Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение
высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт

По лабораторной работе №2

По дисциплине: прикладная математика

Факультет: ИТиП

Группа: M32001

Авторы:

Андреев Артём Русланович
Слюсаренко Сергей Владимирович
Шевченко Валерий Владимирович

УНИВЕРСИТЕТ ИТМО

Санкт-Петербург 2022

Лабораторная работа #2

1. Реализуйте метод градиентного спуска.
2. Оцените, как меняется скорость сходимости, если для поиска величины шага использовать различные методы:
   (a) постоянная величина шага (в зависимости от величины);
   (b) метод дробления шага;
   (c) метод золотого сечения;
   (d) метод Фибоначчи;
3. Проанализируйте траекторию реализованных методов для нескольких квадратичных функций: придумайте две-три квадратичные двумерные функции, на которых работа метода будет отличаться, рассмотрите различные начальные приближения, нарисуйте графики с линиями уровня функций и траекториями методов.
4. Проанализируйте, зависит ли сходимость методов от выбранной точки начального приближения.
5. Реализуйте один из методов сопряженных направлений (любой, по выбору):
   (b) метод Флетчера-Ривса;
6. Сравните траектории, полученные методом градиентного спуска и методом сопряженных направлений, при фиксированном начальном приближении.
7. Для защиты лабораторной работы необходимо знать описание методов на языке математики, пояснять полученные результаты, а также уметь обосновать разумность примененных Вами методов для данных функций.
8. По результатам выполнения лабораторной работы необходимо подготовить отчет. Отчет должен содержать ссылку на реализацию, необходимые тесты, таблицы и рисунки.

Метод градиентного спуска.

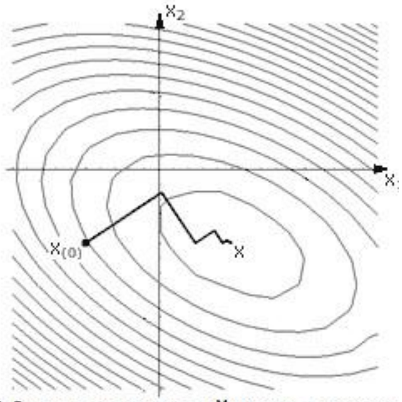
Градиент – вектор, который содержит все частные производные по соответствующей координате. Он обладает таким свойством, что он смотрит в сторону увеличения функции по всем координатам (в пределах дельта окрестности), т. е. с определенной точностью можно сказать, что функция будет уменьшаться при движении в направлении антиградиента. Само собой нельзя сказать, что функция всегда будет уменьшаться при движении в сторону антиградиента, поэтому мы делаем шаги определенной величины.

Как величины этого шага могут быть найдены:

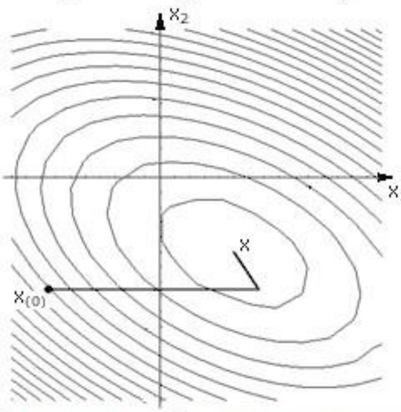
1) С константным шагом (ну или почти) – Мы идем по выбранному нами направлению, затем, когда понимаем, что функция не уменьшается, мы уменьшаем шаг (обычно в 2 раза).
2) Пока не выполнится условие: $f(x_k) - f(x_k - \alpha_k f'(x)) \leq \varepsilon \alpha_k |f'(x_k)|^2 \quad (\varepsilon - \text{точность})$, мы дробим длину шага (то есть умножаем на коэффициент величиной от 0 до 1).
3) С использованием методов оптимизации нулевого порядка – суть заключается в том, что мы оптимизируем шаг. Будем искать точку минимума функции $g(\alpha) = f(x_k - \alpha f'(x_k))$. Соответственно мы свели задачу к задаче одномерного поиска, для решения которой воспользуемся алгоритмами из лабораторной работы №1.

Метод Флетчера-Ривса.

Этот метод является одним из методов сопряженных градиентов (направлений). Сначала стоит объяснить разницу между методом сопряженных градиентов и методом наискорейшего градиентного спуска. Как мы можем заметить, направление может меняться очень долго, пока наконец не будет достигнут минимум. Метод сопряжённых градиентов решает эту проблему.



Метод наискорейшего спуска



Метод сопряжённых градиентов

Вводится понятие сопряжённости векторов по отношению к какой-то матрице. Вектора x и y называют A-сопряженными, если скалярное произведение x и Ay равно нолю. Получается, что два соседних направления являются сопряжёнными, по отношению к квадратичной форме исходной квадратичной функции. Есть несколько способов определения нового направления. Можно воспользоваться методами линейной алгебры, но для этого нужно знать иметь квадратичную форму. Метод Флетчера-Ривса не использует эту матрицу. Согласно этому методу, новое сопряженное направление получается сложением антиградиента в точке поворота и предыдущего направления, умноженного на отношение соседних значений градиента в квадрате.

В случае квадратичной функции метод Флетчера-Ривса гарантирует сходимость за n (или менее) шагов, где n – размерность вектора направления. Если функция не квадратичная, то метод Флетчера-Ривса перестаёт быть итеративным и конечным.

Алгоритмы (https://github.com/JabaJabila/ITMO_AppliedMath/tree/main/Lab2):

Метод градиентного спуска:

```
import math import
numpy as np
from algorithms.stepSizeFunction import StepSizeFunction from
oracle.firstOrderOracle import FirstOrderOracle

def gradient_descent(
oracle: FirstOrderOracle,
start_x: np.array,
        step_size_func: StepSizeFunction,
        eps: float, max_iter=-1, max_alpha=0.5, refresh_each=100,
exit_clause="both"):  # argument/function/both
      if max_alpha <=
0:          max_alpha =
1
      steps =
[start_x]
iteration = 0

    prev_x = start_x + 2 * eps
curr_x = start_x
    prev_f = oracle.function(start_x) + eps * 2
curr_f = oracle.function(start_x)     alpha =
max_alpha
      while (iteration < max_iter or max_iter < 0) \              and
check_exit_clause(prev_x, curr_x, prev_f, curr_f, eps, exit_clause):          prev_x,
prev_f = curr_x, curr_f          grad = oracle.gradient(prev_x)
          if np.all(np.abs(grad) <= eps /
100):             break
          def func(a):             return
oracle.function(prev_x - a * grad)

        alpha = step_size_func.calc_step(func, 0,z alpha, eps, oracle, prev_x)
        # refresh          if alpha <= eps / 100 or
iteration == refresh_each:             alpha = max_alpha
          curr_x = prev_x - alpha * grad
curr_f = oracle.function(curr_x)
          while step_size_func.split_step and curr_f >=
prev_f:             alpha /= 2          if alpha <= eps /
100:            return prev_x, steps
            alpha = step_size_func.calc_step(func, 0, alpha, eps, oracle, prev_x)
curr_x = prev_x - alpha * grad             curr_f = oracle.function(curr_x)

steps.append(curr_x)
iteration += 1

    return curr_x, steps
  def check_exit_clause(prev_x, curr_x, prev_f, curr_f, eps,
condition):
    if condition == "argument":
        return math.dist(prev_x, curr_x) >= eps
elif condition == "function":
        return math.fabs(curr_f - prev_f) >= eps
    return math.dist(prev_x, curr_x) >= eps and math.fabs(curr_f - prev_f) >= eps
```

