Homework 1. Prokhorov

- Домашку присылать в виде `.pdf` файла на адрес `homework@merkulov.top`.
- Дедлайн: **28 марта 22:59**.
- Есть несколько способов конвертировать `.ipynb` в `.pdf`. Самый простой - сохранить ноутбук как `.html`, а затем распечатать это в `.pdf` файл, нажав `ctrl + P` в браузере.
- Займитесь этим вопросом заранее, чтобы в последний момент не получить из за этого 0 баллов.
- Все ячейки должны быть запущены, а графики построены.

Sequence convergence

Problem 1

Определить скорость сходимости следующих последовательностей:

1. $r_k = \left\{(0.707)^k\right\}_{k=1}^{\infty}$
2. $r_k = \left\{(0.707)^{2^k}\right\}_{k=1}^{\infty}$
3. $r_k = \left\{\dfrac{1}{k^2}\right\}_{k=1}^{\infty}$
4. $r_k = \left\{\dfrac{1}{k!}\right\}_{k=1}^{\infty}$
5. $r_k = \begin{cases} \frac{1}{k}, & \text{if } k \text{ is even} \\ \frac{1}{k^2}, & \text{if } k \text{ is odd} \end{cases}$
6. $r_k = \begin{cases} \frac{1}{k^k}, & \text{if } k \text{ is even} \\ \frac{1}{k^{2k}}, & \text{if } k \text{ is odd} \end{cases}$

Во всех задачах $r^* = 0$, поэтому для, например, линейной сходимости будем проверять условие

$$|r_{k+1}| \leq q|r_k|, \qquad 0 < q < 1$$

- $|r_{k+1}| = 0.707|r_k|, \qquad q = 0.707 \in (0, 1)$

Сходимость линейная. Сверхлинейной сходимости нет, потому что

$$\frac{|r_{k+1}|}{|r_k|} = 0.707 \nrightarrow 0 \qquad \text{при} \qquad n \to \infty$$

- $|r_{k+1}| = (0.707^2)^{2^k} = (0.5)^{2^k} \leq q(0.707)^{2^k}, \qquad \text{подходит любое } \ 0.5 \leq q < 1$

Сходимость квадратичная.

- $|r_k| \leq Ck^\alpha, \qquad C = 1, \ \alpha = -\frac{1}{2}$

Сходимость сублинейная. Линейной сходимости нет, потому что не существует такого $q \in (0, 1)$, что

$$\frac{|r_{k+1}|}{|r_k|} = \left(1 - \frac{1}{k+1}\right)^2 \leq q$$

- $|r_{k+1}| = c_k|r_k|, \qquad c_k = \dfrac{1}{k+1} \to 0$

Сходимость сверхлинейная. Квадратичной сходимости нет, потому что не существует такого $q \in (0, 1)$, что

$$\frac{|r_{k+1}|}{|r_k|^2} = \frac{k!}{k+1} \leq q$$

- $|r_k| \leq Ck^\alpha, \qquad C = 1, \ \alpha = -1$

Сходимость сублинейная. Линейной сходимости нет, потому что не существует такого $q \in (0, 1)$, что для всех нечетных k

$$\frac{r_{k+1}}{r_k} = \frac{k^2}{k+1} \leq q$$

- $\limsup\limits_{k\to\infty} (r_k)^{1/k} = \limsup\limits_{k\to\infty} \begin{Bmatrix} 1/k \\ 1/k^2 \end{Bmatrix} = 0$

По тесту корней (root test), сходимость сверхлинейная. Квадратичной сходимости нет, потому что для нечетных k

$$\frac{r_{k+1}}{r_k^2} \to \infty \qquad \text{при} \qquad k \to \infty$$

Line search

Problem 2

Рассмотрите функцию $f(x) = x \cdot e^x + \sin e^x, \quad x \in [-20, 0]$. Рассмотрите методы локализации решения, при которых отрезок $[a, b]$ делится на 2 части в фиксированной пропорции $t : x_t = a + t * (b - a)$ (максимум дважды на итерации - как в методе дихотомии).

Проведите эксперименты при различных значениях $t \in [0, 1]$ и постройте график зависимости $N(t)$ - значения количества итераций, необходимых для достижения ε - точности от параметра t. Считать $\varepsilon = 10^{-7}$.

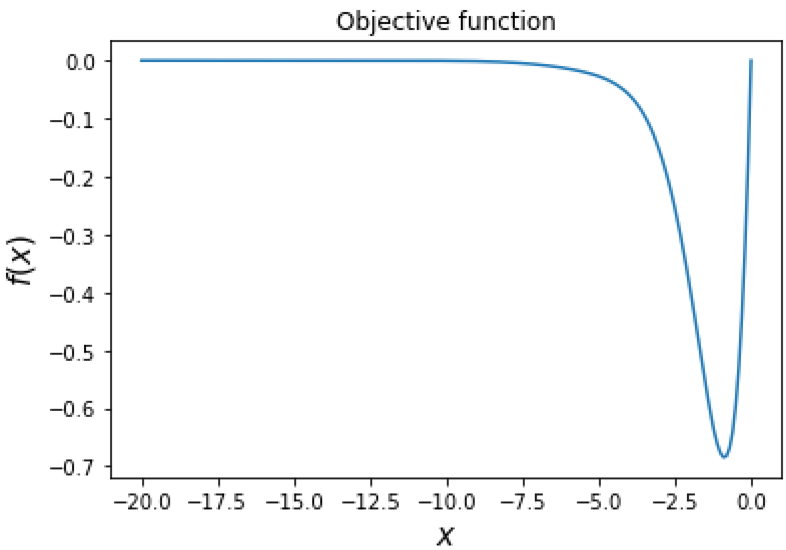
Обратите внимание, что в случае $t = 0.5$ данный метод точно совпадает с методом дихотомии.

