Portfolio optimisation problems with hard-to-optimise objective functions

Martin Vesely*

Abstract

The main aim of this paper is to introduce a new algorithm for portfolio optimisation tasks with hard-to-optimise objective functions. The algorithm is based on a heuristic approach called simulated annealing and is suitable for problems with objective functions that are non-differentiable, discontinuous, or have many local extremes (or a combination of all of these features). The performance of the new method is demonstrated on two common problems in quantitative finance: currency portfolio optimisation with a VaR objective function and replication of a benchmark index by its significantly smaller subset of securities. To compare the performance of the new algorithm, several other heuristic methods are introduced. The paper can also be considered as a brief introduction to heuristic optimisation methods for users in economics and finance.

"Simplicity is power."
Traditional proverb and motto of this paper.

* Risk Management Department, Czech National Bank; email: martin.vesely@cnb.cz.

1. Introduction and motivation

It is well known that practical problems differ from academic ones, as the real world is complex and contains links that are not always apparent. This is especially true for economic and social issues such as portfolio optimisation.

Even conservative investors such as central banks and international development organisations sometimes have to reconsider their investment strategy as a result of changing financial markets (eg a prolonged period of negative interest rates). A quantitative approach can help decision-makers to better understand markets and project different outcomes of their investment decisions under different scenarios and circumstances. Portfolio optimisation is an inseparable part of this decision-making support, as strategic asset allocation is crucial for long-term capital preservation and income generation. But achieving a realistic optimisation result can be complex, as many variables have to be incorporated into a model.

Mathematical functions describing financial market behaviour without much simplification are not mathematically elegant, especially when statistically and empirically derived relations are incorporated. These functions can be non-linear, non-differentiable or discontinuous. Moreover, functions in optimisation problems can have many local minima, maxima or both. The number of extremes can be so high that a function graph might suggest an "egg holder" rather than a mathematical function[1]. If a function has some of these features, it can be impossible to employ some optimisation algorithms for extreme searching, because they usually rely on a good deal of assumptions about the function's shape and features. Functions describing real world behaviour are hardly ever linear (although sometimes they can be good approximations). Thus, linear programming is often avoided. One can say that this is not a problem since many algorithms for non-linear optimization exist, such as quadratic programming and gradient methods (eg Newton or conjugate gradient methods). However, to assume that the function is quadratic could sometimes be too restrictive as well[2]. Gradient methods can cope with other kinds of non-linearity but they work properly only with functions having at least a first derivative with respect to all variables. This means that in case the function does not have them, one needs to deploy non-derivative methods, eg the simplex method[3]. While the non-derivate optimisation methods provide means to skirt most of the potential problems so far described, even these methods can have problems with discontinuous functions. Additionally, each method discussed above can be trapped in local extremes of the function. In other words, the method could find some extremes, however, not the global one that had been sought.

The difficulties brought about by the effort to describe market behaviour as precisely as possible can be addressed by the so-called heuristic methods. These are

[1] The so-called "egg holder" function is used for testing an optimisation algorithm's ability to find extremes. See for example: en.wikipedia.org/wiki/Test_functions_for_optimisation.

[2] Moreover, quadratic programming guarantees finding of the global extreme only for a subset of quadratic functions (in particular those with a positive definite matrix). Otherwise, the global extreme does not have to be found.

[3] This method should not be confused with linear programming simplex. The name for both is the same but the principle and group of problems they can solve are different. For more information on the simplex method see Nelder and Mead (1965).

based on generating random solutions to an optimisation problem. As simplistic as it sounds, random algorithms can be highly sophisticated. Indeed, heuristic algorithms are usually based on physical and biological processes described by mathematical rules but rooted in the probability realm instead of a deterministic one. For example, heuristic algorithms are inspired by processes like a cooling of hot materials (simulated annealing[4]), cosmology (so-called multiverse optimisation[5] based on the theory that many parallel universes exist simultaneously), the evolution of life-forms (family of so-called genetic algorithms[6]) or the behaviour of predators (eg algorithm mimicking hunting habits of grey wolf packs[7]).

The most important feature of heuristic algorithms is an absence of any special assumptions about the optimised function properties, such as continuity, convexity etc. While the function itself can be mathematically complex, a heuristic algorithm is, in contrast to deterministic methods, still able to find an optimum. On the other hand, it has some disadvantages. Due to its stochastic nature, there is no guarantee that the heuristic method can find a global or even a local optimum. This disadvantage can be addressed by running optimisation many times from the same or different initial points, and then simply picking the best solution. Since generating a huge amount of random numbers can be time-consuming, heuristic methods are generally slower compared with deterministic ones. As powerful computers are more affordable nowadays, one can cope with this drawback easily. A bigger challenge is probably having to choose the right heuristic method for a particular problem because there is no straightforward guideline for this. The best practice is to employ several heuristic methods, such as those inspired by different natural phenomenon, and select the best solution. This paper will introduce some of these methods and present a new one designed purely for portfolio optimisation.

The rest of the paper is organised as follows. First, some optimisation methods are briefly introduced. The new method is then described in detail, followed by a demonstration of the new method on two portfolio optimisation tasks: (i) a minimisation of empirical VaR (ie problem with non-linear objective function) and (ii) an index replication (ie problem with non-linear constraints). Finally, the conclusion is presented.

2. Examples of heuristic optimisation methods

In this section, some examples of optimisation algorithms are presented, beginning first with the simplex method (the only representative of deterministic approach), followed by the simulated annealing technique (which is introduced as a basis for the new method) and, finally, multiverse and grey wolf optimisers.

[4] See Metropolis et al (1953), Izrailev and Scharf (1990) and Corana et al (1987).

[5] See Mirjalili et al (2016).

[6] See Rasheed et al (1997).

[7] See Mirjalili et al (2014).

Before introducing the algorithms, it is important to understand the term "stop condition", which is, in short, a criterion that the optimisation algorithm has been completed. There are two types of stop conditions:

1. the maximal number of iterations, ie the number of repeating computational steps of the algorithm, is reached; and
2. the desired precision of the solution is reached

These two types of stop conditions are usually combined. The first one ensures that the algorithm will stop within a finite time to prevent the process from being stuck in an endless loop. The second is used for saving computational time when the required precision is reached.