Метод Флетчера-Ривса:

```
import math import
numpy as np
from algorithms.stepSizeFunction import StepSizeFunction from
oracle.firstOrderOracle import FirstOrderOracle

def fletcher_reeves(
```

```
        oracle: FirstOrderOracle,
start_point: np.array,
step_size_func: StepSizeFunction,
eps1, eps2, max_iter, max_step):
points = [start_point]      directions =
[]      counter = 0
    end_condition_counter = 0
      while counter < max_iter:         current_gradient_value =
oracle.gradient(points[counter])          if
np.linalg.norm(current_gradient_value) < eps1:              return
points[counter], points
          if counter == 0:             directions.append(-
current_gradient_value)         else:            prev_gradient_value
= oracle.gradient(points[counter - 1])             prev_B =
(np.linalg.norm(current_gradient_value) /
np.linalg.norm(prev_gradient_value)) ** 2
            directions.append(-current_gradient_value + prev_B * directions[counter -
1])           def func(t):             return oracle.function(points[counter]
+ t * directions[counter])

        step_size = step_size_func.calc_step(func, 0, max_step, eps1 / 1000, oracle, 0)
points.append(points[counter] + step_size * directions[counter])
          current_point_result =
oracle.function(points[counter])         next_point_result =
oracle.function(points[counter + 1])
        if np.linalg.norm(points[counter + 1] - points[counter]) < eps2 and \
math.fabs(next_point_result - current_point_result) < eps2:
            end_condition_counter += 1
if end_condition_counter == 2:
                return points[counter], points
else:              end_condition_counter = 0

        counter += 1
      return points[counter],
points
```

Методы выбора шага:

Постоянная величина шага:

```
from algorithms.stepSizeFunction import StepSizeFunction from
oracle import firstOrderOracle

class ConstStep(StepSizeFunction):
split_step: bool
      def __init__(self):          super().__init__()
self.split_step = True      def calc_step(self, function, left_bound,
```

```
right_bound, eps, oracle: firstOrderOracle, curr_x):          return
right_bound
```

Метод дробления шага:

```
from algorithms.stepSizeFunction import StepSizeFunction from
oracle import firstOrderOracle
  class
SplittingStep(StepSizeFunction):
split_step: bool     delta: float
      def __init__(self,
delta=0.5):
        super().__init__()
        self.delta = delta if 0 < delta < 1 else 0.5     def
calc_step(self, function, left_bound, right_bound, eps, oracle:
firstOrderOracle, curr_x):          alpha = right_bound
        while oracle.function(curr_x) - oracle.function(
                curr_x - alpha * oracle.gradient(curr_x)) < eps * alpha * (sum(curr_x
** 2) ** 0.5):
alpha *= self.delta

        return alpha
```

Метод золотого сечения:

```
from algorithms.stepSizeFunction import StepSizeFunction
import golden_ratio_algorithm from oracle import
firstOrderOracle

class GoldenRatioStep(StepSizeFunction):
split_step: bool
      def __init__(self):          super().__init__()     def
calc_step(self, function, left_bound, right_bound, eps, oracle:
firstOrderOracle, curr_x):
        return golden_ratio_algorithm.find_min(function, left_bound, right_bound,
eps)[0]
```

Метод Фибоначчи:

```
from algorithms.stepSizeFunction import StepSizeFunction
import fibonacci_algorithm from oracle import
firstOrderOracle

class FibonacciStep(StepSizeFunction):
    split_step: bool
      def __init__(self):          super().__init__()     def
calc_step(self, function, left_bound, right_bound, eps, oracle:
firstOrderOracle, curr_x):
        return fibonacci_algorithm.find_min(function, left_bound, right_bound, eps)[0]
```

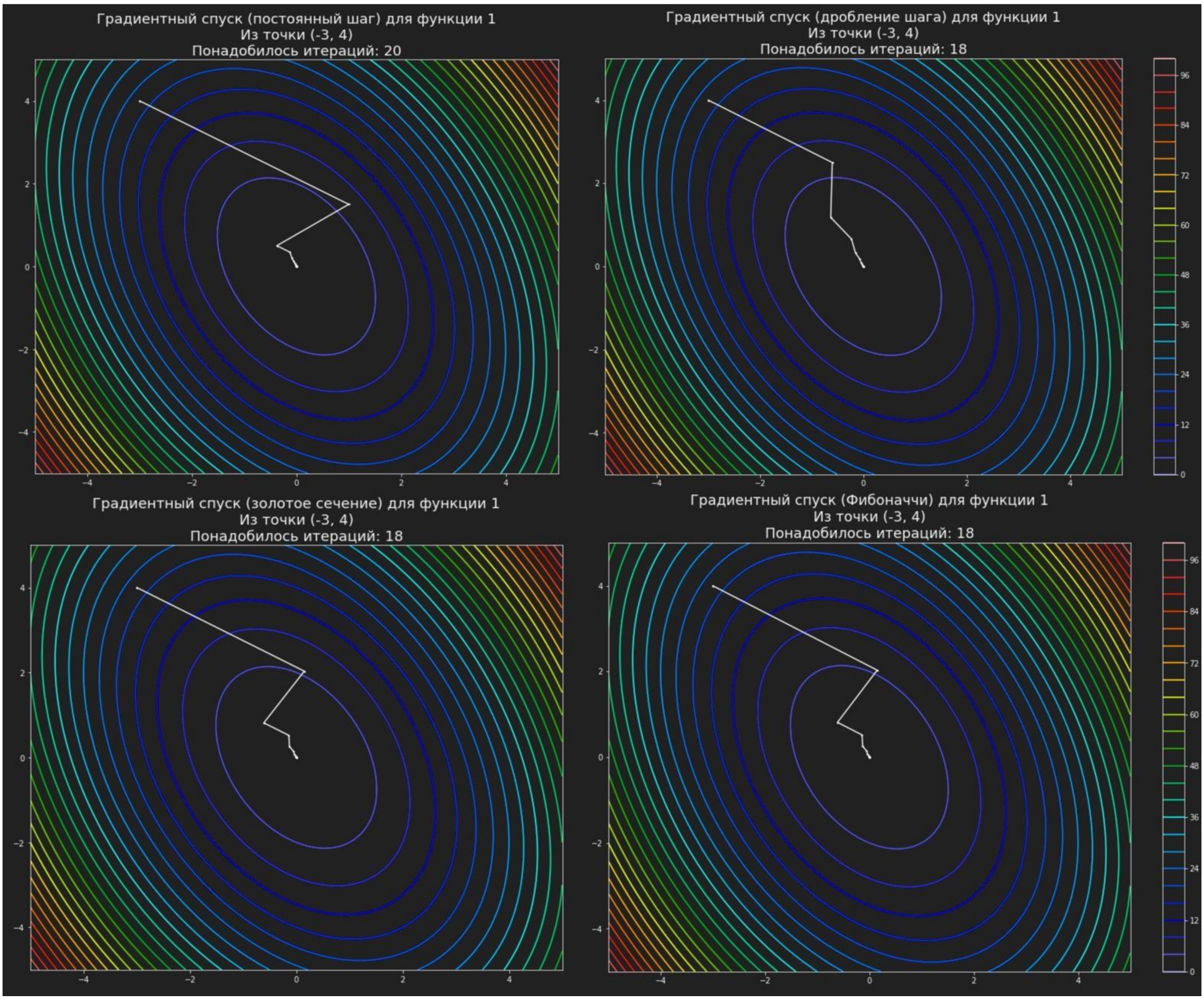
Возьмём 2 квадратичные двумерные функции и проанализируем траектории нахождения минимума градиентным спуском, используя разные методы поиска длины шага:

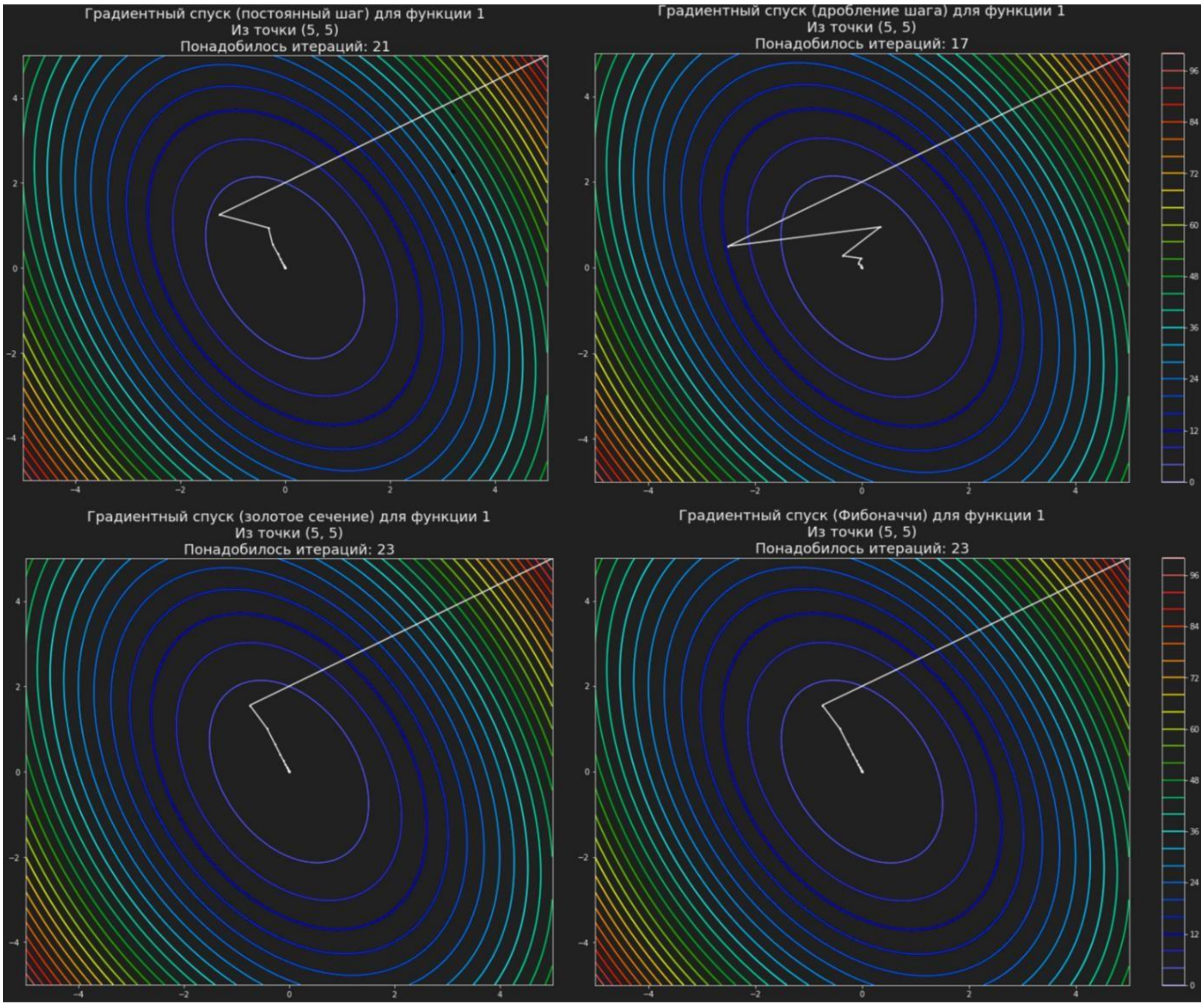
Функция 1:

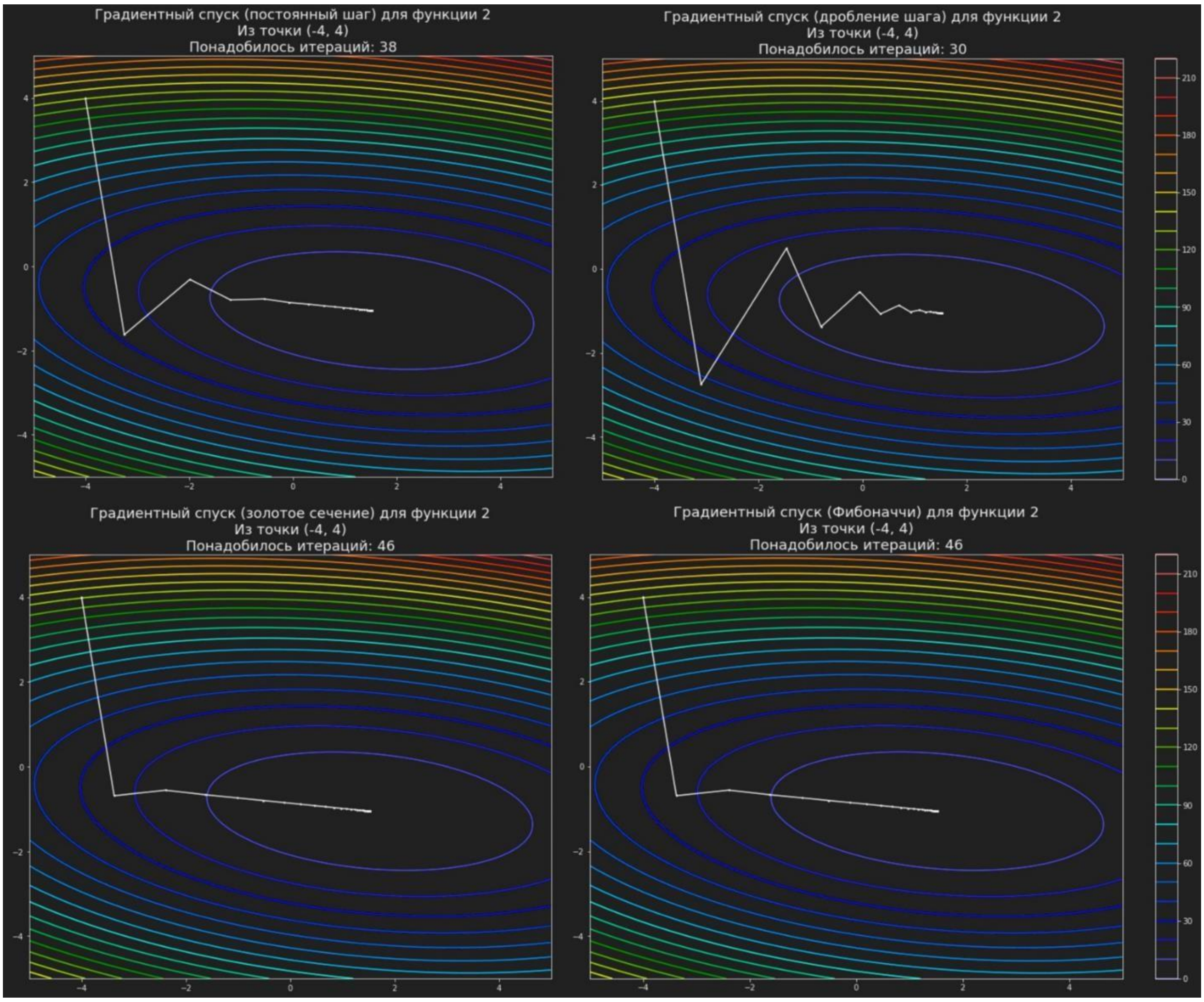
$$f(x,y) = 2x^2 + xy + y^2$$

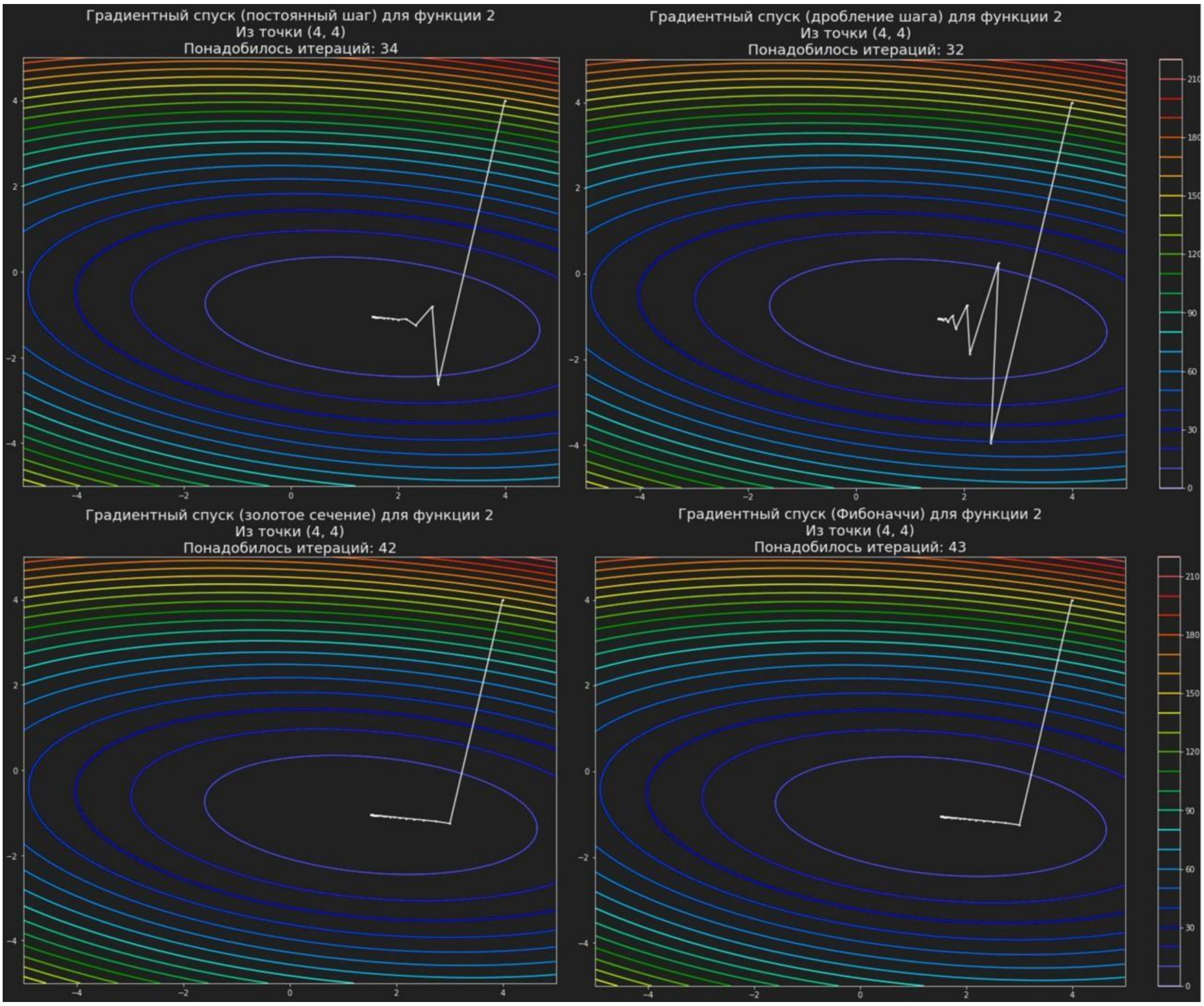
Функция 2:

$$f(x,y) = x^2 + (x-1)(y-2) + 5(y+1)^2$$




Градиентный спуск (постоянный шаг) для функции 1
Из точки (5, 5)
Понадобилось итераций: 21
Градиентный спуск (дробление шага) для функции 1
Из точки (5, 5)
Понадобилось итераций: 17
Градиентный спуск (золотое сечение) для функции 1
Из точки (5, 5)
Понадобилось итераций: 23
Градиентный спуск (Фибоначчи) для функции 1
Из точки (5, 5)
Понадобилось итераций: 23


