Национальный исследовательский университет ИТМО
Прикладная математика и информатика

Методы оптимизации

Отчет по лабораторной работе №2
“Алгоритмы минимизации многомерных функций”

Выполнили:

Михайлов Максим
Загребина Мария
Кулагин Ярослав

Команда:
$\forall \bar{R} \in \mathscr{R}^n : \mathbf{R}(\bar{R}) \in \mathscr{R}$
(КаМаЗ)

Группа: М3237

Санкт-Петербург, 17 апреля 2021 г.

1 Цели

1. Реализовать алгоритмы минимизации функций:
   - Метод градиентного спуска
   - Метод наискорейшего спуска
   - Метод сопряженных градиентов
2. Исследовать скорость сходимости в зависимости от выбора алгоритма одномерной оптимизации в методе наискорейшего спуска.
3. Проанализировать траектории методов на различных квадратичных функциях
4. Исследовать зависимость числа итераций от следующих входных параметров:
   - Число обусловленности
   - Размерность пространства

2 Вычислительная схема методов

2.1 Метод градиентного спуска

Алгоритм метода:

1. Выбрать $\varepsilon > 0, \alpha > 0, x \in E_n$, вычислить $f(x)$.
2. Вычислить $\nabla f(x)$. Проверить условие $||\nabla f(x)|| < \varepsilon$. Если оно выполнено, то завершить процесс, иначе перейти к шагу 3.
3. Найти $y = x - \alpha \nabla f(x)$ и $f(y)$. Если $f(y) < f(x)$, принимаем $\alpha = \frac{\alpha}{2}$ и выполняем этот шаг ещё раз. Иначе положить $x = y, f(x) = f(y)$ и перейти к шагу 2.

2.2 Метод наискорейшего спуска

Алгоритм метода:

1. Выбрать $\varepsilon > 0, x \in E_n$, вычислить $f(x)$
2. Вычислить $\nabla f(x)$. Проверить условие $||\nabla f(x)|| < \varepsilon$. Если оно выполнено, то завершить процесс, иначе перейти к шагу 3.
3. Решить задачу одномерной оптимизации

$$\Phi(\alpha) \to \min \quad \Phi(\alpha) = f(x - \alpha \nabla f(x)), \alpha > 0$$

Положить $x = x - \alpha \nabla f(x)$, перейти к шагу 2.

2.3 Метод сопряженных градиентов

Алгоритм метода:

1. Выбрать $\varepsilon > 0, x \in E_n$, вычислить $\nabla f(x)$, задать $p = -\nabla f(x)$
2. Вычислить $\nabla f(x)$. Проверить условие $||\nabla f(x)|| < \varepsilon$. Если оно выполнено, то завершить процесс, иначе перейти к шагу 3.
3. Вычислить следующие значения:

$$t = Ap$$
$$\alpha = \frac{||\nabla f(x)||^2}{\langle t, p \rangle}$$
$$\tilde{x} = x + \alpha \cdot p$$
$$\nabla f(\tilde{x}) = \nabla f(x) + \alpha \cdot t$$
$$\beta = \frac{||\nabla f(\tilde{x})||^2}{||\nabla f(x)||^2}$$
$$p = -\nabla f(\tilde{x}) + p \cdot \beta$$

   И принять $\tilde{x}$ за x, $\nabla f(\tilde{x})$ за $\nabla f(x)$

3 Ход работы

3.1 Скорость сходимости в зависимости от выбора алгоритма одномерной оптимизации

Следующие измерения проводились для функции $f = x^2 + y^2 - xy + 4x + 3y - 1$ с начальной точкой $(2, 2)$, искомой точностью $\varepsilon = 10^{-6}$.

Аналитическое решение задачи:

$$\begin{cases} f'_x = 2x - y + 4 \\ f'_y = 2y - x + 3 \end{cases}$$
$$\begin{cases} 0 = 2x - y + 4 \\ 0 = 2y - x + 3 \end{cases}$$
$$\begin{cases} x = -\frac{11}{3} \\ y = -\frac{10}{3} \end{cases}$$

1. Метод золотого сечения

№	x_1	x_2	α	Итераций одномерного поиска
0	2	2	0.666666	19
1	1.48785	1.57321	0.666666	19
2	0.982971	1.13785	0.666666	19
3	0.484755	0.69488	0.666666	19
4	-0.00744836	0.245237	0.666666	19
5	-0.494338	-0.210155	0.666666	19
6	-0.976651	-0.670391	0.666666	19
7	-1.45515	-1.13459	0.666666	19
8	-1.93063	-1.60188	0.666666	19
9	-2.4039	-2.07141	0.666666	19
10	-2.87578	-2.54234	0.666666	19
11	-3.34709	-3.01384	0.451891	19
12	-3.66675	-3.33325	0.000112783	19
13	-3.66667	-3.33333	8.85782e-07	19
14	-3.66667	-3.33333	0	0

