Домашнее задание 1.

Присылайте .pdf файл, сохраненный из этого ноутбука (можно просто нажать `Ctrl + P`) на `homework@merkulov.top`.

Дедлайн: 28.03.21 23:59:59

Convergence speed

1. Show with the definition that the sequence $\left\{\frac{1}{k}\right\}_{k=1}^{\infty}$ does not have a linear convergence rate (but it converges to zero).

Покажем это от протвного: пусть линейная сходимость есть. По её определению выполнено:

$$|x_{k+1}| < q * |x_k|$$

Тогда для нашей последовательности

$$\frac{1}{k+1} \leqslant q \cdot \frac{1}{k} \Rightarrow \frac{k}{k+1} \leq q$$

Получаем, что $q \geqslant 1$ при $k \to \infty$, но $q \in (0, 1)$ по определению. Получили противоречие, что есть линейная сходимость. То что последовательность сходится к 0 - очевидно из школьной математики (и этот факт уже использовал для проверки определения)

1. Determine the convergence or divergence of a given sequence $r_k = 0.707^k$.

Проерим определение линейной сходимости к 0, которое записали в предыдущем пункте, получим:

$$\left\|0.707^{k+1}\right\| \leq 0.707 * q^k \text{ , где } q = 0.707$$

Значит, по определению такая последовательность сходится к 0 линейно

1. Determine the convergence or divergence of a given sequence $r_k = 0.707^{2^k}$.

Тут проверим определение квадратичной сходимости к 0:

$$\left\| 0.707^{2^{k+1}} \right\| \leq 0.707^2 * q^{2^k}, \text{ где } q = 0.707$$

Выполнено! Поэтому эта последовательность обладает квадратичной скоростью сходимости

1. Determine the convergence or divergence of a given sequence $r_k = \frac{1}{k!}$.

Заметим, что данная последовательность строго положительных чисел, сходящаяся к нулю, значит, можем применить ratio test. Подсчитаем α

$$\alpha = \lim_{k \to \infty} \frac{r_{k+1}}{r_k} = \lim_{k \to \infty} \frac{k!}{(k+1)!} = \lim_{k \to \infty} \frac{1}{k+1} = 0$$

если $\alpha = 0$, то $\{r_k\}_{k=m}^{\infty}$ имеет сверхлинейную сходимость

1. Determine the convergence or divergence of a given sequence $r_k = \begin{cases} \frac{1}{k}, & \text{if } k \text{ is even} \\ \frac{1}{k^2}, & \text{if } k \text{ is odd} \end{cases}$.

В последнем примере применим root test для определения скорости сходимость (наша последовательность - последовательность неотрицательных чисел, сходящаяся к нулю)

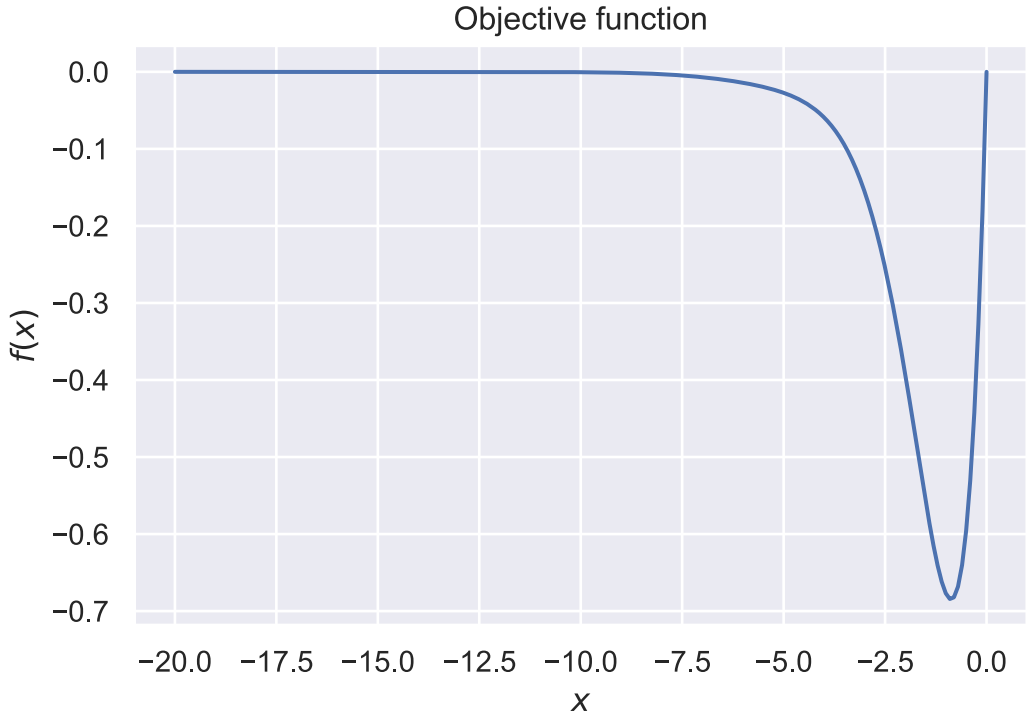
$$\alpha = \lim_{k \to \infty} \sup r_k^{1/k} = \lim_{k \to \infty} \frac{1}{k^{1/k}} = \lim_{k \to \infty} e^{\frac{\log k}{k}} = e^{\lim\limits_{k \to \infty} \frac{\log k}{k}} = e^0 = 1$$

Ели $\alpha = 1$, то $\{r_k\}_{k=m}^{\infty}$ имеет сублинейную сходимость.

Line search

1. Consider the function

$$f(x) = (x + \sin x)e^x, \quad x \in [-20, 0]$$

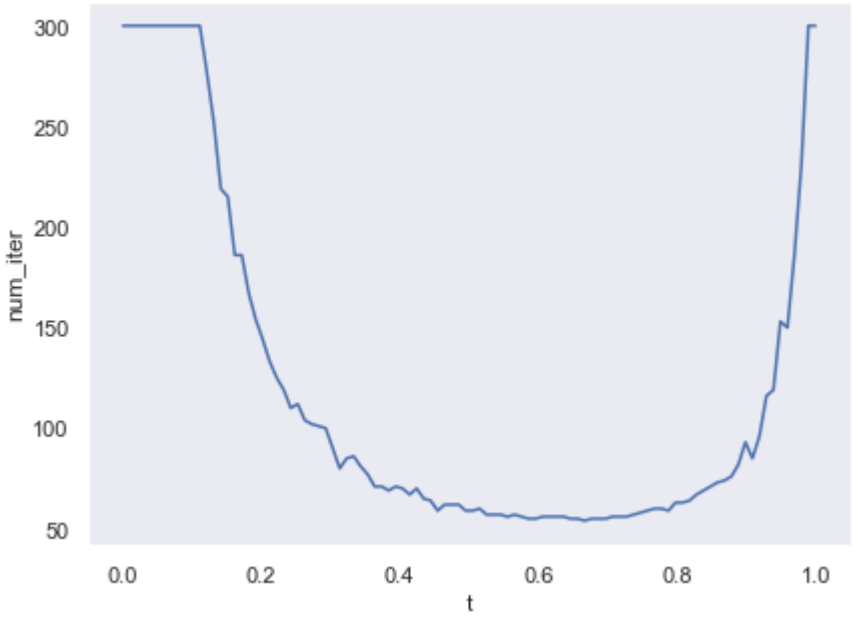


Consider the following modification of solution localization method, in which the interval $[a, b]$ is divided into 2 parts in a fixed proportion of t : $x_t = a + t * (b - a)$ (maximum twice at iteration - as in the dichotomy method). Experiment with different values of $t \in [0, 1]$ and plot the dependence of $N(t)$ - the number of iterations needed to achieve ε - accuracy from the t parameter. Consider $\varepsilon = 10^{-7}$. Note that with $t = 0.5$ this method is exactly the same as the dichotomy method.

```
import numpy as np
from matplotlib import pyplot as plt
import math


def my_function(x):
    return (x + np.sin(x)) * np.exp(x)


def line_search(fun, t, x_a = -20, x_b = 0, tol = 10**(-7), maxfev=None,
```

```
            maxiter=300, **options):
        x_best = x_a
        funcalls = 1
        niter = 0
        stop = False
        success = False

        x_c = x_a + t * (x_b - x_a)
        while not stop and niter < maxiter:
            niter += 1
            x_y = x_a + t * (x_c - x_a)
            funcalls += 1
            if fun(x_y) <= fun(x_c):
                x_b = x_c
                x_c = x_y
            else:
                x_z = x_b + t * (x_b - x_c)
                funcalls += 1
                if fun(x_z) <= fun(x_c):
                    x_a = x_c
                    x_c = x_z
                else:
                    x_a = x_y
                    x_b = x_z
                    x_c = x_a + t * (x_b - x_a)

            x_best = x_c
            f_best = fun(x_c)


            if abs(x_b - x_a) < tol:
                success = True

            if (maxfev is not None and funcalls >= maxfev) or success:
                stop = True
                break


        return x_best, niter
```

```
data = []
X = []
t_iter = np.linspace(0, 1, 100)
for t in t_iter:
    x_best, n_iter = line_search(my_function, t)
    data.append(n_iter)
    X.append(x_best)
data[0] = 300
fig = plt.figure(figsize = (15,5))
ax1 = plt.subplot(121)
ax1.plot(t_iter, data)
ax1.set_ylabel('num_iter')
ax1.set_xlabel('t')
ax1.grid()
```



При t = 0 наш алгоритм никуда не сдвинется (вернее он сразу выйдет, так как x_b = x_a: войдёт в первую ветку, так как

```
my_function(-20), my_function(0)
```

```
(-4.3104792859396486e-08, 0.0)
```

Чтобы узнать при каком t достигается минимальное количество итераций, заполним это значение 300 (что соответсвует максимальному числу итераций) и найдём минимум в массиве, а затем получим, какой t дал этот минимум

```
data[0] = 200
min_iter = min(data)
min_iter
```

```
54
```

```
t_iter[data.index(min_iter)]
```

```
0.6666666666666667
```

```
X[data.index(min_iter)]
```

```
-0.8743578493427908
```

Заметим, что для даннной функции оптимальный шаг по числу итераций (не по числу вызовов функции!) равняется 0.67, а не 0.5 как в методе дихотомии (при t = 0.5 мы находим оптимум за 59 итераций, когда оптимальное 54)

```
data[50]
```

```
59
```

Это можно объяснить тем, что минимум функции находится возле правой границы, поэтому делить побольше, чем половина - выгоднее

Zero order optimization

Global optimization with scipy

Implement Rastrigin function $f : \mathbb{R}^d \to \mathbb{R}$ for d = 10. link

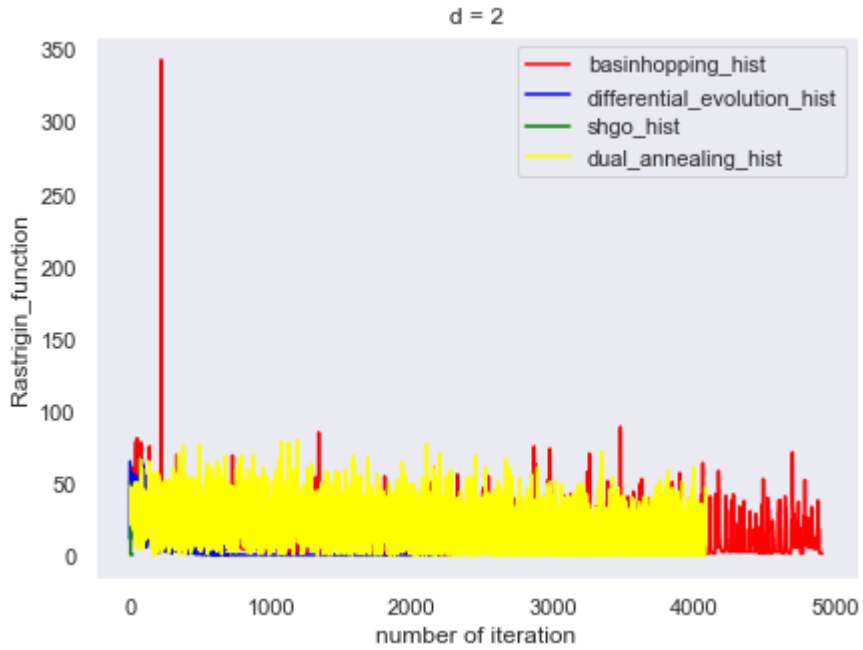
$$f(\mathbf{x}) = 10d + \sum_{i=1}^{d} \left[x_i^2 - 10\cos(2\pi x_i) \right]$$

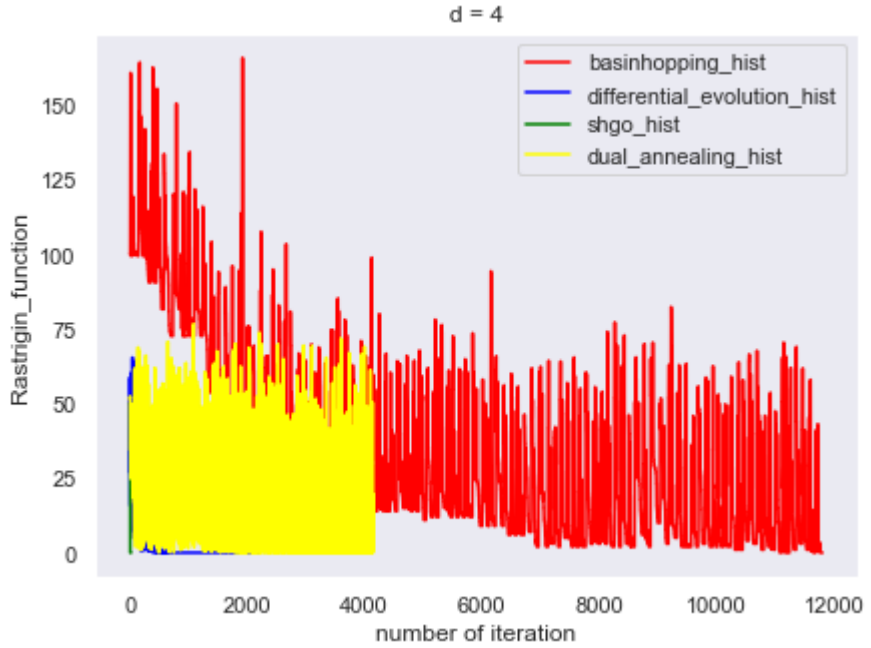
- Consider global optimization from here.
- Plot 4 graphs for different d from {2,4,8,16,32}. On each graph you are to plot f from N_{fev} for 5 methods: `basinhopping`, `brute`, `differential_evolution`, `shgo`, `dual_annealing` from scipy, where N_{fev} - the number of function evaluations. This information is usually avalable from `specific_optimizer.nfev`. If you will need bounds for the optimizer, use $x_i \in [-5, 5]$.

