1 Sequence convergence

Example 1 *Show with the definition that the sequence* $\frac{1}{k}_{k=1}$ *does not have a linear convergence rate*
$lim_{r->\infty}\frac{r_{k+1}}{r_k} = lim_{r->\infty}\frac{1\cdot k}{k+1} = 1$. Поэтому сходимость сублинейная по тесту отношений.

Example 2 **Determine the convergence or divergence of a given sequence** $r_k = \frac{1}{k^2}$
$lim_{r->\infty}\frac{r_{k+1}}{r_k} = lim_{r->\infty}\frac{k^2}{(k+1)^2} = 1$. Поэтому сходимость сублинейная по тесту отношений.

Example 3 **Show with the definition that the sequence** $r_k = \frac{1}{k^k}$ **does not have a quadratic convergence rate (but it converges to zero)**
$lim_{r->\infty}\frac{r_{k+1}}{r_k} = lim_{r->\infty}\frac{k^k}{(k+1)^k} = lim_{r->\infty}(\frac{k}{k+1})^k \cdot \frac{1}{k+1} = 0$.Подтвердили сходимость к нулю.
Скажем, что $q = 0.9$; $C = 1$. $lim_{r->\infty}(\frac{k}{k+1})^k \cdot \frac{1}{k+1} - 1 \geq Cq^{2k}$ Так как $k+1$ растёт медленнее, чем константа в степени $2k$, так как полином первой степени растёт медленее константы в степени. Другими словами: $lim_{r->\infty}(\frac{k}{k+1})^k \cdot \frac{1}{C} \cdot (\frac{1}{(q^2\cdot(k+1))^{\frac{1}{k}}})^k \geq 1$. Здесь я пытаюсь показать, что корень k-ой степени, а затем степень k "съедят"полином, превратив в константу и $(\frac{1}{q})^{2k}$ устремит дробь к бесконечности, что противоречит определению квадратичной сходимости.

Example 4 **Determine the convergence or divergence of a given sequence** $r_k = 0.707^k$
Возьмём root test. $lim_{r->\infty}r_k\frac{1}{k} = lim_{r->\infty}0.707 = 0.707$ По тесту это линейная сходимость.

Example 5 $r_k = 0.707^{2^k}$
$r_k = (0.707)^{2k+1} \leq C(0.707)^{2k},\ C = 1$, например, получаем квадратичную сходимость.

Example 6 **Show that the sequence** $x_k = 1 + (0.5)^{2k}$ **is quadratically converged to 1.**
Давайте покажем, что есть $M = 100$, что, начиная с некоторого $k = 1$, $r_{k+1} \leq 100r_k^2$, раскроем скобки для $r_k = 1 + (0.5)^{2k} \Rightarrow 1 + (0.5)^{2k+1} \leq 100 + 100 * 2 * 1 * (0.5)^{2k} + (0.5)^{2k} \Rightarrow 0 \leq 99 + ...$
Показано.

Example 7 **Determine the convergence or divergence of a given sequence где в условии** $(\frac{1}{4})^{2k}$
Будем использовать тест отношений, каждый член последовательности $r_k = 0.707^{2^k}$ сходится к 0, при k стремится к бесконечности. Сначала допустим, что $k+1 - even$, а $k - odd$
$lim_{k->\infty}\frac{r_{k+1-even}}{r_{k-odd}} = lim_{k->\infty}\frac{(\frac{1}{4})^{2^{k+1}}}{k^{-1}*(\frac{1}{4})^{2^k}} = lim_{k->\infty}k * (\frac{1}{4})^{2^k} = 0.$

Теперь наоборот, $k - even$, а $k+1 - odd$. $lim_{k->\infty}\frac{r_{k+1-odd}}{r_{k-even}} = lim_{k->\infty}\frac{(\frac{1}{4})^{2^k}}{k^{-1}*(\frac{1}{4})^{2^k}} = lim_{k->\infty}\frac{1}{k} = 0.$
Получаем сверх-линейную сходимость по критерию.

Example 8 $\frac{1}{k^2} - odd;\ \frac{1}{k} - even$
Заметим, что $|r_k| \leq Ck^\alpha;\ C = 1;\ \alpha = -1$. Также $\frac{1}{k^2} \leq r_k \leq \frac{1}{k}$, а обе последовательности имеют сублинейную сходимость, поэтому у r_k ,будет такая же сходимость, иначе бы у последовательности $\frac{1}{k^2}$ она была бы другой, так как эта последовательность меньше либо равна r_k.

Example 9 **Let be a sequence of non-negative numbers and let be some integer. Prove that sequence is linearly convergent with constant if and only if a the sequence converged linearly with constant .**

В одну сторону: пусть r_k сходится линейно, тогда существуют $q; C$, удовлетворяющие условию, начиная с некоторого k, а значит они удовлетворяют условию, начиная с некоторого $k + s$, поэтому r_{k+s} также сходится линейно.
В обратную сторону: пусть r_{k+s} сходится линейно, тогда в тесте отношений для него существует q, удовлетворяющий условиям теста корней: $q = lim_{k->\infty}\ sup_k\ r_k^{\frac{1}{k}}$. Тест корней берётся, по двум причинам, во-первых, r_k неотрицательные, во-вторых, из определения предела следует, что изменение k на некоторое конечное число s не влияет на существование и значение предела, поэтому, что для $k + s$, что для k будет одно и то же значение q, поэтому последовательность r_k также сойдётся линейно. Другими словами, мы использовали предельный переход, чтобы сделать сдвиг на конечное число.
Доказано в обе стороны. (Продолжение на следующей странице)

2 Line search

Example 2 Show that if $0 < c_2 < c_1 < 1$, there may be no step lengths that satisfy the Wolfe conditions (sufficient decrease and curvature condition).
Запишем из 14.pdf Curvative condition: $-\nabla f(x_k - \alpha\nabla f(x_k))^T\nabla f(x_k) \geq c_2\nabla f(x_k)^T(-\nabla f(x_k))$, и допустим, что $c_2 \in (0; c_1)$. Вспоминаем, что левая часть уравнения, это это производная функции $\nabla_\alpha\phi(\alpha)$, и условие кривизны говорит нам, что наклон функции $\phi(\alpha)$ в целевой точке больше наклона $\nabla_\alpha\phi(\alpha)(0)$, домноженного в c_2 раз.
Запишем условие sufficient decrease:

$$f(x_k - \alpha\nabla f(x_k)) \leq f(x_k) - c_1 \cdot \alpha \cdot \nabla f(x_k)^T(\nabla f(x_k))$$

так как $c2 < c1$, то верно неравенство:

$$f(x_k - \alpha\nabla f(x_k)) \leq f(x_k) - c_2 \cdot \alpha \cdot \nabla f(x_k)^T(\nabla f(x_k))$$

или же

$$f(x_k - \alpha\nabla f(x_k)) - f(x_k) \leq c_2 \cdot \alpha \cdot \nabla f(x_k)^T(-\nabla f(x_k))$$

Перенесём альфа в другую часть и подставим сюда Curvative condition:

$$\frac{f(x_k - \alpha\nabla f(x_k)) - f(x_k)}{\alpha} \leq \nabla f(x_k - \alpha\nabla f(x_k))^T\nabla f(x_k)$$

перенесём $\nabla f(x_k)$ из правой в левую и получим,

$$\frac{f(x_k - \alpha\nabla f(x_k)) - f(x_k)}{\alpha \cdot \nabla f(x_k)} \leq -\nabla f(x_k - \alpha\nabla f(x_k))^T$$

и тем более

$$c_2 \cdot \frac{f(x_k - \alpha\nabla f(x_k)) - f(x_k)}{\alpha \cdot \nabla f(x_k)} \leq -\nabla f(x_k - \alpha\nabla f(x_k))^T$$

Получим, что наклон функции $\phi(\alpha)$ в целевой точке меньше либо равен наклону $\nabla_\alpha\phi(\alpha)(0)$, домноженному на c_2, что противоречит Curvative condition, который записан выше. Значит, наше предположение неверно и при $0 < c_2 < c_1 < 1$ у нас может не быть последовательных длин, которые удовлетворяют условиям Вольфа. Доказано.

Example 3 Show that the one-dimensional minimizer of a strongly convex quadratic function always satisfies the Goldstein conditions
Обратимся к интернету и выясним 2 вещи: 1) Строго выпуклая квадратичная функция имеет вид $\frac{1}{2} \cdot x^TAx - b^Tx + const$, 2) одномерный минимизатор зададим, как $\alpha_k = -\frac{\nabla f_k^T p_k}{p_k^T \cdot Ap_k} = -\frac{(x_k^TA - b^T)p_k}{p_k^TAp_k}$, так как градиент данной функции равен: $Ax - b$, считаем A симметричной и положительно определённой.
Запишем условие Гольдштейна : $f(x_k)+(1-c)\alpha_k\nabla f_k^Tp_k \leq f(x_k+\alpha_kp_k) \leq f(x_k)+c\alpha_k\nabla f_k^Tp_k,\ \forall c \in (0, \frac{1}{2})$ (1). Рассмотрим функцию в среднем неравенстве и запишем её разложение до 1-ого порядка: $f(x_k+\alpha_kp_k) = \frac{1}{2}x_k^TAx_k+\alpha_kx_k^TAp_k+\frac{1}{2}\alpha_k^2p_k^TAp_k-b^Tx_k-\alpha_kb^Tp_k+const = f(x_k)+\alpha_kx_k^TAp_k+\frac{1}{2}\alpha_k^2p_k^TAp_k - \alpha_kb^Tp_k$.
Заметим, что $(1-c)\alpha_k\nabla f_k^Tp_k = (1-c)\alpha_k(x_k^TA - b^T)p_k = \alpha_k(x_k^TAp_k - cx_k^TAp_k - b^Tp_k + cb^Tp_k)$, поэтому $c\alpha_k\nabla f_k^Tp_k = \alpha(x_k^TA - b^T)p_k$
Запишем условия Г. другим образом в виде неравенства (1) $\alpha_kx_k^TAp_k - \alpha_kcx_k^TAp_k - \alpha b^Tp_k + \alpha cb^Tp_k \leq \alpha_kx_k^TAp_k + \frac{1}{2}\alpha_k^2p_k^TAp_k - \alpha_kb^Tp_k \leq \alpha_kc(x_k^TA - b^T)p_k$
С одной стороны: $-cx_k^TAp_k + cb^Tp_k \leq \frac{1}{2}\alpha_kp_k^TAp_k \Rightarrow 2c(b^T - x_k^TA)p_k \leq (b^T - x_k^TA)p_k$, где справа записан градиент, неравенства верны для $\forall c \in (0, \frac{1}{2}$, так как $(b^T - x_k^TA)p_k$ неотрицательно

Похожим образом получаем для второго условия: $x_k^T A p_k + \frac{1}{2}\alpha_k p_k^T A p_k - b^T p_k \leq c(x_k^T A - b^T)p_k$ переносим в другую часть и выносим с за скобку $\alpha_k p_k^T A p_k \leq 2(c-1)(x_k^T A - b^T)p_k$ В левой меняем местами неравенства из вывода выше, а в правой части меняем знак. $(b^T - x_k^T A)p_k \leq 2(1-c)(b^T - x_k^T A)p_k$

Так как $(b^T - x_k^T A)p_k$ неотрицательно, то сокращаем обе части и $2(1-c) \geq 1$, поэтому $c \leq \frac{1}{2}$. Получим выполнение условий Гольдштейна $f(x_k) + (1-c)\alpha_k \nabla f_k^T p_k \leq f(x_k + \alpha_k p_k) \leq f(x_k) + c\alpha_k \nabla f_k^T p_k,\ \forall c \in (0, \frac{1}{2})$

3 Gradient descent

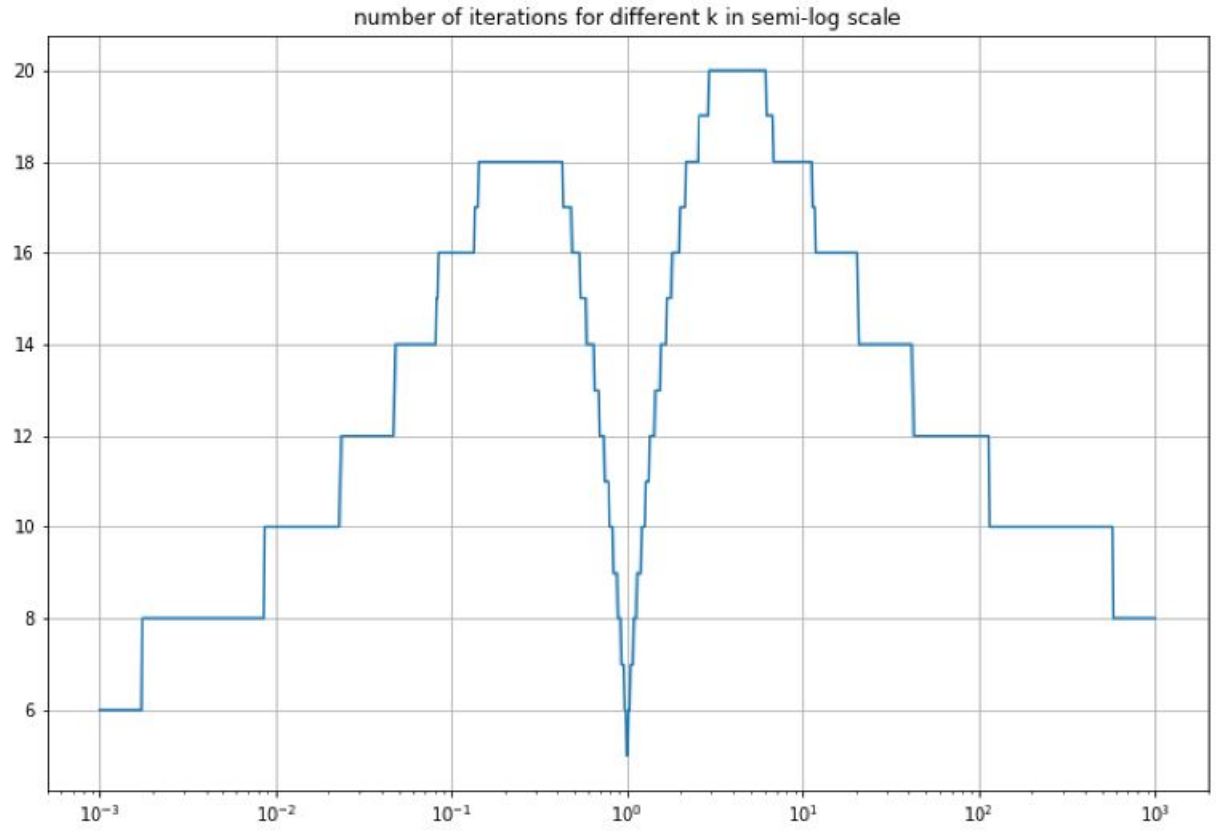
Hobbit village Закодено!

Functional class Утверждение: у функции, дважды непрерывно дифференцируемой, гессиан - симметричная матрица, то есть смешанные производные равны при разном порядке дифференцирования, поэтому спектральная норма равна $||\nabla^2 f(x)|| = max_i|\lambda_i|$. Так как положительная полуопределённость матрицы равносильна тому, что её собственные числа $\lambda_i \geq 0$, то $L \geq ||\nabla^2 f(x)|| <=> -LI \preceq \nabla^2 f(x) \preceq LI$
Пусть выполнена левая часть условия, тогда $\forall x, y$ применим разложение по Тейлору: $\nabla f(y) = \nabla f(x) + \nabla^2 f(\xi)(y - x), \ \ \forall \xi \in [x, y]$ Из чего следует $||\nabla f(x) - \nabla f(y)|| = ||\nabla^2 f(\xi)(y - x)|| \leq ||\nabla^2 f(x)|| \cdot ||x - y||||x - y|| \Rightarrow f \in C_L^{2,1}$ Доказано в одну сторону. В обратную есть контрпример.
.
Здесь признаюсь, что мне подсказали с x^2, "Искать положительную функцию, нужно перевернуть в нуле". Пусть $f(x) = x^2 * 0.5 \ x >= 0; \ and \ f(x) = -x^2 * 0.5 \ \ x < 0$.
Найдём её градиент = $\nabla f(x) = |x| \ x >= 0, \nabla f(x) = |x|; \ x < 0$. Тогда $\nabla^2 f(x) = sign(x)$
Очевидно, что $L = 1$. Очевидно, что $||sign(x)|| = 1 = L$. Поэтому она должна принадлежать классу, указанному в условии. Но она не принадлежит ему, так как $sign(x)$ не непрерывна в нуле, что явно выражено на графике. Получим противоречие. В обратную сторону теорема не работает. Грустно.
Вторая часть:
От противного: пусть существует точка, в которой, $||\nabla^2 f(x)|| > L$. Тогда существует собственное значение $\nabla^2 f(x)$, некоторое $\lambda_i : \ |\lambda| > L$. Для удобства возьмём его положительным, а h собственным вектором единичной длины. Получим, при $t \neq 0$ по Тейлору: $\nabla f(x + th) - \nabla f(x) = \nabla^2 f(x)th + o(t) = \lambda th + r(t) \cdot t$ где лучший факультет $||r(t)|| = o(1)$. Перенесём th в левую часть и применим "умный ноль"ю $\frac{\nabla f(x+th)-(x)}{x+th-x} = ||\lambda h + r(t)|| \Rightarrow \lambda > L \ for \ t \to 0$. Взяв достаточно малое t получим, что неравенство верно. Тогда $f \notin C_L^{2,1}$. Поэтому наше предположение неверно и данное условие можно записать в таком виде.

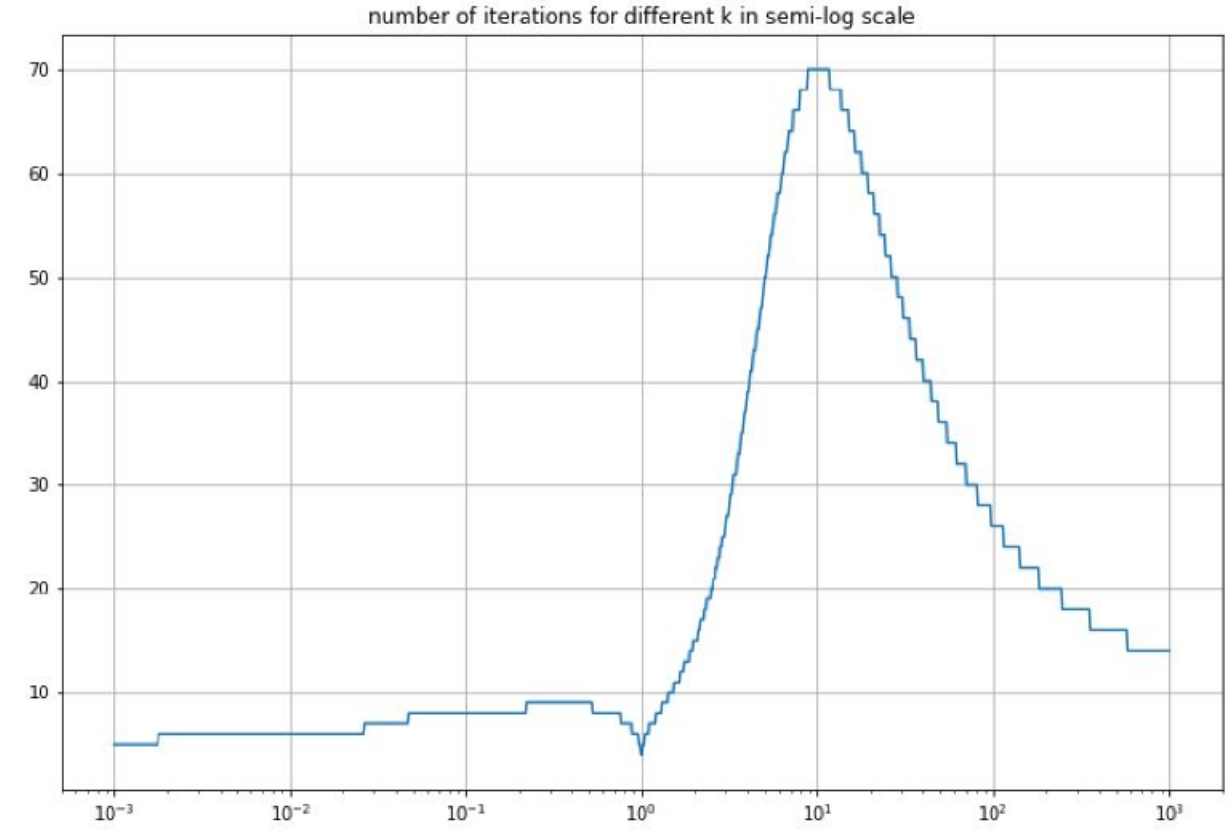
How condition number affects gradient optimization

3.1 Функция $f(x_1, x_2 = x_1^2 + kx_2^2$

Рассмотрим различные начальные точки. Если взять её, например, $x_0 = (10, 10)$ тогда случаи коэффициента $k = 10; \ k = 10^{-1}$ будут симметричными и также для других степеней, то есть график числа итераций получается симметричным относительно единицы для k.

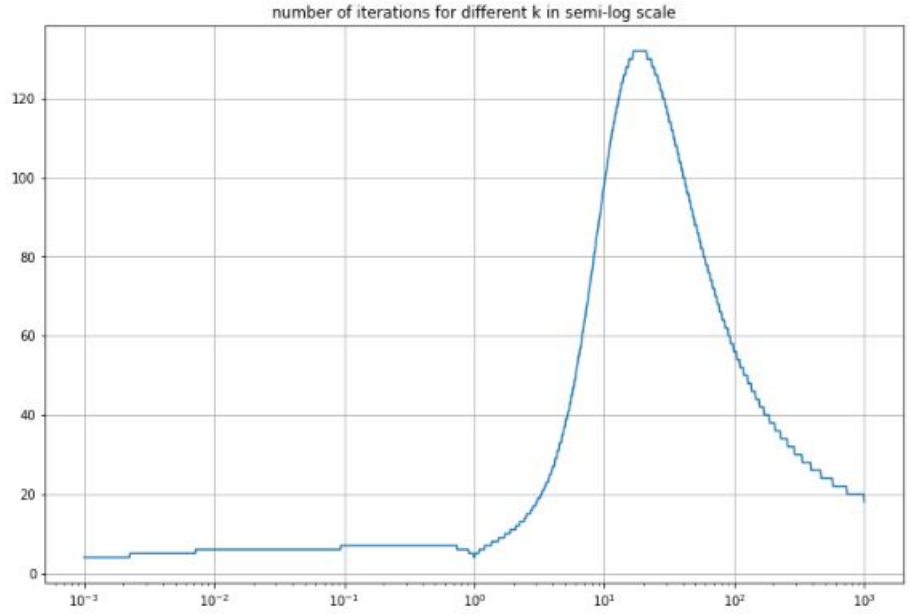


Смещаем начальную точку в сторону оси X_1, например для точки $(10, 2)$:

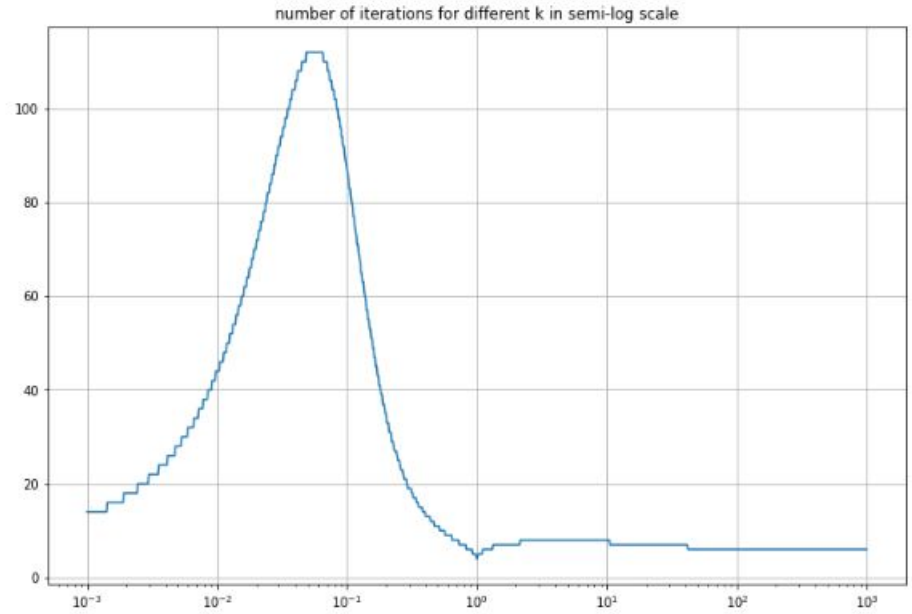


Для наглядности добавим точки (10,1) (1,10), (10,5) код будет прикреплён в колабе.

