# Постановка задачи

Минимизировать функцию $f(x) = \frac{1}{2}x^T Ax + bx$, где $x \in \mathbb{R}^6$,

$A_{6\times6}$ — произвольная положительно определенная матрица, A∈ {R} ^ {n×n}

$b$ — произвольный ненулевой вектор размерности 6, $b \in \mathbb{R}^{n\times1}$

$x_0$ — произвольный начальный ненулевой вектор размера 6, отдаленный от точного решения, $x \in \mathbb{R}^{n\times1}$

$$A^* = \begin{pmatrix} 65.1140547 & -26.7627461 & -22.0076980 & -82.7767129 & 35.0062152 & 75.5819058 \\ -26.7627461 & 229.0660173 & 19.9323298 & 11.3176738 & -41.5887940 & 27.5077199 \\ -22.0076980 & 19.9323298 & 47.8558662 & 92.3011760 & -20.1879154 & -18.1501243 \\ -82.7767129 & 11.3176738 & 92.3011760 & 231.2486741 & -30.5692642 & -103.3184336 \\ 35.0062152 & -41.5887940 & -20.1879154 & -30.5692642 & 130.1539779 & 82.9497276 \\ 75.5819058 & 27.5077199 & -18.1501243 & -103.3184336 & 82.9497276 & 189.0452850 \end{pmatrix}$$

**Из-за нехватки места количество записываемых десятичных цифр сократилось до 7.*

Процедура создания матрицы $A$ заключается в создании случайной обратимой матрицы $U$ (с определителем больше 0) и присвоении $A$ равного $M^T M$. Это гарантирует, что $A$ будет положительно определенной матрицей.

Тем не менее, можно убедиться, что матрица $A$ является положительно определенной, получив ее собственные значения и проверив, все ли они положительны и что $A$ симметрична. Ниже приводится набор собственных значений $A$:

$$\lambda = \{1.172, 17.858, 60.389, 158.527, 245.585, 408.953\}$$

$$b = \begin{pmatrix} 9.814255243856422 \\ -0.3400832138148502 \\ -4.2742798826246275 \\ 0.5729798470675629 \\ 1.0452978781511018 \\ -6.258477409122025 \end{pmatrix} \qquad x_0 = \begin{pmatrix} -6.998099039133139 \\ -5.532602553195874 \\ 5.222094920523634 \\ 3.71860843139695 \\ -5.88821505753657 \\ 4.175398532986163 \end{pmatrix}$$

$$x_{\text{точ}} = -A^{-1}b = \begin{pmatrix} -1.550648425654797 \\ -0.22570089760570633 \\ 3.4869725406369905 \\ -2.0145864008544634 \\ 0.6327463127507499 \\ -0.3579729727467522 \end{pmatrix}$$

# Метод градиента

$$x_{k+1} = x_k - \lambda f'(x_k), \text{ где } \lambda = 10^{-4}$$

Первая производная функции: $f'(x) = \frac{1}{2}(A^T + A)x + b$

Приравнивая производную к нулю, получаем вектор $x_{\text{точ}} \in \mathbb{R}^{n\times1}$

$$x_{\text{точ}} = -A^{-1}b = \begin{pmatrix} -1.550648425654797 \\ -0.22570089760570633 \\ 3.4869725406369905 \\ -2.0145864008544634 \\ 0.6327463127507499 \\ -0.3579729727467522 \end{pmatrix}$$

Алгоритм отработал за 24406 шагов. Условие выхода из цикла: $\|x_{k+1} - x_k\| < \varepsilon = 10^{-5}$

