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
Anastasia Miroshnikova
j4133c
Accepted by
Dr Petr Chunaev

St. Petersburg
2020

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

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

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

where $\delta_k \in N(0,1)$ 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:

$$D(a, b, c, d) = \sum_{k=0}^{100} (F(x_k, a, b, c, d) - y_k)^2.$$

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 $\varepsilon = 0.001$ 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.).

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, 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 in-state. 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 meta-heuristic algorithm, and it makes no assumptions about the problem being optimized and can search very large spaces of candidate solutions. However, it it 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 simple mathematical formulae 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.

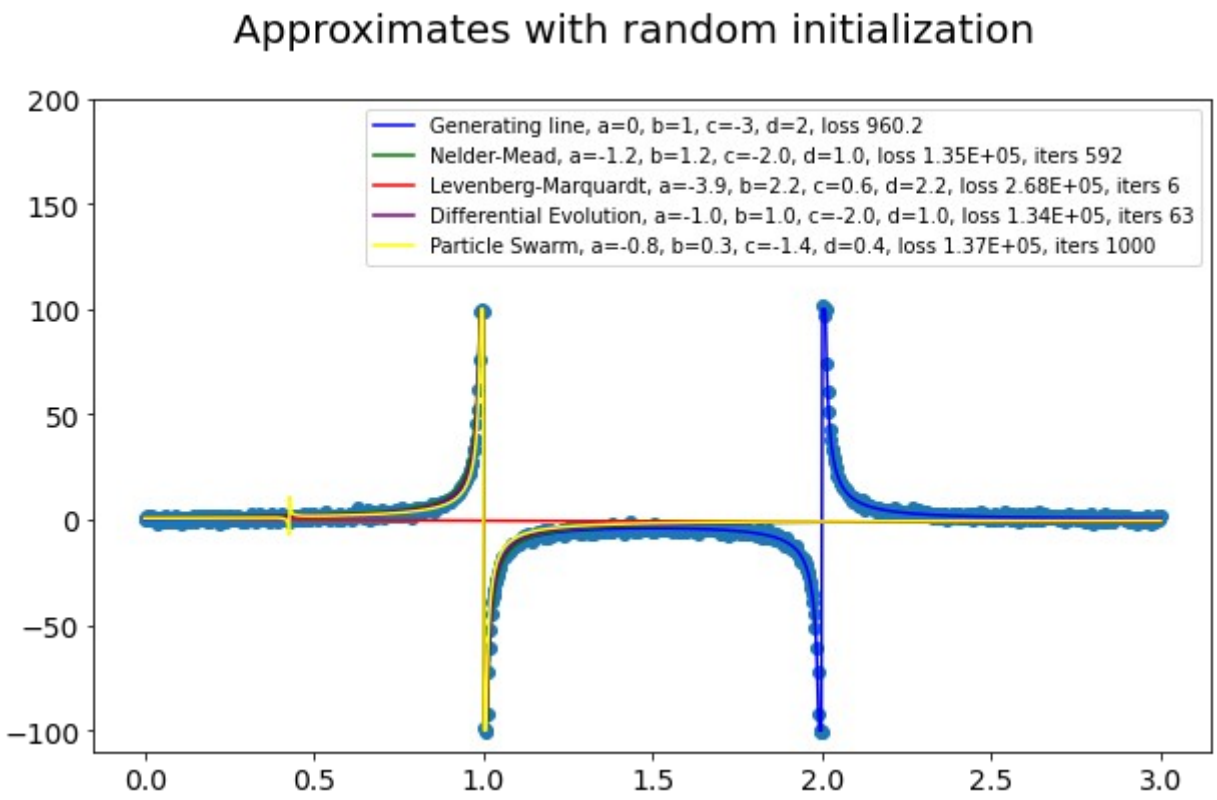
Results

Two meta-heuristic algorithms were chosen to solve the problem – differential evolution (*scipy* Python library) and particle swarm optimization (*pyswarms* Python library). *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 Nelder-Mead algorithm, which works under the assumption that a function to be optimized is convex, and 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 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 5 times with different initial parameters from $U(-4,4)$. They were also initialized with true values of parameters (known from generating formula) to see if the algorithms are able to find parameter values

which are more optimal than true values themselves. The best values of the parameters were chosen from the first 5 experiments with random initialization.