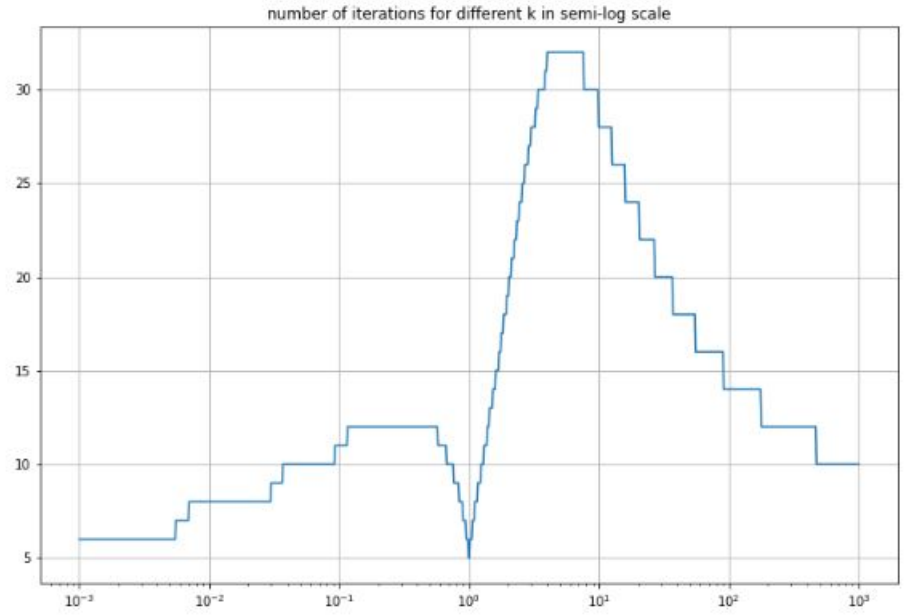
Точка (10,1)



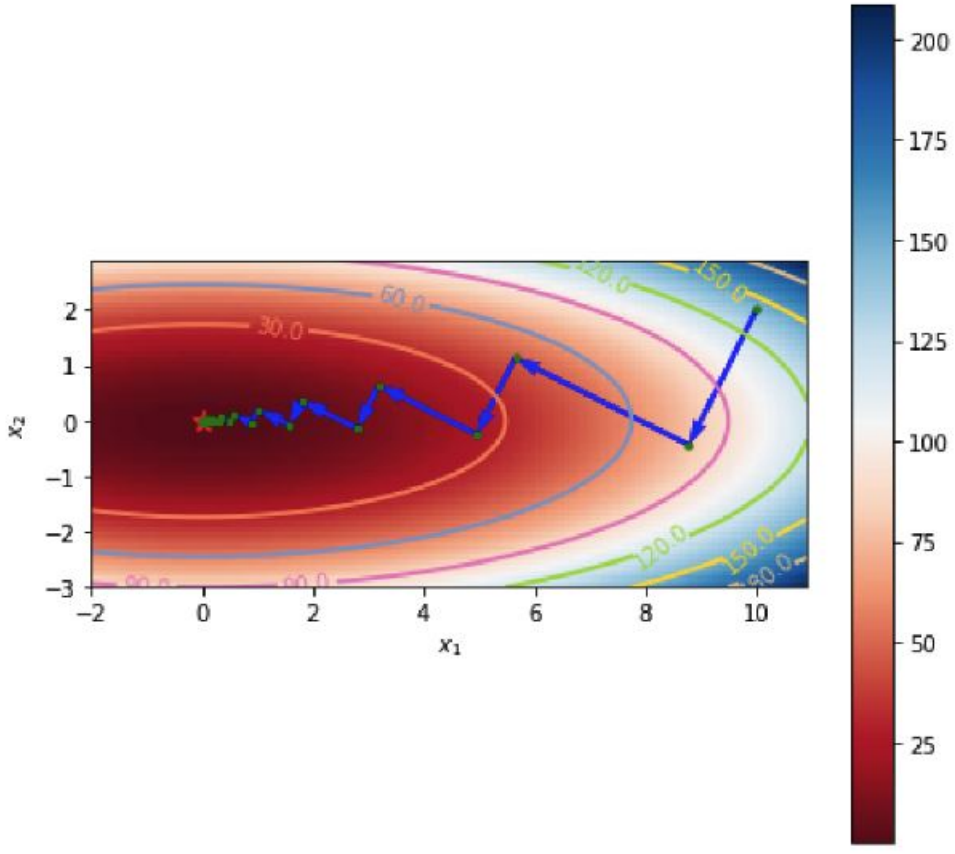
Точка (1,10)



Точка (10,5)

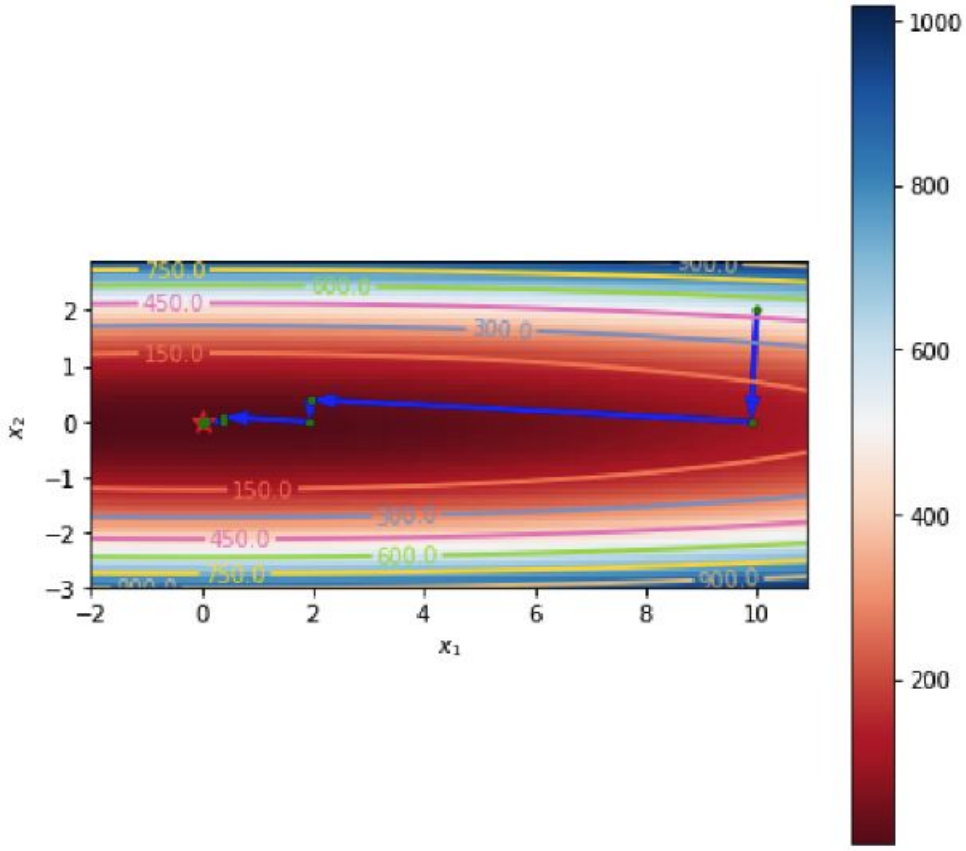


Рассмотрим случай $k = 10$. При нём спуск идёт под углом почти в 45 градусов по отношению к осям эллипса и шагов получается много, так при такой расстановке стрелочка направлена постоянно "мимо"центра. То есть уменьшение функции на каждой итерации невелико.



Зато, попав практически вдоль оси, мы можем попасть в точку, очень близкую к оптимуму, и так как мы проходим в этом случае почти по касательной к линии уровня, то перпендикуляр к ней является наиболее оптимальным шагом и его спуск и совершает, поэтому шагов сильно меньше. При малом k минимум находится в узком овраге, и мы имеем стандартный случай, в котором наискорейший спуск сходится медленно. Это верно для значений k сильно меньших

10 или сильно больших 10. Возьмём k = 100.

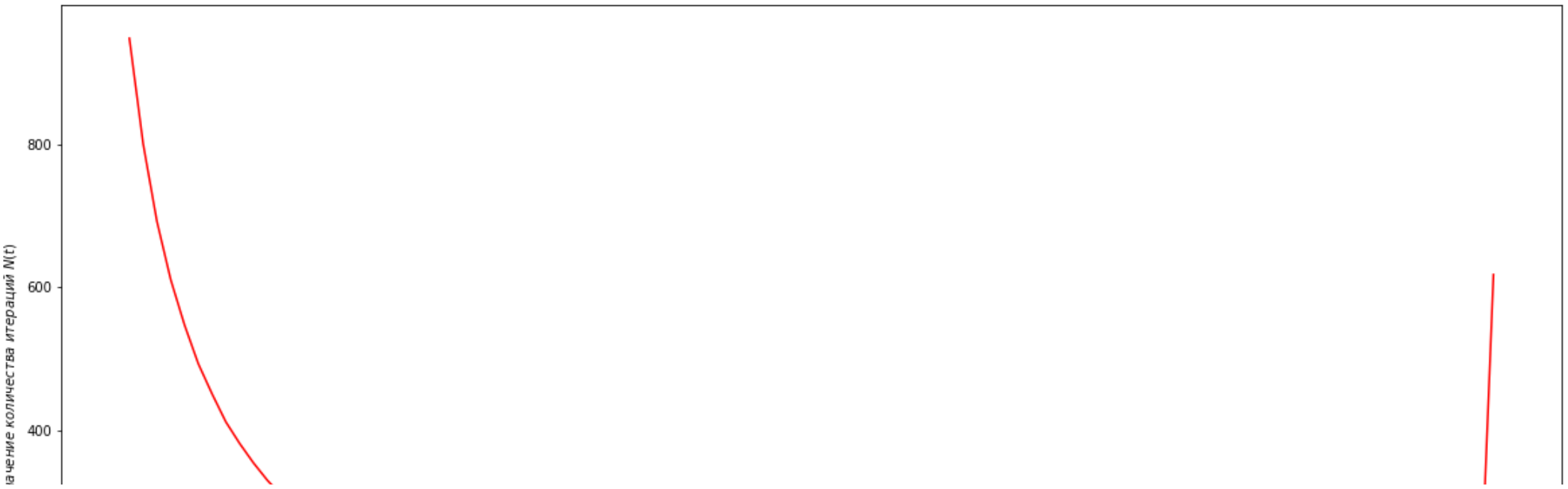


И принтскрин кода, который также есть в колабе:

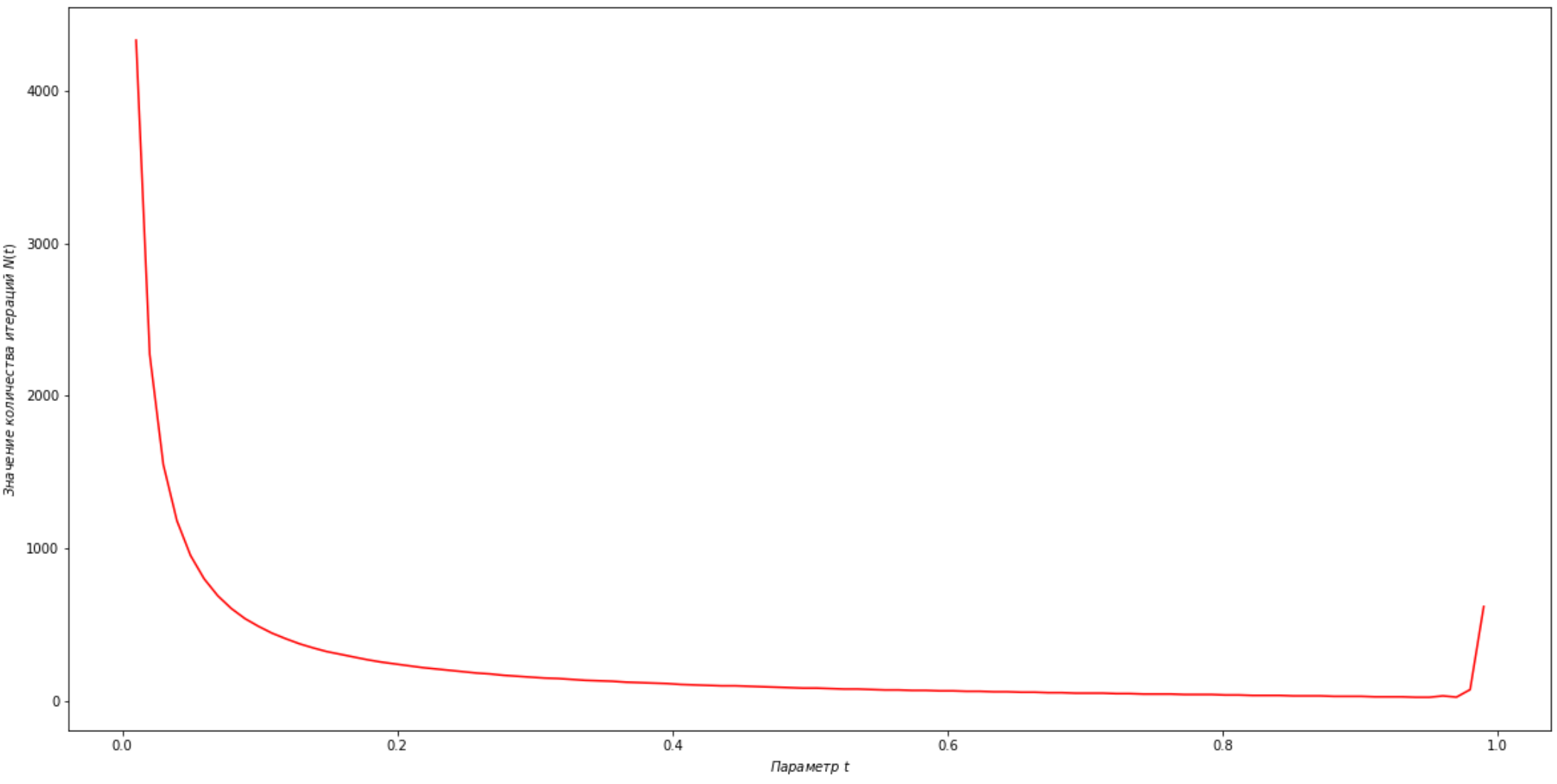
```
net_space = np.logspace(-3,3,1000)
n_iter = []
x_0 = [10,5]
for i in net_space:
 step = steepest_descent(x_0, f, df, i, df_eps = 1e-7)
 n_iter.append(len(step))
fig, ax = plt.subplots(figsize = (12,8))
ax.semilogx(net_space, n_iter)
ax.set(title='number of iterations for different k in semi-log net\scale')
ax.grid()
fig.savefig("t_right_semilog.png")
plt.show()
```

Projected gradient descent Запишем функцию Лаганжа $L(..) = f(x) + consractioms$ или же $L(\nu, x) = ||Ax - b||_2^2 + \nu^T(Cx - d)$. Теперь по теореме ККТ допишем условия дополняющей нежёсткости: $\lambda_i g_i = 0$ или же $2A^T(Ax - b) + C^T\nu = 0 = \nabla_x L$. Второе условие имеет вид: $(Cx - d) = 0$. По условию $rgA = n;\ rgC = k$, то есть максимальный, поэтому существует псевдообратная матрица. Выразим x через неё: $x = (A^TA)^{-1}(A^Tb - \frac{1}{2}C^T\nu)$ и вторая часть $x = C^{-1}d$.
Рассмотрим условие Слейтра: для них необходимо $f(x)_i < 0$, однако таких ограничений в условии нет, к тому же существует $x_{extr} :\ Cx_{extr} = d$, то есть ограничение вида $Cx = d$ выполнено. Значит, оптимальное решение двойственной $g(\nu) = inf_x L(x, \nu) \to max$ совпадает с решением экстремальной задачи в теореме ККТ. Подставим x в $Cx = d$: $C(A^TA)^{-1}(A^Tb - \frac{1}{2}C^T\nu*) = d => \nu* = -2(C^T)^{-1}((A^TA)C^{-1}d - A^Tb)$ Наконец, найдём решение по выражению выше: $x_{sol} = (A^TA)^{-1}(A^Tb - \frac{1}{2}C^T\nu*) = (A^TA)^{-1}(A^Tb - \frac{1}{2}C^T \cdot -2(C^T)^{-1}((A^TA)C^{-1}d - A^Tb)) = (A^TA)^{-1}(A^Tb - \frac{1}{2}C^T \cdot (-2)(C^T)^{-1}((A^TA)C^{-1}d - A^Tb))$ Приводим подобные и получаем: $(A^TA)^{-1}(-1 \cdot C^T((C(A^TA)^{-1}C^T)^{-1} \cdot (C(A^TA)^{-1}A^Tb - d)) + A^Tb)$
А программная часть будет в колабе.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from mpl_toolkits.axes_grid1 import make_axes_locatable
import operator
import imageio
%matplotlib inline
import matplotlib.pyplot as plt
# Имплементируем функцию на заданном интервале
arr = np.linspace(-20.0,0.0,200)
def func(x):
  return (x + np.sin(x))*np.exp(x)
plt.plot(arr, func(arr))
plt.xlabel('$x$', fontsize = 10)
plt.ylabel('$f(x)$', fontsize = 10)
plt.show()


#Имплементриуем линейный поиск
def Ln_search(func, a, b, t, epsilon = 1e-7):
  N = 0
  dot = a + (b - a)*t #точка разбиения
  while(b-a > epsilon):
    next_dot = a + (b - a)*t
    N += 2
    if func(next_dot) <= func(dot): # То есть ближе к минимуму
      b = dot
      dot = next_dot
    else: # Если наоборот дальше от минимума
      z = b + (dot - b)*t
      N+=1 # +1 шаг алгоритма
      if func(dot) <= func(z):
        b = z
        a = next_dot
      else:
        a = dot
        dot = z
  return (a+b)/2 ,N



"""
Здесь аккуратно, так как крайние значения t = 0 или 1 крайне неоптимальны, так как на графике, если точка разбиения слева, то она не сильно мен
С другой стороны, если точка разбиения справа, то алгоритм будет постоянно промахиваться мимо минимума и это тоже не оптимальный случай. Поэтом
"""
arr = np.linspace(0.05,0.99,100)
N_min = 1e20
t_min = 0
N_mean = []
for t in arr:
  N_value = []
  for j in range(100): #Берём сразу 100 значений функции и находим среднее среди них, чтобы минимизировать случайную погрешность
    x, N =Ln_search(func, -20 , 0, t, epsilon = 1e-7)
    if (N_min > N):
      N_min = N
      t_min = t
    N_value.append(N)
  N_value = np.array(N_value)
  N_mean.append(N_value.mean())
#print(N_)
plt.figure(figsize=(20,10))
plt.plot(arr,N_mean,color = 'red')
plt.xlabel('$Параметр ~t$', fontsize = 10)
plt.ylabel('$Значение~ количества~ итераций~ N(t)$', fontsize = 10)
plt.show()
print("Минимальное значение соответствует точке t = ",t_min, "N(t) = ",N_min )
```



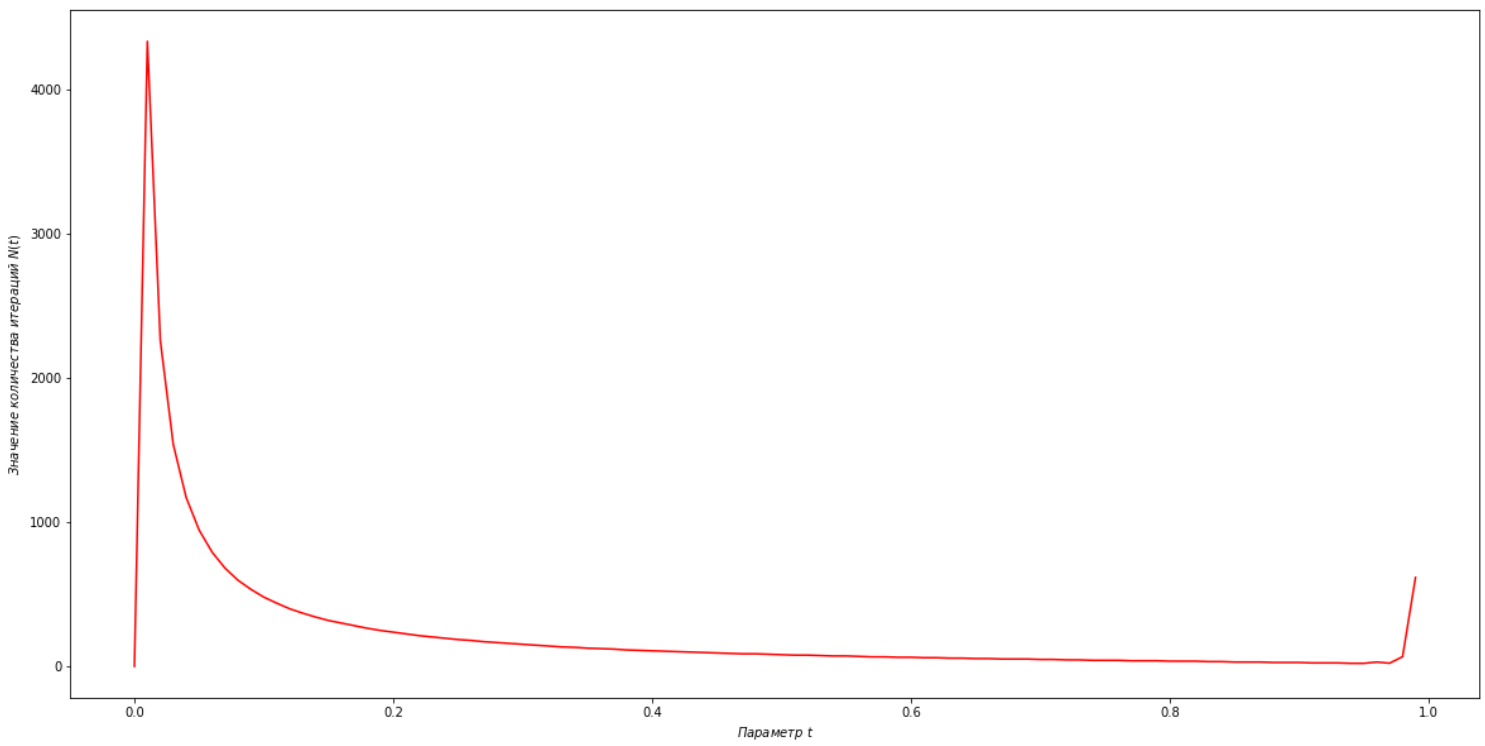

```
arr = np.linspace(0.01,0.99,100)
N_min = 1e20
t_min = 0
N_mean = []
for t in arr:
  N_value = []
  for j in range(100): #Берём сразу 100 значений функции и находим среднее среди них, чтобы минимизировать случайную погрешность
    x, N =Ln_search(func, -20 , 0, t, epsilon = 1e-7)
    if (N_min > N):
      N_min = N
      t_min = t
    N_value.append(N)
  N_value = np.array(N_value)
  N_mean.append(N_value.mean())
#print(N_)
plt.figure(figsize=(20,10))
plt.plot(arr,N_mean,color = 'red')
plt.xlabel('$Параметр ~t$', fontsize = 10)
plt.ylabel('$Значение~ количества~ итераций~ N(t)$', fontsize = 10)
plt.show()
print("Минимальное значение соответствует точке t = ",t_min, "N(t) = ",N_min )
```




