

Команда ГАДы:  
Крылов Андрей m3234  
Фунин Георгий m3234  
Дударев Денис m3235

# Отчет

## Лабораторная работа №1

### Методы нулевого и первого порядка

## Введение

В ходе данной лабораторной работы мы подробно рассмотрели методы нулевого и первого порядка. Мы реализовали градиентный спуск и различные вариации одномерного поиска, а также сравнили результаты с библиотечными реализациями. В лабораторной работе были использованы Scala 3 и Python 3.

## 1 Описание методов

### Градиентный спуск

- Самостоятельно реализованный метод
- Использует хвостовую рекурсию для оптимизации памяти
- Гиперпараметры в виде порога сходимости и максимального количества итераций
- Возможность использовать любой метод выбора следующего шага

Листинг 1: Реализация Градиентного спуска

---

```
1 def gradientDescent(x_0: Point, h: Scheduling): (Point, Int) = {
2
3     functionEvalCount = 0
4     gradientEvalCount = 0
5
6     val logBuffer = ArrayBuffer[String]()
7     if (record_flag) logBuffer.append(x_0.printCords)
8
9     @tailrec
10    def recursion(x_k: Point, k: Int): (Point, Int) = {
11        val gradNorm = findGradient(x_k).N
12        if (gradNorm < eps * 1e-8 || k > 1e5) (x_k, k)
13        else {
14            val next = step(x_k, k, h)
15            if (record_flag) logBuffer.append(next.printCords)
16            recursion(next, k + 1)
17        }
18    }
19 }
```

---

### Стратегии выбора следующего шага

#### 1.1 Фиксированные шаги

- Константный шаг
- Убывающая последовательность

## 1.2 Адаптивные шаги

- Условие Армихо

$c = 0.5$  — параметр для условия достаточного убывания

Ограничение в количестве итераций —  $1e5$

Листинг 2: Реализация правила Армихо

```
1 case class armijoRule(f: Point => Double) extends Scheduling {
2     override def toString(): String = "Armijo Rule"
3
4     override def func = (_: _, x) => {
5         val c: Double = 0.5
6         val gradF: Point = QuadFunc(f).findGradient(x)
7         val N2: Double = math.pow(gradF.N, 2)
8
9         @tailrec
10        def rec(h: Double, iter: Int): Double = {
11            val armijoBool = f(x - gradF * h) <= f(x) - c * h * N2
12            if (iter > 1e5) h
13            else if (armijoBool) h
14            else rec(h * c, iter + 1)
15        }
16        rec(1, 0)
17    }
18 }
```

- Условия Вольффе

$c1 = 0.001$  — параметр для условия достаточного убывания

$c2 = 0.9$  — параметр для условия кривизны

Ограничение в количестве итераций —  $1e5$

Листинг 3: Реализация правила Вольффе

```
1 case class wolfeRule(f: Point => Double) extends Scheduling {
2     override def toString(): String = "Wolfe Rule"
3
4     override def func = (_: _, x) => {
5         val c1: Double = 0.001
6         val c2: Double = 0.9
7         val function: QuadFunc = QuadFunc(f)
8         val gradF: Point = function.findGradient(x)
9         val N2: Double = math.pow(gradF.N, 2)
10
11        @tailrec
12        def rec(h: Double, iter: Int): Double = {
13            val armijoBool = f(x - gradF * h) <= f(x) - c1 * h * N2
14            val wolfeBool = function.findGradient(x - gradF * h).N <= c2 * gradF.N
15            if (iter > 1e5) h
16            else if (!armijoBool) rec(h * 0.5, iter + 1)
17            else if (!wolfeBool) rec(h / c2, iter + 1)
18            else h
19        }
20        rec(1, 0)
21    }
22 }
```

## 2 Графики

Реализуем отображение графиков на Python, который:

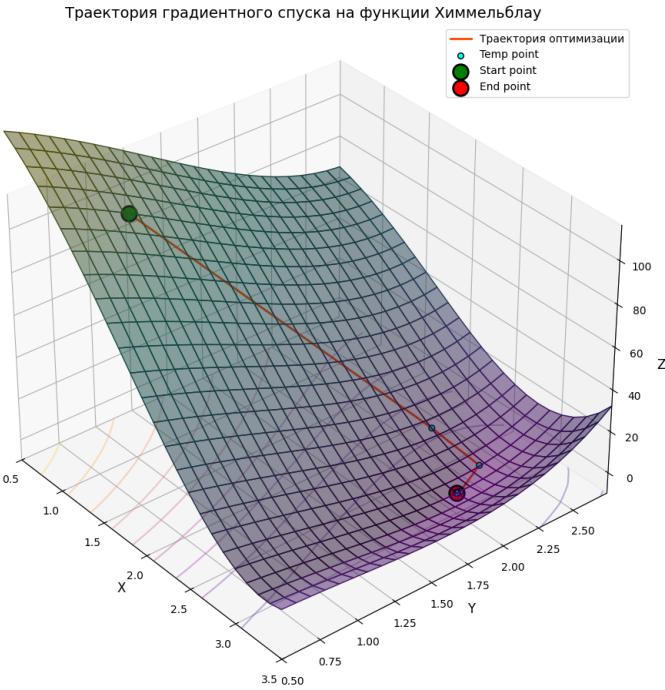
- отображает визуализацию 3D
- отрисовывает траекторию градиентного спуска
- отображает линии уровня функции

