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
Tiulkov Nikita
J4133c

Accepted by
Dr Petr Chunaev

St.Petersburg
2021

1. 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.

2. 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:

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

where δ_k $N(0,1)$ 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 approximant. 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.

3. Brief theoretical part

3.1. Gradient Descent method

Gradient descent is based on the observation that if $f(x)$ is defined and differentiable in a neighbourhood of a point a, then $f(x)$ decreases fastest in a neighbourhood of a in the direction of $-\nabla_a f$. One obtains the following formula:

$$a_{n+1} = a_n - \gamma \nabla_{a_n} f$$

for $\gamma \in \mathbb{R}_+$ small enough, then $f(a_n) \geq f(a_{n+1})$. With this observation in mind, one starts with a guess a_0 for a local minimum of f, and considers the sequence $\{a_n\}$ such that

$$a_{n+1} = a_n - \gamma \nabla_{a_n} f, n \geq 0$$

Here the value of the step size γ_n may be non-fixed and changed at every iteration (many possible ways to choose).

3.2. Conjugate Gradient Descent method

Given a function $f(x), x \in \mathbb{R}^n$ and an initial approximation a_0, one starts in the steepest descent direction:

$$\Delta a_0 = -\nabla_{a_0} f$$

Find the step length $\alpha_0 := argmin_\beta f(a_0 + \alpha \Delta a_0)$ and the next point $a_1 = a_0 + \alpha \Delta a_0$. After this iteration, the following steps constitute one iteration of moving along a subsequent conjugate direction s_n, where $s_0 = \Delta a_0$:

- Calculate the steepest direction $\Delta a_n = -\nabla_{a_n} f$.
- Compute β_n according to certain formulas.
- Update the conjugate direction $s_n = \Delta a_n + \beta_n s_{n-1}$.
- Find $\alpha_n = argmin_\alpha f(a_n + \alpha s_n)$.
- Update the position: $a_{n+1} = a_n + \alpha_n s_n$.

The choice of β_n due to Fletcher-Reeves:

$$\beta_n^{FR} = \frac{\Delta a_n^T \Delta a_n}{\Delta a_{n-1}^T \Delta a_{n-1}}$$

The choice of β_n due to Polak-Ribiere:

$$\beta_n^{PR} = \frac{\Delta a_n^T (\Delta a_n - \Delta a_{n-1})}{\Delta a_{n-1}^T \Delta a_{n-1}}$$

3.3. Newton's method

Let $f : \mathbb{R} \to \mathbb{R}$ be convex and twice-differentiable. Find the roots of f' by constructing a sequence an from a_n initial guess a_0 s.t. $a_n \to x^*$ as $n \to \infty$, where $f'(x^*) = 0$, i.e. x^* is a stationary point of f.

From the Taylor expansion of f near a_n (think that $x^* \approx a_n + \Delta a$),

$$f(a_n + \Delta a) \approx T_f(\Delta a) := f(a_n) + f'(a_n)\Delta a + \frac{1}{2} f''(a_n)(\Delta a)^2$$

Use this quadratic functions as approximants to f in a neighbourhood of a_n and find their minimum points (take into account that $f''(x) > 0$):

$$0 = \frac{dT_f(\Delta a)}{d\Delta a} = f'(a_n) + f''(a_n)\Delta a \Rightarrow \Delta a = -\frac{f'(a_n)}{f''(a_n)}$$

Incrementing a_n by this Δa yields a point closer to x^*:

$$a_{n+1} = a_n + \Delta a = a_n - \frac{f'(a_n)}{f''(a_n)}$$

It is proved that for the chosen class of f, $a_n \to x^*$ as $n \to \infty$.

3.4. Levenberg-Marquardt algorithm

The application of LMA is the **least-squares curve fitting problem**: given a set $(x_i, y_i)_{i=1}^m$, find the parameters β (column vector) of the model curve $f(x, \beta)$ so that the sum of the squares of the deviations $S(\beta)$ is minimized:

$$arg\min_\beta S(\beta) \equiv arg\min_\beta \sum_{i=1}^m [y_i - f(x_i, \beta)]^2$$

