

Методы стохастической оптимизации.  
Настройка гиперпараметров.  
лабораторная работа №4

Чистов Олег Дмитриевич

Высотин Данил Абузарович

---

M3234  
**GitHub**  
@korivkosan

# Содержание

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Основное задание</b>  | <b>2</b>  |
| 1.1      | Разработанный нами UI . . . . .  | 2         |
| 1.2      | Разберите теоретическое описание и реализуйте метод стохастической оптимизации (метод имитации отжига) . . . . . | 5         |
| 1.3      | Сравните его эффективность на методах и примерах из лаб. 1 и/или лаб 2 . . . . .                                 | 7         |
| <b>2</b> | <b>Дополнительное задание 1</b>  | <b>11</b> |
| <b>3</b> | <b>Дополнительное задание 2</b>  | <b>15</b> |

# 1 Основное задание

## 1.1 Разработанный нами UI

1. **Возможность работать с функциями 1, 2 и более переменных**
  - UI позволяет пользователю вводить и обрабатывать функции различных размеров, от простых однопеременных до многопеременных функций, что делает его универсальным инструментом для математических исследований и анализа.
2. **Визуализация функций 1 и 2 переменных**
  - Интерфейс поддерживает графическое представление функций одной и двух переменных, что помогает пользователю лучше понять поведение функций и их свойства через визуальные средства.
3. **Удобное переключение между различными измерениями пространства**
  - Пользователь может легко переключаться между анализом функций различных размерностей, что упрощает исследование многомерных функций и пространств.
4. **Ускорение процесса изменения параметров и гиперпараметров исследования**
  - UI предоставляет быстрый доступ к изменению параметров и гиперпараметров, что позволяет пользователю оперативно настраивать исследуемые функции и получать результаты в реальном времени.
5. **Быстрое переключение между методами исследования**
  - Интерфейс оснащен функцией быстрого переключения между различными методами исследования, такими как градиентный спуск, методы численного интегрирования и др., что позволяет эффективно сравнивать результаты.
6. **Удобная визуализация данных**
  - Визуализация данных в интерфейсе предоставляет такие возможности, как перемещение видимости на графике, поворот графика, изменение масштаба и отображение хода поиска минимума, что делает процесс анализа интуитивно понятным и наглядным.
7. **Обработка ошибок ввода без перезапуска системы**
  - В случае ошибок ввода система способна обработать их без необходимости перезапуска, что экономит время и усилия пользователя.
8. **Быстрое переключение между функциями**
  - Пользователь может оперативно переключаться между различными функциями, что ускоряет процесс исследования и анализа.
9. **Возможность удобного ввода собственной функции**
  - Интерфейс позволяет легко вводить и редактировать пользовательские функции, обеспечивая гибкость и удобство в использовании.
10. **Читаемый вывод**
  - Все результаты и данные выводятся в читаемом и структурированном формате, что облегчает восприятие и анализ информации.

К содержанию

## Пример UI

Метод отжига

Функция:  $x^2 + y^2$

Измерение: 2, vars: x y

Левая граница x: -2 Правая граница x: 2 Шаг по x: 0.1

Левая граница y: -2 Правая граница y: 2 Шаг по y: 0.1

Применить

Метод: отжиг с постоянным шагом температуры

Шаг точки: 0.1

Шаг температуры: 0.01

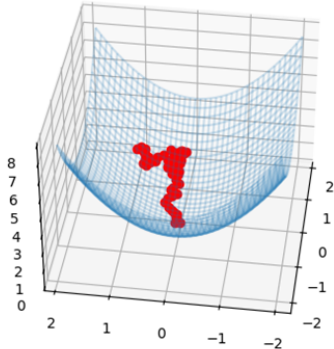
Результат: (-0.09, 0.02)