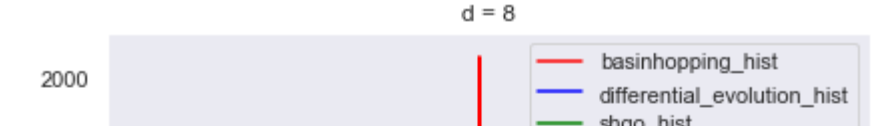
```
import scipy.optimize as opt


def Rastrigin_function(x):
    d = len(x)
    global hist
    res = 10*d
    for el in x:
        res += el**2 - 10* np.cos(2*np.pi*el)
    hist.append(res)
    return res


def make_hist(d):
    basinhopping_hist = []
    brute_hist = []
    differential_evolution_hist = []
    shgo_hist = []
    dual_annealing_hist = []
```

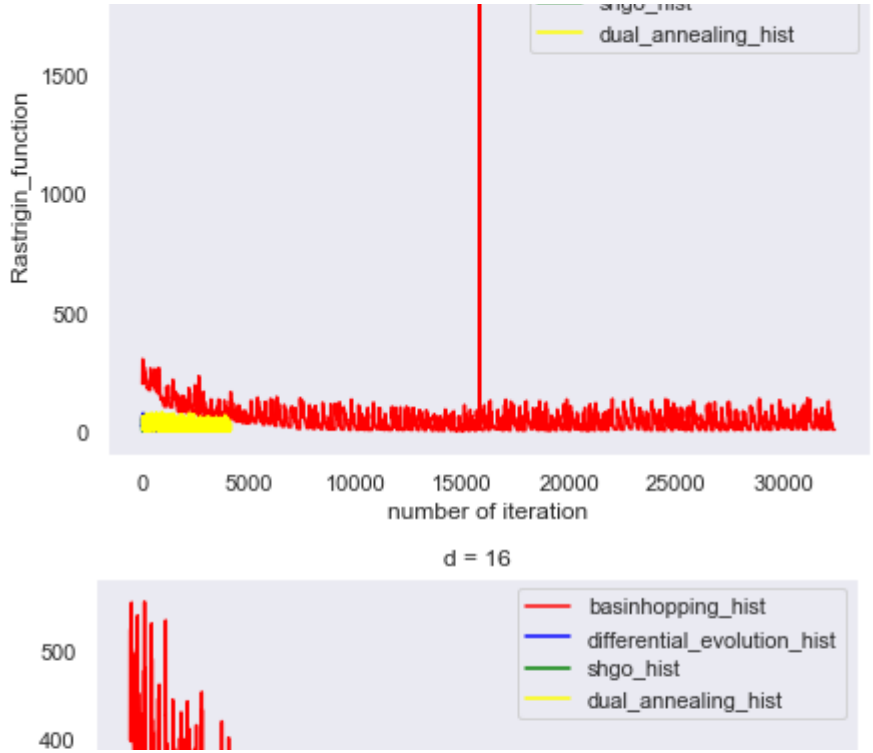
```
    x0 = [-5.0] * d
    bounds=tuple([(-5, 5) for i in range(d)])

    global hist
    hist = []
    ret = opt.basinhopping(Rastrigin_function, x0)
    # print(ret.fun)
    basinhopping_hist = hist

    # ret = opt.brute(Rastrigin_function)

    hist = []
    ret = opt.differential_evolution(Rastrigin_function, bounds=((-5, 5), (-5, 5)))
    # print(ret.fun)
    differential_evolution_hist = hist

    hist = []
    ret = opt.shgo(Rastrigin_function, bounds=((-5, 5), (-5, 5)))
    # print(ret.fun)
    shgo_hist = hist

    hist = []
    ret = opt.dual_annealing(Rastrigin_function, bounds=((-5, 5), (-5, 5)))
    # # print(ret.fun)
    dual_annealing_hist = hist

    return basinhopping_hist, differential_evolution_hist, shgo_hist, dual_annealing_hist


def plot_graphic(d):
    basinhopping_hist, differential_evolution_hist, shgo_hist, dual_annealing_hist = make_hist(d)
    fig = plt.figure(figsize = (15,5))
    ax = plt.subplot(121)
    ax.plot(range(len(basinhopping_hist)), basinhopping_hist, label="basinhopping_hist", color="red")
    ax.plot(range(len(differential_evolution_hist)), differential_evolution_hist, label="differential_evolution_hist", color=
    ax.plot(range(len(shgo_hist)), shgo_hist, label="shgo_hist", color="green")
    ax.plot(range(len(dual_annealing_hist)), dual_annealing_hist, label="dual_annealing_hist", color="yellow")
    ax.set_ylabel('Rastrigin_function')
```

```
        ax.set_xlabel('number of iteration')
        ax.grid()
        ax.legend(loc = 'best')
        ax.set_title(f'd = {d}')


    for d in [2,4,8,16,32]:
        plot_graphic(d)
```


d = 2
basinhopping_hist
differential_evolution_hist
shgo_hist
dual_annealing_hist
Rastrigin_function
number of iteration
0
50
100
150
200
250
300
350
0
1000
2000
3000
4000
5000

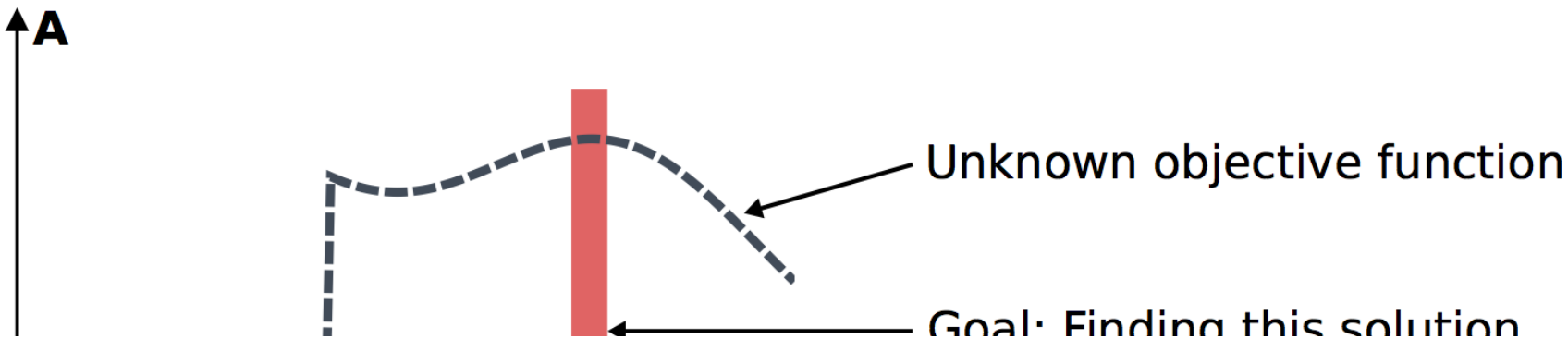

d = 4
basinhopping_hist
differential_evolution_hist
shgo_hist
dual_annealing_hist
Rastrigin_function
number of iteration
0
25
50
75
100
125
150
0
2000
4000
6000
8000
10000
12000

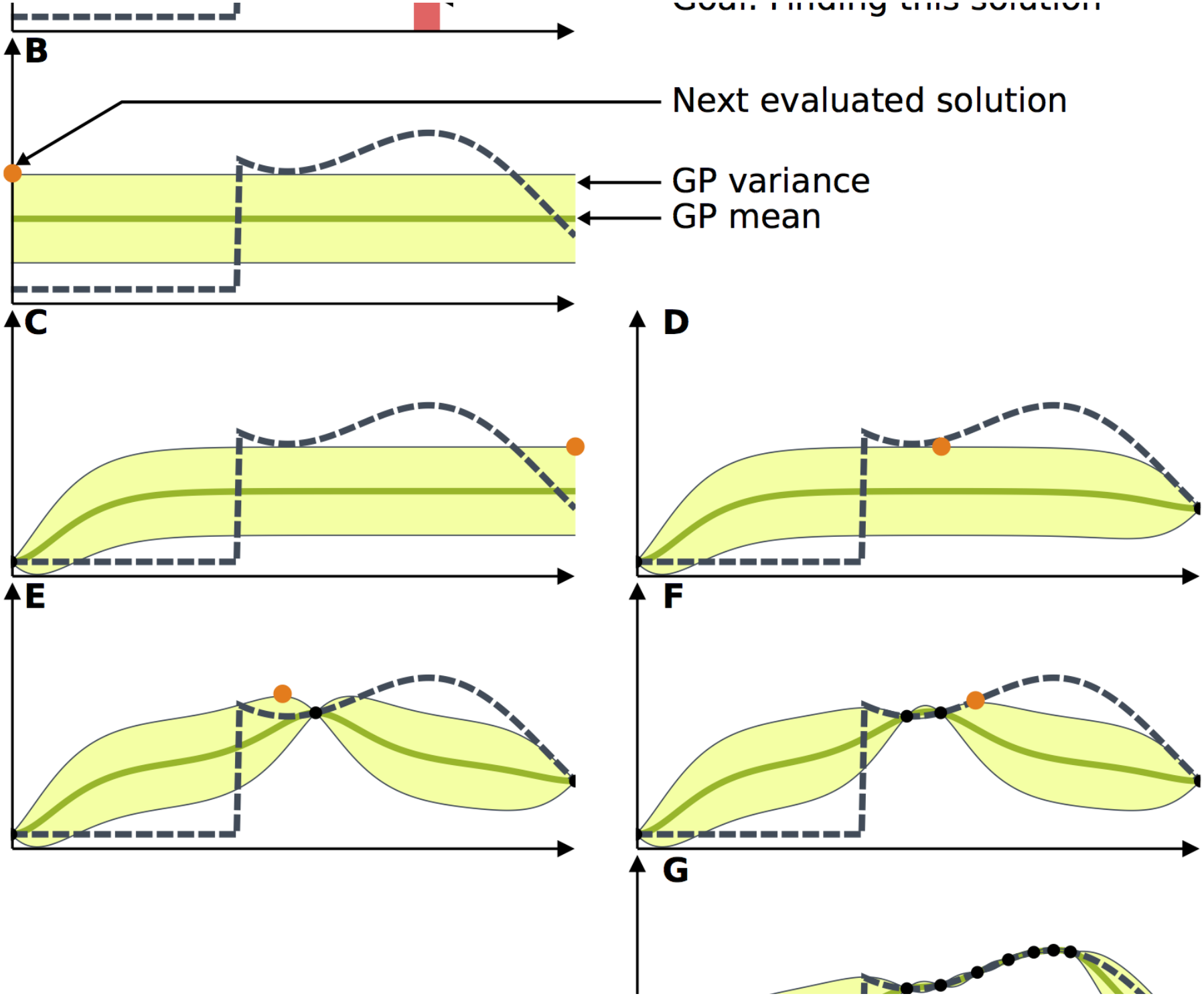

d = 8
basinhopping_hist
differential_evolution_hist
2000



Hyperparameter search with optuna

Machine learning models often have hyperparameters. To choose optimal one between them one can use GridSearch or RandomSearch. But these algorithms computationally uneffective and don't use any sort of information about type of optimized function. To overcome this problem one can use [bayesian optimization]. Using this method we optimize our model by sequentially chosing points based on prior information about function.




B
Next evaluated solution
GP variance
GP mean
C
D
E
F
G

.

In this task you will use optuna package for hyperparameter optimization RandomForestClassifier. Your task is to find best Random Forest model varying at least 3 hyperparameters on iris dataset. Examples can be find here or here

```
!pip install optuna
```

ПОКАЗАТЬ СКРЫТЫЕ ВЫХОДНЫЕ ДАННЫЕ

```
import sklearn.datasets
import sklearn.ensemble
import sklearn.model_selection
import sklearn.svm

import optuna

iris = sklearn.datasets.load_iris()
x, y = iris.data, iris.target
```

```
print(x.shape, y.shape)
```

```
(150, 4) (150,)
```

```
# Here is the example of objective function, which depends on two variables (n_estimators, max_depth). But you'll need to ch

def objective():
    iris = sklearn.datasets.load_iris()  # Prepare the data.

    clf = sklearn.ensemble.RandomForestClassifier(
        n_estimators=5, max_depth=3)  # Define the model.
```

```
    return sklearn.model_selection.cross_val_score(
        clf, iris.data, iris.target, n_jobs=-1, cv=3).mean()  # Train and evaluate the model.

print('Accuracy: {}'.format(objective()))
```

```
 Accuracy: 0.9538398692810457
```

```
# добавил ещё параметр criterion (как мы выбираем порого в дереве: джини или энтропия)
def objective(trial):
    iris = sklearn.datasets.load_iris()

    n_estimators = trial.suggest_int('n_estimators', 2, 25)
    max_depth = int(trial.suggest_int('max_depth', 1, 32, log=True))
    criterion = trial.suggest_categorical('criterion',{'gini', 'entropy'})

    clf = sklearn.ensemble.RandomForestClassifier(
        n_estimators=n_estimators, max_depth=max_depth, criterion=criterion)

    return sklearn.model_selection.cross_val_score(
        clf, iris.data, iris.target, n_jobs=-1, cv=3).mean()

study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100)

trial = study.best_trial

print('Accuracy: {}'.format(trial.value))
print("Best hyperparameters: {}".format(trial.params))
```

```
[I 2021-03-25 11:19:58,180] A new study created in memory with name: no-name-730182d8-f01e-4273-be55-e59e7b94f0e1
[I 2021-03-25 11:19:59,300] Trial 0 finished with value: 0.8688725490196078 and parameters: {'n_estimators': 11, 'ma
[I 2021-03-25 11:20:00,311] Trial 1 finished with value: 0.9538398692810457 and parameters: {'n_estimators': 5, 'max
[I 2021-03-25 11:20:00,340] Trial 2 finished with value: 0.861111111111111 and parameters: {'n_estimators': 13, 'max
[I 2021-03-25 11:20:00,359] Trial 3 finished with value: 0.9534313725490197 and parameters: {'n_estimators': 7, 'max
[I 2021-03-25 11:20:00,376] Trial 4 finished with value: 0.9538398692810457 and parameters: {'n_estimators': 5, 'max
[I 2021-03-25 11:20:00,403] Trial 5 finished with value: 0.960375816993464 and parameters: {'n_estimators': 16, 'max
[I 2021-03-25 11:20:00,426] Trial 6 finished with value: 0.9673202614379085 and parameters: {'n_estimators': 13, 'ma
[I 2021-03-25 11:20:00,441] Trial 7 finished with value: 0.9538398692810457 and parameters: {'n_estimators': 2, 'max
```

```
[I 2021-03-25 11:20:00,457] Trial 8 finished with value: 0.7581699346405228 and parameters: {'n_estimators': 3, 'max
[I 2021-03-25 11:20:00,488] Trial 9 finished with value: 0.9538398692810457 and parameters: {'n_estimators': 17, 'ma
[I 2021-03-25 11:20:00,540] Trial 10 finished with value: 0.954248366013072 and parameters: {'n_estimators': 25, 'ma
[I 2021-03-25 11:20:00,577] Trial 11 finished with value: 0.9607843137254902 and parameters: {'n_estimators': 19, 'm
[I 2021-03-25 11:20:00,618] Trial 12 finished with value: 0.9738562091503268 and parameters: {'n_estimators': 22, 'm
[I 2021-03-25 11:20:00,659] Trial 13 finished with value: 0.9607843137254902 and parameters: {'n_estimators': 23, 'm
[I 2021-03-25 11:20:00,696] Trial 14 finished with value: 0.9468954248366014 and parameters: {'n_estimators': 21, 'm
[I 2021-03-25 11:20:00,720] Trial 15 finished with value: 0.960375816993464 and parameters: {'n_estimators': 10, 'ma
[I 2021-03-25 11:20:00,756] Trial 16 finished with value: 0.9669117647058822 and parameters: {'n_estimators': 15, 'm
[I 2021-03-25 11:20:00,790] Trial 17 finished with value: 0.9673202614379085 and parameters: {'n_estimators': 19, 'm
[I 2021-03-25 11:20:00,834] Trial 18 finished with value: 0.9599673202614379 and parameters: {'n_estimators': 25, 'm
[I 2021-03-25 11:20:00,858] Trial 19 finished with value: 0.8431372549019608 and parameters: {'n_estimators': 12, 'm
[I 2021-03-25 11:20:00,894] Trial 20 finished with value: 0.9607843137254902 and parameters: {'n_estimators': 20, 'm
[I 2021-03-25 11:20:00,922] Trial 21 finished with value: 0.9538398692810457 and parameters: {'n_estimators': 18, 'm
[I 2021-03-25 11:20:00,962] Trial 22 finished with value: 0.9669117647058822 and parameters: {'n_estimators': 22, 'm
[I 2021-03-25 11:20:01,001] Trial 23 finished with value: 0.960375816993464 and parameters: {'n_estimators': 24, 'ma
[I 2021-03-25 11:20:01,032] Trial 24 finished with value: 0.9673202614379085 and parameters: {'n_estimators': 15, 'm
[I 2021-03-25 11:20:01,059] Trial 25 finished with value: 0.9468954248366014 and parameters: {'n_estimators': 10, 'm
[I 2021-03-25 11:20:01,096] Trial 26 finished with value: 0.9673202614379085 and parameters: {'n_estimators': 19, 'm
[I 2021-03-25 11:20:01,128] Trial 27 finished with value: 0.9669117647058822 and parameters: {'n_estimators': 15, 'm
[I 2021-03-25 11:20:01,169] Trial 28 finished with value: 0.9669117647058822 and parameters: {'n_estimators': 21, 'm
[I 2021-03-25 11:20:01,203] Trial 29 finished with value: 0.9607843137254902 and parameters: {'n_estimators': 17, 'm
[I 2021-03-25 11:20:01,241] Trial 30 finished with value: 0.960375816993464 and parameters: {'n_estimators': 19, 'ma
[I 2021-03-25 11:20:01,272] Trial 31 finished with value: 0.9538398692810457 and parameters: {'n_estimators': 14, 'm
[I 2021-03-25 11:20:01,305] Trial 32 finished with value: 0.9673202614379085 and parameters: {'n_estimators': 22, 'm
[I 2021-03-25 11:20:01,347] Trial 33 finished with value: 0.9669117647058822 and parameters: {'n_estimators': 22, 'm
[I 2021-03-25 11:20:01,371] Trial 34 finished with value: 0.9607843137254902 and parameters: {'n_estimators': 12, 'm
[I 2021-03-25 11:20:01,407] Trial 35 finished with value: 0.9738562091503268 and parameters: {'n_estimators': 20, 'm
[I 2021-03-25 11:20:01,449] Trial 36 finished with value: 0.9673202614379085 and parameters: {'n_estimators': 23, 'm
[I 2021-03-25 11:20:01,485] Trial 37 finished with value: 0.960375816993464 and parameters: {'n_estimators': 23, 'ma
[I 2021-03-25 11:20:01,524] Trial 38 finished with value: 0.9468954248366014 and parameters: {'n_estimators': 21, 'm
[I 2021-03-25 11:20:01,559] Trial 39 finished with value: 0.9673202614379085 and parameters: {'n_estimators': 20, 'm
[I 2021-03-25 11:20:01,598] Trial 40 finished with value: 0.7450980392156863 and parameters: {'n_estimators': 17, 'm
[I 2021-03-25 11:20:01,631] Trial 41 finished with value: 0.9473039215686274 and parameters: {'n_estimators': 18, 'm
[I 2021-03-25 11:20:01,666] Trial 42 finished with value: 0.9599673202614379 and parameters: {'n_estimators': 20, 'm
[I 2021-03-25 11:20:01,699] Trial 43 finished with value: 0.9468954248366014 and parameters: {'n_estimators': 24, 'm
[I 2021-03-25 11:20:01,730] Trial 44 finished with value: 0.9538398692810457 and parameters: {'n_estimators': 23, 'm
[I 2021-03-25 11:20:01,759] Trial 45 finished with value: 0.9599673202614379 and parameters: {'n_estimators': 7, 'ma
[I 2021-03-25 11:20:01,796] Trial 46 finished with value: 0.9534313725490197 and parameters: {'n_estimators': 18, 'm
[I 2021-03-25 11:20:01,833] Trial 47 finished with value: 0.9607843137254902 and parameters: {'n_estimators': 19, 'm
[I 2021-03-25 11:20:01,877] Trial 48 finished with value: 0.9538398692810457 and parameters: {'n_estimators': 25, 'm
[I 2021-03-25 11:20:01,910] Trial 49 finished with value: 0.960375816993464 and parameters: {'n_estimators': 16, 'ma
[I 2021-03-25 11:20:01,950] Trial 50 finished with value: 0.6862745098039215 and parameters: {'n_estimators': 20, 'm
```

```
[I 2021-03-25 11:20:01,982] Trial 51 finished with value: 0.954248366013072 and parameters: {'n_estimators': 13, 'ma
[I 2021-03-25 11:20:02,027] Trial 52 finished with value: 0.9669117647058822 and parameters: {'n_estimators': 21, 'm
[I 2021-03-25 11:20:02,072] Trial 53 finished with value: 0.9673202614379085 and parameters: {'n_estimators': 24, 'm
[I 2021-03-25 11:20:02,109] Trial 54 finished with value: 0.9669117647058822 and parameters: {'n_estimators': 22, 'm
[I 2021-03-25 11:20:02,158] Trial 55 finished with value: 0.960375816993464 and parameters: {'n_estimators': 23, 'ma
```