2.1 Simplex method (Nelder-Mead method)

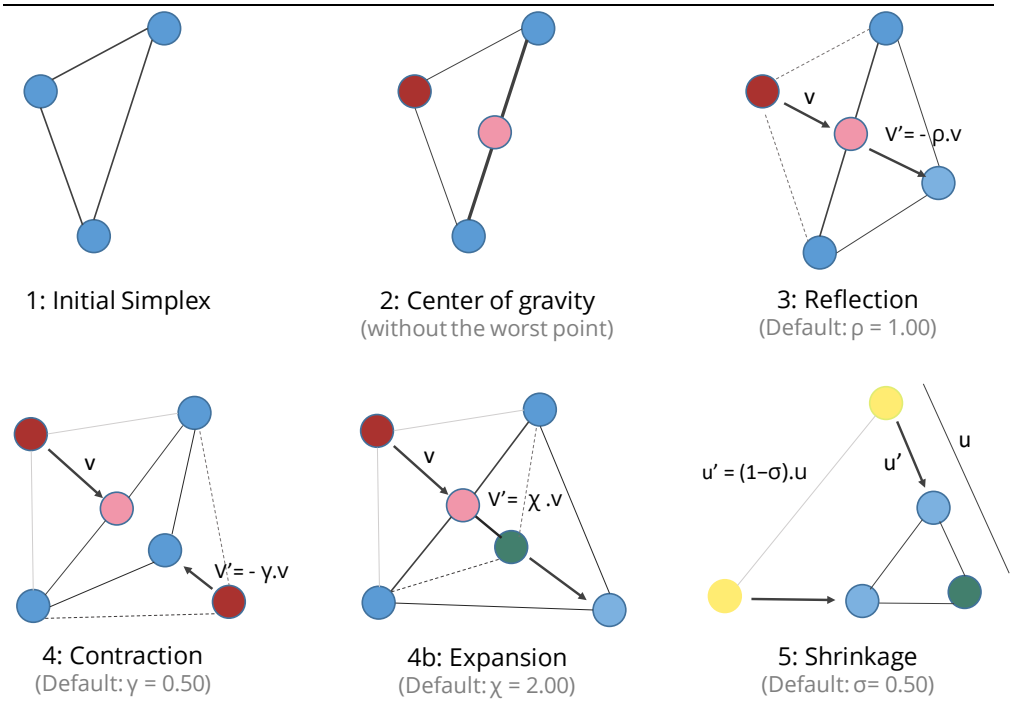
The simplex method is the only deterministic method presented in this paper, as it does not use derivatives and thus does not rely on assumptions about the shape of the problem objective function. This means that one does not have to sacrifice the accuracy of a model for restrictions imposed by a used method.

The simplex method uses a geometrical shape called a simplex. In the n dimensional space, the simplex is a regular body with $n+1$ vertices (eg in 1D space, it is a line, in 2D it is an equilateral triangle and in 3D a regular tetrahedron).

An initial solution is specified by the user in the first step of the algorithm. Then a simplex with the initial solution in its geometrical locus is created. In the next step, the objective function values in the simplex vertices are determined. The next step involves choosing a point with the lowest value[8] of the objective function from the vertices set and the simplex centre. Then, the simplex is modified according to the location of the point with the lowest value in order to surround the point as closely as possible. The simplex can be changed by three operations – reflection, contraction (shrinking) or expansion. These steps are repeated until the stop condition is met. During the process the simplex goes to the minimum of an objective function and reduces itself as illustrated in Figure 1.

[8] This is assumed that minimising of a function is desired. The highest value is chosen in case of maximisation.

Illustration of the simplex algorithm Figure 1



1: Initial Simplex

2: Center of gravity
(without the worst point)

3: Reflection
(Default: ρ = 1.00)

4: Contraction
(Default: γ = 0.50)

4b: Expansion
(Default: χ = 2.00)

5: Shrinkage
(Default: σ= 0.50)

Source: http://capsis.cirad.fr/capsis/documentation/optimisation, licensed under Creative Commons BY-NC-SA 3.0.

A detailed description of the simplex method (including the formulas for operations shown in Figure 1) is provided in Nelder and Mead (1965).

2.2 Simulated annealing (Metropolis algorithm)

Simulated annealing belongs among the oldest heuristics[9], with a long record of successful use in sectors including basic research, defense, chemistry, machinery etc.

Simulated annealing is based on simulation of material cooling. In technical terms, it searches for the configuration of particles set with the lowest possible energy at a given temperature. The algorithm was initially used in areas of physics and chemistry before being extended to other fields including business optimisation tasks. However, for portfolio optimisation, this application requires modifications that will be outlined in the third section.

Let us start with the general simulated annealing algorithm. Note that minimising the objective function is assumed. The algorithm has the following steps:

1. An initial solution is established.
2. The objective function value is calculated for the initial solution.
3. The following steps (4 to 7) repeat until the stop condition is met.
4. A "noise operator" is applied to the solution, ie some member(s) of the solution is (are) changed randomly.

[9] The algorithm was invented by physicist Nicholas Metropolis during his collaboration with Edward Teller, the father of the hydrogen bomb. Therefore, the alternative name "Metropolis algorithm" is often used.

5. The objective function is calculated for the new solution obtained in step 4.
6. In case the objective function value is better than the previous one, the new solution is preserved and the algorithm returns to step 4.
7. In case the objective function is worse than the previous one, the following sub-algorithm is used:
   7.1. A uniformly distributed number between zero and one is generated (denoted by x_i)[10].
   7.2. Probability p_i of whether the worse solution can be preserved is determined.
   7.3. In case $p_i > x_i$ the solution is preserved; otherwise, the solution is abandoned.
   7.4. The algorithm returns to step 4.
8. At the stop condition, the best found solution is returned as the final result.

In the first step, the initial solution of the problem is generated. It can be established in many ways, eg it can be a purely random, an expert estimation or a result evaluated by another optimisation algorithm. However, for simplicity, it is sufficient to consider the initial solution as an external input (a method on how to prepare random initial solutions will be shown later). The second step is self explanatory. The third step deals with the stop condition as previously defined. The fourth step (the "noise operator") is aimed towards the random change in the solution of the problem. There is no general approach to noise operator generation. This depends on the nature of the solved problem, eg a random vector with uniformly distributed members is added to the current solution as proposed by Metropolis (1953). The noise operator application merely leads to the solution change; hence it performs the actual process of an optimisation. The fifth and sixth steps are self explanatory.

What can be unusual for readers familiar with deterministic methods is step 7. Its purpose is to avoid getting trapped in a local optimum (so-called "nesting"). The sub-algorithm can be likened to a process "to shake" the solution out of a potential nest. Let us have a look at some subtleties of this sub-algorithm.

The calculation of the probability in step 7.2 can be done in many ways. The one proposed by Metropolis (the so-called Metropolis criterion (Corana et al (1987)) is derived from the laws of thermodynamics. The probability is determined by the formula

$$p_i = \exp\left(-\frac{f_{new} - f_{old}}{kT_i}\right), \quad (1)$$

where f_{new} is the current value of the objective function (ie worse one), f_{old} is the former one, k is the so-called Boltzmann constant and T_i is temperature in current iteration. The Boltzmann constant is a very important number in thermodynamics[11], however, for application in finance, the constant k can be an arbitrary positive real number, which can be used for fine-tuning the algorithm performance. Temperature

[10] Note that the new random number is generated in each iteration.

[11] The Boltzmann constant plays a role in the relationship between entropy and number of quantum states of particle sets. As a result, it can be found in many formulas in thermodynamics, eg for pressure of gas, energy distribution of particles (so-called Boltzmann distribution) etc. Its value is approximately $1.381 \times 10^{-23} JK^{-1}$.

T does not have any interpretation in the finance world either, but for historical reasons this parameter will be called so in this paper.

The temperature should decrease during the optimisation process. This dynamic ensures that an increasingly smaller number of solutions with worse objective function values are preserved in the process because decreasing temperature leads to decreasing probability *p* (as *T* in formula (1) is approaching zero, *p* is approaching zero as well). This approach links the method to the material cooling simulation. With decreasing temperature, particles of the material are more stable and do not jump to higher energy levels as often (ie the shaking the solution is not as intensive as at the beginning and it gradually diminishes).

The temperature decreasing process (the so-called "cooling scheme") is described by the following formula (the so-called exponential rule[12]):

$$T_i = T_0 q^{-i}, \tag{2}$$

where q is a fixed number in an interval between zero and one (excluding interval boundaries), T_0 is initial temperature (any positive real number) and i is a natural number increasing during the run of the algorithm. In practical implementation, i is the number of current iteration.

