General formulation

$$
\begin{split}
& \min_{x \in \mathbb{R}^n} f(x) \\
\text{s.t. } g_i(x) \leq& 0, \; i = 1, \ldots, m \\
h_j(x) =& 0, \; j = 1, \ldots, k
\end{split}
$$

Some necessary or/and sufficient conditions are known (See Optimality conditions. KKT and Convex optimization problem)

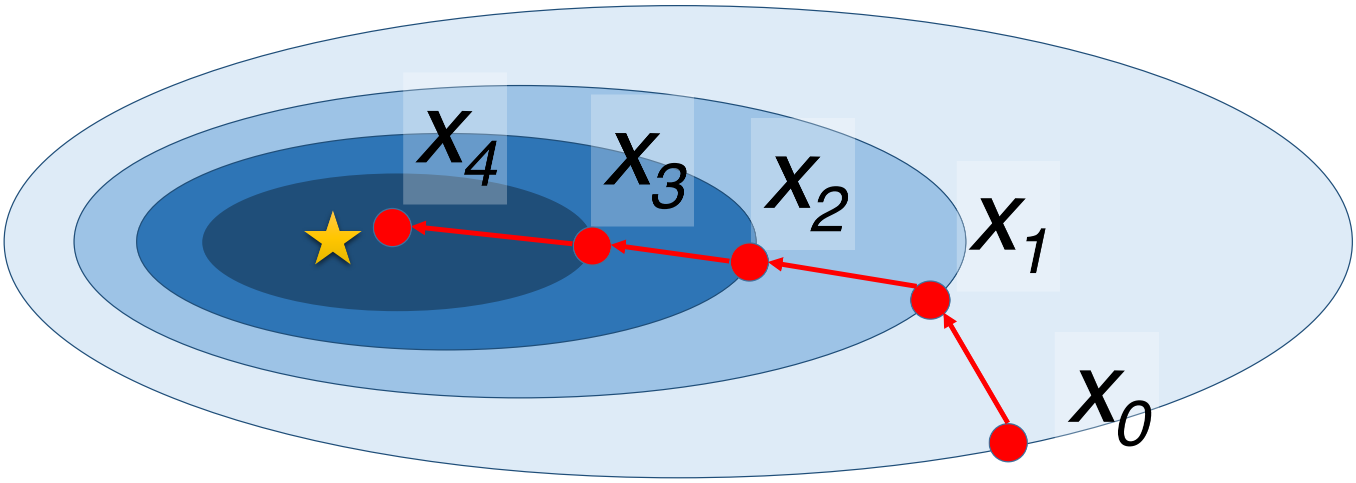
- In fact, there might be very challenging to recognize the convenient form of optimization problem.
- Analytical solution of KKT could be inviable.