2. Метод фибоначчи

№	x_1	x_2	α	Итераций одномерного поиска
0	2	2	0.666666	30
1	1.48785	1.57321	0.666666	30
2	0.98297	1.13785	0.666666	30
3	0.484754	0.694879	0.666666	30
4	-0.00745045	0.245235	0.666666	30
5	-0.494341	-0.210157	0.666666	30
6	-0.976654	-0.670394	0.666666	30
7	-1.45516	-1.13459	0.666666	30
8	-1.93064	-1.60188	0.666666	30
9	-2.40391	-2.07142	0.666666	30
10	-2.87578	-2.54235	0.666666	30
11	-3.3471	-3.01384	0.451883	30
12	-3.66675	-3.33325	0.000113101	30
13	-3.66667	-3.33333	8.85782e-07	0

3. Метод золотого сечения

№	x_1	x_2	α	Итераций одномерного поиска
0	2	2	0.666666	27
1	1.48785	1.57321	0.666666	27
2	0.982971	1.13785	0.666666	27
3	0.484755	0.69488	0.666666	27
4	-0.00744873	0.245237	0.666666	27
5	-0.494339	-0.210155	0.666666	27
6	-0.976651	-0.670392	0.666666	27
7	-1.45515	-1.13459	0.666666	27
8	-1.93063	-1.60188	0.666666	27
9	-2.4039	-2.07141	0.666666	27
10	-2.87578	-2.54234	0.666666	27
11	-3.34709	-3.01384	0.45189	27
12	-3.66675	-3.33325	0.000113103	27
13	-3.66667	-3.33333	8.85782e-07	0

4. Метод Брента

№	x_1	x_2	α	Итераций одномерного поиска
0	2	2	0.666666	14
1	1.48785	1.57321	0.666666	14
2	0.982971	1.13785	0.666666	14
3	0.484754	0.694879	0.666666	14
4	-0.00744959	0.245236	0.666666	14
5	-0.49434	-0.210156	0.666666	14
6	-0.976653	-0.670393	0.666666	14
7	-1.45515	-1.13459	0.666666	14
8	-1.93064	-1.60188	0.666666	14
9	-2.4039	-2.07142	0.666666	14
10	-2.87578	-2.54235	0.666666	14
11	-3.34709	-3.01384	0.451887	5
12	-3.66675	-3.33325	0.000113191	18
13	-3.66667	-3.33333	8.85782e-07	0

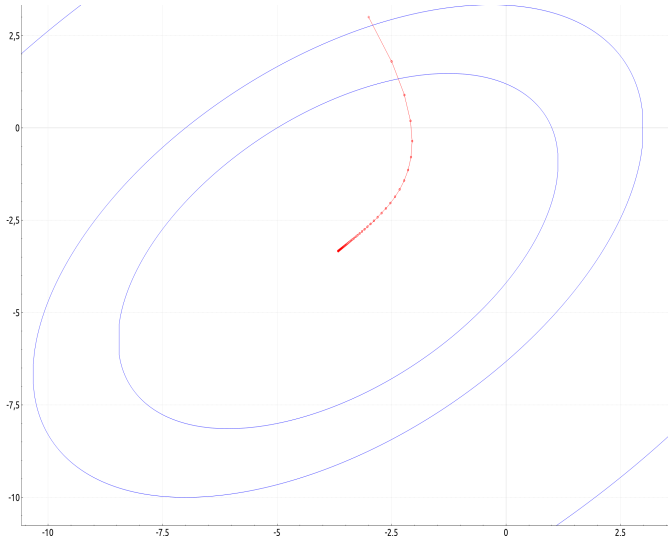
Таким образом, количество итераций метода наискорейшего спуска не зависит от выбора способа одномерного поиска, но это влияет на скорость работы. Метод парабол не работает для данной функции, т.к. при одномерной оптимизации используется функция с минимумом на границе отрезка допустимых решений. Такие функции не оптимизируются методом парабол, в силу невозможности найти начальное приближение $\tilde{x}$ такое, что $f(x_1) \leq f(\tilde{x})$, $f(x_2) \geq f(\tilde{x})$, $\tilde{x} \neq x_1$, $\tilde{x} \neq x_2$, где x_1, x_2 — границы отрезка, на котором производится оптимизация.

