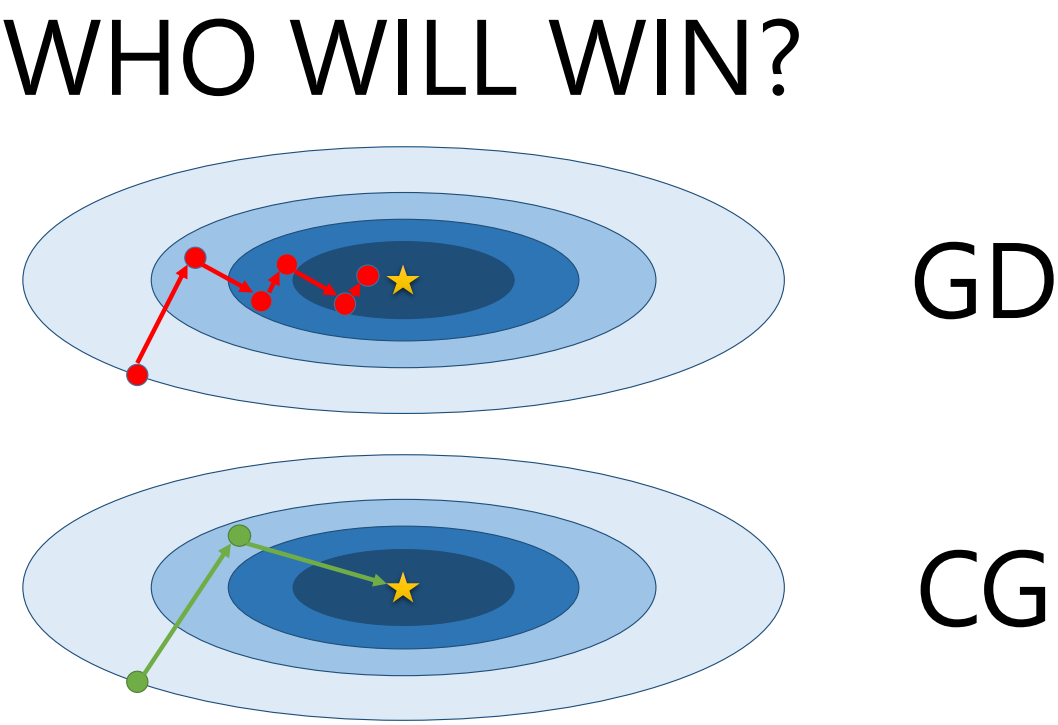
Conjugate gradients

Introduction



Originally, the conjugate gradients method was created to solve a system of linear equations.

$$Ax = b$$

Without special efforts the problem can be presented in the form of minimization of the quadratic function, and then generalized on a case of *non* quadratic function. We will start with the parabolic case and try to construct a conjugate gradients method for it. Let us consider the classical problem of minimization of the quadratic function:

$$f(x) = \frac{1}{2}x^\top Ax - b^\top x + c \to \min_{x \in \mathbb{R}^n}$$

Here $x \in \mathbb{R}^n, A \in \mathbb{R}^{n \times n}, b \in \mathbb{R}^n, c \in \mathbb{R}$.

Method of conjugate gradients for the quadratic function

We will consider symmetric matrices $A \in \mathbb{S}^n$ (otherwise, replacing $A' = \frac{A + A^\top}{2}$ leads to the same optimization problem). Then:

$$\nabla f = Ax - b$$

Then having an initial guess x_0, vector $d_0 = -\nabla f(x_0)$ is the direction of the fastest decrease. The procedure of the steepest descent in this direction is provided by the procedure of line search:

$$
\begin{aligned}
g(\alpha) &= f(x_0 + \alpha d_0) \\
&= \frac{1}{2}(x_0 + \alpha d_0)^\top A(x_0 + \alpha d_0) - b^\top (x_0 + \alpha d_0) + c \\
&= \frac{1}{2}\alpha^2 {d_0}^\top A d_0 + {d_0}^\top (Ax_0 - b)\alpha + (\frac{1}{2}{x_0}^\top A x_0 + {x_0}^\top d_0 + c)
\end{aligned}
$$

