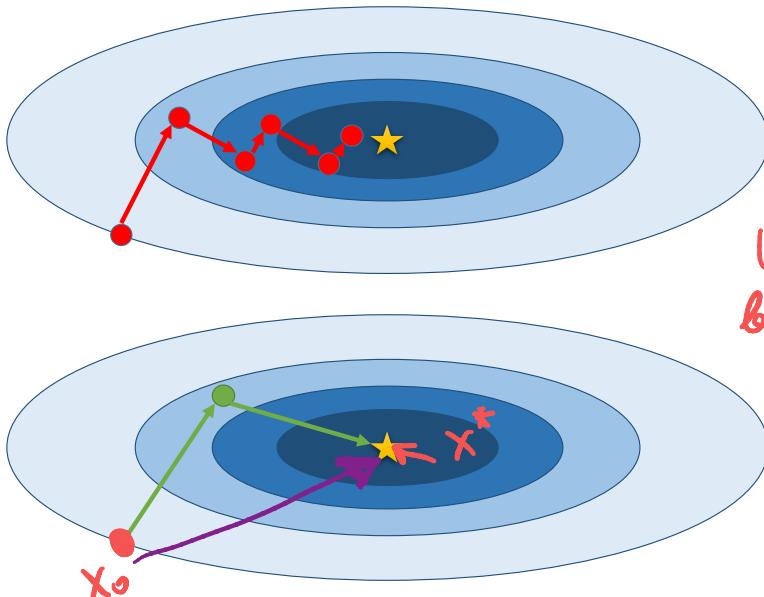


# Conjugate gradients

## Introduction

# WHO WILL WIN?



GD

Угээ: нэмээнд н  
бекороб таакх, то!  
 $x^* - x_0 = \sum_{i=1}^n d_i d_i$

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:

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

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

$$\begin{aligned} A &\in \mathbb{S}^n_+ \\ A &= \frac{A+A^\top}{2} \end{aligned}$$

## 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:

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

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:

$$g'(\alpha_k) = (d_k^T A d_k) \alpha_k + d_k^T (A x_k - b) = 0$$

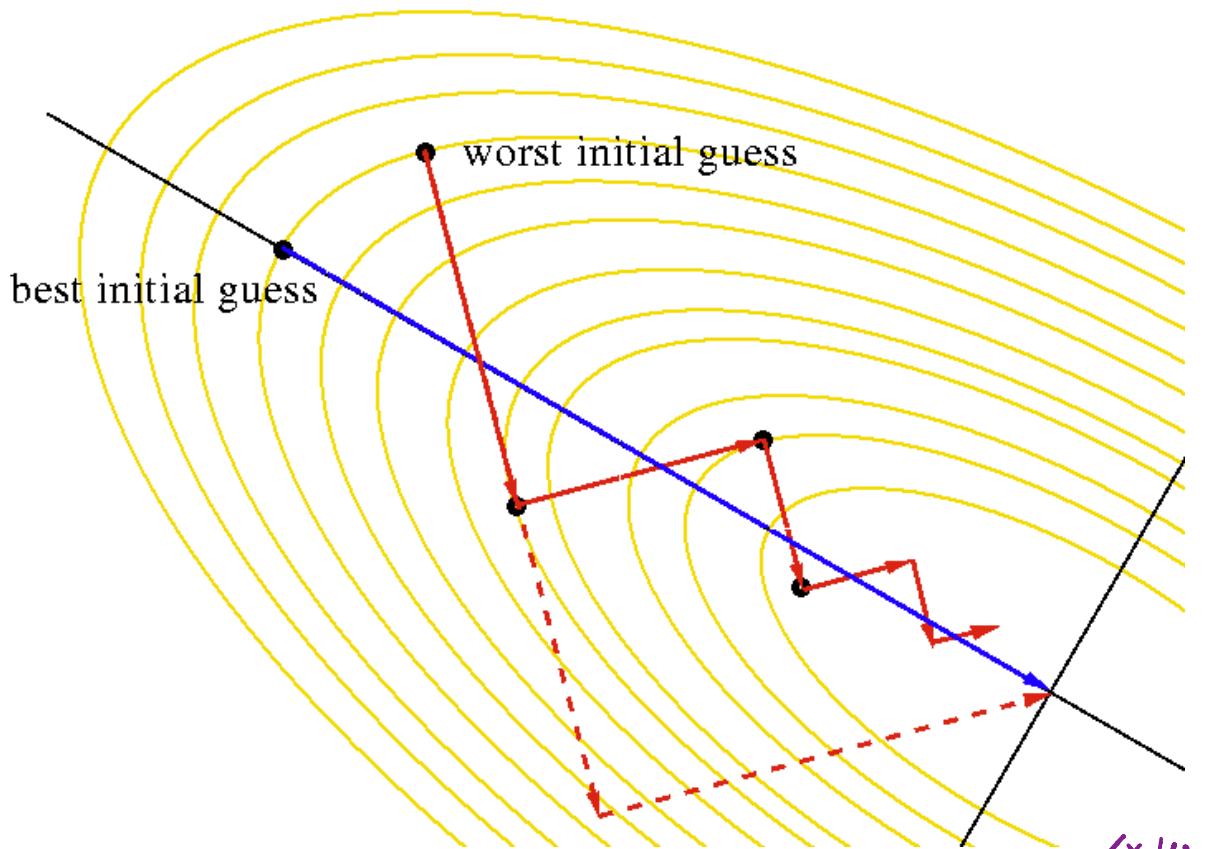
$$\alpha_k = -\frac{d_k^T (A x_k - b)}{d_k^T A d_k} = \frac{d_k^T d_k}{d_k^T A d_k}.$$