In [7]:

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

def f_2(x):
    return (x + np.sin(x))*np.exp(x)

x = np.linspace(-20,0, 200)
plt.plot(x, f_2(x))
plt.title('Objective function')
plt.xlabel('$x$', fontsize=15)
plt.ylabel('$f(x)$', fontsize=15)
plt.show()
```



Моя интерпретация задания:

- Целевая функция считается унимодальной
- Под числом итераций понимается число вызовов функции f
- Остановка происходит при достижении eps-точности по координате

In [8]:

```
# exact solution calculated with Wolfram
x_exact = -0.874357860856
```

In [9]:

```
def line_search(f, a, b, t=0.5, eps=1e-7):

    # returns: (x_opt, N)
    #  x_opt - optimal point
    #  N - number of function calls (at unique points)

    N = 0

    c = a + (b - a) * t
    while (b - a > eps):
        y = a + (c - a) * t
        N += 2
        if f(y) <= f(c):
            b = c
            c = y
        else:
            z = b + (c - b) * t
            N += 1
            if f(c) <= f(z):
                a = y
                b = z
            else:
                a = c
                c = z

    return (a+b)/2, N
```

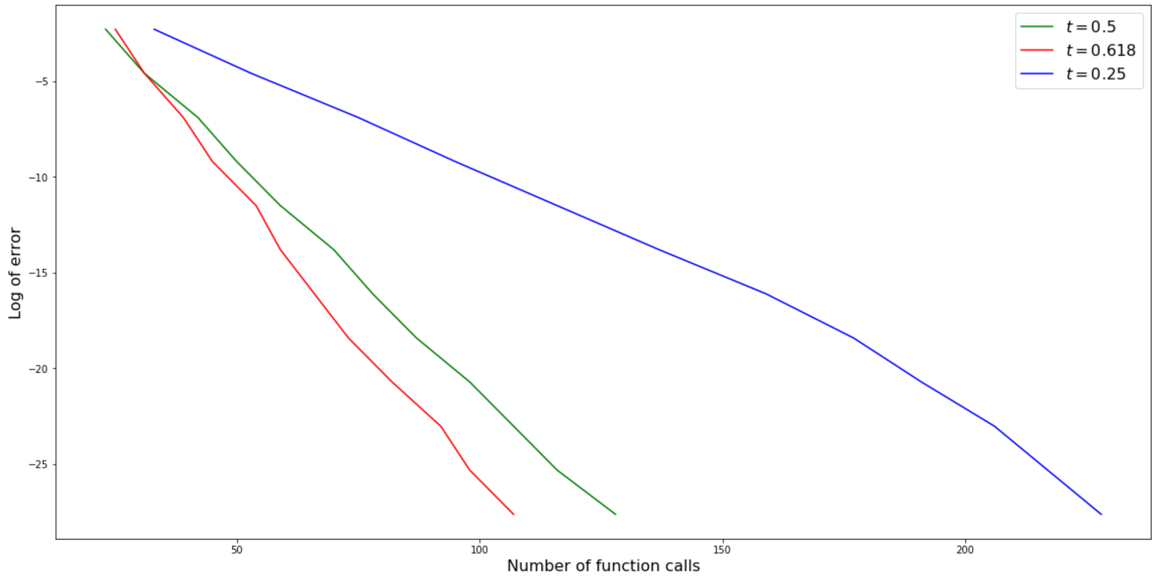
In [10]:

```
# зависимость числа итераций от точности для t = 0.5, t = 1/(golden ratio), t = 0.25
# видим, что сходимость линейная

epss = [1e-1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6, 1e-6, 1e-7, 1e-8, 1e-9, 1e-10, 1e-11
, 1e-12]
Ns_1 = []
Ns_2 = []
Ns_3 = []

for e in epss:
    x, N = line_search(f_2, -20, 0, 0.5, eps=e)
    Ns_1.append(N)
    x, N = line_search(f_2, -20, 0, 2/(np.sqrt(5)+1), eps=e)
    Ns_2.append(N)
    x, N = line_search(f_2, -20, 0, 0.25, eps=e)
    Ns_3.append(N)

