FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION
OF HIGHER EDUCATION
ITMO UNIVERSITY

Report

on the practical task No. 4

Algorithms for unconstrained nonlinear optimization. Stochastic and metaheuristic algorithms

Performed by

Dmitry Grigorev, Maximilian Golovach

J4133c

Accepted by

Dr Petr Chunaev

St. Petersburg

2022

1. Goal of the work

The goal of this work consists of the following points:

- to become familiar with some stochastic and metaheuristic algorithms in the context of nonlinear optimization problems;
- to apply the methods on practical problems and compare them with each other;
- besides, to apply Nelder-Mead and Levenberg-Marquardt algorithms to these problems and compare obtained results with results of the metaheuritic algorithms.

2. Formulation of the problem

In this task here are two problems under consideration:

2.1. Problem *I*

Let $x_k = \frac{3k}{1000},\ k = 0, \ldots, 1000$ and y_k are defined as follows with i.i.d. $\delta_k \sim N(0, 1)$:

$$y_k = \begin{cases} -100 + \delta_k & \text{if } f(x_k) < -100, \\ 100 + \delta_k & \text{if } f(x_k) > 100, \\ f(x_k) + \delta_k & \text{otherwise,} \end{cases}$$

where $f(x) = \frac{1}{x^2 - 3x + 2}$ — rational function with two vertical asymptotes $x = 1$ and $x = 2$. The problem is to approximate the generated data $(x_k, y_k)_{k=0}^{1000}$ by rational function

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

which optimally fits the data in the sense of least squares

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

The solution has to be found with precision $\varepsilon = 10^{-3}$ in at most 1000 iterations.

This problem has to be tackled with both stochastic and metaheuristic algorithms such as:

- Simulated Annealing,
- Particle Swarm,
- Differential Evolution

and deterministic Nelder-Mead and Levenberg-Marquardt algorithms.

2.2. Problem *II*

Given the data from `https://people.sc.fsu.edu/~jburkardt/datasets/cities/cities.html` with at least 15 cities having land transport connections between them, we have to tackle the corresponding Travelling Salesman Problem by means of Simulated Annealing approach.

All functions which will be obtained have to be visualized with the given data and the line which generates these data. Furthermore, we have to compare the algorithms in terms of precision, number of iterations and number of function evaluations using the results.

3. Brief theoretical part

The theory is taken from [1].

Suppose that we are given with a function $f:\ G \subset \mathbb{R}^n \to \mathbb{R}$ where G is connected open set in $\mathbb{R}^n$. We need to minimize this function on G.

3.1. Simulated Annealing

The name of this algorithm comes from annealing in metallurgy: a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects.

The optimization here starts from evaluating the value of the objective function $f(x_0)$ at an initial random point $x_0 \in G$. At each step the next point x_k is obtained from the current temperature T_k which describes the mobility of moves from x_{k-1} to x_k. There are lots of rules to get to x_k (**neighbor**) from the previous point. One of them is as follows:

$$x_k^{(i)} = x_{k-1}^{(i)} + \left((x_{\max}^{(i)} - x_{k-1}^{(i)})r_{k-1}^{(i)} + (x_{\min}^{(i)} - x_{k-1}^{(i)})s_{k-1}^{(i)}\right)\frac{T_{k-1}}{T_0},\ i = 1, \ldots, n,$$

where $r_{k-1}^{(i)},\ s_{k-1}^{(i)} \sim U(0,1)$ are i.i.d (with respect to both upper and lower indices) uniformly distributed random variables and $x_{\min}^{(i)}$ and $x_{\max}^{(i)}$ define the search range for the solution. There are also lots of ways to define the rule to which the sequence $\{T_k\}$ obeys. Here we consider the following one with the **annealing coefficient** $p \geq 1$:

$$T_k = T_0\left(1 - \frac{k-1}{k_{\max}-1}\right)^p,$$

where $k_{\max}$ is the maximum number of iterations. We accept the new point x_k if $f(x_k) \leq f(x_{k-1})$ (however, there are also a number of ways to accept the new point but we limited ourselves on this one).