The approach shown above is the original one used for optimisation based on thermodynamics laws. In case of portfolio optimisation, one can adapt formulas (1) and (2) more freely. The best practice is to try to do optimisation with different values of parameters k in (1) and q in (2) or even replace the exponential function in (2) with a linear one (this will be shown in Section 3) and in the end to choose the best algorithm setting.

Because of step 7 and a possible worsening of objective function value, the best solution found so far and its respective objective function value have to be saved in some auxiliary variables and both have to be updated if a better solution is found. This is done in step 5. In the last step (8), the saved value of the objective function is compared with the actual one and a better solution is returned as the final result. Apparently, three variables for solutions and the respective objective function values are used – actual solution, previous solution (these two are updated in each algorithm iteration) and the best solution (this is updated only when better solution is found).

2.3 Multiverse optimisation

Section 2.3 and 2.4 describe two examples of optimisation algorithms developed in this decade. The first method[13] is inspired by recent discoveries in the field of cosmology, namely multiverse theories[14]. These assume that many universes exist at one point in time. The multiverse optimisation considers a set of a problem's possible solutions to be a set of universes. Hence the method operates with more than one solution simultaneously in contrast to simulated annealing.

According to the theory, the universes can be interconnected with white holes (ie an entrance-only gate to a universe) and black holes (ie an exit-only gate from a

[12] See Corana et al (1987).

[13] See Mirjalili et al (2016).

[14] Plural is used since there are more theories (eg quantum and superstring theories) predicting the presence of more than one universe.

universe). Particular parts of one universe are connected by wormholes (ie shortcuts among different parts of a universe). Solutions are modified by exchanging their specified parts (ie some vector members) through white and black holes. This means that part of a particular solution is replaced by a generally different part of another solution. In other words, one solution accepts the part via a white hole and the other solution donates it via a black hole. Which solution will be the acceptor and which one will be the donor depends on the circumstances in each universe (details are described in the original paper). Wormholes are used for interchanging parts of one particular solution. It is noteworthy (especially for readers already familiar with optimisation heuristics) that the method is similar to the genetic optimisation[15] – the former way of changing a solution (ie white-black holes connection) is akin to a crossing of genes, while the latter one (ie wormhole connection) is similar to a mutation. These two ways of changing a solution can occur with some probability. During the run of the algorithm (ie multiverse aging), the probability that a change occurs through a wormhole is increasing; conversely the probability of a change through black-white holes connection is decreasing. It is an analogy to temperature decrease in simulated annealing. The objective function value for each universe is revaluated in each iteration and the best one is preserved until a better one is found. The algorithm stops when the stop condition is fulfilled.

2.4 Grey wolf optimisation

The grey wolf optimiser is based on the hunting behaviour of a grey wolf pack[16]. It assumes that the pack (ie set of possible solutions) comprises four types of wolves – alpha, beta, delta and omega individuals. While the number of omega wolves is not limited, the other types of wolves have only one representative each. The alpha individual is the head of the pack and thus is considered to be the strongest and smartest wolf. In the optimisation realm, the alpha is identified with the best solution of a problem found so far. Analogically, beta is the second best solution and delta is the third best one. Other solutions are considered to be omegas. During a hunt, the alpha, beta and delta are guides for omega wolves. These wolves adjust their position according to their guides in order to encircle prey. The very same approach is used in optimisation. Omega solutions are slightly changed randomly, but with respect to the three best solutions in all iterations. In other words, a change in each element of a particular omega solution follows both random numbers and the respective elements of alpha, beta and delta solutions. Mathematically speaking, each new omega is a random linear combination of alpha, beta and delta solutions. When omega solutions are updated, the objective function is calculated for them and new alpha, beta and delta and omegas are chosen. The alpha is compared with the best solution obtained so far in previous iterations. In case the new alpha is better, it is preserved as the best solution onward. Steps of algorithm are repeated until the stop condition is fulfilled.

2.5 A few words on algorithm selection

Several heuristics were introduced, however it is not always obvious which method is more suitable for a specific task. One method can easily solve some tasks while

[15] See for example Rasheed et al (1997) for more information on genetic optimisation.

[16] See Mirjalili et al (2014).

completely fail in others – there is no panacea for optimisation tasks. A common practice is to horse-race multiple methods, fine tune their parameters and choose the winners for a hard-to-optimise problem.

3. The new method

In this section, a modified version of the simulated annealing (hereafter "the new method") for portfolio optimisation is introduced[17].

Methods for initial solution creation and imposing constraints are discussed as well. These approaches are not limited only to portfolio optimisation and may be used for any optimisation.

3.1 Modification of simulated annealing for portfolio optimisation

To solve a particular problem (the portfolio optimisation in this case), one could adapt three parts of the simulated annealing algorithm:

1. Noise operator
2. Temperature decreasing process (cooling scheme)
3. Nesting in local extremes avoidance (shaking the solution)

The modifications of these algorithm's parts proposed below are designed to enable the incorporation of the following constraints for asset weights into the algorithm directly (ie a user can be sure that these conditions are fulfilled automatically):

1. $w_i \in < 0; 1 >$
2. $\sum_{assets} w_i = 1$

These constraints are commonly used in portfolio optimisation without short positions[18].

To present the design of the algorithm, let us start with the description of the noise operator. Assume that an initial solution fulfilling the above-listed constraints was prepared (a method facilitating this will be described in Section 3.2). Then a random number uniformly distributed between –0.5 and 0.5 is generated[19] – denote it as ε. Additionally, assume that w is vector containing n asset weights.

[17] This is the author's actual contribution.

[18] They are used in tasks dealing with probabilities as well. This fact expands the set of tasks where the proposed method can be used.

[19] There is no particular reason for choosing an interval between –0.5 and 0.5. Any other interval can be used. The behaviour of the algorithm can be fine-tuned by changing interval boundaries. The proposed value worked well for tasks presented in Sections 4 and 5.

Two weights of assets are chosen randomly (denote them w_i and w_j) and their values are changed according to the following formulas:

$$w_i \coloneqq w_i + \varepsilon \quad (3)$$

$$w_j \coloneqq w_j - \varepsilon \quad (4)$$

This modification of weight ensures that constraint no 2 is preserved because one weight is increased by ε, while another one is decreased by the same number. However, the first constraint is not necessarily preserved at the same time (eg if ε = 0.4 and w_j = 0.1, then $w_j \coloneqq$ 0.1 - 0.4 = -0.3 < 0). To avoid this constraint breaching, formula (4) has to be rewritten as

$$w_j \coloneqq \min\left[\max\left(w_j - \varepsilon, 0\right), 1\right]. \quad (5)$$

A similar modification has to be done for formula (3) too. Thanks to formula (5), the first constraint is fulfilled; however, the second one fulfilled before can be broken now because of weight cutting in formula (5).