Assuming that the point of the zero derivative in this parabola is the minimum (for positive matrices it is guaranteed, otherwise it is not a fact), and also, rewriting this problem for the arbitrary (k) direction of the method, we have:

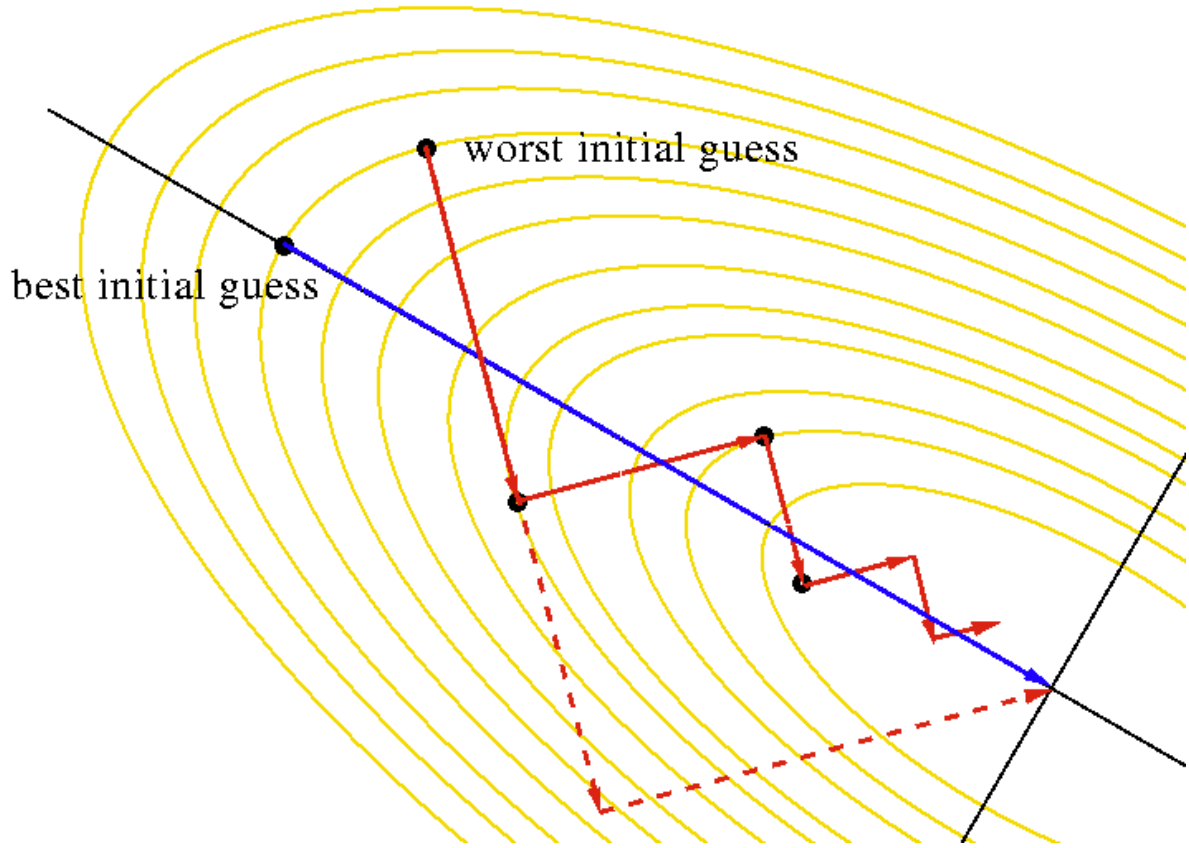
$$
g'(\alpha_k) = (d_k^\top A d_k)\alpha_k + d_k^\top (Ax_k - b) = 0
$$

$$
\alpha_k = -\frac{d_k^\top (Ax_k - b)}{d_k^\top A d_k} = \frac{d_k^\top d_k}{d_k^\top A d_k}.
$$