3.2 Траектории методов на различных функциях

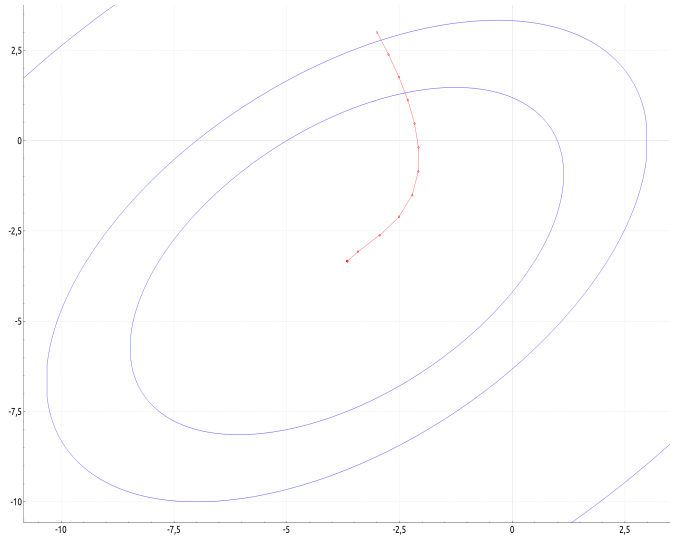
Во всех вычислениях для алгоритма наискорейшего спуска использовался метод Брента.
$\varepsilon = 10^{-6}$
Начальная точка (-3, 3)

1. $f_1 = x^2 + y^2 - xy + 4x + 3y - 1$
   Точка минимума (-3.66667 -3.33333), $\alpha = 0.5$ для метода градиентного спуска, начальная точка (-3, 3)

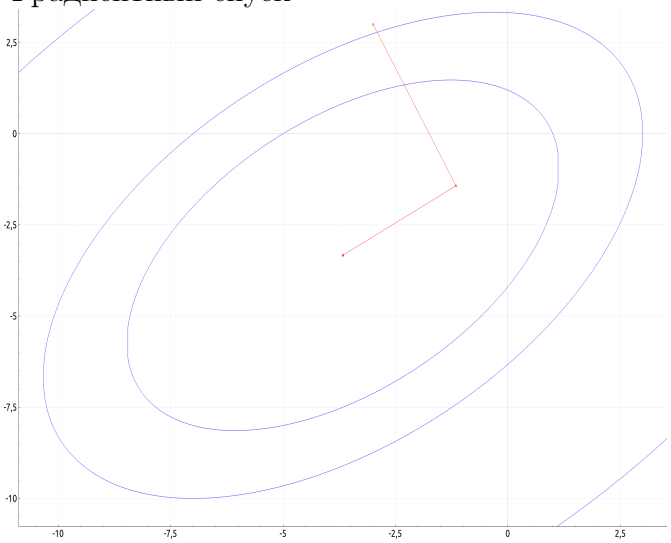
Метод	x_1	x_2	Количество итераций
Градиентный спуск	-3.66667	-3.33333	33
Наискорейший спуск	-3.66667	-3.33333	22
Сопряженные градиенты	-3.66667	-3.33333	3



