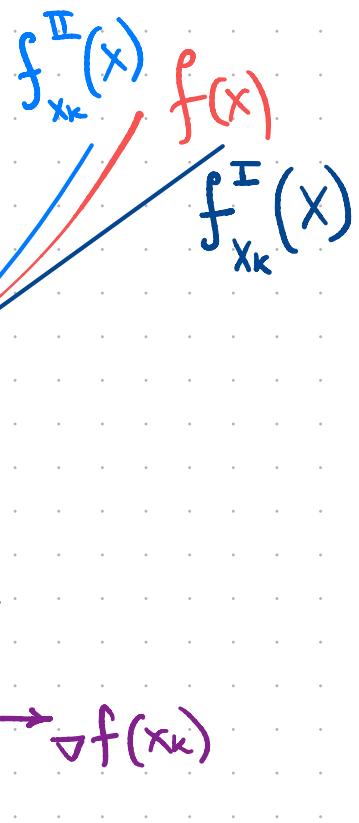


$$GD: \quad x_{k+1} = x_k - \lambda \cdot \nabla f(x_k)$$



вектор

$$f_{x_k}^I(x) = f(x_k) + \langle \nabla f(x_k), x - x_k \rangle$$

направление

Ряд
Тейлора

$$f_{x_k}^{II}(x) = f(x_k) + \langle \nabla f(x_k), x - x_k \rangle + \frac{1}{2} \langle x - x_k, \nabla^2 f(x_k)(x - x_k) \rangle$$

$$f(x_k) + \nabla f(x_k)^T (x - x_k) + \frac{1}{2} (x - x_k)^T \nabla^2 f(x_k) (x - x_k)$$

$$x_{k+1} = \arg \min_{x \in \mathbb{R}^n} \left[f(x_k) + \nabla f(x_k)^T (x - x_k) + \frac{1}{2} (x - x_k)^T I (x - x_k) \right]$$

$$\nabla_x \tilde{f}(x) = 0 \Rightarrow \nabla f(x_k) + I(x - x_k) = 0$$

$$\nabla f(x_k) + x_{k+1} - x_k = 0 \Rightarrow x_{k+1} = x_k - \nabla f(x_k)$$

$$f(x) = \frac{1}{2} x^T I x \quad \nabla^2 f = I$$

$$\Leftrightarrow \frac{1}{2} (x_1^2 + x_2^2 + \dots + x_n^2)$$

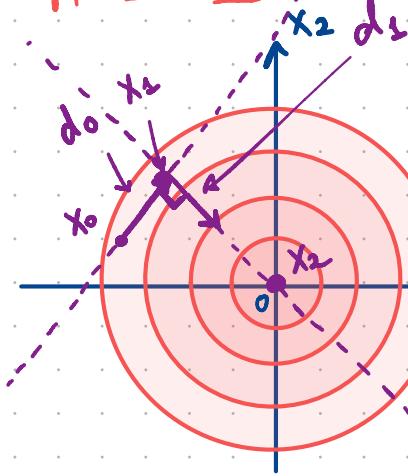
В квадратичной форме $f(x) = \tilde{f}(x)$