Let us therefore introduce a new variable:

$$\Delta = \varepsilon - \left(w_i^{new} - w_i^{old}\right), \quad (6)$$

where w_i^{new} is the value of i^{th} asset weight after the modification by formula (5) and w_i^{old} is the same weight before the modification. Hence the difference between these weights expresses how much of ε was "consumed" during i^{th} weight modification. Clearly, Δ stands for the remainder (ie the "unconsumed part") of ε. In the next step, a new weight is selected randomly and modified according to formula (5) but ε is set to be equal to Δ (ie $\varepsilon \coloneqq \Delta$). Again, a new Δ is determined by formula (6). This cycle runs until Δ reaches zero, ie original ε is fully consumed. Once Δ is equal to zero, the same approach is used for the second weight w_j with ε set back to the original value (ie random number generated before first change of the w_i). While ε was added to the weight w_i, it is subtracted from weight w_j. Note that for weight w_j, the cycle can run differently because generally $w_i \neq w_j$ and there does not have to be any necessity to use the formula (6). For example, for ε = 0.4 and w_j = 0.5, w_j := 0.5 – 0.4 = 0.1, which is not a forbidden weight value, thus its modification is done in one step. As a result of these modifications in asset weights, both the first and second constraints are fulfilled and the solution is changed as desired.

The second and third modifications of the original algorithm, ie temperature changing and nesting avoidance, are interconnected, therefore they are described together. The temperature decreases in each iteration according to the formula:

$$T_i = 1 - \frac{i}{I_{max}}, \quad (7)$$

where i is the number of current iteration and I_{max} is the maximal number of iterations required. To incorporate the cooling scheme into the algorithm, temperature calculated by formula (7) is used in the noise operator for random number ε modification as follows:

$$\varepsilon \coloneqq \varepsilon\, T_i\,. \quad (8)$$

Clearly, formula (8) ensures that the solution change is getting smaller during the run of the algorithm.

To avoid local minima, temperature decreasing is omitted (ie T_i is set to one) in some iterations. For example, cooling is stopped each tenth iteration[20].

The number of iterations (I_{max}) can be changed according to user requirements and/or the nature of the problem solved.

During practical tests, it was observed that exponential cooling (2) and linear cooling (7) led to the same ability of the method to find the optimal solution. Moreover, it was discovered that using the Metropolis criterion (step 7 in the original simulated annealing algorithm) did not have any positive impact on results and can even have a negative one. Besides, removing the criterion leads to a more comprehensible and simple algorithm.

To finalise this section, a pseudo-code of the method is provided below. Numbers at the end of some rows represent links to respective formulas provided in the algorithm description above. Note that step numbering differs from the one used in Section 2.2 as the method is only inspired by simulated annealing.

Pseudocode for the new method:

```
1.  n := number of assets in portfolio;
2.  Sol_New := create initial solution with n assets;
3.  FOR iters = 1 TO maxIter REPEAT
    3.1.      Sol_Old := Sol_New;
    3.2.      T := 1 - iters/maxIter;                                  (7)
    3.3.      epsilon := RAND() - 0.5;
    3.4.      IF MOD(iters,10) <> 0 THEN epsilon := epsilon*T;         (8)
    3.5.      delta := epsilon;
    3.6.      FOR j = 1 TO 2 REPEAT
      3.6.1.  WHILE delta <> 0 DO
        3.6.1.1.  p := RAND_INTEGER(FROM 1 TO n);
        3.6.1.2.  x := Sol_New (p);
        3.6.1.3  Sol_New (p) := MAX(MIN(Sol_New(p) + delta,1),0);    (5)
        3.6.1.3.  delta := delta - (Sol_New(p) - x);                 (6)
      3.6.2.    END WHILE
      3.6.3.  delta:= -1*epsilon;
    3.7.      END FOR
    3.8.      IF objectiveFun(Sol_New) > objectiveFun (Sol_Old) THEN
               Sol_New := Sol_Old;
4.  END FOR
5.  RETURN Sol_New;
```

3.2 Initial solution

The initial solution, maximal number of iteration and objective function formula are inputs provided by the user. While determining the two latter inputs is quite easy, defining the initial solution could be harder. To ensure correct running of the method, the initial solution has to fulfill the asset weight constraints discussed at the beginning of Section 3.1.

One possible way is to set one of the initial vector members equal to one and others to zero. However, this can lead to strange behaviour of the algorithm (such as nesting) or the algorithm can stop far from an optimal solution. Practical tests show that a performance of the method improves with a high degree of randomness

[20] There is no exact explanation for chosen number. It is purely a practical fine-tuning of the method and a matter of empirical experience. However, it was observed that this setting provides good results for the optimisation of testing objective functions (eg the Rosenbrock banana function or egg holder function).

among initial solutions. The reason for this is simple; in order to get as many as possible different solutions to pick up the best one, it seems logical to start searching for it from as many as possible different initial points.

The method of producing random initial solutions fulfilling the above-discussed constraints follows. Firstly, n random numbers on an interval between zero and some positive number m are generated. Their distribution is arbitrary, however, it should be bounded on a finite interval (uniform and beta distributions are good examples). These numbers are then normalised by their sum. A sum of rescaled weights is naturally one. Moreover, since generated random numbers are positive and divided by their sum, which is apparently higher than any of them, rescaled numbers lie in the interval between zero and one. Clearly, both constraints imposed on portfolio weights in Section 3.1 are fulfilled.

3.3 Constrained optimisation and penalisation function

Except for the previously discussed constraints imposed on asset weights, the method has been hitherto considered as an unconstrained optimisation tool. However, in practice other constraints are often required (eg a maximum share of a particular issuer in portfolio market value or a target portfolio duration). To take them into account, the objective function has to be modified by adding a penalty function.

Firstly, let us assume that a problem contains a constraint of "less than" type, ie $l(w) \leq L$, where $l(w)$ is a general non-linear real function and L is a real number. Note that the variable w is a vector. Let us consider that the original objective function $f(w)$ is minimised. The modified objective function has the following form:

$$F(w) = f(w) + \lambda \max(l(w) - L; 0). \tag{9}$$

This new function is minimised as well. In case the constraint is fulfilled, the difference $l(w) - L$ is negative, thus the maximum of the difference and zero remains zero and the objective function $f(w)$ is unaffected. Conversely, when the constraint is not fulfilled, a positive number is added to the minimised objective function, which leads to its value increase (the penalty). Parameter λ is set by the user; it expresses the strength of the penalty or is used for scaling.

Analogically, it is possible to deal with a "greater than" constraint $g(w) \geq G$, where $g(w)$ is a non-linear real function and G is a real number. The term added to the objective function has following form:

$$\max(G - g(w); 0). \tag{10}$$

Equal type constraints, ie $e(w) = E$, where $e(w)$ is a non-linear real function and E is a real number, can be described by the term:

$$\mathrm{abs}\big(E - e(w)\big). \tag{11}$$

Hence the objective function enriched with the penalty function has the following shape (combination of (9), (10) and (11)):

$$\begin{aligned} F(w) = f(w) &+ \sum_{i=1}^{n_1} \lambda_i \max(l_i(w) - L_i; 0) + \sum_{i=1}^{n_2} \ddot{\lambda}_i \max(G_i - g_i(w); 0) \\ &+ \sum_{i=1}^{n_3} \ddot{\ddot{\lambda}}_i \,\mathrm{abs}(e_i(w) - E_i; 0). \end{aligned} \tag{12}$$

4. Practical task no 1: portfolio optimisation with VaR objective function

In this section, the method introduced in Section 3.1 is tested on a practical portfolio optimisation task – optimal portfolio currency composition.

4.1 The task

Fifteen currencies[21] and gold were included in the optimisation and their percentage weights were searched for in order to minimise VaR of the portfolio. Historical data and the Monte Carlo simulation were used for the preparation of 1,000 scenarios with different annual returns of each currency and gold against the Czech Koruna (CZK). It is worth noting that these data are an external input and their preparation is not connected with the method for portfolio optimization, thus details on the simulation and capital market assumptions are not provided.