Значение: 0.008910

Начальная координата x: 1

Начальная координата y: 1

Запустить



Метод: отжиг с постоянным шагом температуры

Шаг точки: отжиг с постоянным шагом температуры

Шаг температуры: градиентный спуск

Результат: Нелдер-мид

Функция:  $x^2 + y^2$

Измерение:  $x^2 + y^2$

Левая граница x:  $(x^2 + y^2) * ((x-1)^2 + (y-1)^2)$

Правая граница x: 2

К содержанию

Интерфейс был реализован с помощью библиотеки языка python под названием Tkinter

Метод отжига

|                 |             |                  |
|-----------------|-------------|------------------|
| Функция:        | $x^2 + y^2$ | $x^2 + y^2$      |
| Измерение       | 2           | 2, vars: x y     |
| Левая граница x | -2          | Правая граница x |
| Левая граница y | -2          | Правая граница y |
|                 |             | Шаг по x 0.1     |
|                 |             | Шаг по y 0.1     |

Применить

Метод: градиентный спуск

Шаг точки: 0.4

Условие останова: шаг < эпсилон 0.001

Метод одномерного поиска: Метод золотого сечения

Результат: (0.00, 0.00)

Значение: 0.000000

Начальная координата x: 3

Начальная координата y: 5

Запустить

Метод отжига

|           |                                 |                                 |
|-----------|---------------------------------|---------------------------------|
| Функция:  | $x_1^2 + x_2^2 + x_3^2 + x_4^2$ | $x_1^2 + x_2^2 + x_3^2 + x_4^2$ |
| Измерение | 4                               | 4, vars: x1 x2 ...              |

Применить

Метод: градиентный спуск

Шаг точки: 0.1

Условие останова: шаг < эпсилон 0.001

Метод одномерного поиска: Метод золотого сечения

Результат: (0.00, 0.00, -0.00, 0.00)

Значение: 0.000024

Число итераций: 44

Количество вычислений функции: 740

Начальная координата x1: 31

Начальная координата x2: 22

Начальная координата x3: -20

Начальная координата x4: 10

К содержанию

## 1.2 Разберите теоретическое описание и реализуйте метод стохастической оптимизации (метод имитации отжига)

В данной работе в качестве стохастического метода оптимизации мы решили разобрать метод имитации отжига.

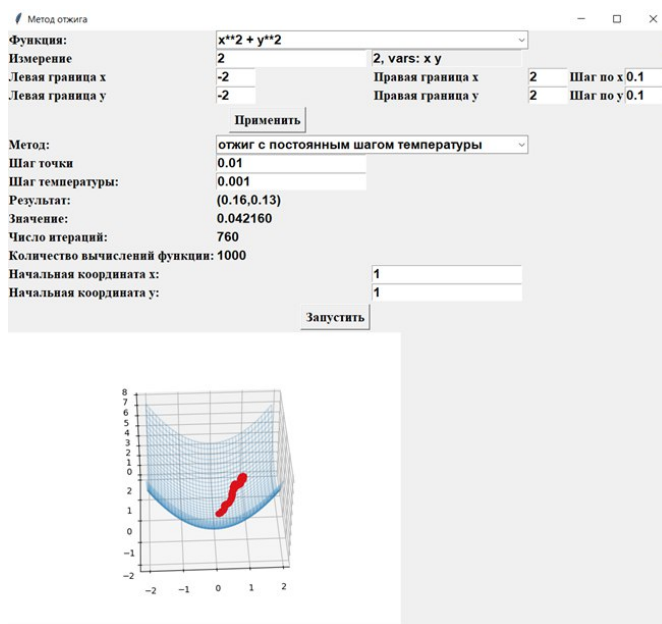
В простейшей реализации данного метода есть температура, которая изначально равна 1, она линейно убывает с каждым шагом на заданный шаг.

```
1 temp = 1
2 points = [np.array(start_point)]
3 while temp > 0:
4     try:
5         next_point = new_point_rand(points[-1], step)
6     except RuntimeError:
7         break
8     if function(*next_point) < function(*points[-1]):
9         points.append(next_point)
10    elif random.random() < temp:
11        points.append(next_point)
12    temp -= temp_step
13 values = np.array([function(*start_point) for start_point in points])
```