## Используемые библиотеки

- numpy — работа с массивами данных
- matplotlib.pyplot — создание 3D-графиков

- `matplotlib.colors.LightSource` — создание освещения для 3D-графиков

Пример полученного графика с точки  $(1, 1)$  на функции Химмельблау



### 3 Описание результатов

Рассмотрим работу методов фиксированного шага, таких как Константный и Убывающая последовательность на функции

$$z = 3x^2 + y^2 - 2xy \text{ в точке } (1, 1)$$

Метод	Параметры	Итерации	в.Функции	в.Градиента	$x$	$y$
Constant $\frac{1}{c}$	$c = 100$	2577	20620	5155	0.00000000000003232837	0.0000000000000780476
Dec. seq. $\frac{1}{k+c}$	$c = 1$	100001	800012	200003	-0.00000010495930779253	-0.00000025339418437001

График Constant

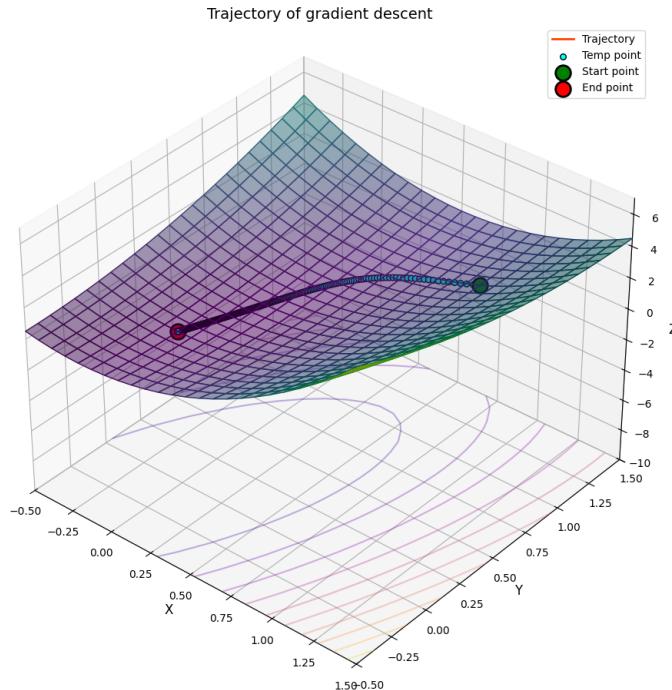
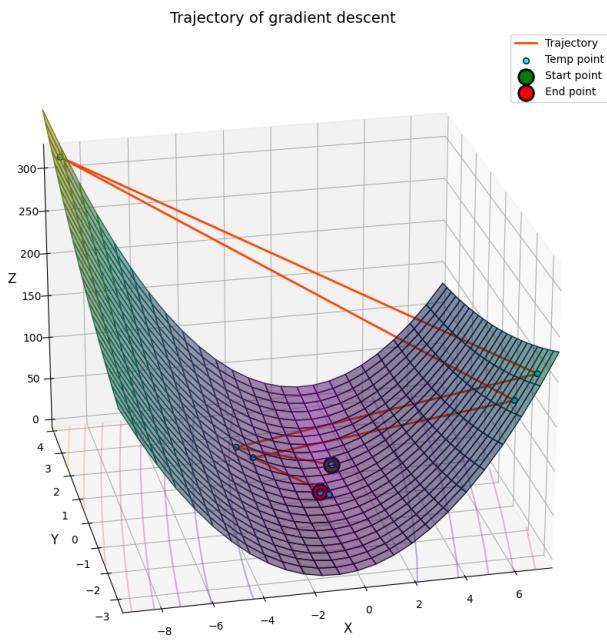


График Decreasing sequence



По полученным результатам видно, что константный метод отработал лучше, чем убывающая последовательность: результат точнее, а количество итераций сравнительно небольшое, в то время как метод убывающей последовательности вышел за ограничение итераций.