*Промежуточные результаты:*

$$x_{\frac{m}{4}} = x_{6101} = \begin{pmatrix} -1.3719100792558692 \\ -0.20442898556213976 \\ 2.887074967708234 \\ -1.6794775156491377 \\ 0.5082733976393556 \\ -0.25169818502348007 \end{pmatrix} \qquad x_{\frac{3m}{4}} = x_{18304} = \begin{pmatrix} -1.507841726609269 \\ -0.22060378567950611 \\ 3.343406739106548 \\ -1.9343835479751248 \\ 0.6029623016937389 \\ -0.332553486463615 \end{pmatrix}$$

$$x_{\frac{m}{2}} = x_{12203} = \begin{pmatrix} -1.4631521073720928 \\ -0.21528246944462 \\ 3.1935259611955886 \\ -1.850652841976869 \\ 0.5718681918959697 \\ -0.30601588147387093 \end{pmatrix} \qquad x_m = x_{24406} = \begin{pmatrix} -1.5297081304989302 \\ -0.22320747880544625 \\ 3.416742639117708 \\ -1.9753525554732039 \\ 0.6181764908218194 \\ -0.34553820068193253 \end{pmatrix}$$

*Промежуточные значения функционала:*

$$f\left(x_{\frac{m}{4}}\right) = f(x_{6101}) = -13.837969386830299$$

$$f\left(x_{\frac{m}{2}}\right) = f(x_{12203}) = -14.074776746789134$$

$$f\left(x_{\frac{3m}{2}}\right) = f(x_{18304}) = -14.131440448625039$$

$$f(x_m) = f(x_{24406}) = -14.145004232600975$$

Значение функционала в точке $x_{\text{точ}}$: $f(x_{\text{точ}}) = -14.149271102339297$ (точное решение)

*Погрешности метода градиента:*

$$\Delta(x_m, x_{\text{точ}}) = \begin{pmatrix} |x_{m1} - x_{\text{точ }1}| \\ \vdots \\ |x_{m6} - x_{\text{точ }6}| \end{pmatrix} = \begin{pmatrix} 0.020940295155866817 \\ 0.0024934188002600777 \\ 0.07022990151928266 \\ 0.03923384538125951 \\ 0.014569821928930526 \\ 0.012434772064819688 \end{pmatrix}$$

$$\Delta(f(x_m), f(x_{\text{точ}})) = 0.004266869738321688$$

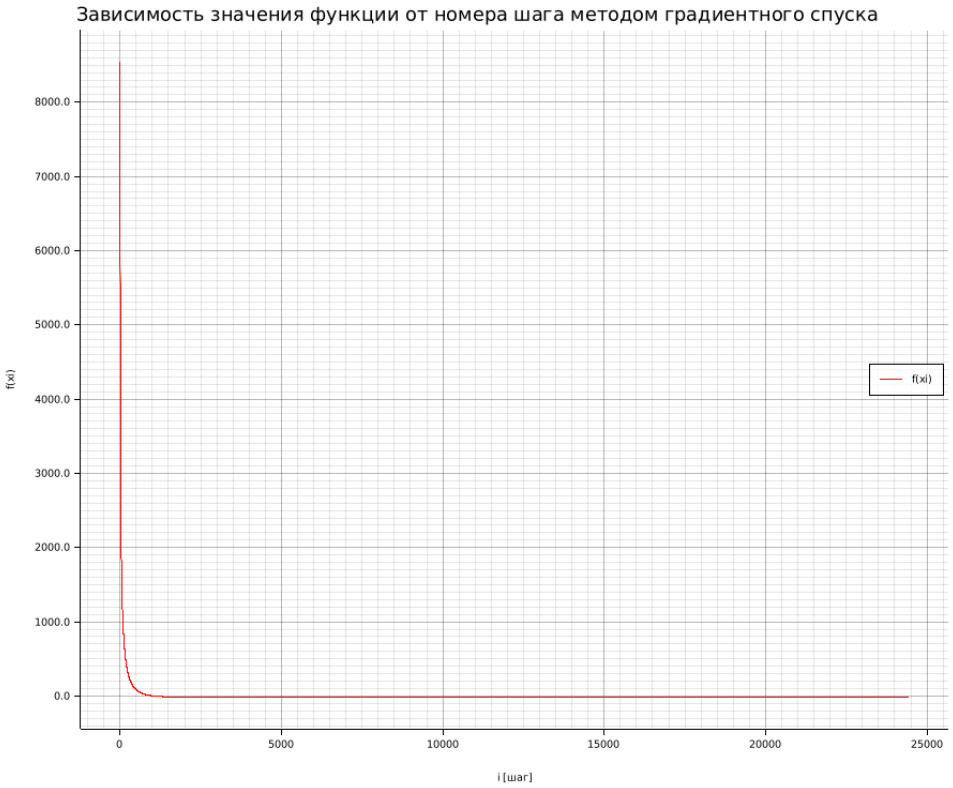


Рисунок 1: График зависимости значения функции от номера шага методом градиентного спуска

Следует отметить, что график не точно отражает реальное значение $f(x_{\text{точ}})$ при приближении к точному значению. Вот почему было явно определено значение $f(x_{\text{точ}})$, которое составляет примерно -14.149271102339297, и это значение, к которому сходится функция.

