Национальный исследовательский университет ИТМО
Факультет информационных технологий и программирования
Прикладная матеметика и информатика

Методы оптимизации
Отчет по лабораторной работе №1

Работу
выполняли:
Кольченко Антон M32371
Гайнанов Ильдар M32371
Муфтиев Руслан M32331

Предподаватель:
Казанков В.К.

Санкт-Петербург
2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Постановка задачи:

1. Реализация градиентного спуска с постоянным шагом (learning rate).
2. Реализация метода одномерного поиска - метод дихотомии и градиентный спуск на его основе.
3. Анализ траектории градиентного спуска на примере различных квадратичных функций.
4. Для каждой функции:
   (a) исследовать сходимость градиентного спуска с постоянным шагом и сравнить полученные результаты для выбранных функций;
   (b) сравнить эффективность градиентного спуска с использованием одномерного поиска с точки зрения количества вычислений минимизируемой функции и ее градиентов;
   (c) исследовать работу методов в зависимости от выбора начальной точки;
   (d) исследовать влияние нормализации (scaling) на сходимость на примере масштабирования осей плохо обусловленной функции;
   (e) нарисовать графики с линиями уровня и траекториямиметодов для каждого случая;
5. Реализовать генератор случайных квадратичных функций n переменных с числом обусловленности k.
6. Исследовать зависимость числа итераций $T(n, k)$, необходимых градиентному спуску для сходимости в зависимости от размерности пространства $2 \leqslant n \leqslant 10^3$ и числа обусловленности оптимизируемой функции $1 \leqslant k \leqslant 10^3$
7. Проведение множественных экспериментов и для полученния среднего значения числа итераций.
8. Реализовать одномерный поиск с учетом условий Вольфе и исследовать его эффективность, а также сравнить полученные результаты с реализованными ранее методами.

1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

В этом разделе будут описаны все вычислительные схемы требующихся методов.

1.1 Градиентный спуск с постоянным шагом (learning rate)

Принцип работы:

Суть метода градиентного спуска заключается в том, чтобы идти в направлении скорейшего спуска. Таким образом, за достаточное количество шагов можно дойти до локального минимума функции, если таковой существует.

Вход: функция $f : \mathbb{R}^n \to \mathbb{R}$, стартовая точка $x = (x_1,x_2,...,x_n)$, точность ε, размер шага λ

Выход: найденная точка локального минимума

Алгоритм:

1. $x^{[k+1]} = x^{[k]} - \lambda \nabla f(x^{[k]})$.
2. Повторять шаг 1, пока $|x^{[k+1]} - x^{[k]}| > \varepsilon$.

1.2 Метод дихотомии

Принцип работы:

Вход: функция $f : \mathbb{R} \to \mathbb{R}$, стартовая точка x, точность ε, отрезок, на котором ищется минимум $[a,b]$

Выход: найденная точка локального минимума

Алгоритм(для поиска минимума):

1. На каждом шаге процесса поиска делим отрезок $[a,b]$ пополам, $x = \frac{a+b}{2}$ - координата середины отрезка $[a,b]$.
2. $x_1 := \frac{a+b}{2} + \delta, x_2 := \frac{a+b}{2} - \delta, \delta \in (0; \frac{a-b}{2})$
3. Вычисляем значение функции f в x_1 и x_2:
   $F_1 = f(x_1), F_2 = f(x_2)$
4. Сравниваем F_1 и F_2 и выбираем отрезок для дальнейшего рассмотрения:
   - Если $F_1 < F_2$, то отбрасываем отрезок $[x_2,b]$, тогда $b = x_2$, рассматриваем $[a, x_2]$.
   - Иначе рассматриваем $[x_1, b]$.

5. Деление отрезка $[a,b]$ продолжается до тех пор, пока его длина не станет меньше заданной точности ε, т.е. $|b-a| \leqslant \varepsilon$

1.3 Градиентный спуск на основе метода дихотомии

Принцип работы:

жюдюювжд Принцип работы данной модификации абсолютно схож с вычислительной схемой обычного градиентного спуска, кроме одного пункта. Вместо того, чтобы прыгать на опрделенный шаг, для начала мы найдем точку минимума функции на одномерном отрезке $[x^{[k]}, x^{[k]} + \lambda\nabla f(x^{[k]})]$. Этот прием поможет сходиться с ответом в некоторых случаях за меньшее количество итераций, но количество процессорного времени, затраченного алгоритмом, будет выше, чем для обычного градиентного спуска для достижения высокой точности (будет проиллюстрировано в практической части).

Алгоритм:

Вход: функция $f : \mathbb{R}^n \to \mathbb{R}$, стартовая точка x, точность ε, размер шага λ

1. $x^{'[k]} = x^{[k]} - \lambda\nabla f(x^{[k]})$.
2. Найти с помощью метода дихотомии локальный минимум на отрезке $[x^{[k]}, x^{'[k]}]$. Обозначим этот минимум как $x^{[k+1]}$
3. Повторять шаги 1-2, пока $|x^{[k+1]} - x^{[k]}| > \varepsilon$.

1.4 Оптимизация градиентного спуска с помощью условия Вольфе

Принцип работы: У градиентного спуска с постоянным шагом есть недостаток: если алгоритм находится слишком близко к точке минимума, то с постоянным шагом алгоритм может "перескочить"через точку экстремума, что приведет к тому, что будет затрачено больше итераций, чем нужно. Поэтому стоит ввести механизм изменения размера шага. В данном случае мы будем использовать критерий Вольфе.

Определение критерия Вольфе:

p - направление, в котором мы хотим изменить x, в случае градиентного спуска - антиградиент функции в точке x

$f(x + \alpha p) \leqslant f(x) + c_1\alpha\nabla f^T p$

$|\nabla f(x + \alpha p)^T p| \geqslant |c_2\nabla f^T p|$

$0 < c_1 < c_2 < 1$, c_1 близко к 0, c_2 близко к 1

Вход: функция $f : \mathbb{R}^n \to \mathbb{R}$, стартовая точка x, точность ε, начальный шаг α_0
Выход: найденная точка локального минимума
Алгоритм:

1. $\alpha = \alpha_0$
2. Уменьшаем α до того момента, пока критерий Вольфе не начнет исполняться.
3. $x^{[k+1]} = x^{[k]} - \alpha \nabla f(x^{[k]})$.
4. Повторять шаги, пока $|x^{[k+1]} - x^{[k]}| > \varepsilon$.