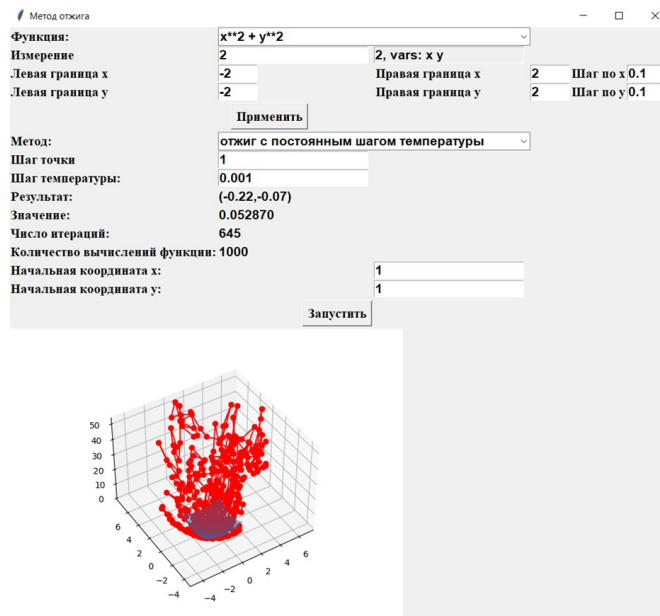
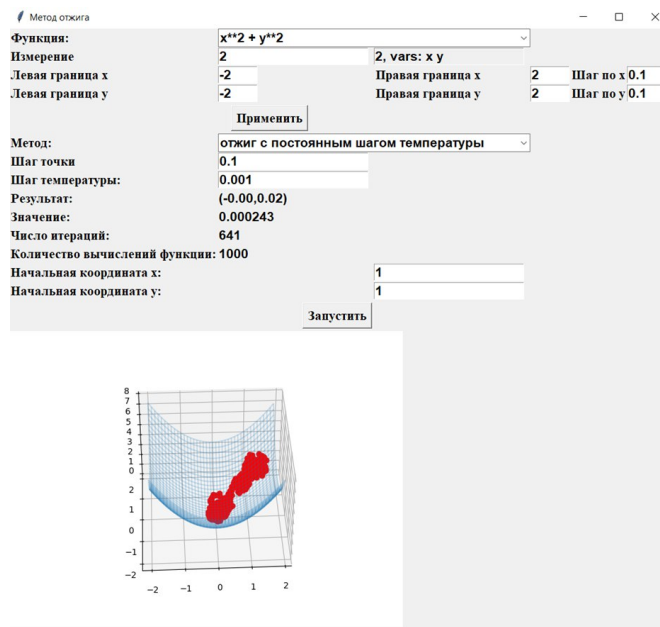
Если значение функции в следующей точке меньше исходного, тогда мы перемещаемся в эту точку. Иначе, если значение функции в следующей точке больше предыдущей, то с вероятностью  $t$  мы туда идем иначе стоим.

Функция получения новой точки - случайная точка из бруса  $\{x_i - step < x < x_i + step \text{ for } x_i \text{ in point}\}$  где  $step$  задается пользователем

Таким образом, мы блуждаем по полю, изначально неосторожно но затем все более и более обдуманней. Свобода блуждания также сильно зависит от выбора  $step$ :



К содержанию



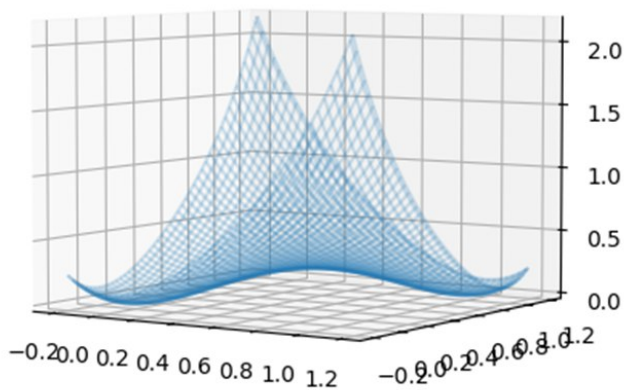
Заметим, что метод с хорошей точностью ( $\pm 0.1$  в среднем) находит точку минимума, несмотря на то, что метод сильно зависит от случайности.