Улучшили точность предсказаний на 2.08% по сравнению с дефолтной моделью (где задали гиперпараметры наугад). Неплохой результат, так как точность уже была близка к 100%. Стоит отметить, что модель достаточно сильно увеличилась в размерах, то есть вместо 5 деревьев с глубиной 3, теперь 22 с глубиной 10

Gradient descent

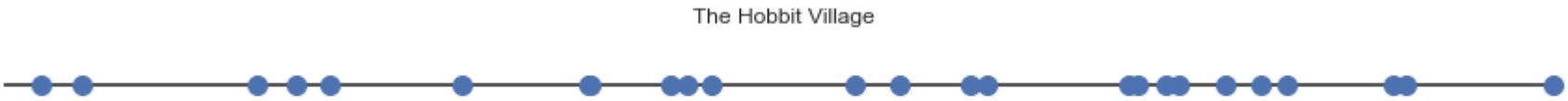
Hobbit village

```
%matplotlib inline
import numpy as np
from matplotlib import pyplot as plt
import matplotlib as mpl
import seaborn as sns
import copy
sns.set()
```

Below one can find function plotting the village

```
def plot_village(coordinates, l=1):
    # Checking, that all the coordinates are less than l
    assert (coordinates <= l).all(), 'All the houses should be in a village'

    # Draw horizontal line
    plt.hlines(0, 0, l)
    plt.xlim(0, l)
```

```
    plt.ylim(-0.5, 0.5)

    # Draw house points
    y = np.zeros(np.shape(coordinates))
    plt.title('The Hobbit Village')
    plt.plot(coordinates,y,'o',ms = 10)
    plt.axis('off')
    plt.xlabel('Coordinates')
    fig = plt.gcf()
    fig.set_size_inches(15, 1)
    plt.show()


N = 25
l = 1
x = np.random.rand(N)*l

plot_village(x, l)
```



The inhabitants of a one-dimensional village want to connect to the Internet, so they need a central service station from which a cable will stretch to all the houses in the village. Let the price of the cable to be pulled from the station to each house independently be determined by some function p(d) . Then it is clear that the village will have to pay the following amount for access to the World Wide Web:

$$P(w, x) = \sum_{i=1}^{N} p(d_i) = \sum_{i=1}^{N} p(|w - x_i|)$$

w - station location, x_i - location of i house.

Write analytical solution w^* for minimization $P(w, x)$, if $p(d) = d^2$

==YOUR ANSWER==

Запишем функцию потерь. В точке минимума градиент по ω этой функции равен 0, отсюда находим

$$P(\omega, d) = \sum_{i=1}^{N} (\omega - x_i)^2$$

$$\nabla_\omega P = 2 \sum_{i=1}^{N} (\omega - x_i) = 0$$

$$\omega^* = \frac{\sum_{i=1}^{N} X_i}{N}$$

Найдём аналитическое решение для наших данных

```
np.sum(x)/N
```

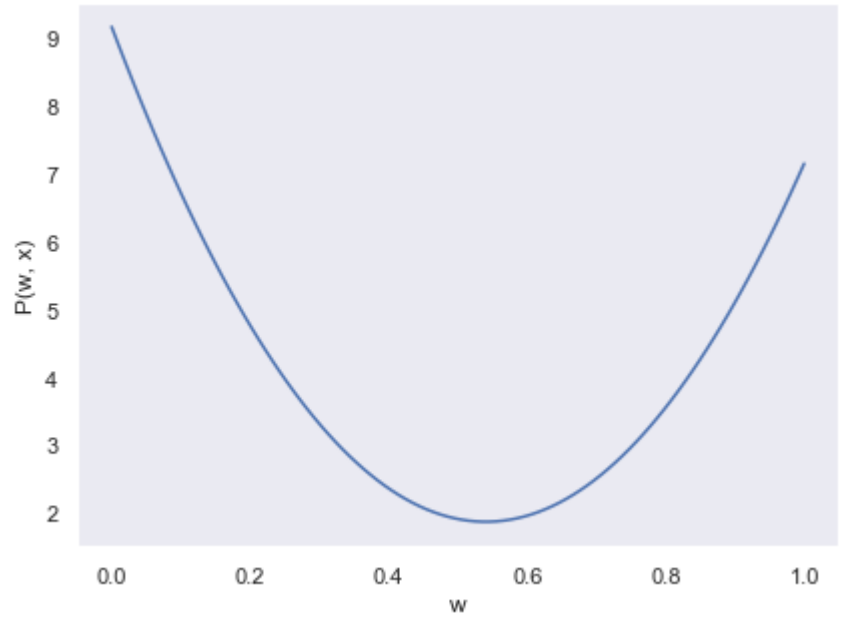
```
0.5403705292277458
```

Write loss function $P(x, w)$

```
def P(w, x):
    return np.sum((w - x)**2)
```

Plot loss function on the range $(0, l)$

```
data = []
for el in np.linspace(0, l, 100):
    data.append(P(el, x))
fig = plt.figure(figsize = (15,5))
ax = plt.subplot(121)
ax.plot(np.linspace(0, l, 100), data)
ax.set_ylabel('P(w, x)')
ax.set_xlabel('w')
ax.grid()
```



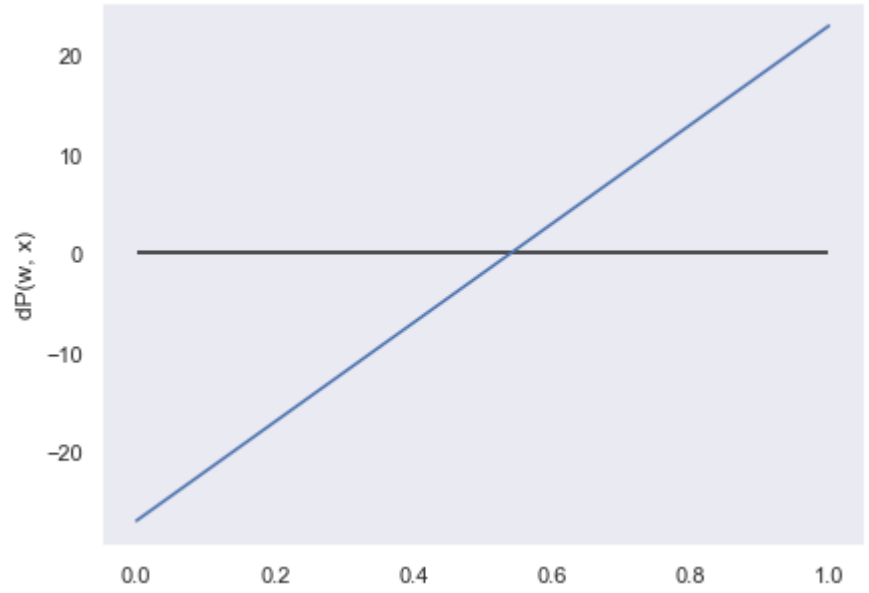
Write gradient of loss function

```
def dP(w, x):
    return 2 * np.sum(w-x)
```

Plot gradient of loss function on the range $(0, l)$. Which point on the graph is of particular interest? Why?

```
data = []
for el in np.linspace(0, 1, 100):
    data.append(dP(el, x))
fig = plt.figure(figsize = (15,5))
ax = plt.subplot(121)
ax.plot(np.linspace(0, 1, 100), data)
ax.hlines(0, 0, 1)
ax.set_ylabel('dP(w, x)')
# ax.set_xlabel('w')
```

```
ax.grid()
```



Понятно, что функция градиента - линейная. Для использования градиентного спуска нам интересно, когда градиент очень маленьки (или вооюще 0), так как для таких w шаг будет очень маленьким и спуск не будет происходить. Понятно, что это будет происходить близко к оптимальной точке

Write function `gradient_descent`, which returns w_k after a fixed number of steps.

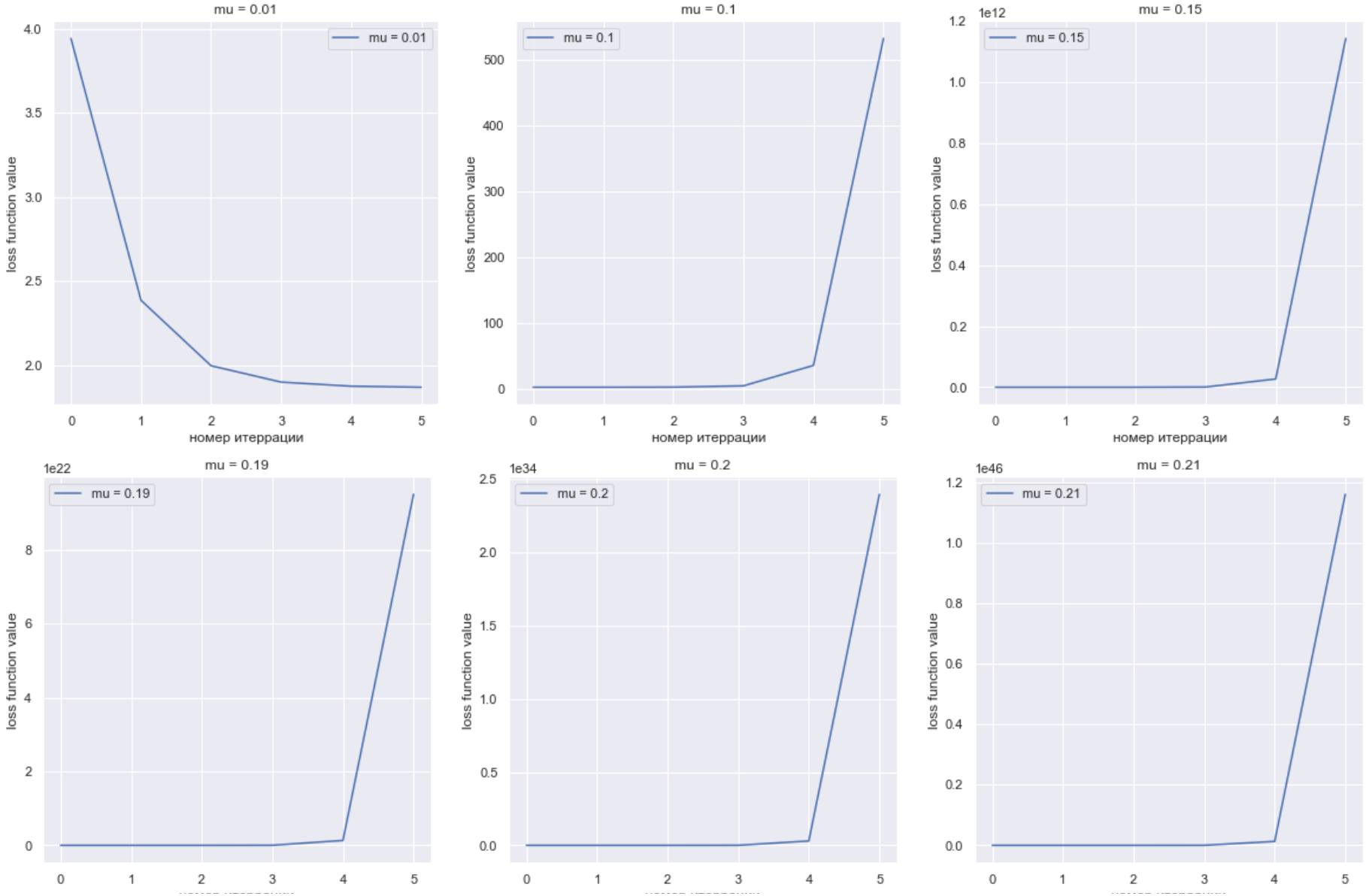
$$w_{k+1} = w_k - \mu \nabla P(w_k)$$

```
def gradient_descent(x, dP, w0, mu, Nsteps):
    for i in range(Nsteps):
        w0 -= mu * dP(w0, x)
    return w0
```

Modify `gradient_descent` to return all optimization trajectory. Plot loss function trajectory for the following learning rates $\mu = 0.01, 0.1, 0.15, 0.19, 0.20, 0.21.$

Draw conclusions.

```
def gradient_descent(x, dP, w0, mu, Nsteps):
    history = []
    for i in range(Nsteps):
        history.append(P(w0, x))
        w0 -= mu * dP(w0, x)
    return history, w0


mu = [0.01,0.1,0.15,0.19,0.20,0.21]
N = 6
w_init =np.random.rand(1)
f, axes = plt.subplots(1, 3, figsize=(20, 6))
for i in range(3):
    axes[i].plot(range(N), gradient_descent(x, dP, w_init , mu[i], N)[0], label=f'mu = {mu[i]}')
    axes[i].legend(loc = 'best')
    axes[i].set_xlabel('номер итеррации')
    axes[i].set_ylabel('loss function value')
    axes[i].set_title(f'mu = {mu[i]}')

f, axes = plt.subplots(1, 3, figsize=(20, 6))
for i in range(3):
    axes[i].plot(range(N), gradient_descent(x, dP, w_init, mu[i+3], N)[0], label=f'mu = {mu[i+3]}')
    axes[i].legend(loc = 'best')
    axes[i].set_xlabel('номер итеррации')
    axes[i].set_ylabel('loss function value')
    axes[i].set_title(f'mu = {mu[i+3]}')
```

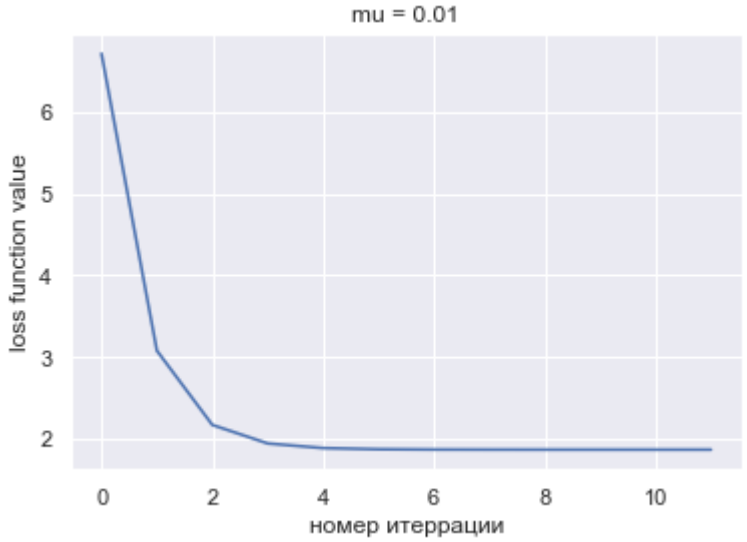


Видим, что только при одном значении $\mu = 0.01$ наш градиентный спуск сходится, а при всех остальных он расходится. Это можно объяснить тем, что шаг градиентного спуска в остальных случаях очень большой для данной задачи и мы проскакиваем минимум и начинаем расходиться и идти вверх по параболе (функция, которую мы минимизируем - парабола), а не вниз к минимуму. То есть каждый шаг мы попадаем на другую ветку параболы при этом увеличивая значения лосс функции