```
Минимальное значение соответствует точке t =  0.9405050505050505 N(t) =  23
```

```
arr = np.linspace(0,0.99,100)
N_min = 1e20
t_min = 0
N_mean = []
for t in arr:
  N_value = []
  for j in range(100): #Берём сразу 100 значений функции и находим среднее среди них, чтобы минимизировать случайную погрешность
    x, N =Ln_search(func, -20 , 0, t, epsilon = 1e-7)
    if (N_min > N):
```

```
      N_min = N
      t_min = t
    N_value.append(N)
  N_value = np.array(N_value)
  N_mean.append(N_value.mean())
#print(N_)
plt.figure(figsize=(20,10))
plt.plot(arr,N_mean,color = 'red')
plt.xlabel('$Параметр ~t$', fontsize = 10)
plt.ylabel('$Значение~ количества~ итераций~ N(t)$', fontsize = 10)
plt.show()
print("Минимальное значение соответствует точке t = ",t_min, "N(t) = ",N_min )
```




```
Минимальное значение соответствует точке t =  0.0 N(t) =  2
```

```
"""
Если справа поставить 1, код будет очень долго компилироваться
"""
arr = np.linspace(0.05,1,100)
N_min = 1e20
t_min = 0
N_mean = []
for t in arr:
  N_value = []
  for j in range(100): #Берём сразу 100 значений функции и находим среднее среди них, чтобы минимизировать случайную погрешность
    x, N =Ln_search(func, -20 , 0, t, epsilon = 1e-7)
    if (N_min > N):
      N_min = N
      t_min = t
    N_value.append(N)
  N_value = np.array(N_value)
  N_mean.append(N_value.mean())
#print(N_)
plt.figure(figsize=(20,10))
plt.plot(arr,N_mean,color = 'red')
plt.xlabel('$Параметр ~t$', fontsize = 10)
plt.ylabel('$Значение~ количества~ итераций~ N(t)$', fontsize = 10)
plt.show()
print("Минимальное значение соответствует точке t = ",t_min, "N(t) = ",N_min )
```

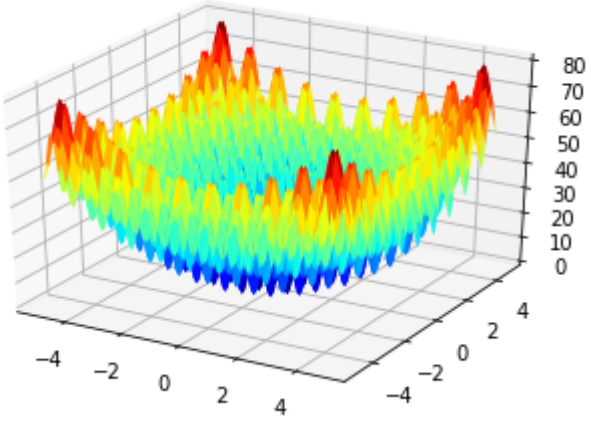
```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-9-c870fb797f51> in <module>()
      6   N_value = []
      7   for j in range(100): #Берём сразу 100 значений функции и находим среднее среди них,
чтобы минимизировать случайную погрешность
----> 8     x, N =Ln_search(func, -20 , 0, t, epsilon = 1e-7)
      9     if (N_min > N):
```

```
<ipython-input-1-4eb4e9f05c8a> in func(x)
     10 arr = np.linspace(-20.0,0.0,200)
     11 def func(x):
---> 12   return (x + np.sin(x))*np.exp(x)
     13 plt.plot(arr, func(arr))
     14 plt.xlabel('$x$', fontsize = 10)
```

11 сек. выполнено в 15:44

```
from numpy import arange
import numpy as np
from numpy import exp
from numpy import sqrt
from numpy import cos
from numpy.random import rand
from numpy import e
from numpy import pi
from numpy import meshgrid
from matplotlib import pyplot
from mpl_toolkits.mplot3d import Axes3D
from scipy.optimize import basinhopping
import math
import scipy.optimize as opt
import matplotlib.pyplot as plt


def objective(v):
  dim = v.size
  res = 10*dim + sum([np.power(v_i, 2) - 10*np.cos(2*math.pi*v_i) for v_i in v])
    #sum += np.power(array[i], 2) - 10*np.cos(2*math.pi*array[i])
  return res

d = 2
# objective function
def objective(v):
  array = [0]*d
  sum = 0
  for i in range(d):
    array[i] = v[i]
  for i in range(d):
    sum += np.power(array[i], 2) - 10*np.cos(2*math.pi*array[i])
  return (10*2 + sum)
# define range for input
r_min, r_max = -5.0, 5.0
# sample input range uniformly at 0.1 increments
xaxis = arange(r_min, r_max, 0.1)
yaxis = arange(r_min, r_max, 0.1)
# create a mesh from the axis
x, y = meshgrid(xaxis, yaxis)
# compute targets
v = [0]*d
v[0] = x
v[1] = y
v[1] = y
results = objective(v)
# create a surface plot with the jet color scheme
figure = pyplot.figure()
axis = figure.gca(projection='3d')
axis.plot_surface(x, y, results, cmap='jet')
# show the plot
```

```
<mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x7f5e35082510>
```




```
d_arr = [1, 2, 3, 4, 5]
bhop = {}
bhop = [[0]*20 for i in d_arr]
de = {}
de  =[[0]*20 for i in d_arr]
shgo = {}
shgo= [[0]*21 for i in d_arr]
duaann = {}
duaann= [[0]*20 for i in d_arr]
brute = {}
brute = [[0]*20 for i in d_arr]
d = 2
zer_0 = np.ones(d)
#print(de)
```

```
[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0,
```

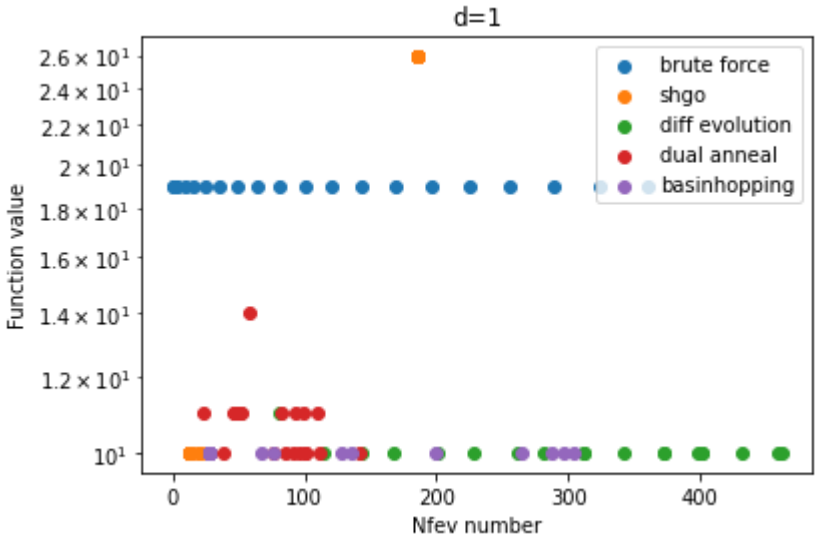
```
for d in range(1,6):
  for i in range(-10, 10):
    borders = [(-5,5) for i in range(2**d)]
    valbhop = opt.basinhopping(objective,np.ones(2**d),niter=i)

    bhop[d-1][i+10]= (valbhop.nfev, valbhop.fun)
    """
    valde = opt.differential_evolution(objective, borders, maxiter=i)
    de[d-1][i+10]= ((valde.nfev, valde.fun))
    valdua = opt.dual_annealing(objective, borders, maxiter=i)
    duaann[d-1][i+10]= ((valdua.nfev, valdua.fun))
    """



for d in range(1,6):
  for i in range(-10, 10):

    borders = [(-5,5) for k in range(2**d)]
    valde = opt.differential_evolution(objective, borders, maxiter=i+10)
    de[d-1][i+10]= ((valde.nfev, valde.fun))


for d in range(1,6):
  for i in range(1, 21):
    borders = [(-5,5) for i in range(2**d)]
    valdua = opt.dual_annealing(objective, borders, maxiter=i)
    duaann[d-1][i-1]= ((valdua.nfev, valdua.fun))
print(duaann)
```

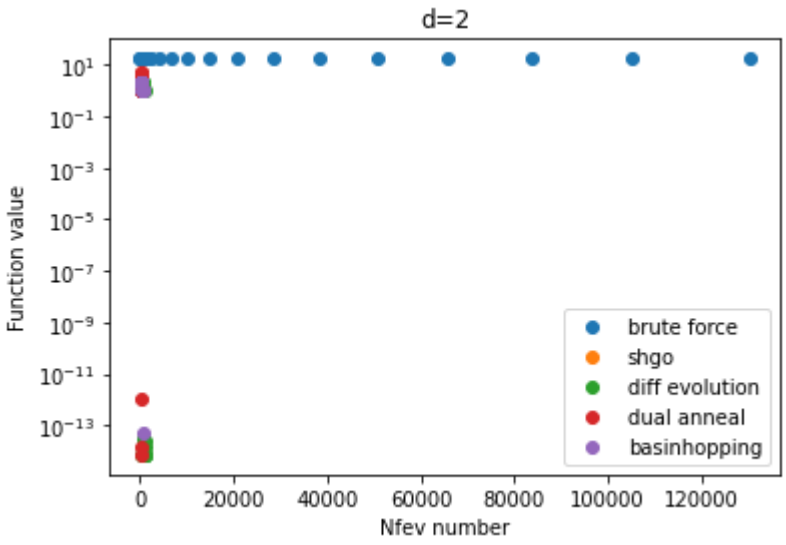
```
[[(23, 10.994959057093297), (39, 10.000000000000005), (46, 10.994959057093302), (59, 13.979831190554092), (48, 10.994959057093297), (52,
```

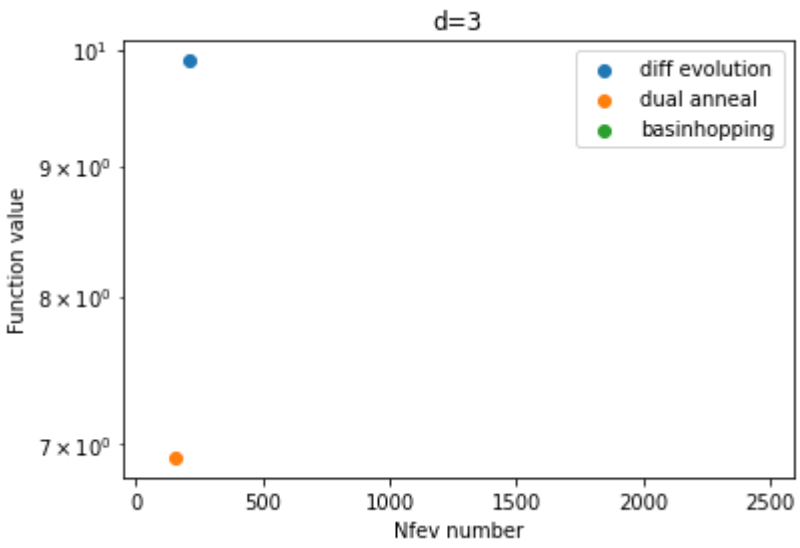
```
for d in range(1,3):

  for i in range(1, 22):
    borders = [(-5,5) for i in range(2**d)]

    valshgo = opt.shgo(objective, borders, options={"maxfev":i})
    shgo[d-1][i-1]= ((valshgo.nfev, valshgo.fun))

print(shgo[2])
```

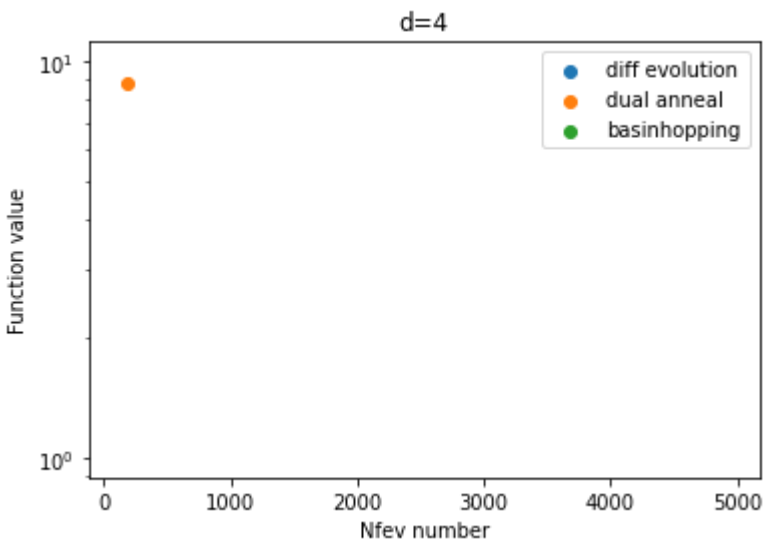
```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

```
for d in range(1,3):

  for extr in range(1, 21):
    borders = [slice(-5,5, 1.1) for i in range(2**d)]
    zer_0, Funca_val, net, jout = opt.brute(objective, borders, Ns = extr, full_output=True)
    brute[d-1][extr-1]= (((extr-1)**(2**d), Funca_val))



for d in range(1,6):
  if d < 3:
    plt.scatter(*zip(*brute[d-1]), label = 'brute force')
    plt.scatter(*zip(*shgo[d-1]), label = "shgo")
  plt.scatter(*zip(*de[d-1]) , label= 'diff evolution')
  plt.scatter(*zip(*duaann[d-1]) , label="dual anneal")
  plt.scatter(*zip(*bhop[d-1]), label = 'basinhopping')

  plt.xlabel('Nfev number')
  plt.ylabel('Function value')
  plt.yscale('log')
  plt.legend()
  plt.title(f'd={d}')
  plt.savefig(f"d={d}.png")
  plt.show()
```

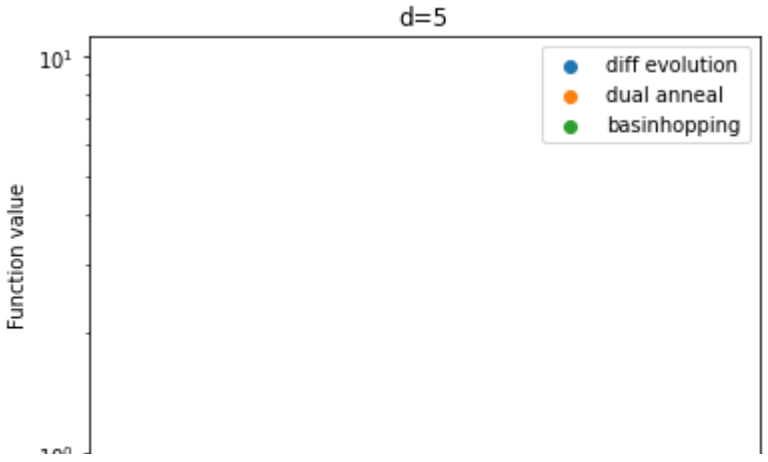









```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:11: UserWarning: Data has no positive values, and therefore cannot be log-sc
  # This is added back by InteractiveShellApp.init_path()
```



Домашнее задание 1.

The file should be sent in the `.pdf` format created via $LaTeX$ or typora or printed from pdf with the colab\jupyter notebook. The only handwritten part, that could be included in the solution are the figures and illustrations.

Deadline: 15.04.21 21:59:59

```
### YOUR CODE HERE
```

2. Show that if $0 < c_2 < c_1 < 1$, there may be no step lengths that satisfy the Wolfe conditions (sufficient decrease and curvature condition).
3. Show that the one-dimensional minimizer of a strongly convex quadratic function always satisfies the Goldstein conditions.

Zero order optimization

Global optimization with scipy

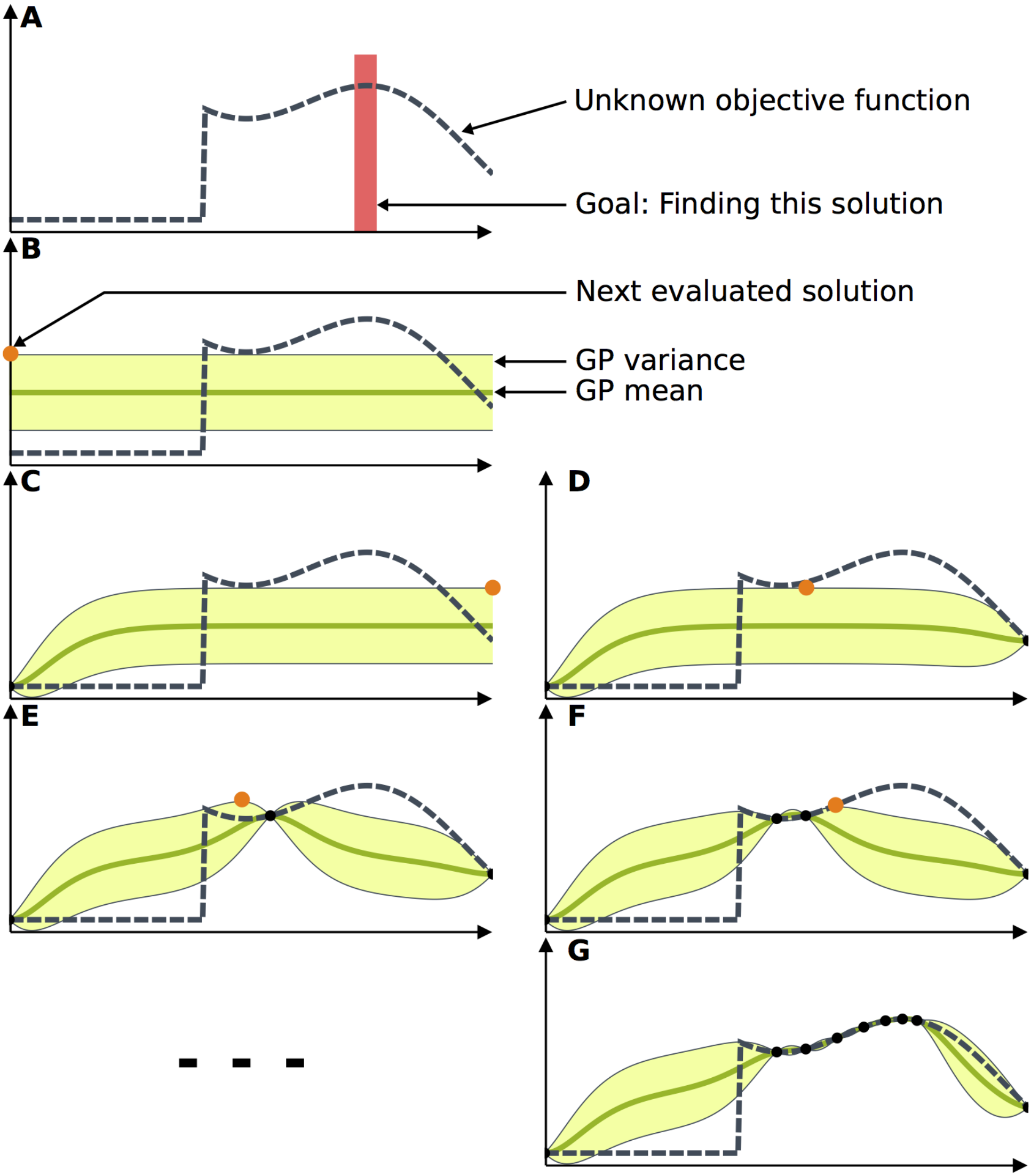
Implement Rastrigin function $f : \mathbb{R}^d \to \mathbb{R}$ for d = 10. link

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

- Consider global optimization from here.
- Plot 5 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 $x_i \in [-5, 5]$.

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.



.

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
```

```
     Collecting optuna
       Downloading optuna-2.10.0-py3-none-any.whl (308 kB)
          |████████████████████████████████| 308 kB 5.3 MB/s
     Collecting alembic
       Downloading alembic-1.7.7-py3-none-any.whl (210 kB)
          |████████████████████████████████| 210 kB 30.8 MB/s
     Requirement already satisfied: scipy!=1.4.0 in /usr/local/lib/python3.7/dist-packages (from optuna) (1.4.1)
     Requirement already satisfied: PyYAML in /usr/local/lib/python3.7/dist-packages (from optuna) (3.13)
     Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from optuna) (4.64.0)
     Collecting cmaes>=0.8.2
       Downloading cmaes-0.8.2-py3-none-any.whl (15 kB)
     Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from optuna) (1.21.5)
     Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.7/dist-packages (from optuna) (21.3)
     Collecting cliff
```

```
  Downloading cliff-3.10.1-py3-none-any.whl (81 kB)
     |████████████████████████████████| 81 kB 5.6 MB/s
Collecting colorlog
  Downloading colorlog-6.6.0-py2.py3-none-any.whl (11 kB)
Requirement already satisfied: sqlalchemy>=1.1.0 in /usr/local/lib/python3.7/dist-packages (from optuna) (1.4.35)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/dist-packages (from packaging>=20.0->optuna) (3.0.8)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.7/dist-packages (from sqlalchemy>=1.1.0->optuna) (4.11.3)
Requirement already satisfied: greenlet!=0.4.17 in /usr/local/lib/python3.7/dist-packages (from sqlalchemy>=1.1.0->optuna) (1.1.2)
Collecting Mako
  Downloading Mako-1.2.0-py3-none-any.whl (78 kB)
     |████████████████████████████████| 78 kB 4.6 MB/s
Requirement already satisfied: importlib-resources in /usr/local/lib/python3.7/dist-packages (from alembic->optuna) (5.6.0)
Collecting cmd2>=1.0.0
  Downloading cmd2-2.4.1-py3-none-any.whl (146 kB)
     |████████████████████████████████| 146 kB 10.4 MB/s
Collecting autopage>=0.4.0
  Downloading autopage-0.5.0-py3-none-any.whl (29 kB)
Collecting stevedore>=2.0.1
  Downloading stevedore-3.5.0-py3-none-any.whl (49 kB)
     |████████████████████████████████| 49 kB 2.1 MB/s
Requirement already satisfied: PrettyTable>=0.7.2 in /usr/local/lib/python3.7/dist-packages (from cliff->optuna) (3.2.0)
Collecting pbr!=2.1.0,>=2.0.0
  Downloading pbr-5.8.1-py2.py3-none-any.whl (113 kB)
     |████████████████████████████████| 113 kB 16.9 MB/s
Requirement already satisfied: attrs>=16.3.0 in /usr/local/lib/python3.7/dist-packages (from cmd2>=1.0.0->cliff->optuna) (21.4.0)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages (from cmd2>=1.0.0->cliff->optuna) (4.1.1)
Collecting pyperclip>=1.6
  Downloading pyperclip-1.8.2.tar.gz (20 kB)
Requirement already satisfied: wcwidth>=0.1.7 in /usr/local/lib/python3.7/dist-packages (from cmd2>=1.0.0->cliff->optuna) (0.2.5)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (from importlib-metadata->sqlalchemy>=1.1.0->optuna) (
Requirement already satisfied: MarkupSafe>=0.9.2 in /usr/local/lib/python3.7/dist-packages (from Mako->alembic->optuna) (2.0.1)
Building wheels for collected packages: pyperclip
  Building wheel for pyperclip (setup.py) ... done
  Created wheel for pyperclip: filename=pyperclip-1.8.2-py3-none-any.whl size=11137 sha256=d4c638fdf1297e1a4a3c6d2a90f9b6f6bb380435cf0de3
  Stored in directory: /root/.cache/pip/wheels/9f/18/84/8f69f8b08169c7bae2dde6bd7daf0c19fca8c8e500ee620a28
Successfully built pyperclip
Installing collected packages: pyperclip, pbr, stevedore, Mako, cmd2, autopage, colorlog, cmaes, cliff, alembic, optuna
Successfully installed Mako-1.2.0 alembic-1.7.7 autopage-0.5.0 cliff-3.10.1 cmaes-0.8.2 cmd2-2.4.1 colorlog-6.6.0 optuna-2.10.0 pbr-5.8.1
```