К содержанию

### 1.3 Сравните его эффективность на методах и примерах из лаб. 1 и/или лаб 2

Рассмотрим градиентный спуск и метод имитации отжига. Одним из главных преимуществ метода имитации отжига перед градиентным спуском является то, что он не “застревает” в локальном минимуме в отличие от второго. Попробуем подтвердить этот факт эксперимента Рассмотрим функцию

$$(x^2 + y^2) \cdot ((x - 1)^2 + (y - 1)^2)$$



Очевидно, она имеет два глобальных минимума в точках  $(0, 0)$  и  $(1, 1)$  с шагом функции 0.1 и шагом температуры 0.001. Запустим 10 раз метод отжига из точки  $(0, 0)$

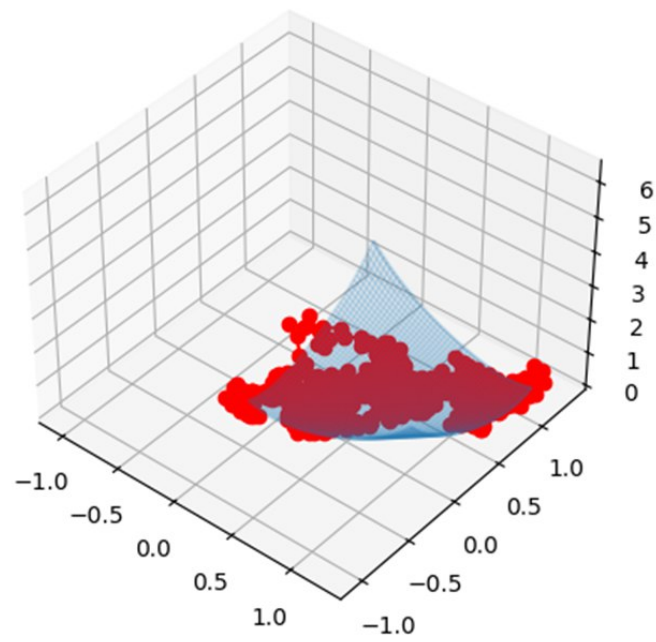
**Получились следующие результаты**

| Номер запуска | Найденный минимум | Отклонение значения от 0 |
|---------------|-------------------|--------------------------|
| 1             | $(0, 0)$          | 0.000526                 |
| 2             | $(0, 0)$          | 0.000481                 |
| 3             | $(1, 1)$          | 0.001009                 |
| 4             | $(1, 1)$          | 0.000792                 |
| 5             | $(1, 1)$          | 0.000148                 |
| 6             | $(0, 0)$          | 0.000531                 |
| 7             | $(0, 0)$          | 0.000224                 |
| 8             | $(0, 0)$          | 0.000042                 |
| 9             | $(0, 0)$          | 0.000370                 |
| 10            | $(0, 0)$          | 0.000493                 |