Градиентный спуск (постоянный шаг) для функции 2
Из точки (-4, 4)
Понадобилось итераций: 38
Градиентный спуск (дробление шага) для функции 2
Из точки (-4, 4)
Понадобилось итераций: 30
Градиентный спуск (золотое сечение) для функции 2
Из точки (-4, 4)
Понадобилось итераций: 46
Градиентный спуск (Фибоначчи) для функции 2
Из точки (-4, 4)
Понадобилось итераций: 46



Из графиков видна разница в поведении поиска минимума при разных способах выбора шага и разного начального приближения. Алгоритм постоянной величины шага использует дробление в случае несоблюдения условия монотонности при поиске и данный алгоритм в среднем отрабатывает немного хуже, чем способ с дроблением шага. Поведение алгоритмов с использование золотого сечения или метода Фибоначчи практически идентичное, поскольку данные алгоритмы одномерной оптимизации сильно схожи по поведению. Данные алгоритмы отрабатывают лучше предыдущих двух, когда размер шага был выбран не оптимально.

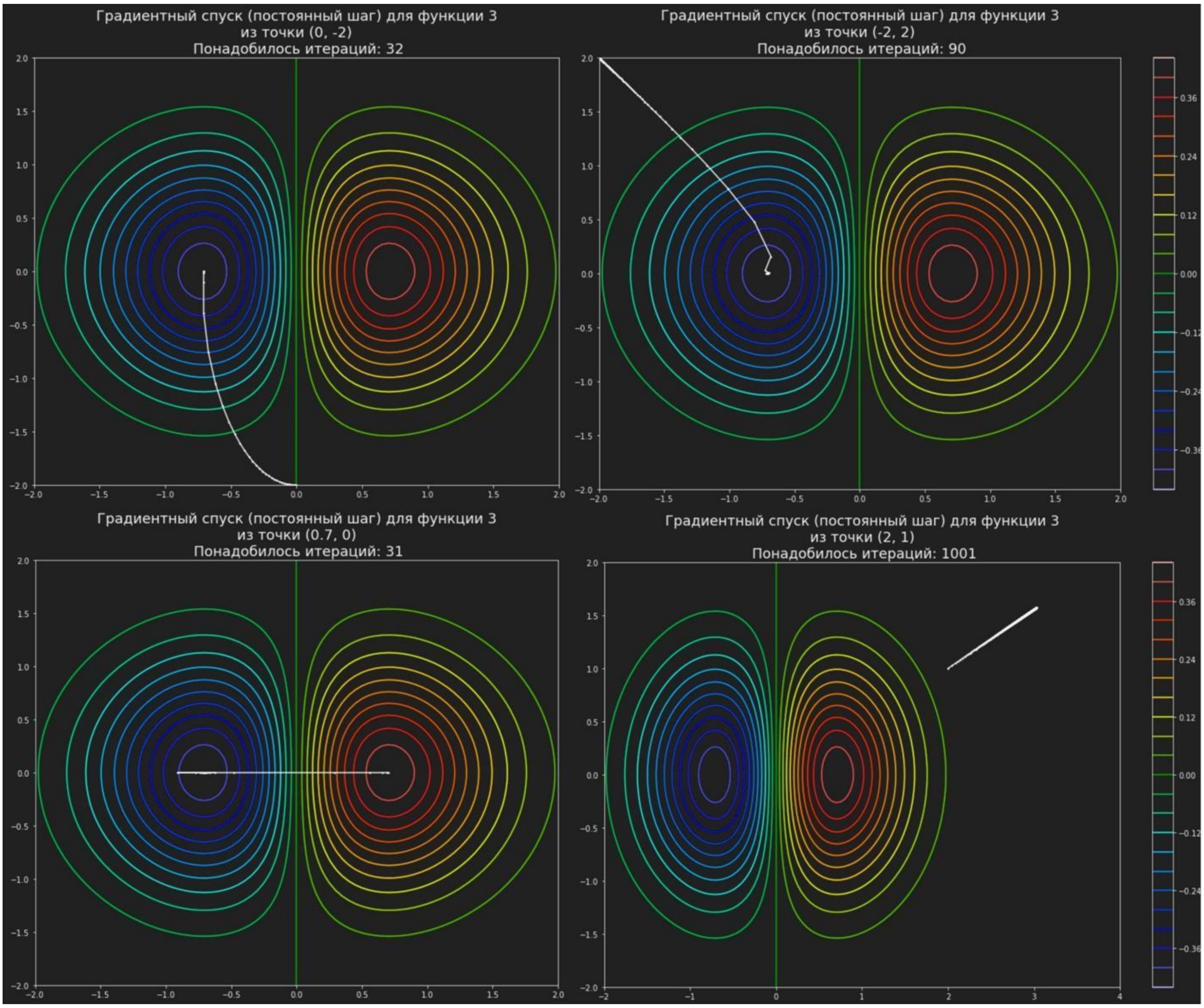
Для наилучше наглядности разницы в выборе начального предложения возьмём две неквадратичные двумерные функции:

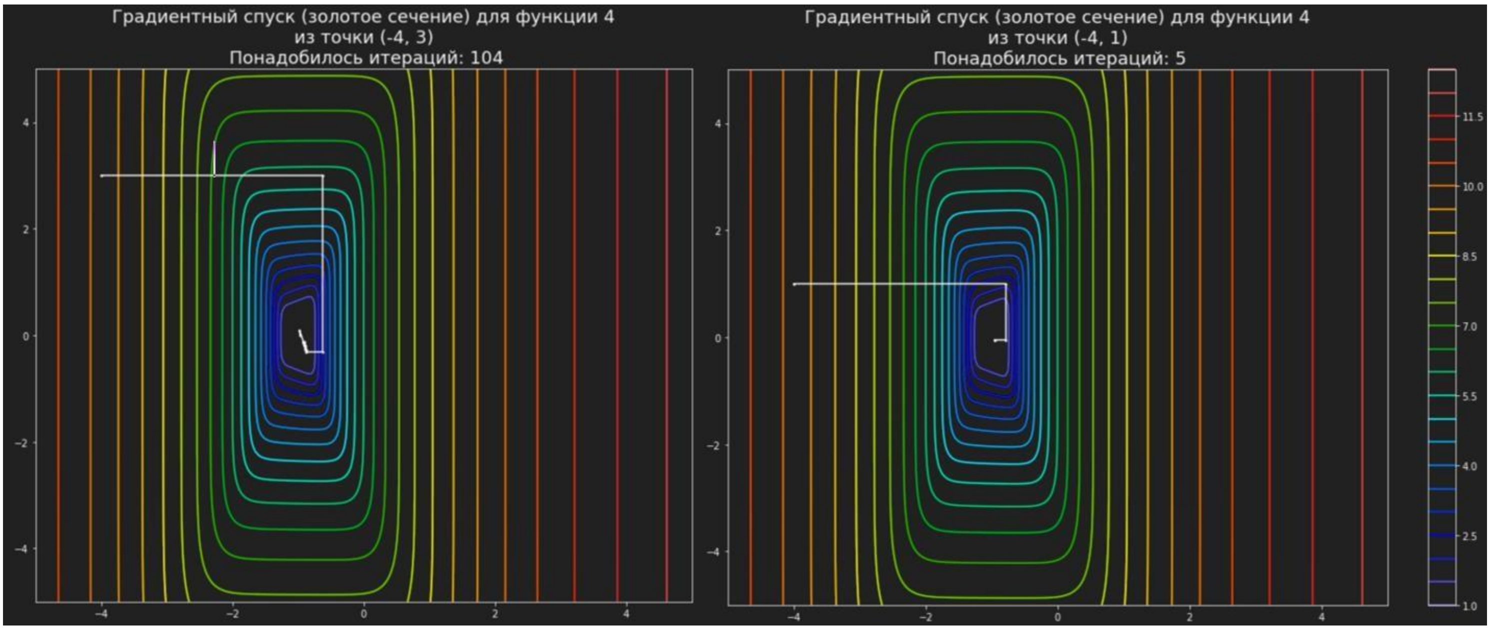
Функция 3:

$f(x, y) = xe^{-(x^2+y^2)}$

Функция 4:

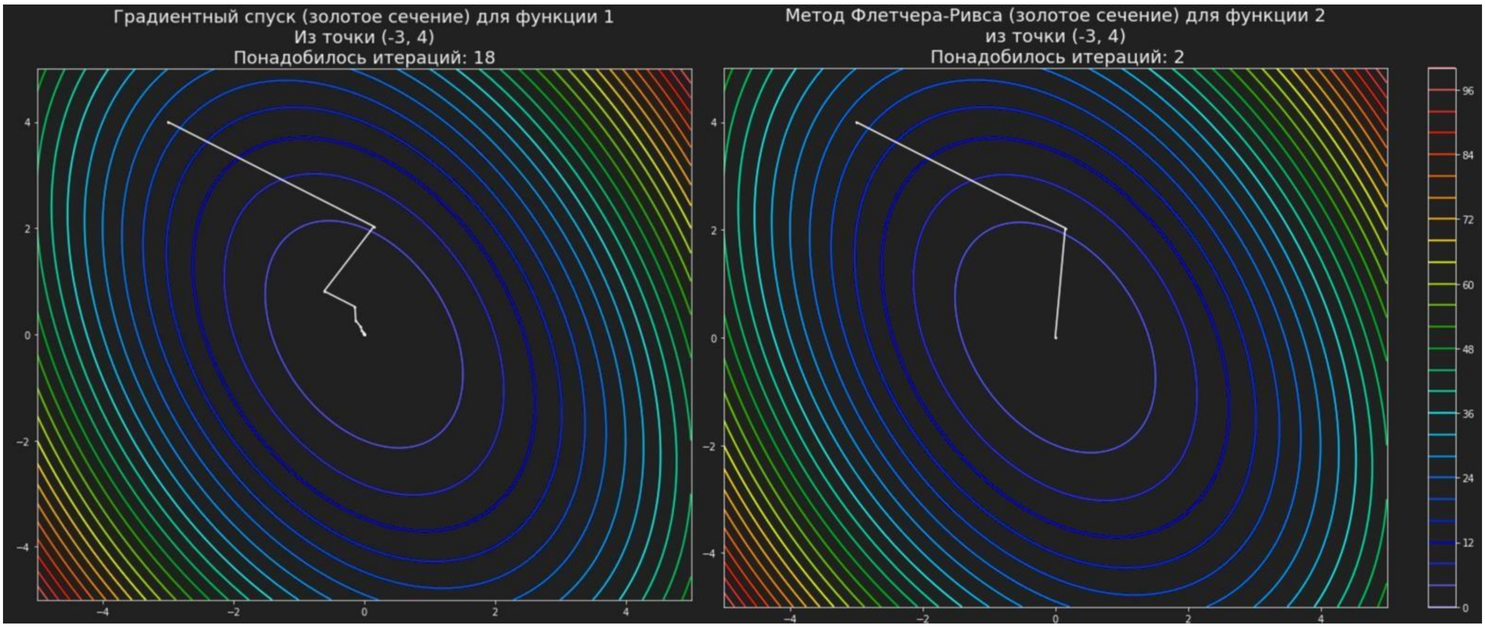
$f(x, y) = log_{10}(10000(x + 1)^8 + x^4y^4 + y^8 + 0,1) + 2$

