio:
Weights:
 0.300464
0.0275181
0.0532872
0.0247239
 0.165968
0.0159604
 0.114999
0.0772612
0.0898907
 0.129928
Sharpe Ratio of the best portfolio: 2.3911
Average Sharpe Ratio: 2.27083
```

Best portfolio's weights and sharpe ratio and average of the generation



Evolution of the Sharpe ratio of the best portfolio and the average Sharpe ratio for each generation

	Average of the Sharpe ratio over the 100 generations
Best portfolio	2,38312
Generation	2,265225

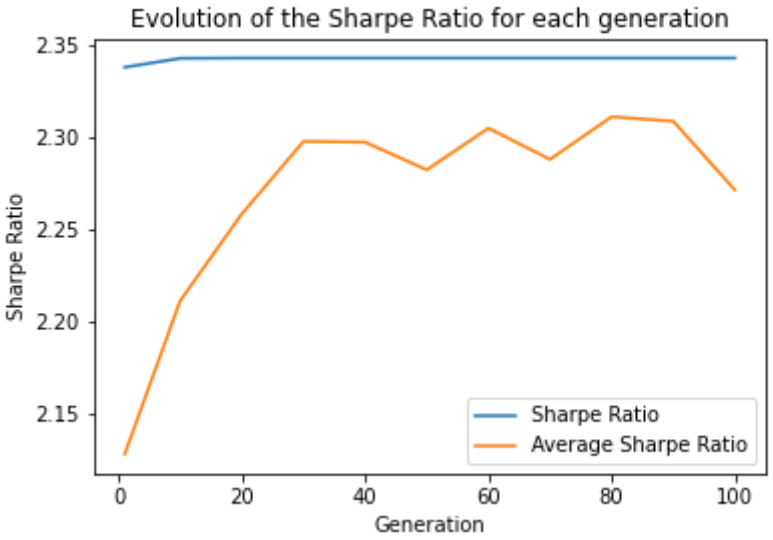
b. Blendcrossover

```
Generation 1 - Best Portfolio:
Weights:
  0.138197
0.00510163
   0.15692
 0.0250322
 0.0796469
  0.105915
  0.138533
  0.133333
 0.0787299
  0.138591
Sharpe Ratio of the best portfolio: 2.3375
Average Sharpe Ratio: 2.1286
```

```
Generation 50 - Best Portfolio:
Weights:
 0.120877
0.0182419
 0.156924
  0.03372
 0.115912
0.0957555
 0.148538
 0.115205
0.0799104
 0.114915
Sharpe Ratio of the best portfolio: 2.34249
Average Sharpe Ratio: 2.28211
```

```
Generation 100 - Best Portfolio:
Weights:
 0.120877
0.0182419
 0.156924
  0.03372
 0.115912
0.0957555
 0.148538
 0.115205
0.0799104
 0.114915
Sharpe Ratio of the best portfolio: 2.34249
Average Sharpe Ratio: 2.27124
```

Best portfolio's weights and sharpe ratio and average of the generation



Evolution of the Sharpe ratio of the best portfolio and the average Sharpe ratio for each generation

	Average of the Sharpe ratio over the 100 generations
Best portfolio	2,341976
Generation	2,2647

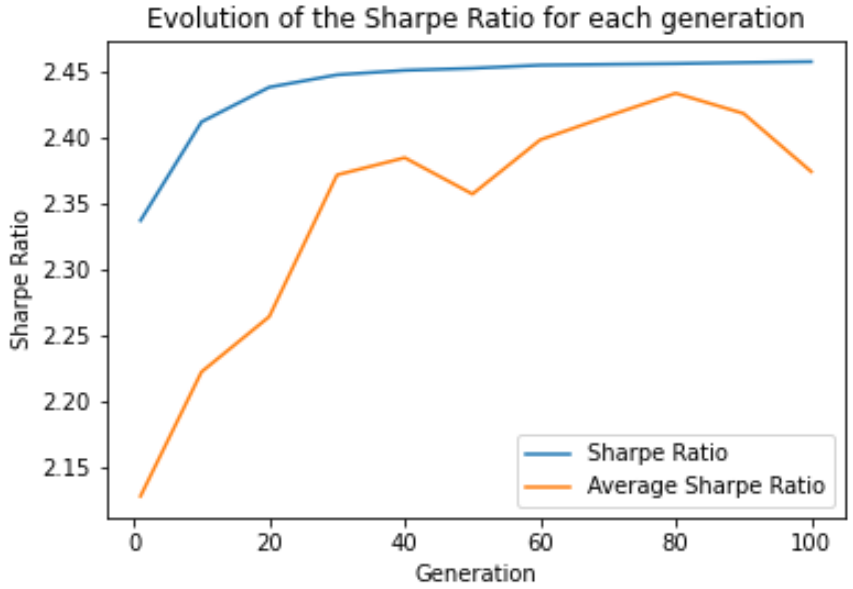
c. TwoPointCrossover