Then let's start our method, as the method of the steepest descent:

$$
x_1 = x_0 - \alpha_0 \nabla f(x_0)
$$

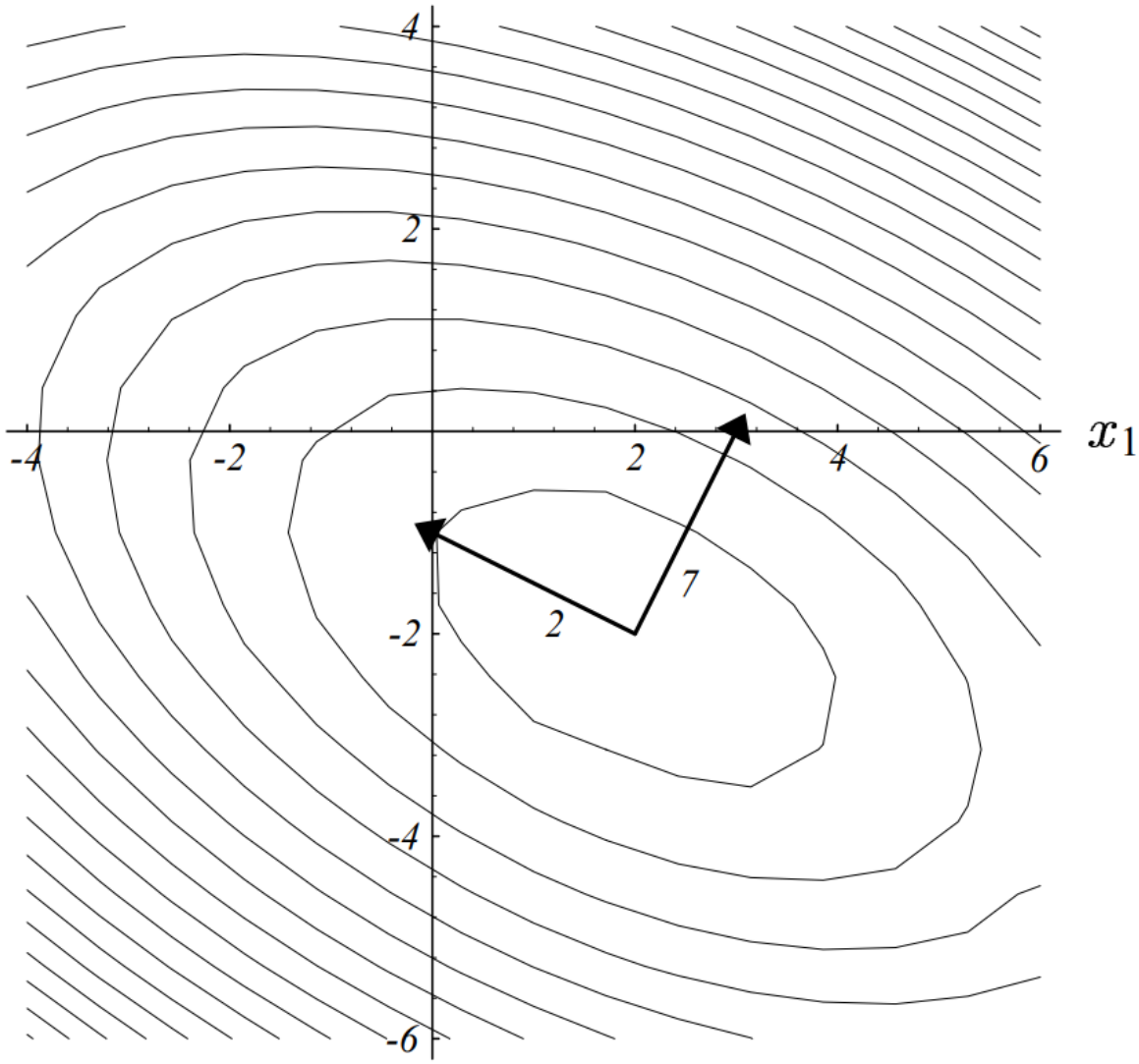
Note, however, that if the next step is built in the same way (the fastest descent), we will "lose" some of the work that was done in the first step and we will get a classic situation for the fastest descent:



In order to avoid this, we introduce the concept of A - conjugated vectors: let's say that two vectors x, y A - are conjugated relative to each other if they are executed:

$$
x^\top A y = 0
$$

This concept becomes particularly interesting when matrix A is positive defined, then x, y vectors will be orthogonal if the scalar product is defined by the matrix A. Therefore, this property is also called A - orthogonality.



