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

*Vdovkina Sophia*

*Syrchenko Arina*

*Academic group: J4133c*

Accepted by

Dr Petr Chunaev

St. Petersburg

2022

**Goal**

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

**Formulation of the problem**

Generate random numbers and . Furthermore, generate the noisy data , where , according to the following rule:

,

where are values of a random variable with standard normal distribution. Approximate the data by the following linear and rational functions:

1. (linear approximant)
2. (rational approximation)

by means of least squares through the numerical minimization (with precision ) of the following function:

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 in a plot separately for each type of approximant so that one can compare the results for the numerical methods used. 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**

If it is possible to calculate first- and second-order derivatives of a function, this information can be successfully utilized in solving the optimization problem.

*Gradient Descent*

Gradient descent is a first-order iterative optimization method for differentiable functions. Gradient descent is based on the observation that if the multi-variable function is defined and differentiable in a neighborhood of a point *a*, then *f(x)* decreases fastest in the direction of . It can be written down as the following formula:

Gradient descent algorithm may converge faster, if step size is chosen correctly.

*Conjugate gradient descent*

The conjugate gradient descent is an iterative method for unconditional optimization in a multidimensional space. It works when the function is approximately quadratic near the minimum, which is the case when the function is twice differentiable at the minimum and the second derivative is non-singular there.

This method simulates such a physical quantity as momentum in order to step out of the local minimum. The conjugate gradient method is seen as an evolution of the gradient descent where instead of doing small steps towards the negative gradient, it uses an adjustable step length and performs a line search in this direction until it reaches the minimum.

*Newton’s method*

Newton's method in optimization is applied to the derivative *f ′* of a twice-differentiable function *f* to find roots of constructing a sequence from an initial approximation so that as , where

From the Taylor expansion of *f* near ,

.

We use this quadratic function (with respect to ) as an approximant to *f* in a neighborhood of . The vertex of the corresponding parabola gives us the next point . To find the vertex x-coordinate, we write:

. Incrementing by this gives us a point closer :

It is proved that for the chosen class of *f* one has as .

Newton’s method uses Hessian and its inverse to find the optimum. Also, it requires the function to be convex and have invertible Hessian. Computing the Hessian can be a time demanding task. Quasi-Newton methods are aimed to find an approximation to the Hessian to simplify the task.

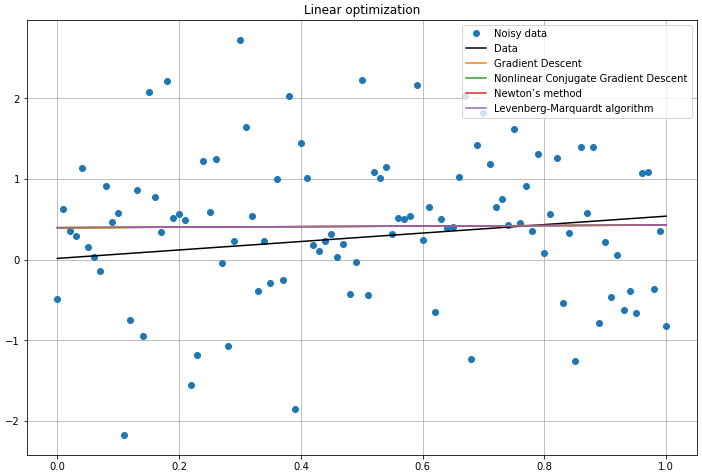
*Levenberg-Marquardt algorithm*

The Levenberg-Marquardt algorithm interpolates between the Gauss–Newton algorithm (GNA) and gradient descent 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 that the least squares function is locally quadratic and then it shall find the minimum of the function approximated by the Taylor expansion. The Levenberg-Marquardt method uses gradient-descent method when parameters are far from the optimal value. The Gauss-Newton method is used when parameters are close to the optimal value.

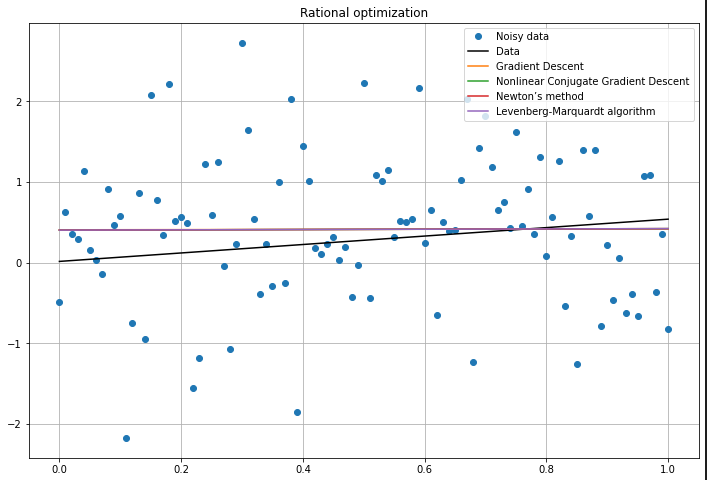
**Results**

To solve the optimization problem Gradient Descent, Conjugate Gradient Descent, Newton’s method and Levenberg-Marquardt algorithm were used.

Data was generated by the rules provided in the task, pictures 1 – 2 respectively show each of the approximating functions (linear and rational) using proposed methods.