plt.figure(figsize=(20,10))
plt.plot(Ns_1, np.log(epss), label='$t = 0.5$', color='green')
plt.plot(Ns_2, np.log(epss), label='$t = 0.618$', color='red')
plt.plot(Ns_3, np.log(epss), label='$t = 0.25$', color='blue')
plt.xlabel('Number of function calls', fontsize=16)
plt.ylabel('Log of error', fontsize=16)
plt.legend(fontsize=16)
plt.show()
```

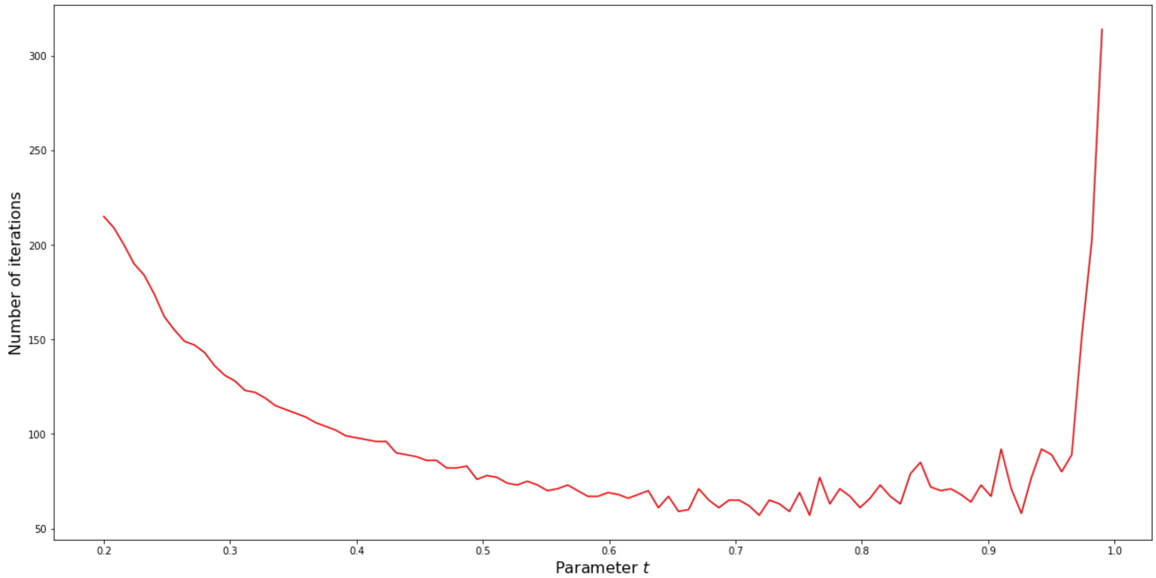


In [11]:

```
# Зависимость числа итераций от параметра t для ошибки eps=1e-7
# минимум получается в правой части по двум причинам:
# 1. оптимальным с точки зрения скорости сходимости  в общем случае является метод золотог
о сечения (0.618)
# 2. для конкретной функции точки минимума справа, поэтому оптимальнее идти вправо сразу

ts = np.linspace(0.2, 0.99, 100)
Ns = []
for t in ts:
    N_ = []
    for i in range(10):
        x, N = line_search(f_2, -20, 0, t, eps=1e-7)
        N_.append(N)
    N_ = np.array(N_)
    Ns.append(N_.mean())

plt.figure(figsize=(20,10))
plt.plot(ts, Ns, color='red')
plt.xlabel('Parameter $t$', fontsize=16)
plt.ylabel('Number of iterations', fontsize=16)
plt.show()
```



Zero order methods

Problem 3

Давайте располагать базовые станции беспроводной сети оптимально! Пусть у вас есть $N_{obj} = 10$ кластеров из 10 абонентов каждый. Давайте с помощью генетического алгоритма постепенно искать оптимальное количество и расположение базовых станций, чтобы минимизировать стоимость расстановки таких станций.

Ниже представлен один из возможных вариантов реализации генетического алгоритма.

Популяция

Это список из массивов размера `[N_stations x 2]`. Каждая особь при этом представляет собой набор координат станций на плоскости. Генерация случайного

Мутация

Определяется функцией `mutation()`. Из всех особей выбирается `mutation_rate` часть и к `mutation_rate` части её станций прибавляется случайный Гауссов шум. После этого к популяции добавляется особь со случайным количеством станций со случайными координатами.

Скрещивание

Определяется функциями `children_creation()` и `breed()`. Двум наборам станций ставится в соответствие третяя станция, из которой взяты четные станции одного родителя и нечетные станции другого.

Оценка стоимости особи

Определяется функцией `evaluate_generation()`. Итоговая стоимость, соответствующая конкретной особи складывается из себестоимости построения базовых станций (каждая стоит `station_cost`) за вычетом прибыли от каждого клиента. Прибыль от каждого клиента обратна пропорциональна расстоянию до "своей" базовой станции. Каждый клиент присоединяется только к одной (ближайшей) базовой станции с помощью функции `find_nearest_station()`. Кроме того, прибыль от каждого абонента обратно пропорциональна числу абонентов на данной базовой станции (у каждой станции есть число подсоединенных к ней абонентов `stations_load`). Заметим так же, что, начиная с некоторой близости к абонента к базовой станции, прибыль клиента перестает расти (в нашем алгоритме она одинакова в радиусе `0.1` от базовой станции, после чего линейно убывает).

Ваша задача состоит в том, чтобы придумать любые модификации к предложенным процедурам в рамках генетического алгоритма так, чтобы итоговое качество работы алгоритма было лучше. Предложите, опишите и протестируйте идеи улучшения алгоритма.

In [ ]:

```
%matplotlib notebook

import numpy as np
from scipy.spatial.distance import cdist
from random import shuffle, sample
from copy import deepcopy
import random
from plotly.subplots import make_subplots
import plotly.graph_objects as go
from IPython.display import clear_output
import matplotlib.pyplot as plt

def generate_problem(N_obj, N_abon_per_cluster):
    abonents = np.zeros((N_obj*N_abon_per_cluster,2))
    for i_obj in range(N_obj):
        center = np.random.random(2)
        cov    = np.random.random((2,2))*0.1
        cov    = cov @ cov.T
        xs, ys = np.random.multivariate_normal(center, cov, N_abon_per_cluster).
T
        abonents[i_obj*N_abon_per_cluster:(i_obj+1)*N_abon_per_cluster, 0] = xs
        abonents[i_obj*N_abon_per_cluster:(i_obj+1)*N_abon_per_cluster, 1] = ys
    return abonents

def plot_problem(abonents):
    plt.figure(figsize=(10,6))
    plt.plot(abonents[:,0], abonents[:,1], 'go')
    plt.title('The village')
#     plt.savefig('bs_problem.svg')
    plt.show()

def random_solution(abonents, N_solutions = 100):
    x_min, x_max = abonents[:,0].min(), abonents[:,0].max()
    y_min, y_max = abonents[:,1].min(), abonents[:,1].max()
    population = []

    for i_sol in range(N_solutions):
        N_stations = int(np.random.random(1)[0]*10)+1
        stations = np.zeros((N_stations,2))
        stations[:,0], stations[:,1] = np.random.random(N_stations)*(x_max - x_m
in), np.random.random(N_stations)*(y_max - y_min)
        population.append(stations)
    return population

def find_nearest_station(dist_matrix):
    return np.argmin(dist_matrix, axis=1)

def pairwise_distance(abonents, stations):
    return cdist(abonents, stations)

def evaluate_generation(abonents, population, station_cost = 1, abonent_profit_b
ase = 1):
    costs = []
    for creature in population:
        N_stations, N_users = len(creature), len(abonents)
        total_cost          = N_stations*station_cost
        dist_matrix         = pairwise_distance(abonents, creature)
        stations_assignment = find_nearest_station(dist_matrix)
        stations_load       = np.ones(N_stations)
```

```
        stations_load       = np.array([1/(sum(stations_assignment == i_st)+1) f
or i_st, st in enumerate(stations_load)])

        for i_ab, abonent in enumerate(abonents):
            dist_to_base = dist_matrix[i_ab, stations_assignment[i_ab]]
            total_cost  -= stations_load[stations_assignment[i_ab]]*abonent_prof
it_base/(max(0.1, dist_to_base))

        costs.append(total_cost)
    return np.array(costs)

def mutation(population, mutation_rate = 0.3):
    N_creatures = len(population)
    x_min, x_max = -1, 1
    y_min, y_max = -1, 1
    mutated_creatures = sample(range(N_creatures), int(mutation_rate*N_creatures
))
    for i_mut in mutated_creatures:
        N_stations = len(population[i_mut])
        mutated_stations = sample(range(N_stations), int(mutation_rate*N_station
s))
        for i_st_mut in mutated_stations:
            population[i_mut][i_st_mut] += np.random.normal(0, 0.01, 2)

    N_new_stations = max(1, int(random.random()*mutation_rate*N_creatures))
    for i in range(N_new_stations):
        new_stations = np.zeros((N_new_stations,2))
        new_stations[:,0], new_stations[:,1] = np.random.random(N_new_stations)*
(x_max - x_min), np.random.random(N_new_stations)*(y_max - y_min)
        population.append(new_stations)
    return population

def children_creation(parent1, parent2):
    # whoisbatya
    batya = random.random() > 0.5
    if batya:
        child = np.concatenate((parent1[::2], parent2[1::2]))
    else:
        child = np.concatenate((parent1[1::2], parent2[::2]))
    return np.array(child)

def breed(population):
    new_population = deepcopy(population)
    random.shuffle(new_population)
    N_creatures = len(population)
    for i in range(N_creatures//2):
        children = children_creation(population[i], population[i+1])
        new_population.append(children)
    return new_population

def selection(abonents, population, offsprings = 10):
    scores = evaluate_generation(abonents, population)
    best = np.array(scores).argsort()[:offsprings].tolist()
    return [population[i_b] for i_b in best], population[best[0]]


def let_eat_bee(N_creatures, N_generations, N_obj = 10, N_abon_per_cluster = 10
):
    abonents = generate_problem(N_obj, N_abon_per_cluster)

    costs_evolution = np.zeros((N_generations, N_creatures))
```

```
    population = random_solution(abonents, N_creatures)
    best_creatures = []
    for generation in range(N_generations):
        population                = mutation(population)
        population                = breed(population)
        population, best_creature = selection(abonents, population, N_creatures)
        best_creatures.append(best_creature)

        costs_evolution[generation, :] = evaluate_generation(abonents, populatio
n)