ИДЕАЛЬНЫЙ МИР

$$f(x) = \frac{1}{2}(x_1^2 + x_2^2)$$

$d_0 \perp d_1$, $b=0$, $c=0$

$$A = I$$



$$f(x) = \frac{1}{2}x^T Ax - b^T x + c$$

$$\nabla f = Ax - b$$

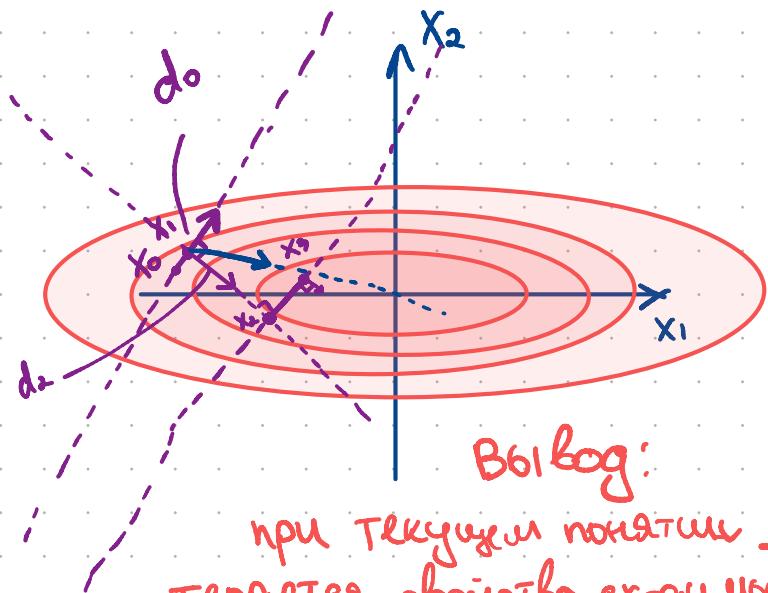
$$\nabla^2 f = A$$

Напр.
do выбор.
шаг?
 $d_1 \perp d_0$
шаги
наискор.
спуск.

$$A \in \mathbb{S}_{++}^n$$

РЕАЛЬНЫЙ МИР

$d_0 \perp d_1$



Выбор:

при текущем понятии \perp
теряется свойство сходимости
за 2 итерации.

Решение: меняем понятие \perp .

$$a \perp_A b : a^T A b = 0$$

$$\langle a, I b \rangle = a^T A b$$

$$\langle a, b \rangle_A = a^T A b$$

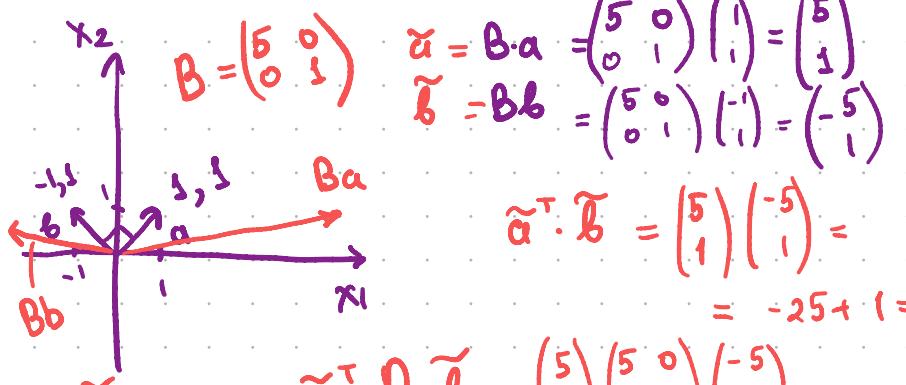
$$\langle a, b \rangle_A = 0$$

Оказывается, что для любых
2 \perp_A направлений с
выбором наилучшего шага
мы опять сойдемся за
2 итерации.

Для n-мерных задач:

n направлений (A-ортогональных друг другу)

n шагов алгоритма.



$$(\tilde{a} \cdot \tilde{b})_{B^{-2}} = 0$$

$$\tilde{a}^T \cdot B \tilde{b} = \begin{pmatrix} 5 \\ 1 \end{pmatrix} \begin{pmatrix} 5 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} -5 \\ 1 \end{pmatrix} =$$

$$= \begin{pmatrix} 5 \\ 1 \end{pmatrix} \begin{pmatrix} -25 \\ 1 \end{pmatrix} \neq 0$$

$$(B \tilde{a})^T B^{-2} \tilde{b} = \begin{pmatrix} 5 \\ 1 \end{pmatrix} \begin{pmatrix} \frac{1}{5} & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} -5 \\ 1 \end{pmatrix} =$$