$d_0, d_1, X_1$

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:



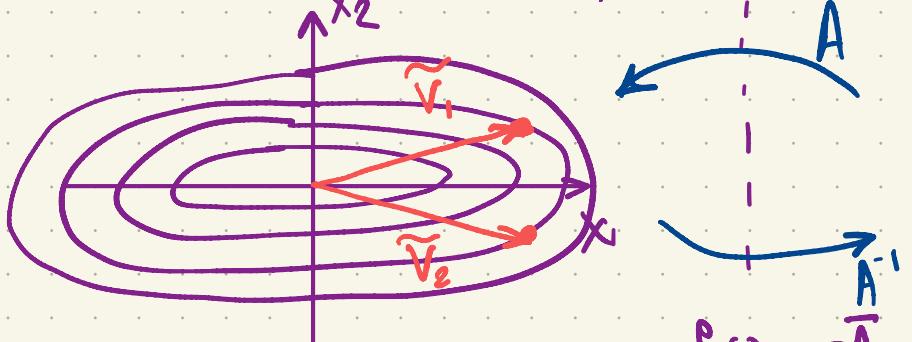
In order to avoid this, we introduce the concept of  $A$ -conjugated vectors: let's say that two vectors  $x, y$  are  $A$ -conjugated relative to each other if they are executed:

$$x^T A y = 0$$

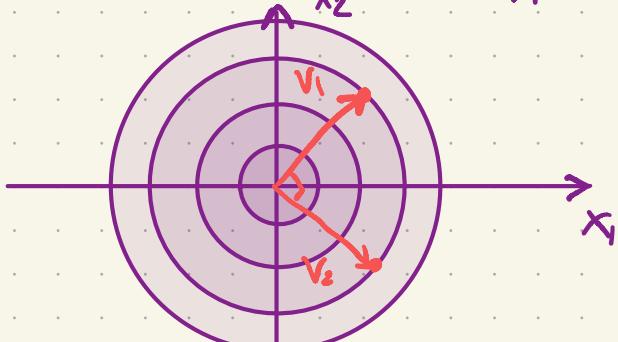
если  $A = I$ , то  $x^T y$   
 $\langle x, y \rangle_A = x^T A y$

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.

Реальний мир  $\bar{A} \neq I$



Ідеальний світ  $\bar{A} = I$



$$f(x) = x^T \bar{A} x$$

$$\alpha(A) = \|A\| \cdot \|\bar{A}^{-1}\| = \frac{\sigma_{\max}(A)}{\sigma_{\min}(A)}$$

