Университет ИТМО

Факультет информационных технологий и программирования

Направление прикладной математики и информатики

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2

курса «Методы оптимизации»

Выполнили студенты:

Мозжевилов Данил, Кучма Андрей
Группы: M3238, M3239

Санкт-Петербург, 18 апреля 2021 г.

Оглавление

Глава 1

Методы многомерной оптимизации

1.1. Постановка задачи и цель работы

1. Реализовать алгоритмы:

   - Метод градиентного спуска
   - Метод наискорейшего спуска
   - Метод сопряженных градиентов

   Оценить как меняется скорость сходимости, если для поиска величины шага используются различные методы одномерного поиска.

2. Проанализировать траектории методов для нескольких квадратичных функций: придумайте две-три квадратичные двумерные функции, на которых работа каждого из методов будет отличаться. Нарисовать графики с линиями уровня функций и траекториями методов.

3. Исследовать, как зависит число итераций, необходимое методам для сходимости, от следующих двух параметров:

   - числа обусловленности $k \geq 1$ оптимизируемой функции
   - размерности пространства n оптимизируемых переменных

   Сгенерировать от заданных параметров k и n квадратичную задачу размерности n с числом обусловленности k и запустить на ней методы многомерной оптимизации с некоторой заданной точностью. Замерить число итераций $T(n, k)$, которое потребовалось сделать методу до сходимости.

1.2. Общая схема того, как мы реализовывали алгоритмы

В начале мы создали классы Matrix, DiagonalMatrix и Vector и для них перегрузили операторы $'+'$, $'-'$, $'*'$ и $'[\,]'$ (класс DiagonalMatrix появился только под конец, когда мы уже начали тестировать и узнали, что для тестов нужны только диагональные матрицы и оказалось, что в коде для матриц испльзовался только оператор $'*'$, поэтому мы не стали реализовывать остальные перегрузки для этого класса).

Далее мы решили не использовать лямда-функции для задания квадратичных форм, а сделать отдельные классы QuadraticFunction и DiagonalQuadraticFunction, в которых храниться матрица A, вектор b и число c, и просто передавать их в качестве параметров в реализуемые алгоритмы, к тому же в классе можно хранить всю историю обращения к функции, что мы и делали.

Также мы создали класс GeneratorQudraticFunction, который генерировал рандомые вектора по заданной размерности и числу обусловленности.

Пример того, как выглядели наши сгенерированные функции:

```
1 38.0198 208.636 276.712 419.618 517.318 549.321 565.029 598.464 641
1 56.0696 86.0772 94.8904 129.966 133.73 151.615 295.072 304.457 330.866
1 121.07 250.754 316.186 452.644 463.517 492.598 526.129 690.467 706.2
```

Первая строчка это диагональная матрица A. В данном случае с числом обусловленноости 641. Вторая строчка это вектор b. Прибавление константы мы решили не генерировать, так как на поиск точки минимума она не влияет. Третья строка это начальное приближение x_0.

Все сгенерированные функции мы сохраняли в файлы, и благодаря этому не приходилось заново генерировать функции для каждого запуска программы, а также была возможность запускать тестирование на каждом методе отдельно.

Точность для алгоритмов мы решили задать всего лишь 0.01, так как при тестировании не хотелось ждать по 30 минут, пока алгоритмы найдут необходимый минимум для всех сгенерированных функций. Да и это не требовалось, так как судя по данным, которые мы получали, этой точности хватало, чтобы получать приближение до пяти знаков после запятой. Также при вычислении минимума у функции размерности $n = 10^4$ пришлось ограничится числом обусловленности до $k = 1000$, так как наши алгоритмы работали очень долго.

Отрисовывали графики мы средствами ЯП Python. Основная библиотека, которая использовалась — matplotlib. Была написана программа, которая по трём выходным .csv от основной программы, генерировала код и исполняла его. Так же в целях получать более понятные графики отрисовывание векторов и кривых уровня делалось только для первых 15 входных точек.

1.3. Иллюстрации работы градиентных методов на двумерных квадратичных функциях

1.3.1. Первая функция

Рассмотрим функцию

$$f(x,y) = x^2 - xy + 4y^2 + 2x + y$$

. В матричном виде ее вид $f(x) = 1/2 * (Ax, x) + b * x$, где $A = \begin{pmatrix} 2 & -1 \\ -1 & 8 \end{pmatrix}$ и $b = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$.

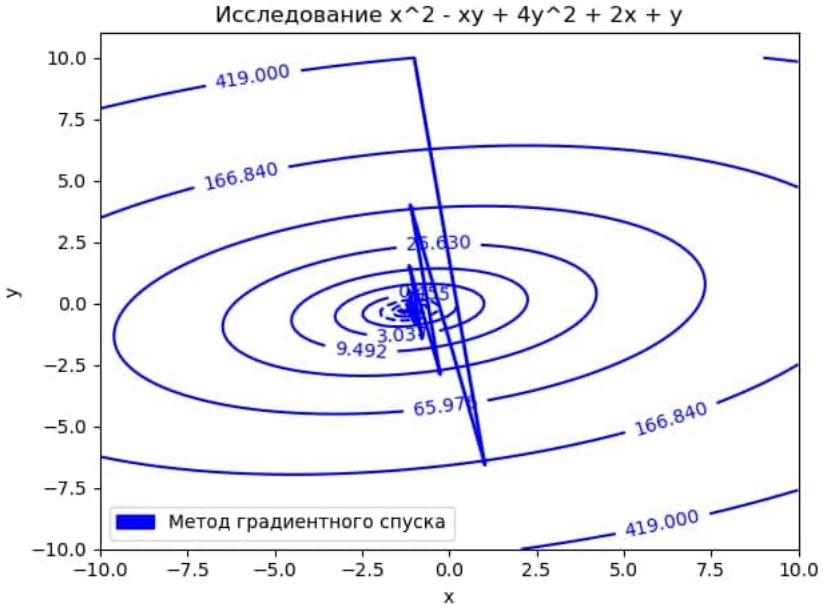
$det(A - \lambda E) = \begin{vmatrix} 2-\lambda & -1 \\ -1 & 8-\lambda \end{vmatrix} = (2-\lambda) * (8-\lambda) - 1 = 15 - 10 * \lambda + \lambda^2 = (5+\sqrt{10}-\lambda) * (5-\sqrt{10}-\lambda)$. Собственные значение матрицы A положительны, следовательно квадратичная форма f положительно определенная, а следовательно выпукла вниз. Число обусловленности $k = \frac{5+\sqrt{10}}{5-\sqrt{10}} \approx 4.4415$. Для начала найдем точку минимума функции аналитически.