Start with an initial guess for β. In each iteration step, the parameter vector β is replaced by a new estimate $\beta + \Delta\beta$. To determine $\Delta\beta$, the function $f(x_i, \beta + \Delta\beta)$ is approximated by its linearization:

$$f(x_i, \beta + \Delta\beta) \approx f(x_i, \beta) + J_i\Delta\beta, J_i = (\nabla_{x_i} f(\beta))^T$$

The sum $S(\beta)$ has its minimum at a zero gradient with respect to β. The above first-order approximation of $f(x_i, \beta + \Delta\beta)$ gives

$$S(\beta + \Delta\beta) \approx \sum_{i=1}^m [y_i - f(x_i, \beta) - J_i\Delta\beta]^2$$

or in vector notation,

$$S(\beta + \Delta\beta) \approx [y - f(\beta)]^T[y - f(\beta)] - 2[y - f(\beta)]^T J\Delta\beta + \Delta\beta^T J^T J\Delta\beta,$$

where J is the Jacobian matrix, whose i-th row equals J_i , and where $f(\beta)$ and y are vectors with i-th component $f(x_i, \beta)$ and y_i , respectively.

Taking the derivative of $S(\beta + \Delta\beta)$ with respect to $\Delta\beta$ and setting to zero gives

$$(J^T J)\Delta\beta = J^T[y - f(\beta)],$$

that is in fact a system of linear equations with respect to $\Delta\beta$.

The system may be replaced by

$$(J^T J + \lambda I)\Delta\beta = J^T[y - f(\beta)],$$

where **I** is the identity matrix, giving the increment $\Delta\beta$ to the estimated parameter vector β.

4. Results

function name	method name	a, b	iterations
D_linear	Gradient Descent	[0.5009323764523131, 1.0356589877630593]	59
D_linear	Newton’s method	[0.5039486389356569, 1.0359324051989947]	2
D_linear	Conjugate Gradient Descent	[0.5039486736475228, 1.0359324002434107]	2
D_linear	Levenberg-Marquardt algorithm	[0.4999980652250948, 1.0391730290120378]	36
D_rational	Gradient Descent	[1.0729581684909388, -0.3146800321393276]	68
D_rational	Newton’s method	[1.0713381481952011, -0.3165058169382578]	9
D_rational	Conjugate Gradient Descent	[1.0714687969782188, -0.3163675460078928]	13
D_rational	Levenberg-Marquardt algorithm	[1.0861692486887515, -0.304212718556741]	39

Figure 1: Result table for linear and rational approximants

The Gradient Descent method was implemented natively, and since it was the simplest of the proposed ones, the number of iterations for this method is the largest. The Levenberg-Marquardt algorithm follows next in efficiency, it was used by implemented method in Python (optimize.root + method = lm). The Newton’s method and Conjugate Gradient Descent implemented in scipy showed the fastest results. Comparing the results obtained with the results from the second task, it seems clear that the methods of the first- and second-orders show a more effective result than direct methods.