Iterative methods

Typically, the methods generate an infinite sequence of approximate solutions

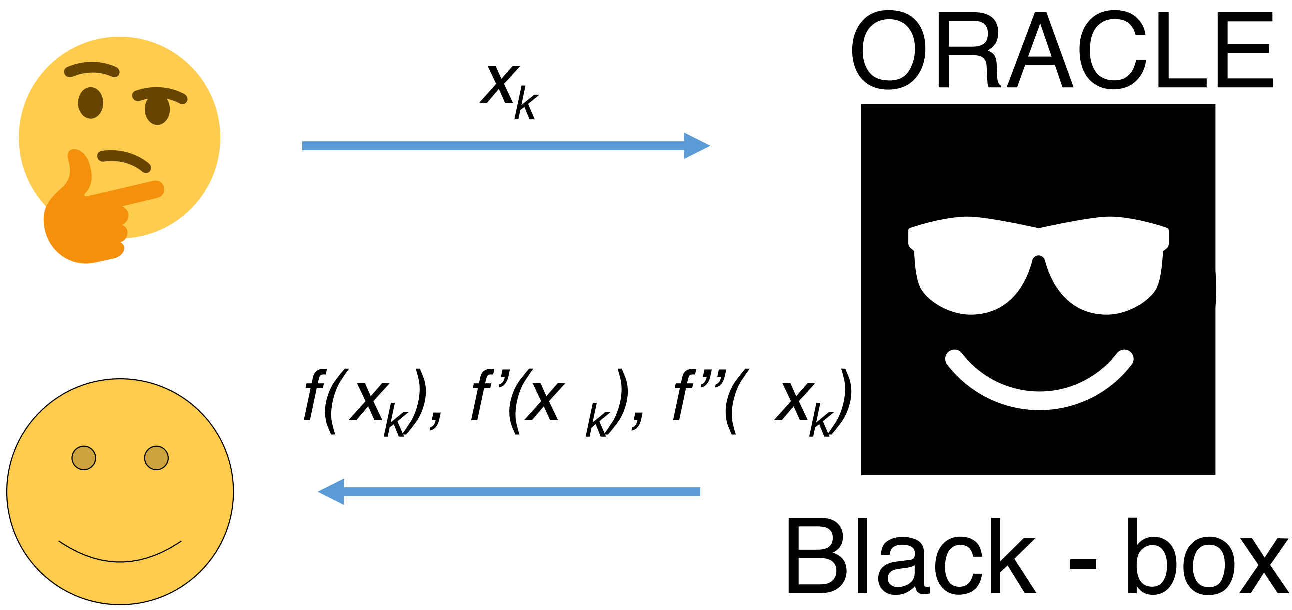
$$
\{x_t\},
$$

which for a finite number of steps (or better - time) converges to an optimal (at least one of the optimal) solution x_*.




```
def GeneralScheme(x, epsilon):
    while not StopCriterion(x, epsilon):
        OracleResponse = RequestOracle(x)
        x = NextPoint(x, OracleResponse)
    return x
```

Oracle conception



Complexity

Challenges

Unsolvability

In general, **optimization problems are unsolvable.** ¯_(ツ)_/¯

Consider the following simple optimization problem of a function over unit cube:

$$
\begin{split}
& \min_{x \in \mathbb{R}^n} f(x) \\
\text{s.t. } & x \in \mathbb{B}^n
\end{split}
$$

We assume, that the objective function $f(\cdot) : \mathbb{R}^n \to \mathbb{R}$ is Lipschitz continuous on $\mathbb{B}^n$:

$$
|f(x) - f(y)| \leq L\|x - y\|_{\infty} \forall x, y \in \mathbb{B}^n,
$$

with some constant L (Lipschitz constant). Here $\mathbb{B}^n$ - the n-dimensional unit cube

$$
\mathbb{B}^n = \{x \in \mathbb{R}^n \mid 0 \leq x_i \leq 1, i = 1, \dots, n\}
$$

Our goal is to find such $\tilde{x} : |f(\tilde{x}) - f^*| \leq \varepsilon$ for some positive ε. Here f^* is the global minima of the problem. Uniform grid with p points on each dimension guarantees at least this quality:

$$\|\tilde{x} - x_*\|_\infty \leq \frac{1}{2p},$$

which means, that

$$|f(\tilde{x}) - f(x_*)| \leq \frac{L}{2p}$$

Our goal is to find the p for some ε. So, we need to sample $\left(\frac{L}{2\varepsilon}\right)^n$ points, since we need to measure function in p^n points. Doesn't look scary, but if we'll take $L = 2, n = 11, \varepsilon = 0.01$, computations on the modern personal computers will take 31,250,000 years.

Stopping rules

- Argument closeness:

$$\|x_k - x_*\|_2 < \varepsilon$$

- Function value closeness:

$$\|f_k - f^*\|_2 < \varepsilon$$

- Closeness to a critical point

$$\|f'(x_k)\|_2 < \varepsilon$$

But x_* and $f^* = f(x_*)$ are unknown!

Sometimes, we can use the trick:

$$\|x_{k+1} - x_k\| = \|x_{k+1} - x_k + x_* - x_*\| \leq \|x_{k+1} - x_*\| + \|x_k - x_*\| \leq 2\varepsilon$$

Note: it's better to use relative changing of these values, i.e. $\dfrac{\|x_{k+1} - x_k\|_2}{\|x_k\|_2}$.