Теперь рассмотрим аддитивные методы и сравним их

Метод	Параметры	Итерации	в.Функции	в.Градиента	<i>x</i>	<i>y</i>
Armijo rule	$c = 0.5$	89	1608	268	0.0000000000002842171	0.0000000000005684342
Wolfe rule	$c_1 = 0.001, c_2 = 0.9$	91	2734	547	-0.0000000000000000	0.0000000000002842171

График Armijo rule

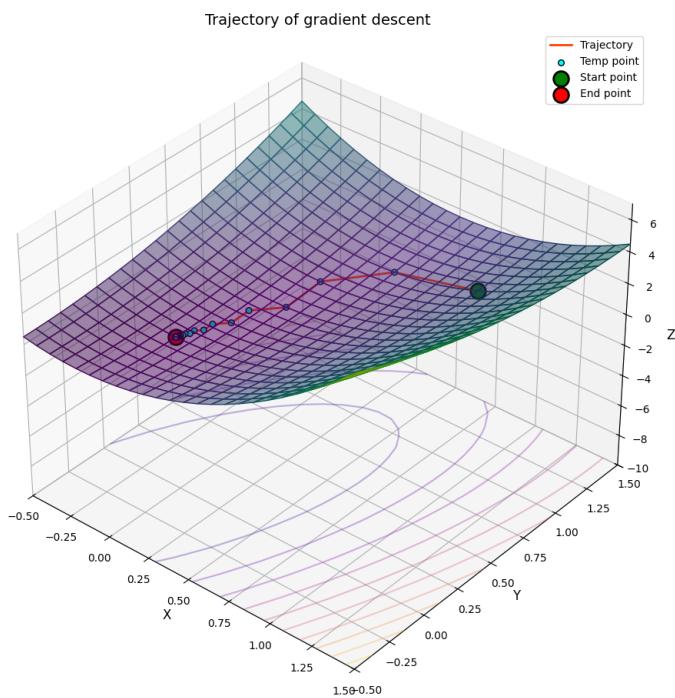
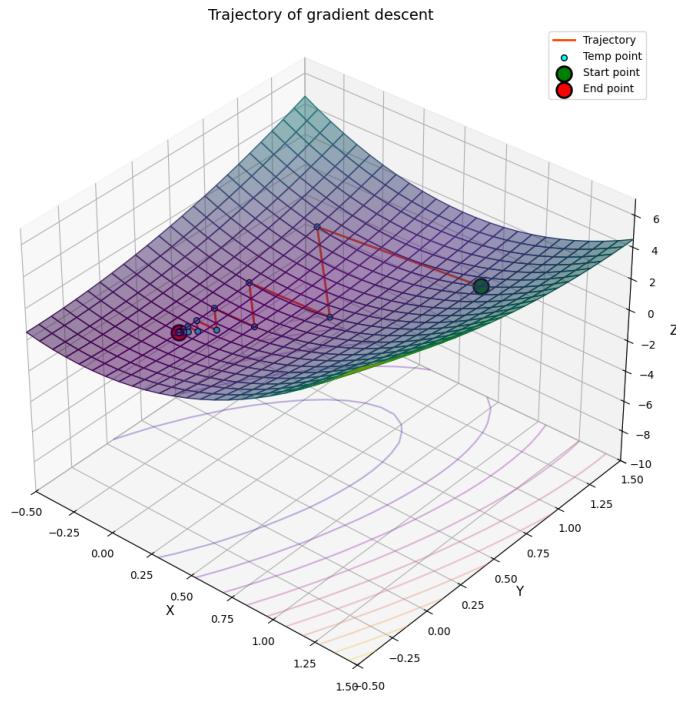


График Wolfe rule



Анализируя результаты, очевидно, что адаптивные методы, как и ожидалось, показали себя намного эффективнее фиксированных: результаты получены с большей точностью, а количество итераций в разы меньше. Лучше всего показало условие Вольфена, результат получился самым точным.

Ниже приведён дополнительный пример, где разница в работе методов видна ярко:

$$z = x^2 + y^2 \text{ в точке } (5, 5)$$

Метод	Параметры	Итерации	в.Функции	в.Градиента	$x$	$y$
Constant $\frac{1}{c}$	$c = 100$	1613	12908	3227	0.00000000000003520709	0.00000000000003520709
Dec. seq. $\frac{1}{k+c}$	$c = 1$	1819	14556	3639	-0.00000000000003533808	-0.00000000000003533808
Armijo rule	$c = 0.5$	4	72	13	0.00000000000000000000	0.00000000000000000000
Wolfe rule	$c_1 = 0.001, c_2 = 0.9$	2	52	11	-0.00000000000000000000	-0.00000000000000000000