```
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 choose at least 3.
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: 0.96
```

```
### YOUR CODE HERE


"""


import numpy as np
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

import optuna

X, y = load_iris(return_X_y=True)
X_train, X_valid, y_train, y_valid = train_test_split(X, y)
```

```
def objective(trial):
    n_estimators = trial.suggest_int("n_estimators", 50, 400)
    clf = RandomForestClassifier(n_estimators=n_estimators, random_state=0)
    clf.fit(X_train, y_train)
    return clf.score(X_valid, y_valid)


study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=3)
```

```
>>> from sklearn import datasets, linear_model
>>> from sklearn.model_selection import cross_val_score
>>> diabetes = datasets.load_diabetes()
>>> X = diabetes.data[:150]
>>> y = diabetes.target[:150]
>>> lasso = linear_model.Lasso()
>>> print(cross_val_score(lasso, X, y, cv=3))
[0.33150734 0.08022311 0.03531764]
"""

    '\n\n\nimport numpy as np\nfrom sklearn.datasets import load_iris\nfrom sklearn.ensemble import RandomForestClassifier\nfrom sklearn.mod
    el_selection import train_test_split\n\nimport optuna\n\nX, y = load_iris(return_X_y=True)\nX_train, X_valid, y_train, y_valid = train_t
    est_split(X, y)\n\n\ndef objective(trial):\n    n_estimators = trial.suggest_int("n_estimators", 50, 400)\n    clf = RandomForestClassif
    ier(n_estimators=n_estimators, random_state=0)\n    clf.fit(X_train, y_train)\n    return clf.score(X_valid, y_valid)\n\n\nstudy = optun
    a.create_study(direction="maximize")\nstudy.optimize(objective, n_trials=3)\n\n\n\n\n>>> from sklearn import datasets, linear_model\n>>>
    from sklearn.model_selection import cross_val_score\n>>> diabetes = datasets.load_diabetes()\n>>> X = diabetes.data[:150]\n>>> y = diabe
```

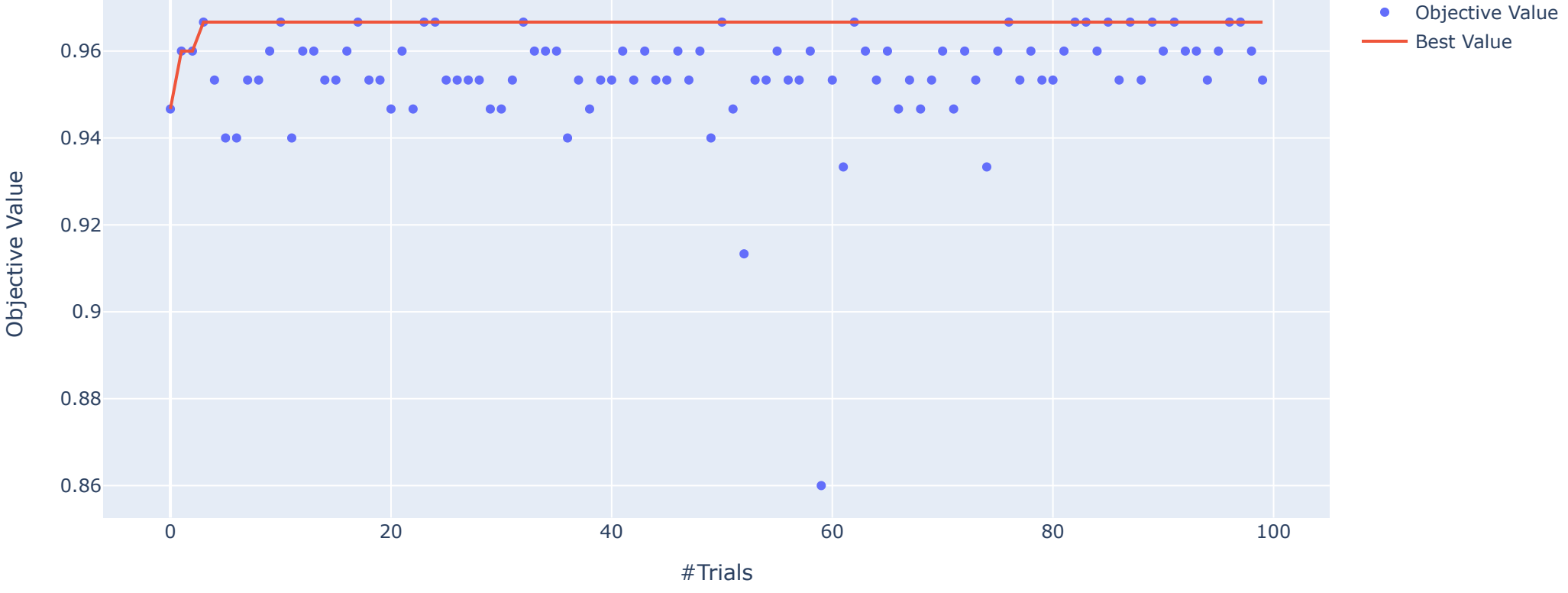
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
def f(trial):
    n_estimators = trial.suggest_int("n_estimators", 50, 400)
    max_depth = int(trial.suggest_float('max_depth', 1, 32, log=True))
    min_samples_leaf = trial.suggest_int("min_samples_leaf", 1, 5)
    clf = sklearn.ensemble.RandomForestClassifier(n_estimators=n_estimators, max_depth = max_depth, min_samples_leaf=min_samples_leaf)
    ret = sklearn.model_selection.cross_val_score(clf, x, y, cv=3)
    return ret.mean()

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

trial = study.best_trial

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

```
[I 2022-04-16 11:56:45,848] Trial 44 finished with value: 0.9533333333333333 and parameters: {'n_estimators': 273, 'max_depth': 1.11393
[I 2022-04-16 11:56:46,934] Trial 45 finished with value: 0.9533333333333333 and parameters: {'n_estimators': 258, 'max_depth': 1.73214
[I 2022-04-16 11:56:48,353] Trial 46 finished with value: 0.96 and parameters: {'n_estimators': 340, 'max_depth': 1.204964394268541, 'm
[I 2022-04-16 11:56:49,133] Trial 47 finished with value: 0.9666666666666667 and parameters: {'n_estimators': 185, 'max_depth': 1.60871
[I 2022-04-16 11:56:49,922] Trial 48 finished with value: 0.9466666666666667 and parameters: {'n_estimators': 189, 'max_depth': 2.57223
[I 2022-04-16 11:56:50,669] Trial 49 finished with value: 0.9533333333333333 and parameters: {'n_estimators': 178, 'max_depth': 1.46268
[I 2022-04-16 11:56:51,275] Trial 50 finished with value: 0.9533333333333333 and parameters: {'n_estimators': 144, 'max_depth': 1.64756
[I 2022-04-16 11:56:52,697] Trial 51 finished with value: 0.96 and parameters: {'n_estimators': 339, 'max_depth': 1.2315820261479775,
[I 2022-04-16 11:56:54,306] Trial 52 finished with value: 0.96 and parameters: {'n_estimators': 384, 'max_depth': 1.2280354298809544,
[I 2022-04-16 11:56:54,785] Trial 53 finished with value: 0.9466666666666667 and parameters: {'n_estimators': 108, 'max_depth': 1.16286
[I 2022-04-16 11:56:55,681] Trial 54 finished with value: 0.94 and parameters: {'n_estimators': 215, 'max_depth': 1.4636356498258862,
[I 2022-04-16 11:56:57,273] Trial 55 finished with value: 0.96 and parameters: {'n_estimators': 381, 'max_depth': 1.0233558210604057,
[I 2022-04-16 11:56:58,136] Trial 56 finished with value: 0.9533333333333333 and parameters: {'n_estimators': 199, 'max_depth': 2.74926
[I 2022-04-16 11:56:59,179] Trial 57 finished with value: 0.9666666666666667 and parameters: {'n_estimators': 246, 'max_depth': 1.79675
[I 2022-04-16 11:57:00,233] Trial 58 finished with value: 0.9533333333333333 and parameters: {'n_estimators': 245, 'max_depth': 2.20773
[I 2022-04-16 11:57:00,579] Trial 59 finished with value: 0.8799999999999999 and parameters: {'n_estimators': 79, 'max_depth': 1.837326
[I 2022-04-16 11:57:01,559] Trial 60 finished with value: 0.9666666666666667 and parameters: {'n_estimators': 227, 'max_depth': 1.58115
[I 2022-04-16 11:57:02,689] Trial 61 finished with value: 0.9533333333333333 and parameters: {'n_estimators': 268, 'max_depth': 1.54655
[I 2022-04-16 11:57:03,638] Trial 62 finished with value: 0.9666666666666667 and parameters: {'n_estimators': 223, 'max_depth': 1.41358
[I 2022-04-16 11:57:04,680] Trial 63 finished with value: 0.9533333333333333 and parameters: {'n_estimators': 249, 'max_depth': 1.76295
[I 2022-04-16 11:57:05,662] Trial 64 finished with value: 0.9466666666666667 and parameters: {'n_estimators': 234, 'max_depth': 1.42616
[I 2022-04-16 11:57:06,408] Trial 65 finished with value: 0.9466666666666667 and parameters: {'n_estimators': 174, 'max_depth': 2.06846
[I 2022-04-16 11:57:07,456] Trial 66 finished with value: 0.9466666666666667 and parameters: {'n_estimators': 223, 'max_depth': 1.35282
[I 2022-04-16 11:57:08,410] Trial 67 finished with value: 0.9466666666666667 and parameters: {'n_estimators': 215, 'max_depth': 2.40934
[I 2022-04-16 11:57:09,482] Trial 68 finished with value: 0.9666666666666667 and parameters: {'n_estimators': 254, 'max_depth': 1.32946
[I 2022-04-16 11:57:10,326] Trial 69 finished with value: 0.96 and parameters: {'n_estimators': 200, 'max_depth': 4.174243674229171, 'm
[I 2022-04-16 11:57:11,000] Trial 70 finished with value: 0.96 and parameters: {'n_estimators': 158, 'max_depth': 7.041800172397913, 'm
[I 2022-04-16 11:57:12,135] Trial 71 finished with value: 0.9666666666666667 and parameters: {'n_estimators': 263, 'max_depth': 1.11632
[I 2022-04-16 11:57:13,188] Trial 72 finished with value: 0.9533333333333333 and parameters: {'n_estimators': 251, 'max_depth': 1.35824
[I 2022-04-16 11:57:14,334] Trial 73 finished with value: 0.9466666666666667 and parameters: {'n_estimators': 273, 'max_depth': 1.10426
[I 2022-04-16 11:57:15,298] Trial 74 finished with value: 0.9533333333333333 and parameters: {'n_estimators': 231, 'max_depth': 1.00114
[I 2022-04-16 11:57:17,310] Trial 75 finished with value: 0.96 and parameters: {'n_estimators': 294, 'max_depth': 1.5726622649929014,
[I 2022-04-16 11:57:18,494] Trial 76 finished with value: 0.96 and parameters: {'n_estimators': 281, 'max_depth': 6.129877647971909, 'm
[I 2022-04-16 11:57:19,521] Trial 77 finished with value: 0.94 and parameters: {'n_estimators': 241, 'max_depth': 1.8735276787729414,
[I 2022-04-16 11:57:20,642] Trial 78 finished with value: 0.96 and parameters: {'n_estimators': 267, 'max_depth': 23.875783076843607,
[I 2022-04-16 11:57:21,200] Trial 79 finished with value: 0.9466666666666667 and parameters: {'n_estimators': 134, 'max_depth': 1.10741
[I 2022-04-16 11:57:22,101] Trial 80 finished with value: 0.9466666666666667 and parameters: {'n_estimators': 212, 'max_depth': 1.31556
[I 2022-04-16 11:57:23,071] Trial 81 finished with value: 0.9533333333333333 and parameters: {'n_estimators': 228, 'max_depth': 1.64469
```

```
[I 2022-04-16 11:57:24,154] Trial 82 finished with value: 0.9533333333333333 and parameters: {'n_estimators': 251, 'max_depth': 9.19778
[I 2022-04-16 11:57:25,254] Trial 83 finished with value: 0.96 and parameters: {'n_estimators': 261, 'max_depth': 1.5140017714064349,
[I 2022-04-16 11:57:26,679] Trial 84 finished with value: 0.9466666666666667 and parameters: {'n_estimators': 317, 'max_depth': 2.08744
[I 2022-04-16 11:57:28,620] Trial 85 finished with value: 0.9466666666666667 and parameters: {'n_estimators': 257, 'max_depth': 1.25443
[I 2022-04-16 11:57:31,283] Trial 86 finished with value: 0.9533333333333333 and parameters: {'n_estimators': 287, 'max_depth': 1.76473
[I 2022-04-16 11:57:33,303] Trial 87 finished with value: 0.9533333333333333 and parameters: {'n_estimators': 239, 'max_depth': 1.14466
[I 2022-04-16 11:57:35,524] Trial 88 finished with value: 0.9533333333333333 and parameters: {'n_estimators': 276, 'max_depth': 1.36761
[I 2022-04-16 11:57:37,400] Trial 89 finished with value: 0.96 and parameters: {'n_estimators': 185, 'max_depth': 12.723386729564742,
[I 2022-04-16 11:57:38,343] Trial 90 finished with value: 0.96 and parameters: {'n_estimators': 222, 'max_depth': 5.345529329968613, 'm
[I 2022-04-16 11:57:39,449] Trial 91 finished with value: 0.9666666666666667 and parameters: {'n_estimators': 262, 'max_depth': 1.64721
[I 2022-04-16 11:57:40,146] Trial 92 finished with value: 0.9666666666666667 and parameters: {'n_estimators': 166, 'max_depth': 1.26785
[I 2022-04-16 11:57:41,243] Trial 93 finished with value: 0.9666666666666667 and parameters: {'n_estimators': 266, 'max_depth': 1.64154
[I 2022-04-16 11:57:41,950] Trial 94 finished with value: 0.9466666666666667 and parameters: {'n_estimators': 164, 'max_depth': 1.48377
[I 2022-04-16 11:57:42,450] Trial 95 finished with value: 0.94 and parameters: {'n_estimators': 115, 'max_depth': 1.698142055308438, 'm
[I 2022-04-16 11:57:43,578] Trial 96 finished with value: 0.9533333333333333 and parameters: {'n_estimators': 263, 'max_depth': 7.85414
[I 2022-04-16 11:57:44,450] Trial 97 finished with value: 0.9666666666666667 and parameters: {'n_estimators': 207, 'max_depth': 1.11919
[I 2022-04-16 11:57:45,669] Trial 98 finished with value: 0.9666666666666667 and parameters: {'n_estimators': 291, 'max_depth': 1.88773
[I 2022-04-16 11:57:46,485] Trial 99 finished with value: 0.9666666666666667 and parameters: {'n_estimators': 195, 'max_depth': 1.07626
Accuracy: 0.9666666666666667
```

```
optuna.visualization.plot_optimization_history(study)
```

Optimization History Plot



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: Жители одномерной деревни хотят подключиться к интернету, поэтому им нужна центральная станция обслуживания, от которой протянется кабель ко всем домам деревни. Пусть цена кабеля, который нужно протянуть от станции к каждому дому независимо, определяется некоторой функцией p(d). Тогда понятно, что за доступ во всемирную паутину деревне придется заплатить следующую сумму:

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

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

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

$$\nabla P(w*,x) = \sum_{i=1}^{N} 2(w* - x_i) = 0 \Rightarrow \frac{\sum_{i=1}^{N} x_i}{N} = w*$$

```
print(sum(x)/len(x))
```

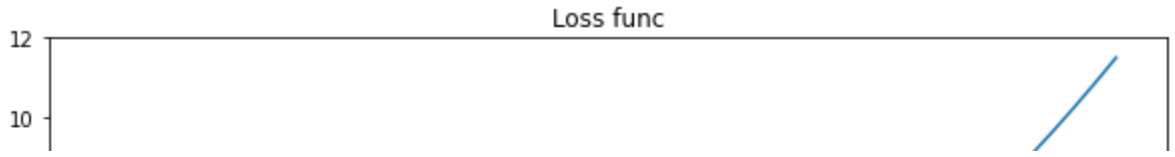
```
0.36488443358603545
```

Write loss function $P(x,w)$

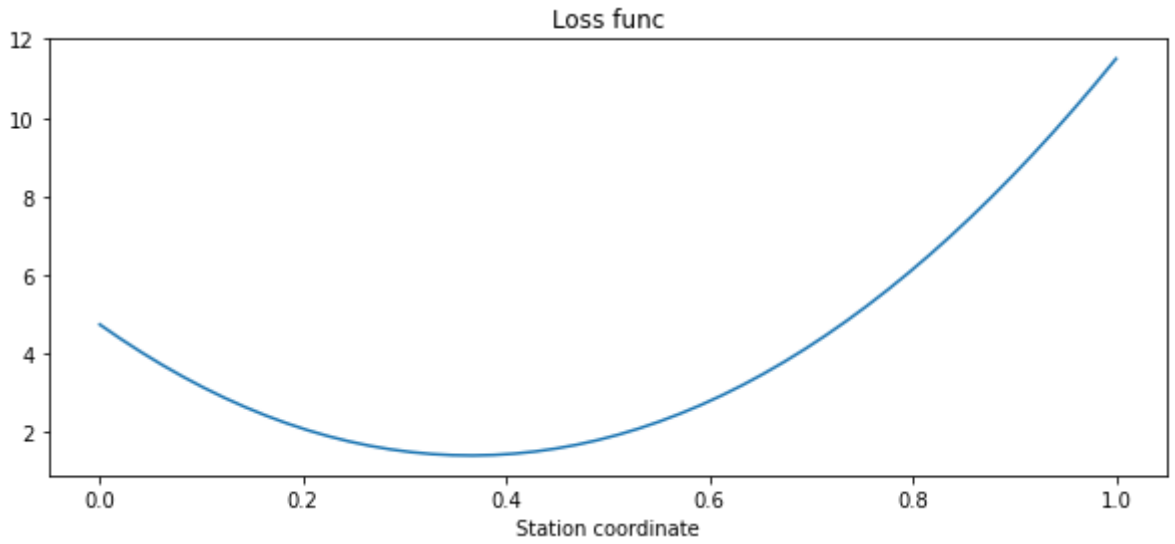
```
def P(w, x):
    P = sum([(w - coord)**2 for coord in x])
    return P
```

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

```
#print(sum(x)/len(x))
N = 25
l = 1

w = np.linspace(0,l)
p = [P(w_i, x) for w_i in w]
plt.title('Loss func')
plt.xlabel('Station coordinate')
plt.plot(w, p)
fig = plt.gcf()
fig.set_size_inches(10,4)
#print(P(0.5, x))
```



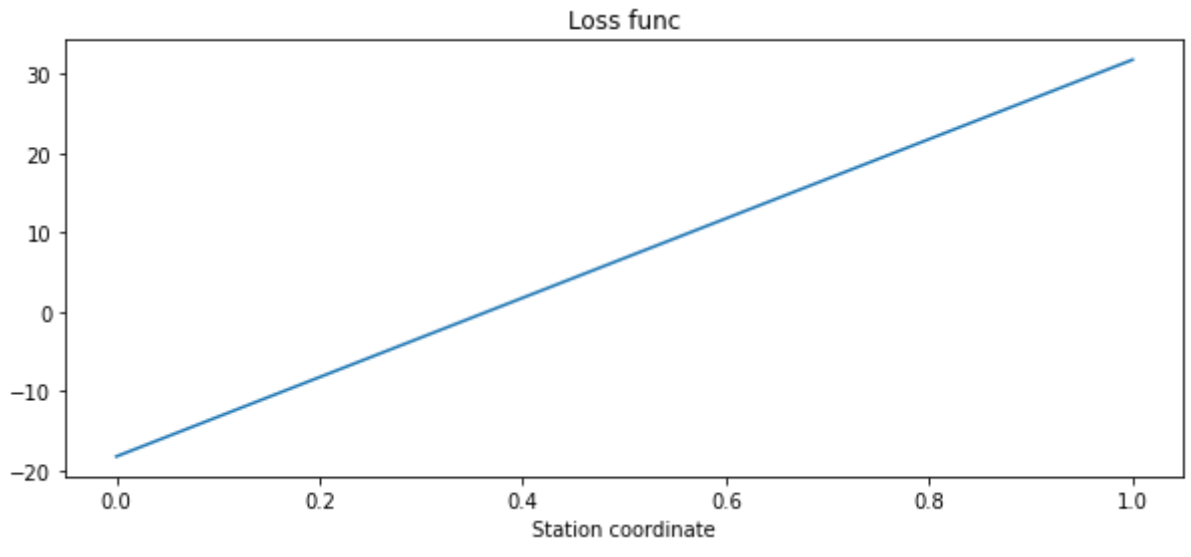

```
#print(sum(x)/len(x))
N = 25
l = 1

w = np.linspace(0,l)
p = [P(w_i, x) for w_i in w]
plt.title('Loss func')
plt.xlabel('Station coordinate')
plt.plot(w, p)
fig = plt.gcf()
fig.set_size_inches(10,4)
#print(P(0.5, x))
```




```
#def P(w, x):
 #   P = sum([(w - coord)**2 for coord in x])
  #  return P

def dP(w, x):
    dP = 2*sum([(w - coord) for coord in x])
    return dP
```

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

```
w = np.linspace(0,l)
p = [dP(point, x) for point in w]
plt.title('Loss func')
plt.xlabel('Station coordinate')
plt.plot(w, p)
fig = plt.gcf()
fig.set_size_inches(10,4)
```



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

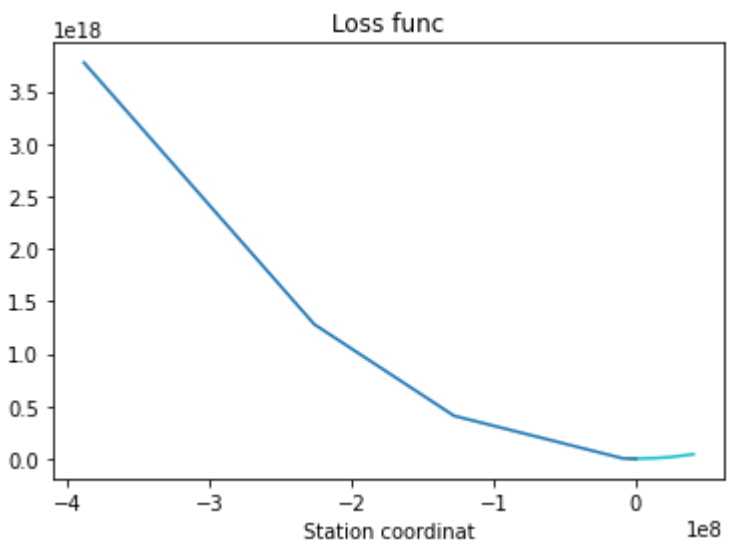
$$w_{k+1} = w_k - \mu \nabla P(w_k)$$