```
Generation 1 - Best Portfolio:
Weights:
  0.138197
0.00510163
   0.15692
 0.0250322
 0.0796469
  0.105915
  0.138533
  0.133333
 0.0787299
  0.138591
Sharpe Ratio: 2.3375
Average Sharpe Ratio: 2.1286
```

```
Generation 50 - Best Portfolio:
Weights:
   0.25846
0.00461281
  0.138739
 0.0089484
  0.240248
0.00794283
  0.114661
 0.0818842
  0.021635
   0.12287
Sharpe Ratio: 2.45289
Average Sharpe Ratio: 2.35757
```

```
Generation 100 - Best Portfolio:
Weights:
  0.255698
 0.0019634
  0.135342
0.00850297
  0.235023
0.00249628
  0.117884
 0.0841859
 0.0150271
  0.143878
Sharpe Ratio: 2.4579
Average Sharpe Ratio: 2.37457
```

Best portfolio's weights and sharpe ratio and average of the generation

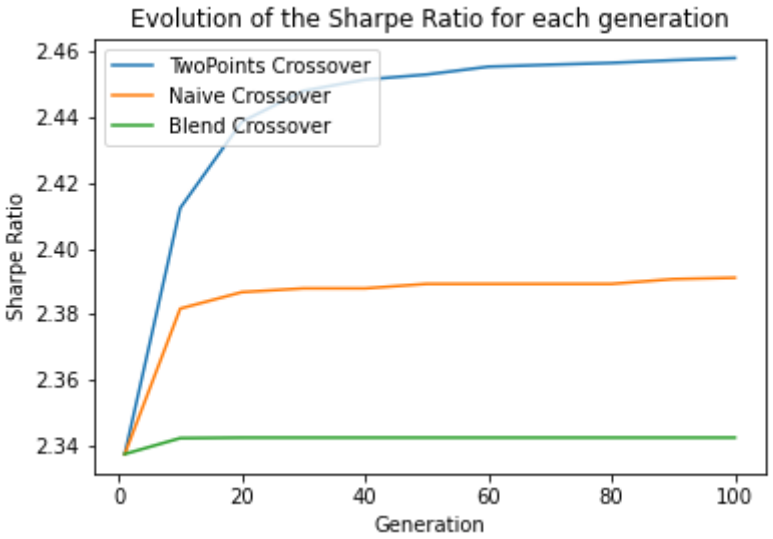


Evolution of the Sharpe ratio of the best portfolio and the average Sharpe ratio for each generation

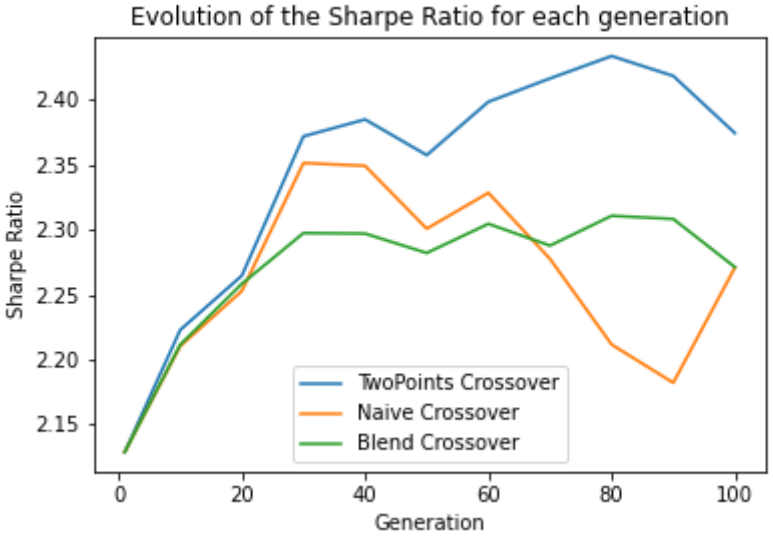
	Average of the Sharpe ratio over the 100 generations
Best portfolio	2,437414444
Generation	2,348515556

d. Comparison

We can synthesize these results through two graphs showing the evolution of the Sharpe ratio of the best portfolio and the average Sharpe ratio for each generation.



Evolution of the Sharpe ratio of the best portfolio for each generation



Evolution of the averaged Sharpe ratio for each generation

It seems that the TwoPoints crossover function is better than the other two for our optimization problem.

2. Minimizing the volatility

To validate our results, we are going to change the fitness function. To do this, we will change the optimization problem and try to minimize the portfolio variance. The code remains unchanged, only the fitness function is different.
Rerunning the program gives us the following results:

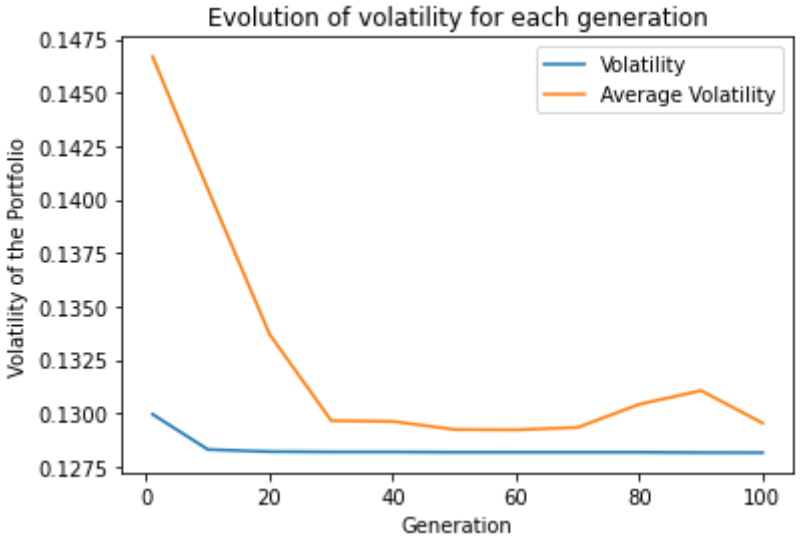
a. NaiveCrossover