Рассмотрим работу методов на примере мультимодальной функции, возьмём функцию Химмельблау:

$$z = (x^2 + y - 11)^2 + (x + y^2 - 7)^2 \text{ в точке } (-4, -5)$$

Метод	Параметры	Итерации	в.Функции	в.Градиента	$x$	$y$
Constant $\frac{1}{c}$	$c = 100$	30	244	61	-3,77931025337774600000	-3,28318599128616830000
Dec. seq. $\frac{1}{k+c}$	$c = 1$	3	28	7	6810,61321789811300000000	656774945658196860000000
Armijo rule	$c = 0.5$	43	1210	130	-3,77931025337774740000	-3,28318599128617000000
Wolfe rule	$c_1 = 0.001, c_2 = 0.9$	79	4252	788	3,58442834033049100000	-1,84812652696440360000

Заметим, что условия Армихо и Вольфена отработали отлично и даже нашли различные, но верные точки.

График Armijo rule

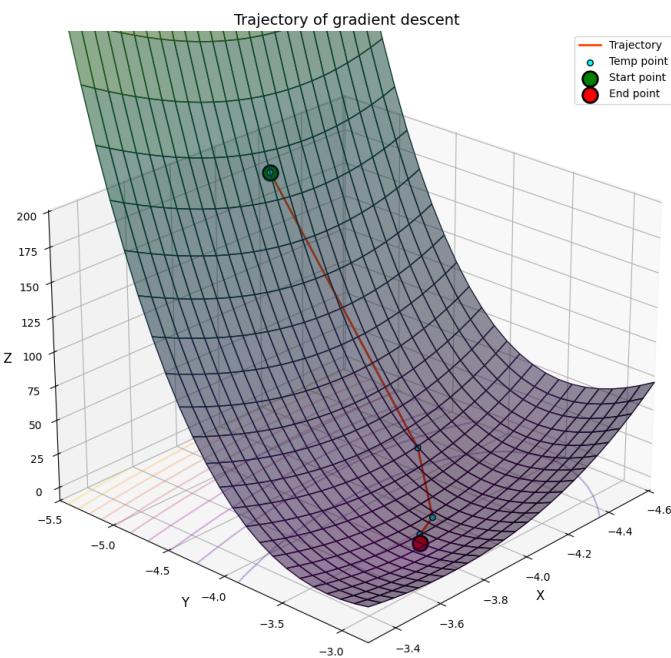
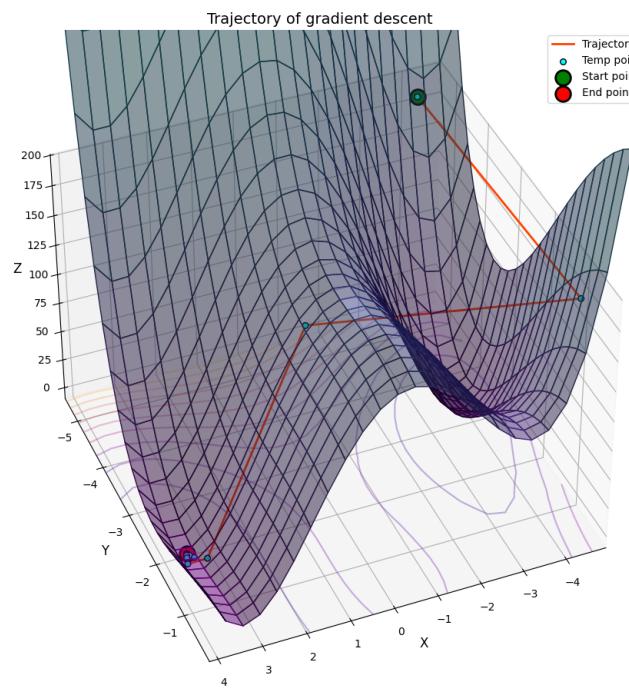