Градиентный спуск



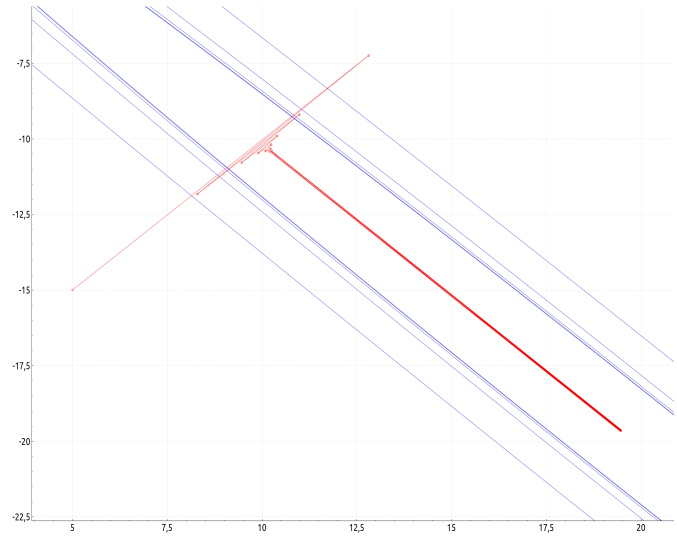
Наискорейший спуск



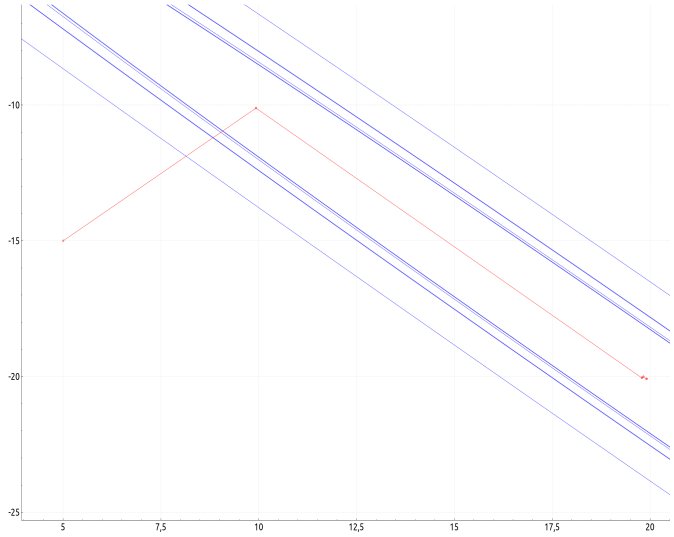
Сопряженные градиенты

2. $f_2 = 254x^2 + 506xy + 254y^2 + 50x + 130y - 111$
   Точка минимума (19.9112, -20.0888), $\alpha = 0.00196$, начальная точка (5, -15)

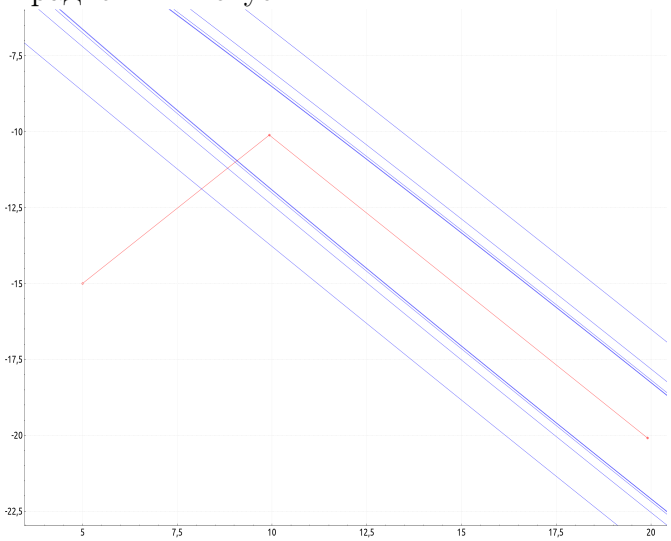
Метод	x_1	x_2	Количество итераций
Градиентный спуск	19.7061	-19.8836	994
Наискорейший спуск	6.40234	-13.6108	1002
Сопряженные градиенты	19.9112	-20.0888	3



Градиентный спуск



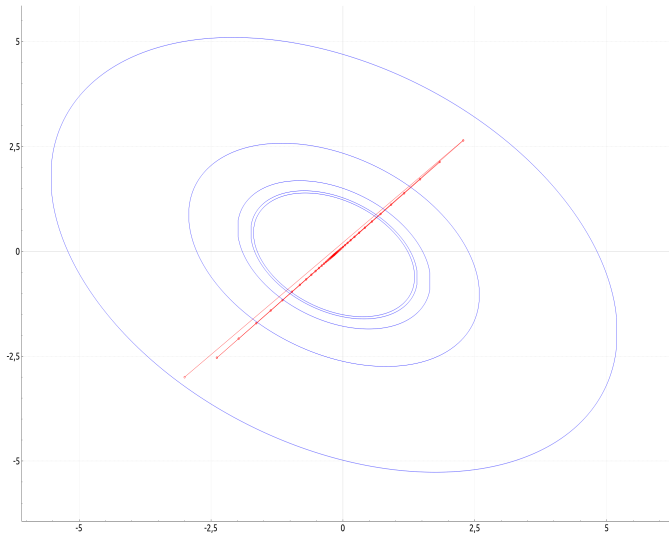
Наискорейший спуск



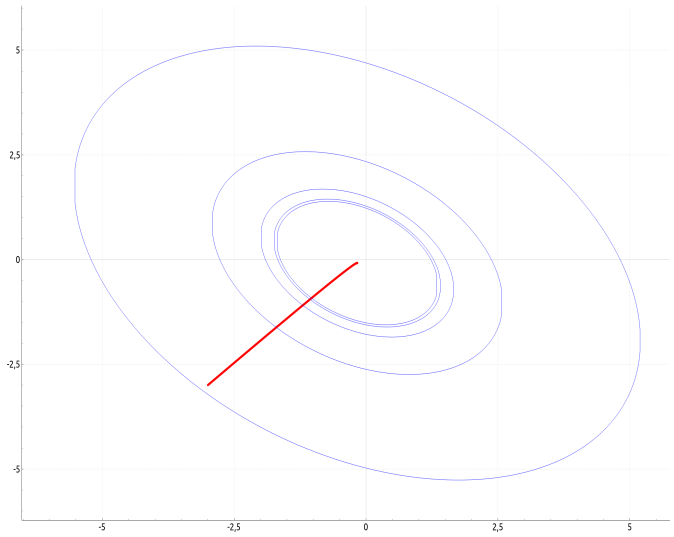
Сопряженные градиенты

3. $f_3 = 108x^2 + 116y^2 + 80xy + 43x + 33y - 211$
   Точка минимума (-0.195879, -0.13802), $\alpha = 0.00446$, начальная точка (-3, -3)