Надем точку, в которой градиент данной функции обращается в ноль. Это и будет точка минимума функции. $grad\ f = \begin{pmatrix} 2*x - y + 2 & -x + 8y + 1 \end{pmatrix}^T = (0\ 0)^T$. Решив систему линейных уравнений, получаем

$$x_m = -\frac{17}{15},\ y_m = -\frac{4}{15},\ f(x_m, y_m) = -\frac{19}{15}$$

Теперь покажем как наши методы находили минимум этой функции. Начальную точку для всех методов мы взяли $x_0 = (-1, 10)$

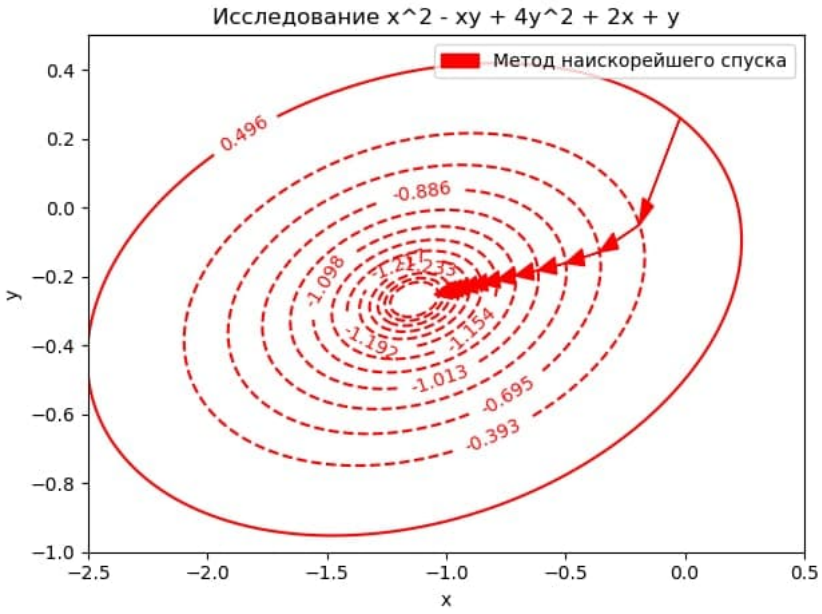
1. Метод градиентного спуска



Как мы видим у этой функции происходят биения.

Всего в алгоритме произошло 35 итераций. Найденная точка минимума $x_m = [-1.1333329668, -0.26666772037]$.

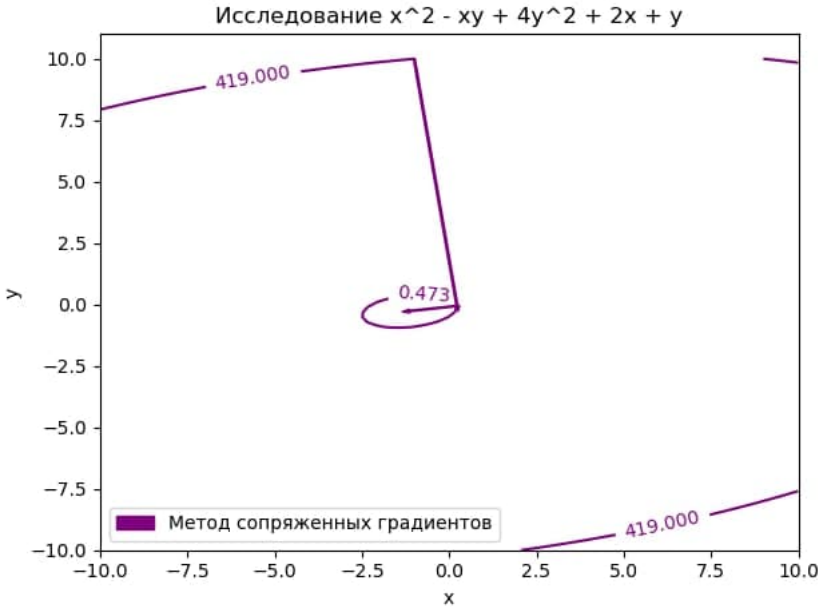
2. Метод наискорейшего спуска



Мы увеличили масштаб, так как большинство итераций происходило около минимума.

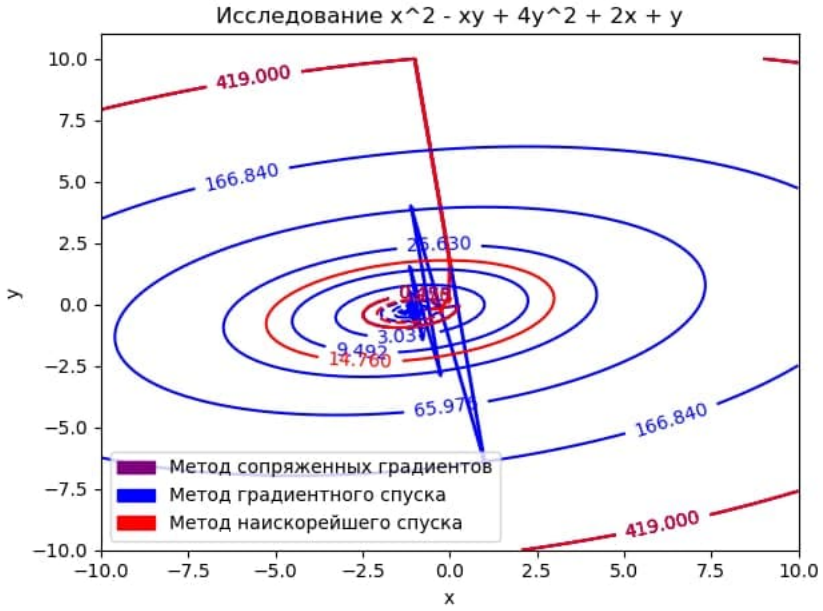
Итераций произошло больше, чем в предыдущем методе – 63. Найденная точка минимума $x_m = [-1.1333273611, -0.2666656975]$. Также заметим, что вектора не касаются линий уровня, мы это объясним ниже.

3. Метод сопряженных градиентов



Всего итераций 2. Найденная точка минимума
$x_m = [-1.1333333333, -0.26666666667]$.

Также общая картинка для всех методов.



1.3.2. Вторая функция

Рассмотрим вторую функцию для исследования

$$f(x, y) = 8x^2 + 5y^2 + 10xy + 5x + 6y$$

Аналогично предыдущей функции находим собственные значения, убедимся, что они положительны, находим число обусловленности, находим точку минимума и само значение минимума.