График Wolfe rule



Теперь рассмотрим методы с фиксированными шагами:  
График Constant

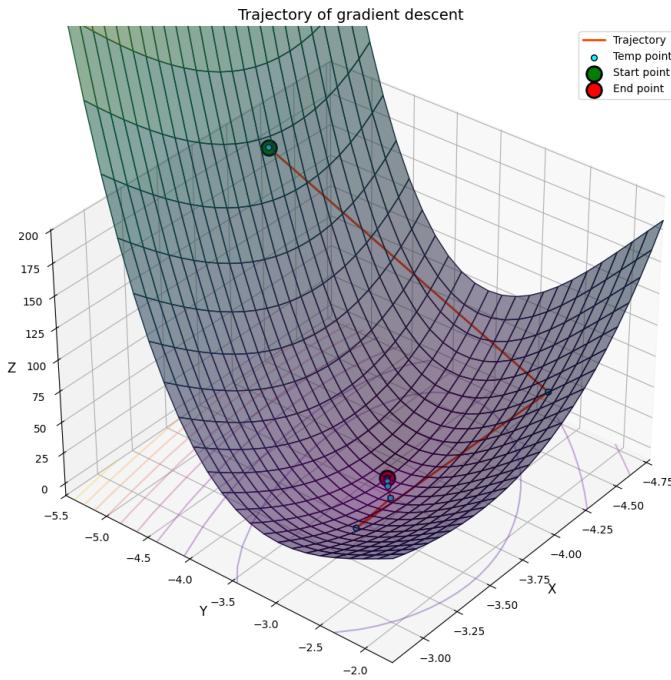
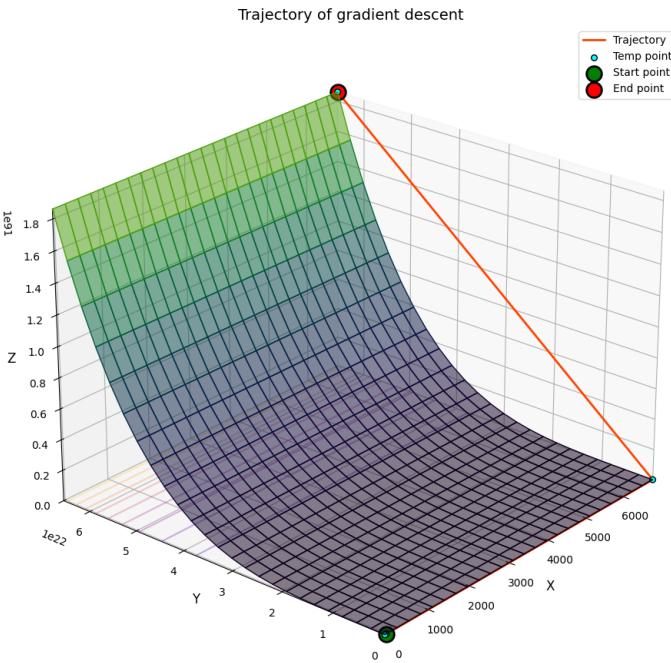


График Decreasing sequence



Как видим, убывающая последовательность не смогла найти минимум, константный шаг справился, но это негарантировано и нам просто повезло.

Выберем точку  $(-10, -10)$  и заметим, что тут оба метода не нашли минимум.

Метод	Параметры	Итерации	в.Функции	в.Градиента	$x$	$y$
Constant $\frac{1}{c}$	$c = 100$	4	36	9	-77309411326755340	-674309865469409700
Dec. seq. $\frac{1}{k+c}$	$c = 1$	2	20	5	-53170309515.99983	-62657028097.99979
Armijo rule	$c = 0.5$	45	1276	136	-3.77931025337774740000	-3.28318599128617000000
Wolfe rule	$c_1 = 0.001, c_2 = 0.9$	50	3124	571	3.00000000000000100000	2.00000000000000100000

## Сравнение с `scipy.optimize`

Рассмотрим библиотеку `scipy.optimize` и сравним с аналогами.

На примере функции Матьяса

$$z = 0.26(x^2 + y^2) - 0.48xy \text{ в точке } (1, 1)$$

Метод	Параметры	Итерации	в.Функции	в.Градиента	$x$	$y$
SciPy Armijo		663	1327	664	0.00000000000176127636	0.00000000000176127636
SciPy Wolfe	$c_1 = 0.001, c_2 = 0.9$	156	625	625	0.00000000000154016988	0.00000000000154016988
Armijo rule	$c = 0.5, c_2 = 0.9)$	663	9286	1990	0,00000000000176127638	0,00000000000176127638
Wolfe rule	$c_1 = 0.001, c_2 = 0.9$	188	2413156	402381	0,00000000000169037754	0,00000000000169037754

На примере функции Бута

$$z = (x + 2y - 7)^2 + (2x + y - 5)^2 \text{ в точке } (-4, -10)$$

Метод	Параметры	Итерации	в.Функции	в.Градиента	$x$	$y$
SciPy Armijo		17	52	18	1.0000000000002442491	2.9999999999997557509
SciPy Wolfe	$c_1 = 0.001, c_2 = 0.9$	69	246	139	1.0000000000002531308	2.9999999999997646327
Armijo rule	$c = 0.5$	82	1658	247	1,0000000000002950000	2,9999999999997200000
Wolfe rule	$c_1 = 0.001, c_2 = 0.9$	108	13805926	2301096	1,0000000000001840000	2,9999999999997560000

Анализируя результаты видно, что полученные точки приблизительно равны, но наши реализации значительно чаще проводят итераций, вычисляют значение функции, а также градиент.

## Зашумленные функции

Выберем обычную функцию  $z = x^2 + y^2$  и добавим к ней шум в виде псевдослучайных чисел из стандартного нормального распределения, умноженных на константу.