Local nature of the methods

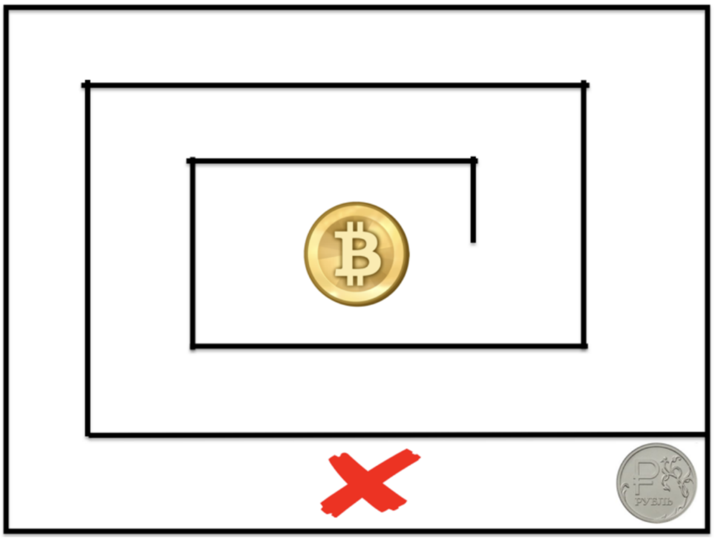


TABLE OF CONTENTS

Problem

Suppose, we have a problem of minimization of a function $f(x) : \mathbb{R} \to \mathbb{R}$ of scalar variable:

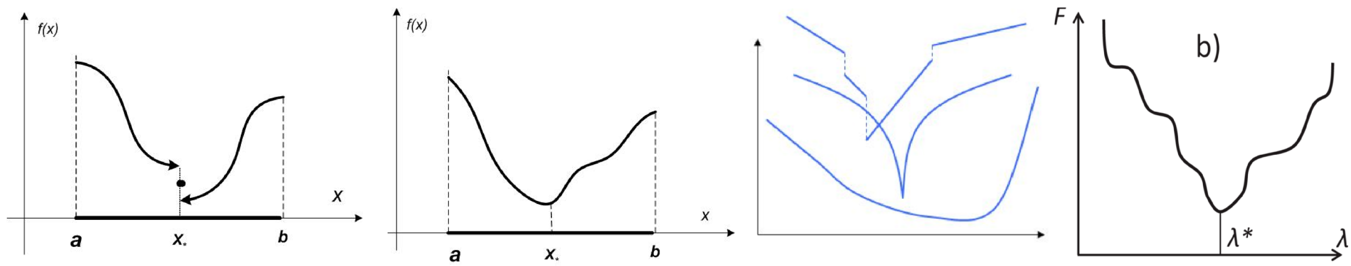
$$
f(x) \to \min_{x \in \mathbb{R}}
$$

Sometimes, we refer to the similar problem of finding minimum on the line segment $[a, b]$:

$$
f(x) \to \min_{x \in [a,b]}
$$

Line search is one of the simplest formal optimization problems, however, it is an important link in solving more complex tasks, so it is very important to solve it effectively. Let's restrict the class of problems under consideration where $f(x)$ is a *unimodal function*.