Проверим, что для $\mu = 0.01$ сходится к аналитическому решению

```
N = 12
hist, w_best = gradient_descent(x, dP, 0.1 , 0.01, N)
plt.plot(range(N), hist)
plt.xlabel('номер итеррации')
plt.ylabel('loss function value')
plt.title(f'mu = {0.01}')
print(f"w* = {w_best}")
```

```
w* = 0.5402630168915085
```



Видим, что через 12 шагов решение совпадает с аналитическим с точность до 3го знака. При этом мы приближаемся очень точно, несмотря на то, что не меняем шаг, так как градиент при приближении к точке минимума становится меньше (отмечал это выше)

The village decided to lay cable using new technology. That's why the price of the cable changed to:

$$p(d) = |d|$$

Write new function `P, dP`. Plot graphs for various x and w.

```
def P(w, x):
    Loss = 0
    for  el in x:
        Loss += np.abs(w-el)
    return Loss

def dP(w, x):
    res = 0
    for el in x:
        res += np.sign(w-el)
    return res
```

Write new analytical solytion w^*

==YOUR ANSWER==

Рассуждаем аналогично прошлому разу: в точке минимума производная по w равна 0:

$$\nabla_w P = \sum_i^N \text{sign}(w - x_i) = 0$$

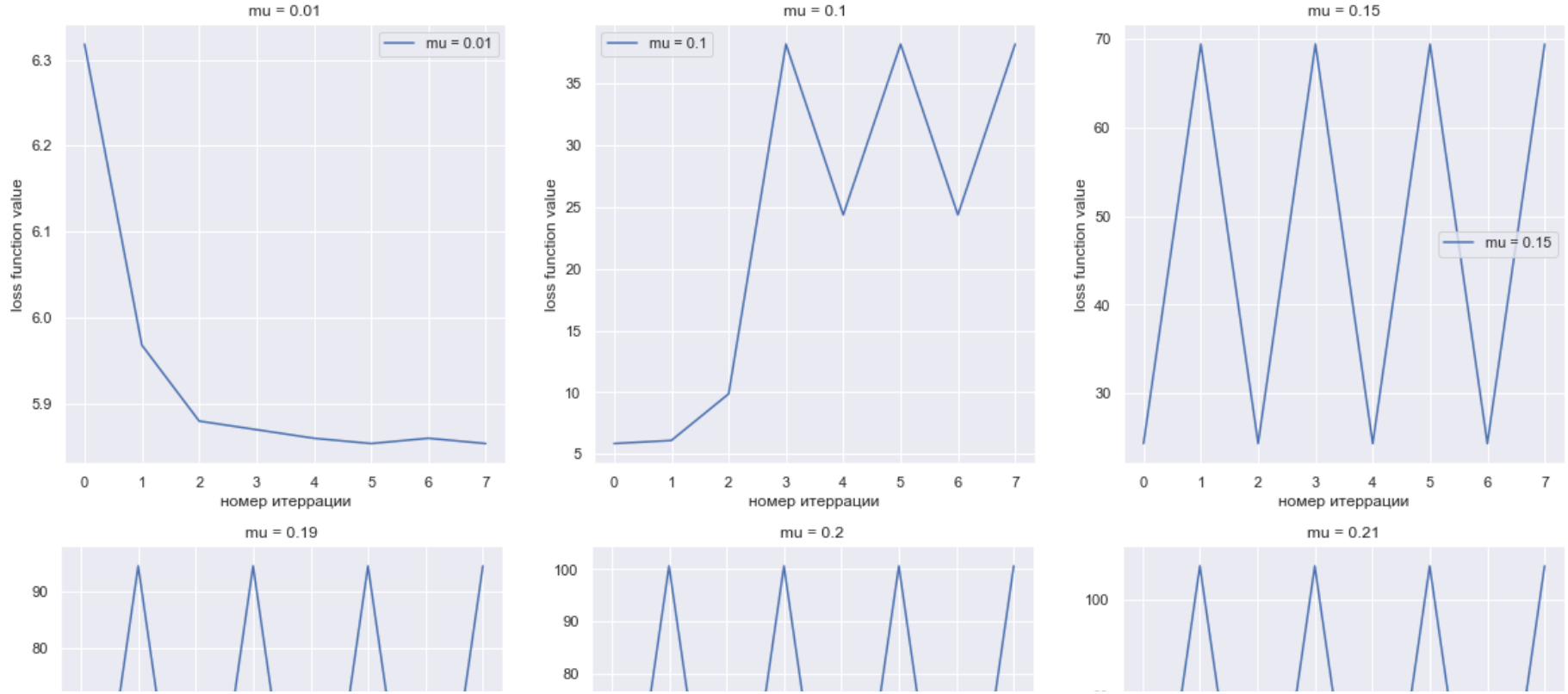
Это равенство выполнено, когда количество положительных равно количеству отрицательных слагаемых, то есть $w^* = \text{mediana}(x)$ (в нашем случае N = 25 нечётное число, если бы было чётное, то взяли бы серединку между центральными)

```
def analttical_solution(x):
    return np.median(x)
```

Plot loss trajectory for new $p(d)$.

```
[0.01, 0.1, 0.15, 0.19, 0.29, 0.31]
```

```
mu = [0.01,0.1,0.15,0.19,0.20,0.21]
N = 8
w_init = np.random.rand(1)
f, axes = plt.subplots(1, 3, figsize=(20, 6))
for i in range(3):
    axes[i].plot(range(N), gradient_descent(x, dP, w_init , mu[i], N)[0], label=f'mu = {mu[i]}')
    axes[i].legend(loc = 'best')
    axes[i].set_xlabel('номер итеррации')
    axes[i].set_ylabel('loss function value')
    axes[i].set_title(f'mu = {mu[i]}')

f, axes = plt.subplots(1, 3, figsize=(20, 6))
for i in range(3):
    axes[i].plot(range(N), gradient_descent(x, dP, w_init, mu[i+3], N)[0], label=f'mu = {mu[i+3]}')
    axes[i].legend(loc = 'best')
    axes[i].set_xlabel('номер итеррации')
    axes[i].set_ylabel('loss function value')
    axes[i].set_title(f'mu = {mu[i+3]}')
```



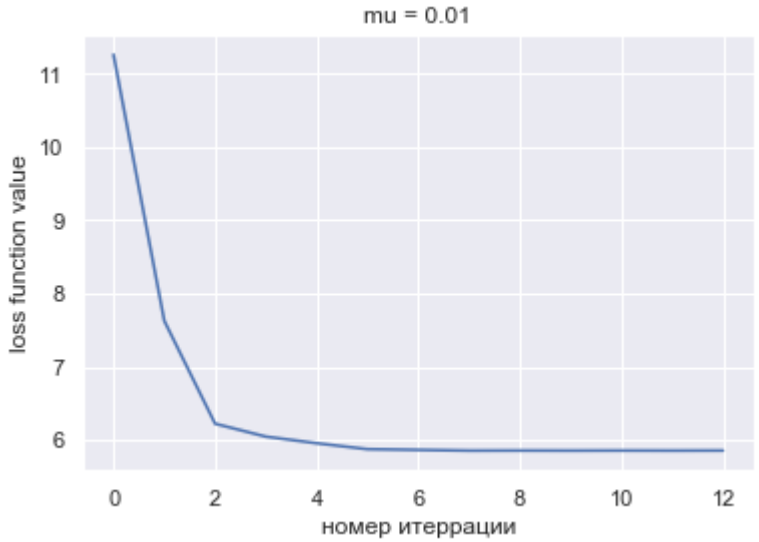
Снова видим, что градиентный спуск сходится только при $\mu = 0.01$, так как все остальные значения градиентного шага большие для нашей задачи (проскакиваем точку минимума и как бы <<поднимаеся>> вверх по функции модуля) При этом стоит заметить, что затем шаги повторяются. То есть скорее всего мы ходим горизонтально по функции модуля. Так как тут у нас константный градиент и он не меняется при отдалении от точки минимума (или при приближении к ней), то такая ситуация возможна

Проверим, что сходимся к аналитическому решению при $\mu = 0.01$

```
N = 13
hist, w_best = gradient_descent(x, dP, 0.1 , 0.01, N)
plt.plot(range(N), hist)
plt.xlabel('номер итеррации')
plt.ylabel('loss function value')
plt.title(f'mu = {0.01}')
```

```
print(f"w* = {w_best}")
print(f"w_analytical = {analttical_solution(x)}")
```

```
w* = 0.5700000000000001
w_analytical = 0.5744859334720218
```



Хмммм, аналитическое и решение полученное при помощи градиентного спуска отличаются уже во втором знаке. При этом если брать чётное кол-во шагов градиентного спуска, то получаем значение 0.5800..01, если нечётное, то 0.5700..01. Хотя аналитическое не то и не другое!

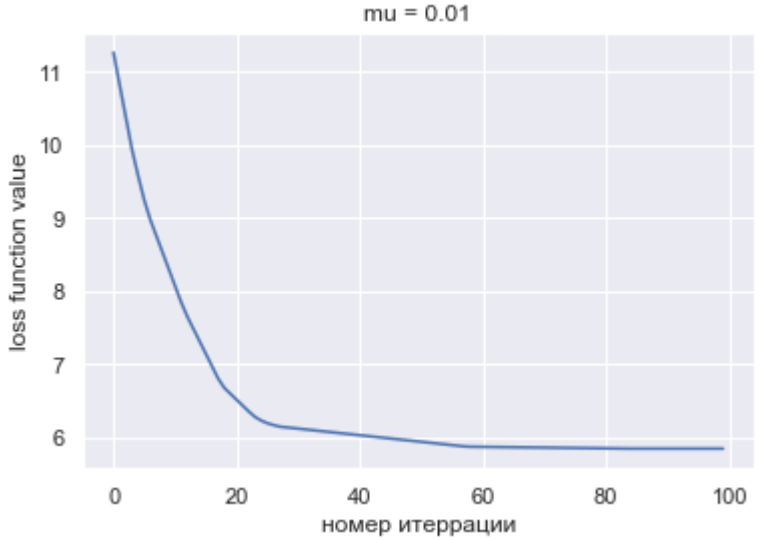
Это объясняется тем, что в отличие от предыдущего случая знаяения градиента не уменьшается при подходе к точке минимума, оно всегда 0.01 или -0.01, поэтому мы не сойдёмся с точность выше чем до 2го знака после запятой к истинному решению

Эту проблему можно побороть изменяя шаг градиентного спуска при приближении к минимуму. Можно просто взять шаг в 0.001 и проверит, что теперь точность увеличится ещё до одного знака после запятой. Проверим это:

```
N = 100
hist, w_best = gradient_descent(x, dP, 0.1 , 0.001, N)
plt.plot(range(N), hist)
plt.xlabel('номер итеррации')
plt.ylabel('loss function value')
```

```
plt.title(f'mu = {0.01}')
print(f"w* = {w_best}")
print(f"w_analytical = {analttical_solution(x)}")
```

```
w* = 0.5740000000000003
w_analytical = 0.5744859334720218
```



Гипотеза подвердилась! (соответственно, так как уменьшили шаг в 10 раз, то число итераций для схождения возросло в 10 раз)

After several years, the goverment propose to destroy the first station but choose locations for two new stations. In this conditions cost of connecting all house calculated by new formula:

$$P(w_1, w_2, x) = \sum_{i=1}^{N} p(d_i) = \sum_{i=1}^{N} p\left(\min\left(|w_1 - x_i|, |w_2 - x_i|\right)\right)$$

Write new `P`, `dP`.

Так как дальше подобная задача будет для $P(d) = d^2$, то сейчас решу для $p(d) = |d|$: то есть просто суммируем минимум из двух модулей

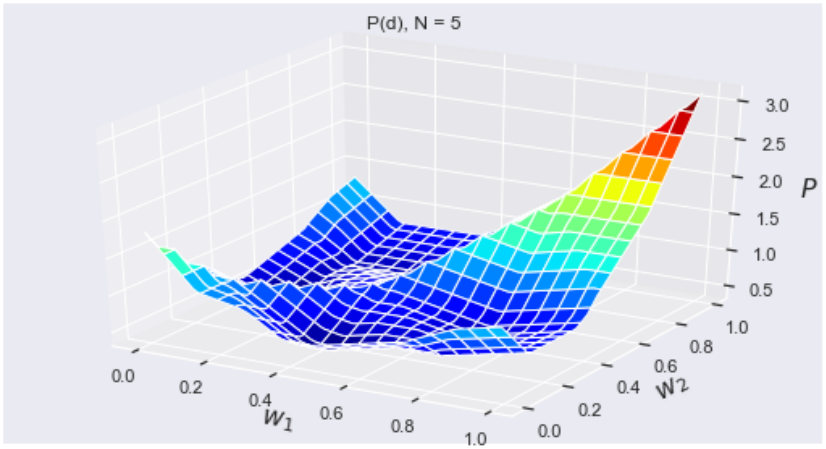
```
def P(w1, w2, x):
    Summ = 0
    for el in x:
        Summ += min(abs(w1-el), abs(w2-el))
    return Summ

def dP(w1, w2, x):
    dw1 = 0
    dw2 = 0
    for el in x:
        if min(abs(w1-el), abs(w2-el)) == abs(w1-el):
            dw1 += np.sign(w1 -el)
        else:
            dw2 += np.sign(w2 -el)
    return np.array([dw1, dw2])
```

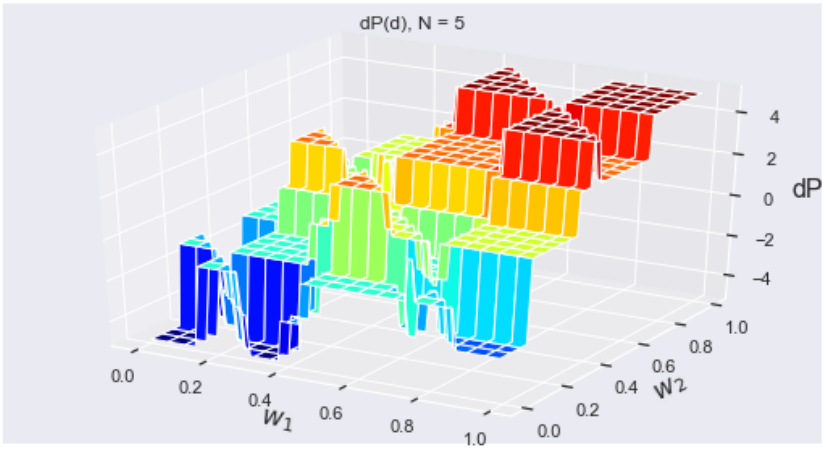
Plot $P(w_1, w_2), \nabla P(w_1, w_2)$ for different number of houses N. Comment on what happens as you increase N.