2 ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1 Реализация градиентного спуска с постоянным шагом (learning rate)

Сейчас и в последующем будем считать, что $\varepsilon = 10^{-6}$, в крайних случаях, если значение изменится, об этом будет сказано

```
import numpy as np
import scipy
import matplotlib.pyplot as plt

eps = 1e-6

def grad(f,x):
    h = 1e-6
    l = f(x[:,np.newaxis] + h * np.eye(x.size))
    r = f(x[:,np.newaxis] - h * np.eye(x.size));
    return (l - r)/(2*h)
```

Листинг 2.1 — Функция градиента

```
def gradient_descent(f,x,lr,lim=2000):
    points = []
    points.append(x)
    while f(x) - f(x - lr * (g := grad(f,x))) > eps:
        x = x - lr * g
        points.append(x)
        if len(points) > lim:
          return np.array([])
    return np.array(points)
```

Листинг 2.2 — Градиентный спуск

```
def f(x):
    return (x[0] - 2) ** 2 + (x[1] + 3) ** 2
t = np.linspace(-10,10,100)
X, Y = np.meshgrid(t,t)
ax = plt.figure().add_subplot(projection='3d')
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_zlabel("f(x,y)")
ax.plot_surface(X, Y, f(np.stack((X, Y))))
```

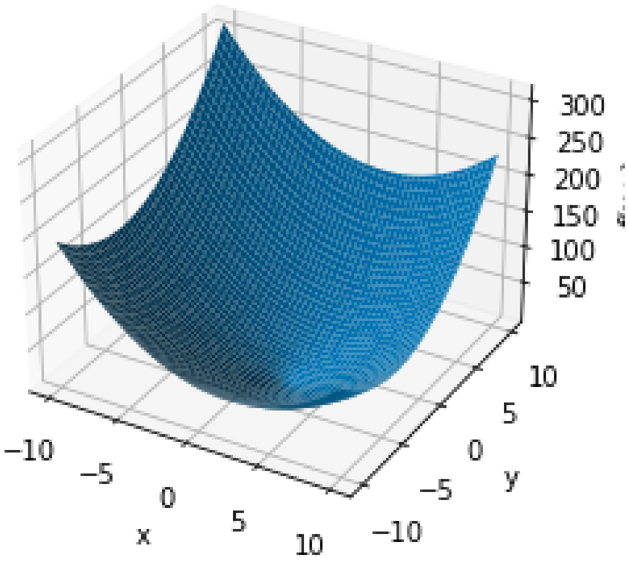

Рисунок 1 — Функция и ее график

```
lr = 0.1
x = np.array([-1,1])
points = gradient_descent(f,x,lr)
print("x = ",points[-1])
print("f(x) = ",f(points[-1]))
plt.plot(points[:, 0], points[:, 1], 'o-')
plt.contour(X, Y, f([X, Y]), levels=sorted([f(p) for p in points]))
```

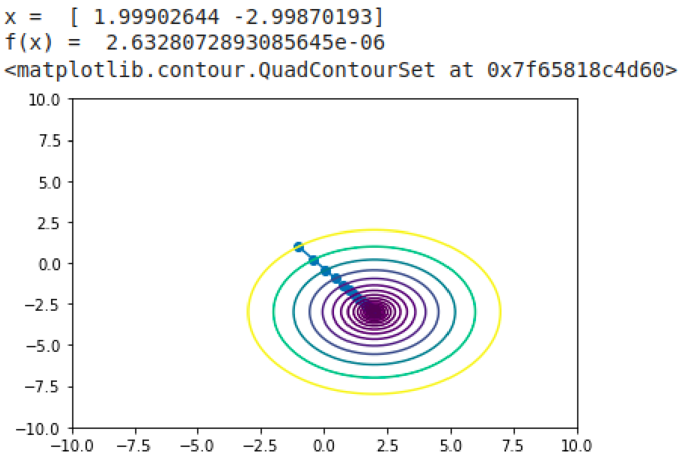

Рисунок 2 — Графическая интерпретация итераций алгоритма

2.2 Реализация градиентного спуска на основе одномерного поиска

2.2.1 Метод дихотомии

```
def f(x):
    return (x / 16 - 1) ** 2 * (2 * x ** 3 - 15 * x) - x ** 2

alpha = 12
beta = 19
t = np.linspace(alpha,beta,10000)
y = (0.2 * t - 1) * (0.2 * t - 6) ** 3 + 70
plt.plot(t,f(t))
plt.xlabel("x")
plt.ylabel("f(x)")
plt.show()
```