Simulated Annealing in Travelling Salesman Problem (TSP)

Let $\{d_{ij}\}_{i,j=1}^{n}$ be the set of distances between n points (for example, cities). Every cyclic path (so-called **tour**) between these points can be expressed in the terms of permutations $\pi = (\pi_1, \pi_2, \ldots, \pi_n) \in \mathbb{G}_n$ where city π_i is connected with π_{i+1} $\forall i = 1, \ldots, (n-1)$ and the city π_n is connected with π_1. The task of TSP is to minimize the function w.r.t. π:

$$f(\pi) = \sum_{i=1}^{n-1} d_{\pi_i \pi_{i+1}} + d_{\pi_n \pi_1}.$$

The algorithm of Simulated Annealing is presented in [2]. To put it briefly, the idea of choice of random neighbor from the current tour is as follows: with probability 0.5 in the current tour π we revert random sub-tour $(\pi_{i_1}, \ldots, \pi_{i_k})$, otherwise this sub-tour is inserted between cities π_j and π_{j+1} where the edge $\pi_j \pi_{j+1}$ is randomly chosen from the rest of edges:

$$(\pi_1, \ldots, \pi_{i_1-1}, \underline{\pi_{i_1}, \ldots, \pi_{i_k}}, \pi_{i_k+1}, \ldots, \pi_j, \pi_{j+1}, \ldots, \pi_n) \to$$
$$\to (\pi_1, \ldots, \pi_{i_1-1}, \pi_{i_k+1}, \ldots, \pi_j, \underline{\pi_{i_1}, \ldots, \pi_{i_k}}, \pi_{j+1}, \ldots, \pi_n).$$

3.2. Differential Evolution

Let we have an initial population $x_1^{(0)}, \ldots, x_m^{(0)} \in G$ of constant volume m. At each iteration for each $j = 1, \ldots, m$ **mutant individual** $v_i^{(k)}$ is obtained from the elements of previous population $x_1^{(k-1)}, \ldots, x_m^{(k-1)}$. There are a large variety of ways to produce $v_i^{(k)}$:

$$v_i^{(k)} = x_i^{(k-1)} + K(x_a^{(k-1)} - x_i^{(k-1)}) + F(x_b^{(k-1)} - x_c^{(k-1)}); \quad (1)$$
$$v_i^{(k)} = x_i^{(k-1)} + F(x_b^{(k-1)} - x_c^{(k-1)}); \quad (2)$$
$$v_i^{(k)} = x_a^{(k-1)} + F(x_b^{(k-1)} - x_c^{(k-1)}); \quad (3)$$
$$v_i^{(k)} = x_{\text{best}}^{(k-1)} + F(x_b^{(k-1)} - x_c^{(k-1)}); \quad (4)$$
$$v_i^{(k)} = x_i^{(k-1)} + K(x_{\text{best}}^{(k-1)} - x_i^{(k-1)}) + F(x_b^{(k-1)} - x_c^{(k-1)}); \quad (5)$$
$$v_i^{(k)} = x_{\text{best}}^{(k-1)} + K(x_a^{(k-1)} - x_b^{(k-1)}) + F(x_c^{(k-1)} - x_d^{(k-1)}); \quad (6)$$
$$v_i^{(k)} = x_a^{(k-1)} + K(x_b^{(k-1)} - x_c^{(k-1)}) + F(x_d^{(k-1)} - x_e^{(k-1)}); \quad (7)$$

where $x_{best}^{(k-1)}$ is the best (in terms of value of function f) individual of $(k-1)$-th generation, a, b, c, d, e are randomly chosen different numbers from the set $\{1, \ldots, i-1, i+1, \ldots, m\}$, $0 \le K \le 1$ is the **combination factor** and $0 \le F \le 1$ is the **scaling factor**. As soon as the mutants are obtained, the **trial individuals** $u_1^{(k)}, \ldots, u_m^{(k)}$ are created in the process of