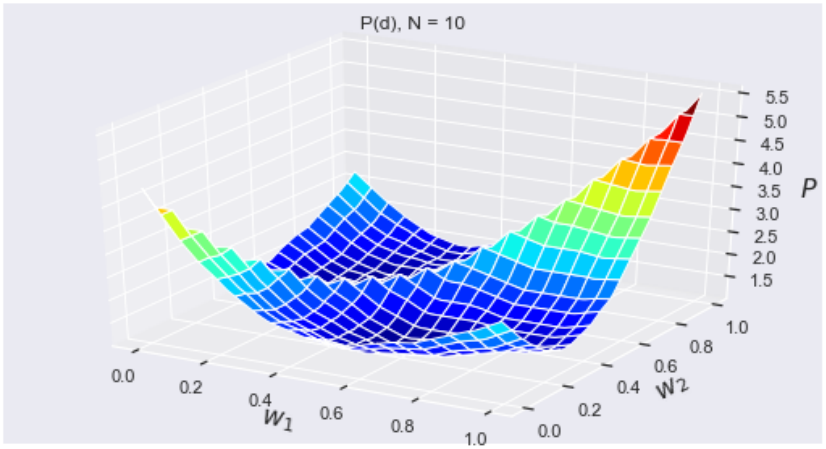
```
def P_buf(w1, w2, x):
    R = np.zeros((len(w1), len(w1)))
    for i in range(len(w1)):
        for j in range(len(w1)):
            R[i][j] = P(w1[i], w2[j], x)
    return R


def dP_buf(w1, w2, x):
    R = np.zeros((len(w1), len(w1)))
    for i in range(len(w1)):
        for j in range(len(w1)):
            R[i][j] = np.sum(dP(w1[i], w2[j], x))
    return R
```

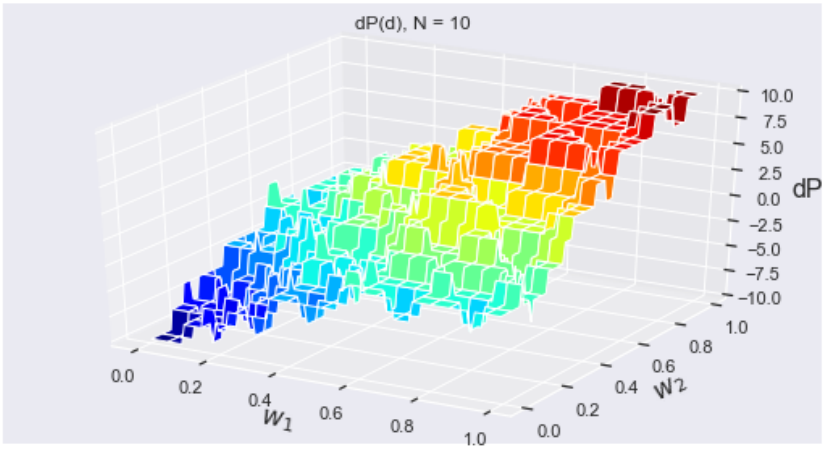
Не совсем понял как построить график для градиента, так как тут у нас 2 значения: по w1 и по w2, поэтому беру и строю сумму градиентов

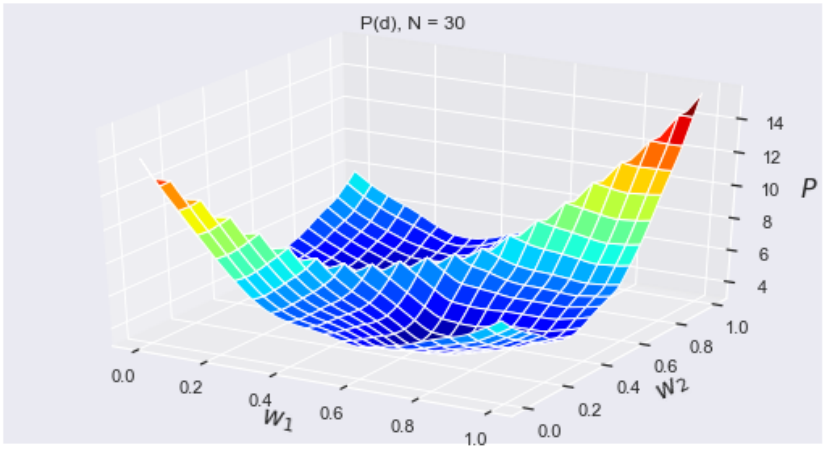
```
import matplotlib.pyplot as plt
import numpy as np
import matplotlib as mpl

from mpl_toolkits.mplot3d import axes3d


for n in [5, 10, 30, 80]:
    x = np.random.rand(n)*1
    fig, ax = plt.subplots(1, 2, figsize=(20, 5), subplot_kw=dict(projection='3d'))

    i = np.arange(0, 1, 0.01)
    X, Y = np.meshgrid(i, i)
    Z = P_buf(i, i, x)
    ax[0].plot_surface(X, Y, Z, rstride=5, cstride=5, cmap='jet')

    ax[0].set_xlabel("$w_1$", fontsize=16)
    ax[0].set_ylabel("$w_2$", fontsize=16)
    ax[0].set_zlabel("$P$", fontsize=16)
    ax[0].set_title(f"P(d), N = {n}")


    ax[1].plot_surface(X, Y, dP_buf(i, i, x), rstride=5, cstride=5, cmap='jet')

    ax[1].set_xlabel("$w_1$", fontsize=16)
    ax[1].set_ylabel("$w_2$", fontsize=16)
    ax[1].set_zlabel("dP", fontsize=16)
    ax[1].set_title(f"dP(d), N = {n}")
```

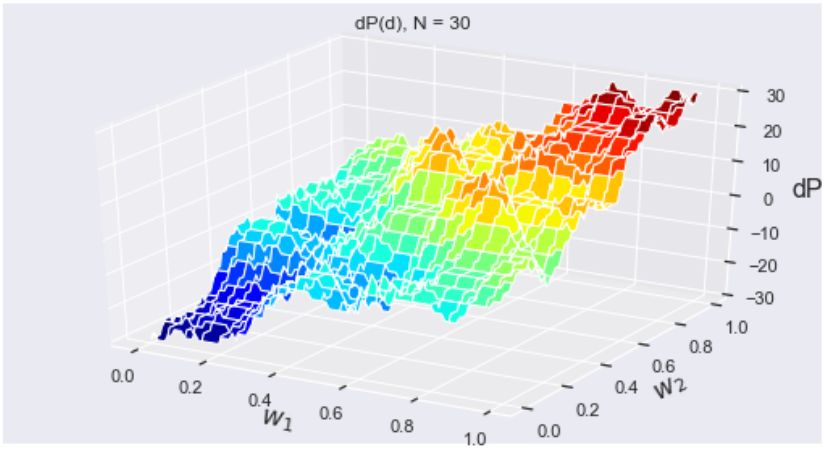

P(d), N = 5
P
w_1
w_2

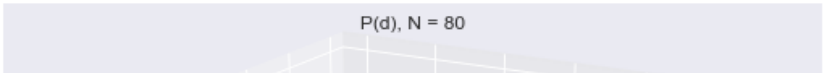

dP(d), N = 5
dP
w_1
w_2

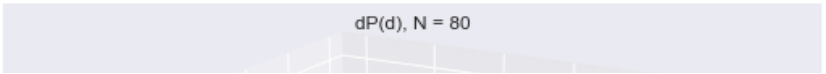

P(d), N = 10
P
w_1
w_2

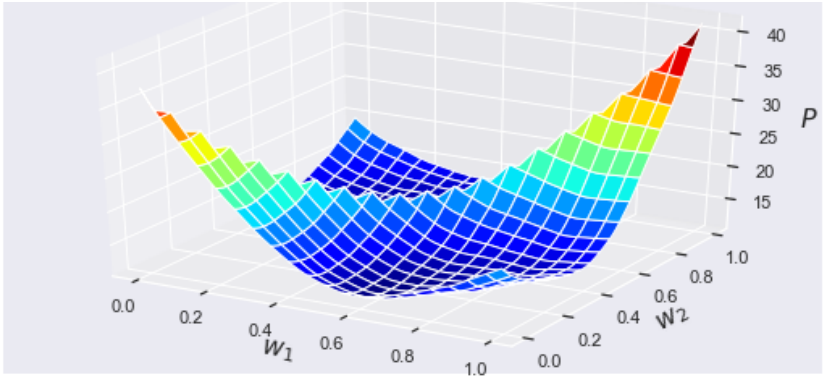

dP(d), N = 10
dP
w_1
w_2

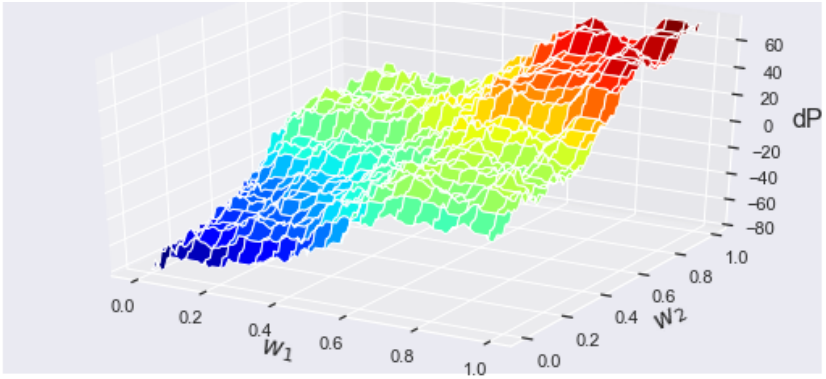

P(d), N = 30
P
w_1
w_2


dP(d), N = 30
dP
w_1
w_2


P(d), N = 80


dP(d), N = 80





При увелечении n график лосс функции принимает одинаковую форму: имеет 2 симметричных минимума относительно $w_1 = w_2$. При больших n они становятся более <<глубокими>>, то есть значение антиградиента, который указывает туда, становится больше

Write new `gradient_descent`, which returns the entire optimization trajectory (w_k) after a fixed number of steps and draws the process on the graphs P and ∇P that were above. To ease visualization try to use `ax.quiver`

```
def beutiful_graph_loss(X_h, Y_h, hist, title=""):
    fig = plt.figure(figsize=(20, 5))
    ax = fig.add_subplot(1, 2, 1, projection='3d')
    i = np.arange(0, 1, 0.01)
    X, Y = np.meshgrid(i, i)
    Z = P_buf(i, i, x)
    ax.plot_surface(X, Y, Z, rstride=5, cstride=5, cmap='jet')
    ax.plot(X_h, Y_h, hist, color="red", marker="*")

    ax.set_xlabel("$w_1$", fontsize=16)
    ax.set_ylabel("$w_2$", fontsize=16)
```

```
        ax.set_zlabel("$P$", fontsize=16)
        ax.set_title(title)


    def gradient_descent(x, dP, w0, mu, Nsteps):
        hist = []
        X = []
        Y = []
        for i in range(Nsteps):
            hist.append(P(w0[0], w0[1], x))
            X.append(w0[0])
            Y.append(w0[1])
            w0 -= mu * dP(w0[0], w0[1], x)
        return hist, w0, X, Y


    N = 5
    hist, w_best, X_h, Y_h = gradient_descent(x, dP, w0=[0.5, 0.2], mu=0.005, Nsteps=N)
    w_best
```

```
     array([0.785, 0.275])
```

```
    beutiful_graph_loss(X_h, Y_h, hist, title="gradient_descent_2D problem with p(d) = |d|")
```

Спускается до глобального минимума! За 5 итерраций!

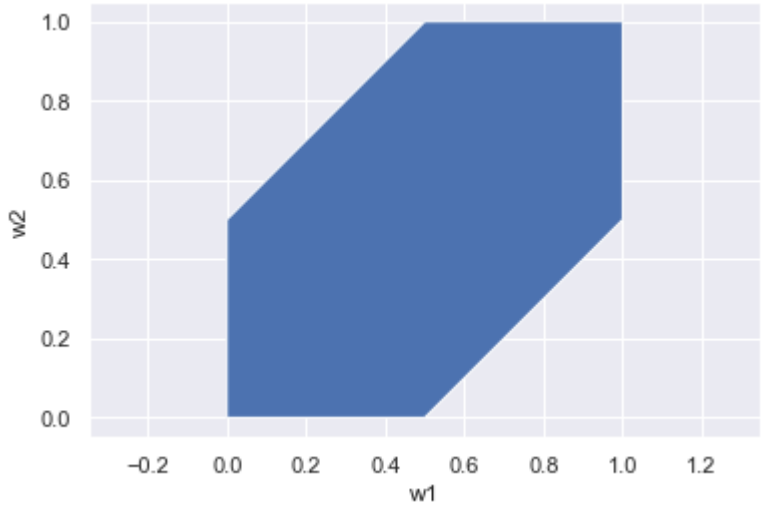
Construction is almost underway, but new safety regulations do not allow stations to be on the distance more than 1/2:

$$|w_1 - w_2| \leq \frac{l}{2}$$

Plot our feasible set. Is it convex?

```
from matplotlib.patches import Polygon
```

```
ax = plt.gca()
M = [[0, 0], [0.5, 0], [1, 0.5], [1, 1], [0.5, 1], [0, 0.5]]
ax.add_patch(Polygon(M))
ax.set_xlabel("w1")
ax.set_ylabel("w2")
plt.axis('equal')
plt.show;
```



По построению видим, что множество допустимых значений - выпуклое (тут считаю l = 1)

Write `conditional_SGD`, which returns the entire optimization trajectory (w_k) after a fixed number of steps and draws the process on the graphs P and ∇P that were above.

The conditional gradient descent method consists in making a gradient step and then checking if the obtained point belongs to the feasible set. If it belongs to the target set, the algorithm continues, otherwise a projection to the feasible set is made.

```
def conditional_SGD(x, dP_sigma, w0, mu, Nsteps, p=0.4):
    hist = []
    X = []
    Y = []
    for i in range(Nsteps):
        # print(dP(w0[0], w0[1], x))
        g = w0 - mu * dP(w0[0], w0[1], x)
        w0 = projection(g)
        hist.append(P(w0[0], w0[1], x))
        X.append(w0[0])
        Y.append(w0[1])
    return hist, w0, X, Y
```

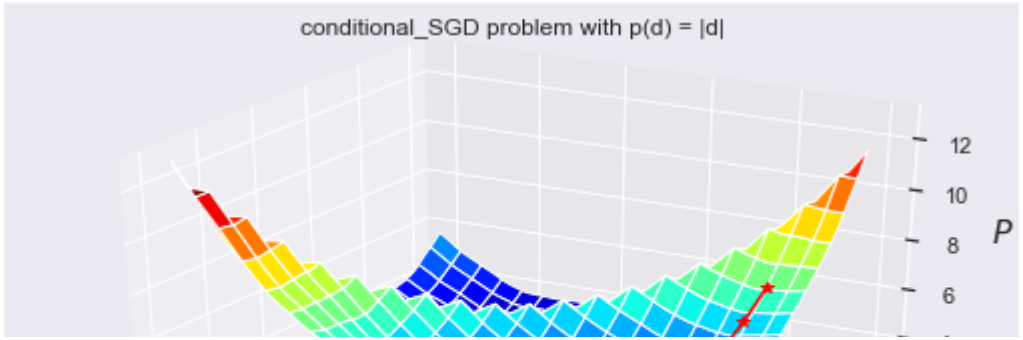
Смотрю на картинку и геометрически нахожу проекцию. В 6 случаях это легко, просто проецируем на горизонтальные и вертикальные прямые. В 2х случая нахожу проекцию на прямые $y = x \pm 0.5$ (вспомнил аналит первого семетра). Если точка попадает в бюджетное множество, то ничего не делаю

```
def projection(w):
    x = w[0]
    y = w[1]
    res = [0, 0]
    if x <= 0 and 0 <= y <= 0.5:
        res = [0, y]
    elif x < 0 and y < 0:
        res = [0, 0]
    elif 0 <= x <= 0.5 and y <= 0:
```

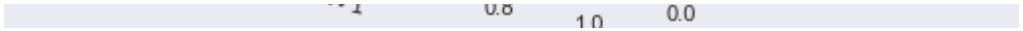
```
        elif 0 <= x <= 0.5 and y <= 0:
            res = [x, 0]
        elif 0.5 <= x <= 1 and y >= 1:
            res = [x, 1]
        elif x >= 1 and 0 <= y <= 0.5:
            res = [1, y]
        elif x > 1 and y > 1:
            res = [1, 1]
        elif 0 <= x <= 0.5 and y > x + 0.5:
            x_p = (x + y - 0.5)/2
            y_p = x_p + 0.5
            res = [x_p, y_p]
        elif 0.5 <= x <= 1 and y < x -0.5:
            x_p = (x + y + 0.5)/2
            y_p = x_p - 0.5
            res = [x_p, y_p]
        else:
            res = [x, y]
        # print(res)
        return np.array(res)


    x = np.random.rand(25)*1


    N = 10
    hist, x_cond, X_h, Y_h = conditional_SGD(x, dP, w0=[0.9, 0.85], mu=0.005, Nsteps=N)
    x_cond
```

```
array([0.87, 0.37])
```

```
beutiful_graph_loss(X_h, Y_h, hist, title="conditional_SGD problem with p(d) = |d|")
```



Заметим (по графику лосс функции), что глобальный минимум не лежит в нашем бюджетном множестве мы его находили в прошлом пункте. Он совсем немного отличается. Поэтому написанная проекция работает правильная и conditional_SGD не заходит не в бюджетное множество и сходится к минимуму



We have same loss function

$$P(w, x) = \sum_{i=1}^{N} p(d_i) = \sum_{i=1}^{N} p(|w - x_i|),$$

where $p(d) = d^2$

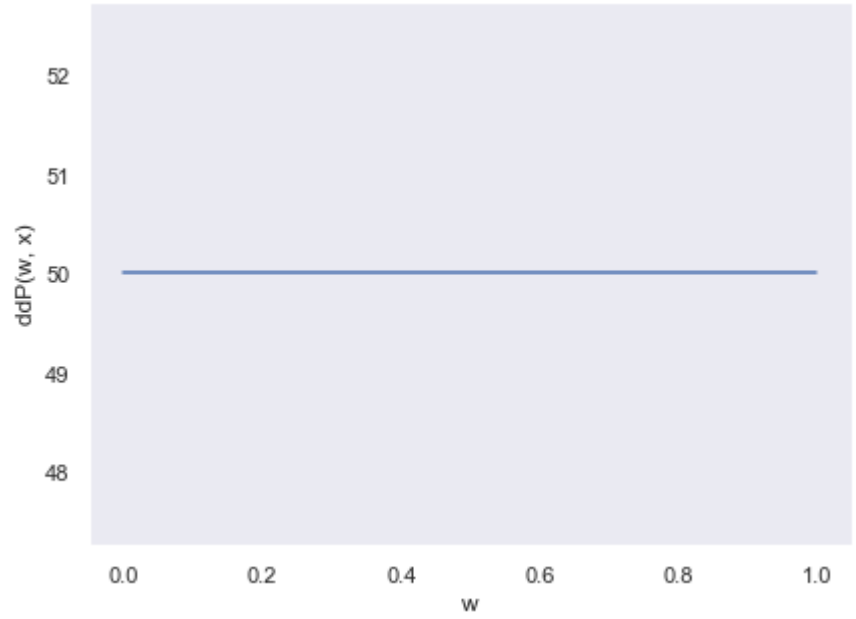
Write functions `P, dP, ddP`. `ddP` has to return hessian of loss function

```
def P(w, x):
    return np.sum((w - x)**2)

def dP(w, x):
    return 2 * np.sum(w-x)

def ddP(w, x):
    return 2 * len(x)
```

Plot `ddP` on the range (0,l)

Так как функция потерь квадратичная, то 2ая производная - это константа))

```
data = []
for el in np.linspace(0, 1, 100):
    data.append(ddP(el, x))
fig = plt.figure(figsize = (15,5))
ax = plt.subplot(121)
ax.plot(np.linspace(0, 1, 100), data)
ax.set_ylabel('ddP(w, x)')
ax.set_xlabel('w')
ax.grid()
```



Write function `newton_descent`, which return all optimization trajectory. Update rule:

$$w_{i+1} = w_i - \nabla^2 f(w_i)^{-1} \nabla f(w_i)$$

```
def newton_descent(x, dP, ddP, w0, Nsteps):
    trajectory = []
```

```
    for i in range(Nsteps):
        trajectory.append(P(w0, x))
        w0 -= ddP(w0, x)**(-1) * dP(w0, x)
    return trajectory, w0
```

Investigate the behavior of Newton's method depending on the different initial point w_0.