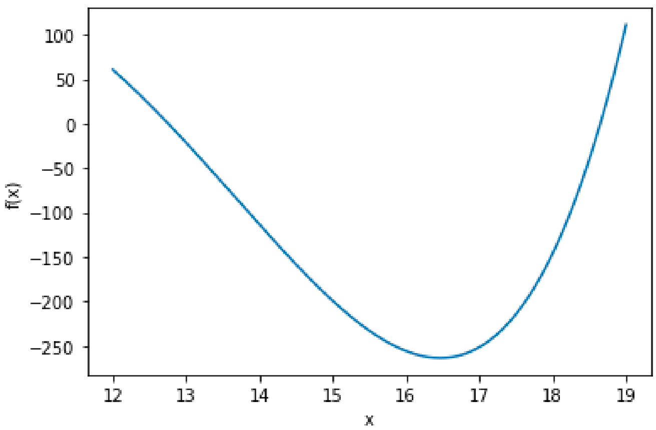

Рисунок 3 — Берем произвольную функцию одного переменного

```
x = [alpha,beta]

points = []
points.append(x[::])

# Dichotomy algorithm
while (x[1] - x[0]) > eps:
    delta = eps / 2
    x1 = (x[0] + x[1]) / 2 - delta / 2
    x2 = (x[0] + x[1]) / 2 + delta / 2
    if f(x1) < f(x2):
        x[1] = x2
    else:
        x[0] = x1
```

```
    points.append(x[::])

points = np.array(points)

ax1 = plt.subplot(1, 2, 1)
ax2 = plt.subplot(1, 2, 2)
ax1.set_xlabel("x")
ax2.set_xlabel("x")
ax1.set_ylabel("f(x)")
ax2.set_ylabel("f(x)")
pos1 = ax2.get_position()
ax2.set_position([pos1.x0 + 0.1,
                  pos1.y0,
                  pos1.width,
                  pos1.height])
ax1.scatter((points[::,0] + points[::,1]) / 2,f((points[::,0] + points[::,1]) /
    2,), color='red')
ax1.plot(t,f(t))
ax2.plot(f((points[::,0] + points[::,1]) / 2))
```

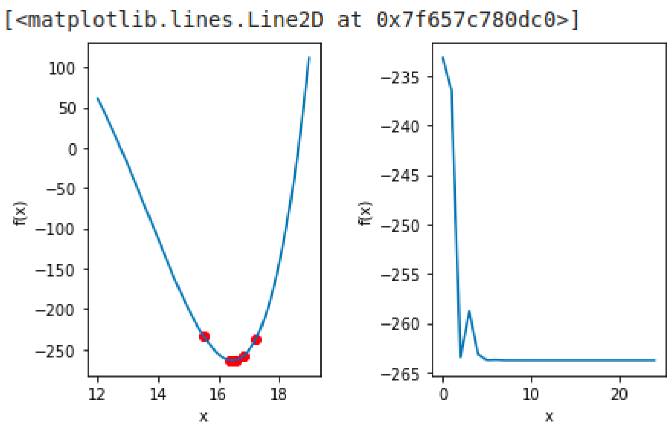

Рисунок 4 — Пример работы дихотомии

2.2.2 Модифицированный градиентный спуск

```python
def dichotomy(f,a,b):
    x = np.array([a,b])
    while dist(x[0],x[1]) > eps:
        delta = (x[1] - x[0]) / 2
        x1 = (x[0] + x[1]) / 2 - delta / 2
        x2 = (x[0] + x[1]) / 2 + delta / 2
        new_x = np.copy(x)
        if f(x1) < f(x2):
            new_x[1] = x2
        else:
            new_x[0] = x1
        x = new_x

    return (x[0] + x[1]) / 2

def gradient_descent_with_dichotomy(f,x,lr,lim=2000):
    points = []
    points.append(x)
    while f(x) - f(x - lr * (g := grad(f,x))) > eps:
        x = dichotomy(f,x,x - lr * g)
        points.append(x)
        if len(points) > lim:
          return np.array([])
    return np.array(points)
```