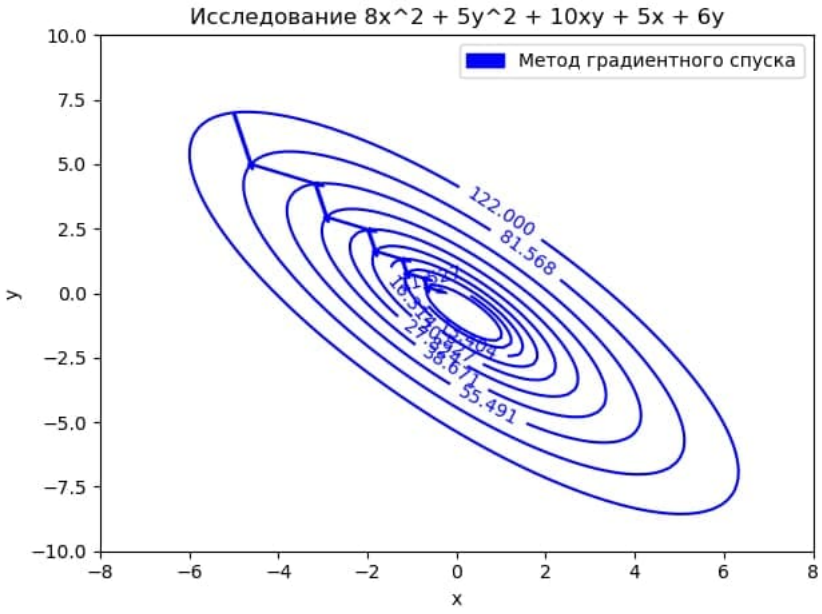
$$k \approx 9.1574$$

$$x_m = \frac{1}{6},\ y_m = -\frac{23}{30}$$

$$f(x_m, y_m) = -\frac{133}{60}$$

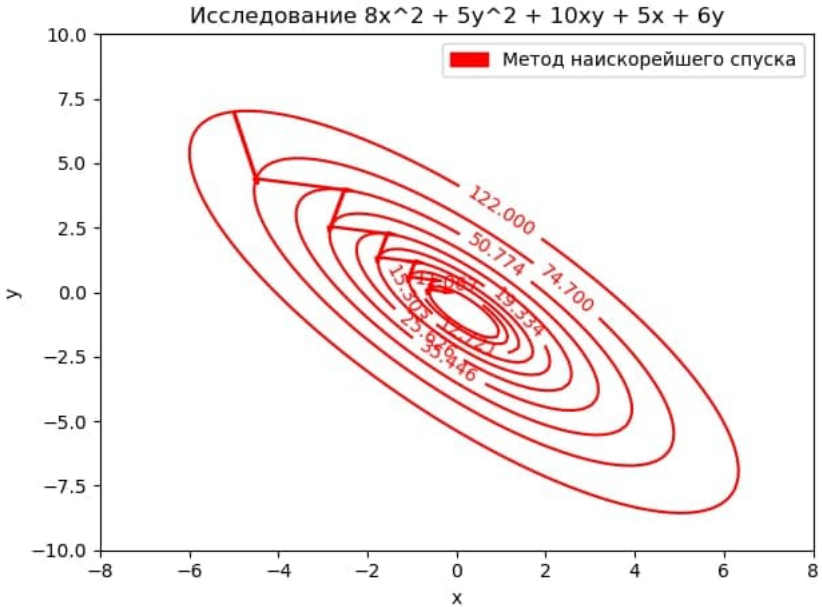
Работа алгоритмов. Начальная точка $x_0 = (-5, 7)$

1. Метод градиентного спуска



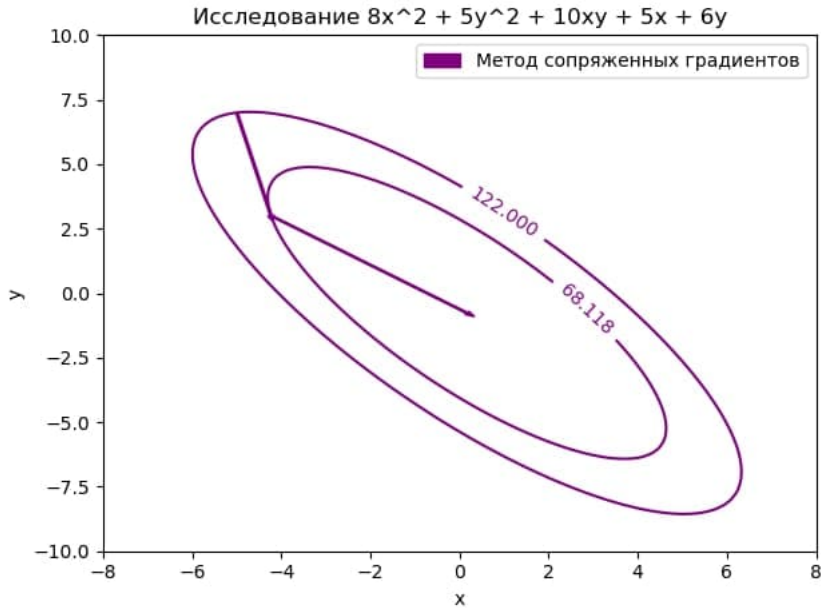
Как мы видим в начале происходит значительно меньше биений, но число итераций все равно больше – 68, из-за того, что в конце их было много (этого нет на рисунке, так как мы ограничили число векторов). Найденная точка минимума $x_m = [0.16666493859, -0.76666406897]$.

2. Метод наискорейшего спуска



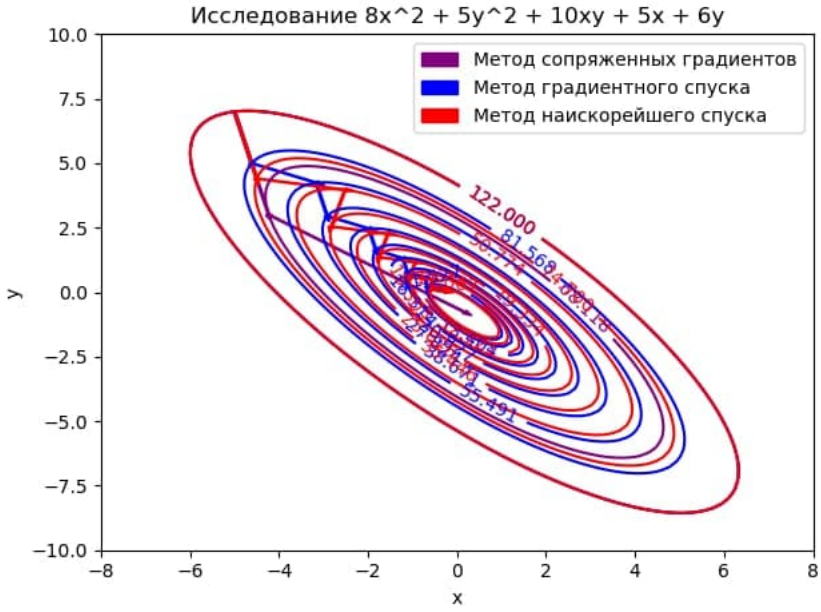
Всего итераций 67. В конце также присутствуют биения. И найденая точка минимума
$x_m = [0.16666492721, -0.7666635937]$. Также отметим, что в отличие от предыдущей функции, уже есть касание линий уровня у большинства векторов.

3. Метод сопряженных градиентов



Как и на прошлой функции всего 2 итерации. Найденная точка минимума $x_m = [0.16666666667, -0.76666666667]$.

Также все методы на одной картинке.