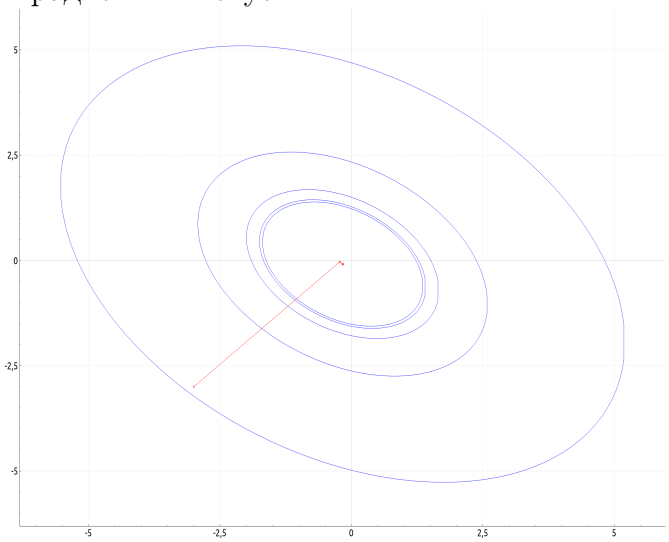
Метод	x_1	x_2	Количество итераций
Градиентный спуск	-0.167826	-0.0843708	157
Наискорейший спуск	-0.167826	-0.08437	626
Сопряженные градиенты	-0.167826	-0.0843704	3