```
from numpy.ma.core import append
def gradient_descent(P, dP, w0, mu, Nsteps):
    init = w0
```

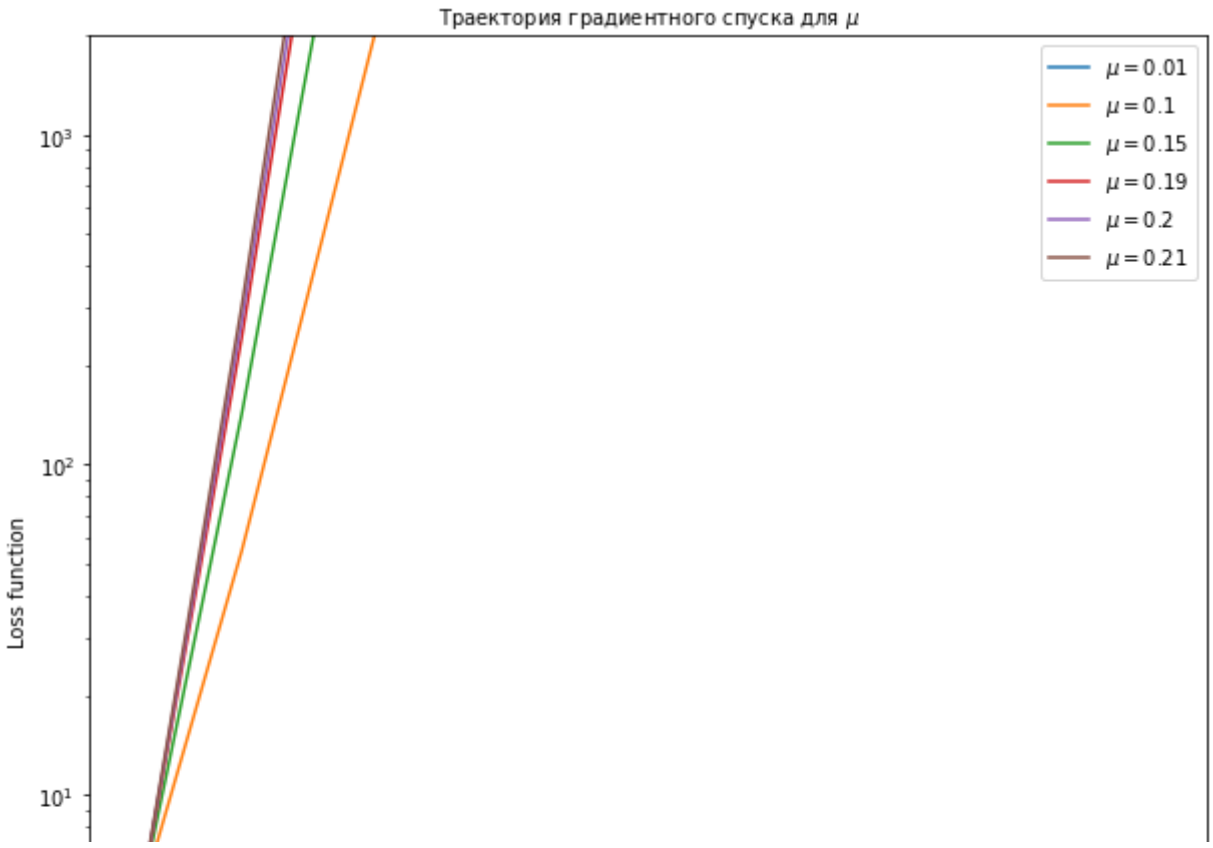
```
    #path = []
    for k in range(Nsteps):
      #path.append(init)
      init -=mu*dP(init, x)
      print(init)
    #path.append(init)
    return init
```

Modify `gradient_descent` to return all optimization trajectory. Plot loss function trajectory for the following learning rates $
Draw conclusions.

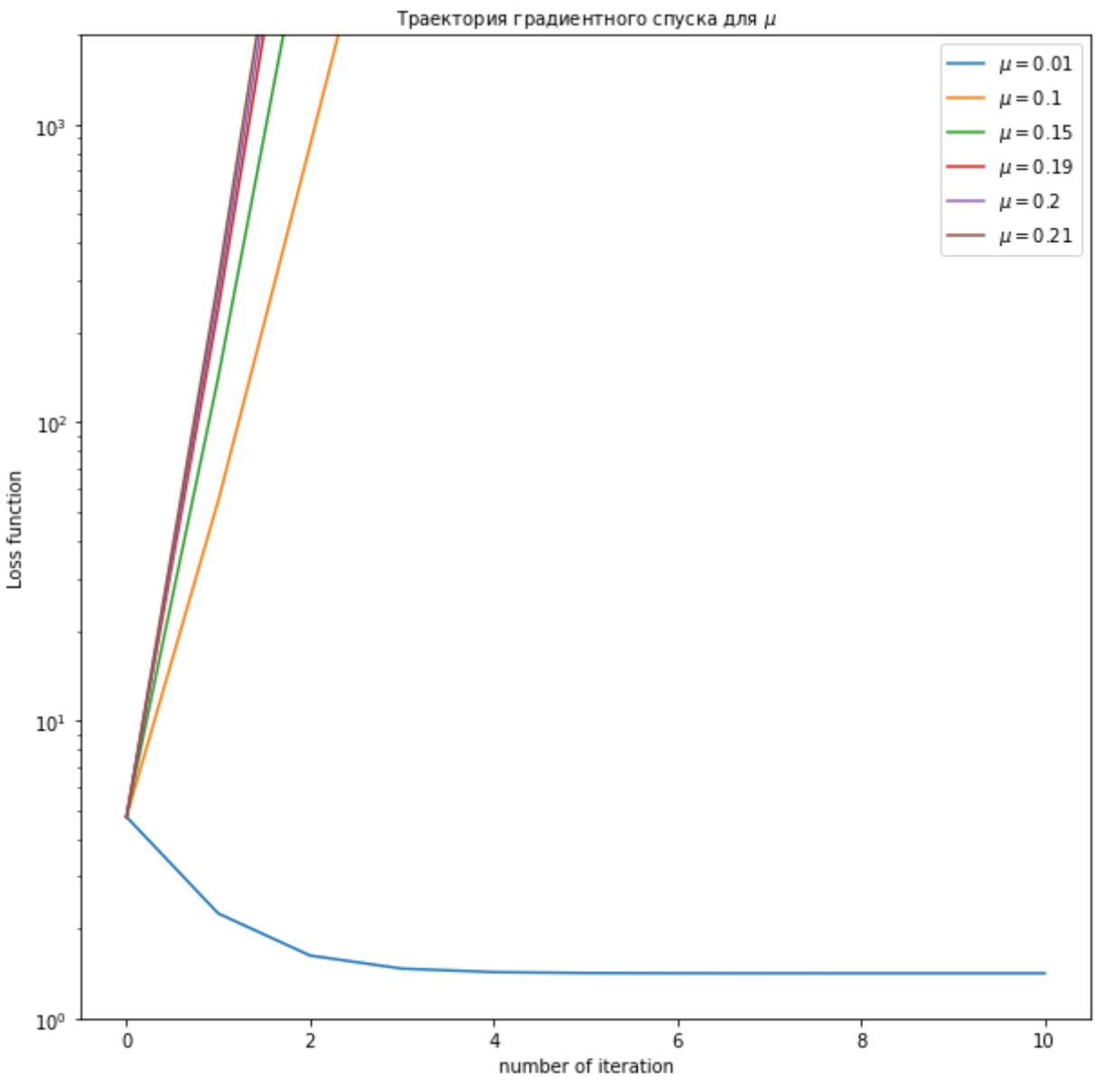
```
def gradient_descent(P, dP, w0, mu, Nsteps):
    init = w0
    path = []
    for k in range(Nsteps):
      path.append(init)
      init -=mu*dP(init, x)
    path.append(init)
    return path


Nsteps = 10
w0 = 0.3
#print(dP(w0,x))
#print(gradient_descent(P, dP, w0, 0.1, Nsteps))
path = [0]*6
i = 0
for mu in [0.01, 0.1, 0.15, 0.19, 0.20, 0.21]:

  path[i] = gradient_descent(P, dP, w0, mu, Nsteps)
  i +=1
plt.plot(path, P(path,x))
plt.title("Loss func")
plt.xlabel("Station coordinat")
plt.show()
```




```
mus = [0.01, 0.1, 0.15, 0.19, 0.20, 0.21]
w0 = 0 # initial guess
Nsteps = 10
plt.figure(figsize=(10,10))
plt.title('Траектория градиентного спуска для $\mu$', fontsize=10)
plt.xlabel('number of iteration', fontsize=10)
plt.ylabel('Loss function', fontsize=10)
plt.ylim(1, 2000)
plt.yscale('log')
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
for mu in mus:
 ws = gradient_descent(P, dP, w0, mu, Nsteps)
 Ps = [P(w, x) for w in ws]
 plt.plot(range(len(ws)), Ps, label=('$\mu = $' + str(mu)))
plt.legend(fontsize=10)
plt.show()
```




```
mus = [0.01, 0.1, 0.15, 0.19, 0.20, 0.21]
w0 = 0 # initial guess
Nsteps = 10
plt.figure(figsize=(10,10))
plt.title('Траектория градиентного спуска для $\mu$', fontsize=10)
plt.xlabel('number of iteration', fontsize=10)
plt.ylabel('Loss function', fontsize=10)
plt.ylim(1, 2000)
plt.yscale('log')
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
for mu in mus:
 ws = gradient_descent(P, dP, w0, mu, Nsteps)
 Ps = [P(w, x) for w in ws]
 plt.plot(range(len(ws)), Ps, label=('$\mu = $' + str(mu)))
plt.legend(fontsize=10)
plt.show()
```



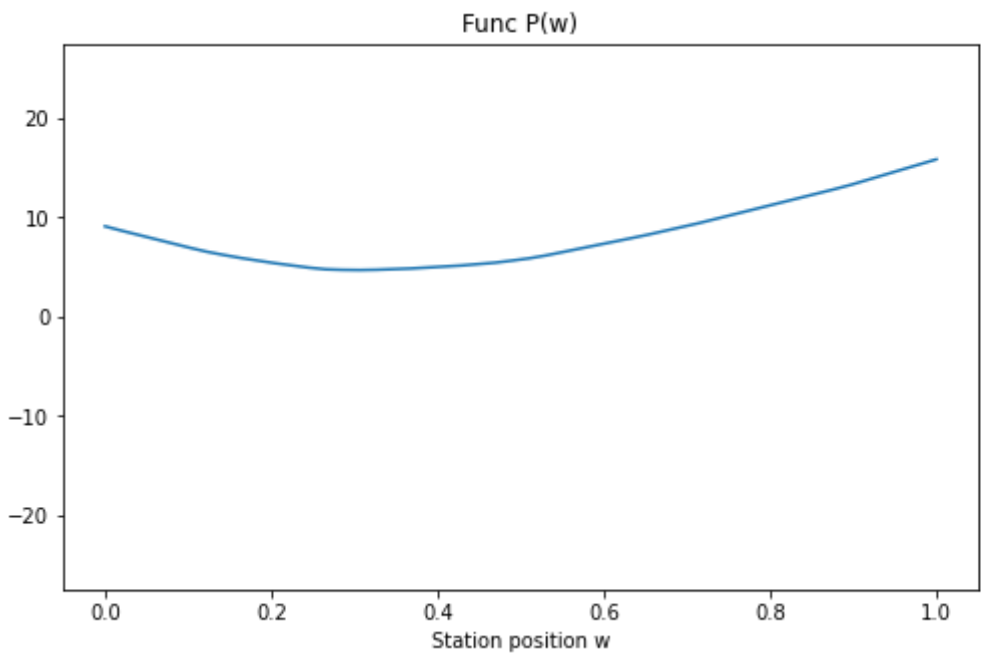
Из графиков видно, что минимальный шаг, начиная с которого градиентный спуск расходится, больше или равен 0.1

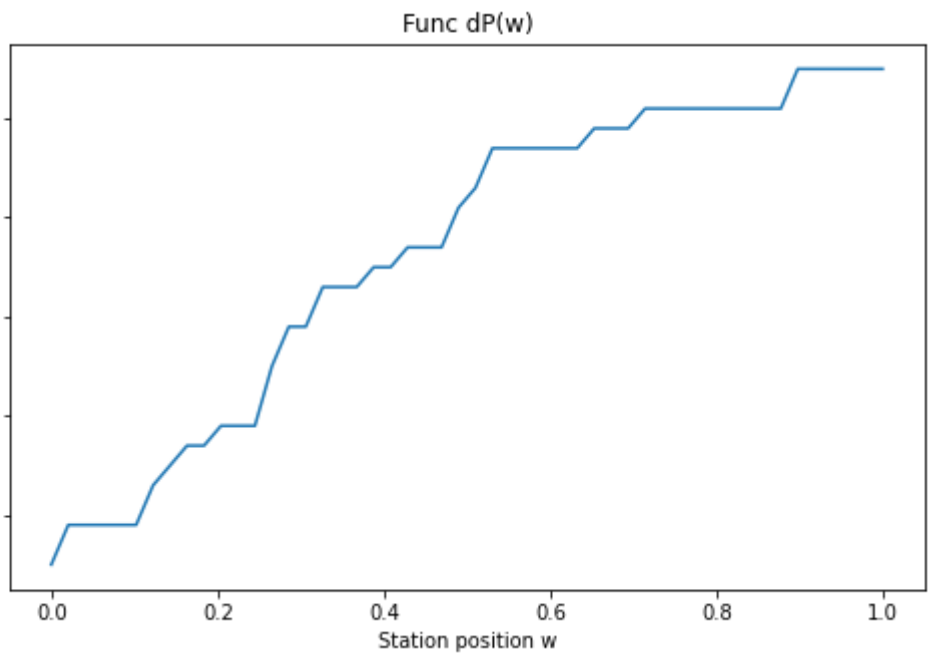
The village decided to lay cable using new technology. That's why the price of the cable changed to:

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

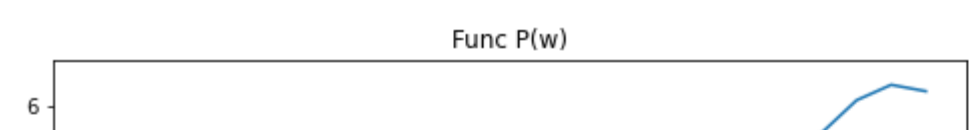
Write new function `P`, `dP`. Plot graphs for various x and w.

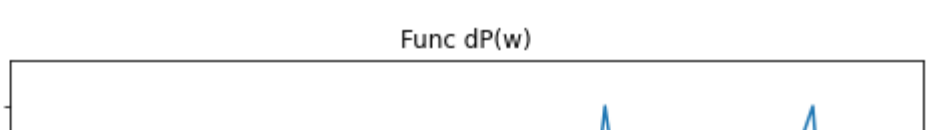
```
import math
def P(w, x):
      P = sum([abs((w - coord)) for coord in x])
      return P

#def dP(w, x):
 #   dP = 2*sum([(w - coord) for coord in x])
  #  return dP

def dP(w, x):
    dP = sum([math.copysign(1,w-coord) for coord in x])
    return dP
#ddP = 0



w = np.linspace(0,1)
p = [P(coord, x) for coord in w]
dp = [dP(coord, x) for coord in w]
funcan, (g1, g2) = plt.subplots(1,2, sharey = True)
g1.plot(w,p)
g1.set_title("Func P(w)")
g1.set_xlabel("Station position w")
g2.plot(w, dp)
g2.set_title("Func dP(w)")
g2.set_xlabel("Station position w")
funcan.set_size_inches(18,5)
plt.show()
```



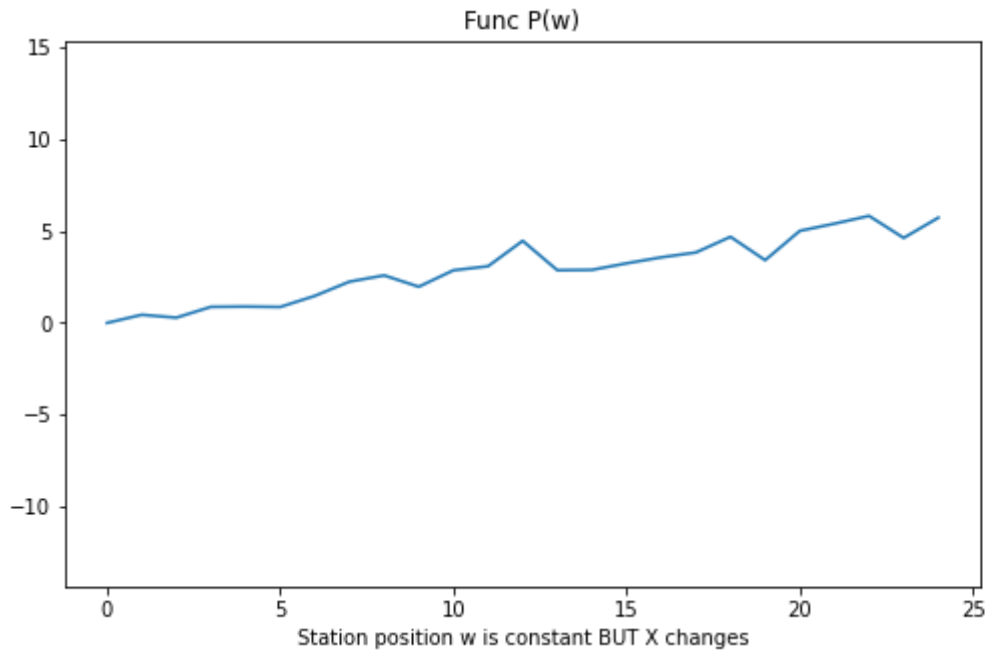


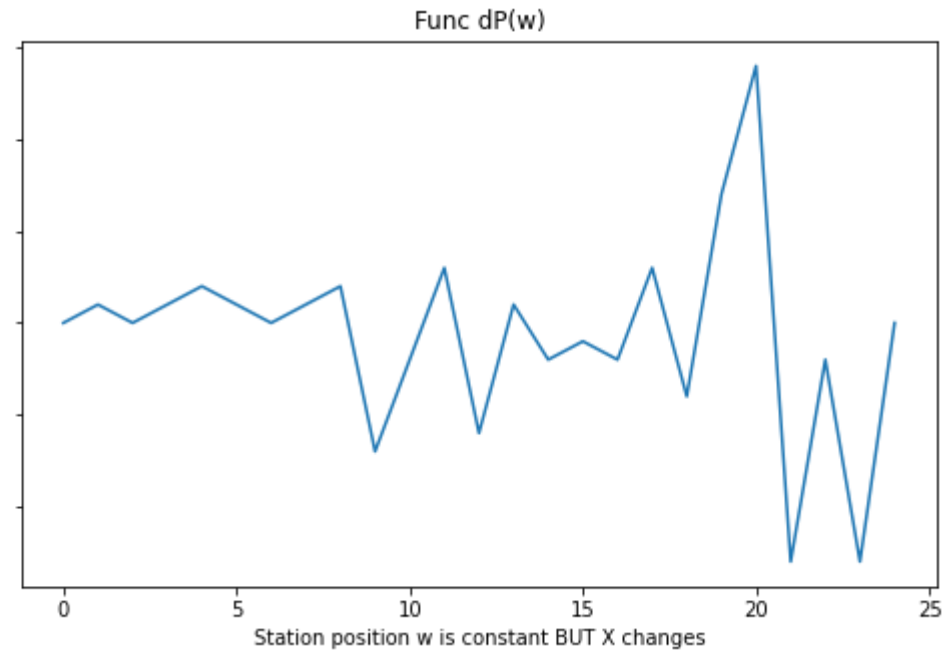

```
#ФИКСИРУЕМ W И МЕНЯЕМ X
w = 0.5
arr_x = ([i for i in range(N)])
arr_y = ([np.random.rand(i)*1 for i in range(N)])
for i in range(N):
  #x = np.random.rand(i)*1
  p = [P(w, arr_y[i]) for i in range(N)]
  dp = [dP(w, arr_y[i]) for i in range(N)]
#print(a)
funcan, (g1, g2) = plt.subplots(1,2, sharey = True)
g1.plot(arr_x,p)
g1.set_title("Func P(w)")
g1.set_xlabel("Station position w is constant BUT X changes")
g2.plot(arr_x, dp)
g2.set_title("Func dP(w)")
g2.set_xlabel("Station position w is constant BUT X changes")
funcan.set_size_inches(18,5)
plt.show()
```






```
#ФИКСИРУЕМ W И МЕНЯЕМ X
w = 0.5
arr_x = ([i for i in range(N)])
arr_y = ([np.random.rand(i)*1 for i in range(N)])
for i in range(N):
  #x = np.random.rand(i)*1
  p = [P(w, arr_y[i]) for i in range(N)]
  dp = [dP(w, arr_y[i]) for i in range(N)]
#print(a)
funcan, (g1, g2) = plt.subplots(1,2, sharey = True)
g1.plot(arr_x,p)
g1.set_title("Func P(w)")
g1.set_xlabel("Station position w is constant BUT X changes")
g2.plot(arr_x, dp)
g2.set_title("Func dP(w)")
g2.set_xlabel("Station position w is constant BUT X changes")
funcan.set_size_inches(18,5)
plt.show()
```





Write new analytical solytion w^*

$$\nabla P(w*, x) = \sum_{i=1}^{N} sign(w * -x_i) = 0 \Rightarrow w*$$

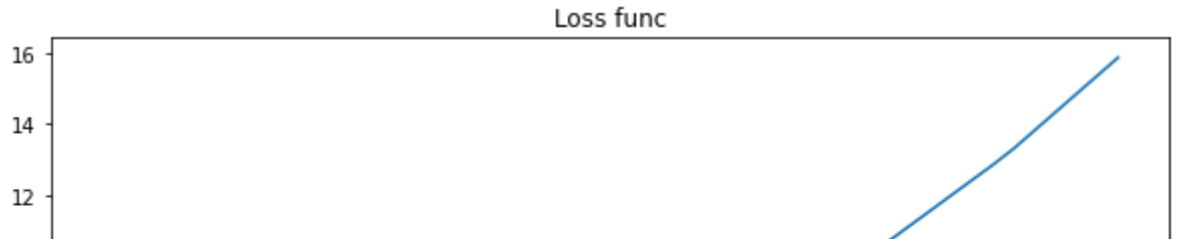
медиана нашего вектора x

```
print(np.median(x))
```

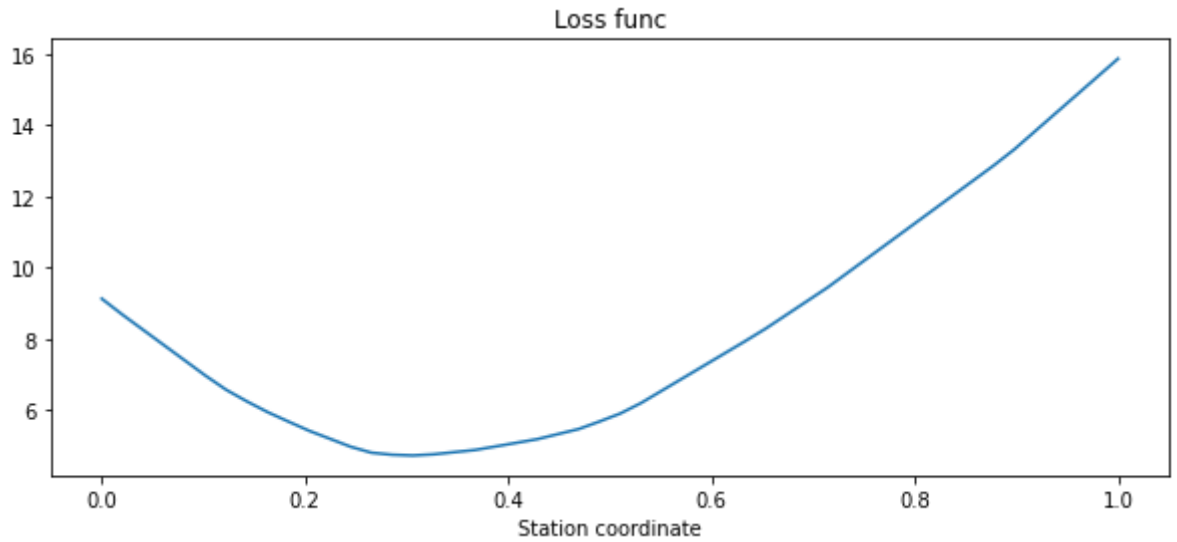
```
0.3064325193333567
```

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

```
#print(sum(x)/len(x))
N = 25
l = 1
#x = np.random.rand(N)*l
#print(P(0.4, x))


w = np.linspace(0,l)
p = [P(w_i, x) for w_i in w]
plt.title('Loss func')
plt.xlabel('Station coordinate')
plt.plot(w, p)
fig = plt.gcf()
fig.set_size_inches(10,4)
#print(P(0.5, x))
```