Then we will build the method in such a way that the next direction is A - orthogonal with the previous one:

$$d_1 = -\nabla f(x_1) + \beta_0 d_0,$$

where β_0 is selected in a way that $d_1 \perp_A d_0$:

$$d_1^\top A d_0 = -\nabla f(x_1)^\top A d_0 + \beta_0 d_0^\top A d_0 = 0$$

$$\beta_0 = \frac{\nabla f(x_1)^\top A d_0}{d_0^\top A d_0}$$

It's interesting that all received A directions are A- orthogonal to each other. (proved by induction)

Thus, we formulate an algorithm:

1. Let $k = 0$ and $x_k = x_0$, count $d_k = d_0 = -\nabla f(x_0)$.
2. By the procedure of line search we find the optimal length of step:

Calculate α minimizing $f(x_k + \alpha_k d_k)$ by the formula

$$\alpha_k = -\frac{d_k^\top (A x_k - b)}{d_k^\top A d_k}$$

3. We're doing an algorithm step:

$$x_{k+1} = x_k + \alpha_k d_k$$

4. update the direction: $d_{k+1} = -\nabla f(x_{k+1}) + \beta_k d_k$, where β_k is calculated by the formula:

$$\beta_k = \frac{\nabla f(x_{k+1})^\top A d_k}{d_k^\top A d_k}.$$

5. Repeat steps 2-4 until n directions are built, where n is the dimension of space (dimension of x).

Method of conjugate gradients for non-quadratic function:

In case we do not have an analytic expression for a function or its gradient, we will most likely not be able to solve the one-dimensional minimization problem analytically. Therefore, step 2 of the algorithm is replaced by the usual line search procedure. But there is the following mathematical trick for the fourth point:

For two iterations, it is fair:

$$x_{k+1} - x_k = cd_k,$$

where c is some kind of constant. Then for the quadratic case, we have:

$$\nabla f(x_{k+1}) - \nabla f(x_k) = (Ax_{k+1} - b) - (Ax_k - b) = A(x_{k+1} - x_k) = cAd_k$$

Expressing from this equation the work $Ad_k = \frac{1}{c}(\nabla f(x_{k+1}) - \nabla f(x_k))$, we get rid of the "knowledge" of the function in step definition β_k, then point 4 will be rewritten as:

$$\beta_k = \frac{\nabla f(x_{k+1})^\top (\nabla f(x_{k+1}) - \nabla f(x_k))}{d_k^\top (\nabla f(x_{k+1}) - \nabla f(x_k))}.$$

This method is called the Polack - Ribier method.

Examples

Example 1

Prove that if a set of vectors $d_1, \ldots, d_k$ - A - are conjugated (all vectors are conjugated in pairs of A), these vectors are linearly independent. $A \in \mathbb{S}^n_{++}$.

Solution:

We'll show, that if $\sum\limits_{i=1}^{k} \alpha_k d_k = 0$, than all coefficients should be equal to zero:

$$\begin{aligned} 0 &= \sum_{i=1}^{n} \alpha_k d_k \\ &= d_j^\top A \left(\sum_{i=1}^{n} \alpha_k d_k \right) \\ &= \sum_{i=1}^{n} \alpha_k d_j^\top A d_k \\ &= \alpha_j d_j^\top A d_j + 0 + \ldots + 0 \end{aligned}$$

Thus, $\alpha_j = 0$, for all other indices one have perform the same process

References

- An Introduction to the Conjugate Gradient Method Without the Agonizing Pain
- The Concept of Conjugate Gradient Descent in Python by Ilya Kuzovkin
- Picture of best\worst initial guess in SD

Code

Open in Colab

Newton method

Intuition

Newton's method to find the equation' roots

Consider the function $\varphi(x) : \mathbb{R} \to \mathbb{R}$. Let there be equation $\varphi(x^*) = 0$. Consider a linear approximation of the function $\varphi(x)$ near the solution ($x^* - x = \Delta x$):

$$\varphi(x^*) = \varphi(x + \Delta x) \approx \varphi(x) + \varphi'(x)\Delta x.$$