$$\tilde{a}^T B^{-2} \tilde{b} \left(\begin{pmatrix} \frac{1}{5} & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 5 \\ 0 \end{pmatrix} \right) \left(\begin{pmatrix} \frac{1}{5} & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} -5 \\ 1 \end{pmatrix} \right) = 0$$

$$B^{-2} = \begin{pmatrix} \frac{1}{5} & 0 \\ 0 & 1 \end{pmatrix}$$

$$B \cdot B^{-1} = \begin{pmatrix} 5 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{5} & 0 \\ 0 & 1 \end{pmatrix} =$$

$$= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$a \rightarrow \tilde{a} = B a$$

$$b \rightarrow \tilde{b} = B b$$

np.linalg.eig
on general use

$$\sigma(A) = \|A\|_1 \cdot \|A\|$$

$$= \frac{\sigma_{\max}(A)}{\sigma_{\min}(A)}$$

$\|A\|_1 = \|A\|_2$

$$\|A\|_2 = \sigma_{\max}(A)$$

unitary

$$A = U \sum V^*$$

unitary

$$U \Sigma V = \text{np.linalg.svd}(A)$$

Singular Value Decomposition

$$\begin{pmatrix} \sigma_{\max} & & \\ & \ddots & \\ & & \sigma_{\min} \end{pmatrix}$$

$$f(x) = \frac{1}{2} x^T I x = 10$$

$$f(a) = \frac{1}{2} a^T I a = f_a$$

$$f(\tilde{a}) = \frac{1}{2} \tilde{a}^T B^T I B a = \tilde{f}_a$$

$$\frac{1}{2} \tilde{a}^T \underbrace{B^T B}_{B^2} a = \tilde{f}_a$$

$$f = \frac{1}{2} x^T \tilde{A} x$$

$$\tilde{A} = (B^T B)^{-1} = B^{-2}$$

U - unitary matrix

$$\|Ua\| = \|a\|$$

$$U^* \leftarrow U^T$$

$$U^* U = U U^T = I$$

Conjugate gradients

Introduction

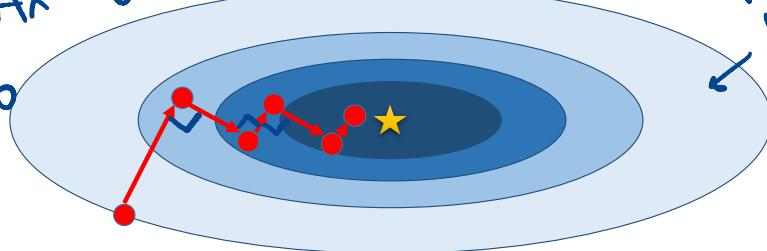
WHO WILL WIN?

$A \in \mathbb{S}_{++}^n$

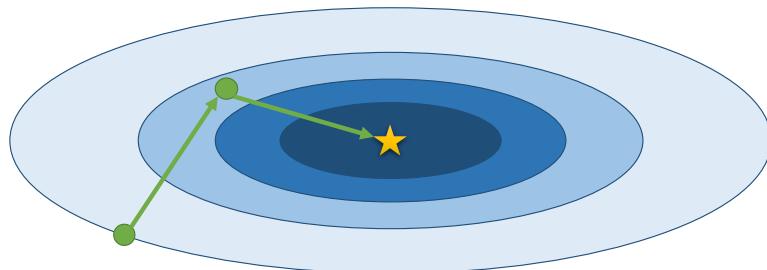
$$f(x) = \frac{1}{2} x^\top A x - b^\top x + c \rightarrow \min_{x \in \mathbb{R}^n}$$

$$\nabla f = Ax - b = 0$$

$$Ax^* = b$$



GD



CG

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:

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

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:

$$\nabla f = Ax - b$$

T.k. введял непарою вектору
хр нравчунненли, то
 $d_0 = -\nabla f(x)$

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:

$$g(\alpha) = f(x_0 + \alpha d_0)$$

$$= \frac{1}{2} (x_0 + \alpha d_0)^\top A (x_0 + \alpha d_0) - b^\top (x_0 + \alpha d_0) + c$$

$$= \frac{1}{2} \alpha^2 d_0^\top A d_0 + d_0^\top (Ax_0 - b) \alpha + (\frac{1}{2} x_0^\top A x_0 + x_0^\top d_0 + c)$$

это квадратичное до
будут все же до

$$d_0 = \frac{d_0^\top d_0}{d_0^\top A d_0}$$

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:

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

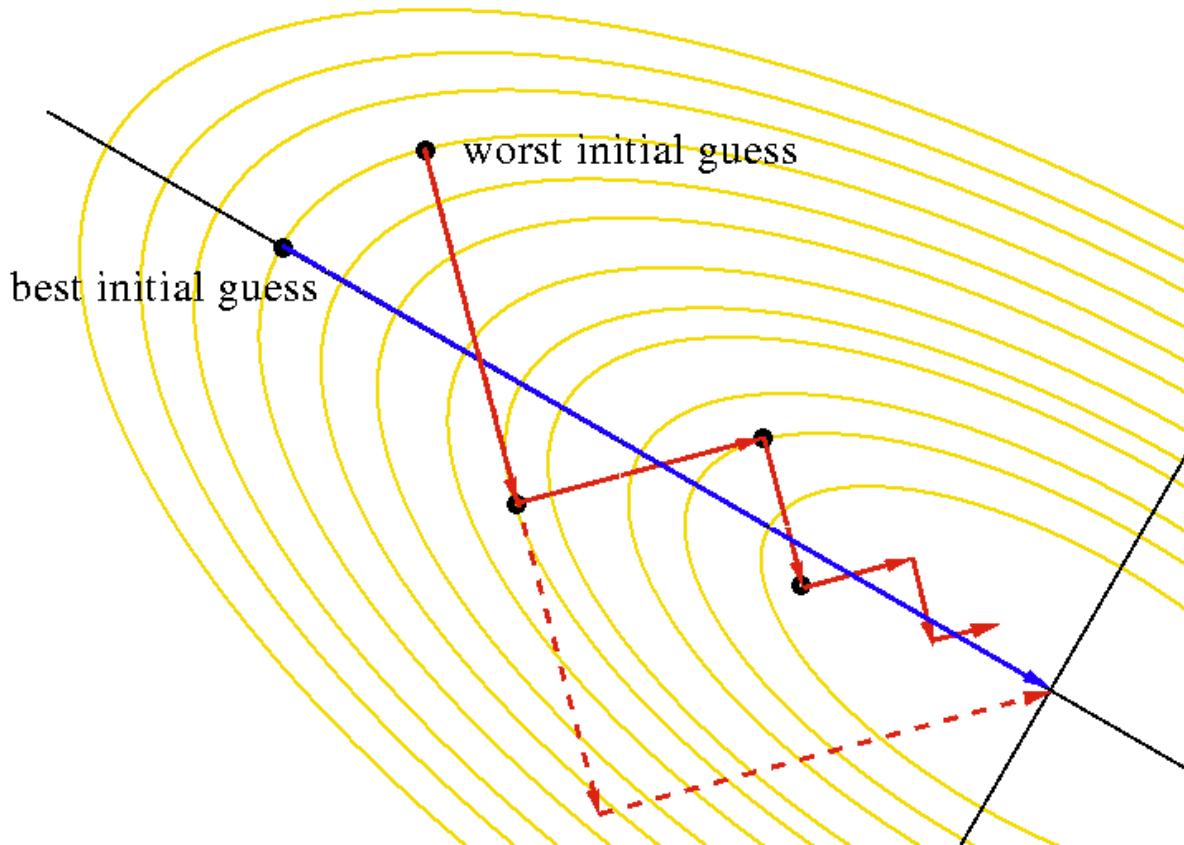
$$\alpha_k = -\frac{d_k^\top (Ax_k - b)}{d_k^\top A d_k} = \frac{d_k^\top d_k}{d_k^\top A d_k}.$$