        # Plotting
        x_min, x_max = 0, 1
        y_min, y_max = 0, 1
        cost_min  = [np.min(costs_evolution[i])  for i in range(generation)]
        cost_max  = [np.max(costs_evolution[i])  for i in range(generation)]
        cost_mean = [np.mean(costs_evolution[i]) for i in range(generation)]

        fig = make_subplots(rows=1, cols=2, subplot_titles=("Topology of the bes
t solution", "Cost function"))
        fig.update_xaxes(title_text="x", range = [x_min,x_max],  row=1, col=1)
        fig.update_yaxes(title_text="y", range = [y_min,y_max], row=1, col=1)
        fig.update_yaxes(title_text="Total cost", row=1, col=2)
        fig.update_xaxes(title_text="Generation", row=1, col=2)

        fig.add_trace(
            go.Scatter(x=abonents[:, 0], y=abonents[:, 1], mode='markers', name=
'abonents',  marker=dict(size=5)),
            row=1, col=1
        )

        fig.add_trace(
            go.Scatter(x=best_creatures[generation][:, 0], y=best_creatures[gene
ration][:, 1], mode='markers', name='stations', marker=dict(size=15)),
            row=1, col=1
        )

        fig.add_trace(
            go.Scatter(x = list(range(generation)), y = cost_min, name='best'),
            row=1, col=2
        )

        fig.add_trace(
            go.Scatter(x = list(range(generation)), y = cost_max, name='worst'),
            row=1, col=2
        )

        fig.add_trace(
            go.Scatter(x = list(range(generation)), y = cost_mean, name='mean'),
            row=1, col=2
        )

        clear_output(wait=True)
        fig.show()