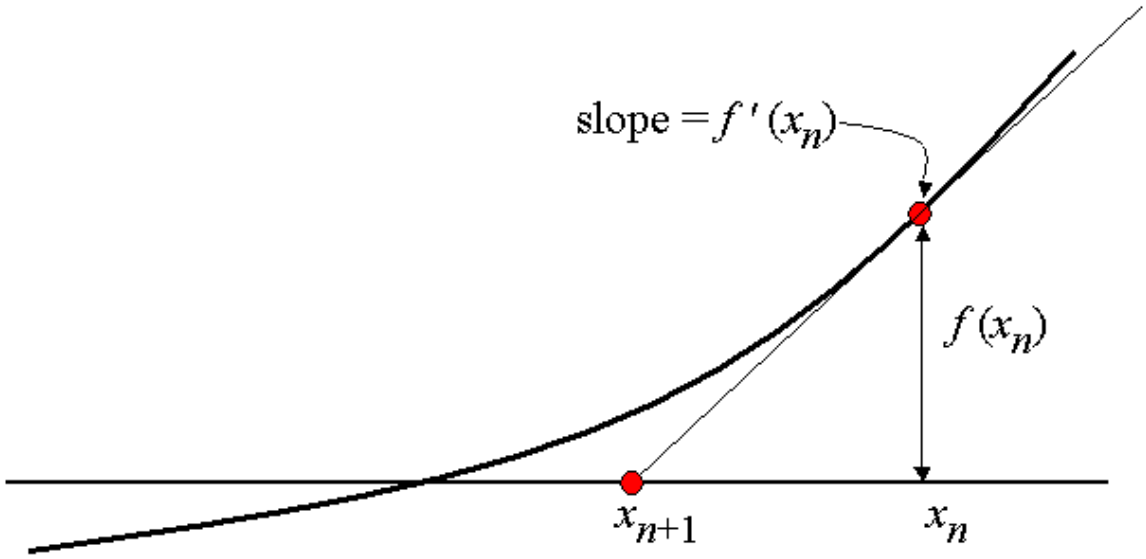
We get an approximate equation:

$$\varphi(x) + \varphi'(x)\Delta x = 0$$

We can assume that the solution to equation $\Delta x = -\dfrac{\varphi(x)}{\varphi'(x)}$ will be close to the optimal $\Delta x^* = x^* - x$.

We get an iterative scheme:

$$x_{k+1} = x_k - \frac{\varphi(x_k)}{\varphi'(x_k)}.$$



This reasoning can be applied to the unconditional minimization task of the $f(x)$ function by writing down the necessary extremum condition:

$$f'(x^*) = 0$$

Here $\varphi(x) = f'(x) \quad \varphi'(x) = f''(x)$. Thus, we get the Newton optimization method in its classic form:

$$x_{k+1} = x_k - [f''(x_k)]^{-1} f'(x_k). \qquad \text{(Newton)}$$

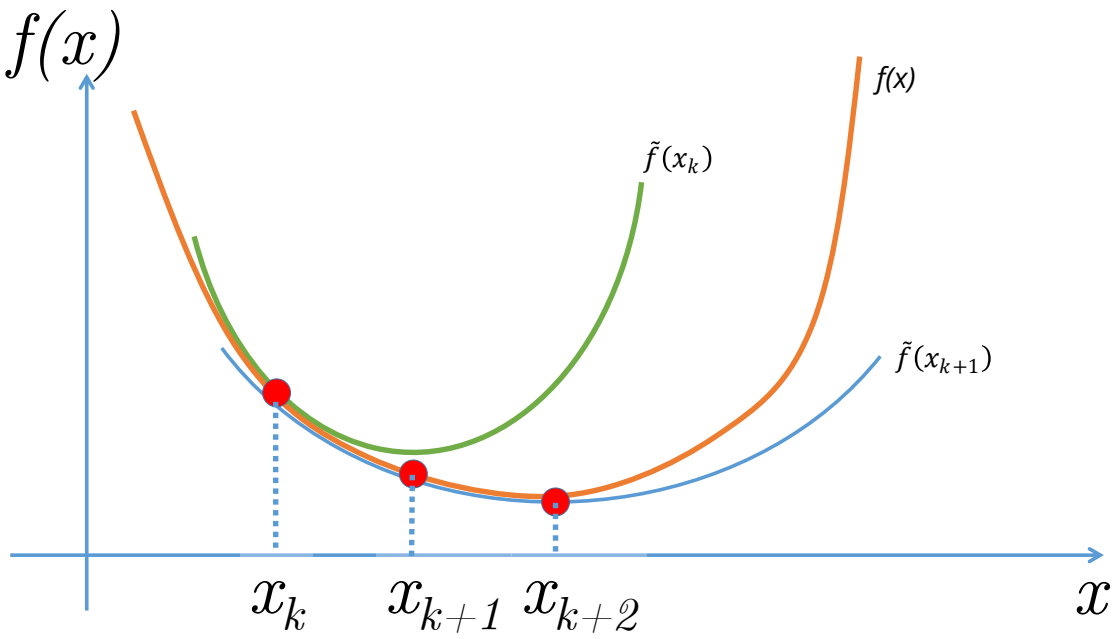
With the only clarification that in the multidimensional case:
$x \in \mathbb{R}^n,\ f'(x) = \nabla f(x) \in \mathbb{R}^n,\ f''(x) = \nabla^2 f(x) \in \mathbb{R}^{n \times n}$.