1.3.3. Третья функция

Ну и наконец третья функция

$$f(x, y) = 22x^2 + 2y^2 + xy + x + y$$

Аналогично с предыдущими

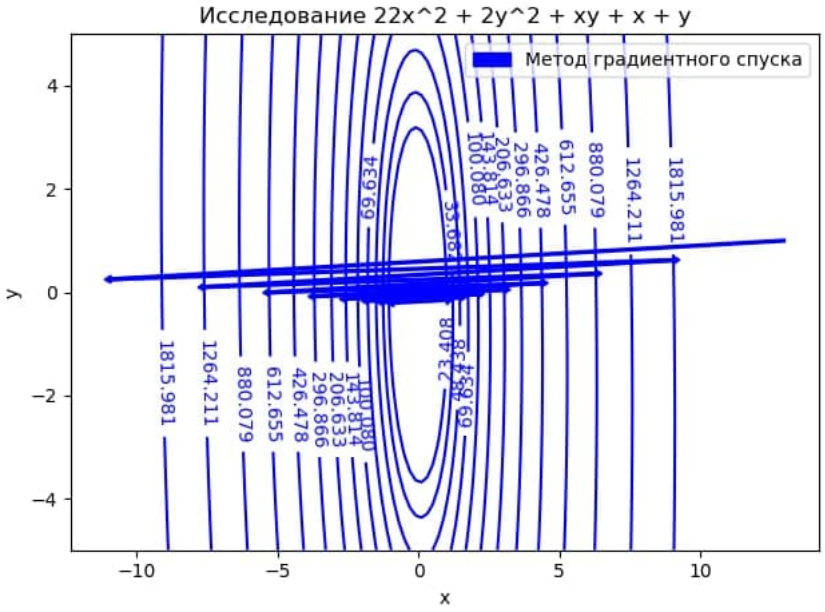
$$k \approx 11.07542$$

$$x_m = -\frac{3}{175},\ y_m = -\frac{43}{175},\ f(x_m, y_m) = -\frac{23}{175}$$

Начальная точка $x_0 = (13, 1)$.

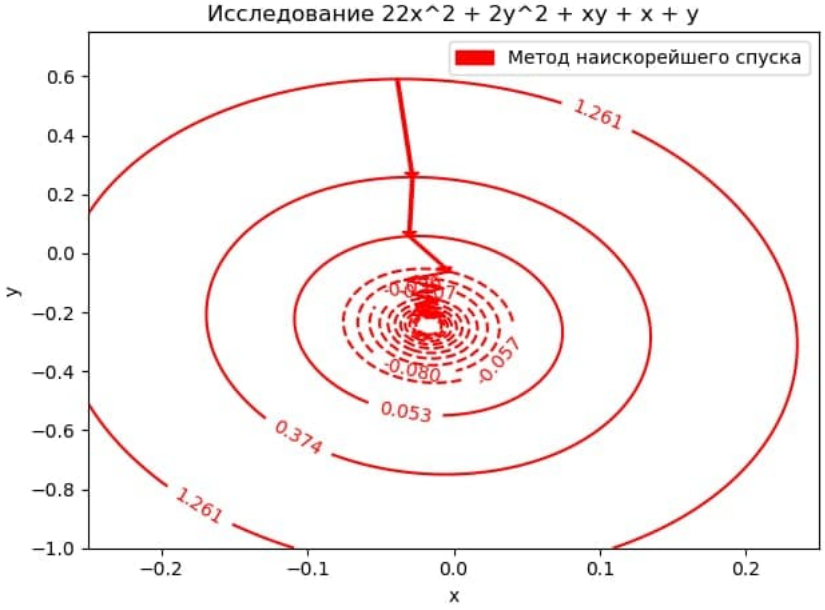
1. Метод градиентного спуска

   Как мы видим в отличие от двух предыдущих функций, если у первой происходили бения только в начале,а у второй только в конце, то тут биения происходят на протежении всего алгоритма, итого итераций – 99. Найденная точка минимума $x_m = [-0.017143071323, -0.24571427597]$.



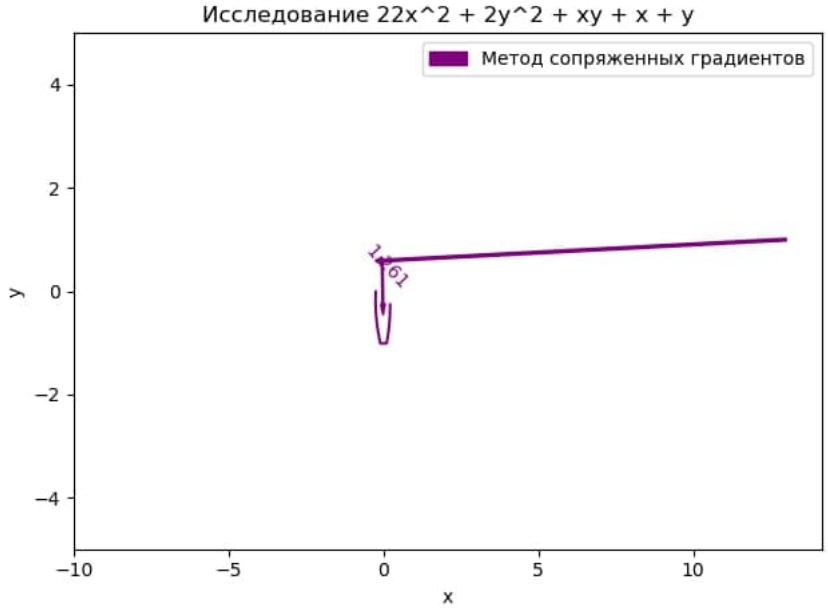
2. Метод наискорейшего спуска

   Всего итераций 67 и также большинство итераций произошло в конце. Найденная точка минимума $x_m = [-0.017142696872, -0.24571224089]$.

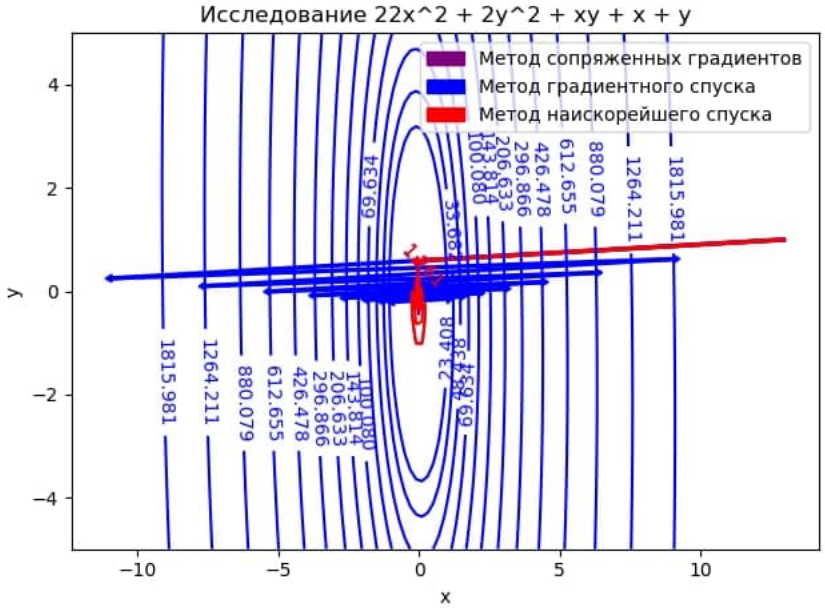


3. Метод сопряженных градиентов

   Аналогично предыдущим методам 2 итреции. Найденная точка минимума $x_m = [-0.017142857143, -0.24571428571]$.