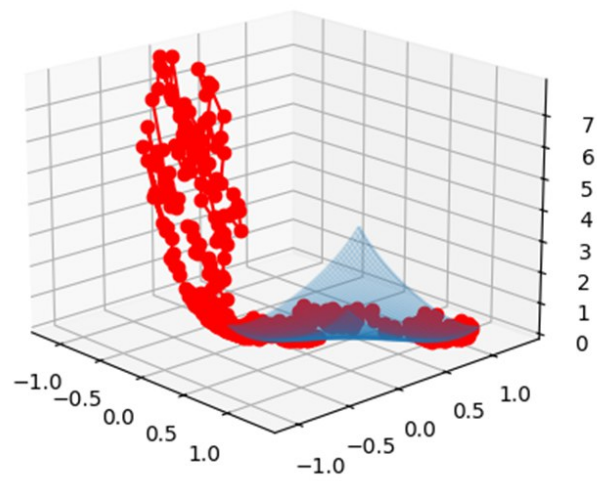
**К содержанию**



Третий запуск

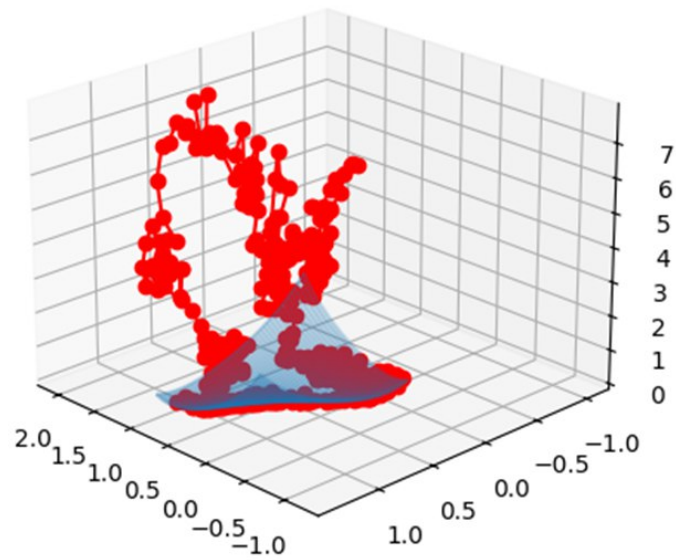


Четвертый запуск

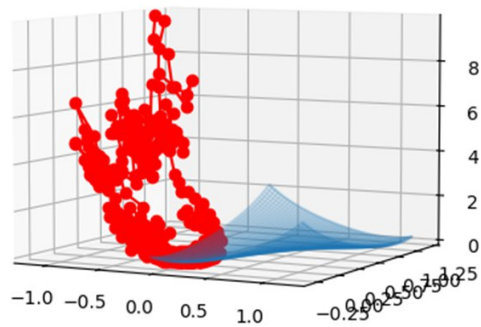


К содержанию

### Шестой запуск



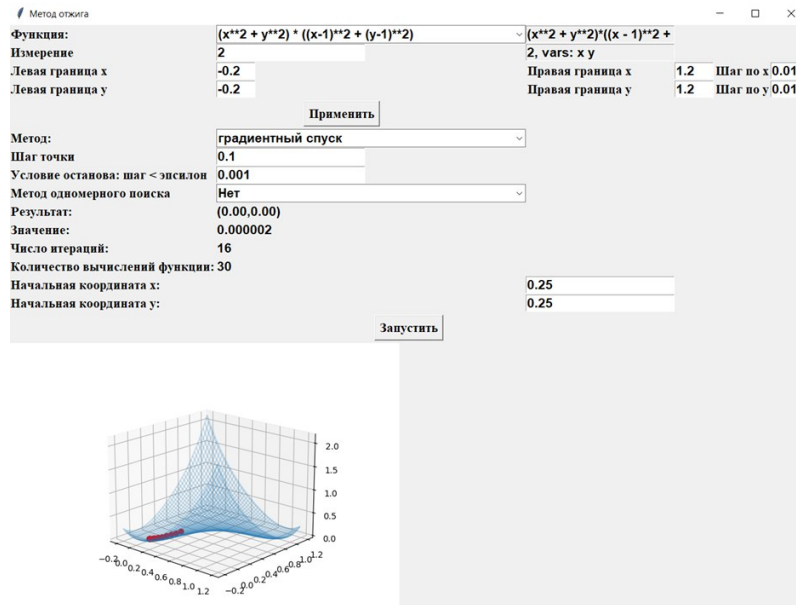
### Девятый запуск



К содержанию

Исходя из полученных данных можно сделать вывод о том, что даже если запустить метод имитации отжига из экстремума, то он может найти другой экстремум притом нередко. Также можно увидеть, что выбранный методом экстремум практически никак не влияет на точность ответа.

Напротив, градиентный спуск, запущенный из любой точки “ямы” вокруг (0, 0) придет в (0,0), что ожидаемо и этот процесс невозможно контролировать. Из преимуществ метода градиентного спуска очевидна скорость работы: за меньшее количество вычислений функции(30 против 1000) он находит минимум.



Также стоит отметить, что одновременно недостатком и преимуществом данного метода является его гибкость. Это является недостатком, так как его сложно внедрить в проекты, где требуется точность. С другой стороны, это дает огромное преимущество при:

1. Поиске глобальных минимумов
2. Изучения функций, на которых другие методы оптимизации расходятся

## К содержанию

## 2 Дополнительное задание 1

Гиперпараметры — это характеристики модели, которые фиксируются до начала обучения (например - глубина решающего дерева, значение силы регуляризации в линейной модели, learning rate для градиентного спуска). Гиперпараметры, в отличие от параметров задаются разработчиком модели перед ее обучением, в свою очередь параметры модели настраиваются в процессе обучения модели на данных.

