

FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION
OF HIGHER EDUCATION
ITMO UNIVERSITY

Report
on the practical task No. 3
“Algorithms for unconstrained nonlinear optimization. First- and second-
order methods”

Performed by
Anastasia Miroshnikova
j4133c
Accepted by
Dr Petr Chunaev

St. Petersburg
2020

Goal

The use of first- and second-order methods (Gradient Descent, Conjugate Gradient Descent, Newton's method and Levenberg-Marquardt algorithm) in the tasks of unconstrained nonlinear optimization.

Formulation of the problem

Generate random numbers $\alpha \in (0,1)$ and $\beta \in (0,1)$. Furthermore, generate the noisy data $\{x_k, y_k\}$, where $k=0, \dots, 100$, according to the following rule:

$$y_k = \alpha x_k + \beta + \delta_k, x_k = \frac{k}{100},$$

where $\delta_k \sim N(0,1)$ are values of a random variable with standard normal distribution. Approximate the data by the following linear and rational functions:

1. $F(x, a, b) = ax + b$ (linear approximant),
2. $F(x, a, b) = \frac{a}{1+bx}$ (rational approximant),

by means of least squares through the numerical minimization (with precision $\varepsilon=0.001$) of the following function:

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

To solve the minimization problem, use the methods of Gradient Descent, Conjugate Gradient Descent, Newton's method and Levenberg-Marquardt algorithm. If necessary, set the initial approximations and other parameters of the methods. Visualize the data and the approximants obtained **separately for each type of approximate**. Analyze the results obtained (in terms of number of iterations, precision, number of function evaluations, etc.) and compare them with those from Task 2 for the same dataset.

Brief theoretical part

Gradient descent is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function. To find a local minimum of a function using gradient descent, one should take steps proportional to the negative of the gradient (or approximate gradient) of the function at the current point. Gradient descent is based on the observation that if the multi-variable function is defined and differentiable in a neighborhood of a point, then it decreases fastest if one goes from that point in the direction of the negative gradient of F .

The idea of the **conjugate gradient method** is that instead of doing small steps in the direction of the negative gradient, we find a minimum of the given function in that direction, and then make a huge step to this point of minimum, and then repeat. This method is quite effective in solving unconstrained optimization problems such as energy minimization.

Newton's method in optimization is applied to the derivative f' of a twice-differentiable function f to find the roots of the derivative (solutions to $f'(x) = 0$), also known as the stationary points of f . The geometric interpretation of **Newton's**

method is that at each iteration, it amounts to the fitting of a paraboloid to the surface of $f(x)$ at the trial value x_k having the same slopes and curvature as the surface at that point, and then proceeding to the maximum or minimum of that paraboloid (in higher dimensions, this may also be a saddle point). If f happens to be a quadratic function, then the exact extremum is found in one step.

The **Levenberg–Marquardt** algorithm, also known as the damped least-squares (DLS) method, is used to solve non-linear least squares problems. This method is a combination of two minimization methods: the **gradient descent** method and the **Gauss-Newton** method. In the gradient descent method, the sum of the squared errors is reduced by updating the parameters in the steepest-descent direction. In the Gauss-Newton method, the sum of the squared errors is reduced by assuming the least squares function is locally quadratic and finding the minimum of the quadratic. The Levenberg-Marquardt method acts more like a gradient-descent method when the parameters are far from their optimal value, and acts more like the Gauss-Newton method when the parameters are close to their optimal value.