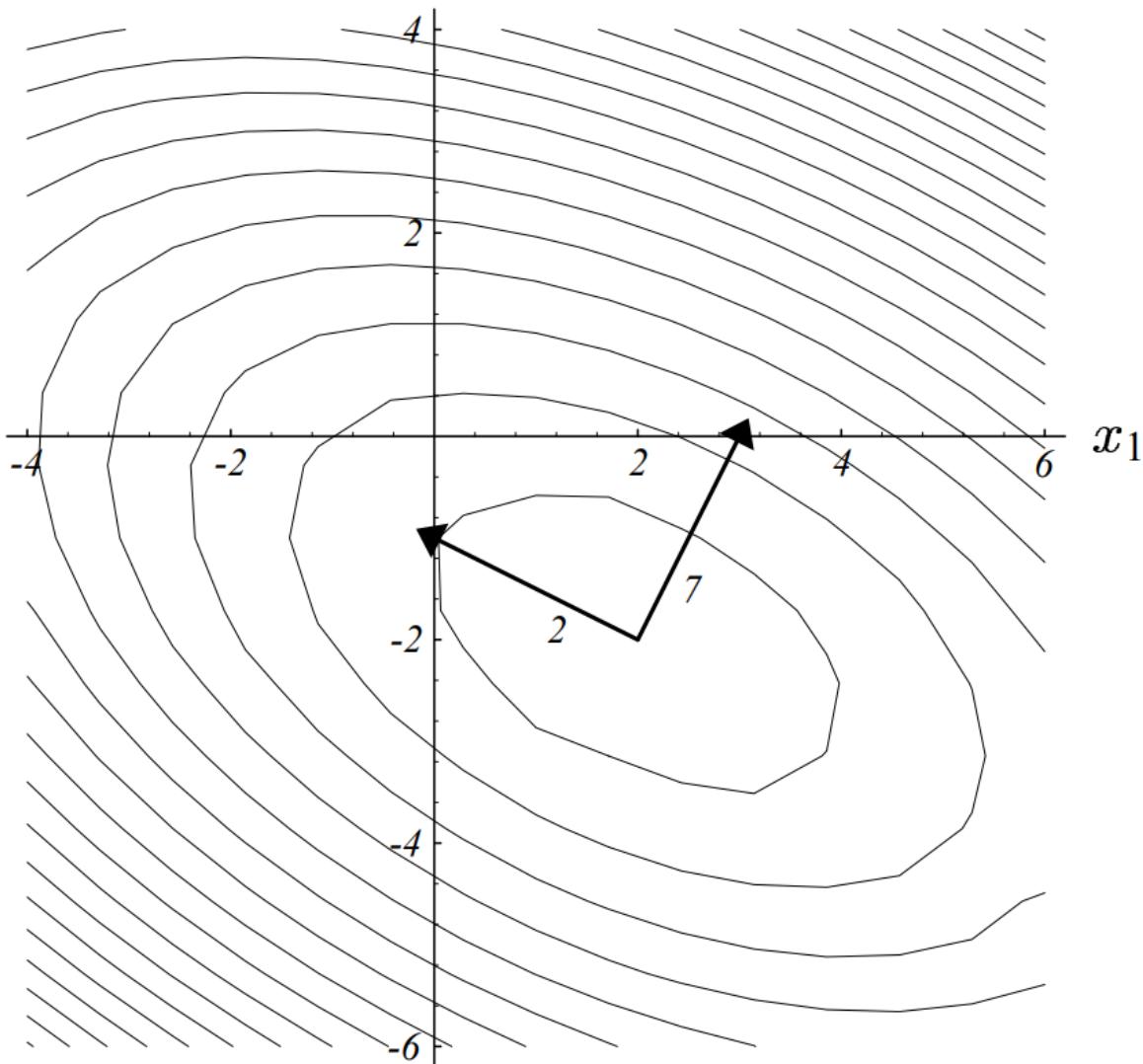
$$\alpha(A) \gg 1$$

$$\tilde{V}_1^T \tilde{V}_0 \neq 0$$

$$\tilde{V}_1^T A \tilde{V}_2 = 0$$

$$\alpha(\bar{A}) = 1$$

$$\tilde{V}_1^T \tilde{V}_2 = 0$$



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  $\beta_0$  is selected in a way that  $d_1 \perp_A d_0$ :

$$x_2 = x_1 + d_1$$

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

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

$$, d_1 = \frac{d_1^\top d_1}{d_1^\top A d_1}$$

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

Thus, we formulate an algorithm:

$$d_2 = -\nabla f(x_2) + \beta_1 d_1$$

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

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

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  $\beta_k$  is calculated by the formula:

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

*н векторов ортогональных  
как берется*

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

$$x^* - x_0 = \sum_{i=1}^n d_i d_i$$

## Method of conjugate gradients for non-quadratic function:

$$x^* = x_0 + \sum_{i=1}^n d_i d_i$$

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:

1) Проверяя так построенных ненулевых  
векторов  $b$  к  $x^*$ ?

2) Проверяя они все ненулевые  $A$ -ортогональны?

For two iterations, it is fair:

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

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  $\beta_k$ , 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  $d_1, \dots, d_k$  -  $A$  - 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  $\sum_{i=1}^k \alpha_i d_i = 0$ , 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 ( $x^* - x = \Delta x$ ):