```
Generation 1 - Best Portfolio:
Weights:
 0.0151709
  0.173452
   0.15963
  0.149749
0.00882738
 0.0261874
   0.14501
 0.0862922
  0.140075
 0.0956054
Volatility of the best portfolio: 0.129956
Average Volatility: 0.146694
```

```
Generation 50 - Best Portfolio:
Weights:
0.00377139
    0.23132
 0.0868981
  0.135684
 0.0631298
 0.0103506
 0.0892333
 0.0354313
  0.108141
  0.236041
Volatility of the best portfolio: 0.128175
Average Volatility: 0.129238
```

```
Generation 100 - Best Portfolio:
Weights:
0.0038581
 0.236638
 0.088896
 0.138804
0.0645812
 0.010092
0.0870036
0.0345459
 0.105438
 0.230143
Volatility of the best portfolio: 0.128155
Average Volatility: 0.129536
```

Best portfolio's weights and volatility and average of the generation



Evolution of the volatility of the best portfolio and the average volatility for each generation

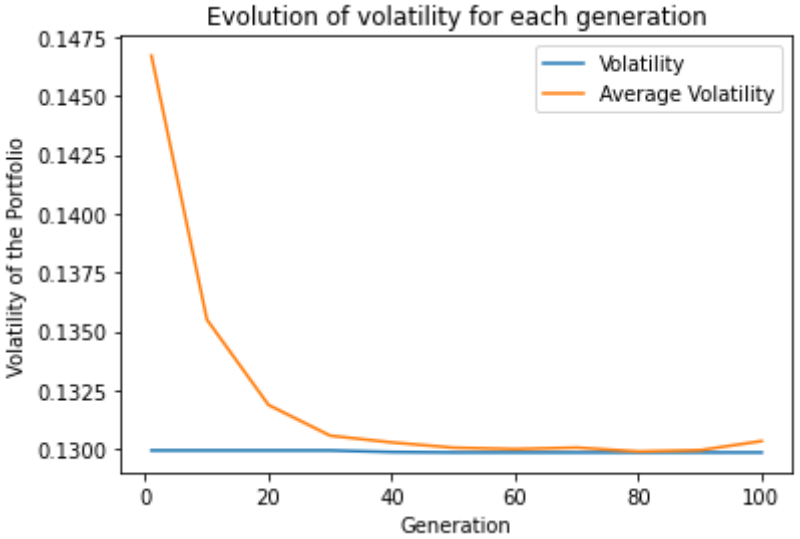
b. Blendcrossover