```
#print(sum(x)/len(x))
N = 25
l = 1
#x = np.random.rand(N)*l
#print(P(0.4, x))


w = np.linspace(0,l)
p = [P(w_i, x) for w_i in w]
plt.title('Loss func')
plt.xlabel('Station coordinate')
plt.plot(w, p)
fig = plt.gcf()
fig.set_size_inches(10,4)
#print(P(0.5, x))
```



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:

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

Write new `P`, `dP`.

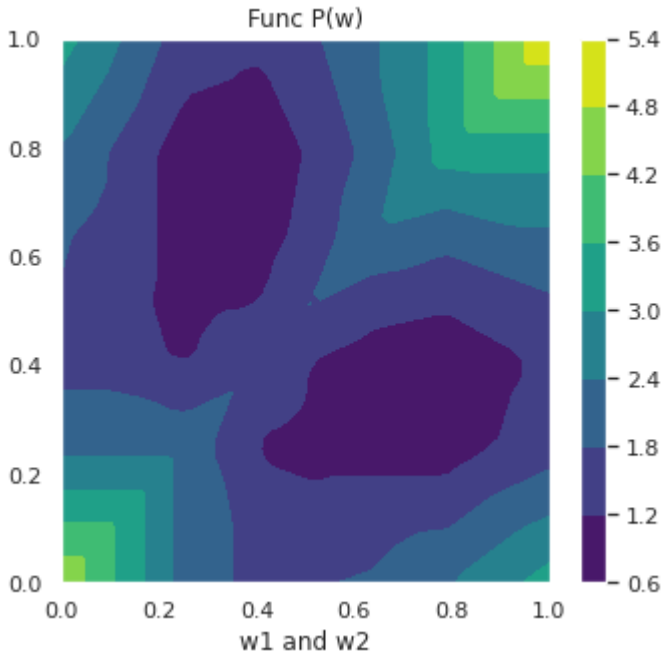
```
def P(w1, w2, x):
    P = sum([min(abs(w1 - coord),abs(w2 - coord)) for coord in x])
    return P

def dP(w1, w2, x):
    """dP = 0
    for coord in range(x):
      if (w1 - coord >= w2 - coord):
        dP +=math.copysign(1,w2-coord)

      if (w1 - coord < w2 - coord):
        dP +=math.copysign(1,w1-coord)
        """
    sum1 = sum([np.copysign(1,w1 - coord)*(abs(w1 - coord) <= abs(w2 - coord)) for coord in x])
    sum2 = sum([np.copysign(1,w2 - coord)*(abs(w1 - coord) > abs(w2 - coord)) for coord in x])
    dP = np.array([sum1, sum2])
    return dP
```

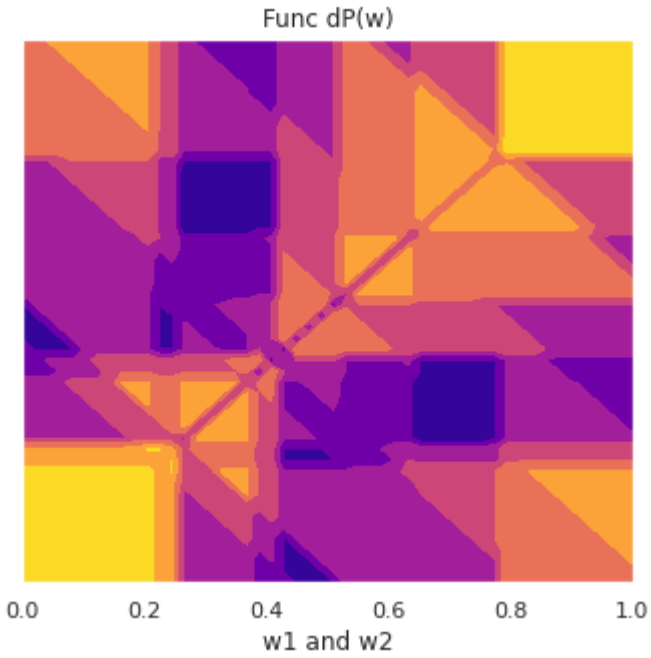
Plot $P(w_1, w_2), \nabla P(w_1, w_2)$ for different number of houses N. Comment on what happens as you increase N.

```
N = 100
l = 1
x = np.random.rand(N)*l
#print(P(0.4, x))


w1 = np.linspace(0,l)
w2 = np.linspace(0,l)
p_arr = np.zeros([w1.shape[0],w1.shape[0]])
```

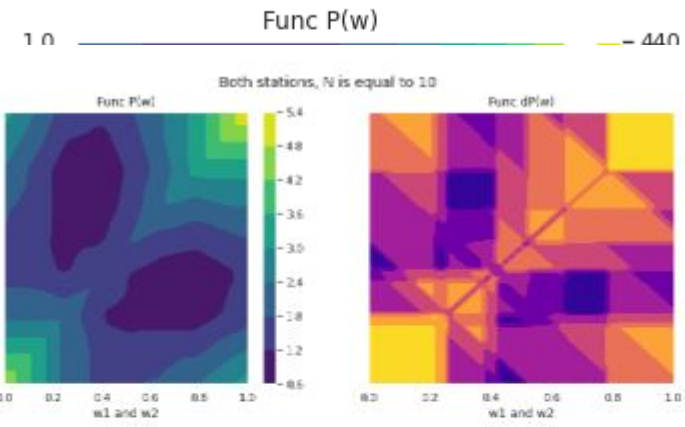
```
p_arr = np.zeros([w1.shape[0],w1.shape[0]])
dp_arr = np.zeros([w1.shape[0],w1.shape[0]])

for N in np.arange(10, 3210, 800):
  l = 1
  x = np.random.rand(N)*l
  i = 0
  for coord1 in w1:
    j = 0
    for coord2 in w2:
      p_arr[i][j] = P(coord1, coord2,x)
      dp_arr[i][j] = np.linalg.norm(dP(coord1,coord2, x))
      j +=1
    i += 1
  funcan, (g1, g2) = plt.subplots(1,2, sharey = True)
  val1 = g1.contourf(w1,w2, p_arr, cmap = "viridis")
  plt.colorbar(val1, ax = g1)
  g1.set_title("Func P(w)")
  g1.set_xlabel("w1 and w2")
  val1 = g2.contourf(w1,w2, dp_arr, cmap = "plasma")
  g2.set_title("Func dP(w)")
  g2.set_xlabel("w1 and w2")
  funcan.set_size_inches(12,5)
  funcan.suptitle("Both stations, N is equal to " + str(N))
  plt.show()
```

Both stations, N is equal to 10

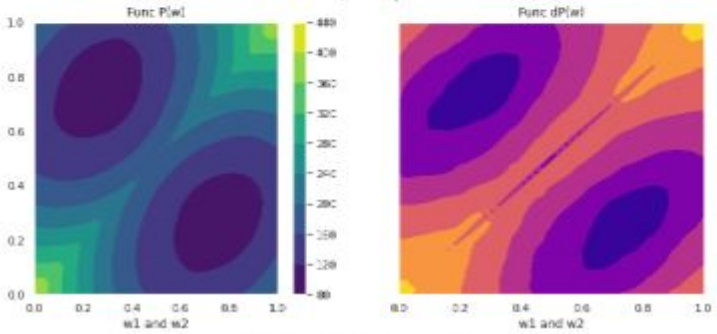




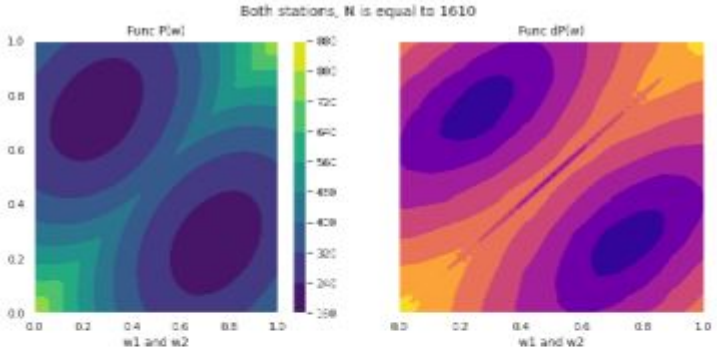
Both stations, N is equal to 810



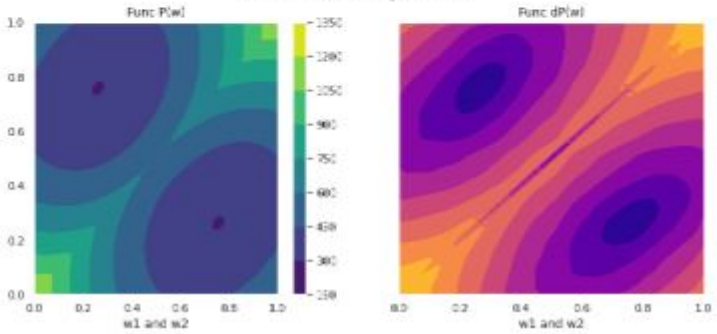
Both stations, N is equal to 810



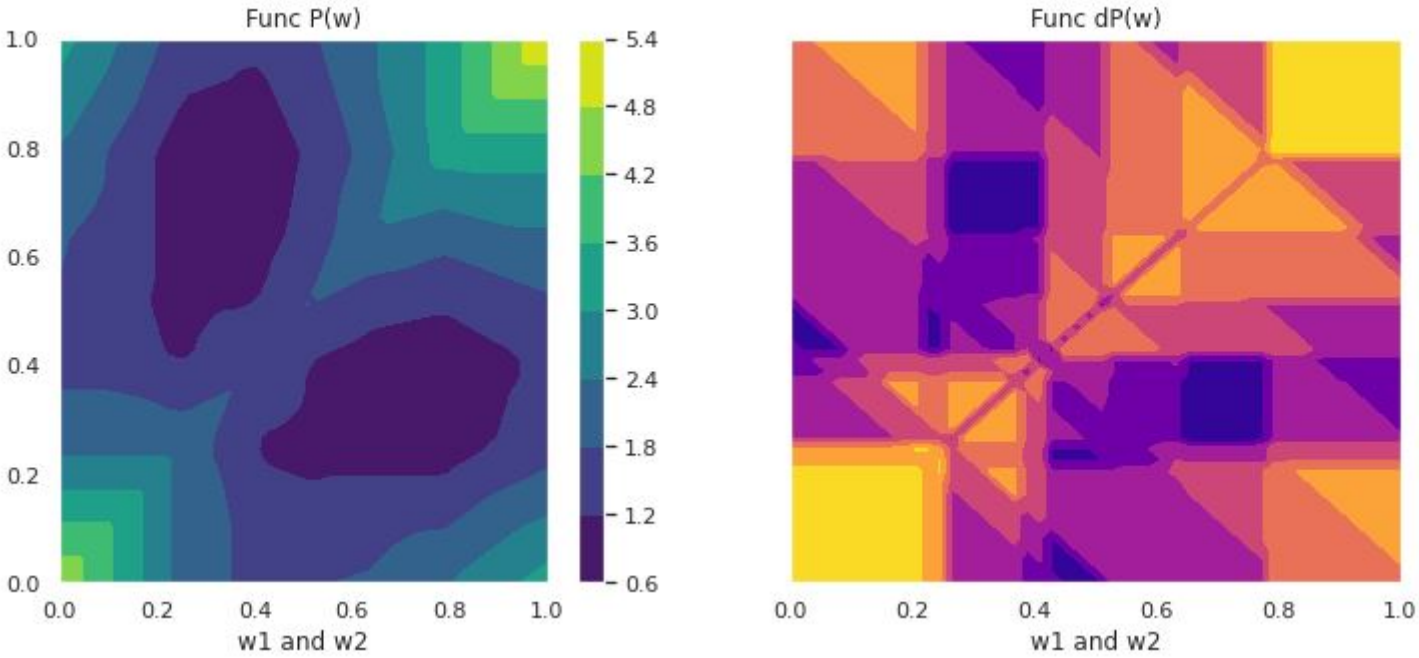
Both stations, N is equal to 1610



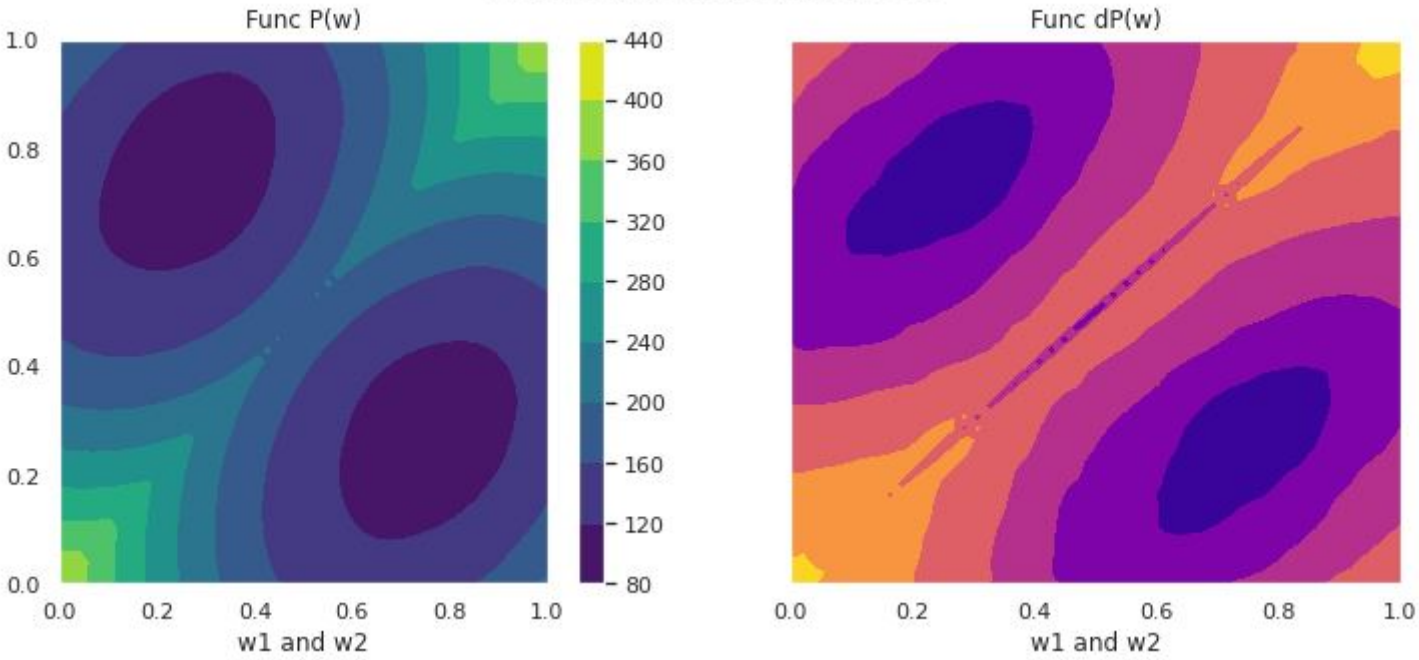
Both stations, N is equal to 2410

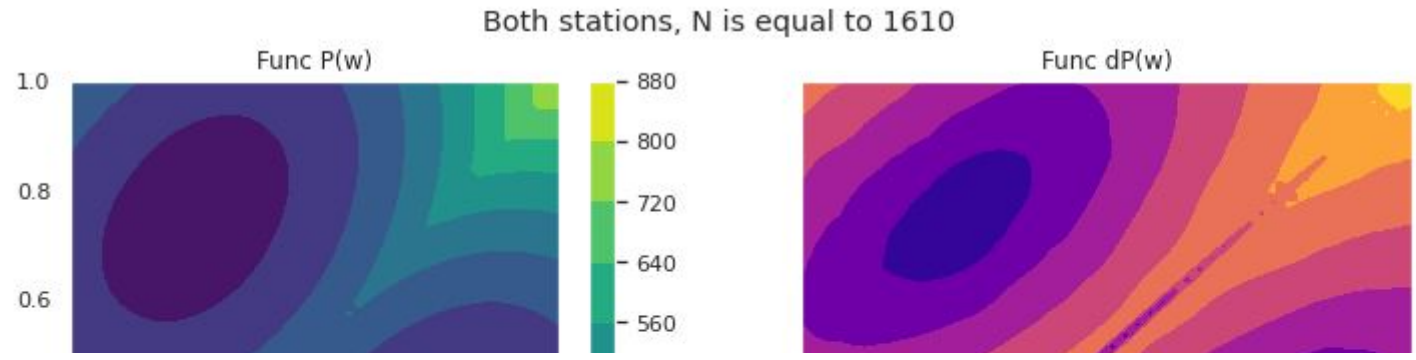


Both stations, N is equal to 10



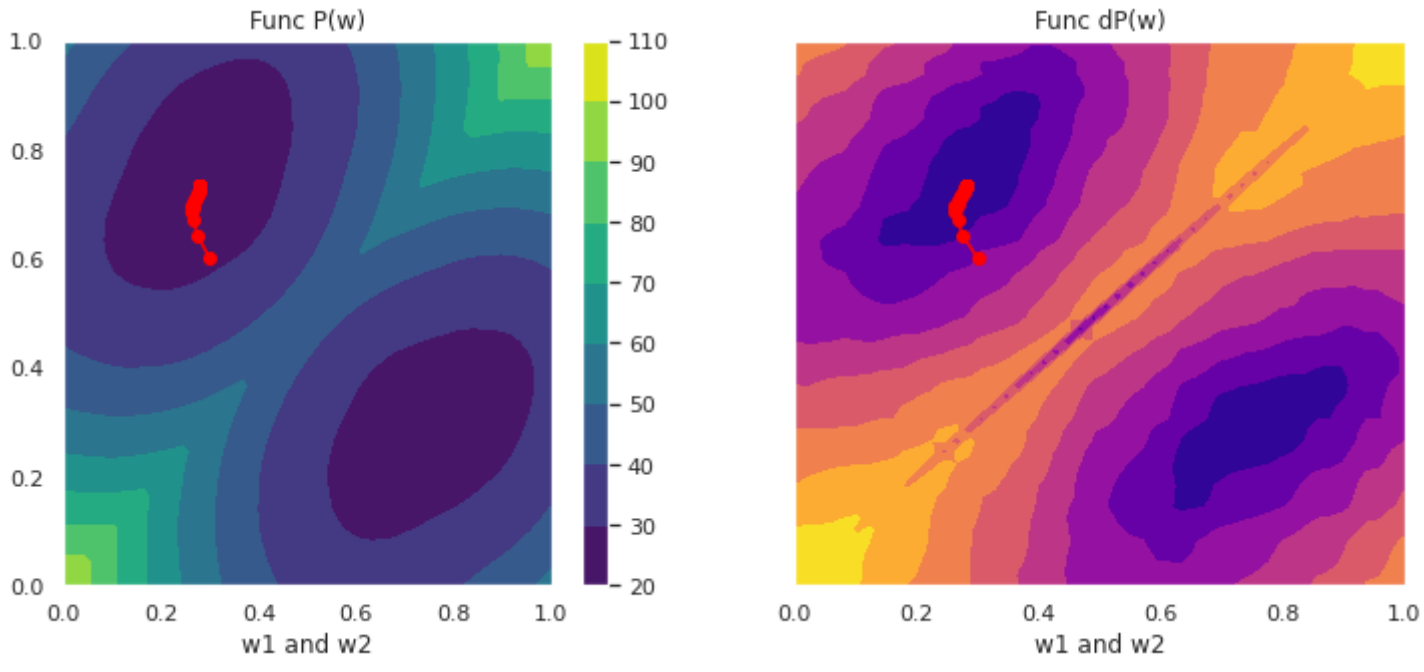
Both stations, N is equal to 810





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 gradient_descent(P, dP, w0, mu, Nsteps):
  prev = np.array(w0)
  path = [w0]
  for k in range(Nsteps):
    w = prev - np.array(mu*dP(prev[0], prev[1], x))
    prev = w
    path.append(w)
  path = np.array(path)

  w1 = np.linspace(0,l)
  w2 = np.linspace(0,l)
  p_arr = np.zeros([w1.shape[0], w1.shape[0]])
  dp_arr = np.zeros([w1.shape[0], w1.shape[0]])
  i = 0
  for coord1 in w1:
    j = 0
    for coord2 in w2:
      p_arr[i][j] = P(coord1, coord2,x)
      dp_arr[i][j] = np.linalg.norm(dP(coord1,coord2, x))
      j +=1
    i += 1
  funcan, (g1, g2) = plt.subplots(1,2, sharey = True)
  val1 = g1.contourf(w1,w2, p_arr, cmap = "viridis")
  plt.colorbar(val1, ax = g1)
  g1.set_title("Func P(w)")
  g1.set_xlabel("w1 and w2")
  val1 = g2.contourf(w1,w2, dp_arr, cmap = "plasma")
  g2.set_title("Func dP(w)")
  g2.set_xlabel("w1 and w2")
  funcan.set_size_inches(12,5)
  g1.plot(path.T[0], path.T[1], 'o', linestyle='-', color='red')
  g2.plot(path.T[0], path.T[1], 'o', linestyle='-', color='red')
  plt.show()

  return path


N = 200
l = 1
x = np.random.rand(N)*l
path = gradient_descent(P, dP, [0.3, 0.6], 0.001, 100)
```

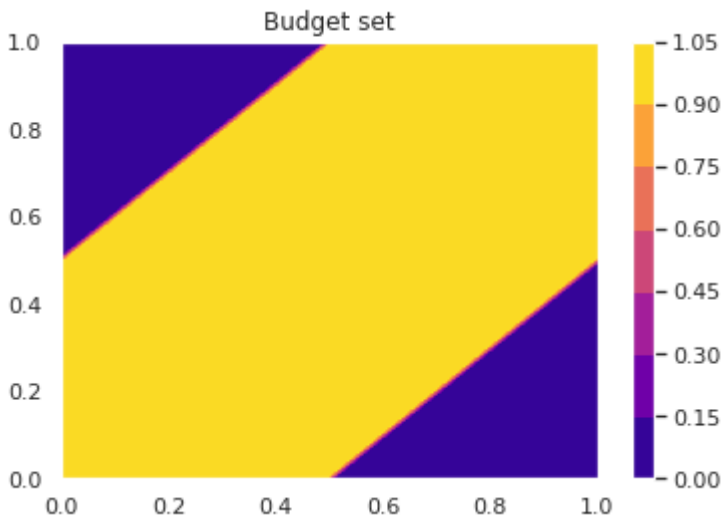


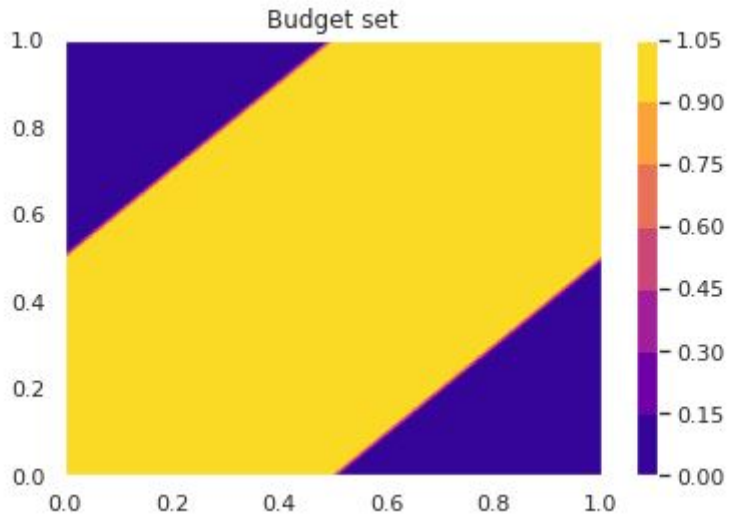
Construction is almost underway, but new safety regulations do not allow stations to be on the distance more than 1/2:

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

Plot our feasible set. Is it convex? YES IT IS

```
w1 = np.linspace(0,1)
w2 = np.linspace(0,1)
p = np.zeros([w1.shape[0], w1.shape[0]])
i = 0
for coord1 in w1:
  j = 0
  for coord2 in w2:
    p_arr[i][j] = abs(coord1 - coord2) <= l/2
    j += 1
  i += 1
funcan, g1 = plt.subplots(1, 1, sharey=True)
c1 = g1.contourf(w1, w2, p_arr, cmap="plasma")
plt.colorbar(c1, ax = g1)
g1.set_title('Budget set')
plt.show()
```



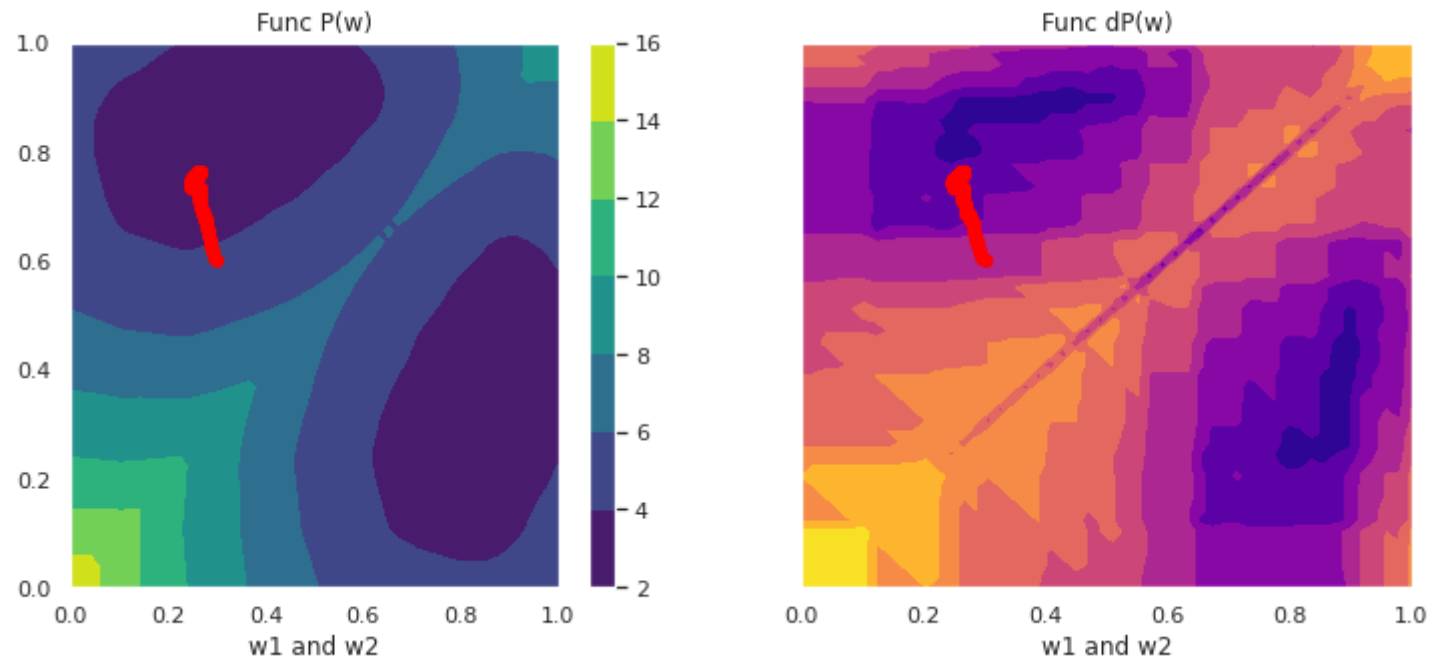


Write `conditional_SGD`, which returns the entire optimization trajectory (w_k) after a fixed number of steps and draws the process on the graphs P and ∇P that were above.