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

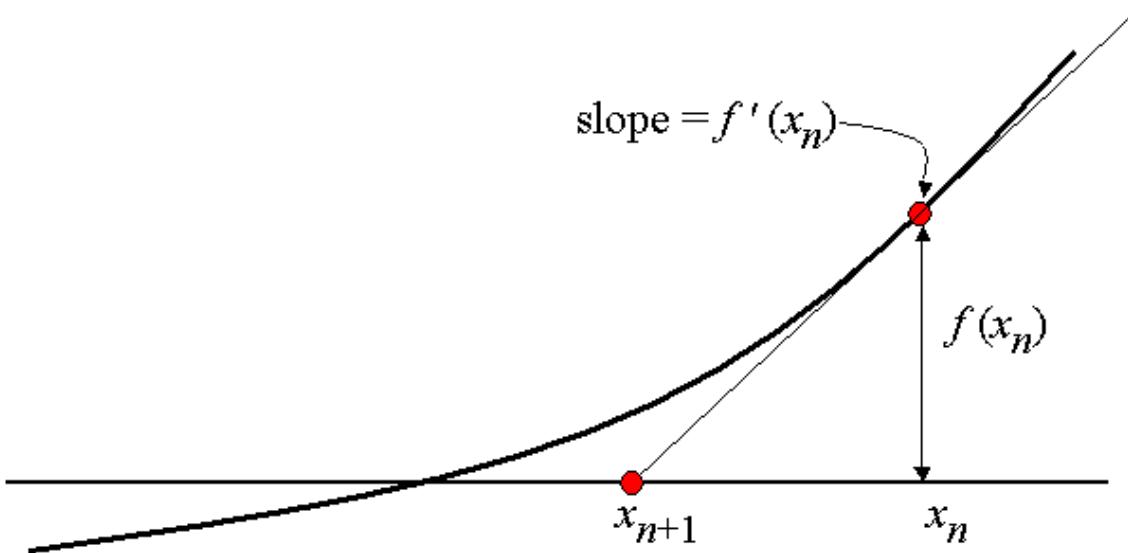
We get an approximate equation:

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

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:

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

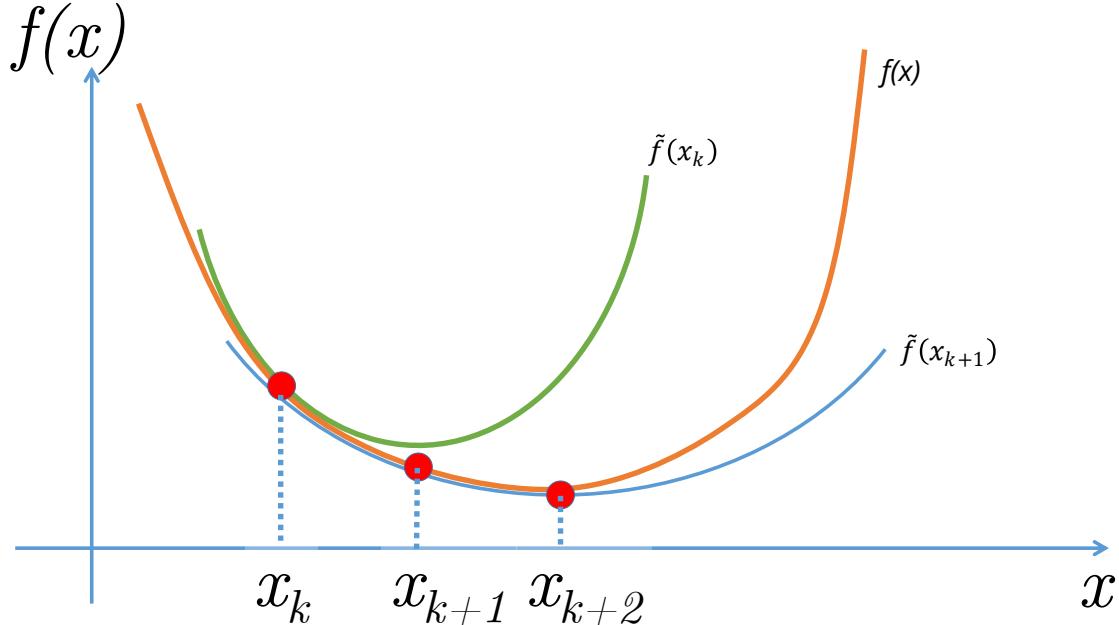
## Second order Taylor approximation of the function

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.

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



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

Let us immediately note the limitations related to the necessity of the Hessian's unorness (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  $G_k$ :

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

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.  $r_k < \frac{l}{M}$ ).