# Приложения

Приложение для этой работы было разделено на две части. Сначала была создана библиотека общего назначения, содержащая модуль под названием «lab1», содержащий все функции алгоритма. Затем основной файл бинарного файла содержит вызовы функций для генерации матрицы, расчета и регистрации необходимых значений, а также создания графика рассчитанных значений.

Весь исходный код этого приложения можно найти по адресу https://github.com/AJMC2002/opt-methods/tree/main/labs.

## Зависимости

- chrono = “0.4.31” — для регистрации в логгере времени каждого запуска.
- log2 = “0.1.10” — библиотека ведения логов.
- nalgebra = “0.32.3” — библиотека линейной алгебры.
- plotters = “0.3.5” — библиотека построения графиков.
- utils = { path = “../utils” } — пользовательская библиотека с алгоритмами, используемыми для этой работы.

## Библиотека

Библиотека использует три пользовательских враппер-типа: `type FloatingType`, `type __GenericSquareMatrix<const N: usize>` и `type __GenericVector<const N: usize>`. Они

определяют общую степень точности, которой будут обладать наши значения, и быстрый способ записи универсальных типов матриц постоянного размера и векторов соответственно.

Первая функция `fn new_positive_definite_matrix` генерирует случайную обратимую матрицу $M$, для которой все ее элементы ограничены аргументами `min` и `max`, затем возвращает положительно определенную матрицу $M^T M$.

Далее `fn gradient_method` принимает наш начальный вектор $x_0$, параметры итерации и функцию первой производной от $f(x)$, чтобы вернуть вектор всех векторов $x_i$, полученных в нашем итерационном процессе.

```rust
// utils/src/lib.rs
pub type FloatingType = f64;

pub mod lab1 {
    use nalgebra::{ArrayStorage, Const, DimMin, SquareMatrix, Vector};

    use crate::FloatingType;

    pub type __GenericSquareMatrix<const N: usize> =
        SquareMatrix<FloatingType, Const<N>, ArrayStorage<FloatingType, N, N>>;
    pub type __GenericVector<const N: usize> =
        Vector<FloatingType, Const<N>, ArrayStorage<FloatingType, N, 1>>;

    pub fn new_positive_definite_matrix<const N: usize>(
        min: FloatingType,
        max: FloatingType,
    ) -> __GenericSquareMatrix<N>
    where
        Const<N>: DimMin<Const<N>, Output = Const<N>>,
    {
        loop {
            let m = (max - min) * __GenericSquareMatrix::<N>::new_random()
                + __GenericSquareMatrix::<N>::from_element(min);
            if m.determinant() > 0 as FloatingType {
                return m.transpose() * m;
            }
        }
    }

    pub fn gradient_method<const N: usize, F>(
        x0: &__GenericVector<N>,
        lambda: FloatingType,
        epsilon: FloatingType,
        f_prime: F,
    ) -> Vec<__GenericVector<N>>
    where
        F: Fn(&__GenericVector<N>) -> __GenericVector<N>,
    {
        let mut x_log = vec![*x0];
        loop {
            let x = x_log.last().unwrap();
            let x_next = x - lambda * f_prime(x);
            if (x_next - x).norm() < epsilon {
                break;
            } else {
```

```
                x_log.push(x_next)
            }
        }
        x_log
    }
}
```