Second order Taylor approximation of the function

Let us now give us the function $f(x)$ and a certain point x_k. Let us consider the square approximation of this function near x_k:

$$\tilde{f}(x) = f(x_k) + \langle f'(x_k), x - x_k \rangle + \frac{1}{2}\langle f''(x_k)(x - x_k), x - x_k \rangle.$$

The idea of the method is to find the point x_{k+1}, that minimizes the function $\tilde{f}(x)$, i.e. $\nabla \tilde{f}(x_{k+1}) = 0$.



$$\begin{aligned}
\nabla \tilde{f}(x_{k+1}) &= f'(x_k) + f''(x_k)(x_{k+1} - x_k) = 0 \\
f''(x_k)(x_{k+1} - x_k) &= -f'(x_k) \\
[f''(x_k)]^{-1} f''(x_k)(x_{k+1} - x_k) &= -[f''(x_k)]^{-1} f'(x_k) \\
x_{k+1} &= x_k - [f''(x_k)]^{-1} f'(x_k).
\end{aligned}$$

Let us immediately note the limitations related to the necessity of the Hessian's unbornness (for the method to exist), as well as its positive definiteness (for the convergence guarantee).

Convergence

Let's try to get an estimate of how quickly the classical Newton method converges. We will try to enter the necessary data and constants as needed in the conclusion (to illustrate the methodology of obtaining such estimates).

$$
\begin{aligned}
x_{k+1} - x^* = x_k - [f''(x_k)]^{-1} f'(x_k) - x^* = x_k - x^* - [f''(x_k)]^{-1} f'(x_k) = \\
= x_k - x^* - [f''(x_k)]^{-1} \int_0^1 f''(x^* + \tau(x_k - x^*))(x_k - x^*) d\tau = \\
= \left(1 - [f''(x_k)]^{-1} \int_0^1 f''(x^* + \tau(x_k - x^*)) d\tau\right)(x_k - x^*) = \\
= [f''(x_k)]^{-1} \left(f''(x_k) - \int_0^1 f''(x^* + \tau(x_k - x^*)) d\tau\right)(x_k - x^*) = \\
= [f''(x_k)]^{-1} \left(\int_0^1 (f''(x_k) - f''(x^* + \tau(x_k - x^*)) d\tau)\right)(x_k - x^*) = \\
= [f''(x_k)]^{-1} G_k (x_k - x^*)
\end{aligned}
$$

Used here is: $G_k = \int_0^1 (f''(x_k) - f''(x^* + \tau(x_k - x^*))d\tau)$. Let's try to estimate the size of G_k:

$$
\begin{aligned}
\|G_k\| = \left\| \int_0^1 (f''(x_k) - f''(x^* + \tau(x_k - x^*))d\tau) \right\| \leq \\
\leq \int_0^1 \|f''(x_k) - f''(x^* + \tau(x_k - x^*))\| \, d\tau \leq \qquad (\text{Hessian's Lipschitz continuity}) \\
\leq \int_0^1 M \|x_k - x^* - \tau(x_k - x^*)\| d\tau = \int_0^1 M \|x_k - x^*\| (1 - \tau) d\tau = \frac{r_k}{2} M,
\end{aligned}
$$

where $r_k = \|x_k - x^*\|$.