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

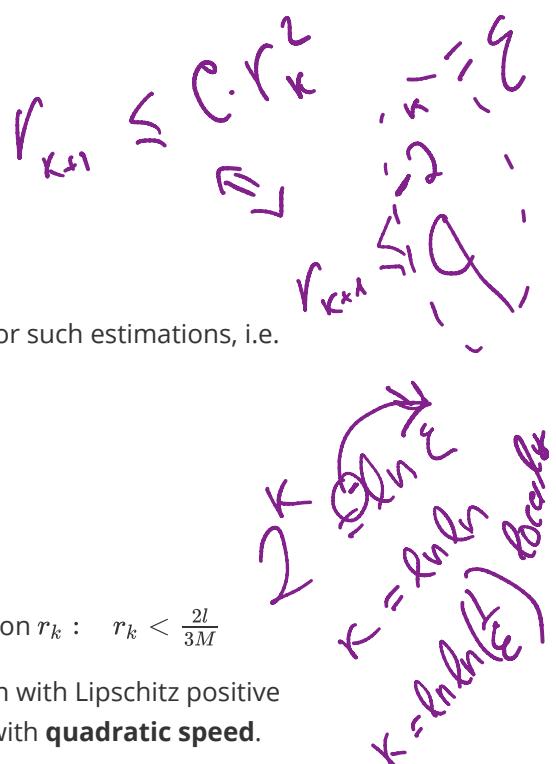
$$r_{k+1} \leq \frac{r_k^2 M}{2(l - Mr_k)}$$

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 square 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.



# 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 (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](#)

$$x_{k+1} = x_k + \alpha_k M^{-1} \nabla f(x_k)$$

## 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:

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

$$M = I - GD$$

$$M = \nabla^2 f - \text{Newton}$$

$$M \approx \nabla^2 f \quad \text{Quasi Newton}$$

$$M = \text{diag}(m)$$

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:

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

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:

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

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

in case you set the task of finding an update  $\Delta B_k$ :

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

## Broyden method

$$SR - 1 \quad \text{Пусть } B_0 = I$$

The simplest option is when the amendment  $\Delta B_k$  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  $\mu_k$  is a scalar and  $q_k$  is a non-zero vector. Then mark the right side of the equation to find  $\Delta B_k$  for  $\Delta z_k$ :

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

We get it:

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

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

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

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

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

$$\Delta y = \nabla f_k - \nabla f_{k+1}$$

$$\Delta x_k = x_{k+1} - x_k$$

# 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  $\varepsilon$  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.

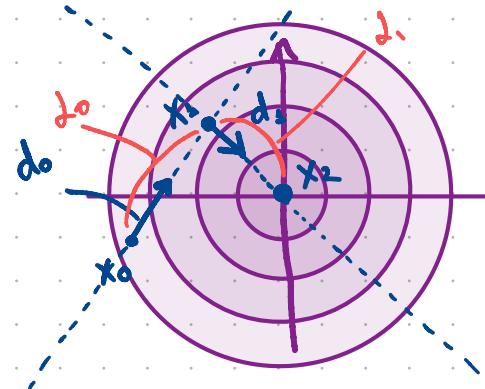
$$f(x) = \frac{1}{2} x^T A x - b^T x + c \rightarrow \min_{x \in \mathbb{R}^n} \quad A \in \mathbb{S}_{++}$$

$$\nabla f = Ax - b$$

$$\nabla^2 f = A$$

ИДЕАЛЬНЫЙ МИР

$$A = I$$



$$x_1 = x_0 + d_0 \cdot \alpha$$

$$x^* = x_0 + \sum_{i=0}^{n-1} d_i \cdot d_i \quad | \quad d_k^T$$

$$d_k^T x^* = d_k^T x_0 + \sum_{i=0}^{n-1} d_i \cdot d_k^T \cdot d_i$$

$$d_k^T (x^* - x_0) = d_k^T d_k$$

$$d_k = \frac{d_k^T (x^* - x_0)}{d_k^T d_k}$$

Вспомнили ответ для шага  
в каскад. спуске

$$d_k = \frac{d_k^T \cdot d_k}{d_k^T \cdot Ad_k} = \frac{d_k^T d_k}{d_k^T d_k} = 1$$

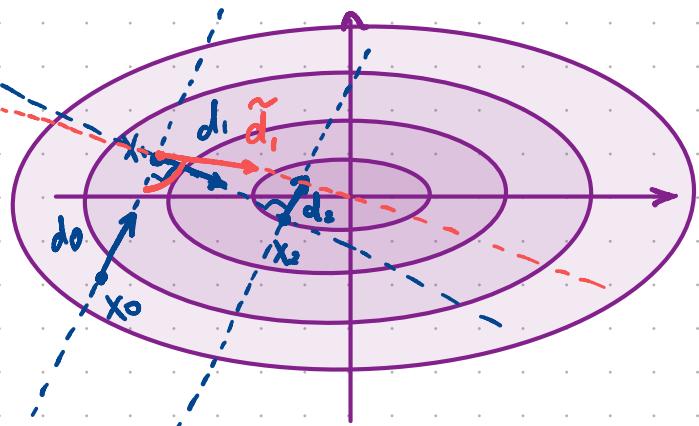
$$d_{k+1} =$$

есть набор  
 $u_1, \dots, u_n$

РЕАЛЬНЫЙ МИР

$$A \neq I$$

$d_0$  сим.  
 $d_1 \perp d_0$   
шаги по  
правилу  
каскад.  
спуска



Проблема: потеряли об-бо ex-ти  
за n шагов

Решение: изменить шаги ортогональн.