Based on these scenarios and on weights for all currencies in the portfolio, empirical VaR can be calculated. Let us denote $\boldsymbol{A}$ a matrix containing annual returns in all scenarios (rows) and for all currencies (columns). In this setup, the matrix element a_{ij} represents the return of j^{th} currency against CZK in i^{th} scenario. Let us denote w a column vector of weights, ie j^{th} vector member is weight of j^{th} currency. It is clear that the result of matrix multiplication $\boldsymbol{A}w$ will be column vector of portfolio expected returns for particular weights in all scenarios. When α percentile of vector $\boldsymbol{A}w$ members is calculated, one gets $(1-\alpha)100\ \%$ empirical VaR of the portfolio for particular weights w.

The goal of the task is to find such weights w that would result in the lowest possible loss, ie to minimise objective function

$$f(w) = \mathrm{VaR}_{(1-\alpha)100\,\%}(\boldsymbol{A}w) = -\mathrm{percentile}_{\alpha\ 100\,\%}(\boldsymbol{A}w). \tag{13}$$

4.2 Issues with current optimisation methods

Multiple attempts to run this optimisation using the MS Excel Solver failed to deliver usable results. Apparently this tool is not able to minimise function (13) with the constraints discussed in Section 3.1. Any possible change in Solver settings did not improve its performance. Irrespective of the initial solution, the solver simply "sat in place" and was not able to move anywhere. Results were better in the case of using a MATLAB function implementing a non-derivative simplex method, however, this approach led to very different results for different starting solutions. The failure to solve the task using deterministic methods was the principal motivation to employ a heuristic approach instead; eventually it led to the developing of the new method described in Section 3.1.

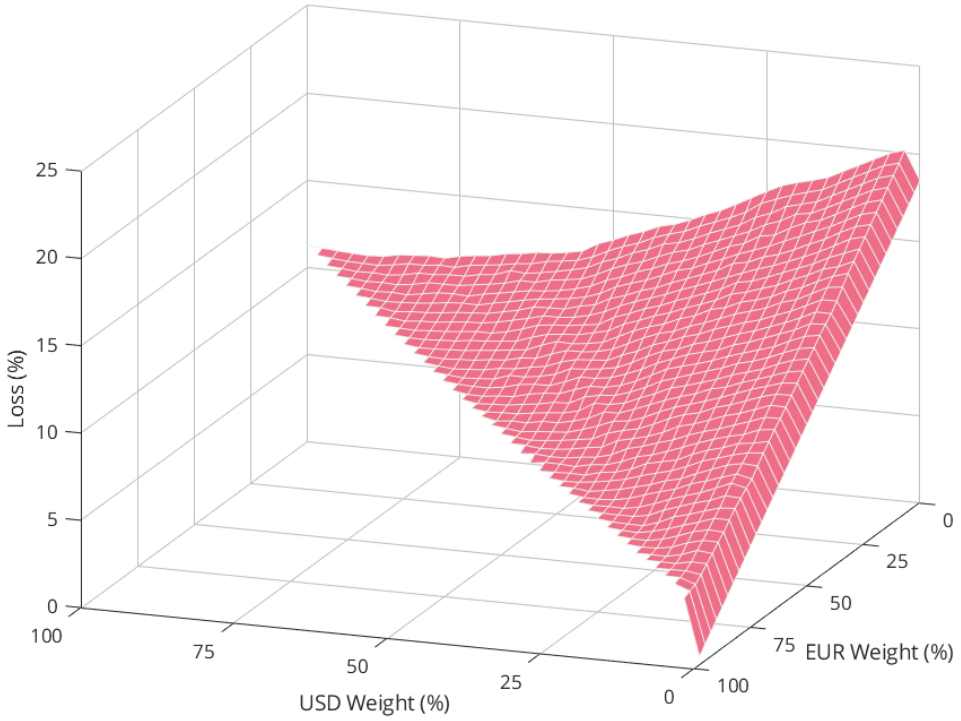
Let us look at the function (13) properties. Because of using empirical VaR and 16 variables, it is difficult to imagine the shape of the function and decide about the presence of local extremes and/or other problems that would prevent algorithms implemented in MS Excel and MATLAB from working. For the case with three variables, a 3D graph can be drawn. It is true that three variables should yield a 4D

[21] In particular USD, EUR, GBP, JPY, AUD, CAD, SEK, CHF, NOK, DKK, PLN, KRW, CNY, NZD and SDR.

graph but only two variables can be changed independently, the third one is determined by the fact that the sum of weights has to equal one. A graph of expected loss in portfolio consisting of the euro, US dollar and gold is shown in Figure 2. As can be seen from the graph, the function is continuous, however, one can see a sharp edge on the right-hand side. Moreover, close examination of the graph reveals that the function is similar to a rippled water surface. That means that the function has many local extremes.

To better illustrate this property, a section of the graph with marked local extremes for a portfolio with zero euro weight is shown in Figure 3. Since the ripples are relatively small (ie extremes are shallow), many optimisation algorithms would have a problem with them[22]. What is more, the function is neither convex nor concave globally. With global convexity being often assumed in deterministic optimisation algorithms, this ambiguity in the function shape is another reason to avoid a deterministic approach and use a heuristic one instead.

Loss function for a portfolio consisting of the euro, US dollar and gold — Figure 2



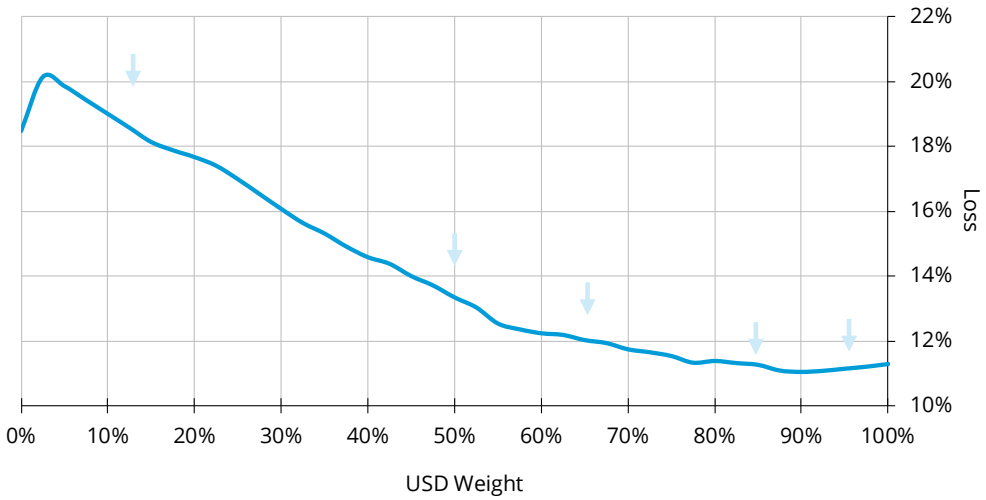
Source: Author's calculations.

[22] This issue is similar to the long and slowly decreasing valley in the Rosenbrock banana function (see for example en.wikipedia.org/wiki/Test_functions_for_optimisation).

Section of the objective function

Figure 3

Percentage



Note: The weight of the euro is set to zero.

Source: Author's calculations.

To conclude this discussion on the function properties; despite the function is seeming relatively uncomplicated at first glance, the issues discussed above disqualify deterministic optimisation algorithms from working even for a relatively simple case of three assets (EUR, USD and gold).