Градиентный спуск



Наискорейший спуск

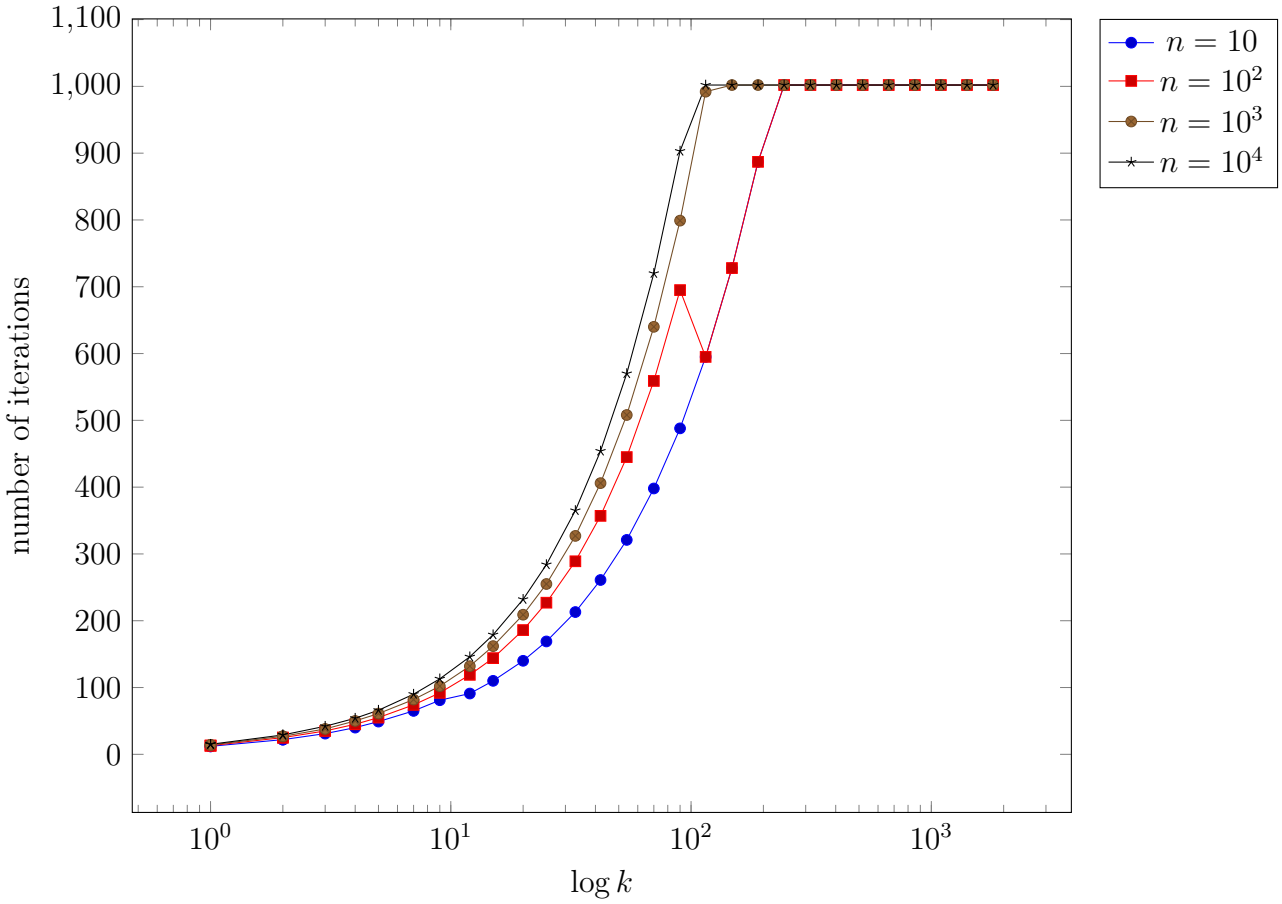


Сопряженные градиенты

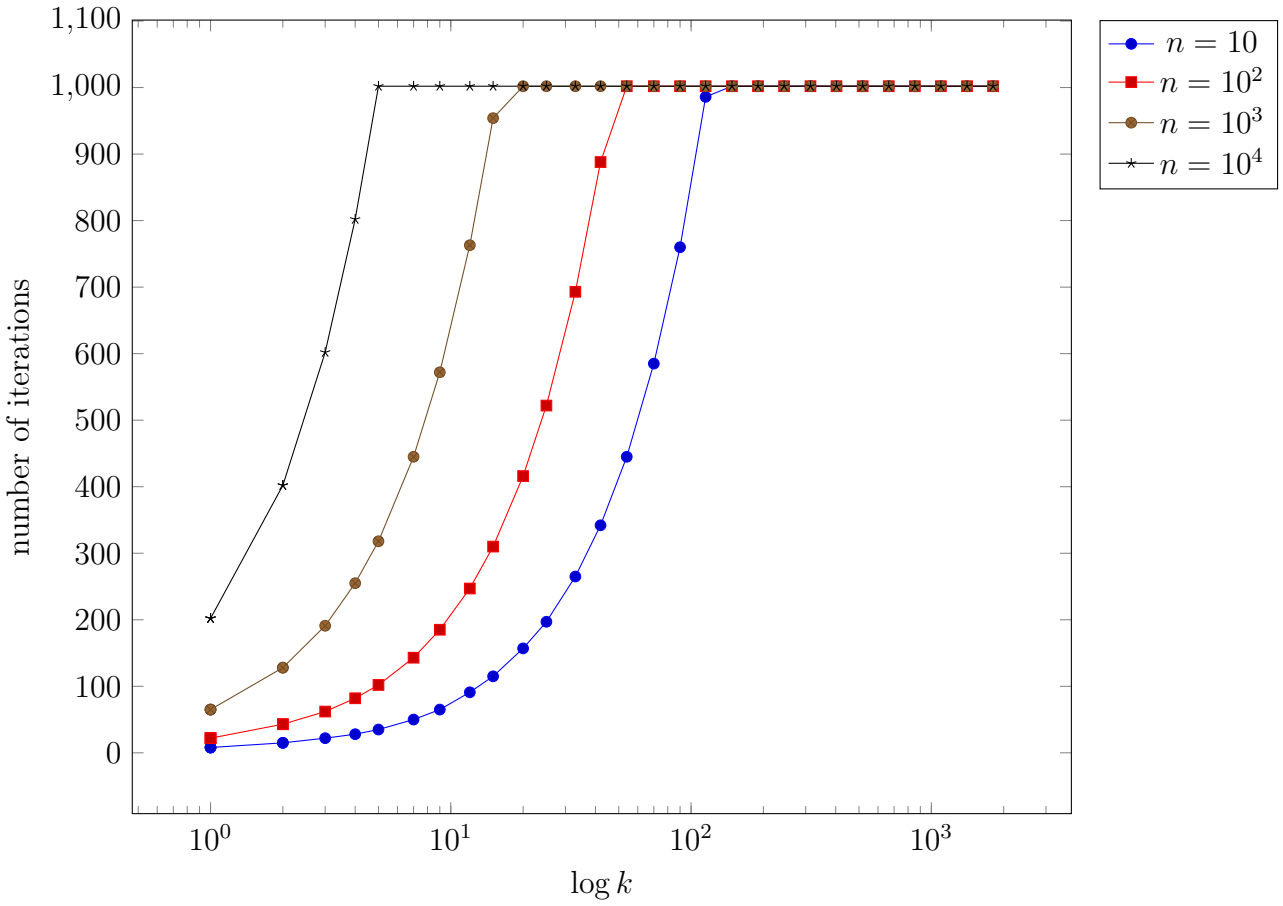
По данным таблиц видно, что первые 2 метода в общем случае не могут найти минимум овражной функции (вторая) и используют во много раз больше итераций для произвольных функций, чем третий метод. И еще что-нибудь на основе траекторий.