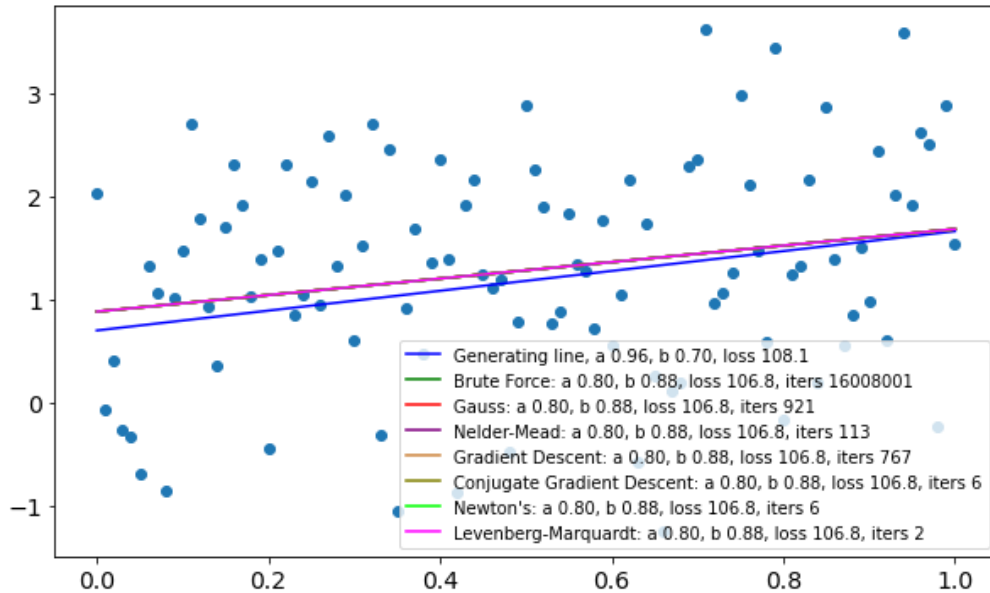
Results

The gradient descent method was implemented using Python 3.6.9 programming language. Conjugate gradient descent, Newton's method, and Levenberg-Marquardt algorithms implementations were taken from **scipy** library. The gradients of the both linear and rational approximates with respect to parameters were calculated in algebraic form. Two methods were used to determine the end of the iterative algorithms: 1) norm of the gradient vector became less than $1 \cdot 10^{-4}$ or 2) the algorithm made 10 000 steps.

All algorithms converged to the same parameter values which coincide with parameters obtained by direct methods – brute-force, Gauss method, and Nelder-Mead.

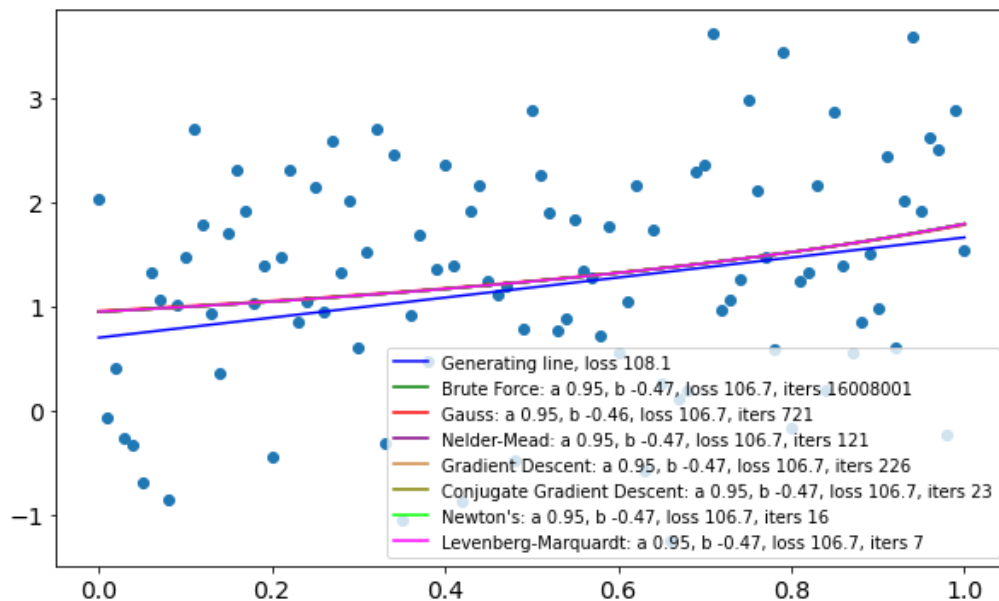
In both linear and rational cases of approximates, all gradient-based method required less function calls than direct methods, except for vanilla Gradient Descent with static learning rate which required less function calls than Gauss method, but more than Nelder-Mead algorithm.

Linear approximates



The most effective in terms of function calls is Levenberg-Marquardt algorithm which required only 2 calls in the case of linear approximate and 7 calls in the case of rational approximate. The other two methods – Conjugate Gradient Descent and Newton's method – required 6 function calls each for the linear approximate and 23 and 16 calls in the case of rational approximate. Best gradient-based method - Levenberg-Marquardt required 20-50 times less function calls than the best direct method for linear type of approximate.

Rational approximates



Conclusions

Gradient-based methods of optimization were used to solve two-dimensional optimization problem introduced in the previous assignment. All methods converged to the same values of parameters as direct methods. The most effective among gradient-based methods in terms of function calls was Levenberg-Marquardt algorithm. Although this algorithm can only be used to solve least-squares problem with quadratic loss.

Appendix

Source code can be found at https://github.com/Miffka/algo_itmo/tree/master/task3.