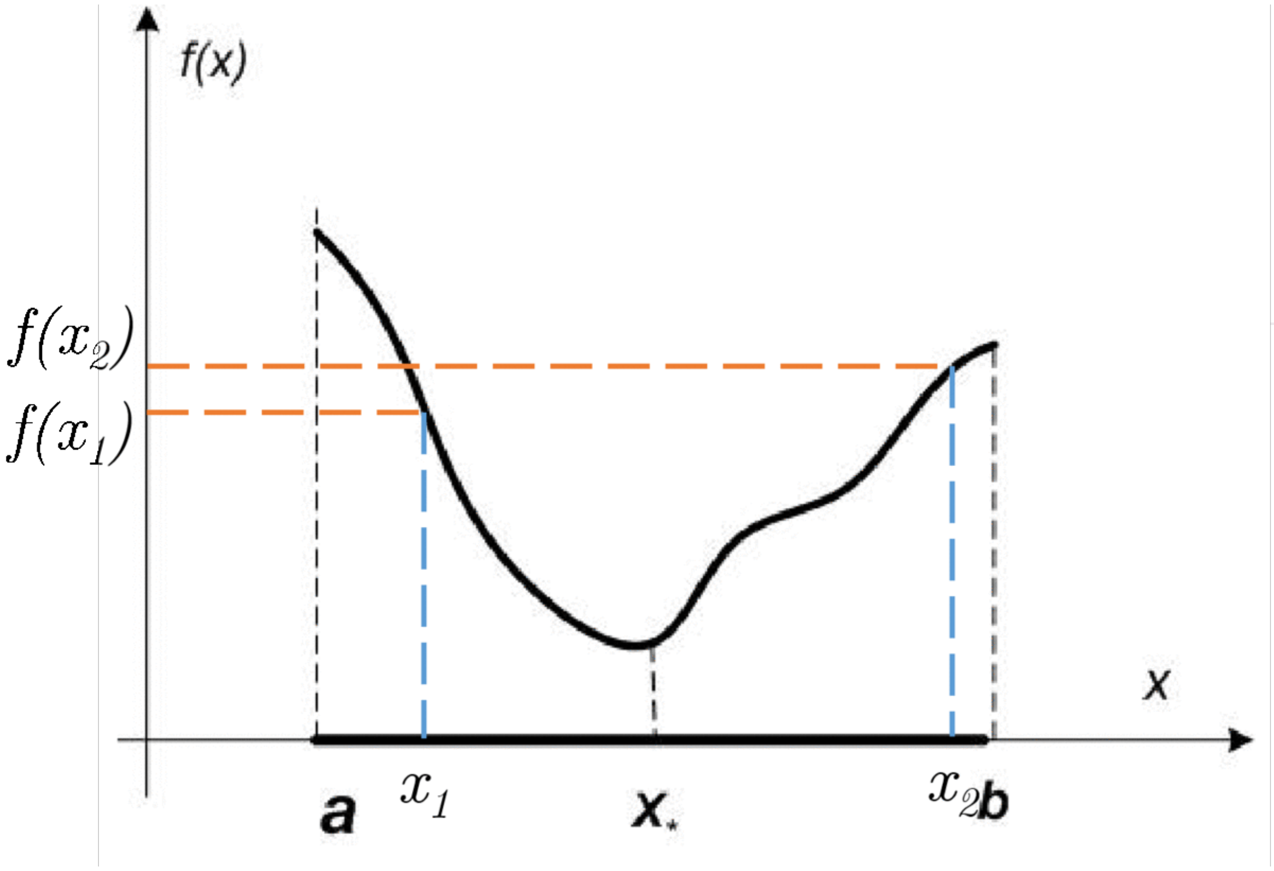
Function $f(x)$ is called **unimodal** on $[a, b]$, if there is $x_* \in [a, b]$, that $f(x_1) > f(x_2) \quad \forall a \le x_1 < x_2 < x_*$ and $f(x_1) < f(x_2) \quad \forall x_* < x_1 < x_2 \le b$



Key property of unimodal functions

Let $f(x)$ be unimodal function on $[a, b]$. Than if $x_1 < x_2 \in [a, b]$, then:

- if $f(x_1) \le f(x_2) \to x_* \in [a, x_2]$
- if $f(x_1) \ge f(x_2) \to x_* \in [x_1, b]$



Code

Open in Colab

References

- CMC seminars (ru)