    fig.write_html("test.html")
    return costs_evolution, abonents, best_creatures


costs_evolution, abonents, best_creatures = let_eat_bee(200, 5)
```

Для демонстрации работы покажите запуск на 200 особях в течение 200 поколений

Обратите внимание, что, изменяя стоиомость постройки станций и дефолтную прибыль от абонента можно прийти к странным экстремальным решениям (например, у каждого абонента по базовой станции). Поэтому фокусируйтесь больше на левую часть картинки и на ощущение того, что предложенное решение - норм.

Мое решение

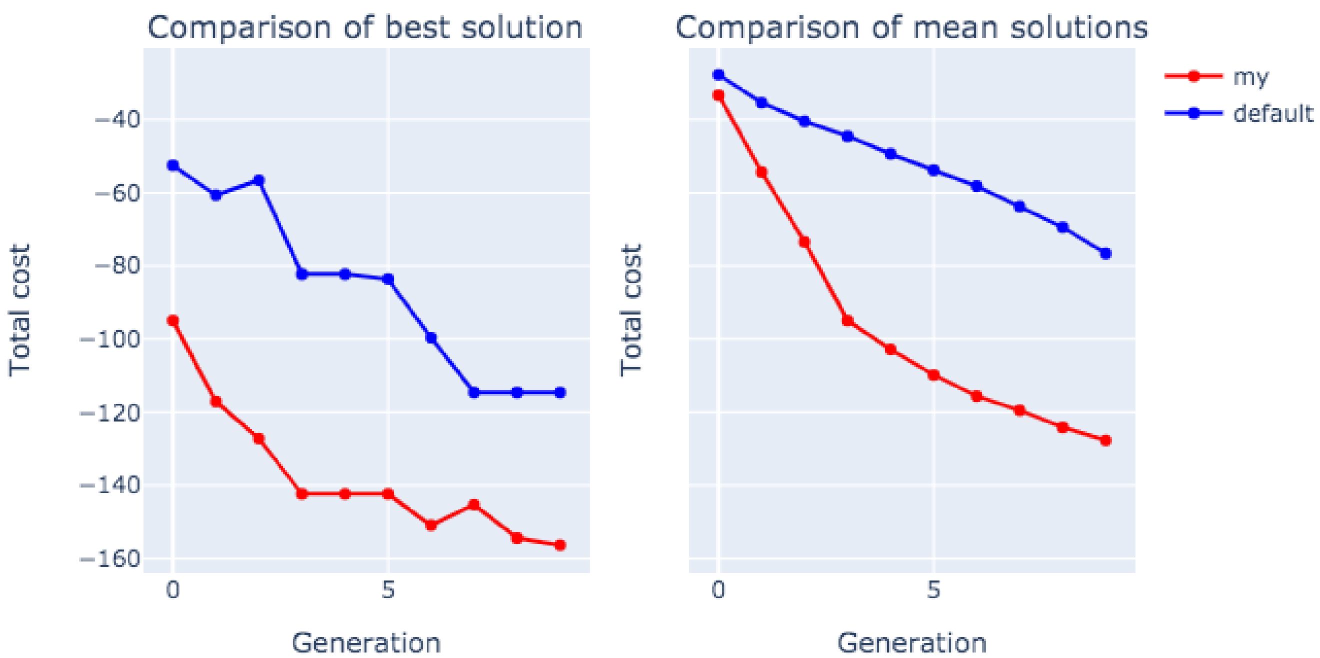
Я считаю, что функция потерь `evaluate_generation()` не подлежит изменениям, потому что это единственный способ сравнить базовую модель с моделью, в которую я внес свои изменения.

Кроме того, с самого начала я зафиксировал одну деревню `test_abonents`, на которой я буду проводить все тесты.

Предлагаемые изменения и результаты

1. Я изменил координаты области, в которой создаются новые случайные станции при мутации. В базовой модели они могут появляться очень далеко он деревни, а я ограничил их внутри деревни.

   Оказалось, что только такое исправление дало сильное улучшение работы. Ниже показаны графики сравнения базового алгоритма без и с этим исправлением на 10 поколениях:



1. Я добавил больше параметров мутации:

   - `mutation_rate_population` - доля популяции, которая мутирует
   - `mutation_rate_creature` - доля мутирующих станций в одном наборе

- `mutation_degree` - стандартое отклонение шума, который добавляется к координатам мутирующих станций
- `new_creatures` - процент, на который увеличивается популяция из-за добавления новых случайных наборов

По этим параметрам я сделал поиск по сетке, чтобы найти более-менее оптимальные. Каждую из моделей я запускал на трех одинаковых начальных популяциях (чтобы они были в равных условиях), на каждом из трех тестов и считал функцию потерь лучшей и средней моделей, а потом усреднил по трем тестам.

Изменения в мутации отражены в функции `my_mutation()` ниже.

Здесь представлены результаты поиска по сетке (отсортированы по значению функции потерь на лучшей особи).

	mut_rate_generation	mut_rate_creature	mut_degree	new_creatures	best_creature_mean	best_creature_std	mean_creature_mean	mean_creature_std
200	0.533333	0.600000	0.010	0.5	-170.736006	4.518138	-148.937088	3.008784
74	0.366667	0.200000	0.010	0.5	-170.015171	9.142333	-149.376790	6.359336
110	0.366667	0.466667	0.010	0.5	-169.727536	3.069992	-148.134605	2.831094
239	0.700000	0.333333	0.048	0.5	-169.305000	5.151040	-139.675642	2.796721
131	0.366667	0.600000	0.048	0.5	-167.433377	3.191255	-139.966362	4.317534
254	0.700000	0.466667	0.010	0.5	-167.291202	16.978232	-138.809013	12.742591
182	0.533333	0.466667	0.010	0.5	-166.822988	3.599356	-138.044840	4.668414
203	0.533333	0.600000	0.048	0.5	-166.487820	10.739020	-136.026805	6.951639
20	0.200000	0.333333	0.010	0.5	-164.562116	7.583180	-138.950519	6.946191
149	0.533333	0.200000	0.048	0.5	-163.949090	4.150271	-141.925851	5.315633

В качестве оптимальной я выберу модель с индексом 110, ее лучшая особь и средняя особь почти совпадают с наилучшими, но у нее меньше дисперсия, то есть меньше вероятность, что ей повезло.

1. В базовой модели при скрещивании двух особей, ребенок получает четные станции от одного родителя, а нечетные - от другого. При этом когда этот ребенок скрещивается дальше, он отдает своему потомку либо только четные, либо только нечетные станции. Проблема в том, что потомок ребенка будет иметь "гены" **только одного** из прародителей.

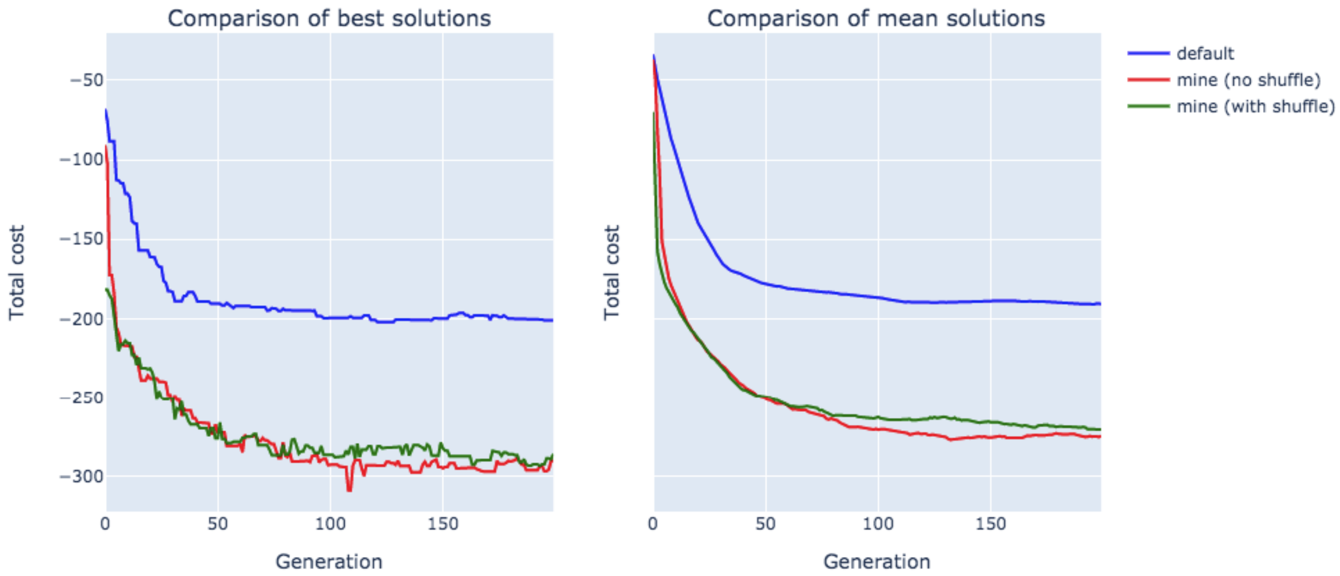
   Предложение: сразу после скрещивания перемешивать станции одной особи.

   Изменения в скрещивании отражены в функции `my_breed()` ниже.

Итоговые результаты

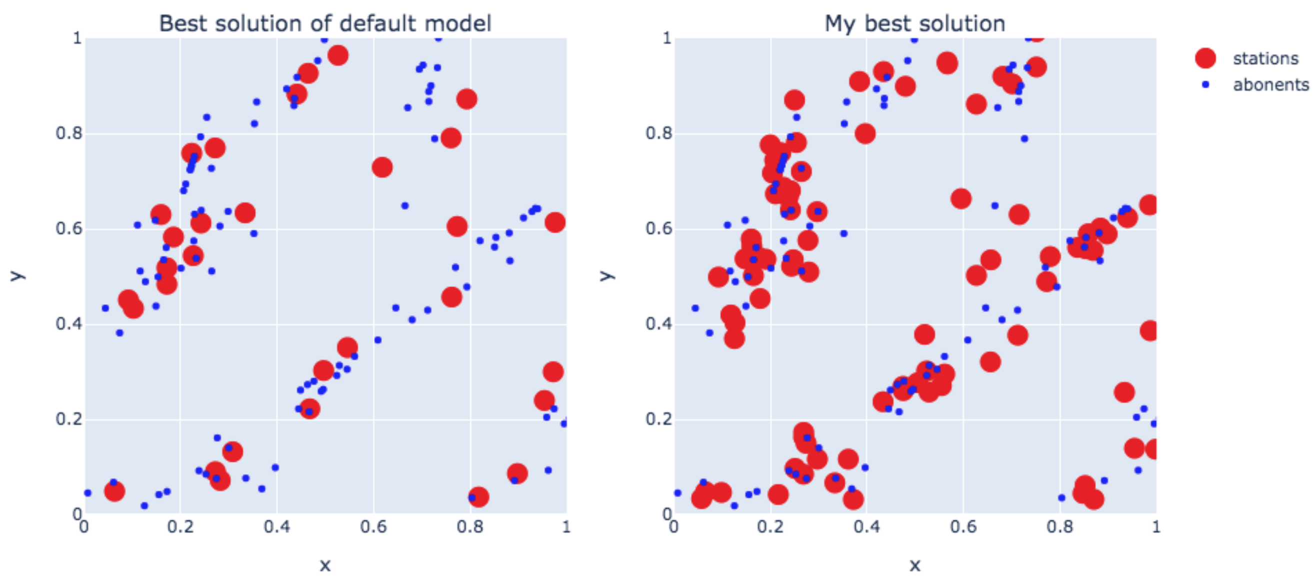
Проверку проводил на новой деревне.

Результаты работы на 200 особях в течение 200 поколений.



Видно, что стратегия перемешивать станции детей дает некоторый выигрыш на начальных поколениях, поэтому если нужно быстро получить неплохое решение, то ее можно использовать. Но на старших поколениях, видимо, это особой роли не играет.

Сравним лучшие распложения станций базового алгоритма и моей модификации (лучшая особь была на 108 поколении, на графике есть заметный пик).



Далее остался весь код и вспомогательные ячейки, их можно не смотреть, все самое информативное я вынес в этот блок.

In [122]:

```
test_abonents = generate_problem(10, 10)
start_population = random_solution(test_abonents, 200)
```

In [112]:

```
def plot_abonents_stations(abonents, creature):
    fig = make_subplots(rows=1, cols=1, subplot_titles=("The village"))
    fig.update_xaxes(title_text="x", range = [0,1], row=1, col=1)
    fig.update_yaxes(title_text="y", range = [0,1], row=1, col=1)
    fig.add_trace(
            go.Scatter(x=abonents[:, 0], y=abonents[:, 1], mode='markers', name=
'abonents',  marker=dict(size=5)),
            row=1, col=1
    )
    fig.add_trace(
            go.Scatter(x=creature[:, 0], y=creature[:, 1], mode='markers', name=
'stations', marker=dict(size=15)),
            row=1, col=1
        )
    fig.show()
```

In [142]:

```
def my_mutation(population, mutation_rate_population = 0.5, mutation_rate_creatu
re = 0.3,
                mutation_degree = 0.05, new_creatures = 0.2):
    # mutation_rate_population - fraction of population that mutates
    # mutation_rate_creature - fraction of stations that mutates
    # mutation_degree - variance of gaussian noise added in mutation
    # new_creatures - fraction of population added as random creatures

    N_creatures = len(population)
    mutated_creatures = sample(range(N_creatures), int(mutation_rate_population*
N_creatures))
    for i_mut in mutated_creatures:
        N_stations = len(population[i_mut])
        mutated_stations = sample(range(N_stations), int(mutation_rate_creature*
N_stations))
        for i_st_mut in mutated_stations:
            population[i_mut][i_st_mut] += np.random.normal(0, mutation_degree,
2)