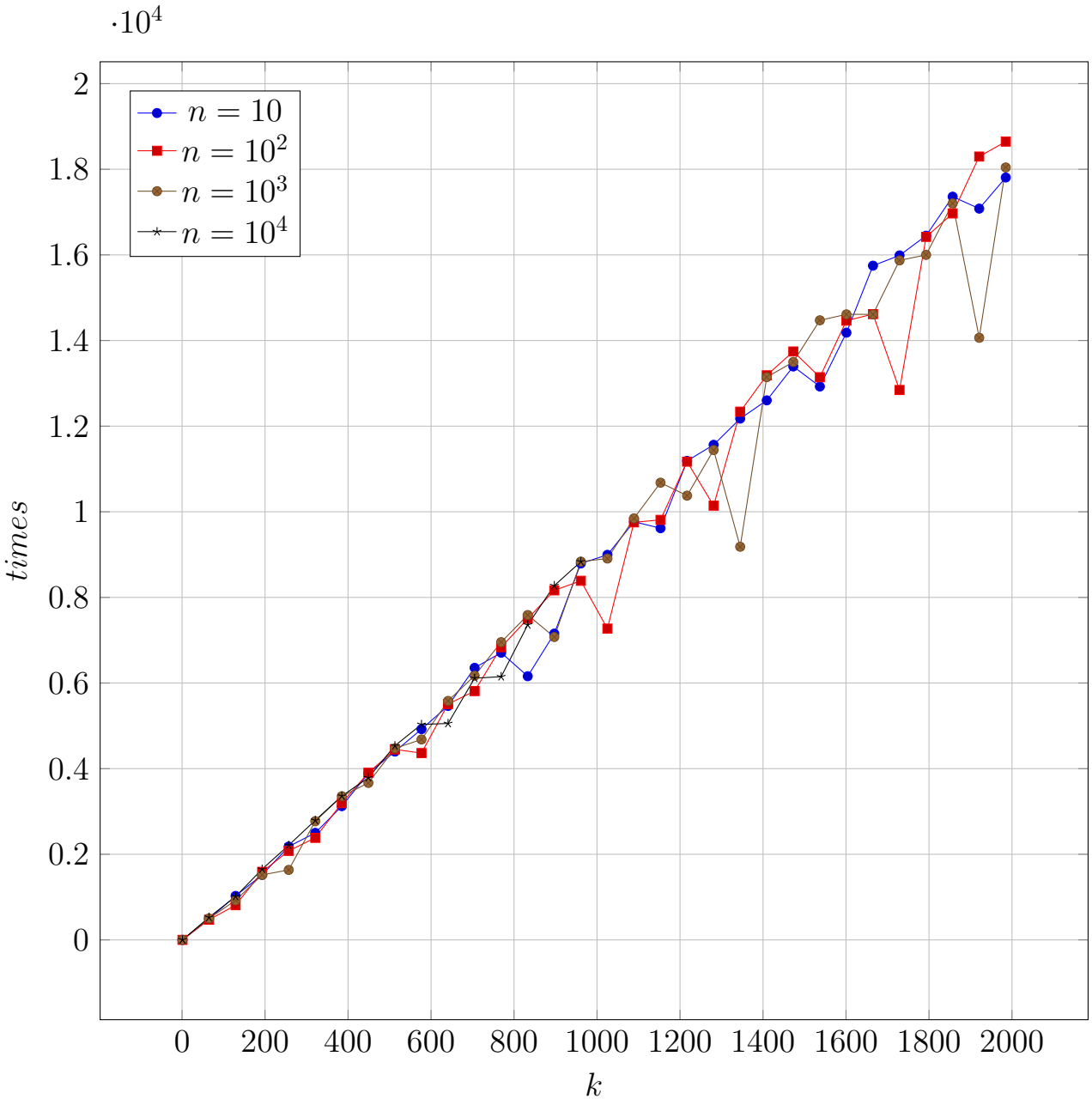
Также все методы на одной картинке.

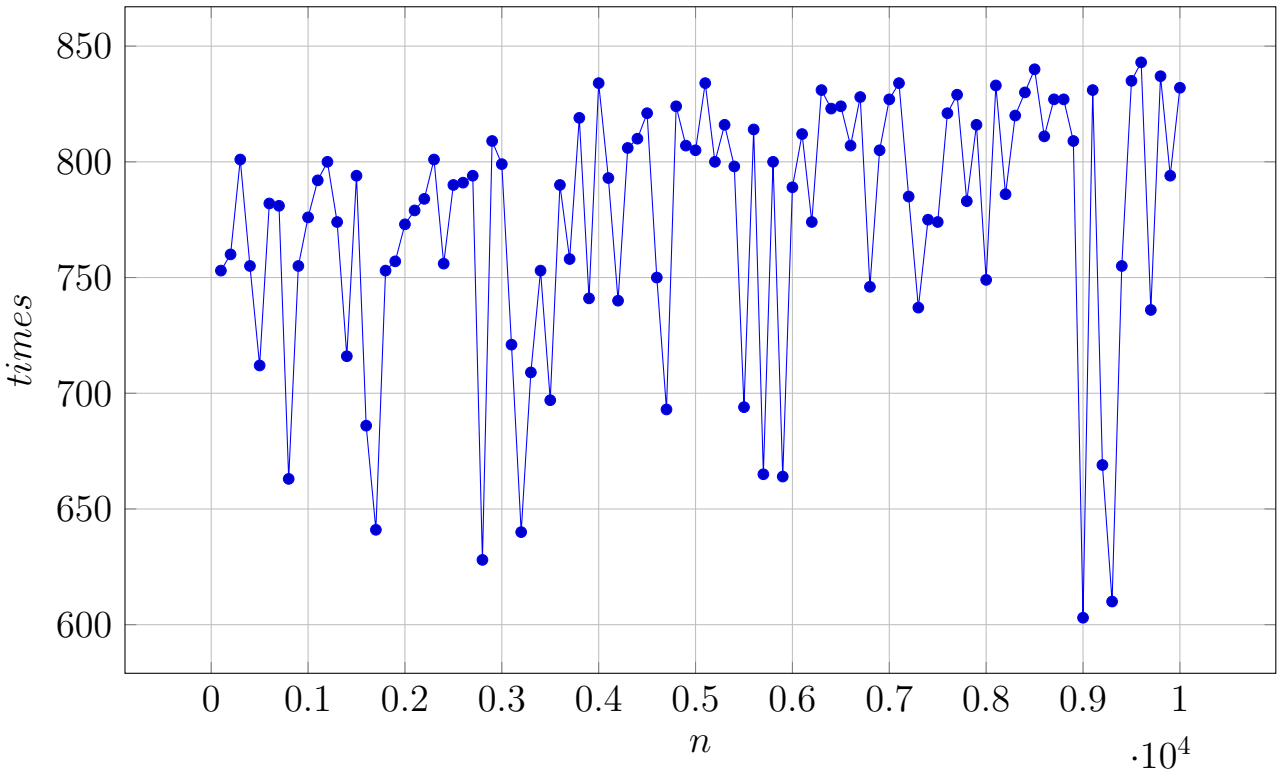


1.4. Метод градиентного спуска

Заметим, что в методе градиентного спуска константа линейной скорости сходимости $q = 2/(l + L)$ не зависит от размерности пространства n, а только от собственных чисел матрицы A квадратичной формы, а следовательно для всех размерностей должны получится схожие результаты, что мы как раз таки видим на графике ниже. Но есть несколько минимумов, которые выбиваются из общей массы. Скорее всего это из-за того, что сгенерированная точка попала в многомерный овраг и из-за этого не происходило сильных биений. По сгенерированным тестам это сложно понять.