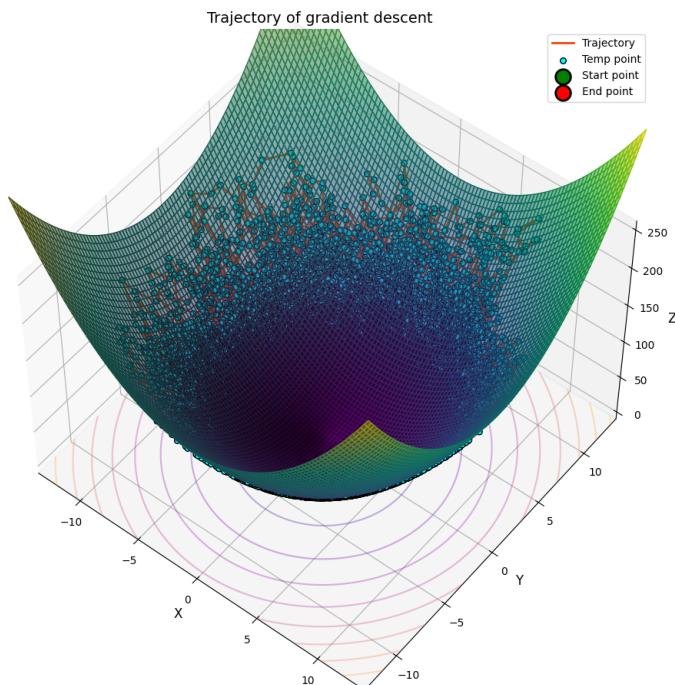
Очевидно, что результат будет каждый раз разным из-за различной величины сгенерированных значений, поэтому будем проводить несколько измерений.

Выберем константу  $10^{-6}$

$$z = x^2 + y^2 + \text{Gaussian} \cdot 10^{-6} \text{ в точке } (-1, 2)$$

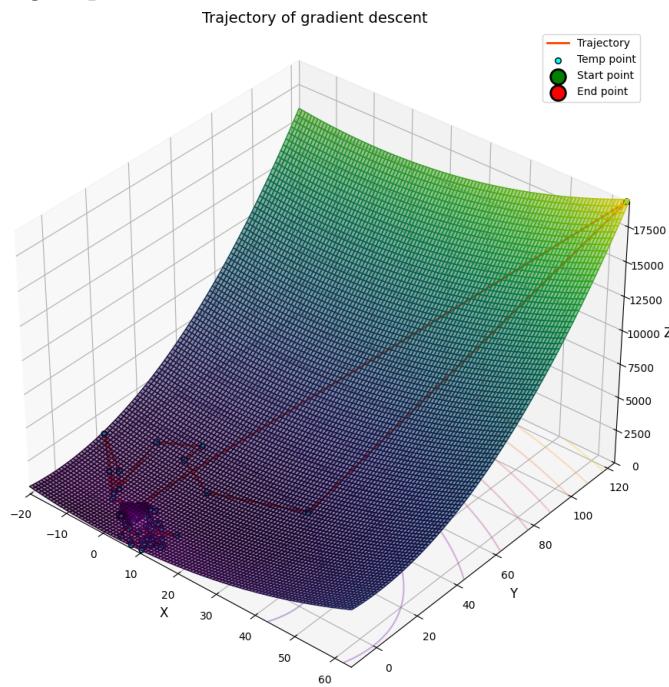
Метод	Параметры	Итерации	в.Функции	в.Градиента	$x$	$y$
Constant $\frac{1}{c}$	$c = 100$	100001	800012	200003	-0,48357856939184707000 -1,98358638303282530000 -0,78738941608935420000 1,50012077435894220000 1,69066339154980570000	5,00760708644271100000 -0,14456260738169080000 -1,19113245723866350000 -2,21970393689131030000 1,73293085111157200000