So, we have:

$$
r_{k+1} \leq \left\| [f''(x_k)]^{-1} \right\| \cdot \frac{r_k}{2} M \cdot r_k
$$

Quadratic convergence already smells. All that remains is to estimate the value of Hessian's reverse.

Because of Hessian's Lipschitz continuity and symmetry:

$$
\begin{aligned}
f''(x_k) - f''(x^*) \succeq -M r_k I_n \\
f''(x_k) \succeq f''(x^*) - M r_k I_n \\
f''(x_k) \succeq l I_n - M r_k I_n \\
f''(x_k) \succeq (l - M r_k) I_n
\end{aligned}
$$

So, (here we should already limit the necessity of being $f''(x_k) \succ 0$ for such estimations, i.e. $r_k < \frac{l}{M}$).

$$
\left\| [f''(x_k)]^{-1} \right\| \leq (l - M r_k)^{-1}
$$

$$
r_{k+1} \leq \frac{r_k^2 M}{2(l - M r_k)}
$$

The convergence condition $r_{k+1} < r_k$ imposes additional conditions on r_k : $\quad r_k < \frac{2l}{3M}$

Thus, we have an important result: Newton's method for the function with Lipschitz positive Hessian converges squarely near ($\|x_0 - x^*\| < \frac{2l}{3M}$) to the solution with **quadratic speed**.

Theorem

Let $f(x)$ be a strongly convex twice continuously differentiated function at $\mathbb{R}^n$, for the second derivative of which inequalities are executed: $lI_n \preceq f''(x) \preceq LI_n$. Then Newton's method with a constant step locally converges to solving the problem with super linear speed. If, in addition, Hessian is Lipschitz continious, then this method converges locally to x^* with a quadratic speed.

Examples

Let's look at some interesting features of Newton's method. Let's first apply it to the function

$$f(x) = x^4$$

Summary

It's nice:

- quadratic convergence near the solution x^*
- affinity invariance
- the parameters have little effect on the convergence rate

It's not nice:

- it is necessary to store the hessian on each iteration: $\mathcal{O}(n^2)$ memory
- it is necessary to solve linear systems: $\mathcal{O}(n^3)$ operations
- the Hessian can be degenerate at x^*
- the hessian may not be positively determined $\rightarrow$ direction $-(f''(x))^{-1}f'(x)$ may not be a descending direction

Possible directions

- Newton's damped method (adaptice stepsize)
- Quasi-Newton methods (we don't calculate the Hessian, we build its estimate - BFGS)
- Quadratic evaluation of the function by the first order oracle (superlinear convergence)
- The combination of the Newton method and the gradient descent (interesting direction)
- Higher order methods (most likely useless)

Code

Open in Colab

Quasi Newton methods

Intuition

For the classic task of unconditional optimization $f(x) \rightarrow \min\limits_{x \in \mathbb{R}^n}$ the general scheme of iteration method is written as:

$$x_{k+1} = x_k + \alpha_k s_k$$

In the Newton method, the s_k direction (Newton's direction) is set by the linear system solution at each step:

$$s_k = -B_k \nabla f(x_k), \quad B_k = f''^{-1}_{xx}(x_k)$$

i.e. at each iteration it is necessary to **compensate** hessian and gradient and **resolve** linear system.

Note here that if we take a single matrix of $B_k = I_n$ as B_k at each step, we will exactly get the gradient descent method.

The general scheme of quasi-Newton methods is based on the selection of the B_k matrix so that it tends in some sense at $k \to \infty$ to the true value of inverted Hessian in the local optimum $f_{xx}^{-1}(x_*)$. Let's consider several schemes using iterative updating of B_k matrix in the following way:

$$B_{k+1} = B_k + \Delta B_k$$

Then if we use Taylor's approximation for the first order gradient, we get it:

$$\nabla f(x_k) - \nabla f(x_{k+1}) \approx f_{xx}(x_{k+1})(x_k - x_{k+1}).$$

Now let's formulate our method as:

$$\Delta x_k = B_{k+1}\Delta y_k, \text{ where } \quad y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$$

in case you set the task of finding an update ΔB_k:

$$\Delta B_k \Delta y_k = \Delta x_k - B_k \Delta y_k$$

Broyden method

The simplest option is when the amendment ΔB_k has a rank equal to one. Then you can look for an amendment in the form

$$\Delta B_k = \mu_k q_k q_k^\top.$$

where μ_k is a scalar and q_k is a non-zero vector. Then mark the right side of the equation to find ΔB_k for Δz_k:

$$\Delta z_k = \Delta x_k - B_k \Delta y_k$$

We get it:

$$\mu_k q_k q_k^\top \Delta y_k = \Delta z_k$$

$$\left(\mu_k \cdot q_k^\top \Delta y_k\right) q_k = \Delta z_k$$

A possible solution is: $q_k = \Delta z_k$, $\mu_k = \left(q_k^\top \Delta y_k\right)^{-1}$.