3.3 Скорость сходимости в зависимости от числа обусловленности и размерности пространства

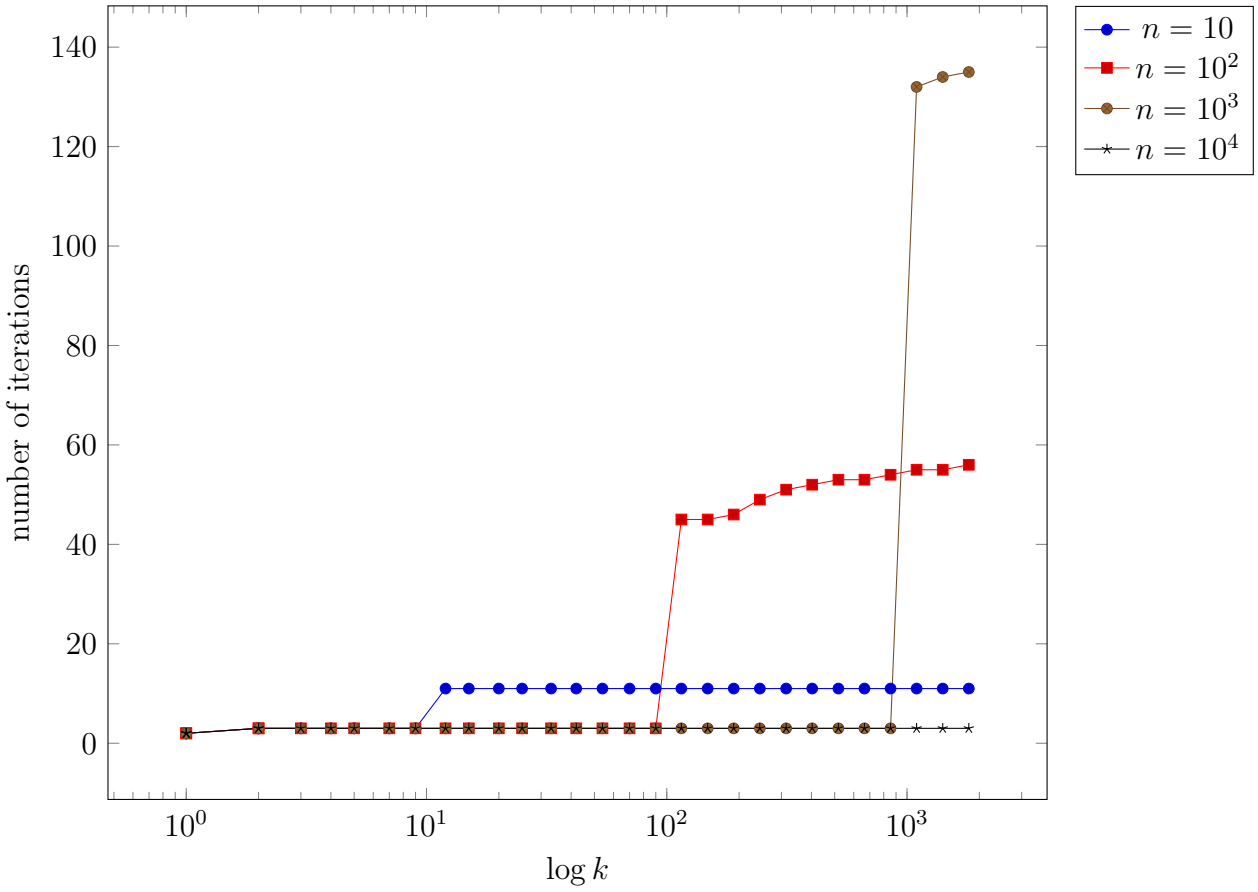
- Метод градиентного спуска



- Метод наискорейшего спуска



- Метод сопряженных градиентов



4 Выводы

5 Графический интерфейс

6 Код

6.0.1 Вектор

```hpp
#pragma once

#include <functional>
#include <vector>

namespace lab2 {
    /**
     * Вектор
     */
    class Vector {
      private:
        std::vector<double> data;

      public:
        Vector(const std::vector<double>& data);
        Vector(std::size_t size,
               const std::function<double(std::size_t)>& generator);

        double operator[](std::size_t idx) const;
        [[nodiscard]] std::size_t size() const;
        [[nodiscard]] double norm() const;

        Vector operator+(Vector other) const;
        Vector operator-(Vector other) const;
        double operator*(const Vector& other) const;
        Vector operator*(double val) const;
        bool operator==(const Vector& other) const;
    };

}  // namespace lab2
```

```cpp
#include "lab2/vector.h"

#include <cmath>
#include <utility>

using namespace lab2;

Vector::Vector(const std::vector<double>& data_) : data(data_) {
    if (size() == 0) {
        throw "Vector is empty";
    }
}

Vector::Vector(std::size_t size,
               const std::function<double(std::size_t)>& generator) {
    data.reserve(size);
    for (std::size_t i = 0; i < size; i++) {
        data.push_back(generator(i));
```

```
    }
}

std::size_t Vector::size() const { return data.size(); }

double Vector::operator[](std::size_t idx) const { return data[idx]; }

double Vector::norm() const { return std::sqrt((*this) * (*this)); }

Vector Vector::operator+(Vector other) const {
    if (size() != other.size()) {
        throw "Size mismatch";
    }
    return Vector(size(), [this, &other](std::size_t i) {
        return (*this)[i] + other[i];
    });
}

Vector Vector::operator-(Vector other) const {
    if (size() != other.size()) {
        throw "Size mismatch";
    }
    return Vector(size(), [this, &other](std::size_t i) {
        return (*this)[i] - other[i];
    });
}

Vector Vector::operator*(double val) const {
    std::vector<double> tmp;
    return Vector(size(), [this, val](std::size_t i) {
        return (*this)[i] * val;
    });
}

double Vector::operator*(const Vector& other) const {
    if (size() != other.size()) {
        throw "Size mismatch";
    }
    double res = 0;
    for (std::size_t i = 0; i < size(); i++) {
        res += (*this)[i] * other[i];