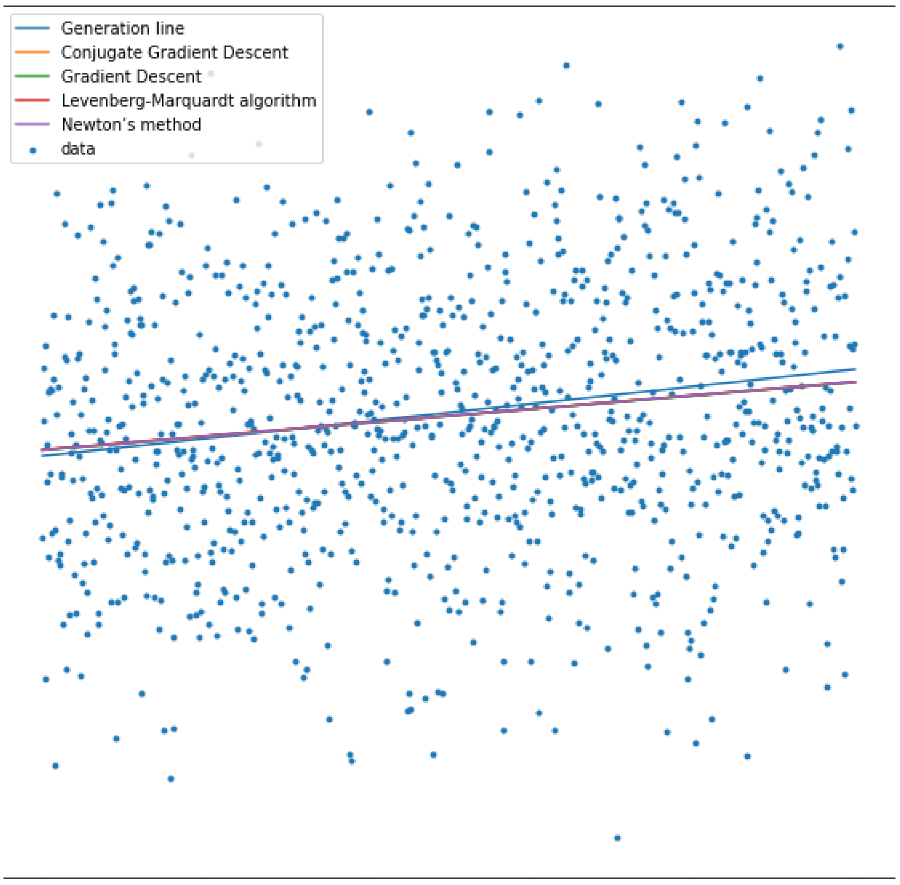

Figure 2: Linear approximant

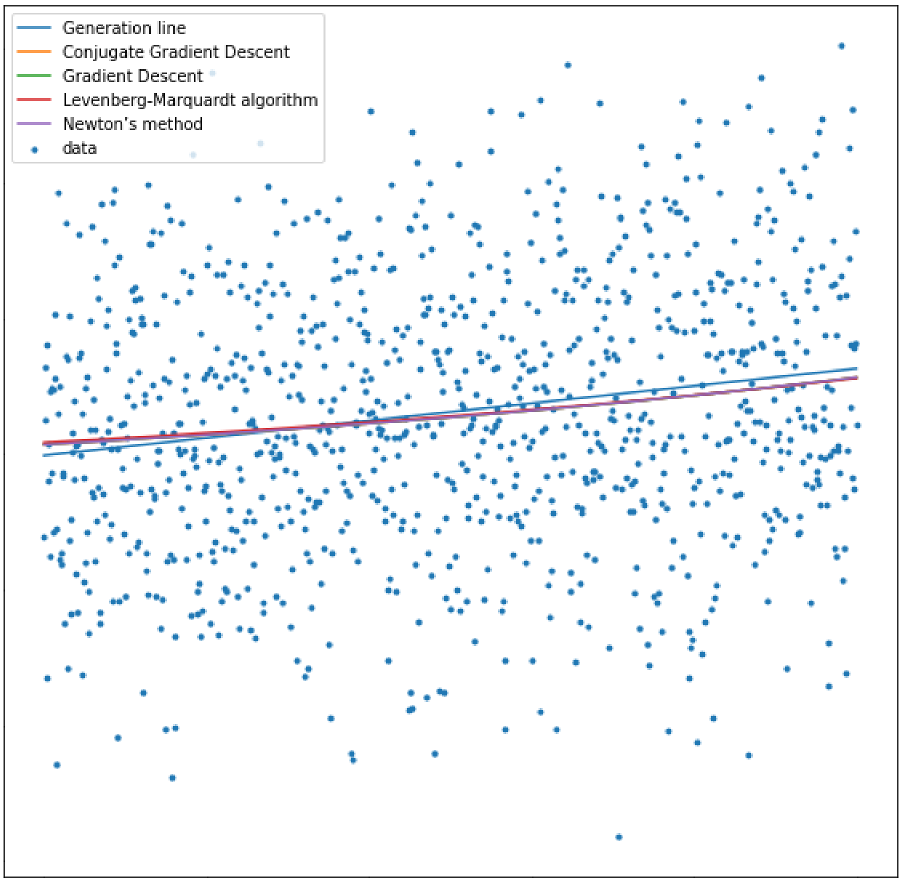

Figure 3: Rational approximant

5. Conclusions

To sum it up, we have measured the amount of iterations of well-known algorithm implementations with specified values, using data plots, representing function and points of steps.

First- and second-order algorithms showed faster iteration results than direct methods. Among these methods the Gradient Descent was the slowest one, Newton's and Conjugate Gradient Descent the fastest.

Appendix

The code of algorithms you can find on GitHub: https://github.com/FranticLOL/ITMO_Algorithms