## График Constant



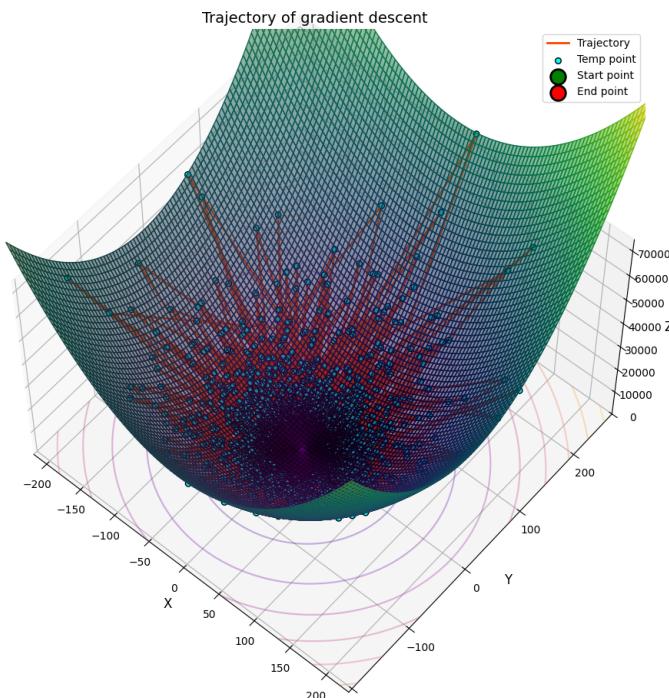
Метод	Параметры	Итерации	в.Функции	в.Градиента	$x$	$y$
Dec. seq. $\frac{1}{k+c}$	$c = 1$	100001	800012	200003	-0,07877278656235014000 0,00270311290196356860 0,10353879579000681000 0,13327056201208520000 0,07288089398803915000	0,01107968910222627000 -0,03271230451367082000 -0,39025861008420976000 -0,03549617156360861000 0,00365682408238988500

График Decreasing sequence



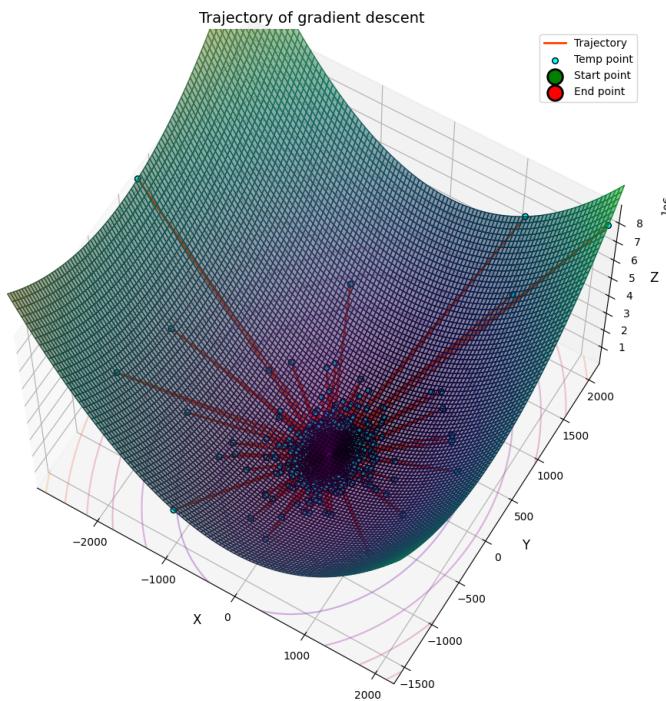
Метод	Параметры	Итерации	в.Функции	в.Градиента	$x$	$y$
Armijo rule	$c = 0.5$	100001	7762096 7617888 7651602 7677380 7856716	300004 300004 300004 300004 300004	-0,24180247187455578000 2,98341440278822800000 -1,92098347876347960000 0,10848842965993373000 0,84580459770425530000	-0,10316421137419336000 -1,51901719783183400000 -1,98462622319896200000 0,00205693195126104750 4,49726691644674700000

График Armijo Rule



Метод	Параметры	Итерации	в.Функции	в.Градиента	$x$	$y$
Wolfe rule	$c_1 = 0.001, c_2 = 0.9$	100001	20145058	3457511	66,96736577528947000000	-2,00901758207260530000
			18681016	3213504	11,03658381785707500000	1,25136069244774000000
			20294140	3482358	8,95183983862080000000	-2,57494116442730640000
			19443592	3340600	0,60337924290465320000	-9,05770844461437700000
			21542302	3690385	-11,01775634377837100000	9,02796473104691000000

### График Wolfe Rule



Изменяя константу, мы лишь будем увеличивать или уменьшать точность результата.