Optuna — это фреймворк для для автоматизированного поиска оптимальных гиперпараметров для моделей машинного обучения. Она подбирает эти параметры методом проб и ошибок.

Рассмотрим применение библиотеки optuna для минимизации значений функции. Мы внедрили поиск минимума с помощью этой библиотеки в приложение

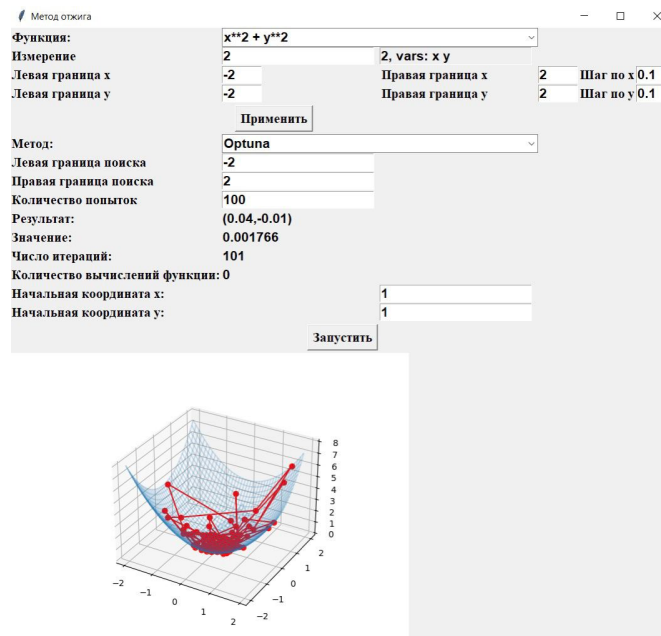
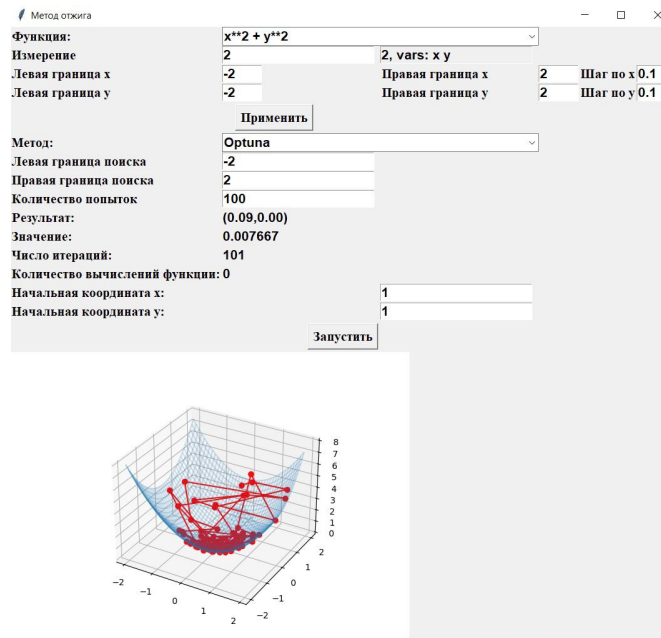
```
1 def objective(trial):
2     return function(*(trial.suggest_float(i, lb, rb) for i in v))
3
4 study = optuna.create_study()
5 study.optimize(objective, n_trials=trials)
6
7 points = []
8 values = []
9 for trial in study.trials:
10     points.append(np.array([trial.params[i] for i in v]))
11     values.append(trial.value)
12
13 res = np.array(list(study.best_params.values()))
14 points.append(res)
15 values.append(function(*res))
16
```

Данный код состоит из 4 частей:

1. Объявляем функцию чтобы оптимизировать
2. Обучение
3. Извлекаем попытки
4. Извлекаем результат