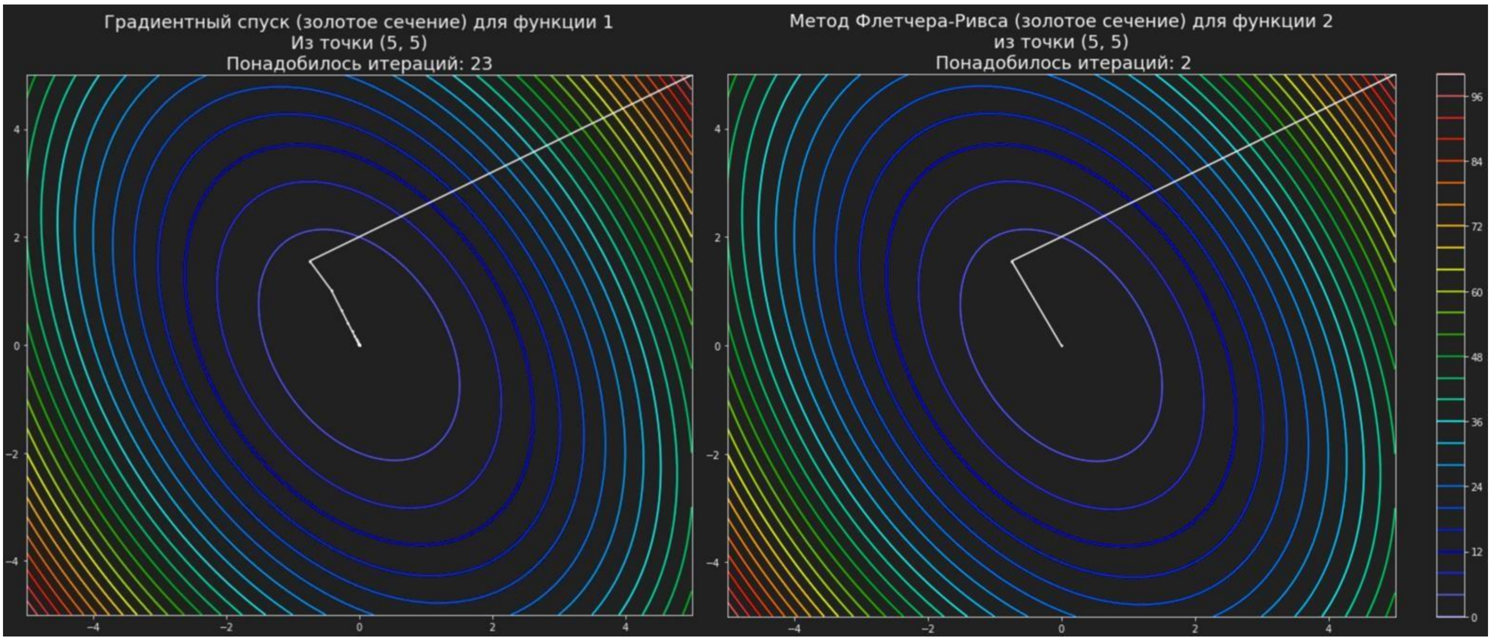


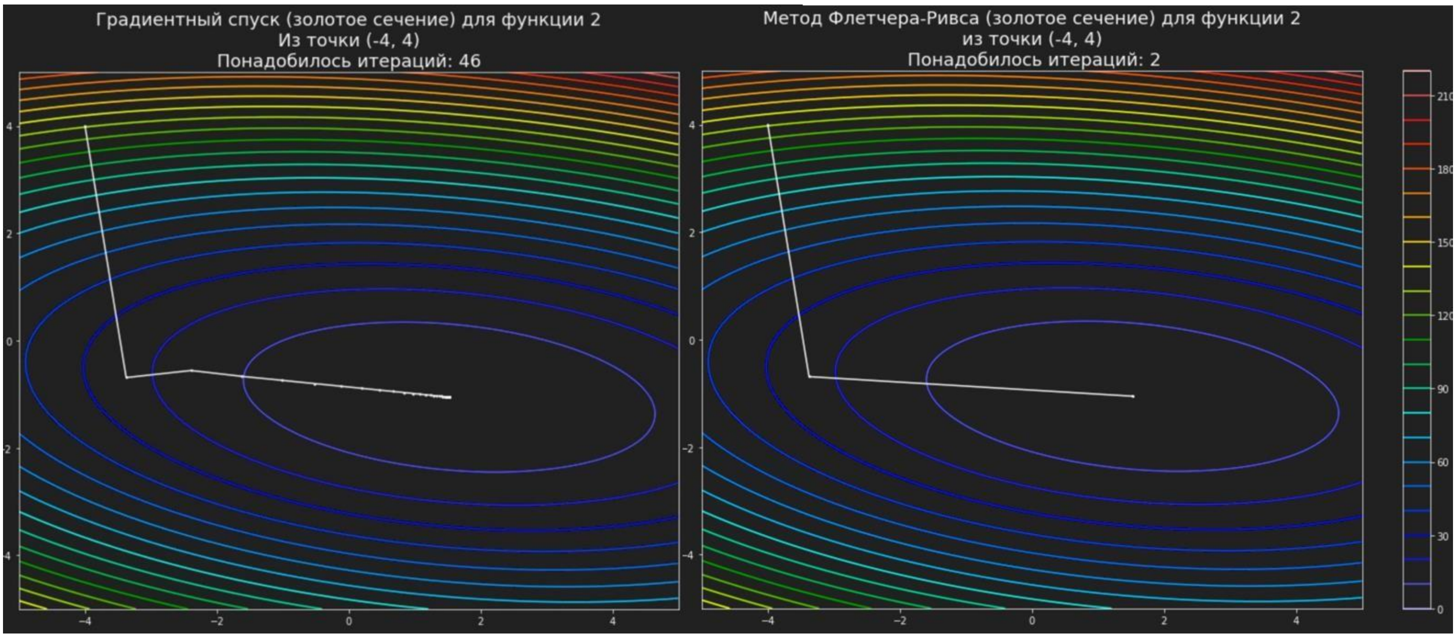


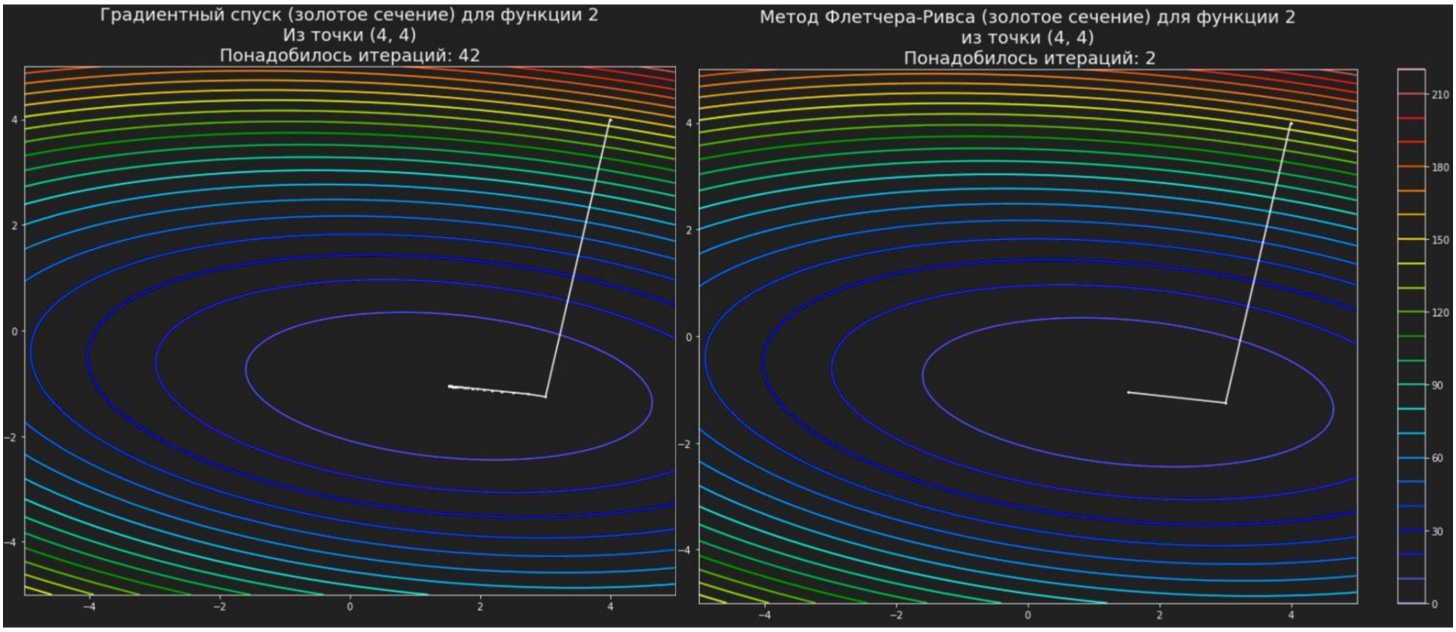
Данные примеры наиболее наглядно демонстрируют разницу в эффективности при неудачном выборе начального приближения. Для третьей функции на последней картинке вовсе видно, что алгоритм не может найти минимум, который отделён от начальной точки максимумом. Поэтому данный алгоритм останавливается после исчерпания лимита попыток (1000 итераций).