4.3 New method results

The new method introduced in Section 3.1 was compared with several other algorithms. The first one was general simulated annealing[23]. Number two and three were a grey wolf optimisation and a multiverse optimisation[24]. The last algorithm was the simplex method with a random initial solution (randomised simplex[25]).

None of the general methods discussed in this paper are able to fulfill both above described constraints imposed on asset weights automatically. Therefore, the objective function (13) has to be modified by adding a penalty function.

Hence (13) is replaced by

[23] Function anneal created by Joachim Vandekerckhove in 2008 was downloaded from the MATLAB Central Website (www.mathworks.com/matlabcentral/fileexchange/10548-general-simulated-annealing-algorithm?s_tid=prof_contriblnk) for this purpose. The function can be used freely according to license.

[24] Grey wolf optimiser and multiverse optimiser were programmed in MATLAB by author of this paper based on Mirjalili et al (2014) and Mirjalili et al (2016), respectively.

[25] Although the simplex method is a deterministic one, it can be converted to semi-heuristic by randomising the starting point. The simplex method itself was implemented in MATLAB function fminsearch.

$$f(w) = -\text{percentile}_{\alpha\,100\,\%}(\boldsymbol{A}w) + \text{abs}\left(1 - \sum_{i=1}^{n} w_i\right) + \sum_{i=1}^{n} \max\{-w_i; 0\} + \sum_{i=1}^{n} \max\{w_i - 1; 0\}, \quad (14)$$

where the first additional term ensures that the sum of weights will equal one, and the second and third ones ensure that weights will be between zero and one, respectively[26]. It is worth emphasising again that the mentioned constraints are incorporated in the new method by design. Thus (13) can be minimised directly by the new method.

Table 1 compares the performance of the new method with that of others. These statistics are based on 1,000 repetitions of the optimisation launched from different random initial points. The maximal number of iterations was set to 10,000 for all employed methods. It is worth noting that for the purposes of this paper, performance means the ability of an algorithm to approach the objective function global optimum as close as possible; consumption of time and computation power is not of interest.

Comparison of newly developed method with other optimisation algorithms — Table 1

Method	Objective function value				
	Best value	Worst value	Average value	Standard deviation	Variation coefficien
New method	0.1101	0.1253	0.1116	0.0012	1.08%
Multiverse optimisation	0.1111	0.1218	0.1143	0.0018	1.57%
Grey wolf optimisation	0.1111	0.1434	0.1165	0.0048	4.12%
Randomised simplex	0.12	0.2545	0.1683	0.0174	10.34%
Simulated annealing	0.1303	0.1858	0.1565	0.0109	6.96%

Source: Author's calculations.

As can be seen in Table 1, the new method delivered the lowest value of the optimised function. However, since the method is heuristic, this does not mean that the global minimum was reached. The value of 0.1101 might still be the local minimum even though the lowest one found so far.

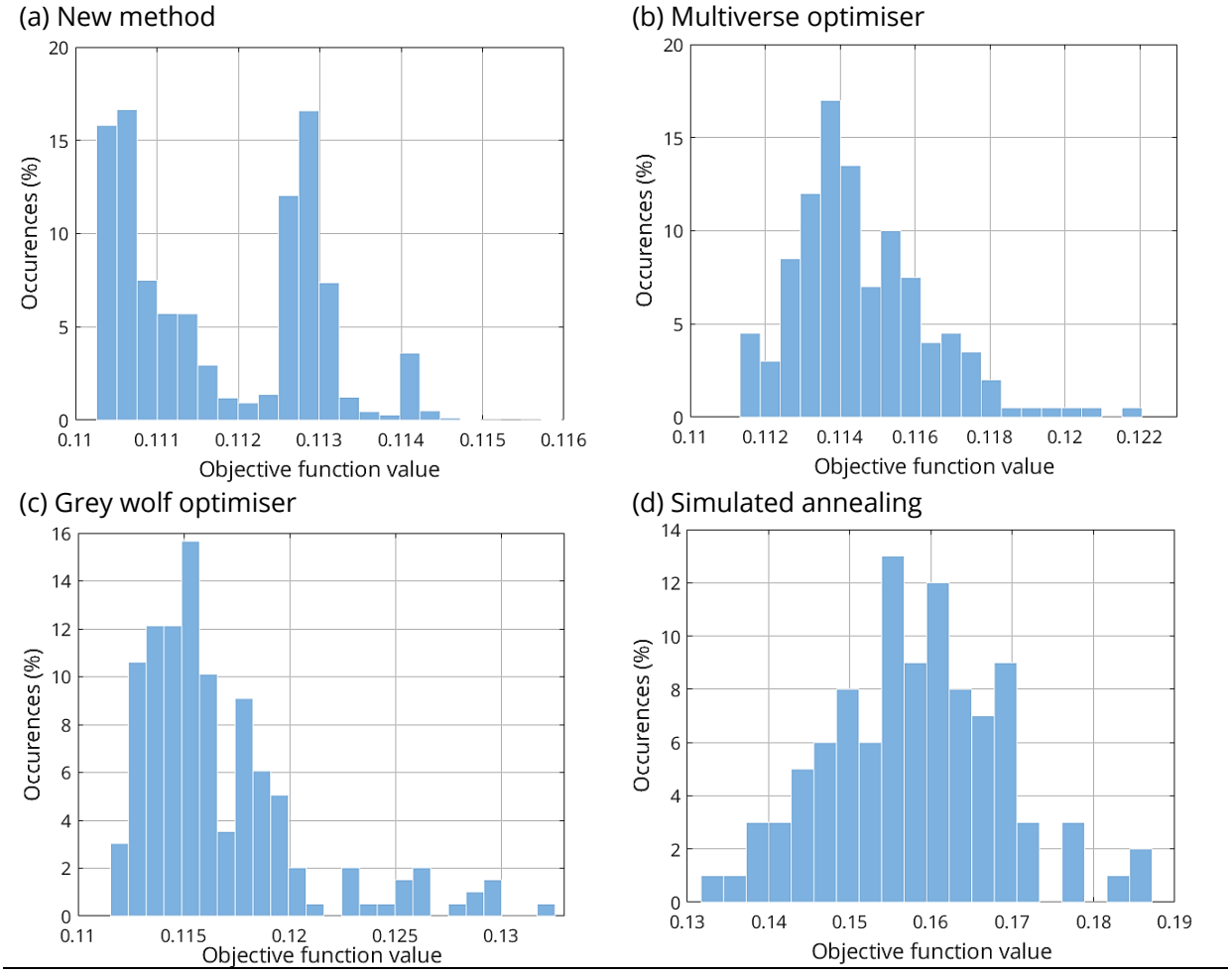
Similarly, the average and variation of optimal value are the best among the compared methods. It is interesting that general simulated annealing, the inspiration for the new method, recorded the worst results measured by the best value of the optimised function. Randomised simplex shows better performance than simulated annealing in terms of the best objective function value, however, it has the highest results fluctuation. Moreover, the average of optimised function values is the worst, too. Both multiverse and grey wolf optimisers reached the same best value of the optimised function but with different variations in results.

[26] The term $\max\{-w_i; 0\}$ can be replaced by $-\min\{w_i; 0\}$ with same effect.