Функция №1

$f(x,y) = \frac{1}{50}x^2 + 50y^2$

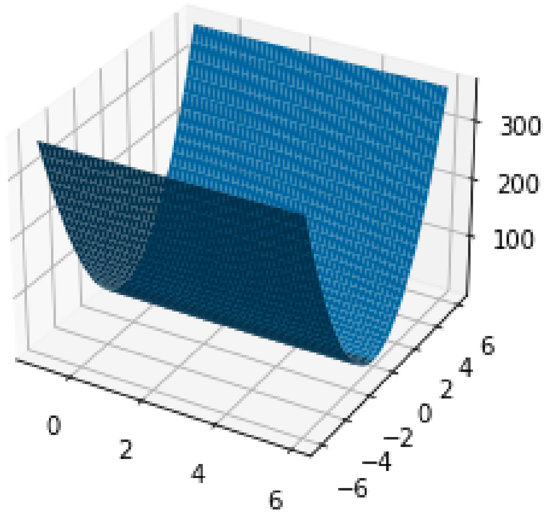

Рисунок 5 — График функции

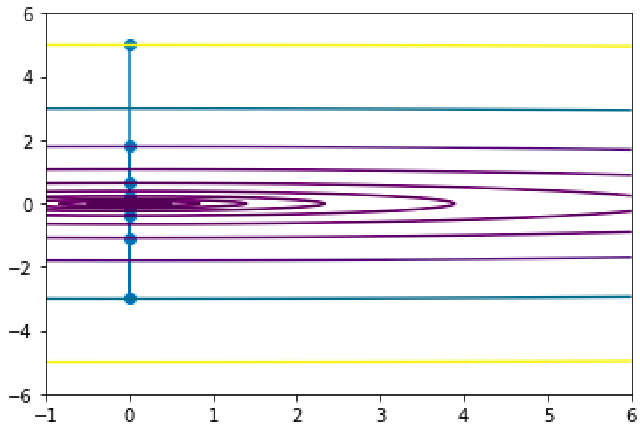

Рисунок 6 — Работа ГС с постоянным шагом

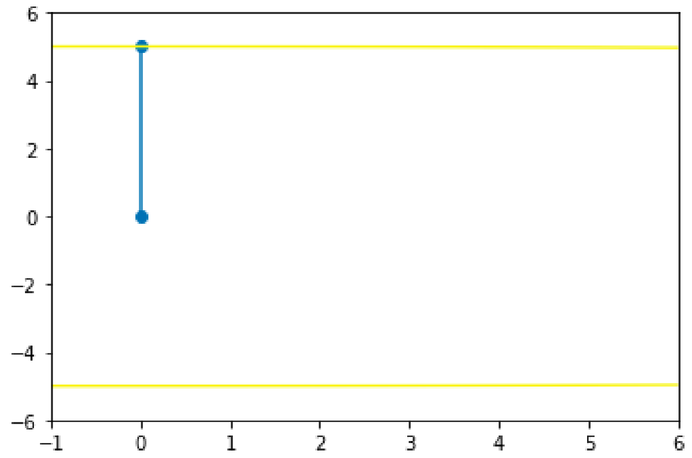

Рисунок 7 — Работа ГС на основе дихотомии

Функция №2

$f(x,y) = \frac{1}{10}x^2 + 10y^2$

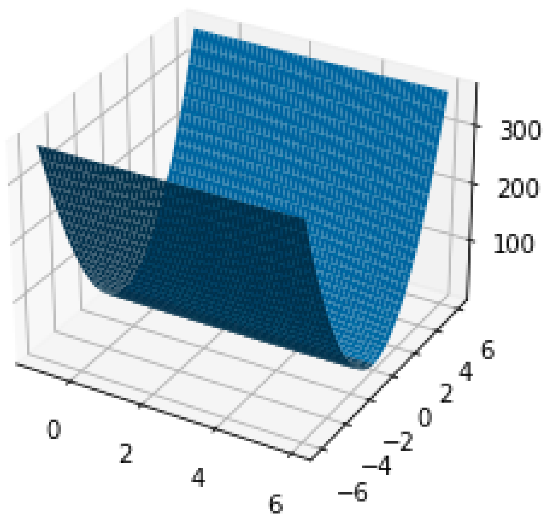

Рисунок 8 — График функции

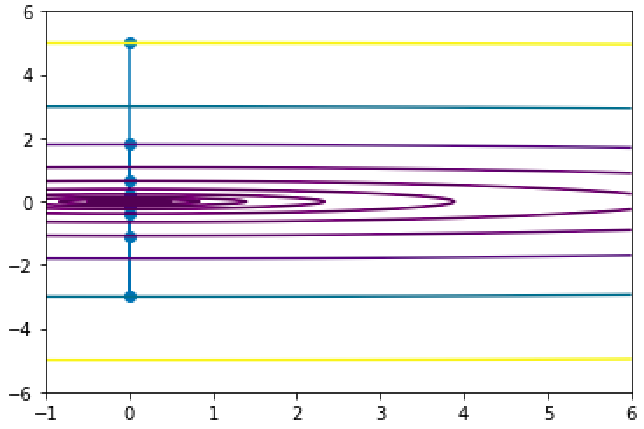

Рисунок 9 — Работа ГС с постоянным шагом

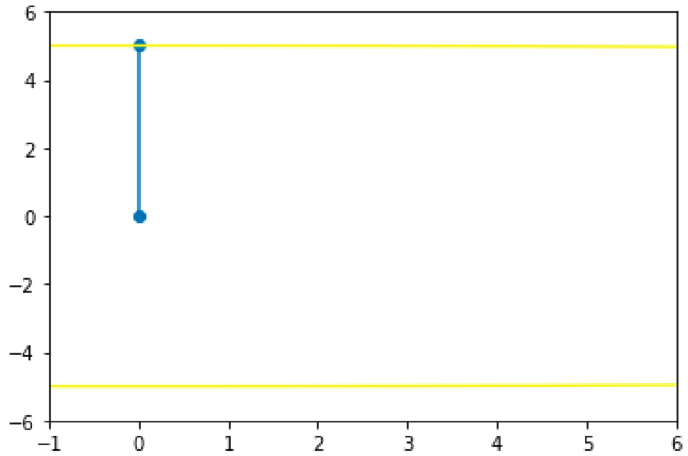

Рисунок 10 — Работа ГС на основе дихотомии

Функция №3

$f(x,y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$

