FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION OF HIGHER EDUCATION
ITMO UNIVERSITY

Report

on the practical task No. 4

“Algorithms for unconstrained nonlinear optimization. Stochastic and meta-heuristic algorithms”

Performed by

*Dmitriy Prusskiy*

*J4132c*

Accepted by

Dr Petr Chunaev

St. Petersburg

2021

## Goal

The use of stochastic and metaheuristic algorithms (Simulated Annealing, Differential Evolution, Particle Swarm Optimization) in the tasks of unconstrained nonlinear optimization and the experimental comparison of them with Nelder-Mead and Levenberg-Marquardt algorithms.

## Formulation of the problem

**I**. Generate the noisy data $(x_k, y_k)$, where $k = 0, \ldots, 1000$, according to the rule:

$$y_k = \begin{cases} -100 + \delta_k, & f(x_k) < -100, \\ f(x_k) + \delta_k, & -100 \le f(x_k) \le 100, \\ 100 + \delta_k, & f(x_k) > 100, \end{cases} \quad x_k = \frac{3k}{1000}, \; f(x) = \frac{1}{x^2 - 3x + 2},$$

where $\delta_k \sim N(0, 1)$ are values of a random variable with standard normal distribution.

Approximate the data by the rational function

$$F(x, a, b, c, d) = \frac{ax+b}{x^2+cx+d}$$

by means of least squares through the numerical minimization of the following function:

$$D(a, b, c, d) = \sum_{k=0}^{1000} (F(x_k, a, b, c, d) - y_k)^2.$$

To solve the minimization problem, use Nelder-Mead algorithm, Levenberg-Marquardt algorithm and **at least two** of the methods among Simulated Annealing, Differential Evolution and Particle Swarm Optimization. If necessary, set the initial approximations and other parameters of the methods. Use ε=0.001 as the precision; at most 1000 iterations are allowed. Visualize the data and the approximants obtained **in a single plot**. Analyze and compare the results obtained (in terms of number of iterations, precision, number of function evaluations, etc.).

**II**. Choose at least 15 cities in the world having land transport connections between them. Calculate the distance matrix for them and then apply the Simulated Annealing method to solve the corresponding Travelling Salesman Problem. Visualize the results at the first and the last iteration. If necessary, use the city dataset from
https://people.sc.fsu.edu/~jburkardt/datasets/cities/cities.html

## Brief theoretical part

**Simulated annealing (SA)** is a probabilistic technique for approximating the global optimum of a given function. Specifically, it is a metaheuristic to approximate global optimization in a large search space for an optimization problem. It is often used when the search space is discrete. For problems where finding an approximate global optimum is more important than finding a precise local optimum in a fixed amount of time, simulated annealing may be preferable to exact algorithms such as gradient descent or branch and bound.

The name of the algorithm comes from annealing in metallurgy, a technique involving heating and controlled cooling of a material. The state of some physical systems, and the function E(s) to be minimized, is analogous to the internal energy of the system in that state. The goal is to bring the system, from an arbitrary initial state, to a state with the minimum possible energy. The principle of simulated annealing can be expressed in the following algorithm. At each step, the simulated annealing heuristic considers some neighboring state of the current state, and probabilistically decides between moving the system to that state or staying instate. These probabilities ultimately lead the system to move to states of lower energy. Typically this step is repeated until the system reaches a state that is good enough for the application, or until a given computation budget has been exhausted.

**Differential evolution (DE)** is a method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. It is a metaheuristic algorithm, and it makes no assumptions about the problem being optimized and can search very large spaces of candidate solutions.

However, it is not guaranteed that an optimal solution will be ever found. DE optimizes a problem by maintaining a population of candidate solutions and creating new candidate solutions by combining existing ones according to its simple formulae, and then keeping whichever candidate solution has the best score or fitness on the optimization problem at hand. In this way the optimization problem is treated as a black box that merely provides a measure of quality given a candidate solution and the gradient is therefore not needed.

**Particle swarm optimization (PSO)** solves a problem by having a population of candidate solutions (dubbed particles) and moving these particles around in the search-space according to a simple mathematical formula over the particle's position and velocity. Each particle's movement is influenced by its local best known position, but is also guided toward the best known positions in the search-space, which are updated as better positions are found by other particles. This is expected to move the swarm toward the best solutions.