Then an iterative amendment to Hessian's evaluation at each iteration:

$$\Delta B_k = \frac{(\Delta x_k - B_k \Delta y_k)(\Delta x_k - B_k \Delta y_k)^\top}{\langle \Delta x_k - B_k \Delta y_k, \Delta y_k \rangle}.$$

Davidon–Fletcher–Powell method

$$\Delta B_k = \mu_1 \Delta x_k (\Delta x_k)^\top + \mu_2 B_k \Delta y_k (B_k \Delta y_k)^\top.$$

$$\Delta B_k = \frac{(\Delta x_k)(\Delta x_k)^\top}{\langle \Delta x_k, \Delta y_k \rangle} - \frac{(B_k \Delta y_k)(B_k \Delta y_k)^\top}{\langle B_k \Delta y_k, \Delta y_k \rangle}.$$

Broyden–Fletcher–Goldfarb–Shanno method

$$\Delta B_k = QUQ^\top, \quad Q = [q_1, q_2], \quad q_1, q_2 \in \mathbb{R}^n, \quad U = \begin{pmatrix} a & c \\ c & b \end{pmatrix}.$$

$$\Delta B_k = \frac{(\Delta x_k)(\Delta x_k)^\top}{\langle \Delta x_k, \Delta y_k \rangle} - \frac{(B_k \Delta y_k)(B_k \Delta y_k)^\top}{\langle B_k \Delta y_k, \Delta y_k \rangle} + p_k p_k^\top.$$

Comparison of quasi Newton methods

Link

Adaptive metric methods

It is known, that antigradient $-\nabla f(x_0)$ is the direction of the steepest descent of the function $f(x)$ at point x_0. However, we can introduce another concept for choosing the best direction of function decreasing.

Given $f(x)$ and a point x_0. Define $B_\varepsilon(x_0) = \{x \in \mathbb{R}^n : d(x, x_0) = \varepsilon^2\}$ as the set of points with distance ε to x_0. Here we presume the existence of a distance function $d(x, x_0)$.

$$x^* = \arg \min_{x \in B_\varepsilon(x_0)} f(x)$$

Than, we can define another *steepest descent* direction in terms of minimizer of function on a sphere:

$$s = \lim_{\varepsilon \to 0} \frac{x^* - x_0}{\varepsilon}$$

Let us assume that the distance is defined locally by some metric A:

$$d(x, x_0) = (x - x_0)^\top A (x - x_0)$$

Let us also consider first order Taylor approximation of a function $f(x)$ near the point x_0:

$$f(x_0 + \delta x) \approx f(x_0) + \nabla f(x_0)^\top \delta x \tag{A1}$$

Now we can explicitly pose a problem of finding s, as it was stated above.

$$\min_{\delta x \in \mathbb{R}^n} f(x_0 + \delta x)$$
$$\text{s.t. } \delta x^\top A \delta x = \varepsilon^2$$

Using $(\mathrm{A1})$ it can be written as:

$$\min_{\delta x \in \mathbb{R}^n} \nabla f(x_0)^\top \delta x$$
$$\text{s.t. } \delta x^\top A \delta x = \varepsilon^2$$

Using Lagrange multipliers method, we can easily conclude, that the answer is:

$$\delta x = -\frac{2\varepsilon^2}{\nabla f(x_0)^\top A^{-1} \nabla f(x_0)} A^{-1} \nabla f$$

Which means, that new direction of steepest descent is nothing else, but $A^{-1} \nabla f(x_0)$.

Indeed, if the space is isotropic and $A = I$, we immediately have gradient descent formula, while Newton method uses local Hessian as a metric matrix.