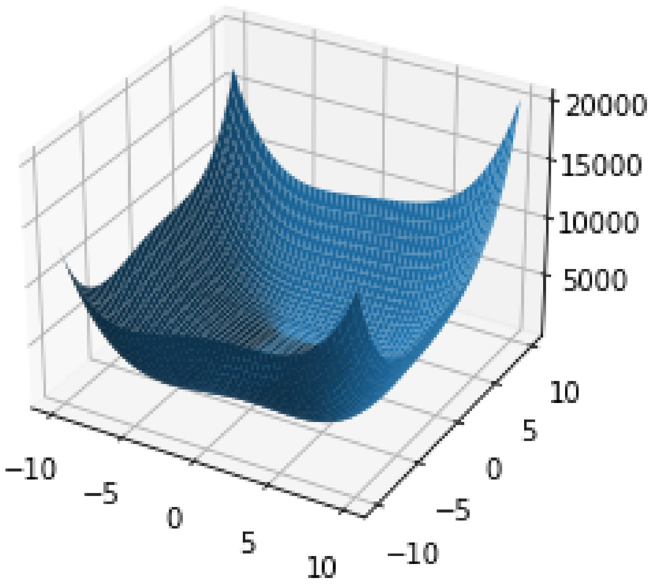

Рисунок 11 — График функции

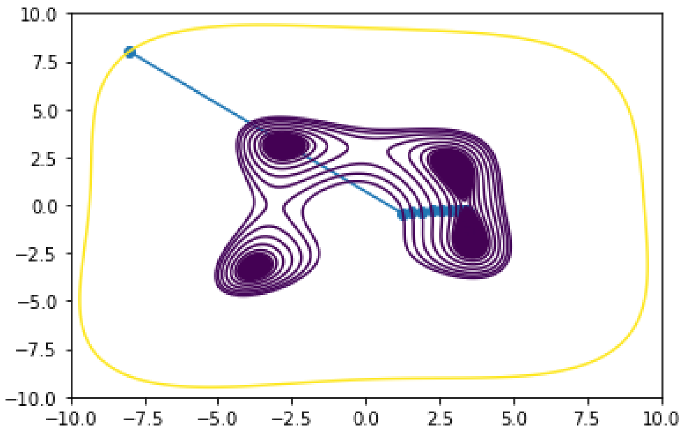

Рисунок 12 — Работа ГС с постоянным шагом

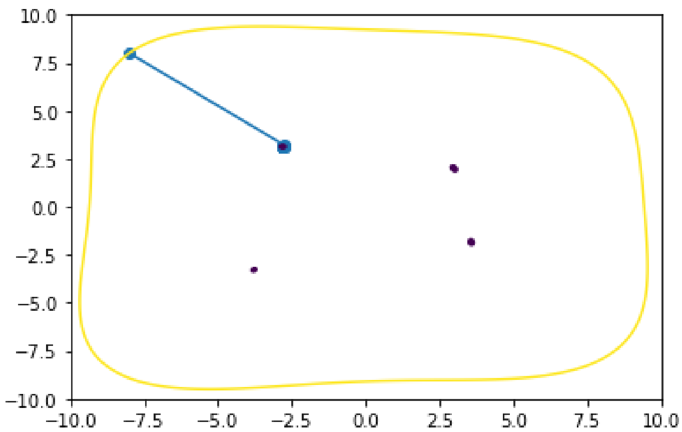

Рисунок 13 — Работа ГС на основе дихотомии

2.3 Дополнительные исследования

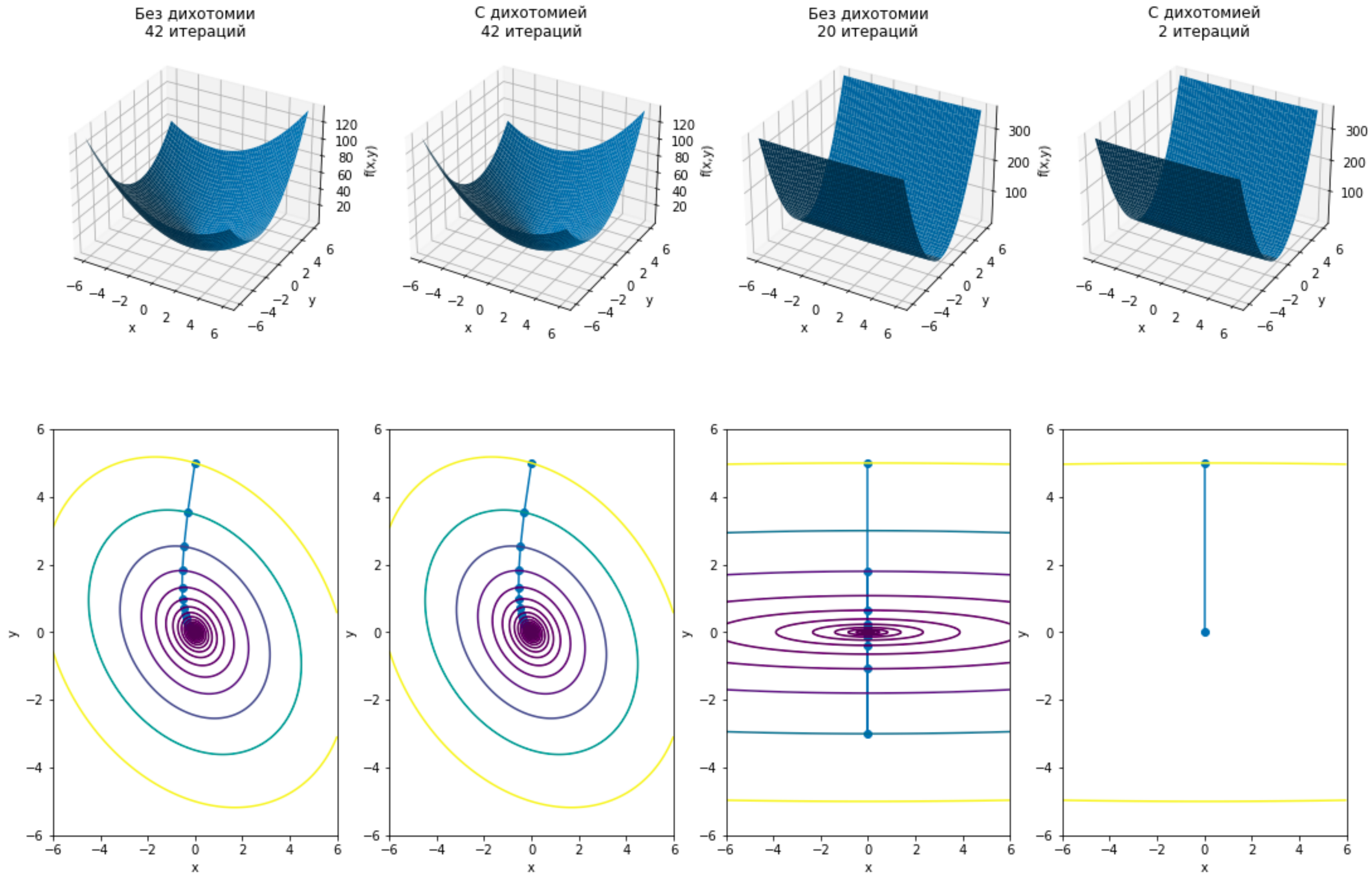

Рисунок 14 — Сравнение количества итераций алгоритмов ГС

Количество вызовов функции и подсчетов градиентов для данных случаев:

1. Вызовы функции: 132
   Подсчеты градиента: 33
2. Вызовы функции: 2230
   Подсчеты градиента: 33
3. Вызовы функции: 80
   Подсчеты градиента: 20
4. Вызовы функции: 120
   Подсчеты градиента: 2