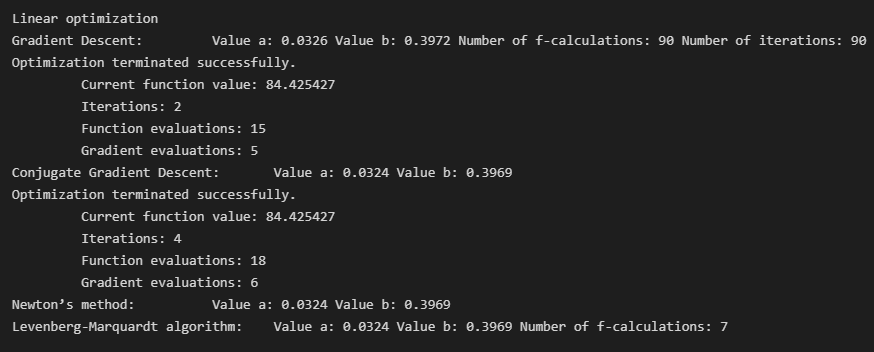


Picture 1 – linear optimization

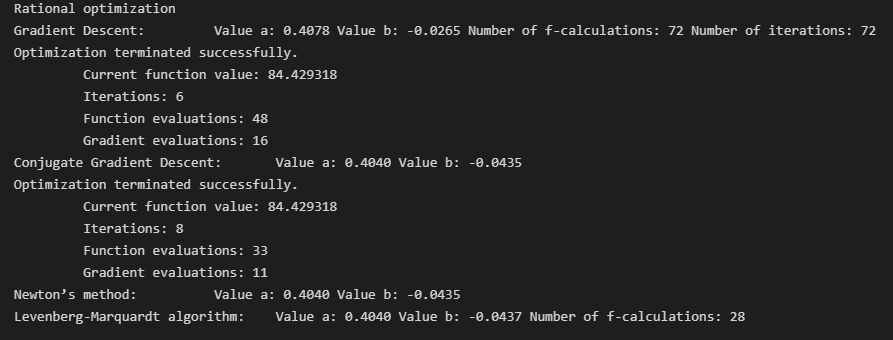


Picture 2 – rational optimization

As can be seen in the pictures 3 – 4 all algorithms converged to the parameter values that are almost identical with the slight difference, which cannot be seen on the plots.



Picture 3 – values obtained with linear optimization



Picture 4 – values obtained with rational optimization

The gradient descent method required the largest number of iterations and calculation of the function, since at each iteration it calculates the gradient and makes small steps in the opposite direction. Newton's method required fewer iterations and f-calculations than gradient descent, however, in the proportion, it calculates function values more often and it uses second order derivatives. Conjugate gradient method performed better in number of iterations as well as function calculations. Levenberg-Marquardt method is the efficient of them all as analytically it is known to make almost two times less iterations, than it does f-calculations.

Further, obtained results were compared with the results obtained by direct methods from Task 2. Number of function calculations, precision and iterations for each method are presented in Table 1. Precision was calculated as the distance between obtained values and initial values, so the smaller the number the more precise it is. For only number of function calculations was obtained, because *optimize.least\_squares* function does not provide any information about the number of iterations.

Table 1 – Methods comparison

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function type | Method | Precision | Function calculations | Iterations |
| Linear | Exhaustive search | 0.3659 | 46 | 23 |
| Gauss algorithm | 0.3659 | 60 | 3 |
| Function type | Method | Precision | Function calculations | Iterations |
| Linear | Nelder-Mead algorithm | 0.3656 | 51 | 28 |
| Gradient Descent | 0.3660 | 74 | 74 |
| Conjugate Gradient Descent | 0.3659 | 15 | 2 |
| Newron’s algorithm | 0.3659 | 15 | 2 |
| Levenberg-Marquardt algorithm | 0.3659 | 6 | - |
| Rational | Exhaustive search | 1.2579 | 104 | 54 |
| Gauss algorithm | 1.2580 | 125 | 5 |
| Nelder-Mead algorithm | 1.2578 | 64 | 34 |
| Gradient Descent | 1.2572 | 76 | 76 |
| Conjugate Gradient Descent | 1.2580 | 81 | 10 |
| Newron’s algorithm | 1.2580 | 36 | 8 |
| Levenberg-Marquardt algorithm | 1.2580 | 20 | - |

In both linear and rational cases of approximates, by and large, gradient-based method required less function calls than direct methods, except for the Gradient Descent method with static learning rate which required less function evaluations than Gauss and exhaustive search, but more than Nelder-Mead algorithm. It can be improved by calculating dynamic learning rate by Barzilai-Borwein method.

Precision remained almost the same for all methods.

**Conclusion**

Gradient-based methods of optimization were used to solve two-dimensional optimization problem. All algorithms converged to the parameter values that are almost identical with the slight difference. The most effective of the gradient-based methods in terms of function evaluations is Levenberg-Marquardt algorithm. During the comparison of direct methods with gradient-based methods it was stated that gradient-based methods generally perform better.

**Appendix**

Source code is available on <https://github.com/sophia-vdovkina/Analysis-and-development-of-algorithms/blob/main/Task%203/task3.ipynb>