cross-over with its constant $C \in [0, 1]$:

$$u_{ij}^{(k)} = \begin{cases} v_{ij}^{(k)} & \text{if } r_{ij}^{(k)} \leq C \text{ or } j \neq s_{ij}^{(k)} \\ x_{ij}^{(k-1)} & \text{otherwise,} \end{cases}$$

where $r_{ij}^{(k)} \sim U(0, 1)$, $s_{ij}^{(k)}$ is j-th element in the vector $s_i^{(k)}$ which is a random permutation of the set $\{1, \ldots, n\}$. In other words, the trial individual has some components of the mutant individual and at least one component of its parents $x_i^{(k-1)}$. Then the new generation is produced:

$$x_i^{(k-1)} = \begin{cases} u_i^{(k)} & \text{if } f(x_i^{(k-1)}) \geq f(u_i^{(k)}), \\ x_i^{(k-1)} & \text{otherwise.} \end{cases}$$

Common choice of parameters is as follows: $C = 0.9,\ F = K = 0.8$. Moreover, the larger is m and the smaller are F and K then the more robust is the algorithm and the more expensive is the optimization process.

3.3. Particle Swarm

This method replicates the behavior of birds looking for food and following the leader. The idea is that each individual in the swarm knows both its own best position and best position of the whole swarm.

Let we are given with an initial population $x_1^{(0)}, \ldots, x_m^{(0)}$ of size m. At each iteration the position of the i-th individual is calculated with respect to the formula:

$$x_i^{(k)} = x_i^{(k-1)} + v_i^{(k)},$$

where $v_i^{(k)}$ is the velocity of i-th individual which is the function of previous velocity $v_i^{(k-1)}$, i-th individual's best position $\widetilde{x}_i$ and the population's best position $\widetilde{x}$:

$$v_i^{(k)} = W v_i^{(k-1)} + C_1 r_1 (\widetilde{x}_i - x_i^{(k-1)}) + C_2 r_2 (\widetilde{x} - x_i^{(k-1)}),$$

where $r_1, r_2 \sim U(0, 1)$ are the random variables, C_1 is the **cognitive factor**, C_2 is the **social factor** and W is the **inertia factor**.

4. Results

4.1. Problem I

First of all, we generated data $(x_k, y_k)_{k=1}^{1000}$ as described in the section 2. We need to find the function $F(a, b, c, d, x) = \frac{ax+b}{x^2+cx+d}$ which fits the data in terms of least squares. To tackle this problem, we applied the following algorithms with corresponding parameters:

- Nelder-Mead method,
- Levenberg-Marquardt method (which uses hessian, not jacobian, as described in the **Task 3**) with Golden Section search of parameter ν in the range $[117, 120]$,
- Differential Evolution in the 7-th form as described in the list 1 with 10 individuals, $F = K = 0.8,\ C = 0.9$,
- Particle Swarm Optimization with $W = C_1 = C_2 = 0.5$ and 100 particles.

In the first two methods we used such stop criteria as maximum number of iterations (here 1000) and distance between two consequential points. In the Particle Swarm one we used both the maximum number of iterations and the small change of swarm's best position criteria. As for the differential evolution, the maximum number of iterations criterion is used and the maximum number of iterations throughout which the best configuration of individuals changes a little is used. Furthermore, we search for solutions in the hypercube $[-10, 10]^4$. The initial point for deterministic algorithms is the point $(-1.5, -2.5, 3.5, 4.5)^{\mathbf{T}}$.

As a result of application of these methods, we obtained 4 curves which are illustrated in the graph 1 with the generated data.