### Бинарный

Для наших вычислений нам нужно получить $f(x), f'(x), f^{-1}(x)$. Была определена структура для хранения этих различных определений функций с учетом их параметров $A$ и $b$.

```
// minimization/src/main.rs
struct Function<const N: usize> {
    a: __GenericSquareMatrix<N>,
    b: __GenericVector<N>,
}

impl<const N: usize> Function<N> {
    fn new(a: __GenericSquareMatrix<N>, b: __GenericVector<N>) -> Self {
        Self { a, b }
    }

    pub fn f(&self, x: &__GenericVector<N>) -> FloatingType {
        (x.transpose() * self.a * x)[(0, 0)] / 2 as FloatingType + self.b.dotc(x)
    }
    pub fn f_prime(&self, x: &__GenericVector<N>) -> __GenericVector<N> {
        (self.a + self.a.transpose()) * x / 2 as FloatingType + self.b
    }

    pub fn f_prime_inv(&self, f_prime_val: __GenericVector<N>) -> __GenericVector<N> {
        2 as FloatingType
            * (self.a.transpose() + self.a).try_inverse().unwrap()
            * (f_prime_val - self.b)
    }
}
```

Следующая основная функция в конечном итоге оказывается довольно простой. Мы получаем нашу случайную матрицу $A$ и векторы $b$ и $x_0$, получаем нужные нам результаты, регистрируем их и генерируем наш график *.

* *Была написана пользовательская функция* `fn plot_steps`*, но поскольку ее реализация на самом деле не входит в рамки данной работы, она не будет обсуждаться подробно.*

```
// minimization/src/main.rs
use chrono::Local;
use log2::{debug, info};
use nalgebra::{matrix, vector, Vector6};
use plotters::prelude::*;
use std::error::Error;
use utils::{
                lab1::{__GenericSquareMatrix,    __GenericVector,    gradient_method,
new_positive_definite_matrix},
    FloatingType,
};

const MIN: FloatingType = -10.0;
const MAX: FloatingType = 10.0;
```

```
const LAMBDA: FloatingType = 0.0001;
const EPSILON: FloatingType = 0.00001;

fn main() -> Result<(), Box<dyn Error>> {
    let _log2 = log2::open("output.log").start();

    let a = new_positive_definite_matrix::<6>(MIN, MAX);
    let b = (MAX - MIN) * Vector6::new_random() + Vector6::from_element(MIN);
    let x0 = (MAX - MIN) * Vector6::new_random() + Vector6::from_element(MIN);

    let function = Function::new(a, b);

    let x_steps = gradient_method(&x0, LAMBDA, EPSILON, |x| function.f_prime(x));
    let x_exact = function.f_prime_inv(Vector6::zeros());

    let steps = x_steps.len() - 1;
    let intermediate_results = [
        x_steps[0],
        x_steps[steps / 4],
        x_steps[steps / 2],
        x_steps[3 * steps / 4],
        x_steps[steps],
    ];

    info!("a {}", a);
    info!("b {}", b);
    info!("x0 {}", x0);
    info!("tochnoe {}", x_exact);
    info!("xm {}", x_steps.last().unwrap());
    info!("m (steps) {}", steps);
    info!(
        "intermediate steps\n{}",
        intermediate_results
            .iter()
            .enumerate()
            .map(|(i, it)| format!("x_{{{}m/{}}}\n{}", i, intermediate_results.len() -
1, it))
            .collect::<Vec<String>>()
            .join("")
    );
    info!("x (exact solution) {}", x_exact);
    info!(
        "f(intermediate steps)\n{}",
        intermediate_results
            .iter()
            .enumerate()
            .map(|(i, it)| format!(
                "f(x_{{{}m/{}}}) = {}\n",
                i,
                intermediate_results.len() - 1,
                function.f(it)
            ))
            .collect::<Vec<String>>()
            .join("")
    );
    info!("f(x) (exact solution) = {}", function.f(&x_exact));
```

```rust
    info!(
        "abs diff x vector = {}",
        (x_steps.last().unwrap() - x_exact).map(FloatingType::abs)
    );
    info!(
        "abs diff f(x) = {}",
        (function.f(x_steps.last().unwrap()) - function.f(&x_exact)).abs()
    );

    plot_steps(
        x_steps
            .iter()
            .map(|x| function.f(x))
            .collect::<Vec<FloatingType>>(),
    )?;

    Ok(())
}
```