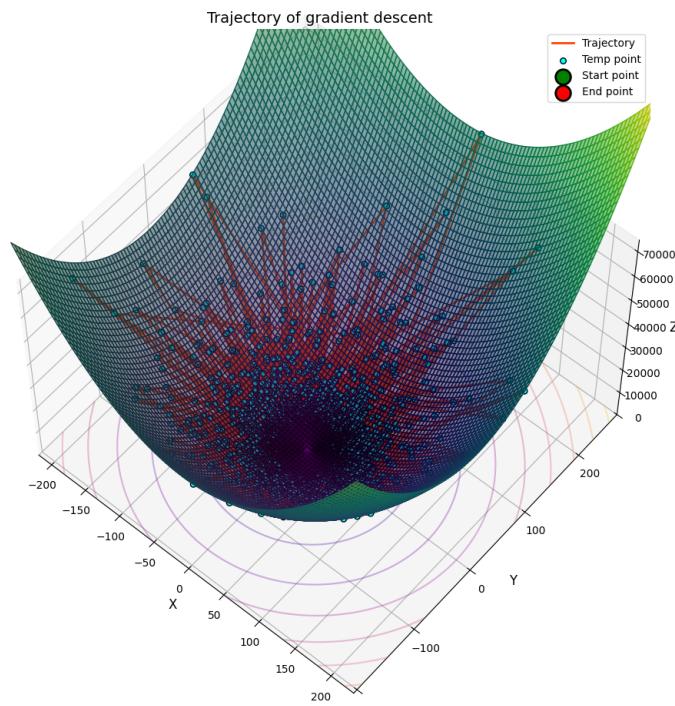
Подводя итоги, лучше всего отработал метод убывающей последовательности, хуже всего — метод Вольфе. Каждый метод не смог завершить работу и достиг предела итераций, даже при увеличении этого порога.

Пример на мультимодальной функции Химмельблау

$$z = (x^2 + y - 11)^2 + (x + y^2 - 7)^2 + \text{Gaussian} \cdot 10^{-6} \text{ в точке } (-1, -2)$$

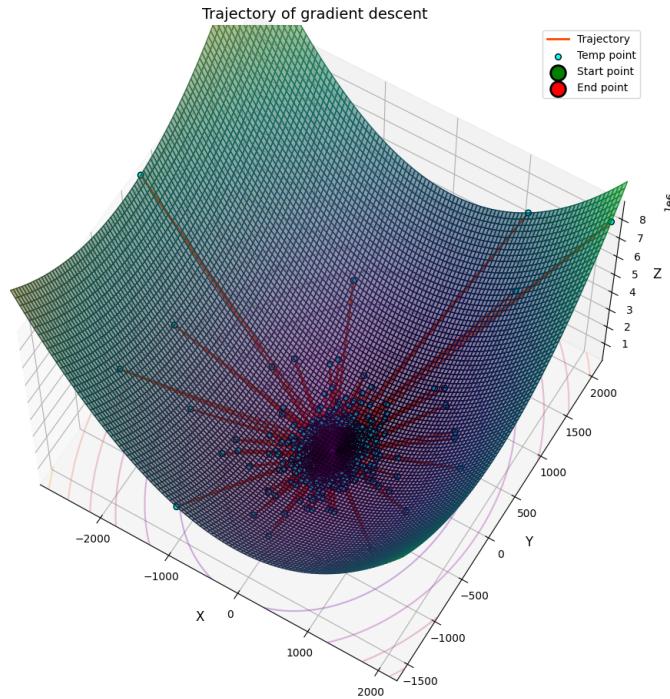
Метод	Параметры	Итерации	в.Функции	в.Градиента	$x$	$y$
Armijo rule	$c = 0.5$	100001	7903290	300004	-2,80350082134287700000	3,13312732783079540000
			7824082	300004	-3,77810637697347530000	-3,28893461101037770000
			7707266	300004	-2,77941182076884140000	3,15049213684655260000
			7688766	300004	3,61207454471641200000	-1,82139253641217350000
			7752936	300004	3,60879361244467760000	1,28362141710712100000

### График Armijo Rule



Метод	Параметры	Итерации	в.Функции	в.Градиента	$x$	$y$
Wolfe rule	$c_1 = 0.001, c_2 = 0.9$	100001	23718760	4053128	3,81428343955739500000	-2,73850905377159100000
			21110632	3618440	-2,74413168679630600000	3,06602944488956000000
			20250250	3475043	-2,55724535632076700000	2,85534453156708600000
			19747294	33391217	3,14967180706243000000	3,14967180706243000000
			19850416	3408404	-8,95518617562693800000	-1,09772060109522850000

График Wolfe Rule



На примере мультимодальной функции видно, что результат хуже: из одной и той же точки мы получаем каждый раз разные результаты, зачастую даже неблизкие к минимумам.

## Вывод

В ходе данной лабораторной работы мы подробно рассмотрели методы нулевого и первого порядка, реализовали градиентный спуск и различные вариации одномерного поиска. По результатам сравнения, самописные функции выдавали приблизи-

тельно те же результаты, что и библиотечные, но значительно больше расходовали вычислительные ресурсы. Рассмотрев наши методы одномерного поиска на самых разных функциях, мы выявили, что, безусловно, адаптивные методы являются наиболее эффективными. Мы активно сравнивали реализованные нами методы и пришли к выводу, что наиболее универсальным является условие Армихо, а более ситуативным — условие Вольфе: оно, безусловно, точнее, но в сложных случаях требует слишком много вычислений, что иногда является критичным.