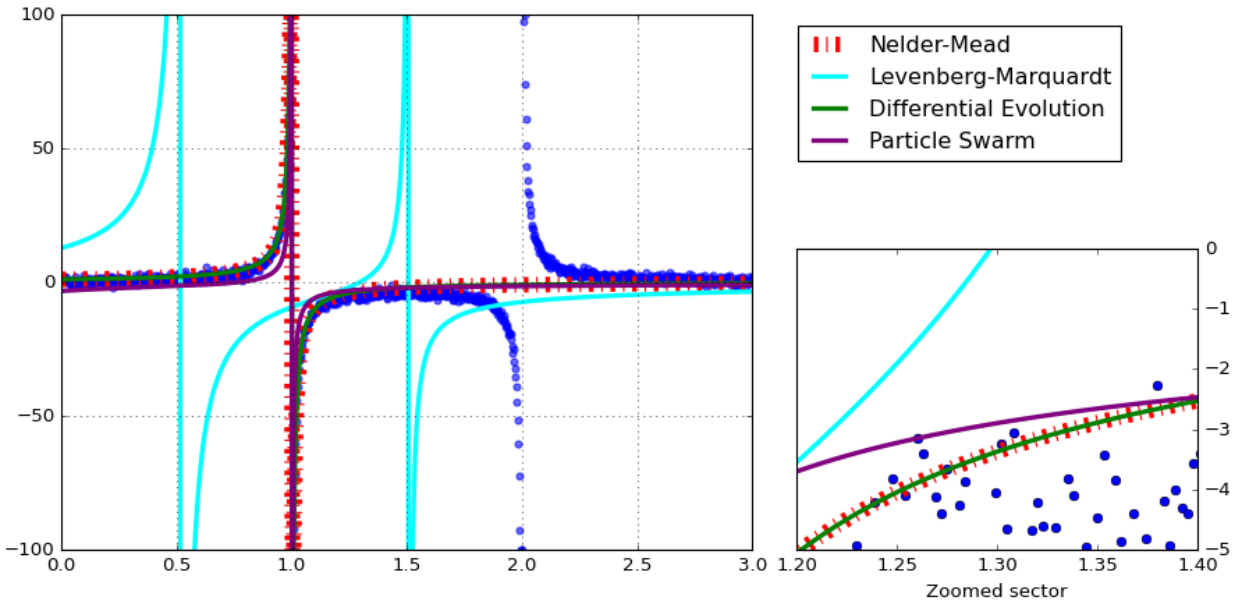

Figure 1: The data and the resulted curves. Additionally zoomed sector of this graph is provided on the right side

As one can see, the resulted from Nelder-Mead and Differential Evolution curves practically coincide with each other. Along with them, the Particle Swarm's curve also well

describes the vertical asymptote in $x = 1$ but ignore another one in $x = 2$. At the same time, Levenberg-Marquardt algorithm provided another solution which describes the data badly at all. Let us analyze the calculation results of these methods which are presented in the table 1.

	Iterations	f-eval.	∇f-eval.	$\nabla^2 f$-eval	matrix inv.	precision
Nelder-Mead	409	695	–	–	–	≈ 1.737
Levenberg-Marquardt	22	440	22	22	396	≈ 11.929
Differential Evolution	591	5920	–	–	–	≈ 1.737
Particle Swarm	57	5800	–	–	–	≈ 5.56237

Table 1: Algorithms' indicators

Here we can see that two metaheuristic algorithms required too many function evaluations since at each iteration they have to calculate its values for each individual/particle. The better one is Nelder-Mead algorithm since it required practically 10 times less function evaluations what resulted in the obtaining of the same solution as Differential Evolution provided. As for Levenberg-Marquardt algorithm, it required 396 4×4-matrices inversions what is expensive for sure. At last, this method converged to another local minimum since the target function is not convex in fact. Due to the ability to avoid being stuck in the neighborhood of local minima metaheuristic algorithms demonstrated themselves here better.

4.2. Problem II

Here we consider the problem of Travelling Salesman according to the data **LAU15**. The task is to find the optimal tour connecting 15 cities by means of Simulated Annealing. As an initial temperature T_0 we put $T_0 = 1$ and the temperature decreases with multiplier $\alpha = 0.9$ every 5 iterations. The picture of initial path whose length is equal to 590 is provided in the figure 2.