На втором графике также видно, что размерность мало на что вилияет, нет явной закономерности.




850
800
750
700
650
600
times
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
n
·10^4

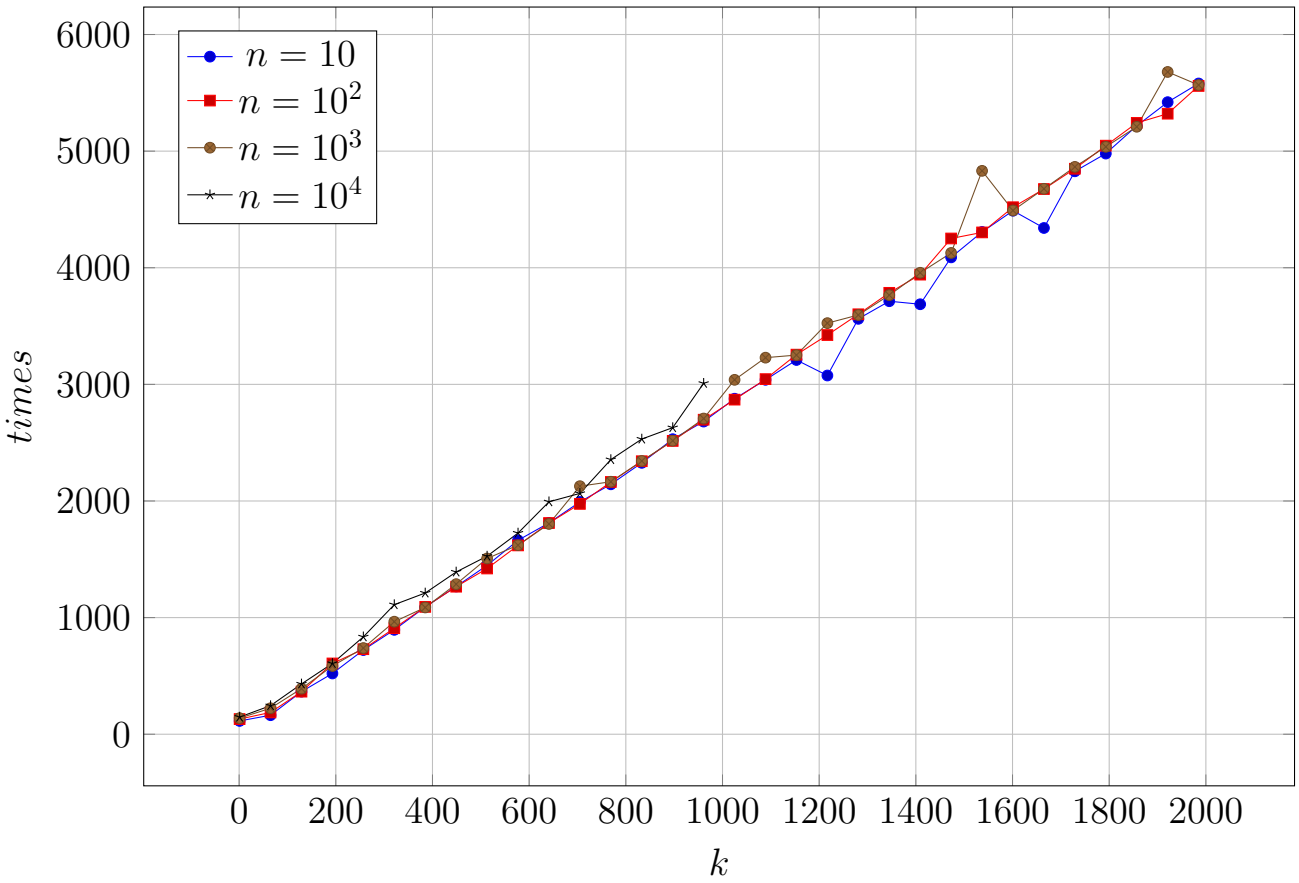
1.5. Метод наискорейшего спуска

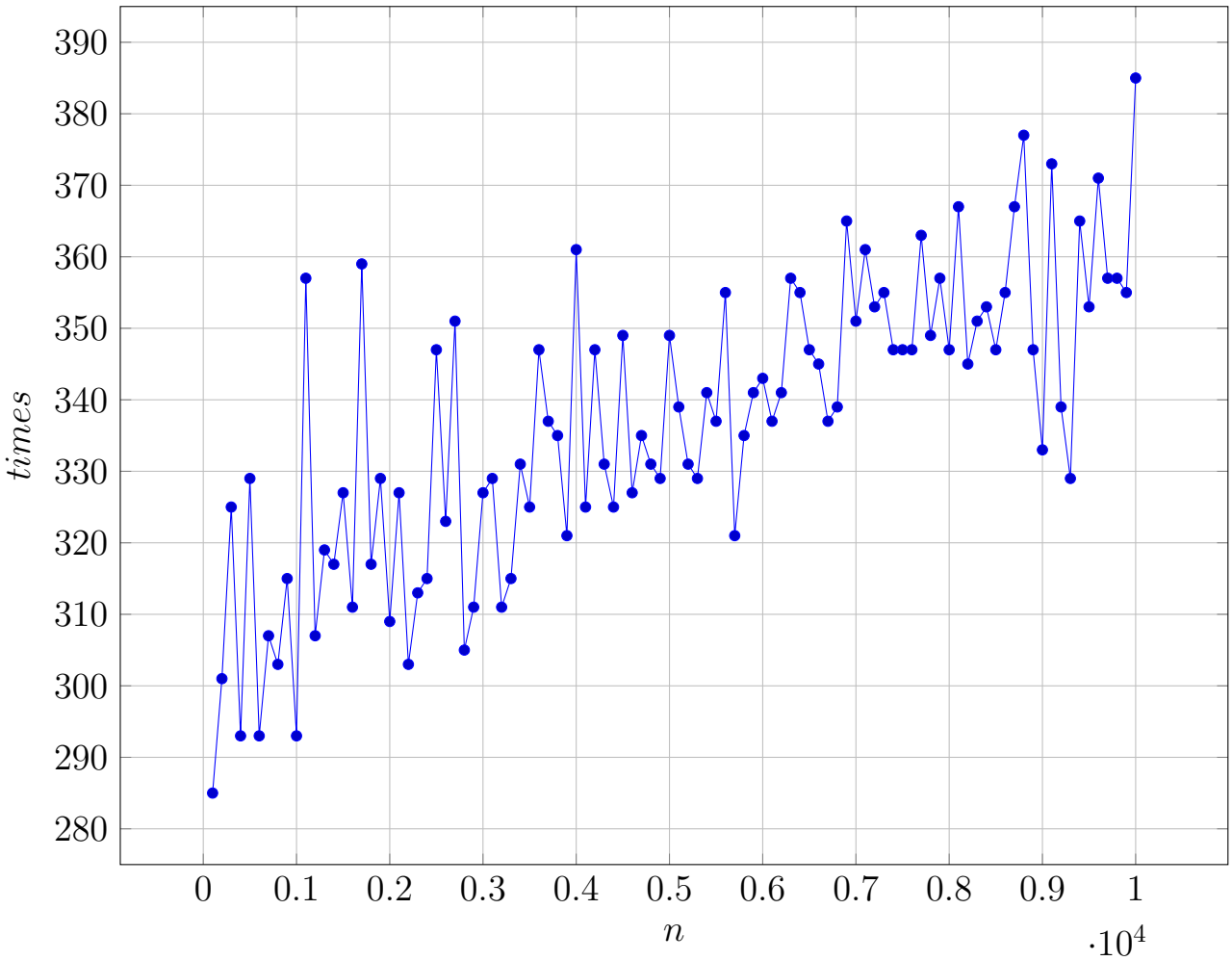
В этом методе получилось практически также как и в предыдущем. Размерность влияет на число итераций, но не значительно. Скорее всего это связано с тем, что константа суперлинейной сходимости в зависимости от размерности стремится к нулю, и поэтому для малых размерностей она стремится быстрее, чем для больших.