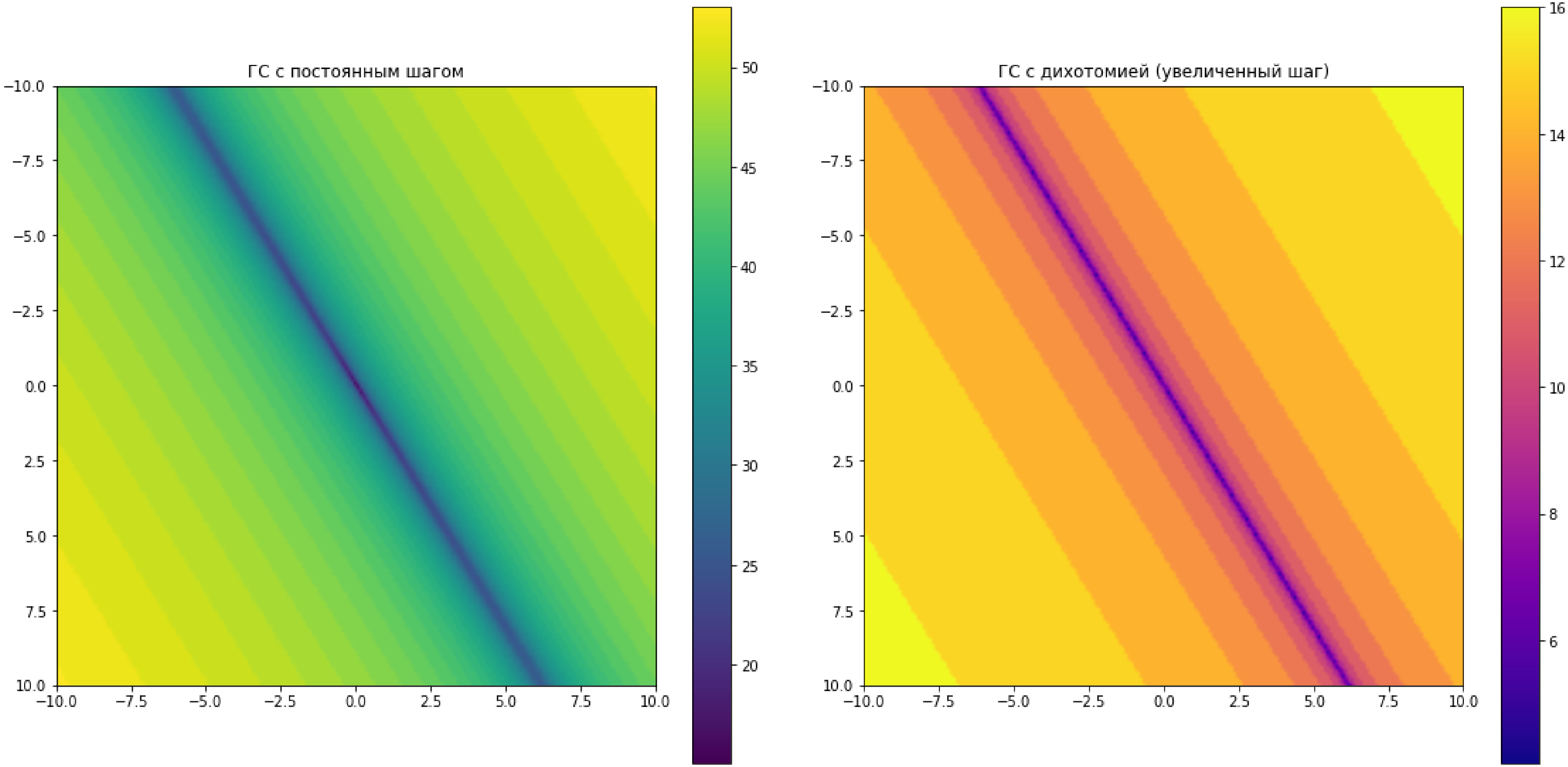

Рисунок 15 — Количество итераций алгоритмов ГС в зависимости от выбора начальной точки

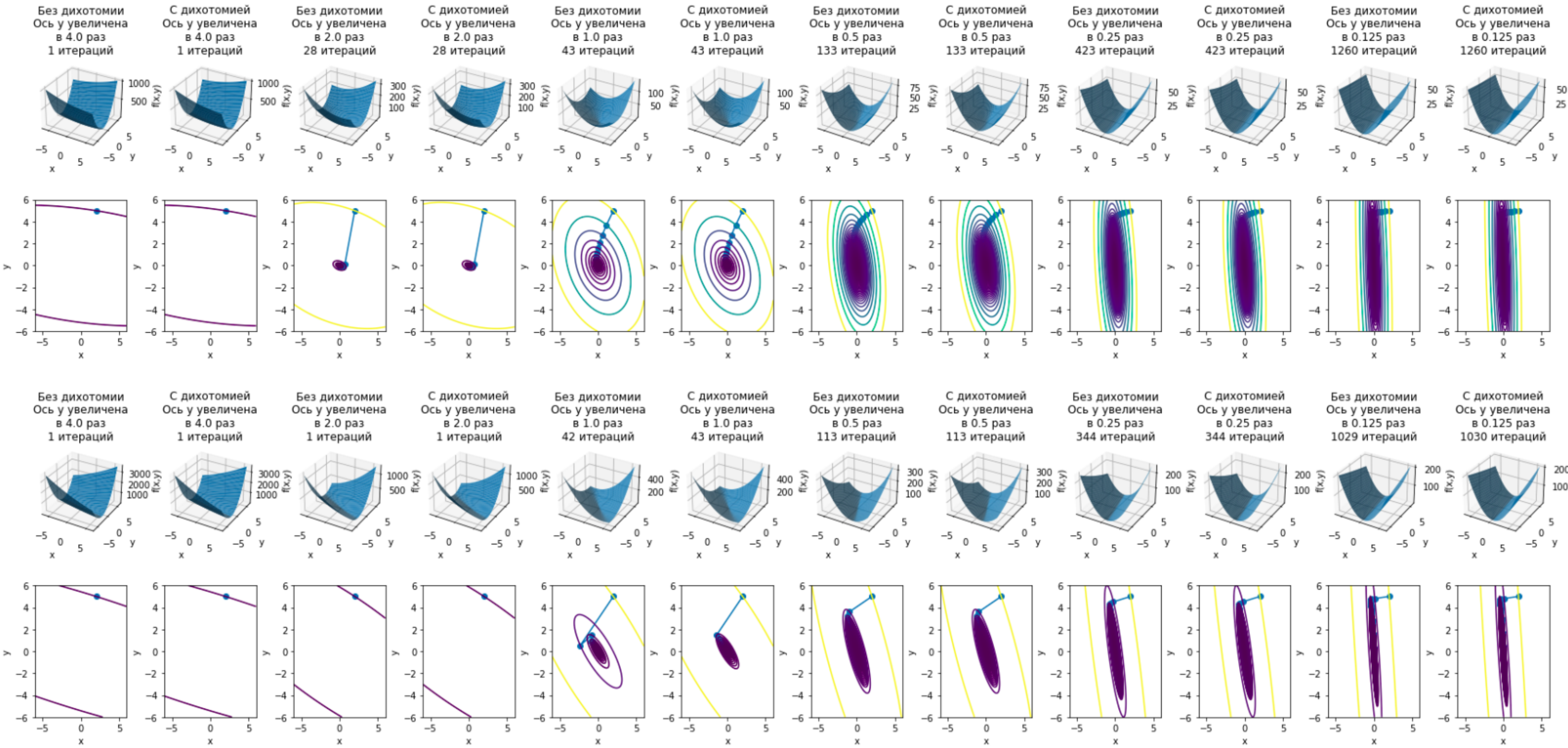

Рисунок 16 — Количество итераций алгоритмов ГС в зависимости от масштабирования осей