## Results

The results were obtained using Python 3.8.5.

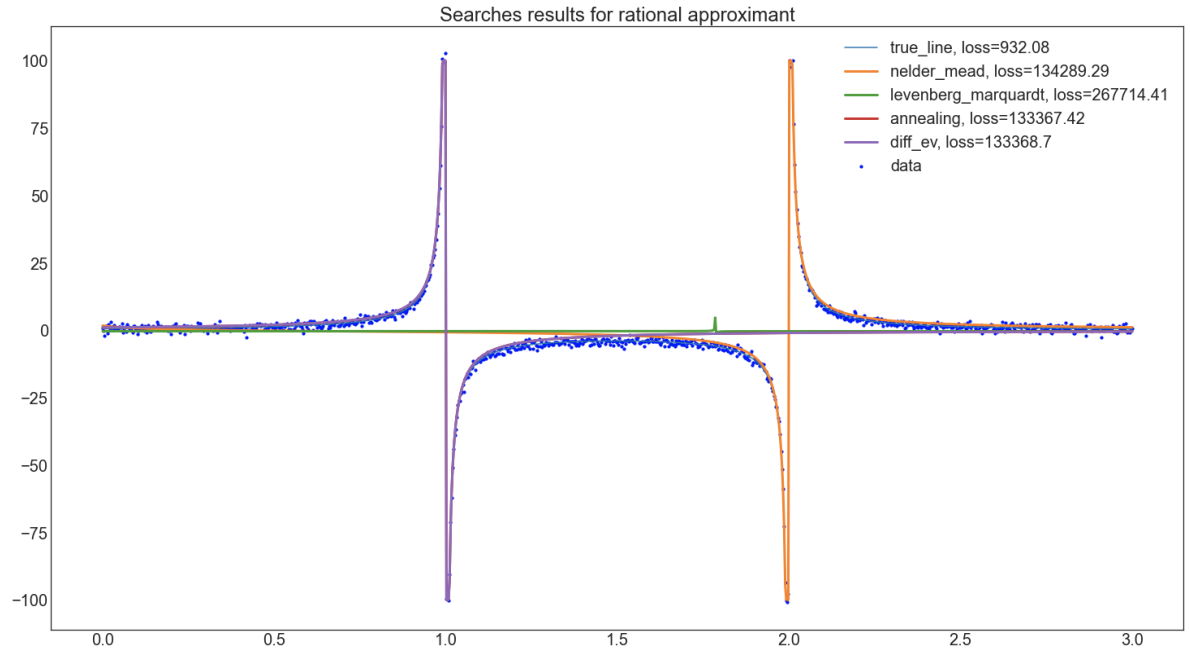
Two meta-heuristic algorithms were chosen to solve the problem – differential evolution and simulated annealing (both taken from scipy library in Python). Scipy implementations of Nelder-Mead and Levenberg-Marquardt algorithms were used for comparison.