Заново сгенерируем x - координаты домов для N = 25

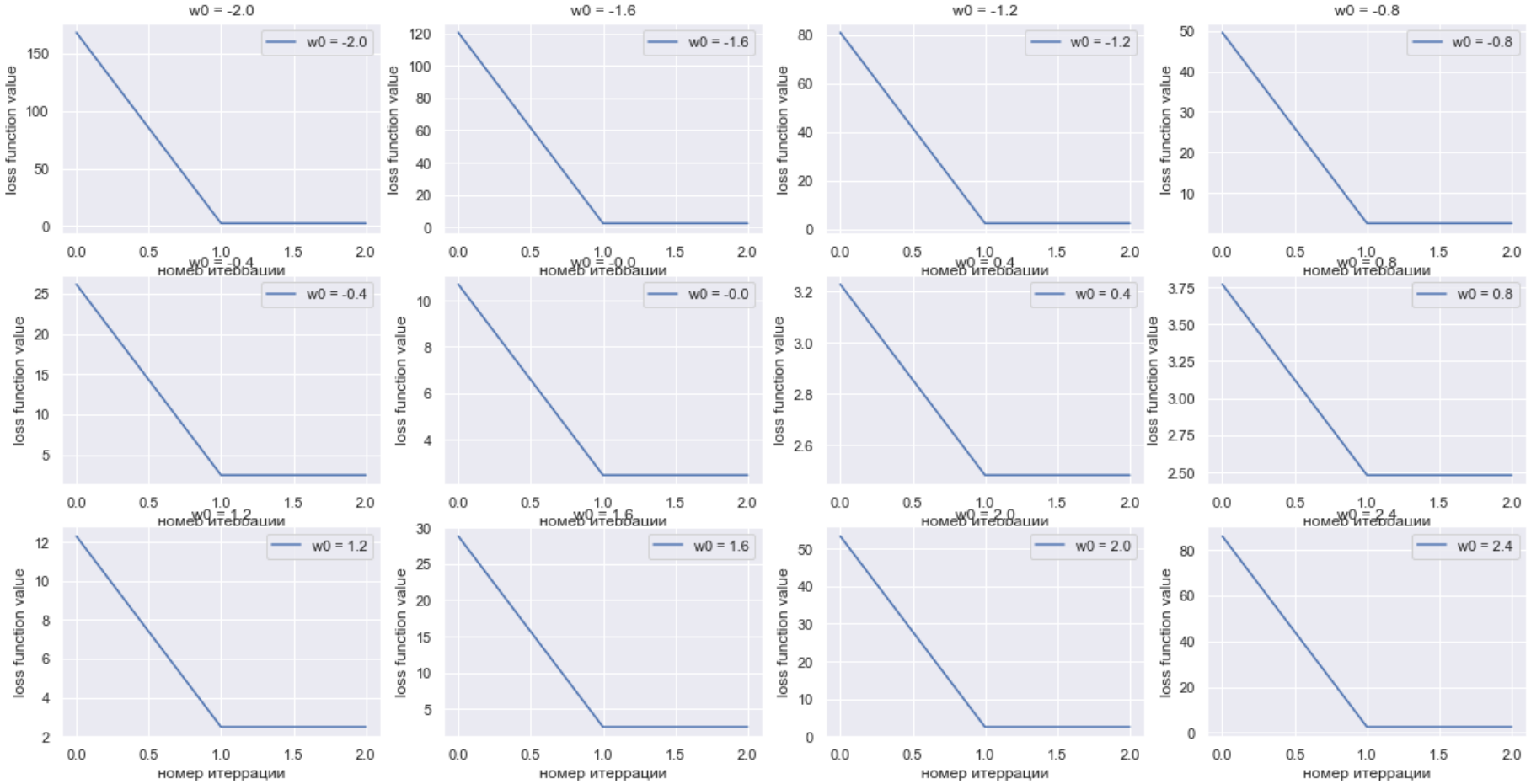
```
x = np.random.rand(25)*1
```

Аналитическое решение:

```
np.sum(x)/25
```

```
0.5729844286226786
```

```
f, axes = plt.subplots(3, 4, figsize=(20, 10))
axes = axes.reshape(-1)
# print(axes.shape)
i = 0
N = 3
for w0 in np.arange(-2, 2.8, 0.4):
    axes[i].plot(range(N), newton_descent(x, dP, ddP, w0, N)[0], label=f'w0 = {round(w0, 1)}')
    axes[i].legend(loc = 'best')
    axes[i].set_xlabel('номер итеррации')
    axes[i].set_ylabel('loss function value')
    axes[i].set_title(f'w0 = {round(w0, 1)}')
    i += 1
```



Вспомним график функции, которую мы оптимизируем - это парабола! Значит метод Ньютона должен сходиться за 1 итерацию (так как аппроксимация параболы - параболой это та же самая парабола и мы шагаем первым шагом в её минимум). Также видим, что можем брать начальную точку далеко до оптимальной и все равно сойдёмся за 1 шаг (в отличие от градиентного спуска)

Например, возьмём и запустим при w0 = 100 и за 1 шаг получим точно решение задачи (совпадает с аналитическим до 12го знака после запятой). Круто!!!!

Оптимальное значение:

```
newton_descent(x, dP, ddP, 100, 1)[1]
```

```
0.5729844286226609
```

Write function `multi_newton`, which solve 2D task:

$$P(w_1, w_2, x) = \sum_{i=1}^{N} p(d_i) = \sum_{i=1}^{N} p\left(\min\left(|w_1 - x_i|, |w_2 - x_i|\right)\right)$$

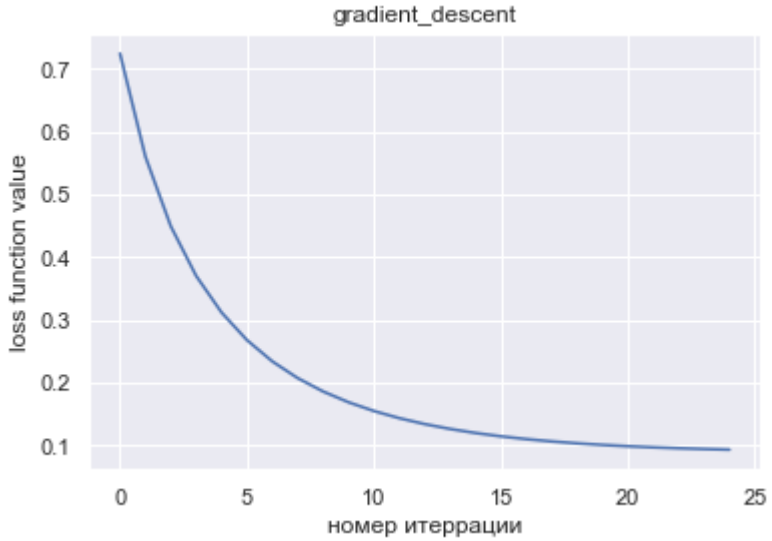
with $p(d) = d^3$ using Newton method and return optimization trajectory. Compare results with gradient descent.

```
def P(w1, w2, x):
    Summ = 0
    for el in x:
        Summ += (min(abs(w1-el), abs(w2-el)))**3
    return Summ

def dP(w1, w2, x):
    dw1 = 0
    dw2 = 0
    for el in x:
        if min(abs(w1-el), abs(w2-el)) == abs(w1-el):
            dw1 += 3 * np.sign(w1 -el) * (abs(w1-el))**2
        else:
            dw2 += np.sign(w2 -el) * 3 * (abs(w2-el))**2
    return np.array([dw1, dw2])

def ddP(w1, w2, x):
    R = np.zeros((2, 2))
    eps = 0.001
    for el in x:
        if min(abs(w1-el), abs(w2-el)) == abs(w1-el):
            R[0][0] += 6 * (abs(w1-el))
        else:
```

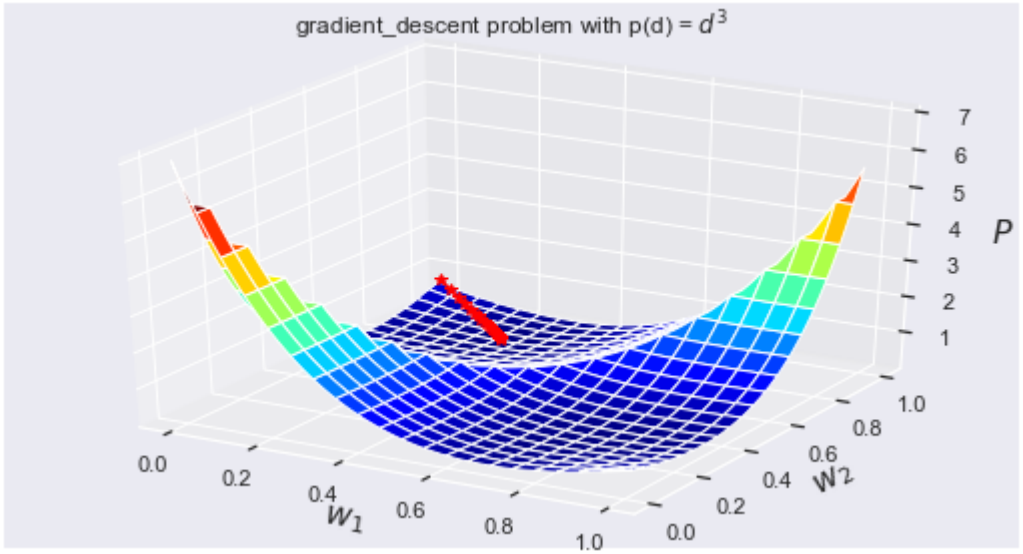
```
            else:
                R[1][1] += 6 * (abs(w2-el))

        return R


    def multi_newton(x, dP, ddP, w0, Nsteps):
        trajectory = []
        X = []
        Y = []
        for i in range(Nsteps):
            trajectory.append(P(w0[0], w0[1], x))
            X.append(w0[0])
            Y.append(w0[1])
            w0 -= np.linalg.inv(ddP(w0[0], w0[1], x)) @ dP(w0[0], w0[1], x)
        return trajectory, w0, X, Y


    def gradient_descent(x, dP, w0, mu, Nsteps):
        trajectory = []
        X = []
        Y = []
        for i in range(Nsteps):
            trajectory.append(P(w0[0], w0[1], x))
            X.append(w0[0])
            Y.append(w0[1])
            w0_0, w0_1 = dP(w0[0], w0[1], x)
            w0_0 *= mu
            w0_1 *= mu
            w0 -= np.array([w0_0, w0_1])

        return trajectory, w0, X, Y


    ### Gradient descent trajectory
    n = 25
    hist_grad, w_grad, X_h, Y_h = gradient_descent(x, dP, np.array([0.0, 1.0]), 0.01, n)
    plt.plot(range(n), hist_grad)
    plt.xlabel('номер итеррации')
    plt.ylabel('loss function value')
    plt.title(f'gradient descent')
```

```
plt.title(f"gradient_descent")
print(f"w* = {w_grad}")
```

```
w* = [0.25843005 0.81100178]
```



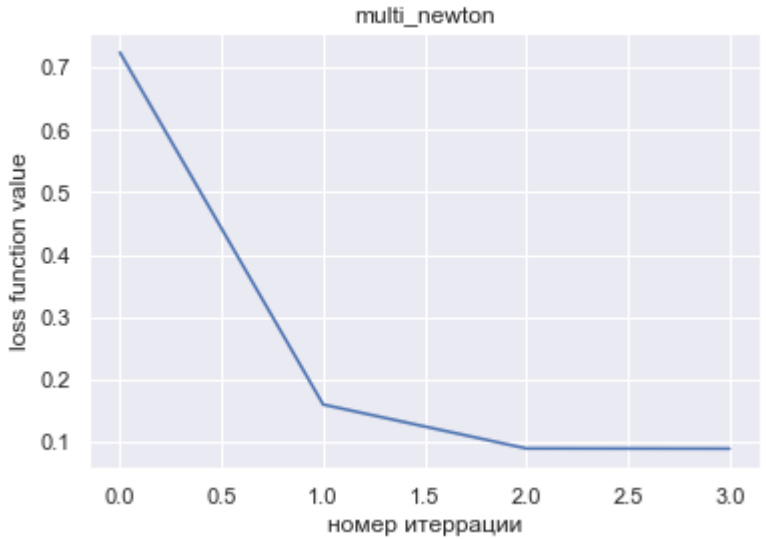

```
beutiful_graph_loss(X_h, Y_h, hist_grad, title="gradient_descent problem with p(d) = $d^3$")
```



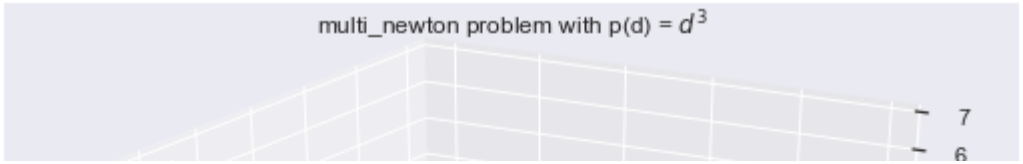

```
### Newton descent trajectory
```

```
### Newton descent trajectory
n = 4
hist_Newton, w_Newton, X_h, Y_h = multi_newton(x, dP, ddP, np.array([0.0, 1.0]), n)
plt.plot(range(n), hist_Newton)
plt.xlabel('номер итеррации')
plt.ylabel('loss function value')
plt.title(f'multi_newton')
print(f"w* = {w_Newton}")
```

```
w* = [0.27658396 0.78571771]
```




```
beutiful_graph_loss(X_h, Y_h, hist_Newton, title="multi_newton problem with p(d) = $d^3$");
```



Видим, что за 3 итеррации метод Ньютона сходится к глобальному минимуму! Градиентый спуск это делает намного дольше: при этом приходится подбирать гиперпараметр - шаг градиентного спуска, в методе Ньютона с этим проще)

Projected gradient descent

Find projection on the S set $\pi_S(y) = \pi$ if:

$$S = \{x \in \mathbb{R}^n \mid Ax = b, A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m\}, y \notin S$$

Hint: Check fmin.xyz

==YOUR ANSWER==

Предположим, что: $\pi = y + \sum_{i=1}^{m} \alpha_i A_i = y + A^T \alpha$. Коэффициент α выбираем так, чтобы $\pi \in S : A\pi = b$, поэтому:

$$A\left(y + A^T \alpha\right) = b$$

$$Ay = b - AA^T \alpha$$

Проверим наше предположение, записав <<критерий>> проекции на выпуклое замкнутое множество (проверяли в 1ом семестре первого задания, что наше множество является таким):

π проекция точки $\mathbf{y}$ на S, при условии $\pi_S(\mathbf{y}) = \pi$ тогда и только тогда, когда для любого $\mathbf{x} \in S$ верно неравенство:

$$\langle \pi - \mathbf{y}, \mathbf{x} - \pi \rangle = 0$$

Проверяем:

$$(\pi - y)^T (x - \pi) \geq 0$$
$$\left(y + A^T\alpha - y\right)^T \left(x - y - A^T\alpha\right) =$$
$$\alpha^T A \left(x - y - A^T\alpha\right) =$$
$$\alpha^T (Ax) - \alpha^T (Ay) - \alpha^T \left(AA^T\alpha\right) =$$
$$\alpha^T b - \alpha^T \left(b - AA^T\alpha\right) - \alpha^T AA^T\alpha =$$
$$\alpha^T b - \alpha^T b + \alpha^T AA^T\alpha - \alpha^T AA^T\alpha = 0 \geq 0$$