$$d_1 \perp d_0 \Rightarrow d_1^T d_0 = 0$$

$$\tilde{d}_1 \perp d_0 \Rightarrow \tilde{d}_1^T A d_0 = 0 \quad | \quad \tilde{d}_1 \cdot d_2 \neq 0$$

ОБНОВЛЕНИЕ  
ОБОЗНАЧЕНИЯ

$$x_k = x_0 + \sum_{j=0}^{k-1} d_j \cdot c_j$$

Что мы хотим если  $d_j$  - А ортогональны,  
+  $d_j$  подбираются каск. спуском  
(линейн. поиск),  
то мы сойдем за n шагов.

$$r_k = -\nabla f(x_k) = b - Ax_k - \text{residual}$$

$$e_k = x_k - x^* - \text{error}$$

① Нетог строим так:

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

Найдем  $d_k$ :  $f(x_{k+1}) \rightarrow \min$

$$\frac{\partial f(x_{k+1})}{\partial d} = 0 \rightarrow \frac{\partial f(x_{k+1})}{\partial x} \cdot d_k = 0$$

$$-r_{k+1}^T \cdot d_k = 0$$

$$r_{k+1}^T A d_k = 0$$

$$e_{k+1}^T A d_k = 0 \quad e_k = x_k - x^*$$

$$(x_{k+1} - x^*)^T A dx$$

$$(x_k + d_k d_k - x^*)^T A d_k = 0$$

$$(e_k + d_k d_k)^T A d_k = 0 \Rightarrow d_k = \frac{-e_k^T A d_k}{d_k^T A d_k} = \frac{-d_k^T A e_k}{d_k^T A d_k} = \frac{d_k^T r_k}{d_k^T A d_k}$$

② Докажем, что при таком выборе  $d_k$  и  $d_0, \dots, d_n$ ,

$$e_0 = \sum_{i=0}^{n-1} \delta_i \cdot d_i$$

коз. разложение  
по выбору  
фиксированных  
коэф.

$$d_k^T A e_0 = \sum_{i=0}^{n-1} \delta_i \cdot d_k^T A d_i$$

$$d_k^T A e_0 = \delta_k d_k^T A d_k$$

$$\delta_k = \frac{d_k^T A e_0}{d_k^T A d_k} = \frac{d_k^T A \left( e_0 + \sum_{i=0}^{k-1} d_i d_i \right)}{d_k^T A d_k} = -\frac{d_k^T r_k}{d_k^T A d_k} = -d_k$$

Мы сходимся  
за n шагов.

$$x_0 - x^* = \sum_{i=0}^{n-1} \delta_i d_i - d_i$$

$$x^* = x_0 - \sum_{i=0}^{n-1} \delta_i d_i$$

Вопрос о Vn : дублируем  $v_1, \dots, v_n = d_1, \dots, d_n$

$$v_j^T A v_i = \lambda_i v_i^T v_i$$

$$v_i^T A v_i = \lambda_i v_i^T v_i$$

$$\lambda_i = \frac{v_i^T A v_i}{v_i^T v_i}$$

$$A = U \Lambda U^+$$

$$\exists v_1, \dots, v_n$$

$$v_i^T v_i = 0$$

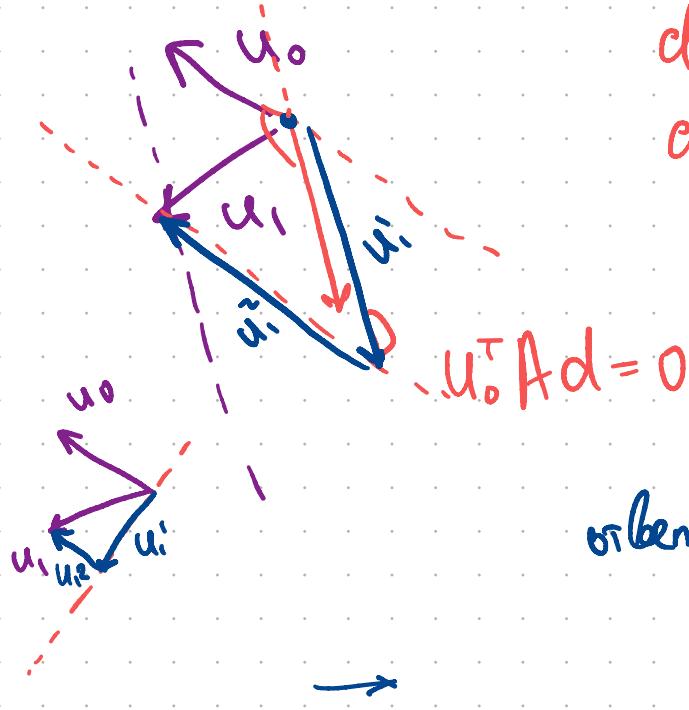
↑ Метод сопоставимых  
направлений.