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

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

если $d_1 = -\nabla f(x_1)$
 $d_1 \perp d_0$
 ~~$d_1 \perp d_0$~~

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:

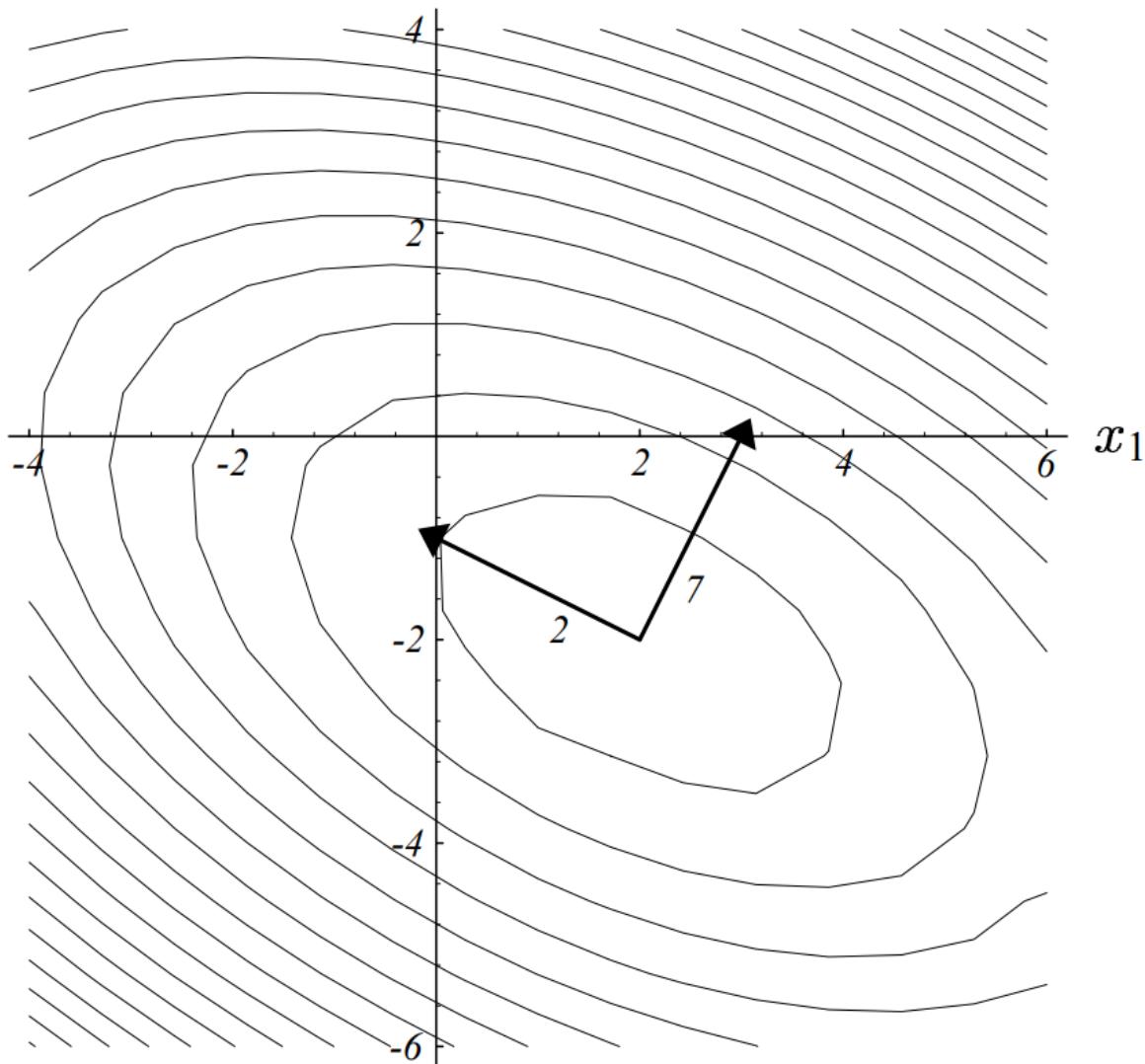


In order to avoid this, we introduce the concept of A -conjugated vectors: let's say that two vectors x, y A -are conjugated relative to each other if they are executed:

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

A -ортогональность
 A -сопряженность

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 $d_1 \perp_A d_0$:

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

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

$$d_{k+1} = -\nabla f(x_{k+1}) + \beta_k d_k$$

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

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

Thus, we formulate an algorithm:

1. Let $k = 0$ and $x_0 = x_0$, count $d_0 = -\nabla f(x_0)$. **Инициал.** + **некий в напр. $-\nabla f(x_0)$**
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

$$d_k = \frac{d_k^\top (-\nabla f(x_k))}{d_k^\top A d_k} \quad \alpha_k = -\frac{d_k^\top (A x_k - b)}{d_k^\top A d_k} = \underset{\alpha}{\operatorname{argmin}} f(x_k + \alpha d_k)$$

3. We're doing an algorithm step:

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

Наибольш. тока в направлении $-\nabla f(x_k)$

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

Делаем шаг.

Справа новое направление $d_{k+1} \perp_A d_k$

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

$$Ax = b \\ x = A^{-1}b \sim O(n^2)$$

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

Также сокращено за
направлений S_{++}

5. Repeat steps 2-4 until n directions are built, where n is the dimension of space (dimension of x). $O(n^2) \sim Ad_k - 3A$ ↓ в результате $O(n^3)$ уменьшена сложность алгоритма

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:

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

$$x_{k+1} - x_k = c d_k,$$

$$x_{k+1} - x_k = d_k d_k \quad |A|$$

$$\begin{aligned} Ax_{k+1} - Ax_k &= d_k (A d_k) \\ \nabla f(x_{k+1}) - \nabla f(x_k) &= d_k A d_k \\ Ad_k &= \frac{\nabla f(x_{k+1}) - \nabla f(x_k)}{d_k} \end{aligned}$$

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:

d_k имеет вид не в форме, а оно есть A и d_k ненулев.

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

This method is called the Polack - Ribier method.

Examples

Example 1

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

Solution:

We'll show, that if $\sum_{i=1}^k \alpha_i d_i = 0$, than all coefficients should be equal to zero:

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

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

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

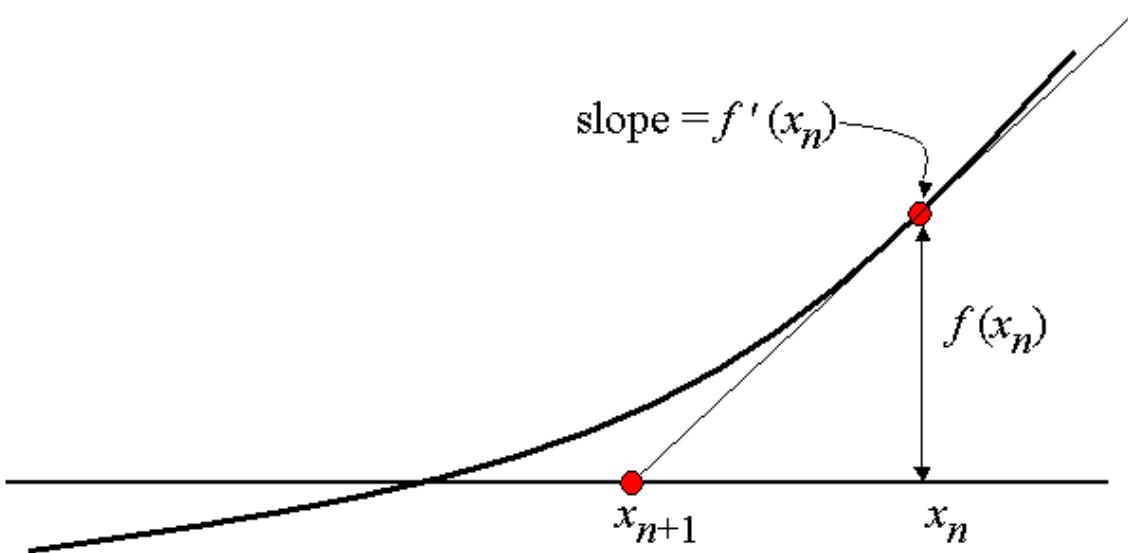
We get an approximate equation:

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

We can assume that the solution to equation $\Delta x = -\frac{\varphi(x)}{\varphi'(x)}$ 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 $\varphi(x) = f'(x)$, $\varphi'(x) = f''(x)$. Thus, we get the Newton optimization method in its classic form:

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

With the only clarification that in the multidimensional case:

$$x \in \mathbb{R}^n, f'(x) = \nabla f(x) \in \mathbb{R}^n, f''(x) = \nabla^2 f(x) \in \mathbb{R}^{n \times n}.$$

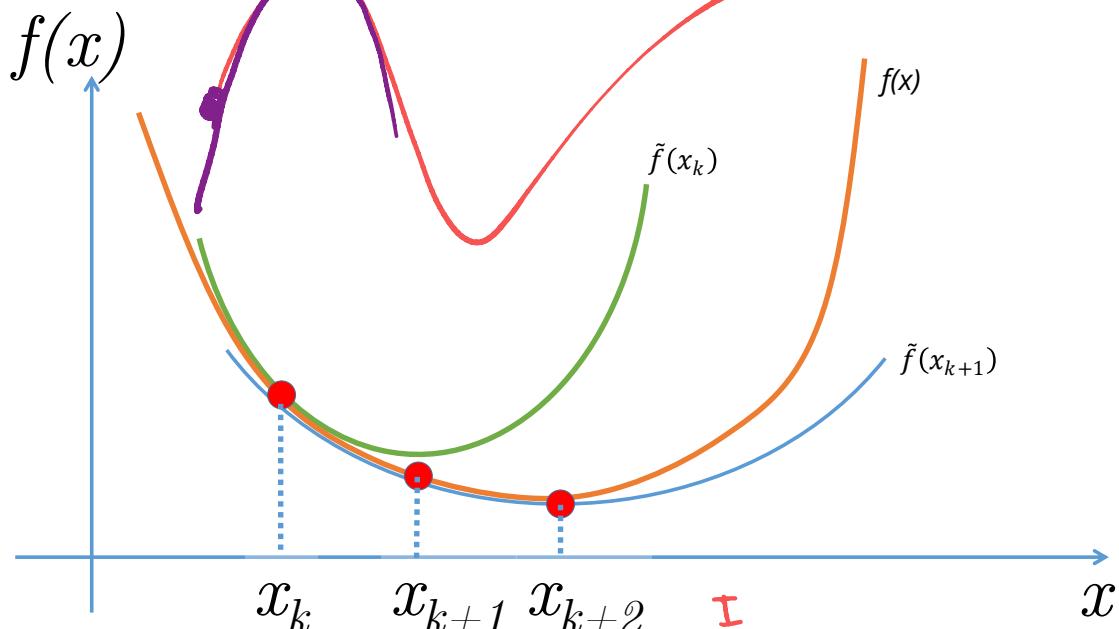
Second order Taylor approximation of the function

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.

$$\nabla \tilde{f}(x_{k+1}) = 0.$$



$$\nabla \tilde{f}(x_{k+1}) = f'(x_k) + f''(x_k)(x_{k+1} - x_k) = 0$$

$$f''(x_k)(x_{k+1} - x_k) = -f'(x_k)$$

$$[f''(x_k)]^{-1} f''(x_k)(x_{k+1} - x_k) = -[f''(x_k)]^{-1} f'(x_k)$$

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

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

$$x_{k+1} = x_k - [I]^{-1} \cdot \nabla f(x_k)$$

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 :