2.4 Анализ градиентного спуска и его модификации на основе дихотомии

Эффективность градиентного спуска сильно зависит от выбора шага. Если шаг будет слишком большой, то алгоритм градиентного спуска может "перескочить"точку минимума (См. графики функций 1 и 2), что приведет к лишним итерациям алгоритма, а в крайне плохих плохих случаях к тому, что градиентный спуск не будет сходиться.

Модификация градиентного спуска с использованием дихотомии позволяет алгоритму сходиться в таких случаях, причем значительно быстрее, чем сходится градиентный спуск с постоянным шагом. Однако если шаг слишком мал, то результат модифицированного алгоритма не будет отличаться от обычного градиентного спуска ничем, кроме времени исполнения, которое у модифицированного алгоритма будет значительно больше (см. дополнительное исследование 1).

На примере функции №3 мы можем заметить, что обычный и модифицированый градиентный спуск могут вернуть различные точки минимума. (В данном случае это не является проблемой, так как приведенная функция имеет 4 точки минимума с равными значениями функции в них.)

Если шаг слишком маленький, то это так же может приводить к лишним итерациям алгоритма. В дополнительном исследовании 2 в каждом тесте не изменялись никакие переменные, кроме точки запуска алгоритма, которая отдалялась от точки минимума, что приводило к увеличению времени исполнения алгоритмов.

Ускорить выполнение можно с помощью нормализации осей функции. По результатам дополнительного исследования можно увидеть, что существуют случаи, когда методы ГС не сходятся или сходятся за очень большое количество итераций, и это можно исправить с помощью нормализации.

2.5 Реализация генератора квадратичных функций n переменных с числом обусловленности k

```
# Quadratic function generation
def gen_f(n,k):
    m = np.random.rand(n, n) * 2
    Q, _ = np.linalg.qr(m)
    D = np.diag(np.array([k] + [1] * (n - 1)))

    result = Q @ D @ np.linalg.inv(Q)
    def f_impl(x):
        return x.T @ result @ x
```

```

    def f(x):
        return np.apply_along_axis(f_impl, 0, x)

    return f

def gen_f_poly(n,k):
    m = np.random.rand(n, n) * 2
    Q, _ = np.linalg.qr(m)
    D = np.diag(np.array([k] + [1] * (n - 1)))

    result = Q @ D @ np.linalg.inv(Q)

    parts = []
    for i in range(n):
      for j in range(n):
        x = f'x{i}^2' if i == j else f'x{i}*x{j}'
        parts.append(f'{result[i][j]}*{x}')

    return ' + '.join(parts)

gen_f_poly(2, 1000)
```

```
'46.25938084690356*x0^2 + 207.76359139996495*x0*x1 + 207.76359139996498*x1*x0 + 954.7406191530964*x1^2'
```

Рисунок 17 — Генератор функций, вывод сгенерированной функции в виде полинома

```
k = 1000
n = 2
f = gen_f(n, k)

x = np.linspace(-1,1,100)
y = np.linspace(-1,1,100)
X, Y = np.meshgrid(x,y)
ax = plt.subplot(111,projection='3d')
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.plot_surface(X,Y,f(np.stack((X,Y))))
```

```
<mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x7f675d17c370>
```

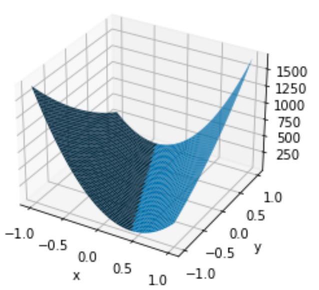

Рисунок 18 — Пример работы генерации функции для $n = 2$ и $k = 1000$

2.6 Исследование зависимости числа итераций, необходимых градиетному спуску для сходимости

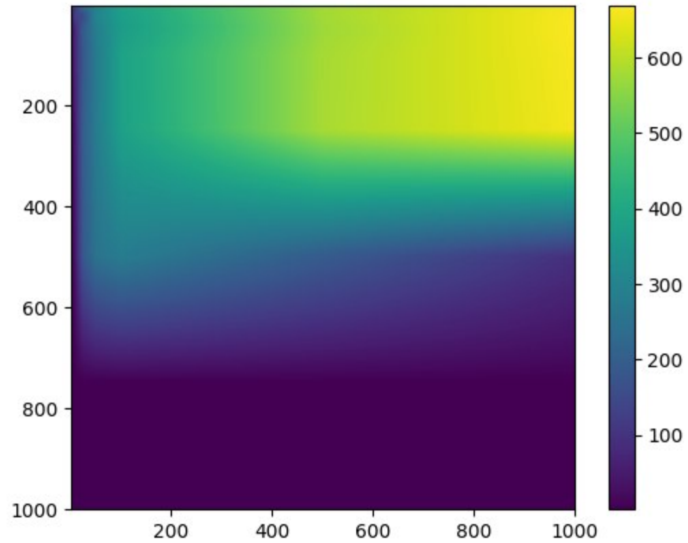

Рисунок 19 — Тепловая карта значений $T(n,k)$ (по горизонтали - n, по вертикали - k)