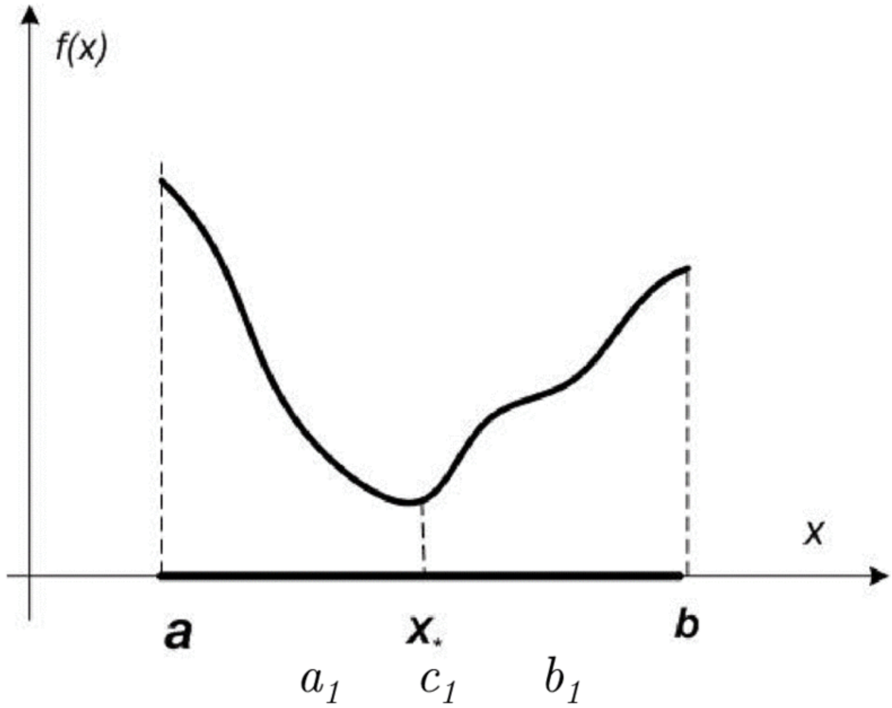
TABLE OF CONTENTS

Idea

We divide a segment into two equal parts and choose the one that contains the solution of the problem using the values of functions.

Algorithm

```
def binary_search(f, a, b, epsilon):
    c = (a + b) / 2
    while abs(b - a) > epsilon:
        y = (a + c) / 2.0
        if f(y) <= f(c):
            b = c
            c = y
        else:
            z = (b + c) / 2.0
            if f(c) <= f(z):
                a = y
                b = z
            else:
                a = c
                c = z
    return c
```



Bounds

The length of the line segment on $k+1$-th iteration:

$$\Delta_{k+1} = b_{k+1} - a_{k+1} = \frac{1}{2^k}(b-a)$$

For unimodal functions, this holds if we select the middle of a segment as an output of the iteration x_{k+1}:

$$|x_{k+1} - x_*| \leq \frac{\Delta_{k+1}}{2} \leq \frac{1}{2^{k+1}}(b-a) \leq (0.5)^{k+1} \cdot (b-a)$$

Note, that at each iteration we ask oracle no more, than 2 times, so the number of function evaluations is $N = 2 \cdot k$, which implies:

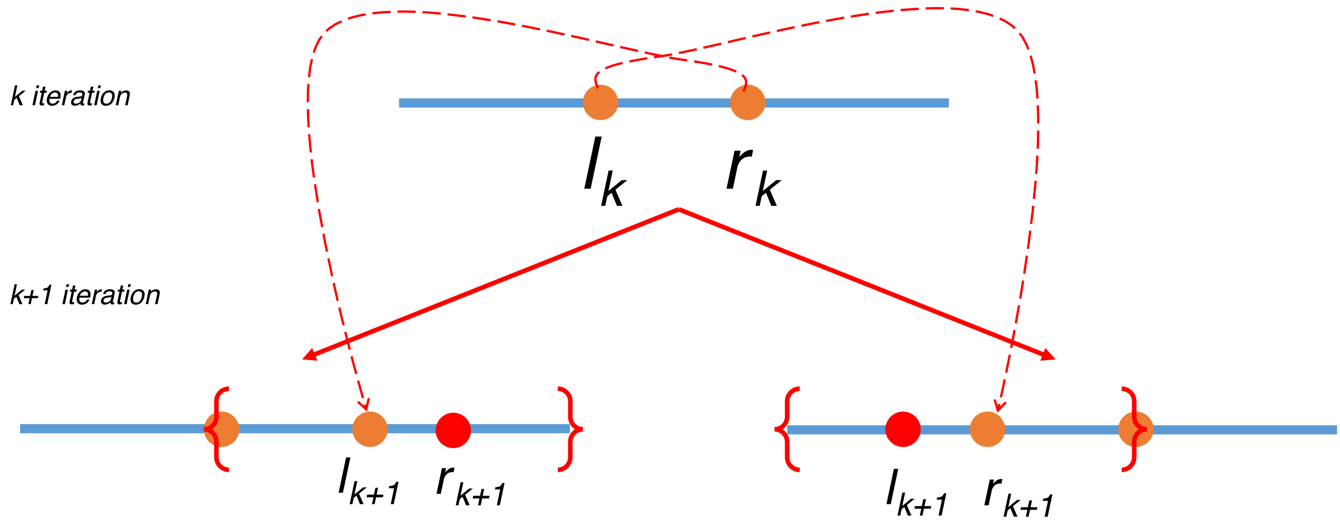
$$|x_{k+1} - x_*| \leq (0.5)^{\frac{N}{2}+1} \cdot (b-a) \leq (0.707)^N \frac{b-a}{2}$$