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

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

So, we have:

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

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

Because of Hessian's Lipschitz continuity and symmetry:

$$\begin{aligned}
f''(x_k) - f''(x^*) &\succeq -Mr_k I_n \\
f''(x_k) &\succeq f''(x^*) - Mr_k I_n \\
f''(x_k) &\succeq lI_n - Mr_k I_n \\
f''(x_k) &\succeq (l - Mr_k) I_n
\end{aligned}$$

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

$$\|[f''(x_k)]^{-1}\| \leq (l - Mr_k)^{-1}$$

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

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

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 continuous, then this method converges locally to x^* with a quadratic speed.

Несколько идей о нелинейной оптимизации

Examples

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



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

$$\nabla^2 f(x_k) \cdot \underbrace{(x_{k+1} - x_k)}_{d_{k+1} \in \mathbb{R}^n} = -\nabla f(x_k)$$

ремесл. сложность $O(n^3)$
 $d_{k+1} = \dots$

Summary

It's nice:

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

+ Оценка быстроты (на каждой итерации проходит уравнение)

+ Если начальное значение засечки x_0 входит в правильных формате \Rightarrow сходимость становится глобальной

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

- По умолчанию сходимость

нужен локальный характер
- считать и хранить $\nabla^2 f$ или $(\nabla^2 f)^{-1}$ гораздо
更容易

$$x_{k+1} = x_k - d_k [\nabla^2 f(x_k)]^{-1} \nabla f(x_k)$$

d_k называется шагом итерации

Possible directions

- Newton's damped method (adaptive 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 $f(x) \rightarrow \min_{x \in \mathbb{R}^n}$ the general scheme of iteration

method is written as:

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

$$16Gb = n^2 \cdot 32 \text{ ГБ}$$
$$n^2 = \frac{n^2 \cdot 4 \text{ байт}}{4 \cdot 10^{10} \cdot 1024 \cdot 1024 \cdot 1024^3}$$
$$n^2 = 4 \cdot 10^9$$
$$n \approx 10^4 - 10^5$$

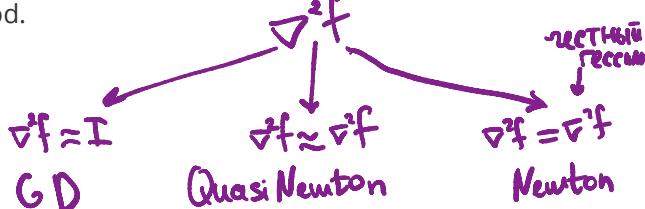
6 DL $n = 10^{10}$
 10^6

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

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

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 $k \rightarrow \infty$ 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:

$$X_{k+1} = X_k - D^{-1} \nabla f(x_k) =$$

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

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:

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

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

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

$$\begin{aligned} &= X_k - \begin{pmatrix} \frac{\partial f}{\partial x_1} & \dots & \frac{\partial f}{\partial x_n} \end{pmatrix} \\ &= X_k - \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \dots & \frac{\partial^2 f}{\partial x_n^2} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_1 \partial x_n} & \dots & \frac{\partial^2 f}{\partial x_n \partial x_1} \end{pmatrix} \\ &\text{Any Cb diag} \begin{pmatrix} \nabla^2 f \\ \nabla f \end{pmatrix} \end{aligned}$$

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

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

$$B \approx (\nabla^2 f)^{-1}$$

$$B_0 = I$$

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 :

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

$$\begin{aligned} B_{k+1} &= B_k + \Delta B_k \\ B_k &\approx (\nabla^2 f)^{-1} \end{aligned}$$

We get it:

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

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

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

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

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

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

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:

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

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 :

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

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

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

Using (A1) it can be written as:

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

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.