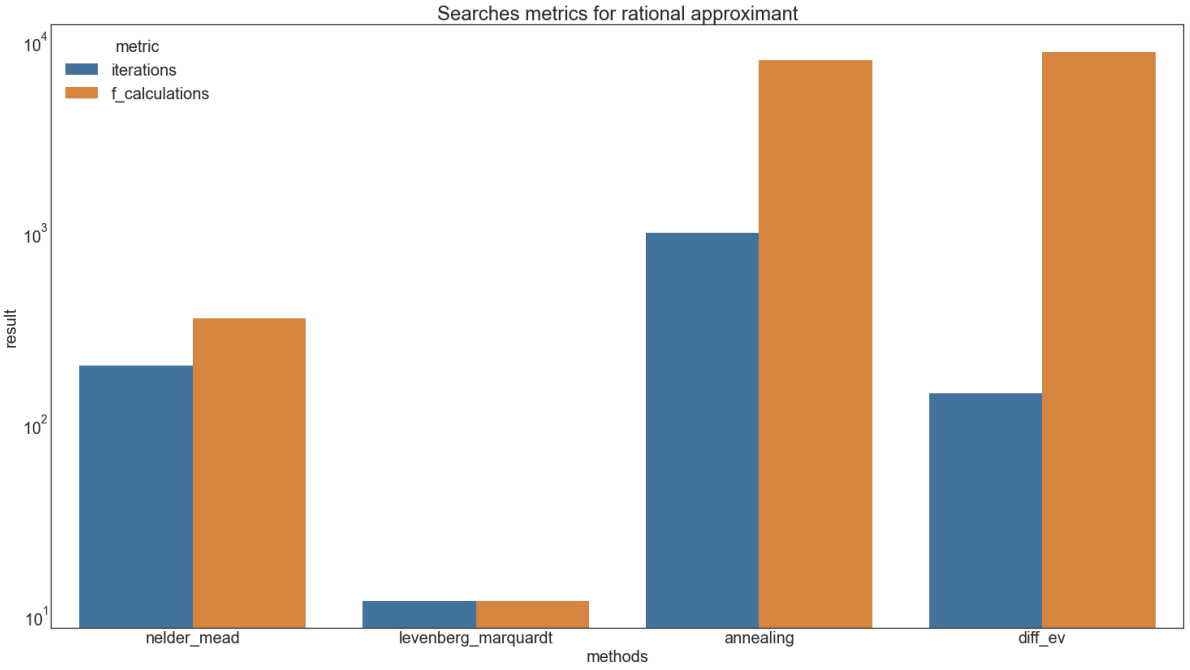
The function to be optimized has 2 breaks within the given scope of definition. This should be a huge disadvantage for both the Nelder-Mead algorithm, which works under the assumption that a function to be optimized is convex, and the Levenberg-Marquardt algorithm, which suggests that the solution lies in the area where the gradient is close to zero. Therefore, the approximate function was clipped as the generation function to have no values more than 100 and less than -100. The gradient of the approximate function for the Levenberg-Marquardt algorithm was computed analytically with respect to the values over 100 and less than -100 (gradient was considered 0 in those points as the function there has constant value).

The optimization algorithms were initialized several times with different initial parameters. Only the differential evolution algorithm managed to find 2 breaks with some initial parameters. Simulated annealing and differential evolution algorithm managed to find 1 break with almost any initial parameters. The Nelder-Mead algorithm managed to find 1 break in 1 of 10 cases. The Levenberg-Marquardt algorithm could not find any breaks in most cases. With some best initial parameters all four algorithms managed to find the same 1 break and get the same loss.

The reason why Levenber-Marquardt shows such a result probably relates to our data - the space where we are looking for optimum is not even continuous, has several optimums, and the neighborhood of the optimum point is relatively narrow. So, if we start in the wrong position - not in the neighborhood of the optimal point in which the loss function is decreasing in the direction of the right optimal point, the gradient as well as Hessian must be pushing the Levenberg-Marquardt optimizer to the optimum point with broader neighborhood i.e. to the point where it fails to capture any gap.

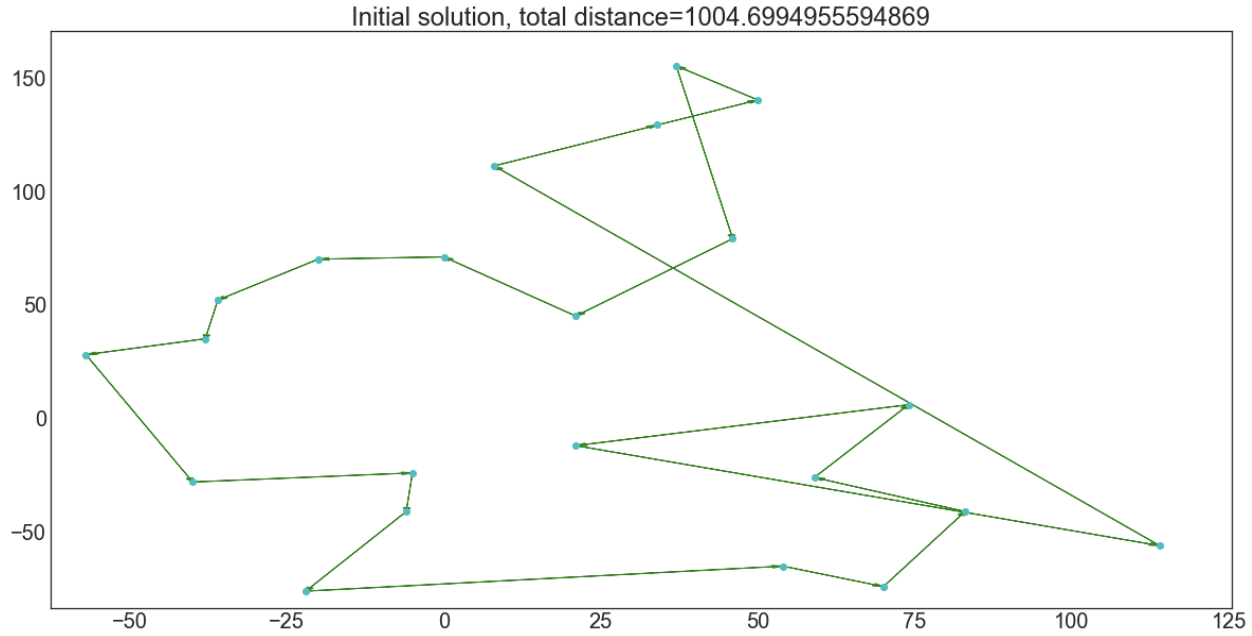
Below are graphs with the results of the study.