Сравним метод градиентного спуска с методом Флетчера-Ривса:

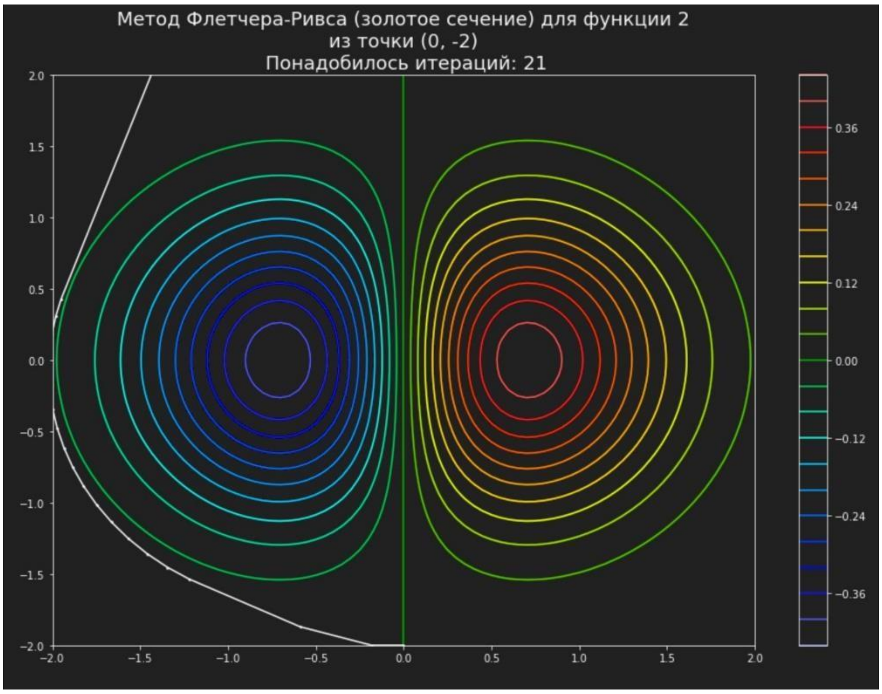


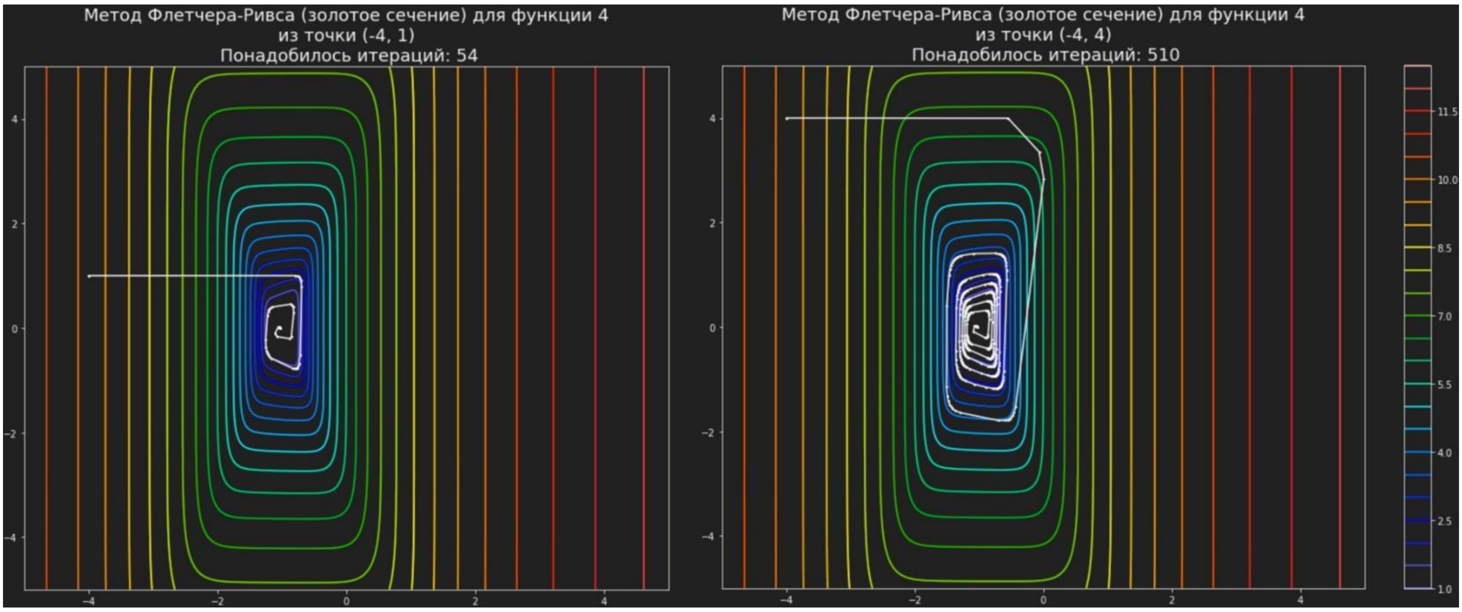






Для квадратичных функций метод Флетчера-Ривса успешно отрабатывает за n шагов, где n – кол-во аргументов функции. Но данный метод гарантирует такой результат только для квадратичных функций. Рассмотрим поведения для функций 3, 4:





Сразу видна разница. Для функции 3 алгоритм не находит точку минимума, траектория огибает спуск и уходит вдаль, алгоритм ломается. Для функции 4 нам повезло и алгоритм находит точку минимума за довольно большое число итераций. Также видно, что эффективность зависит от выбора начального приближения.