To be more illustrative, fluctuations in results are shown in histograms in Figure 4. As can be seen, the new method repeatedly reached optimal values of around 0.1101 and 0.1130. This behaviour suggests that the optimised function has local minima there. Other algorithms were not able to identify these minima as strongly. General simulated annealing and randomised simplex were even not able to reach such low values. It means that the new algorithm is "strongly attracted" to these local minima. Moreover, histograms also show clearly that the results distribution for the new method is narrower in comparison with others. These observations can be considered as proof of superior performance of the new method in this task.

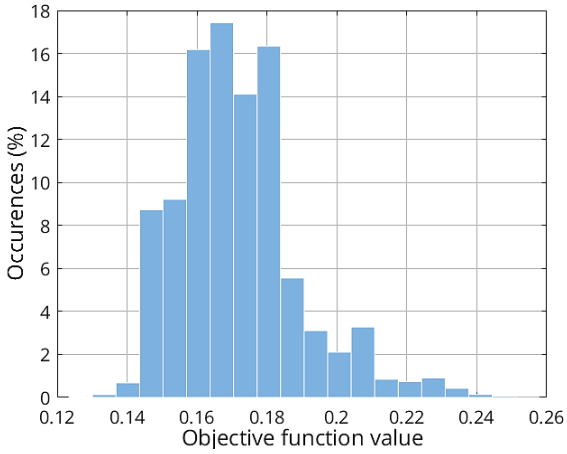
To sum it all up, the proposed method seems to be suitable for portfolio optimisation based on objective functions derived empirically. The method apparently shows better performance in comparison with others based on the statistics presented in Table 1.

Comparison of distributions of optimal values reached by different optimisation algorithms — Figure 4



Comparison of distributions of optimal values reached by different optimisation algorithms — Figure 4

(e) Simplex



Note: Histogram for the new method is trimmed because of outliers. Values higher than 0.117 were recorded in 0.5% of the cases.

Source: Author's calculations.

5. Practical task no 2: benchmark replication with fixed number of bonds

This section demonstrates the second practical test of the new method. A replication of a bond index by its subset with a significantly lower number of bonds in comparison with the total number of index members is carried out.

5.1 The task

Let us consider a bond index consisting of tens or even hundreds of bonds[27]. An investor wants to replicate return and risk of this index by a simpler portfolio containing only units or, at maximum, lower tens of bonds. Let us consider that return is expressed by yield and risk is measured by modified duration and convexity. Apparently, only interest rate risk is taken into account. For the sake of simplicity, credit risk is considered to be same for the whole index (ie only one issuer is represented in the index) and there is no FX risk (ie bonds are denominated in one currency). Clearly, this simplification is not far from reality. For example the investor can be a central bank investing its US dollar reserve portfolio into US Treasuries only. So far the task is solvable by a goal programming since it is linear[28]. However, let us

[27] For example, US government bonds indices or covered bonds indices.

[28] Goal programming is a special case of linear programming. Instead of minimising or maximising the objective function, it aims to make the difference between actual values of some metrics and their goals as small as possible. Technically, it uses a linear programming simplex method for problem solution finding. Note that here the simplex method for linear constrained problems is meant, not the Nelder-Mead simplex method introduced in Section 2.1.

introduce another constraint; maximal or even an exact number of bonds to be included in the investor's portfolio. Because of this special requirement, the task becomes non-linear with discontinuous constraint and it is not possible to solve it with the MS Excel Solver.

To solve the problem, it has to be translated into mathematical language. Return and risk constraints have the following form:

$$\sum_{i=1}^{N} y_i w_i = Y \tag{15}$$

$$\sum_{i=1}^{N} d_i w_i = D \tag{16}$$

$$\sum_{i=1}^{N} c_i w_i = C\,, \tag{17}$$

where *N* is number of bonds in an index; y_i, d_i and c_i are yield, modified duration and convexity of i^{th} bond in the index, respectively; w_i is weight of i^{th} bond in investor's portfolio and finally *Y, D* and *C* are yield, modified duration and convexity of the replicated index, respectively.

Equation (15) is a requirement to make portfolio return equal to an index return. Equations (16) and (17) stand for interest rate risk replication[29]. To fulfil the last requirement, ie to have a specific number of bonds in the investor's portfolio (denote this number *n*), the following constraint has to be added

$$\sum_{i=1}^{N} \mathrm{sgn}(w_i) = n. \tag{18}$$

Function $\mathrm{sgn}(x)$ is a so-called sign function. It returns zero when its argument is equal to zero, one for positive argument and -1 for negative argument. Since short positions are not permitted, all weights are by definition non-negative and function $\mathrm{sgn}(x)$ returns a value of zero for bonds not present in the portfolio and a value of one for bonds included. As a result, the sum in (18) represents the number of bonds in the portfolio. It is worth noting that constraint (18) is the only source of non-linearity and discontinuity in the task.

An objective function for the task is the sum of absolute differences between left and right sides of equations (15) - (18):

$$\begin{aligned} F(w) = \lambda_1 \mathrm{abs}\left(\sum_{i=1}^{N} \mathrm{sgn}(w_i) - n\right) + \lambda_2 \mathrm{abs}\left(\sum_{i=1}^{N} y_i w_i - Y\right) \\ + \lambda_3 \mathrm{abs}\left(\sum_{i=1}^{N} D_i w_i - D\right) + \lambda_4 \mathrm{abs}\left(\sum_{i=1}^{N} C_i w_i - C\right). \end{aligned} \tag{19}$$

The function (19) is minimised.

[29] To better replicate yield curve risk of the benchmark, key rate duration could be used as a constraints.

5.2 Solution and results

The index used for demonstration comprises around 200 US Treasuries with maturity between one and 10 years. The number of bonds was set to five. Yield, modified duration and convexity were set to 2.11, 3.69 and 0.2, respectively.

To obtain results, lambda parameters in (19) had to be set. λ_4 was set to 10, other lambdas were equal to one. This setup represents constraints scaling. As can be seen from the values of parameters above, the convexity is in order of tenths, while other parameters in order of units. That is the reason for the ten times higher value of λ_4. Clearly, lambdas can have different values; however, their ratios should be preserved to reach good results quickly.

Each run of the algorithm can lead to a different portfolio because the task does not have a unique solution and the used heuristic algorithm is probabilistic. Examples of four portfolios compatible with the imposed constraints are presented in Table 2. The table shows selected bonds in each portfolio and compares portfolios parameters with those of the index.

Possible portfolios for the US Treasury index replication Table 2

Portfolio no 1

#	ISIN	Coupon	Maturity	Yield	Duration	Convexity	Weight		Portfolio	Difference
18	US912810EX29	6.75	15-08-26	2.402	6.776	0.565	11.15	Yield	2.1084	–0.08%
49	US912828A834	2.375	31-12-20	2.028	2.859	0.098	16.88	Duration	3.6902	0.01%
72	US912828J272	2	15-02-25	2.405	6.547	0.483	3.08	Convexity	0.2	0.00%
121	US912828RT95	1.375	30-11-18	1.78	0.912	0.013	22.44			
142	US912828TJ95	1.625	15-08-22	2.206	4.404	0.221	46.45			

Portfolio no 2

#	ISIN	Coupon	Maturity	Yield	Duration	Convexity	Weight		Portfolio	Difference
1	US912810EC81	8.875	15-02-19	1.668	1.065	0.017	16.6	Yield	2.1002	–0.46%
8	US912810EK08	8.125	15-08-21	2.083	3.152	0.124	34.1	Duration	3.6901	0.00%
11	US912810EN47	7.625	15-11-22	2.184	4.179	0.212	22.6	Convexity	0.2	–0.01%
13	US912810EQ77	6.25	15-08-23	2.268	4.775	0.276	18.7			
21	US912810FA17	6.375	15-08-27	2.437	7.475	0.688	8			