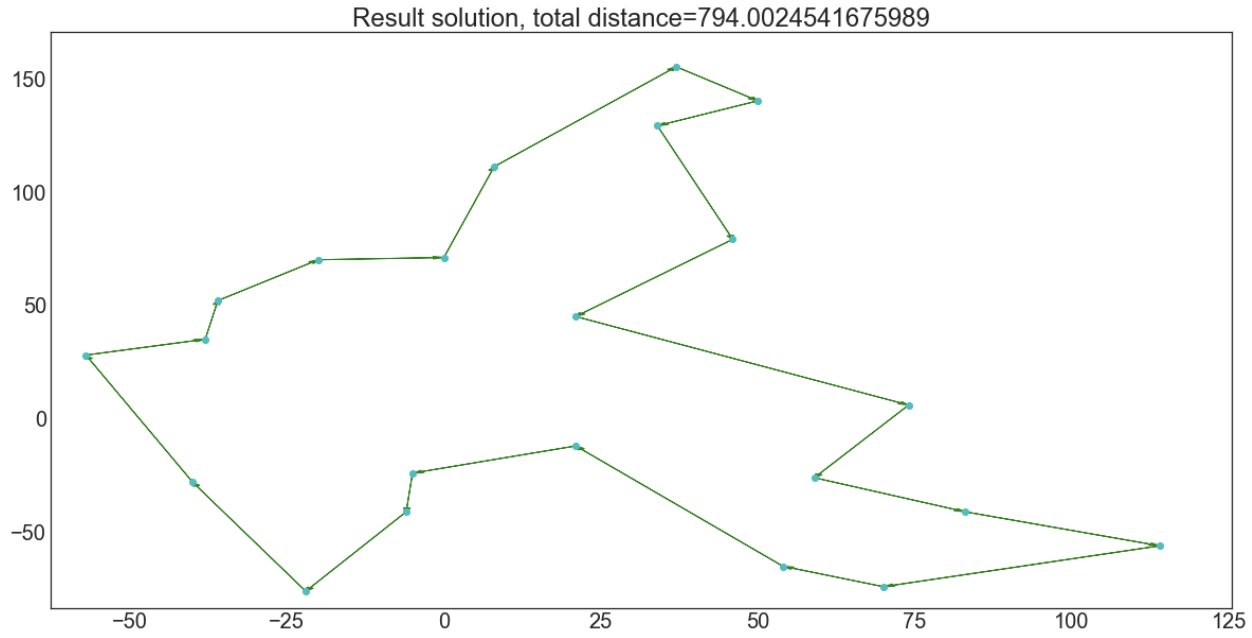



For the second part of the task the WG22 dataset was used. It describes 22 cities in West Germany. The implementation of simulation annealing from https://github.com/chncyhn/simulated-annealing-tsp

Below are results of the initial greedy solution and simulated annealing solution of this task.




## Conclusions

The goal of this study was to stochastic and metaheuristic algorithms in the tasks of unconstrained nonlinear optimization and compare obtained results with Nelder-Mead and Levenberg-Marquardt algorithms. Analysis of the graphs presented in the Results section shows that metaheuristic algorithms solve such tasks better than gradient and numerical methods.

## Appendix

Source code can be found at https://github.com/T1MAX/itmo_algorithms/tree/main/task_4