```
Generation 1 - Best Portfolio:
Weights:
 0.0151709
  0.173452
   0.15963
  0.149749
0.00882738
 0.0261874
   0.14501
 0.0862922
  0.140075
 0.0956054
Volatility of the best portfolio: 0.129956
Average Volatility: 0.146694
```

```
Generation 50 - Best Portfolio:
Weights:
 0.014483
 0.194835
 0.147887
 0.139781
0.0096997
0.0249821
 0.135737
0.0815719
 0.160495
 0.090528
Volatility of the best portfolio: 0.129871
Average Volatility: 0.130073
```

```
Generation 100 - Best Portfolio:
Weights:
 0.0143553
  0.197924
  0.146483
  0.138482
0.00964743
 0.0247629
  0.134486
  0.080841
  0.163298
 0.0897212
Volatility of the best portfolio: 0.12987
Average Volatility: 0.130354
```

Best portfolio's weights and volatility and average of the generation



Evolution of the volatility of the best portfolio and the average volatility for each generation

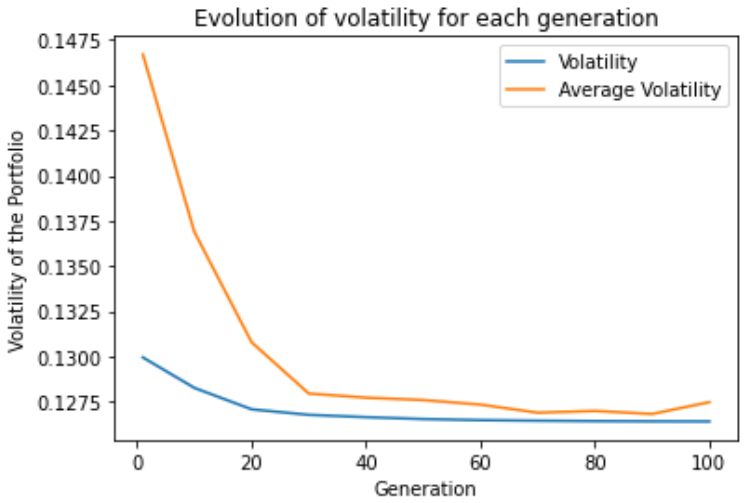
c. TwoPointCrossover

```
Generation 1 - Best Portfolio:
Weights:
 0.0151709
  0.173452
   0.15963
  0.149749
0.00882738
 0.0261874
   0.14501
 0.0862922
  0.140075
 0.0956054
Volatility of the best portfolio: 0.129956
Average Volatility: 0.146694
```

```
Generation 50 - Best Portfolio:
Weights:
0.000703689
   0.248879
   0.104098
   0.158796
 0.00939379
 0.00040625
   0.117181
  0.0290936
   0.122432
   0.209016
Volatility of the best portfolio: 0.126551
Average Volatility: 0.127602
```