Portfolio no 3

#	ISIN	Coupon	Maturity	Yield	Duration	Convexity	Weight		Portfolio	Difference
1	US912810EC81	8.875	15-02-19	1.668	1.065	0.017	10.91	Yield	2.11	0.00%
40	US9128282Y56	2.125	30-09-24	2.384	6.218	0.436	25.93	Duration	3.6897	–0.01%
122	US912828RY80	1.375	31-12-18	1.813	0.989	0.015	6.52	Convexity	0.2	0.01%
134	US912828SX98	1.125	31-05-19	1.858	1.402	0.027	17.53			
206	US912828XR65	1.75	31-05-22	2.214	4.221	0.203	39.11			

Portfolio no 4

#	ISIN	Coupon	Maturity	Yield	Duration	Convexity	Weight		Portfolio	Difference
1	US912810EC81	8.875	15-02-19	1.668	1.065	0.017	15.5	Yield	2.11	0.00%
5	US912810EG95	8.75	15-08-20	1.95	2.342	0.071	17.16	Duration	3.6901	0.00%
12	US912810EP94	7.125	15-02-23	2.241	4.336	0.23	39.16	Convexity	0.1999	–0.03%
58	US912828D564	2.375	15-08-24	2.379	6.051	0.416	22.65			
154	US912828U998	1.25	31-12-18	1.816	0.99	0.015	5.53			

Source: CNB internal benchmark for November 2017; author's calculations.

As can be seen from Table 2, presented portfolios are different. Despite their differences, both modified duration and convexity are replicated well. The yield is lower for portfolio 1 and 2, otherwise the match is perfect.

The number of algorithm iterations was 30,000. A higher number of iterations did not contribute to the better constraints fulfilling while a lower number (around 10,000) sometimes led to breaching convexity constraint. Algorithm coded in MATLAB found the solution on average within 12 seconds on a standard office PC[30].

[30] Running time fluctuated between 11.59 and 13.63 seconds.

5.3 Some practical aspects

Due to the heuristic nature of the method and rounding errors, on occasion, some bonds in the portfolio had extremely low yet non-zero weights (in the order of hundred-thousandth). Although such values are, of course, economically irrelevant, they play an important technical role in equation (18). Despite their negligible values, these weights are positive numbers, hence the sign function returns a one for all of them. As a result, issues with fulfilling constraint (18) can occur. For example, the final solution fulfils constraints imposed on yield, duration and convexity, however, the number of bonds is significantly higher than the desired one, eg 15 instead of five often came up during tests. To avoid this behaviour, the algorithm proposed in Section 3.1 has to be altered. Firstly, a threshold value for a bond weight has to be stated[31]. In each iteration, weights lower than the threshold are set to zero.

In order to fulfil constraint $\sum_{i=1}^{n} w_i = 1$, a value defined as

$$E = \sum_{"tiny\ weight"} \overline{w_i}, \tag{20}$$

where $\overline{w_i}$ are former values of weights set to zero, has to be added to any weight higher than the threshold. This weight is chosen randomly. However, constraint $0 \leq w_i \leq 1$ has to be preserved as well, so the value *E* is spread among many different weights higher than the threshold. It is the same approach as in the cycle in pseudocode step no 3.6.1 of the original algorithm. It is clear, that this modification can be considered as an introduction to the second noise operator.

The method was not compared with other optimising algorithms for this task. It serves only as a demonstration that the method can be used for a quick solution of common tasks an investor can face in practice. Moreover, it shows how to cope with rounding errors and economically insignificant values.

6. Conclusion

The main aim of this paper was to introduce a new heuristic method suitable for portfolio optimisation tasks with hard-to-optimise objective functions. This functions category comprises empirically derived objective functions, eg empirical VaR. Since these functions can lack mathematical elegance, employing heuristic methods could be a good option.

The proposed method is based on a simulated annealing heuristic. The general algorithm was adapted to portfolio without short positions optimisation. Its cooling schema is simpler than in the case of general simulated annealing algorithm, ie linear temperature decreasing is used instead of exponential, and there is no direct connection with thermodynamics. For these reasons, the new method is easily understandable, implementable and usable.

The new method was demonstrated on two practical tasks. The first one dealt with the optimisation of a portfolio currency composition. The objective function was

[31] A threshold equal to 1% was ascertained as the best possible value for this particular task by trial-and-error procedure. Moreover, a 1% share of bond in portfolio can be considered economically meaningful. However, the threshold depends on the opinion of a decision-maker.

empirical VaR. The method was compared to randomised simplex, general simulated annealing, grey wolf optimisation and multiverse optimisation. It was demonstrated that the new method was able to find the lowest value of VaR. The dispersion of possible optimal results was minimal for the new method as well. Hence the method is more suitable for portfolio optimisation tasks in comparison with other employed heuristics.

The second task focused on the replication of a benchmark bond index with its significantly smaller subset. The biggest advantage of the new method in comparison with the MS Excel Solver is its ability to operate with discontinuous constraint imposed on the number of bonds in the subset. It was also shown how important it is to adapt a general algorithm to specific tasks. This case illustrates that despite the power of heuristic optimisation methods, they cannot be used as black boxes.

The secondary aim of this paper is to serve also as a guide for risk and portfolio managers looking for powerful tools for solving their portfolio optimisation problems.

Acknowledgments

I am very grateful to Daniel Krejci, my former manager, who brought to my attention the practical pitfalls of optimisation tasks with the objective function based on empirical VaR measure and who also read and reviewed an earlier draft of my manuscript.

I would like to express my appreciation to Shengting Pan for invaluable advice on how to improve the paper and make it more accessible. He also suggested simplification of the initial solution preparation method in Section 3.2.

My special thanks are extended to George Alexander Komrower for language and stylistic corrections.

I am also highly indebted to the Czech National Bank for providing me with the technical base necessary for programming and testing the new optimisation method.

All errors or mistakes are of course mine.

References

Corana A, M Marchesi, C Martini and S Ridella (1987): "Minimizing multimodal function of continuous variables with the simulated annealing algorithm", *ACM Transactions in Mathematical Software*, vol 13, no 3: 262–80.

Izrailev F and R Scharf (1990): "Dyson's coulomb gas on circle and intermediate eigenvalue statistics", *Journal of Physics A: Mathematical and General*, vol 23, no 6, pp 963–77.

Metropolis N, A Rosenbluth, M Rosenbluth and A Teller (1953): "Equation of state calculations by fast computing machines", *The Journal of Chemical Physics,* vol 21, no 6, pp 1087–92.

Mirjalili S, S M Mirjalili, A Lewis (2014): "Grey wolf optimizer", *Advances in Engineering Software*, vol 69, pp 46–61.

Mirjalili S, S M Mirjalili and A Hatamlou (2016): "Multi-verse optimizer: a nature-inspired algorithm for global optimization", *Neural Computing and Applications*, vol 27, no 2, pp 495–513.

Nelder J and R Mead (1965): "A simplex method for function minimization", *The Computer Journal,* vol 7, no 4, pp 308–13.

Rasheed K, H Hirsch and A Gelsey (1997): "A genetic algorithm for continuous design space search", *Artificial Intelligence in Engineering*, vol 11, no 3, pp 295–305.

Disclaimer