Промежуток одномерной оптимизации мырешили взять $(0, 0.1)$, так как мы так и не поняли какой именно нужно брать, но мы провели несколько экспериментов подбирая правое значения и увидели, что в очень редких случаях искомый минимум находится правее 0.1. Также из-за этого в одном из примеров выше получилось так, что вектора не касались линий уровня.

Для одномерной оптимизации мы использовали метод Брента, как самый быстрый метод.

Точность для одномерной оптимизации решили взять $epsion^2$, где $epsion = 0.01$ точность, которая у нас была задана для всех методов многомерной оптимизации. Решили так сделать, так как если передавать $epsilon$ без изменений, то одномерная оптимизация зачастую даже не запускалась или делала всего одно вычисление функции, так как α^* было меньше, чем 0.01.

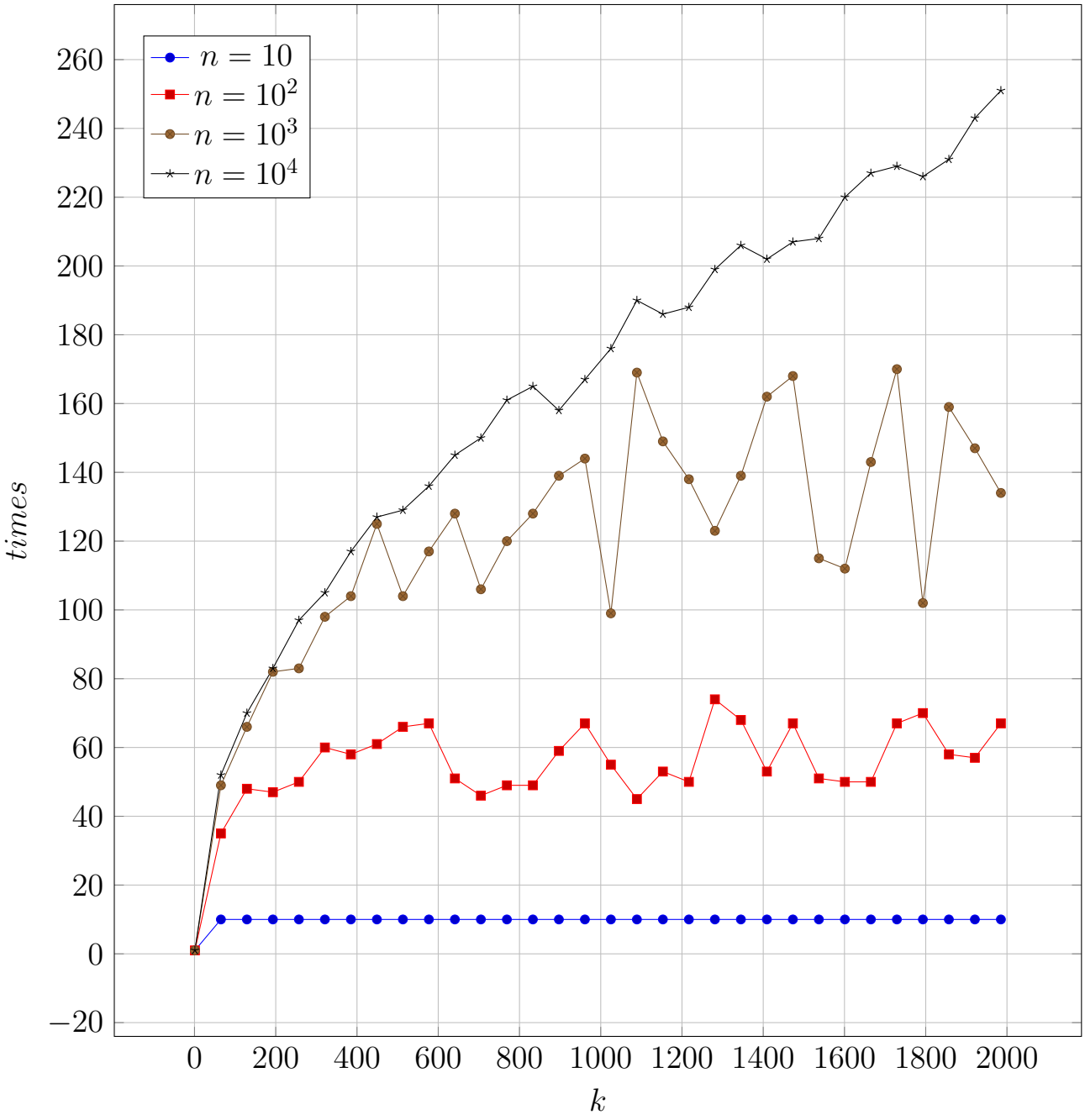


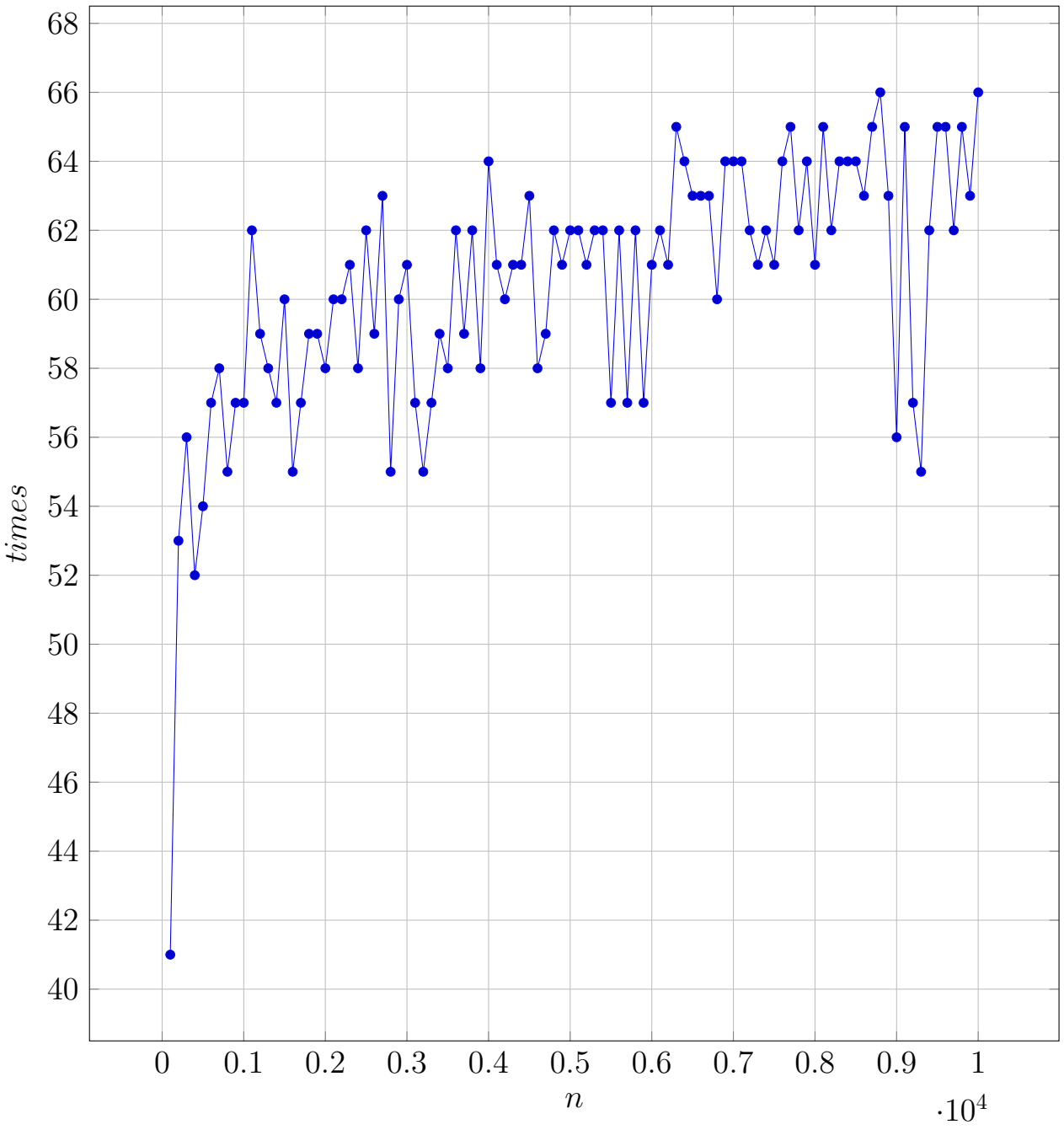

390
380
370
360
350
340
330
320
310
300
290
280
times
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
n
·10^4

1.6. Метод сопряженных градиентов

Для этого метода решили не ограничиваться числом обусловленности $k = 1000$ для размерности $n = 10000$, так как число итераций у этого метода значительно меньше, чем у остальных.

Этот метод не может иметь число итераций больше, чем размерность пространства n, так как попарно ортогональных векторов в пространстве размерности n есть ровно n, а в этом методе мы как раз так выбираем текущий вектор таким образом, что он ортогонален всем предыдущим. Мы как раз и наблюдаем такую тенденцию на наших графиках. Также заметим, что число обусловленности вляет на число итераций только до некоторого момента, а после это число просто колебается вокруг какого-то одного значения.




68
66
64
62
60
58
56
54
52
50
48
46
44
42
40
times
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
n
·10^4

1.7. Код реализованных методов

1.7.1. Метод градиентного спуска

QuadraticFunctionType – это либо обычная квадратичная функция, либо функция у которой основная матрица имеет диагональный вид.

Возвращаем только число итераций, так как вся информация о вычислении функции остается внутри класса QuadraticFunctionType.

```
size_t gradient_descent(QuadraticFunctionType &function,
                        Vector &x0, T alpha) const {
    Vector x = x0;
    T f_x = function.calc(x);
    size_t cnt = 0;
    while (true) {
        Vector gradient = function.gradient(x);
        T norma_gradient = gradient.norma();
        if (norma_gradient < epsilon) {
            break;
        }
        cnt++;
        while (true) {