We ran this algorithm several times to find two best solutions and non-optimal one. The optimal tour has the length of 291. It is demonstrated in the figure 3. Another tour which differs from the previous one a bit has length 295 (see fig. 4). As an example of the non-optimal route with the length of 319, we provide it in the figure 5.

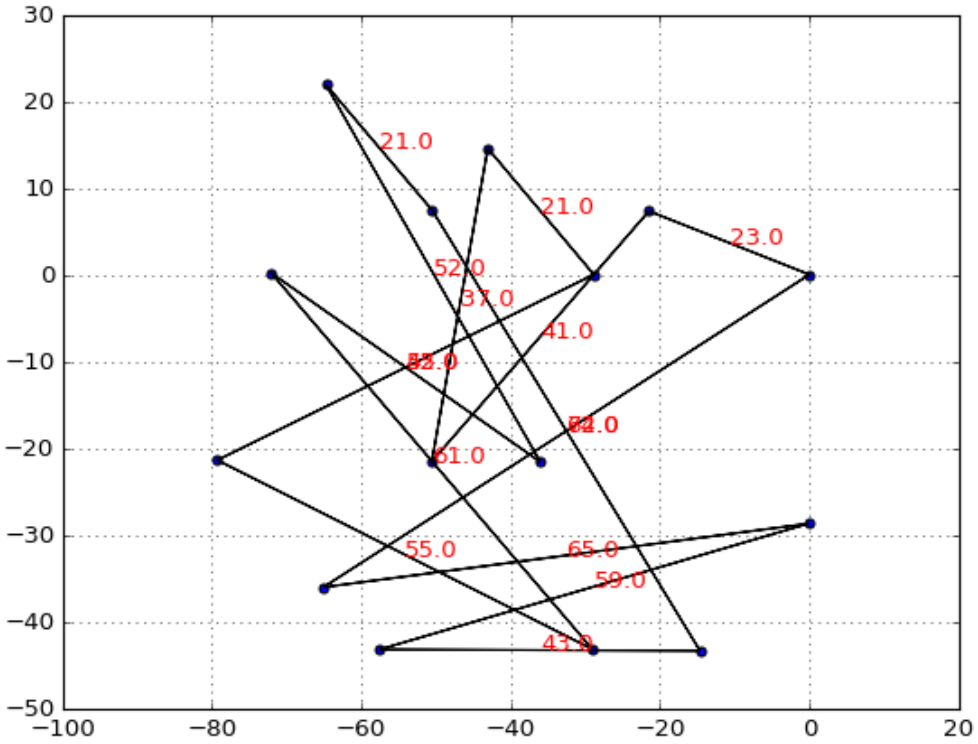

Figure 2: The initial tour used to initiate the optimization process

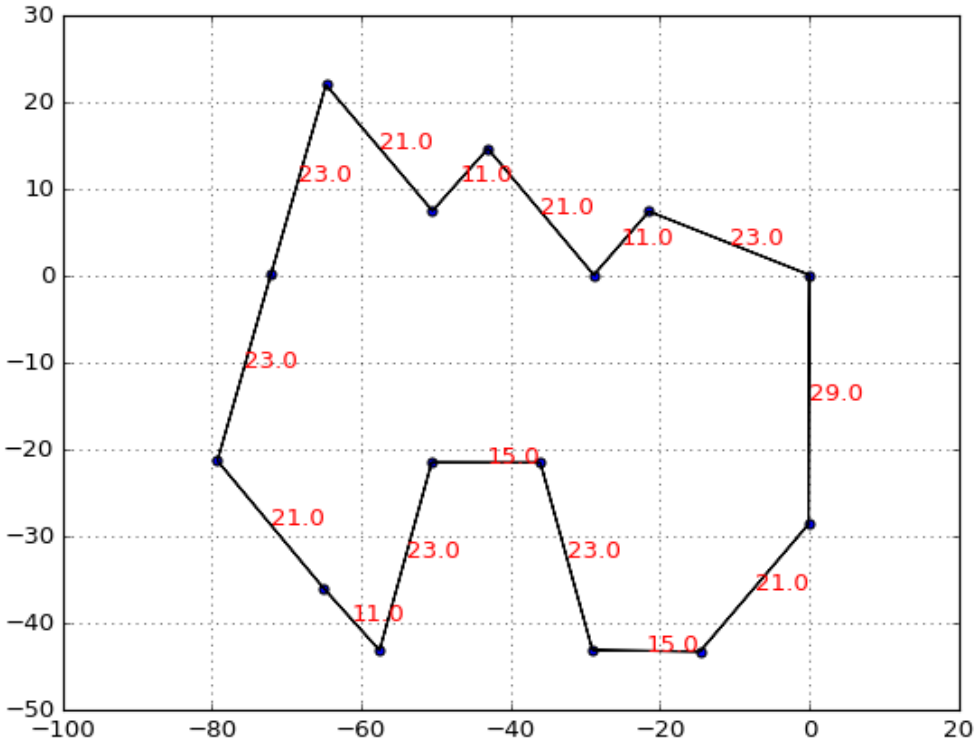

Figure 3: The optimal tour for the dataset

To sum up the results of this task, we have to take into account the fact that simulated annealing is not deterministic algorithm so it may provide non-optimal solution in the first

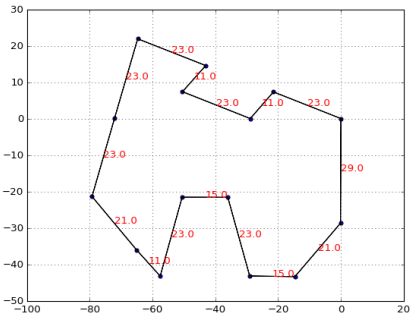

Figure 4: A tour which is not optimal a bit

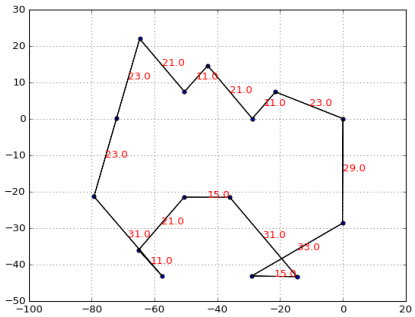

Figure 5: A tour which is not optimal at all

its application. Nevertheless, it does not require any calculations of derivatives.

5. Data structures and design techniques used in algorithms

In the implementation of the algorithms and for random variables generation the Python's package **Numpy** is used since it allows to vectorize calculations what accelerates the evaluations. Particle Swarm algorithm implementation was taken from the package **PySwarm**. The implementation of Nelder-Mead method is taken from the package **Scipy**. For derivation of the gradient and the hessian of the target function in the Problem I we used the well-known package **autograd**.

6. Conclusion

As the result of this work, we considered stochastic and metaheuristic methods of optimization in application to both continuous and discrete problems of optimization. The methods were compared by the number of iterations required to obtain the solution with given precision, the number of function evaluations, precision and other indicators. The implementations of Differential Evolution and Simulated Annealing (for TSP) methods are provided. Data structures and design techniques which were used in the implementations were discussed. The work goals were achieved.

7. Appendix

Algorithms implementation code is provided in [3].

Bibliography

1. Cavazzuti M. Optimization Methods. — Springer Berlin, Heidelberg, 2013. — ISBN: 9783642311864.
2. Li Y., Zhou A., Zhang G. Simulated annealing with probabilistic neighborhood for traveling salesman problems. — 2011. — 07. — Vol. 3. — P. 1565–1569.
3. Grigorev D., Golovach M. Code repository. — `https://github.com/dmitry-grigorev/AlgoAnalysisDevelopment`. — 2022.
4. Deisenroth M. P., Faisal A. A., Ong C. S. Mathematics for Machine Learning. — Cambridge University Press, 2020.