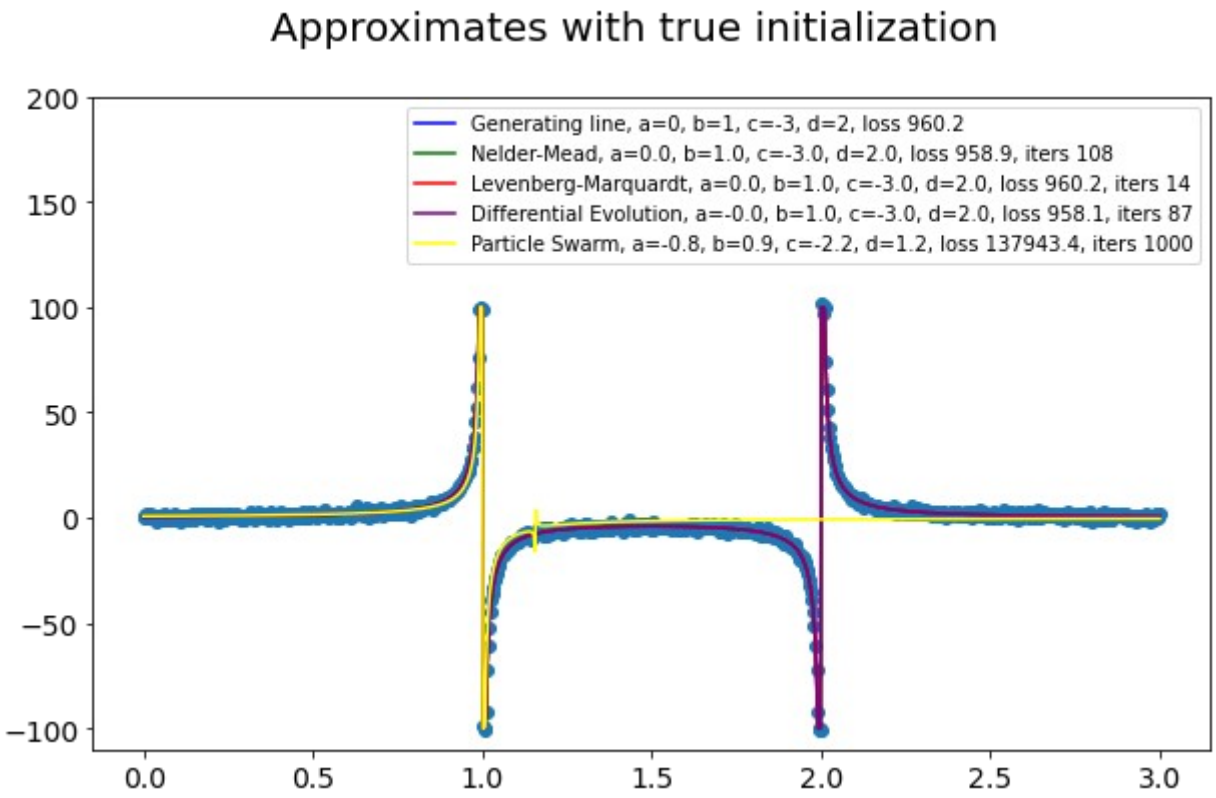


It can be seen that function with true parameter value has value of loss function about 10^3, while the approximations which captured one break have value of $1.35 \cdot 10^5$, and approximations which came to the nearly constant approximation – $2.7 \cdot 10^5$. Among the best experiments, Nelder-Mead algorithm, differential evolution and particle swarm optimization found 1 break, but Levenberg-Marquardt algorithm failed to capture any.

The analysis of 5 experiments with random parameter initialization shows that meta-heuristic algorithms were able to capture one break 2 times out of 5, and Nelder-Mead algorithm was able to do so only 1 time.

Method	Average loss	Breaks captured
Particle Swarm	1.96E+05	2
Differential Evolution	2.16E+05	2
Nelder-Mead	2.42E+05	1
Levenberg-Markquardt	2.72E+05	0

Initialization with true values leaded to the solution with optimal loss value for all algorithms but particle swarm optimization, which shifted to the local optimum and was again able to capture only one break. Differential evolution and Nelder-Mead algorithm were able to find solutions with loss value less than that with true parameter values.



Conclusions

The performance of two meta-heuristic methods – differential evolution and particle swarm optimization was analyzed and compared with that of direct Nelder-Mead algorithm and gradient-based Levenberg-Marquardt algorithm. Both meta-heuristic methods happened to be closer to the solution more often, than Nelder-Mead and Levenberg-Marquardt algorithms. Nevertheless, it was found that particle swarm optimization cannot find the best solution if area near the optimal values is too narrow.

Appendix

Source code can be found at https://github.com/Miffka/algo_itmo/tree/master/task4.