By marking the right side of the last inequality for ε, we get the number of method iterations needed to achieve ε accuracy:

$$K = \left\lceil \log_2 \frac{b-a}{\varepsilon} - 1 \right\rceil$$

Idea

The idea is quite similar to the dichotomy method. There are two golden points on the line segment (left and right) and the insightful idea is, that on the next iteration one of the points will remain the golden point.



Algorithm

```
def golden_search(f, a, b, epsilon):
    tau = (sqrt(5) + 1) / 2
    y = a + (b - a) / tau**2
    z = a + (b - a) / tau
    while b - a > epsilon:
        if f(y) <= f(z):
            b = z
            z = y
            y = a + (b - a) / tau**2
        else:
            a = y
            y = z
            z = a + (b - a) / tau
    return (a + b) / 2
```

Bounds

$$|x_{k+1} - x_*| \leq b_{k+1} - a_{k+1} = \left(\frac{1}{\tau} \right)^{N-1} (b - a) \approx 0.618^k (b - a),$$

where $\tau = \frac{\sqrt{5}+1}{2}$.

- The geometric progression constant **more** than the dichotomy method - 0.618 worse than 0.5
- The number of function calls **is less** than for the dichotomy method - 0.707 worse than 0.618 - (for each iteration of the dichotomy method, except for the first one, the function is calculated no more than 2 times, and for the gold method - no more than one)

This strategy of inexact line search works well in practice, as well as it has the following geometric interpretation:

Sufficient decrease

Let's consider the following scalar function while being at a specific point of x_k:

$$\phi(\alpha) = f(x_k - \alpha \nabla f(x_k)), \alpha \geq 0$$

consider first order approximation of $\phi(\alpha)$:

$$\phi(\alpha) \approx f(x_k) - \alpha \nabla f(x_k)^\top \nabla f(x_k)$$

A popular inexact line search condition stipulates that α should first of all give sufficient decrease in the objective function f, as measured by the following inequality:

$$f(x_k - \alpha \nabla f(x_k)) \leq f(x_k) - c_1 \cdot \alpha \nabla f(x_k)^\top \nabla f(x_k)$$

for some constant $c_1 \in (0, 1)$. (Note, that $c_1 = 1$ stands for the first order Taylor approximation of $\phi(\alpha)$). This is also called Armijo condition. The problem of this condition is, that it could accept arbitrary small values α, which may slow down solution of the problem. In practice, c_1 is chosen to be quite small, say $c_1 \approx 10^{-4}$.

Curvature condition

To rule out unacceptably short steps one can introduce a second requirement:

$$-\nabla f(x_k - \alpha \nabla f(x_k))^\top \nabla f(x_k) \geq c_2 \nabla f(x_k)^\top (-\nabla f(x_k))$$

for some constant $c_2 \in (c_1, 1)$, where c_1 is a constant from Armijo condition. Note that the left-handside is simply the derivative $\nabla_\alpha \phi(\alpha)$, so the curvature condition ensures that the slope of $\phi(\alpha)$ at the target point is greater than c_2 times the initial slope $\nabla_\alpha \phi(\alpha)(0)$. Typical values of $c_2 \approx 0.9$ for Newton or quasi-Newton method. The sufficient decrease and curvature conditions are known collectively as the Wolfe conditions.