Процедура построения  $d_0 \dots d_{n-1}$  через ортогон. Граммис-Шмидт мср.

$u_0, \dots, u_{n-1}$  —  $n$  ННЗ векторов

$$d_0 = u_0$$

$$d_1 = u_1 - P_{d_0}(u_1) = u_1'$$



$$d_k = u_k + \sum_{i=0}^{k-1} \beta_{ik} d_i$$

Как найти  $\beta_{ik}$ ?

$$d_k \perp_A d_{k-1} \Rightarrow$$

$$\beta_{ik} = -\frac{u_k^T A d_i}{d_i^T A d_i}$$

проблема:

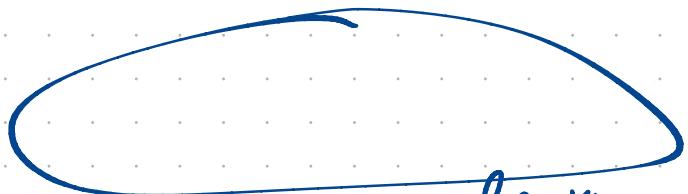
Для подсчёта  $d_k$  надо хранить ВСЕ векторы  $d_0 \dots d_{k-1}$ .

Многое сопряжено с направлением.



Если брать

$$u_0 \dots u_{n-1} = (r_0, \dots, r_{n-1}) \leftarrow \text{небольши}$$



ОКАЗЫВАЕТСЯ, что для подсчёта  $d_k$  нужно знать зкоть  $d_{k-1}$

$$r_k = -\nabla f(x_k) = b - Ax_k \quad (R)$$

$$e_k = x_k - x^* \quad (E)$$

Метог сопр. напр.  $x_i = x_0 + \sum_{j=0}^{i-1} d_j c_j$

$d_0, \dots, d_{n-1}$  - все ненулевые A-сопреженны

напр  $\uparrow$  получаются из  $u_0, \dots, u_{n-1}$

путь Gram-Schmidt.

$$d_j = \frac{c_j^T r_j}{c_j^T A c_j}$$

↑ из наиск. сущеска  
всех  $c_j$

$$d_i = u_i + \sum_{j=0}^{i-1} \beta_{ij} d_j \quad (GS)$$

из  $u_i$  берутся проекции  
 $d_j$  на  $u_i$

$$\beta_{ij} = -\frac{u_i^T A d_j}{d_j^T A d_j} \quad (B) \quad \forall i \in [0, n-1]$$

Пример:  $u_0, \dots, u_{n-1}$

$$d_0 = u_0$$

$$d_1 = u_1 - \frac{u_1^T A d_0}{d_0^T A d_0} \cdot d_0$$

$$d_2 = u_2 - \frac{u_2^T A d_0}{d_0^T A d_0} d_0 - \frac{u_2^T A d_1}{d_1^T A d_1} d_1$$

⋮

$$d_{n-1} = u_{n-1} - \dots - \dots - \dots$$

Услб: нормализовать вектора

$$u_i = r_i$$

↑  
знач не зает  
этой проблеме

знач не зает  
этой проблеме

PROBLEMES:

где  $d_{n-1}$  нужно

найти все

и синтез

гориз

Fun fact:

$$l_i = l_0 + \sum_{j=0}^{i-1} d_j d_j = x_i = x_0 + \sum_{j=0}^{i-1} d_j d_j$$

$$= \sum_{j=0}^{n-1} -d_j d_j + \sum_{j=0}^{i-1} d_j d_j =$$

$$= \sum_{j=i}^{n-1} -d_j d_j . \quad (ER)$$

$$l_i = x_i - x^*$$

$$l_0 = x_0 - x^* =$$

$$= x_0 - (x_0 + \sum_{j=0}^{n-1} d_j d_j)$$

Teneps (ER запускано для фикс. K)  $(-d_i^T A)$

I

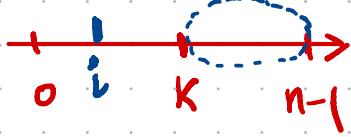
$$l_k = \sum_{j=k}^{n-1} -d_j d_j$$

A-ортогональ

$$-d_i^T A \cdot e_k = \sum_{j=k}^{n-1} d_j d_i^T A d_j$$

$$d_i^T r_k = 0$$

если  $i < k$



Также обозначим:  $r_k$  неподвижна для  $i < k$  на первом

предыдущем

$$d_i^T r_k = 0$$

,  $i < k$ .