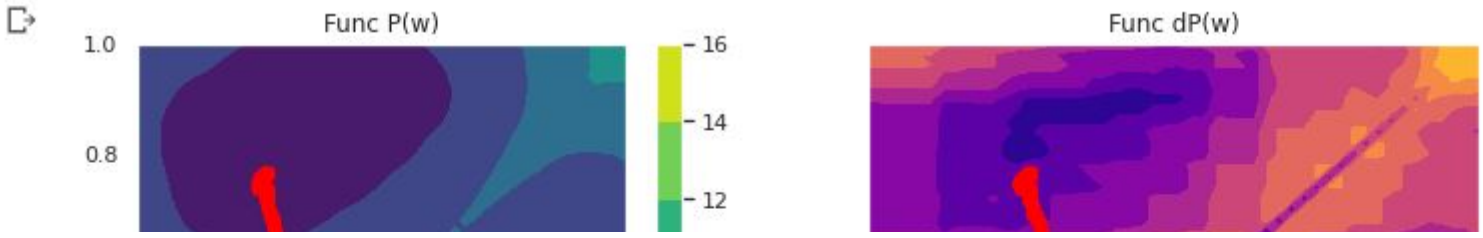
The conditional gradient descent method consists in making a gradient step and then checking if the obtained point belongs to the feasible set. If it belongs to the target set, the algorithm continues, otherwise a projection to the feasible set is made.

```
def dP_sigma(w1, w2, x, p=0.5):
  random_mask = np.random.binomial(1, p, x.shape)
  dP1 = sum([np.sign(w1 - x_i)*(abs(w1 - x_i) < abs(w2 - x_i))*(x_i != 0)
for x_i in x*random_mask])
  dP2 = sum([np.sign(w2 - x_i)*(abs(w1 - x_i) >= abs(w2 - x_i))*(x_i != 0)
for x_i in x*random_mask])
  dP = np.array([dP1, dP2])
  return dP


def P(w1, w2, x):
  P = sum([min([abs(w1 - coord), abs(w2 - coord)]) for coord in x])
  return P

def conditional_SGD(P, dP_sigma, w0, mu, Nsteps):
  prev = np.array(w0)
  path = [w0]
  step = 0.002
  for k in range(Nsteps):
    w = prev - np.array(mu*dP_sigma(prev[0], prev[1], x))
    if (abs(w[0] - w[1]) > l/2):
      if(w[0] - w[1] > l/2):
        while (w[0] - w[1] > l/2):
```

```
        w[0]-=step
        w[1]+=step
    else:
      while (w[1] - w[0] > l/2):
        w[0]+=step
        w[1]-=step
  prev = w
  path.append(w)
path = np.array(path)

w1 = np.linspace(0,l)
w2 = np.linspace(0,l)
p_arr = np.zeros([w1.shape[0], w1.shape[0]])
dp_arr = np.zeros([w1.shape[0], w1.shape[0]])
i = 0

for coord1 in w1:
  j = 0
  for coord2 in w2:
    p_arr[i][j] = P(coord1, coord2,x)
    dp_arr[i][j] = np.linalg.norm(dP(coord1,coord2, x))
    j +=1
  i += 1
funcan, (g1, g2) = plt.subplots(1,2, sharey = True)
val1 = g1.contourf(w1,w2, p_arr, cmap = "viridis")
plt.colorbar(val1, ax = g1)
g1.set_title("Func P(w)")
g1.set_xlabel("w1 and w2")
  val1 = g2.contourf(w1,w2, dp_arr, cmap = "plasma")
  g2.set_title("Func dP(w)")
  g2.set_xlabel("w1 and w2")
  funcan.set_size_inches(12,5)
  g1.plot(path.T[0], path.T[1], 'o', linestyle='-', color='red')
  g2.plot(path.T[0], path.T[1], 'o', linestyle='-', color='red')
  plt.show()

  return path


def projection(w):
    pass


N = 25
l = 1
x = np.random.rand(N)*l
path = conditional_SGD(P, dP_sigma, [0.3, 0.6], 0.001, 200)
print(path[-1])
```




```
[0.263 0.76 ]
```



We have same loss function

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

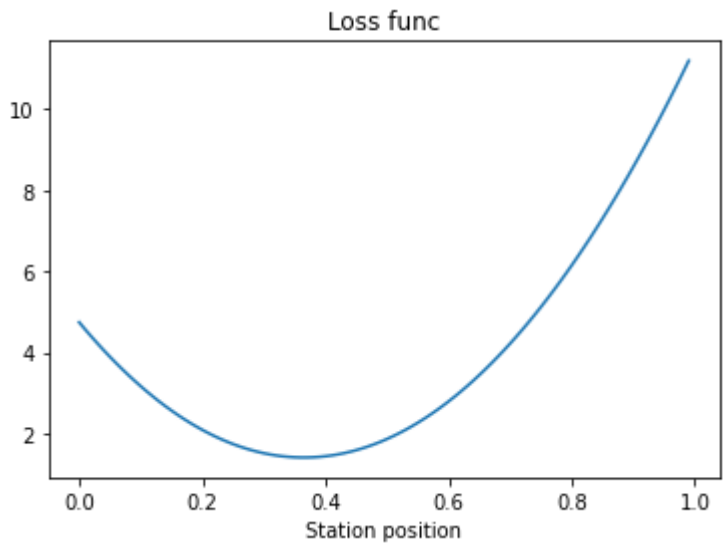
where $p(d) = d^2$

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

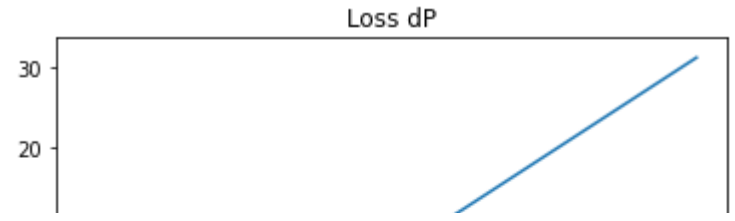
```
def P(w, x):
    P = sum([(w - coord)**2 for coord in x])
    return P

def dP(w, x):
    #coord = x[0]
    #dP = 2*sum((w - coord))
    dP = 2*sum([(w - coord) for coord in x])
    return dP

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


print(dP(w,x))
```

```
[-18.24422168 -17.22381352 -16.20340535 -15.18299719 -14.16258903
 -13.14218086 -12.1217727  -11.10136454 -10.08095637  -9.06054821
  -8.04014005  -7.01973188  -5.99932372  -4.97891556  -3.95850739
  -2.93809923  -1.91769107  -0.8972829    0.12312526   1.14353342
   2.16394159   3.18434975   4.20475791   5.22516608   6.24557424
   7.2659824    8.28639057   9.30679873  10.32720689  11.34761506
  12.36802322  13.38843138  14.40883955  15.42924771  16.44965587
  17.47006403  18.4904722   19.51088036  20.53128852  21.55169669
  22.57210485  23.59251301  24.61292118  25.63332934  26.6537375
  27.67414567  28.69455383  29.71496199  30.73537016  31.75577832]
```

```
plt.plot(np.arange(0, l, 0.01), [P(w,x) for w in np.arange(0,l, 0.01)])
plt.title("Loss func")
plt.xlabel("Station position")
plt.show()
```

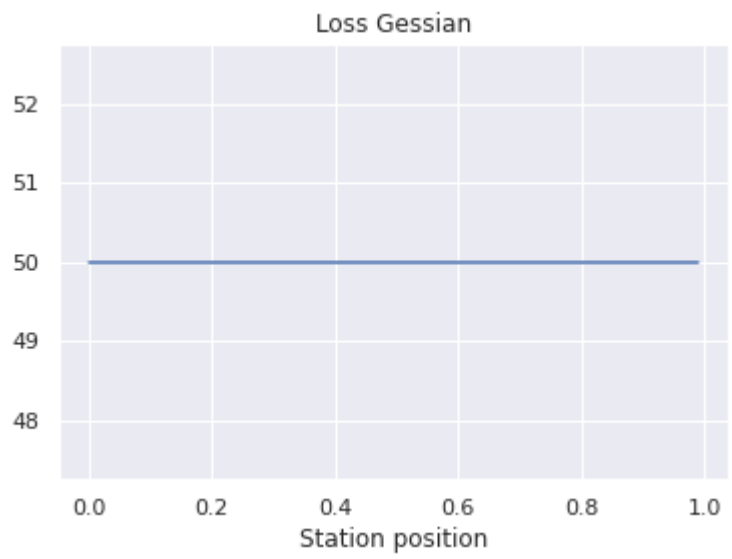



```
plt.plot(np.arange(0, l, 0.01), [dP(w,x) for w in np.arange(0,l, 0.01)])
plt.title("Loss dP")
plt.xlabel("Station position")
plt.show()
```



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


```
plt.plot(np.arange(0, 1, 0.01), [ddP(w,x) for w in np.arange(0,1, 0.01)])
plt.title("Loss Gessian")
plt.xlabel("Station position")
plt.show()
```



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(P, dP, ddP, w0, Nsteps):
    w = w0
    trajectory = [w]
    for k in range(Nsteps):
      trajectory.append(w)
      w -= dP(w, x)/ddP(w,x)
    trajectory.append(w)
    return trajectory
```

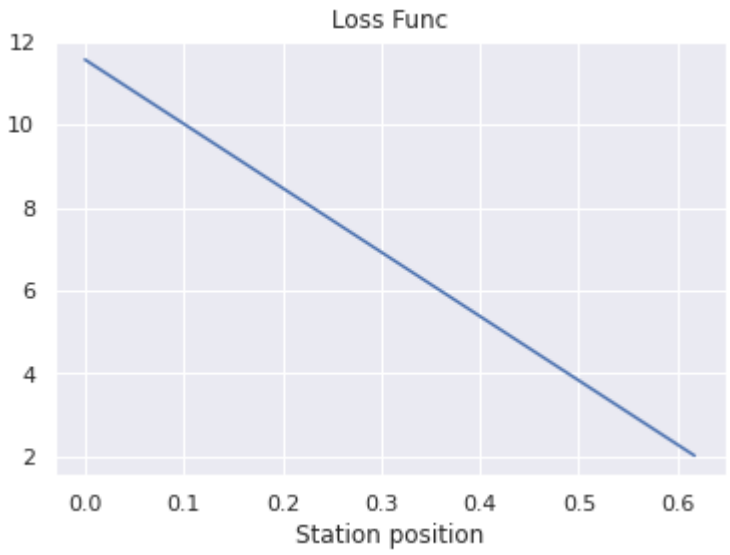
```
type(x)
```

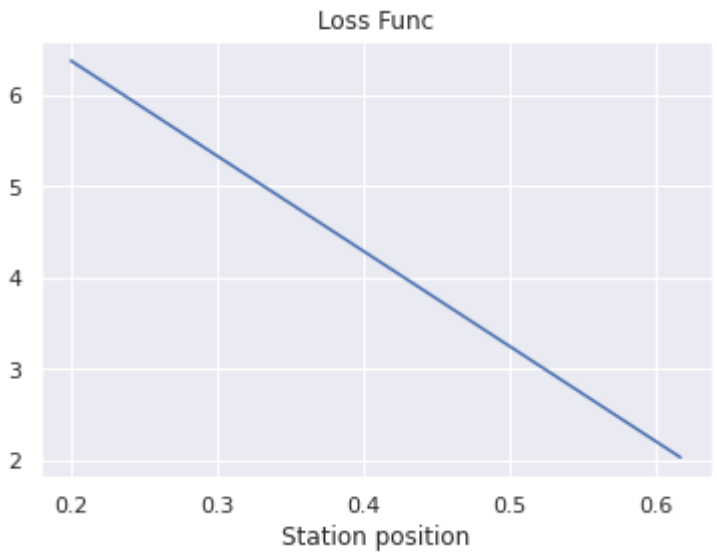
```
numpy.ndarray
```

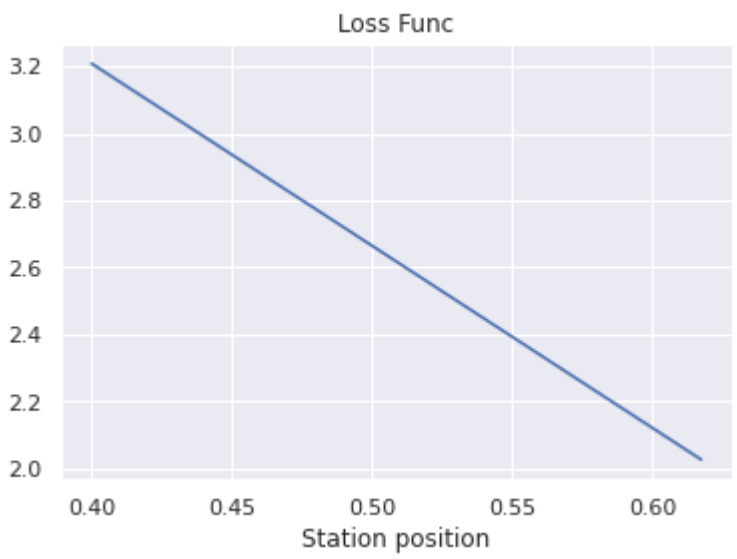
Investigate the behavior of Newton's method depending on the different initial point w_0.

```
Nsteps = 10
print(x)
for w0 in np.arange(0,1, 0.2):
trajectory = newton_descent(P, dP, ddP, w0, Nsteps)
plt.plot(trajectory, P(trajectory, x))
plt.title("Loss Func")
plt.xlabel("Station position")
plt.show()
```

```
[0.85261462 0.1123624  0.79188859 0.21343772 0.64455103 0.10207615
 0.52299976 0.55234989 0.98233888 0.90671653 0.90952903 0.99115424
 0.39773217 0.92158202 0.55658552 0.45415093 0.86083249 0.24424355
 0.30327797 0.24902978 0.64254816 0.88139495 0.71785088 0.95508592
 0.6687888 ]
```

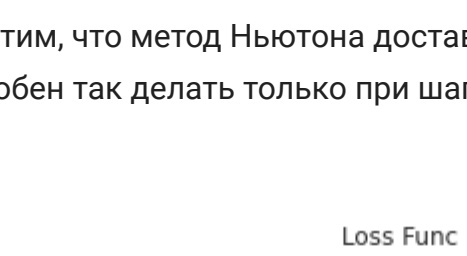






Заметим, что метод Ньютона доставляет минимум квадратичной функции независимо от начальной точки. А градиетный спуск способен так делать только при шаге

$$\frac{1}{\nabla^2 P}$$



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

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

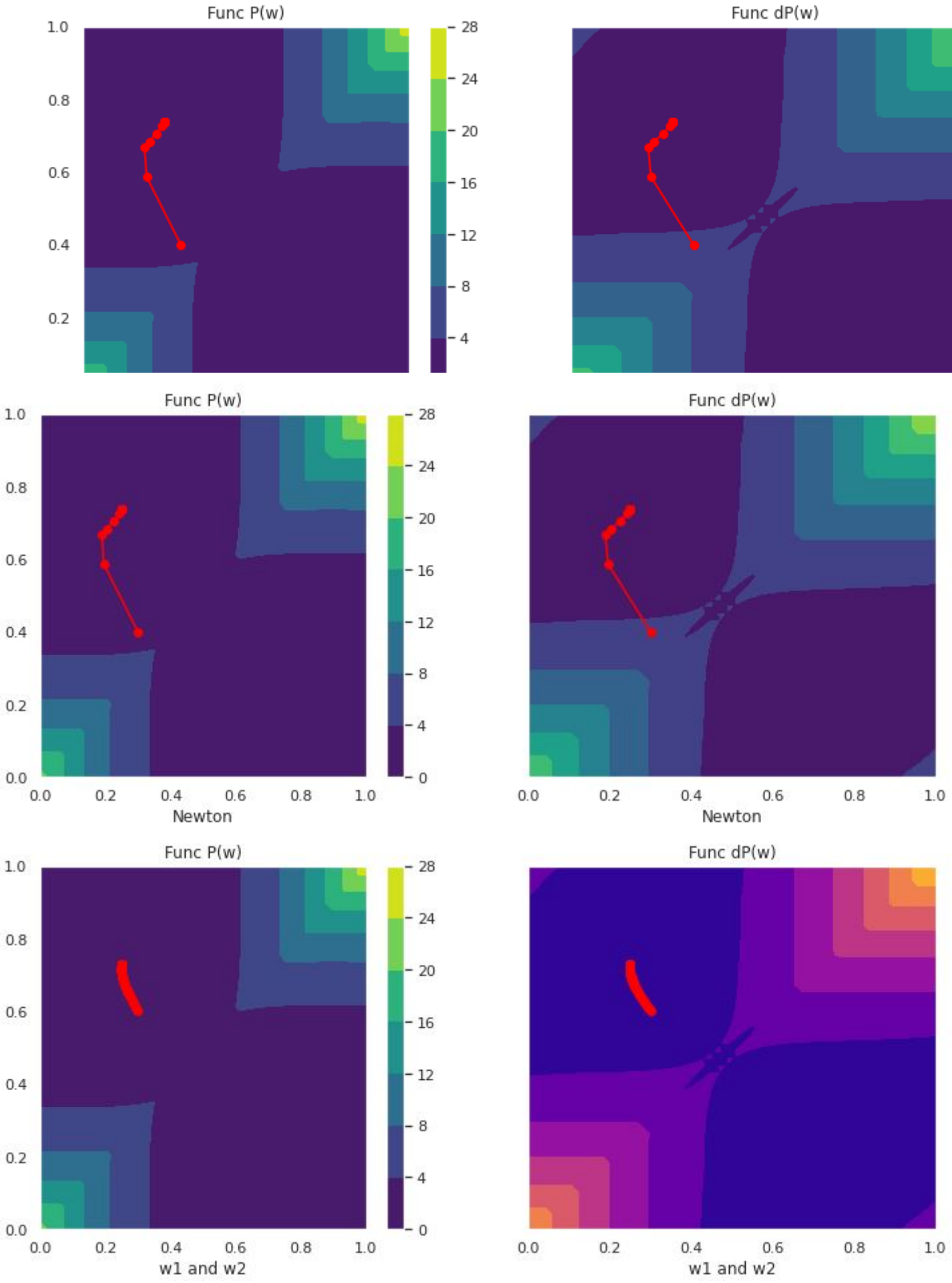
with $p(d) = d^3$ using Newton method and return optimization trajectory. Compare results with gradient descent.

```
def P(w1, w2, x):
  P = sum([min([abs(w1 - coord), abs(w2 - coord)])**3 for coord in x])
  return P
def dP(w1, w2, x):
  dP1 = 3*sum([np.sign(w1 - coord)*(w1 - coord)**2*(abs(w1 - coord) < abs(w2 - coord)) for coord in x])
  dP2 = 3*sum([np.sign(w2 - coord)*(w2 - coord)**2*(abs(w1 - coord) >= abs(w2 -coord)) for coord in x])
  dP = np.array([dP1, dP2])
  return dP
def ddP(w1, w2, x):
```

```
  ddP = np.zeros((2, 2))
  ddP[0][0] = 6*sum([abs(w1 - coord)*(abs(w1 - coord) < abs(w2 - coord)) for coord in x])
  ddP[1][1] = 6*sum([abs(w2 - coord)*(abs(w1 - coord) >= abs(w2 - coord)) for coord in x])
  return ddP
def multi_newton(P, dP, ddP, w0, Nsteps):
  prev = w0
  path = [w0]
  for i in range(Nsteps):
    w = prev - np.linalg.inv(ddP(prev[0], prev[1], x))@(dP(prev[0], prev[1], x))
    prev = w
    path.append(w)
  path = np.array(path)
  w1 = np.linspace(0,l)
  w2 = np.linspace(0,l)
  p_arr = np.zeros([w1.shape[0], w1.shape[0]])
  dp_arr = np.zeros([w1.shape[0], w1.shape[0]])
  i = 0
  for coord1 in w1:
    j = 0
    for coord2 in w2:
      p_arr[i][j] = P(coord1, coord2,x)
      dp_arr[i][j] = np.linalg.norm(dP(coord1,coord2, x))
      j +=1
    i += 1
  funcan, (g1, g2) = plt.subplots(1,2, sharey = True)
  val1 = g1.contourf(w1,w2, p_arr, cmap = "viridis")
  plt.colorbar(val1, ax = g1)
  g1.set_title("Func P(w)")
  g1.set_xlabel("Newton")
  val1 = g2.contourf(w1,w2, dp_arr, cmap = "viridis")
  g2.set_title("Func dP(w)")
  g2.set_xlabel("Newton")
  funcan.set_size_inches(12,5)
  g1.plot(path.T[0], path.T[1], 'o', linestyle='-', color='red')
  g2.plot(path.T[0], path.T[1], 'o', linestyle='-', color='red')
  plt.show()

  return path



#def multi_newton(x, dP, ddP, w0, Nsteps):
    ### YOUR CODE
 #   return trajectory
```

ПЕРВАЯ ПАРА ГРАФИКОВ - ЭТО МЕТОД НЬЮТОНА ДЛЯ P И dP. ВТОРАЯ ПАРА - ГРАДИЕНТНЫЙ СПУСК!