К содержанию

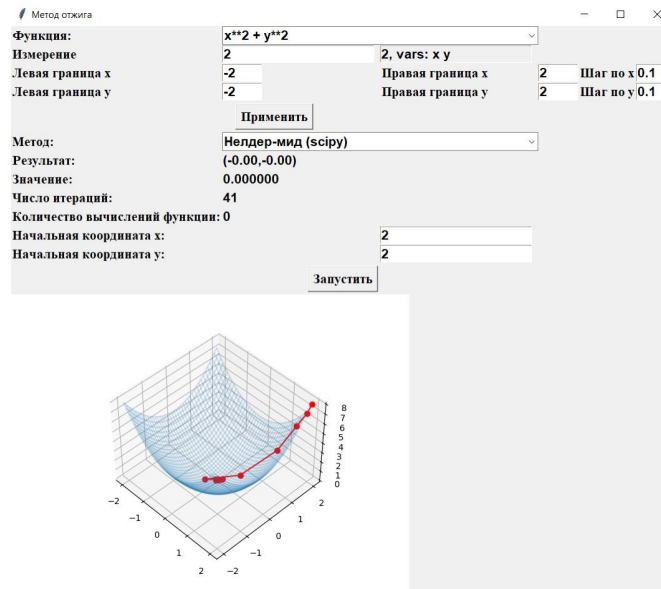
## Рассмотрим 2 попытки:



Значение гиперпараметров не менялось, но поведение поиска изменилось, отсюда вывод:  
В поиске используется случайность

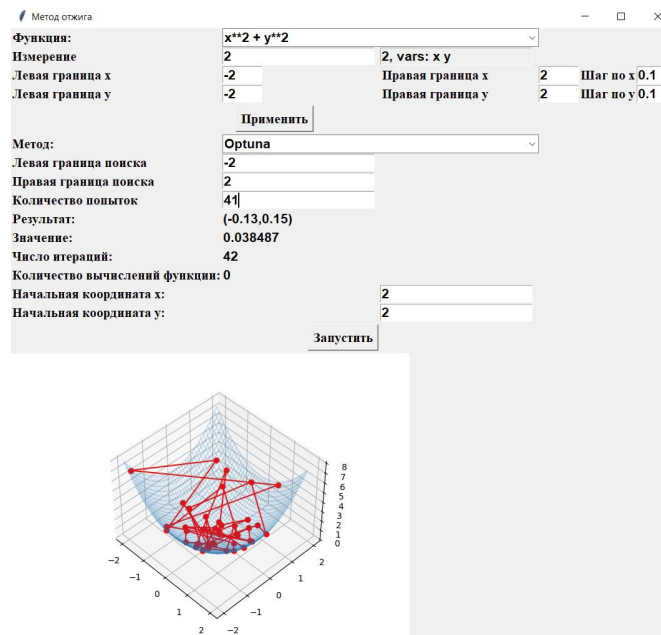
## К содержанию

Сравним этот метод с уже изученными



Метод Нелдера-Мида, запущенный из точки (2, 2) нашел минимум за 41 итерацию.

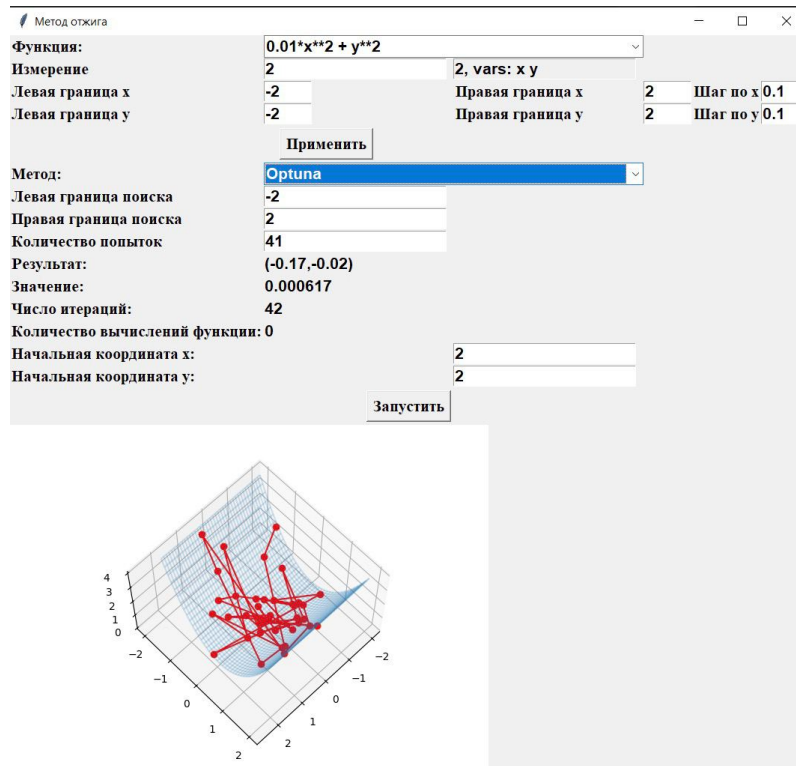
Теперь запустим обучение оптуны несколько раз с числом попыток 41.