            Vector gradient_prod_alpha = gradient * alpha;
            Vector y = x - gradient_prod_alpha;
            T f_y = function.calc(y);
            if (f_y < f_x) {
                x = y;
                f_x = f_y;
                break;
            }
            alpha /= 2;
        }
    }
    return cnt;
}
```

1.7.2. Метод наискорейшего спуска

```
size_t steepest_descent(QuadraticFunctionType &function,
                        Vector x0) const {
    // Инициализирую методы одномерной оптимизации
    search_methods searchMethods(epsilon * epsilon);
    size_t cnt = 0;
    while (true) {
        Vector gradient = function.gradient(x0);
        if (gradient.norma() < epsilon) {
            break;
        }
        cnt++;

        // Лямбдафункция- для одномерной оптимизации
        std::function<T(T)> F = [&](T alpha) {
            Vector grad_alpha = gradient * alpha;
            Vector calc_vec = x0 - grad_alpha;
            return function.calc_without_history(calc_vec);
        };

        range rang(0, 0.1L);
        information_search informationSearch = searchMethods.
   combined_brent(F, rang);
        T alpha_min = informationSearch.point;
        Vector gradient_alpha_min = gradient * alpha_min;
        function.calc(x0);
        x0 = x0 - gradient_alpha_min;
    }
    return cnt;
}
```

1.7.3. Метод сопряженных градиентов

```
size_t conjugate_gradient(QuadraticFunctionType &function, Vector &x0)
    const {
    std::vector<Vector> p;
    Vector gradient0 = function.gradient(x0);
    p.emplace_back(gradient0 * (-1.0L));

    std::vector<Vector> x;
    x.emplace_back(x0);
    function.calc(x.back());

    Vector last_A_pk(x0.size());
    size_t cnt = 0;
    while (true) {
        if (gradient0.norma() < epsilon) {
            break;
        }
        cnt++;
        Vector A_pk = function.hessian() * p.back();
        T A_pk_prod_pk = A_pk * p.back();
        T grad_norma_2 = gradient0 * gradient0;
        T alpha_k = grad_norma_2 / A_pk_prod_pk;

        Vector alpha_pk = p.back() * alpha_k;
        x.emplace_back(x.back() + alpha_pk);

        Данное// вычисление функции не влияет на ход алгоритма.
        Это// нужно только для статики.
        function.calc(x.back());

        Vector alpha_k_A_pk = A_pk * alpha_k;
        Vector gradient1 = gradient0 + alpha_k_A_pk;

        T bk = (gradient1 * gradient1) / (gradient0 * gradient0);

        Vector bk_pk = p.back() * bk;
        Vector antigradient = gradient1 * (-1.0L);
        p.emplace_back(antigradient + bk_pk);

        gradient0 = gradient1;
    }
    return cnt;
}
```