$d_i$

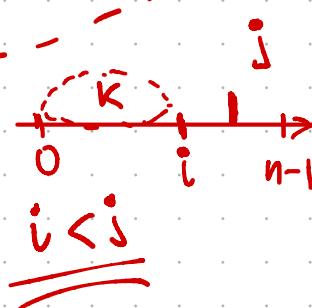
II  $(GS)^T \cdot r_j$   $d_i^T r_j = u_i^T r_j + \left( \sum_{k=0}^{i-1} \beta_{ik} d_k^T r_j \right)$

$$d_i^T r_j = u_i^T r_j + 0$$

0

$$u_i^T r_j = 0$$

$i < j$



Пуемб  $i = j$

$$d_i^T r_i = u_i^T r_i$$

(III)

$$\begin{aligned} r_{i+1} &= -A \cdot e_{i+1} = -A(e_i + d_i d_i) = -Ae_i - d_i A d_i = \\ &= r_i - d_i A d_i \end{aligned}$$

(II)

$$\begin{aligned} r_i^T r_j &= 0 \\ i < j \end{aligned}$$

Если меняем

CG:

$$u_i = r_i, \text{ TO}$$

$$\begin{aligned} r_i^T r_j &= 0 \\ i \neq j \end{aligned}$$

Рассмотрим:

$$(B) \quad \beta_{ij} = -\frac{u_i^T A d_j}{d_j^T A d_j}$$

Рассмотрим  
 $\hat{\alpha}_{i,j}$

$$r_i^T r_{j+1} \stackrel{(III)}{=} r_i^T (r_i - d_j A d_j) = r_i^T r_i - d_j r_i^T A d_j$$

$$d_j r_i^T A d_j = r_i^T r_j - r_i^T r_{j+1}$$

$$r_i^T A d_j = \begin{cases} \frac{1}{d_i} \cdot r_i^T r_i & i=j \\ -\frac{1}{d_{i-1}} r_i^T r_i & i=j+1 \\ 0 & j=i-1 \end{cases}$$

$j < i$

ВТОРО

БОЗРАГИЧЕСКА!

$$\begin{aligned} \beta_{ij} &= \frac{-r_i^T A d_j}{d_j^T A d_j} = + \frac{1}{d_{i-1}} \cdot \frac{r_i^T r_i}{d_{i-1}^T A d_{i-1}} = \\ &= \frac{d_{i-1}^T A d_{i-1}}{d_{i-1}^T r_{i-1}} \cdot \frac{r_i^T r_i}{d_{i-1}^T A d_{i-1}} \end{aligned}$$

$$\beta_{ij} = \frac{r_i^T r_i}{r_{i-1}^T r_{i-1}} \quad \text{где } j = i-1, \text{ иначе } 0$$

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

$$A \in \mathbb{S}^n_{++}$$

Let's  $x_0$

$$1. \quad d_0 = r_0 = b - Ax_0.$$

$$2. \quad \text{For } i = 0, \dots$$

$$3. \quad d_i = \frac{r_i^T r_i}{d_i^T A d_i}$$

$$4. \quad x_{i+1} = x_i + d_i d_i$$

$$5. \quad r_{i+1} = r_i - d_i A d_i$$

$$6. \quad \beta_{i+1} = \frac{r_{i+1}^T r_{i+1}}{r_i^T r_i}$$

$$7. \quad d_{i+1} = r_{i+1} + \beta_{i+1} d_i$$

но  $x_0$ :

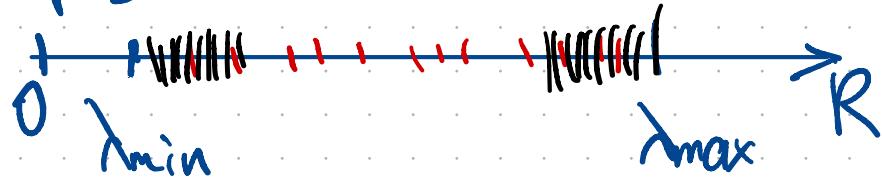
$$r_{i+1} = b - Ax_{i+1}$$

$\|$  вместо формулы  
- линейный  
поиск  
(backtrack)

С6.

Сходимость: **бесконечной арифметике**

сходится тогда и только тогда, когда  $m$  итераций, т.е.  $m$ -штук различных собственных чисел



В **реальности**

метод итерационный

$$\|\ell_K\|_A \leq 2 \left( \frac{\sqrt{x} - 1}{\sqrt{x} + 1} \right)^K \|\ell_0\|_A$$

$$x = \|A\| \cdot \|A^{-1}\| = \frac{\sigma_{\max}}{\sigma_{\min}} = \frac{\lambda_{\max}}{\lambda_{\min}} = \frac{L}{\mu}$$