```
Generation 100 - Best Portfolio:
Weights:
 0.00019451
   0.260276
  0.0961489
   0.170758
 0.00194869
0.000188278
   0.127707
  0.0180646
    0.11991
   0.204804
Volatility of the best portfolio: 0.12642
Average Volatility: 0.127484
```

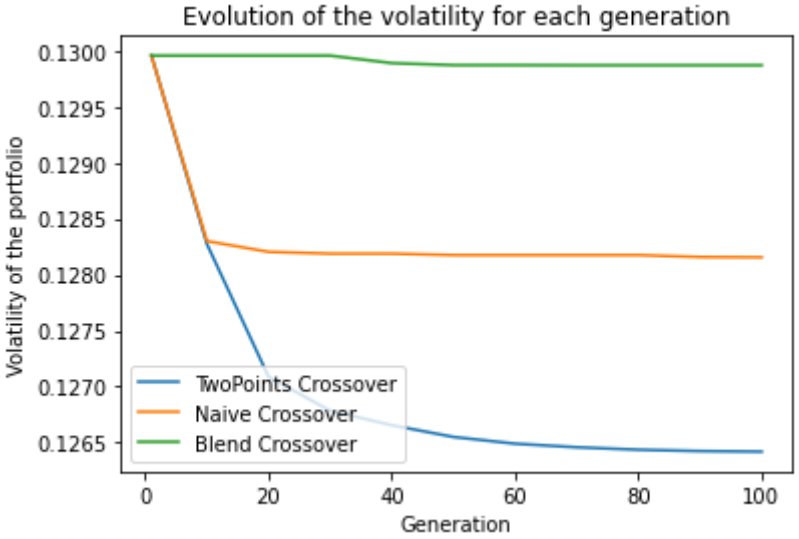
Best portfolio's weights and volatility and average of the generation



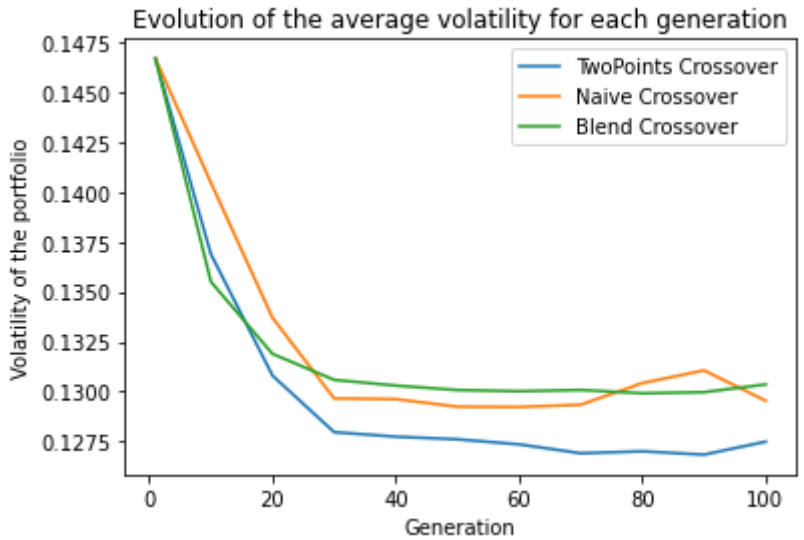
Evolution of the volatility of the best portfolio and the average volatility for each generation

d. Comparison

	Average of the volatility over the 100 generations
Best portfolio Naive	0,128349909
Best portfolio Blend	0,129903182
Best portfolio TwoPoints	0,127049364
Generation Naive	0,132630818
Generation Blend	0,132304818
Generation TwoPoints	0,130294727



Evolution of the volatility of the best portfolio for each generation



Evolution of the averaged volatility for each generation

Once again, the TwoPoints crossover function also seems to be the better crossover function. We can therefore validate our previous results.

Conclusion

By numerical and graphical comparison, we can affirm that the TwoPoints crossover function is the best method for our portfolio optimization problem. It is better overall and in each generation. We have demonstrated this by taking 2 different fitness functions. The other two methods are quite similar on average, but numerically, the Naive crossover function gives a better average Sharpe ratio over the 100 generations and has a lower volatility for the best portfolio.

The TwoPoints Crossover function may yield superior results compared to the blendCrossover method due to a fundamental difference in their approaches. In contrast to blendCrossover, which compels the child portfolios to emulate the best ones, potentially causing rapid convergence towards the current optimal portfolio, TwoPoints Crossover enhances and enforces diversity within the population. This push for increased variety guides the population along new trajectories, opening up paths that might not be explored under the constraints of blendCrossover. The same can be said for the Naive Crossover method.

The provided code constitutes a solid foundation for portfolio optimization using a genetic algorithm. By considering the Sharpe ratio and the volatility, it offers a robust approach to portfolio management. However, this crossover function may not be the most suitable for solving a traffic engineering problem, for example.