    }
    return res;
}

bool Vector::operator==(const Vector& other) const {
    return data == other.data;
}
```

6.0.2 Общий класс матриц

```hpp
#pragma once

#include <functional>
#include <vector>

namespace lab2 {
    /**
     * Вектор
     */
    class Vector {
      private:
        std::vector<double> data;

      public:
        Vector(const std::vector<double>& data);
        Vector(std::size_t size,
               const std::function<double(std::size_t)>& generator);

        double operator[](std::size_t idx) const;
        [[nodiscard]] std::size_t size() const;
        [[nodiscard]] double norm() const;

        Vector operator+(Vector other) const;
        Vector operator-(Vector other) const;
        double operator*(const Vector& other) const;
        Vector operator*(double val) const;
        bool operator==(const Vector& other) const;
    };

}  // namespace lab2
```

```cpp
#include "lab2/vector.h"

#include <cmath>
#include <utility>

using namespace lab2;

Vector::Vector(const std::vector<double>& data_) : data(data_) {
    if (size() == 0) {
        throw "Vector is empty";
    }
}

Vector::Vector(std::size_t size,
               const std::function<double(std::size_t)>& generator) {
    data.reserve(size);
    for (std::size_t i = 0; i < size; i++) {
        data.push_back(generator(i));
    }
}

std::size_t Vector::size() const { return data.size(); }

double Vector::operator[](std::size_t idx) const { return data[idx]; }

double Vector::norm() const { return std::sqrt((*this) * (*this)); }
```

```cpp

Vector Vector::operator+(Vector other) const {
    if (size() != other.size()) {
        throw "Size mismatch";
    }
    return Vector(size(), [this, &other](std::size_t i) {
        return (*this)[i] + other[i];
    });
}

Vector Vector::operator-(Vector other) const {
    if (size() != other.size()) {
        throw "Size mismatch";
    }
    return Vector(size(), [this, &other](std::size_t i) {
        return (*this)[i] - other[i];
    });
}

Vector Vector::operator*(double val) const {
    std::vector<double> tmp;
    return Vector(size(), [this, val](std::size_t i) {
        return (*this)[i] * val;
    });
}

double Vector::operator*(const Vector& other) const {
    if (size() != other.size()) {
        throw "Size mismatch";
    }
    double res = 0;
    for (std::size_t i = 0; i < size(); i++) {
        res += (*this)[i] * other[i];
    }
    return res;
}

bool Vector::operator==(const Vector& other) const {
    return data == other.data;
}
```