Goldstein conditions

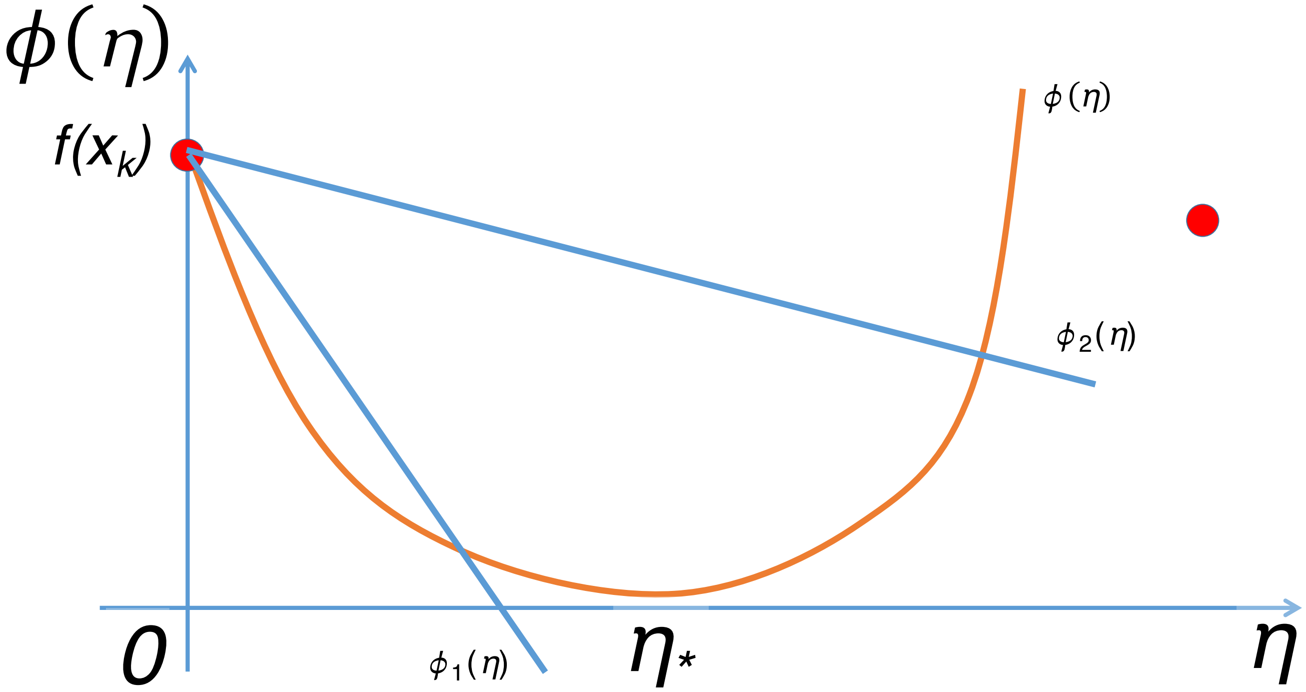
Let's consider also 2 linear scalar functions $\phi_1(\alpha), \phi_2(\alpha)$:

$$\phi_1(\alpha) = f(x_k) - c_1\alpha\|\nabla f(x_k)\|^2$$

and

$$\phi_2(\alpha) = f(x_k) - c_2\alpha\|\nabla f(x_k)\|^2$$

Note, that Goldstein-Armijo conditions determine the location of the function $\phi(\alpha)$ between $\phi_1(\alpha)$ and $\phi_2(\alpha)$. Typically, we choose $c_1 = \rho$ and $c_2 = 1 - \rho$, while $\rho \in (0.5, 1)$.



References

- Numerical Optimization by J.Nocedal and S.J.Wright.