Выполнено, значит, $\pi = y + \sum_{i=1}^m \alpha_i A_i = y + A^T\alpha$ с условием на α (см начало) - искомая проекция

For following problem:

$$\begin{cases} \|Ax - b\|_2^2 \longrightarrow \min\limits_{x \in \mathbb{R}^n}, & A \in \mathbb{R}^{m \times n},\ m \geq n,\ \operatorname{rg} A = n \\ Cx = d. & C \in \mathbb{R}^{k \times n},\ k \leq n,\ \operatorname{rg} C = k \end{cases} \quad b \in \mathbb{R}^m,\ d \in \mathbb{R}^k$$

- Write KKT conditions;
- Find x^* - solution;

==YOUR ANSWER==

Лагранжиан для данной задачи:

$$L(x, \lambda) = \|Ax - b\|_2^2 + \lambda^T (Cx - d)$$

Условия KKT:

$$\begin{cases} \nabla_x \, \mathrm{L}(x, \lambda) : 2A^T (Ax - b) + C^T \lambda = 0 \\ \nabla_\lambda \, \mathrm{L}(x, \lambda) : Cx = d \end{cases}$$

Из 1го условия выражаем x:

$$2A^T Ax = 2A^T b - C^T \lambda \to x = \frac{1}{2} \left(A^T A\right)^{-1} \left(2A^T b - C^T \lambda\right)$$

Из 2го условия выражаем $x = C^{-1} d$ и подставляем в полученное выражение, находим λ:

$$C^{-1}d = \left(A^TA\right)^{-1}A^Tb - \frac{1}{2}\left(A^TA\right)^{-1}C^T\lambda$$

$$\left(A^TA\right)^{-1}C^T\lambda = 2\left(\left(A^TA\right)^{-1}A^Tb - C^{-1}d\right)$$

Следовательно:

$$\lambda = 2\left(C\left(A^TA\right)^{-1}C^T\right)^{-1}\left(C\left(A^TA\right)^{-1}A^Tb - d\right)$$

Тогда, подставляя полученное λ в выражение для x, получим:

$$x = \frac{1}{2}\left(A^TA\right)^{-1}\left(2A^Tb - 2C^T\left(C\left(A^TA\right)^{-1}C^T\right)^{-1}\left(C\left(A^TA\right)^{-1}A^Tb - d\right)\right)$$

Данная задача выпуклая, так как функция, задающая ограничения типа равенств - линейная, а функцию, которую оптимизируем - выпуклая.

Если система $Cx = d$ несовместна, то бюджетное множество пустое и оптимальное значение принимаем за $p^* = \infty$, но у нас в условии система недоопределена: число уравнений меньше, чем число неизвестных, поэтому решение всегда найдётся

Иначе решение системы можно записать через псевдообратную матрицу $x = C^\dagger d$ и будет выполнено условие Слейтера (будет существовать допустимая точка, в которой все ограничения типа равенств выполнены, то есть она принадлежит относительной внутренности бюджетного множества), значит, в этом случае условия ККТ - достаточные и полученное ранее

$$x^* = \left(A^TA\right)^\dagger\left(A^Tb - C^T\left(C\left(A^TA\right)^\dagger C^T\right)^\dagger\left(C\left(A^TA\right)^\dagger A^Tb - d\right)\right)$$

-решение задачи (вместо обратных нужно брать псевдообратные матрицы, так как из условий задачи обратные могут не существовать. Все выкладки с псевдообратными совпадают с просто обратной)

Implement projected gradient descent for following task.

$$\begin{cases} \|Ax-b\|_2^2 \longrightarrow \min\limits_{x \in \mathbb{R}^n}, & A \in \mathbb{R}^{m \times n},\ m \geq n,\ \operatorname{rg} A = n \\ & C \in \mathbb{R}^{k \times n},\ k < n,\ \operatorname{rg} C = k \end{cases} \quad b \in \mathbb{R}^m,\ d \in \mathbb{R}^k$$

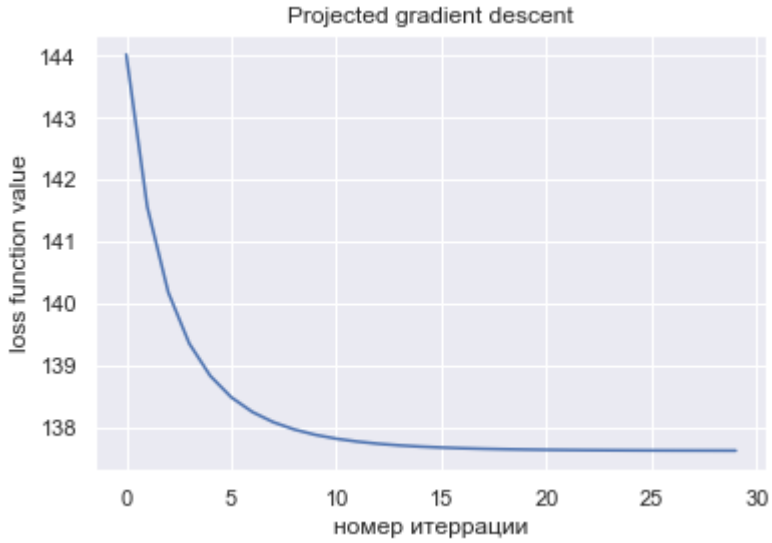
Для нашей функции

$$\frac{\partial}{\partial x}\left(\|A \cdot x - b\|_2^2\right) = 2 \cdot A^\top \cdot (A \cdot x - b)$$

```
import numpy as np


np.random.seed(1)

A = np.random.randn(100, 20)
b = np.random.randn(100)

C = np.random.randn(10, 20)
d = np.random.randn(10)


def f(x):
    e = A @ x - b
    return np.sum(e**2)


def df(x):
    # print(A.T.shape)
    # print((A @ x - y).shape)
    return 2 * A.T @ (A @ x - b)


#def proj(C, d, x):
def proj(y):
    alpha = np.linalg.inv(C @ C.T) @ (d - C @ y)
    return y + C.T @ alpha


def Projected_gradient_descent(f, df, proj, x0, mu, Nsteps):
    hist = []
    for i in range(Nsteps):
```

```
        for i in range(Nsteps):
            g = x0 - mu * df(x0)
            x0 = proj(g)
            hist.append(f(x0))
            # print(C @ x0 - d)
        return hist, x0


    x0 = np.zeros(20)


    ### Projected gradient descent
    N = 30
    hist, x_des = Projected_gradient_descent(f, df, proj, x0, mu=0.001, Nsteps=N)
    plt.plot(range(N), hist)
    plt.xlabel('номер итеррации')
    plt.ylabel('loss function value')
    plt.title(f'Projected gradient descent')
    print(f"w* = {x_des}")
```

```
w* = [ 0.19012338  0.2034396  -0.00115426 -0.2235327  -0.1005886  -0.05238907
 -0.21912633  0.12524721  0.01745041  0.06009852 -0.35010648 -0.21418127
 -0.01199195 -0.22072006  0.02156931 -0.09905324 -0.069765    0.23416598
 -0.00849495 -0.08779023]
```



Comparison with `scipy.optimize`

```
from scipy.optimize import minimize


cons = ({'type': 'eq', 'fun': lambda x:  C @ x - d})


res = minimize(f, x0,
               constraints=cons)
print(res.x)
print(res.message)
```

```
     [ 0.19141192  0.20427828  0.00067643 -0.22399297 -0.09994442 -0.05341131
      -0.21970832  0.12296092  0.01729651  0.05976738 -0.35036119 -0.21625861
      -0.01128401 -0.2206766   0.02382163 -0.10195978 -0.07071159  0.23374888
      -0.00852641 -0.08666988]
     Optimization terminated successfully.
```

Comparison with analytical solution

```
AA = np.linalg.inv(A.T @ A)
x_analytic = AA @ (A.T @ b - C.T @ np.linalg.inv(C @ AA @ C.T) @ (C @ AA @ A.T @ b - d))
x_analytic
```

```
     array([ 0.19142072,  0.20427311,  0.0006669 , -0.22399507, -0.09995472,
            -0.05339808, -0.21970568,  0.12297266,  0.01729464,  0.05976843,
            -0.35036166, -0.21625138, -0.01128753, -0.22067229,  0.02382072,
            -0.10194633, -0.07070533,  0.23375024, -0.00852193, -0.08667476])
```

Видим, что во всех трёх случаях ответ сходится!!!

Во время написания проверил, что функция проекции правильно работает и после неё $C \cdot x_0 == d$. Мне понравился метод, так как в прошлом семестре реально не понимал, зачем находить проекции на такие непонятные множества, а тут это работает :)

С другой стороны тут не очень сложно ищется аналитическое решение (при том для этого мы действуем по алгоритму: записываем условия KKT - пытаемся решить), а проекция идейно сложнее (моё мнение).

Strongly convex function, when Polyak momentum will not converge.

Show empirically, that there is an example (see this paper - page 23) of the Strongly convex function.

- Find the specific funtion $f(x)$. Prove, that it is strongly convex function. Find the Lipschitz constant L of a $\nabla f(x)$ and strong convexity constant μ.
- Implement heavy-ball method and show its divergence with specific choice of hyperparameters.
- Try to vary hyperparameters slightly, write down your observations.

В статье рассматривается одномерная функция, которая задаётся её градиентом:

$$\nabla f(x) = \begin{cases} 25x, & x < 1 \\ x + 24, & 1 \leq x < 2 \\ 25x - 24, & x \geq 2 \end{cases}$$

Если проиентегрировать и занулить константу (нам нужна любая функция, у которой нужный градиент), то получим:

$$f(x) = \begin{cases} \frac{25x^2}{2}, & x < 1 \\ \frac{x^2}{2} + 24x, & 1 \leq x < 2 \\ \frac{25x^2}{2} - 24x, & x \geq 2 \end{cases}$$

Покажем, что константа Липшица: $L = 25$, а константа сильной выпуклости $\mu = 1$

По критерию 1го порядка сильной выпуклости функции: Differentiable $f(x)$ defined on the convex set $S \subseteq \mathbb{R}^n$ μ -strongly convex if and only if $\forall x, y \in S$:

$$f(y) \geq f(x) + \nabla f^T(x)(y-x) + \frac{\mu}{2}\|y-x\|^2$$

Подставим в это выражение Тейлоровскую аппроксимацию для $f(y)$:

$$f(y) = f(x) + \nabla f(y)^\top (y-x)$$

Получим:

$$f(x) + \nabla f(y)^\top (y-x) \geq f(x) + \nabla f^T(x)(y-x) + \frac{\mu}{2}\|y-x\|^2$$

$$(\nabla f(y)^\top - \nabla f(x)^\top)(y-x) \geq \frac{\mu}{2}\|y-x\|^2$$

Для Липшивости если произвести аналогичные рассуждения, нужно проверить, что $\forall x, y$:

$$(\nabla f(y)^\top - \nabla f(x)^\top)(y-x) \leq \frac{L}{2}\|y-x\|^2$$

Получается, чтобы доказать, что мы хотим, нужно показать, что:

$$\forall x, y: \quad \mu\|y-x\|^2 \leq (\nabla f(y) - \nabla f(x))^\top (y-x) \leq L\|y-x\|^2$$

Рассмотрим много случаев для разных x, y. Все они легко проверяются так как для каждого неравенства сокращается либо x либо y и устно подставляю диапозон рассматриваемых значений переменных

▾ 1

$$1 \leq x, y < 2$$

$$(y-x)^2 \leq (24 + y - 24 - x)(y-x) \leq 25(y-x)^2 - \text{ верно}$$

▾ 2

$$x < 1, y \geq 2$$

$$(y-x)^2 \leq (25y-24-25x)(y-x) \leq 25(y-x)^2\text{- верно}$$

так как

$$y-x \leq 25y-24-25x \leq 25(y-x)\text{- верно}$$

3

$$x < 1, 1 \leq y < 2$$

$$(y-x)^2 \leq (y+24-25x)(y-x) \leq 25(y-x)^2\text{- верно}$$

4

$$1 \leq x < 2, y \geq 2$$

$$(y-x)^2 \leq (25y-24-x-24)(y-x) \leq 25(y-x)^2\text{- верно}$$

так как

$$y-x \leq 25y-24-x-24 \leq 25(y-x)$$

5

$$x, y < 1 \text{ или } x, y \geq 2$$

$$(y-x)^2 \leq (25y-25x)(y-x) \leq 25(y-x)^2 - \text{ тоже верно}$$

Показали, что константа Липшица: $L = 25$, а константа сильной выпуклости $\mu = 1$

Реализуем метод тяжёлого шарика Поляка

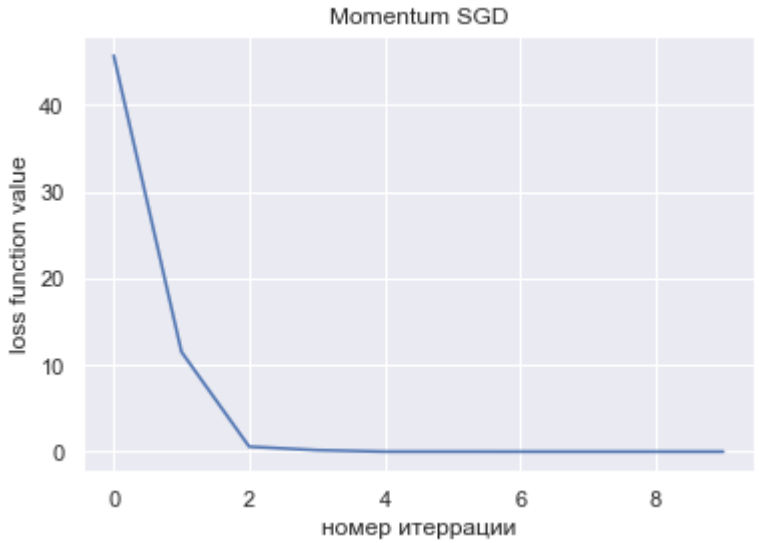
$$x_{k+1} = x_k - \alpha \nabla f(x_k) + \beta (x_k - x_{k-1})$$

```
def Momentum SGD(f, df, x0, alpha, beta, Nsteps):
```

```
        x_1 = x0
        x_2 = x0
        x_new = x0
        hist = []
        X = []
        for i in range(Nsteps):
            hist.append(f(x_new))
            X.append(x_new)
            x_new = x_2 - alpha * df(x_2) + beta*(x_2 - x_1)
            x_1 = x_2
            x_2 = x_new
        return hist, x_new, X


    def f(x):
        if x < 1:
            return 25*x**2/2
        elif 1 <= x < 2:
            return 24*x + 0.5*x**2
        else:
            return 25*x**2/2 - 24*x


    def df(x):
        if x < 1:
            return 25*x
        elif 1 <= x < 2:
            return 24 + x
        else:
            return 25*x - 24


    L = 25
    m = 1


    hist, x_opt, X = Momentum_SGD(f, df, x0=3.1, alpha=1/L, beta=0.1, Nsteps=10)


    plt.plot(range(len(hist)), hist)
```

```
plt.xlabel('номер итеррации')
plt.ylabel('loss function value')
plt.title(f'Momentum SGD')
print(f"x* = {x_opt}")
```

```
x* = 2.6304059999999995e-05
```



Посмотрим на нашу функцию

```
Loss = []
for x in np.linspace(-5, 5, 100):
    Loss.append(f(x))
plt.plot(np.linspace(-5, 5, 100), Loss)
plt.xlabel('x')
plt.ylabel('loss function value')
plt.title(f'график зависимости Loss_fun от координаты')
```

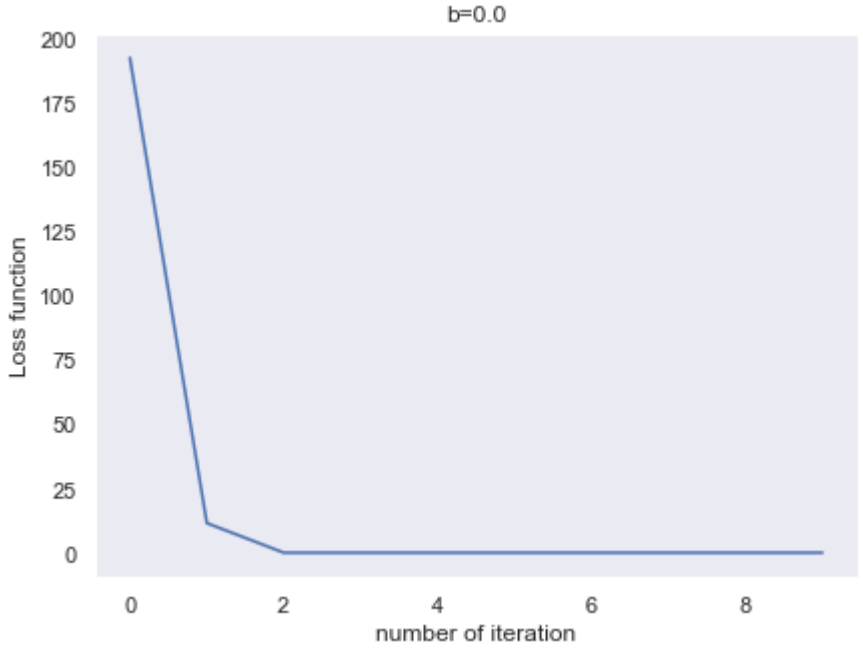
```
Text(0.5, 1.0, 'график зависимости Loss_fun от координаты')
```