```
N = 100
l = 1
x = np.random.rand(N)*l
path1 = multi_newton(P, dP, ddP, [0.3, 0.4], 100)
def dP_sigma(w1, w2, x, p=0.5):
  random_mask = np.random.binomial(1, p, x.shape)
  dP1 = 3*sum([np.sign(w1 - coord)*(w1 - coord)**2*(abs(w1 - coord) < abs(w2 - coord))*(coord != 0) for coord in x*random_mask])
  dP2 = 3*sum([np.sign(w2 - coord)*(w2 - coord)**2*(abs(w1 - coord) >= abs(w2 -coord))*(coord != 0) for coord in x*random_mask])
  dP = np.array([dP1, dP2])
  return dP
path2 = gradient_descent(P, dP, [0.3, 0.6], 0.001, 100)
#path = gradient_descent(P, dP, [0.3, 0.6], 0.001, 100)
```



Newton descent trajectory

How condition number affects gradient optimization

Consider a function of two variables:

$$f(x_1, x_2) = x_1^2 + kx_2^2,$$

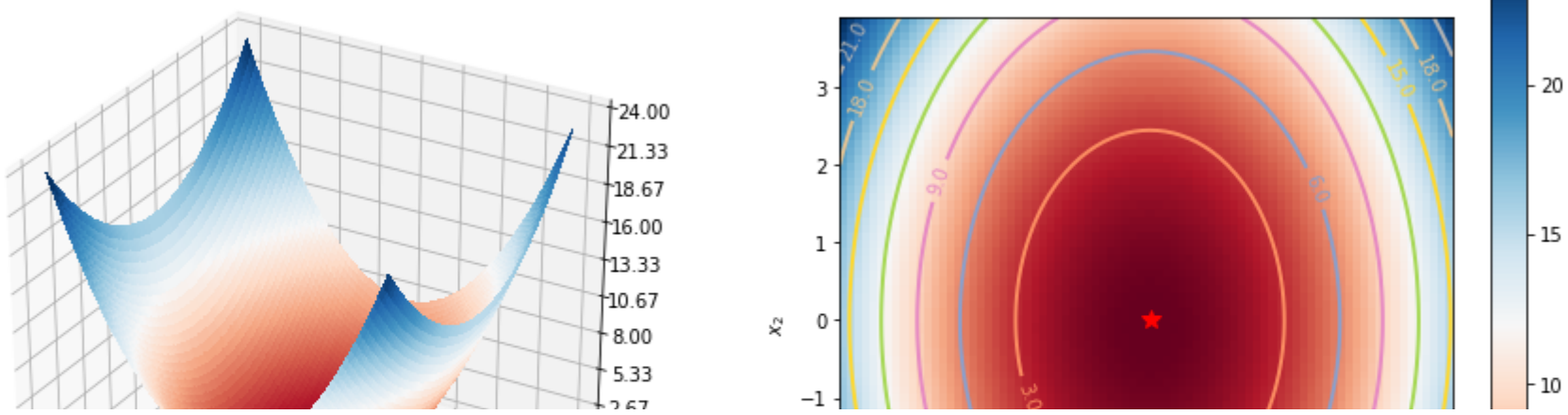
where k is some parameter

```
def f(x, *f_params):
    if len(f_params) == 0:
        k = 2
    else:
        k = float(f_params[0])
    x_1, x_2 = x
    return x_1**2 + k*x_2**2

def df(x, *f_params):
    if len(f_params) == 0:
        k = 2
    else:
        k = float(f_params[0])
    return np.array([2*x[0], 2*k*x[1]])
```

```
%matplotlib inline
```

```
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
import numpy as np

def plot_3d_function(x1, x2, f, title, *f_params, minima = None, iterations = None):
    '''
    '''
    low_lim_1 = x1.min()
    low_lim_2 = x2.min()
    up_lim_1  = x1.max()
    up_lim_2  = x2.max()

    X1,X2 = np.meshgrid(x1, x2) # grid of point
    Z = f((X1, X2), *f_params) # evaluation of the function on the grid

    # set up a figure twice as wide as it is tall
    fig = plt.figure(figsize=(16,7))
    fig.suptitle(title)

    #===============
    #  First subplot
    #===============
    # set up the axes for the first plot
    ax = fig.add_subplot(1, 2, 1, projection='3d')

    # plot a 3D surface like in the example mplot3d/surface3d_demo
surf = ax.plot_surface(X1, X2, Z, rstride=1, cstride=1,
                       cmap=cm.RdBu,linewidth=0, antialiased=False)

ax.zaxis.set_major_locator(LinearLocator(10))
ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))
if minima is not None:
    minima_ = np.array(minima).reshape(-1, 1)
    ax.plot(*minima_, f(minima_), 'r*', markersize=10)



#===============
# Second subplot
#===============
# set up the axes for the second plot
ax = fig.add_subplot(1, 2, 2)

# plot a 3D wireframe like in the example mplot3d/wire3d_demo
im = ax.imshow(Z,cmap=plt.cm.RdBu,  extent=[low_lim_1, up_lim_1, low_lim_2, up_lim_2])
cset = ax.contour(x1, x2,Z,linewidths=2,cmap=plt.cm.Set2)
ax.clabel(cset,inline=True,fmt='%1.1f',fontsize=10)
fig.colorbar(im)
ax.set_xlabel(f'$x_1$')
ax.set_ylabel(f'$x_2$')

if minima is not None:
    minima_ = np.array(minima).reshape(-1, 1)
    ax.plot(*minima_, 'r*', markersize=10)

    if iterations is not None:
        for point in iterations:
            ax.plot(*point, 'go', markersize=3)
        iterations = np.array(iterations).T
        ax.quiver(iterations[0,:-1], iterations[1,:-1], iterations[0,1:]-iterations[0,:-1], iterations[1,1:]-iterations[1,:-1], scale_units='x

    plt.show()

up_lim  = 4
low_lim = -up_lim
x1 = np.arange(low_lim, up_lim, 0.1)
x2 = np.arange(low_lim, up_lim, 0.1)
k=0.5
title = f'$f(x_1, x_2) = x_1^2 + k x_2^2, k = {k}$'

plot_3d_function(x1, x2, f, title, k, minima=[0,0])
```

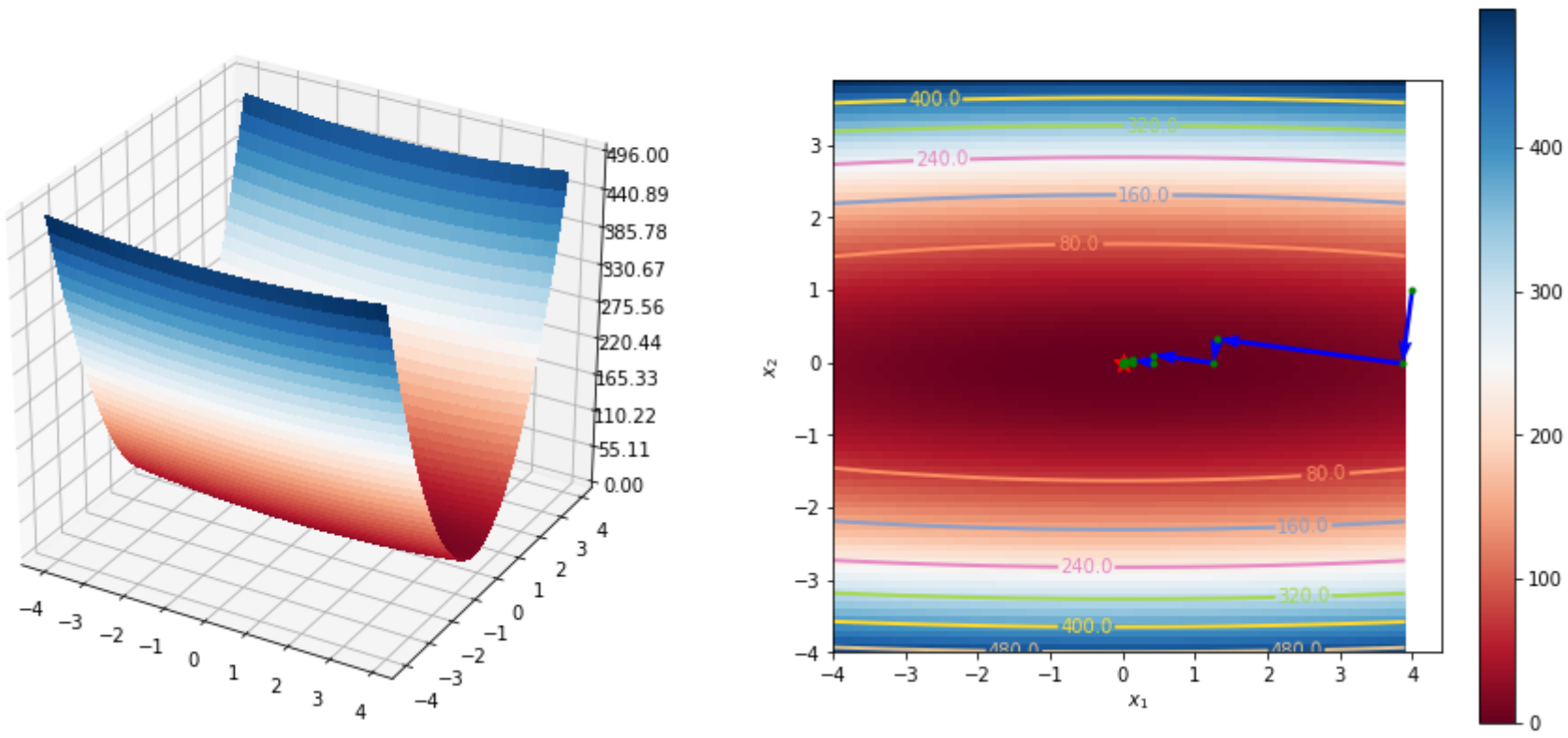
$$f(x_1, x_2) = x_1^2 + kx_2^2, k = 0.5$$



For example, steepest descent algorithm will be plotted with the following code:

```
from scipy.optimize import minimize_scalar

def steepest_descent(x_0, f, df, *f_params, df_eps = 1e-2, max_iter = 1000):
    iterations = []
    x = np.array(x_0)
    iterations.append(x)
    while np.linalg.norm(df(x, *f_params)) > df_eps and len(iterations) <= max_iter:
        res = minimize_scalar(lambda alpha: f(x - alpha * df(x, *f_params), *f_params))
        alpha_opt = res.x
        x = x - alpha_opt * df(x, *f_params)
        iterations.append(x)
    #print(f'Finished with {len(iterations)} iterations')
    return iterations

x_0 = [4,1]
k = 30
iterations = steepest_descent(x_0, f, df, k, df_eps = 1e-9)
title = f'$f(x_1, x_2) = x_1^2 + k x_2^2, k = {k}$'

plot_3d_function(x1, x2, f, title, k, minima=[0,0], iterations = iterations)
```

$$f(x_1, x_2) = x_1^2 + kx_2^2, k = 30$$

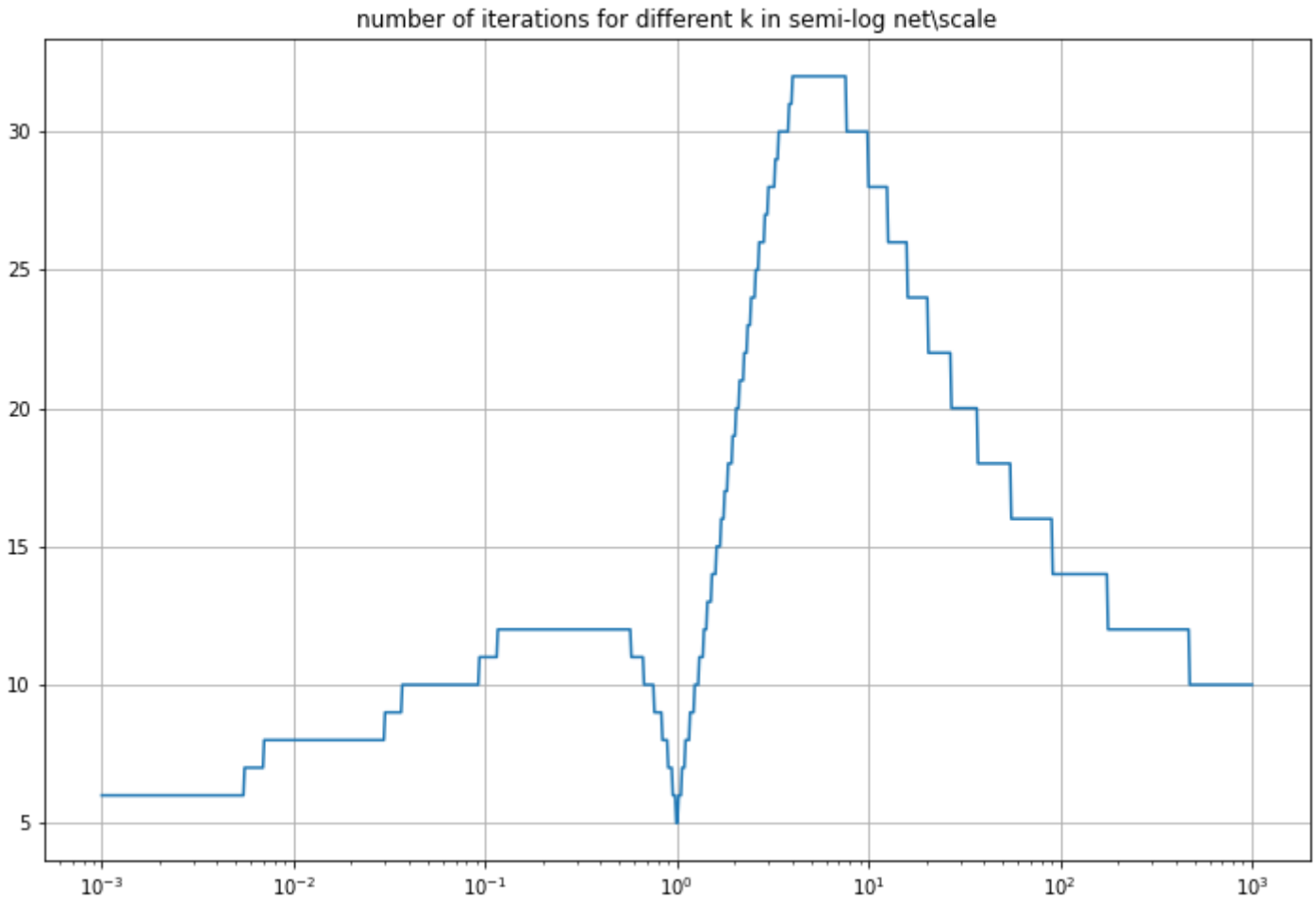


Plot the number of iterations required for the convergence of the steepest descent algorithm (up to the condition $\|\nabla f(x_k)\| \leq \varepsilon = 10^{-7}$) depending on the value of k. Consider the interval $k \in [10^{-3}; 10^3]$ (it will be convenient to use the function `ks = np.logspace(-3,3)`) and plot on the X axis in logarithmic scale `plt.semilogx()` or `plt.loglog()` for double log scale.

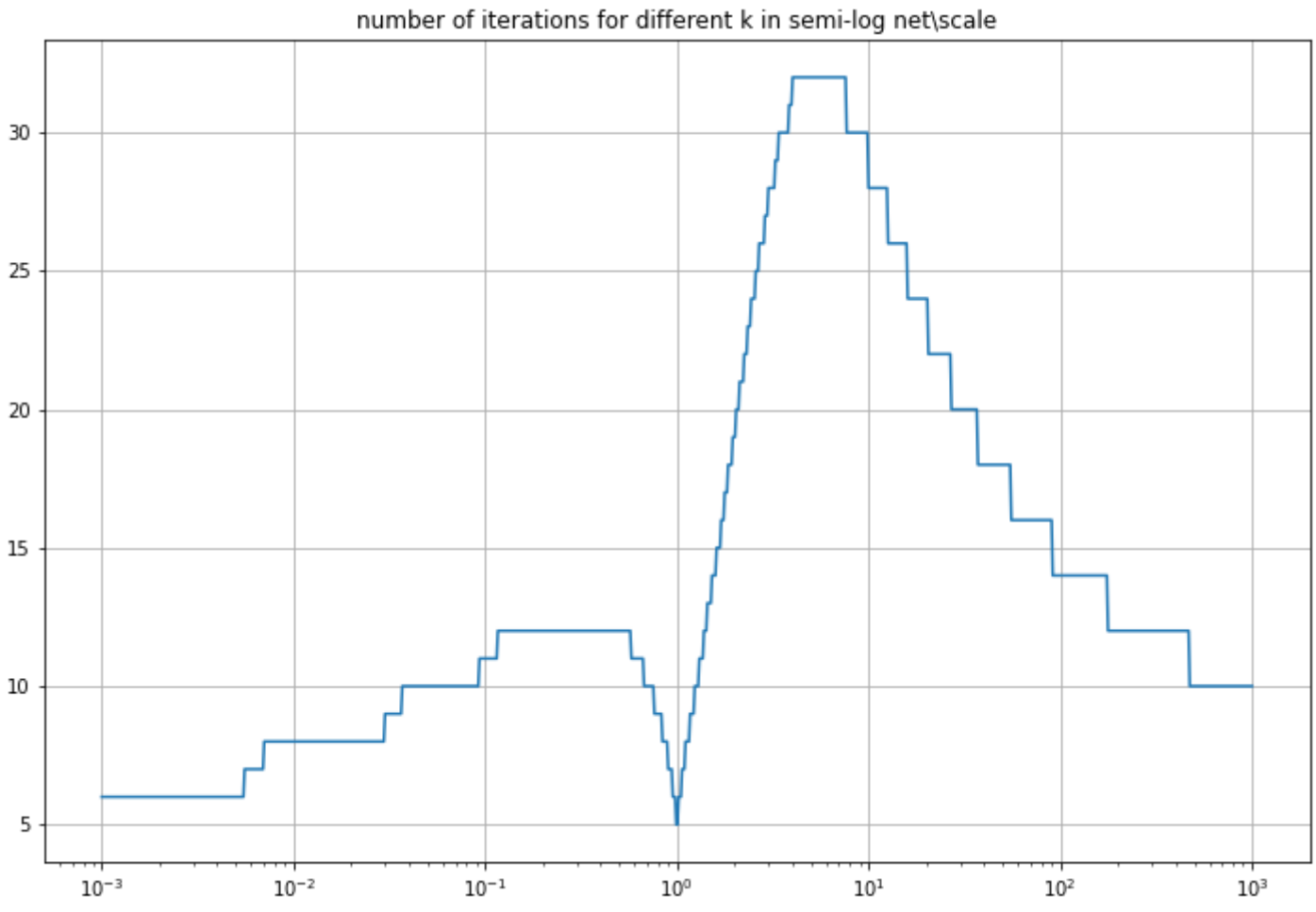
Make the same graphs for the suitable constant stepsize. Explain the results.

```
net_space = np.logspace(-3,3,1000)
n_iter = []
x_0 = [10,5]
for i in net_space:
  step = steepest_descent(x_0, f, df, i, df_eps = 1e-7)
  n_iter.append(len(step))
fig, ax = plt.subplots(figsize = (12,8))
ax.semilogx(net_space, n_iter)
ax.set(title='number of iterations for different k in semi-log net\scale')
ax.grid()
```

```
fig.savefig("t_right_semilog.png")
plt.show()
```




```
net_space = np.logspace(-3,3,1000)
n_iter = []
x_0 = [10,5]
for i in net_space:
  step = steepest_descent(x_0, f, df, i, df_eps = 1e-7)
  n_iter.append(len(step))
fig, ax = plt.subplots(figsize = (12,8))
ax.semilogx(net_space, n_iter)
ax.set(title='number of iterations for different k in semi-log net\scale')
ax.grid()
fig.savefig("t_right_semilog.png")
plt.show()
```



==YOUR ANSWER==

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==

For following problem:

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

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

==YOUR ANSWER==

Implement projected gradient descent for following task.

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

Compare with analytical solution and `scipy.optimize`

```
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):
    #asw = (A@x-b)
    asw = np.dot((A@x-b),(A@x-b))

    return asw


def proj(C, d, x):
    ### YOUR CODE
    C_1 = C.T
    proj = np.linalg.inv(C@C_1)@(d-C@x)
    return x + C_1@(np.linalg.inv(C@C_1)@(d-C@x))
```

```
x = np.zeros(20)
```

```
### Projected gradient descent
"""def gd(dP, w0, mu, Nsteps):
  prev = np.array(w0)
  #path = [w0]
  for k in range(Nsteps):
    w = prev - np.array(mu*dP(prev[0], prev[1], x))
    prev = w
    #path.append(w)
  #path = np.array(path)
  return w"""
from numpy.ma.core import append
def gd(P, dP, w0, mu, Nsteps):
    init = w0
    Arr = [0]*w0.size
    path = [Arr for i in range(1+Nsteps)]
    path[0] = w0
    for k in range(Nsteps):
      path[k+1] = (proj(C,d,path[k] - mu*dP(path[k])))
      #print(init)
    #path.append()
    return path

def df(x):
  return 2*A.T@(A@x - b)

"""
Arr = [0]*x.size
```

```
    #path = np.zeros((Nsteps + 1, w0.size))
path = [Arr for i in range(1+10)]
path
"""
```

```
'\nArr = [0]*x.size\n    #path = np.zeros((Nsteps + 1, w0.size))\npath = [Arr for i in range(1+10)]\npath\n'
```

```
projection = gd(f,df, x, 0.001, 500)
print(projection[-1], f(projection[-1]))
```

```
[ 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] 137.62185939974725
```

Comparison with `scipy.optimize`

Совпадает с значением scipy.*optimize*

```
from scipy.optimize import minimize as min
from scipy.optimize import LinearConstraint as LC
value = min(f, x, jac=df, constraints = LC(C , d, d))
print(value.x)
print(value.fun)
```

```
[ 0.19141191  0.20427827  0.00067643 -0.22399297 -0.09994442 -0.05341131
 -0.21970833  0.12296092  0.01729651  0.05976737 -0.35036119 -0.21625861
 -0.01128399 -0.2206766   0.02382163 -0.10195978 -0.07071159  0.23374888
 -0.00852641 -0.08666988]
137.62185947874644
```

Comparison with analytical solution

Совпадает с значением scipy.*optimize*

```
from numpy.linalg import inv

u_sol = (C@inv(A.T@A)@A.T@b-d)
C_1 = C.T

x_sol = np.dot(inv(A.T @ A),  -C_1@ inv(C@inv(A.T @ A)@C_1)@u_sol+A.T@b)
print(f"analytical solution = {x_sol}")
print(f"function value = {f(x_sol)}")
```

```
from scipy.optimize import minimize as min
from scipy.optimize import LinearConstraint as LC
value = min(f, x, jac=df, constraints = LC(C , d, d))
print(f"scipy.optimize = {value.x}")
print(f"scipy.optimize function value = {value.fun}")
```

```
analytical solution = [ 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]
function value = 137.62185939974722
scipy.optimize = [ 0.19141191  0.20427827  0.00067643 -0.22399297 -0.09994442 -0.05341131
 -0.21970833  0.12296092  0.01729651  0.05976737 -0.35036119 -0.21625861
 -0.01128399 -0.2206766   0.02382163 -0.10195978 -0.07071159  0.23374888
 -0.00852641 -0.08666988]
scipy.optimize function value = 137.62185947874644
```

Получим, что методы дают практически одинаковый результат, ошибка составляет

```
print(f"Невязка = {abs(value.fun - f(x_sol))}")
```

```
Невязка = 7.89992213867663e-08
```

✓ 5 сек. выполнено в 15:29