Среднее значение минимума после 50 попыток получилось 0.0219. Отклонение от желаемого значения относительно велико, поэтому даже Нелдер-мид, метод одномерного поиска выигрывает

К содержанию

На функции  $0.01x^2 + y^2$  ортуна показала среднюю точность лучше - 0.0037



К содержанию

### 3 Дополнительное задание 2

С помощью optuna попытаемся найти `learning_rate`, при котором количество итераций градиентного спуска будет минимально. Для этого создаем такую функцию

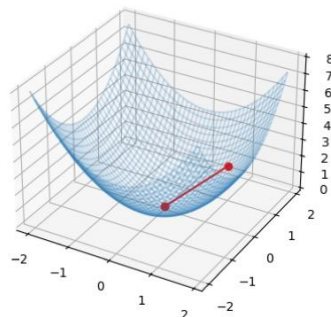
```
1 def objective_gd(trial):
2     step_entry.delete(first=0, tk.END)
3     step_entry.insert(index=0, trial.suggest_float("step", lb, rb))
4     gradient_descent()
5     if distance(points[-1], np.array([0] * dimension)) < float(epsilon_entry.get()):
6         return len(points)
7     else:
8         return ERROR # не нашел
```

```
1 study.optimize(objective_gd, n_trials=trials)
2 step_entry.delete(first=0, tk.END)
3 step_entry.insert(index=0, np.array(list(study.best_params.values()))[0])
4 gradient_descent()
```

После обучения оптимизировать эту функцию в параметре `step` для градиентного спуска будет записан лучший шаг. И действительно, при полученном шаге градиентный спуск находит минимум всего за 4 шага

|                       |                |
|-----------------------|----------------|
| Метод:                | Optuna         |
| Левая граница поиска  | 0              |
| Правая граница поиска | 1              |
| Количество попыток    | 100            |
| Объект изучения       | гиперпараметры |

|  |                    |
|--|--------------------|
| Метод:                                   | градиентный спуск  |
| Шаг точки                                | 0.5001507101948177 |
| Условие останова: шаг < эпсилон          | 0.001              |
| Метод одномерного поиска                 | Нет                |
| Результат:                               | (0.00,0.00)        |
| Значение:                                | 0.000000           |
| Число итераций:                          | 3                  |
| Количество вычислений функции:           | 4                  |
| Начальная координата x:                  | 1                  |
| Начальная координата y:                  | 1                  |
| <input type="button" value="Запустить"/> |                    |



К содержанию



| № эксперимента | 1000 попыток | 100 попыток | 10 попыток |
|----------------|--------------|-------------|------------|
| 1              | 3            | 4           | 5          |
| 2              | 3            | 3           | 4          |
| 3              | 3            | 4           | 7          |
| 4              | 3            | 4           | 4          |
| 5              | 3            | 3           | 4          |
| 6              | 3            | 4           | 6          |
| 7              | 3            | 3           | 6          |
| 8              | 3            | 3           | 5          |
| 9              | 3            | 3           | 6          |
| 10             | 3            | 4           | 6          |

Из таблицы видно, что при 1000 попытках optuna всегда находит оптимальное значение. При 100 попытках с большим шансом найденное значение оказывается оптимальным. При 10 попытках оптимальность ожидает желать лучшего

### **Вывод:**

С помощью этой библиотеки мы смогли вычислить оптимальный `learning_rate` для градиентного спуска, что показывает силу этой библиотеки, но еще далеко не всю ее мощь

## **К содержанию**