2.7 Реализация одномерного поиска с условием Вольфе

```
def check_wolfe(f, x, alpha, dir, c1=0.1, c2=0.9):
  gx = grad(f, x)
  cond1 = f(x + alpha*dir) <= f(x) + alpha * c1 * np.dot(dir, gx)
  cond2 = abs(np.dot(dir, grad(f, x + alpha * dir))) <= abs(c2 * np.dot(dir, gx))
  return cond1 and cond2
```

Листинг 2.3 — Проверка условия Вольфе

```
def find_wolfe(f, x, dir):
    m = mk = 1
    start_alpha = 0.5
    for m in range(1, 20):
        alpha = start_alpha ** m
        if check_wolfe(f, x, alpha, dir):
          mk = m
          break
    return start_alpha ** mk

def gradient_descent_wolfe(f,x,lim=2000):
    points = []
    points.append(x)
```

```
    g = grad(f, x)
    if (np.linalg.norm(g) < 1e-6):
      return np.array(points)
    alpha = find_wolfe(f, x, -g)
    while f(x) - f(x - alpha * g) > 1e-6:
        x = x - alpha * g
        points.append(x)
        if len(points) > lim:
          return np.array([])
        g = grad(f, x)
        if (np.linalg.norm(g) < 1e-6):
          return np.array(points)
        alpha = find_wolfe(f, x, -g)
    return np.array(points)
```

Листинг 2.4 — Нахождение коэффициента для условия Вольфе и градиентный спуск на основе условия Вольфе

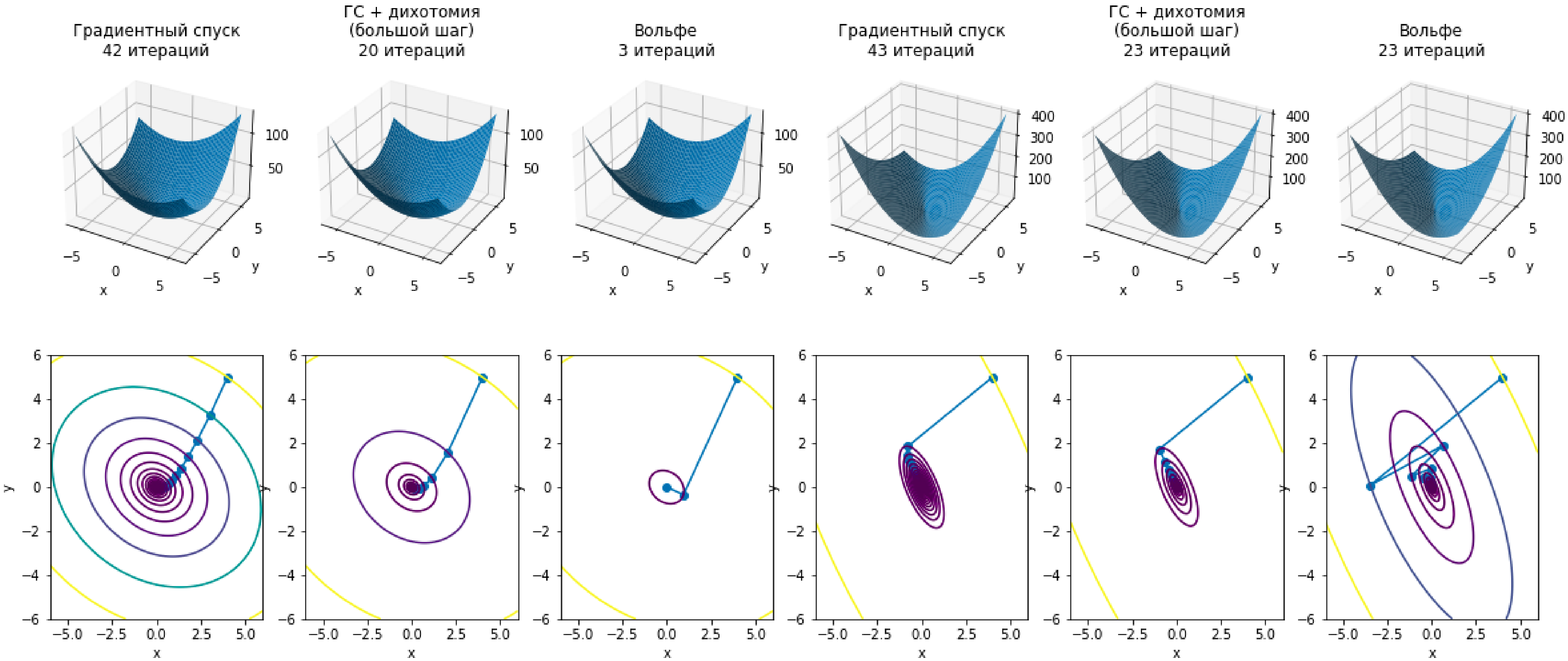

Рисунок 20 — Сравнение методов

3 ВЫВОДЫ

Итак, в данной работе было рассмотрено несколько вариаций реализации метода градиентного спуска для нахождения наиболее оптимального(минимального) значения функции нескольких переменных. Как оказалось, у различных методов имеются свои сильные и слабые стороны. Кроме того, различные модификации алгоритма в некоторых случаях ускоряют процесс поиска, а других, наоборот.

Как оказалось, на сходимость методов градиентного спуска сильно влияет длина шага. Мы рассмотрели метод дихотомии и критерий Вольфе, с помощью которых мы можем подобрать более удачную длину шага. Стандартный градиентный спуск плохо справляется на некоторых функциях. В большинстве случаев градиентный спуск с использованием дихотомии работает за такое же количество итераций, но есть функции, на которых это количество уменьшается в разы. Градиентный спуск же с использованием критерия Вольфе показал себя как достаточно универсальный метод, работающий в большинстве случаев быстрее стандартного градиентного спуска.

Нами так же было рассмотрено влияние различных параметров на скорость сходимости методов градиентного спуска. Мы сделали следующие выводы:

- С ростом числа обусловленности алгоритмы градиентного спуска сходятся быстрее.
- С ростом размерности функции алгоритмы градиентного спуска требуют больше итераций для завершения.