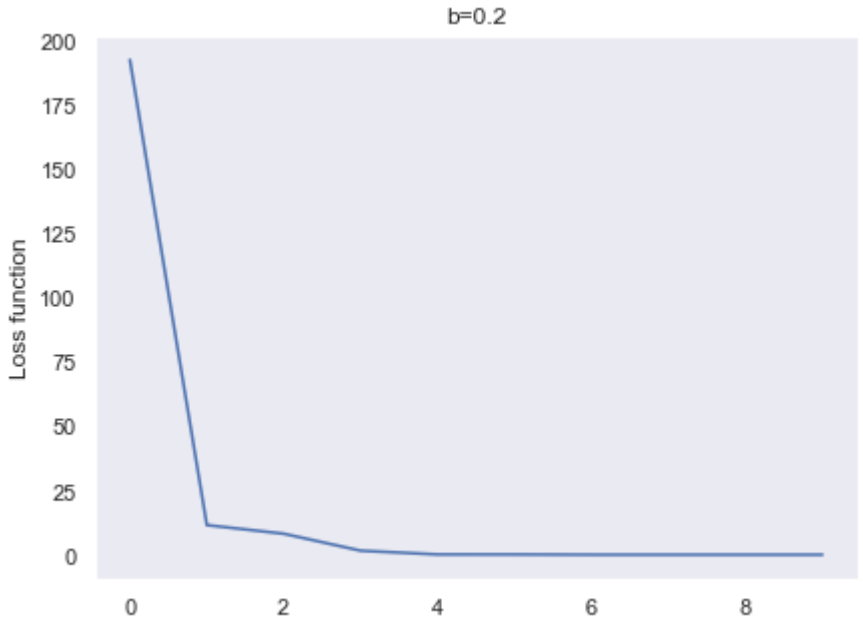
У неё глобальный минимум в 0, но есть ещё и локальный в точке x = 2.

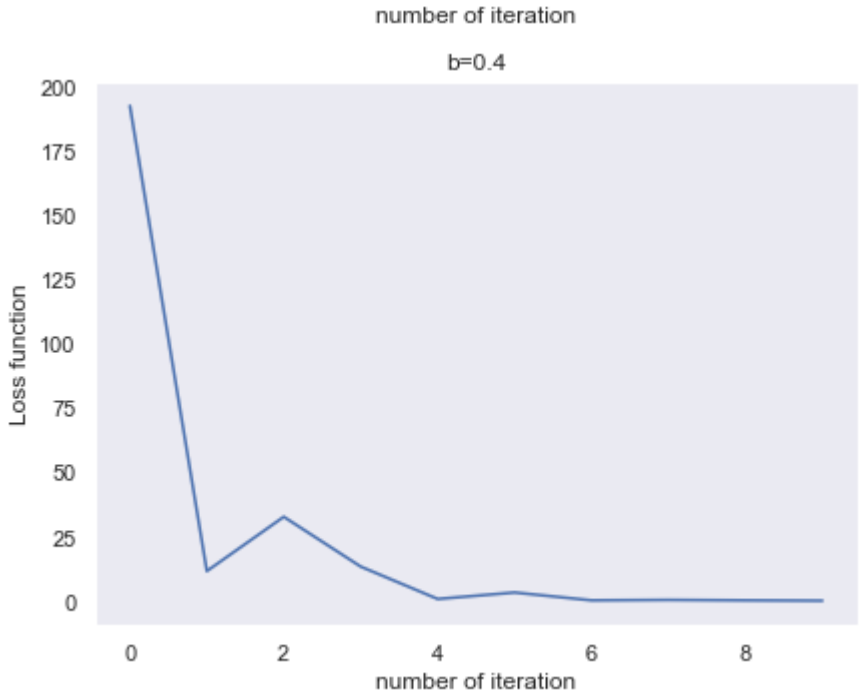
Поподбираем гиперпараметры: зафиксируем α и точку старта. Возьмём её правее 2 (локального минимума) и посмотрим, как значение β будет влиять на сходимость (при $\beta = 0$ мы получим обычный градиентый спуск). α возьмём 1/L, так как тогда у градиентного спуска найлучшая скорость сходимости

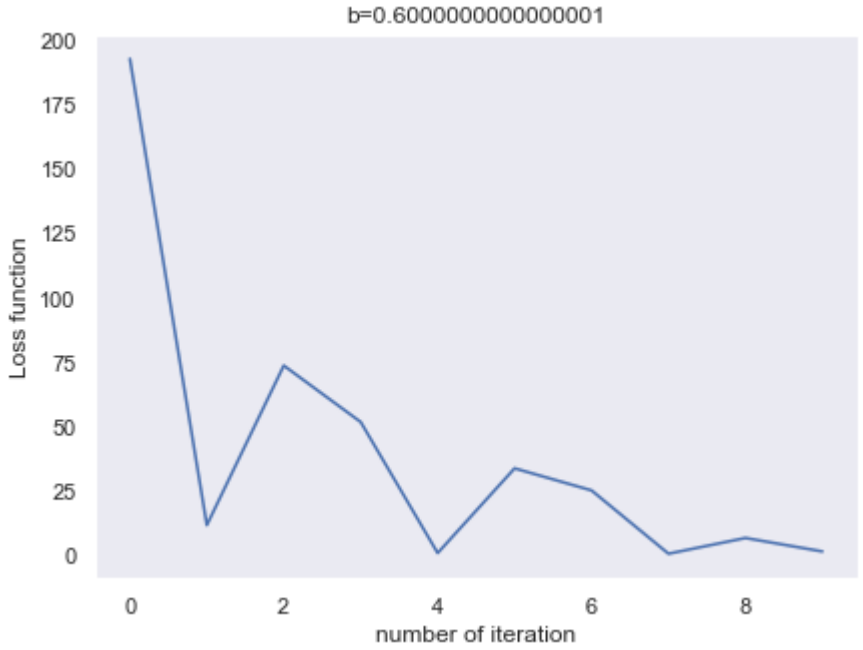
```
for b in np.arange(0, 1, 0.2):
    hist, x_opt, X = Momentum_SGD(f, df, x0=5.0, alpha=1/L, beta=b, Nsteps=10)
    fig = plt.figure(figsize = (15,5))
    ax = plt.subplot(121)
    ax.plot(range(len(hist)), hist, label=f"$\beta$={b}")
    ax.set_ylabel('Loss function')
    ax.set_xlabel('number of iteration')
    ax.grid()
    # ax.legend(loc = 'best')
    ax.set_title(f"b={b}")
    print(x_opt)
```

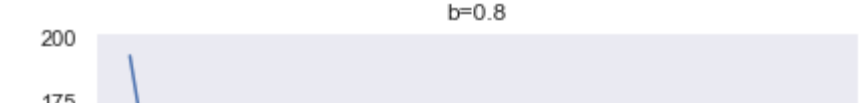
```
0.0
0.001273118719999999
0.051104317440000005
0.24012562268159954
-0.7125725347839998
```






number of iteration
b=0.4
Loss function
200
175
150
125
100
75
50
25
0
0
2
4
6
8
number of iteration


b=0.6000000000000001
Loss function
200
175
150
125
100
75
50
25
0
0
2
4
6
8
number of iteration

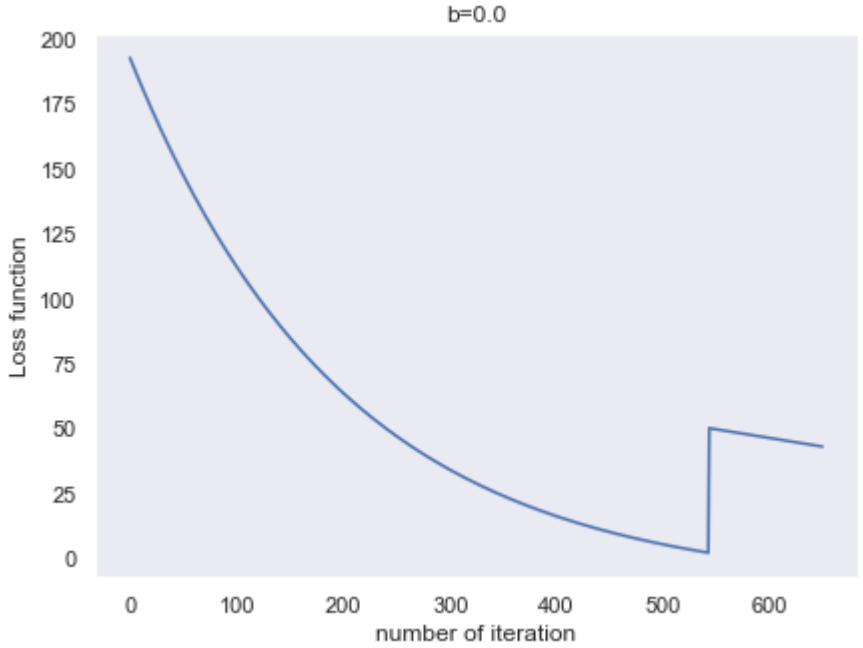

b=0.8
200

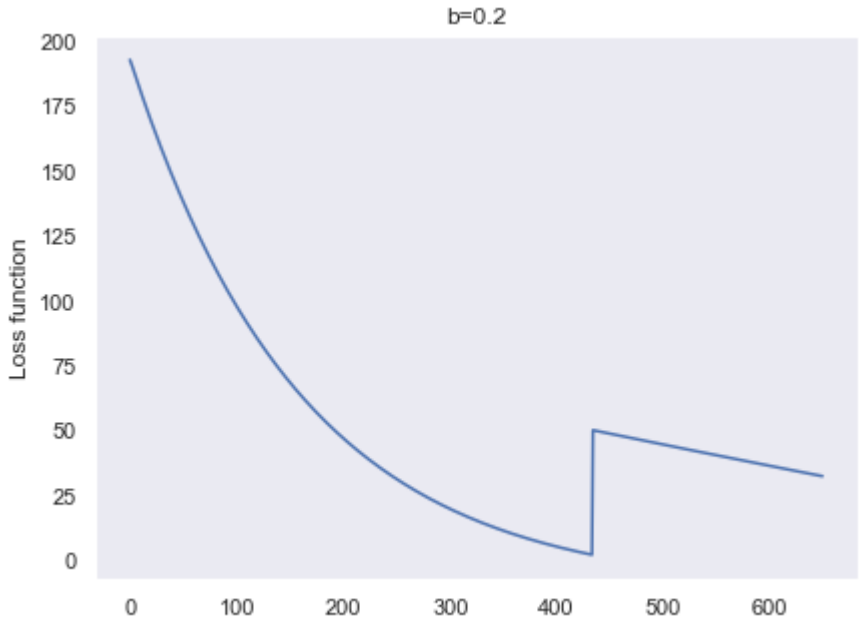
Выводы из этого делаю такие: при больших β тяжёлый шарик начинает больше учитывать значение на предыдущем шаге. Поэтому на первых шагах не уменьшает лосс, а наоборот увеличивает и обычный градиентный спуск работает тут гораздо лучше и легко находит глобальный минимум не упераясь в локальный. Но давайте возьмём α поменьше

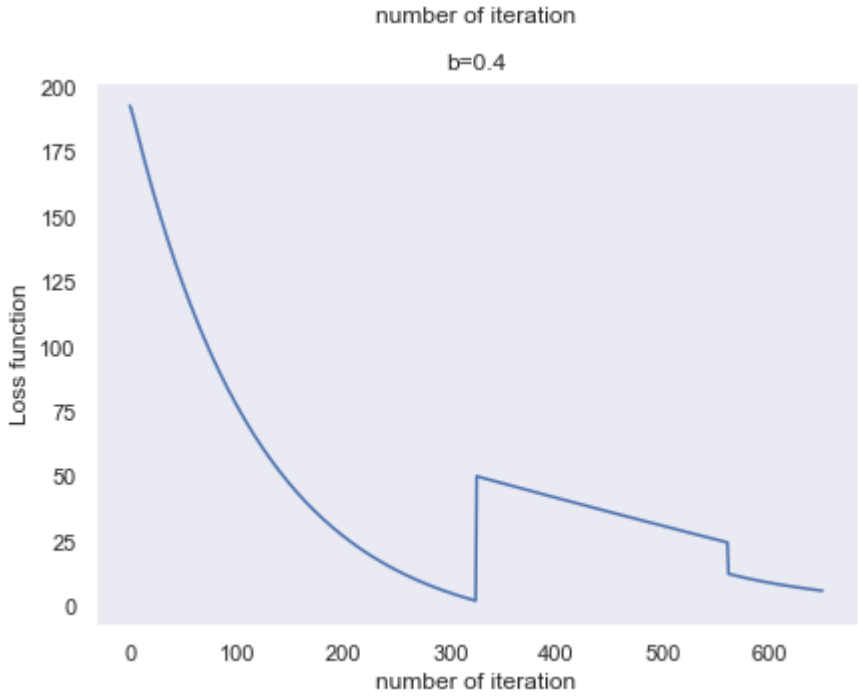
0 2 4 6 8

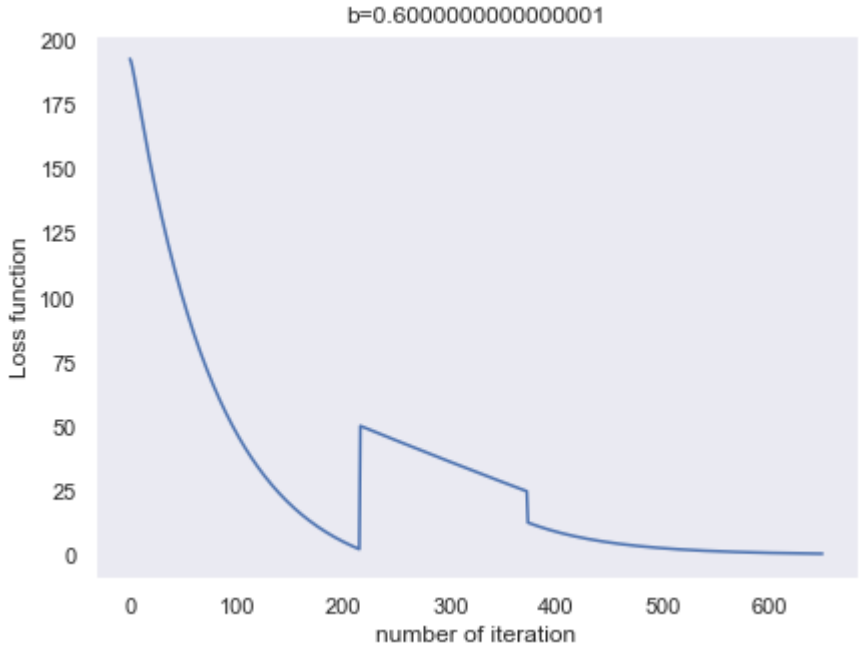
```
for b in np.arange(0, 1, 0.2):
    hist, x_opt, X = Momentum_SGD(f, df, x0=5.0, alpha=0.0001, beta=b, Nsteps=650)
    fig = plt.figure(figsize = (15,5))
    ax = plt.subplot(121)
    ax.plot(range(len(hist)), hist, label=f"$\beta$={b}")
    ax.set_ylabel('Loss function')
    ax.set_xlabel('number of iteration')
    ax.grid()
    # ax.legend(loc = 'best')
    ax.set_title(f"b={b}")
    print(x_opt)
```

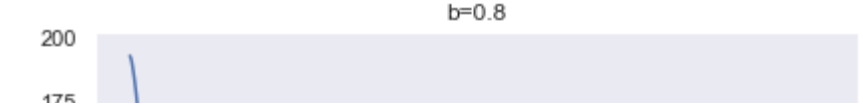
```
1.7210330712334105
1.3056779947172839
0.686163038209825
0.1728686238203956
0.0020601896136340715
```






number of iteration
b=0.4
Loss function
200
175
150
125
100
75
50
25
0
0
100
200
300
400
500
600
number of iteration


b=0.6000000000000001
Loss function
200
175
150
125
100
75
50
25
0
0
100
200
300
400
500
600
number of iteration


b=0.8
200

Для такого маленького α при большем β сходимость будет намного быстрее (мы делаем маленькие шажки в одном и том же направлении, поэтому учитывая предыдущий шаг (ускорение), мы спускаемся быстрее)