    N_new_stations = max(1, int(random.random()*new_creatures*N_creatures))
    for i in range(N_new_stations):
        new_stations = np.zeros((N_new_stations,2))
        new_stations[:,0], new_stations[:,1] = np.random.random(N_new_stations),
np.random.random(N_new_stations)
        population.append(new_stations)
    return population

def my_breed(population, shuffle_children=False):
    new_population = deepcopy(population)
    random.shuffle(new_population)
    N_creatures = len(population)
    for i in range(N_creatures//2):
        children = children_creation(population[i], population[i+1])
        if shuffle_children:
            np.random.shuffle(children)
        new_population.append(children)


    return new_population

def my_selection(abonents, population, offsprings = 10):
    scores = evaluate_generation(abonents, population)
    best = np.array(scores).argsort()[:offsprings].tolist()
    return [population[i_b] for i_b in best], population[best[0]]

def run_default(abonents, start_population, N_creatures, N_generations):
    costs_evolution = np.zeros((N_generations, N_creatures))
    population = deepcopy(start_population)
    best_creatures = []

    for generation in range(N_generations):

        if (generation+1) % 5 == 0:
            print("Generation ",generation+1)

        population                 = mutation(population)
        population                 = breed(population)
        population, best_creature = selection(abonents, population, N_creatures)
        best_creatures.append(best_creature)
```

```
        costs_evolution[generation, :] = evaluate_generation(abonents, populatio
n)

    return costs_evolution, best_creatures


def my_run(abonents, start_population, N_creatures, N_generations,
           mut1=0.5, mut2=0.3, mut3=0.05, mut4=0.2, shuffle_children=False):

    my_costs_evolution = np.zeros((N_generations, N_creatures))
    my_population = deepcopy(start_population)
    my_best_creatures = []

    for generation in range(N_generations):

        if (generation+1) % 5 == 0:
            print("Generation ",generation+1)

        my_population                   = my_mutation(my_population, mut1, mut2,
mut3, mut4)
        my_population                   = my_breed(my_population, shuffle_childr
en)
        my_population, my_best_creature = my_selection(abonents, my_population,
N_creatures)
        my_best_creatures.append(my_best_creature)

        my_costs_evolution[generation, :] = evaluate_generation(abonents, my_pop
ulation)

    return my_costs_evolution, my_best_creatures
```

In [ ]:

```
default_costs, default_bests = run_default(test_abonents, start_population, 200,
200)
```

In [ ]:

```
my_costs, my_bests = my_run(test_abonents, start_population, 200, 200, 0.35, 0.4
7, 0.01, 0.5, False)
```

In [ ]:

```
my2_costs, my2_bests = my_run(test_abonents, start_population, 200, 200, 0.35,
0.47, 0.01, 0.5, True)
```

In [ ]:

```
N_gens = 200

default_min = [np.min(default_costs[i]) for i in range(N_gens)]
default_mean = [np.mean(default_costs[i]) for i in range(N_gens)]

my_min = [np.min(my_costs[i]) for i in range(N_gens)]
my_mean = [np.mean(my_costs[i]) for i in range(N_gens)]

my2_min = [np.min(my2_costs[i]) for i in range(N_gens)]
my2_mean = [np.mean(my2_costs[i]) for i in range(N_gens)]

fig = make_subplots(rows=1, cols=2, shared_yaxes=True,
                    subplot_titles=("Comparison of best solutions", "Comparison
 of mean solutions"))
fig.update_yaxes(title_text="Total cost", row=1, col=1)
fig.update_xaxes(title_text="Generation", row=1, col=1)
fig.update_yaxes(title_text="Total cost", row=1, col=2)
fig.update_xaxes(title_text="Generation", row=1, col=2)


fig.add_trace(
    go.Scatter(x = list(range(N_gens)), y = default_min, line=dict(color="blue"
), name="default"),
    row=1, col=1
)

fig.add_trace(
    go.Scatter(x = list(range(N_gens)), y = my_min, line=dict(color="red"), name
="mine (no shuffle)"),
    row=1, col=1
)

fig.add_trace(
    go.Scatter(x = list(range(N_gens)), y = my2_min, line=dict(color="green"), n
ame="mine (with shuffle)"),
    row=1, col=1
)

fig.add_trace(
    go.Scatter(x = list(range(N_gens)), y = default_mean, line=dict(color="blue"
), showlegend=False),
    row=1, col=2
)

fig.add_trace(
    go.Scatter(x = list(range(N_gens)), y = my_mean, line=dict(color="red"), sho
wlegend=False),
    row=1, col=2
)

fig.add_trace(
    go.Scatter(x = list(range(N_gens)), y = my2_mean, line=dict(color="green"),
showlegend=False),
    row=1, col=2
)
```

```
clear_output(wait=True)
fig.show()
```

In [ ]:

```
fig = make_subplots(rows=1, cols=2, subplot_titles=("Best solution of default mo
del", "My best solution"))
fig.update_xaxes(title_text="x", range = [0,1],  row=1, col=1)
fig.update_yaxes(title_text="y", range = [0,1], row=1, col=1)
fig.update_xaxes(title_text="x", range = [0,1],  row=1, col=2)
fig.update_yaxes(title_text="y", range = [0,1], row=1, col=2)

fig.add_trace(
    go.Scatter(x=default_bests[199][:, 0], y=default_bests[199][:, 1], mode='mar
kers', name='stations',
               marker=dict(size=15, color='red')),
    row=1, col=1
)
fig.add_trace(
    go.Scatter(x=test_abonents[:, 0], y=test_abonents[:, 1], mode='markers', nam
e='abonents',
               marker=dict(size=5, color='blue')),
    row=1, col=1
)

fig.add_trace(
    go.Scatter(x=my_bests[108][:, 0], y=my_bests[108][:, 1], mode='markers', sho
wlegend=False,
               marker=dict(size=15, color='red')),
    row=1, col=2
)
fig.add_trace(
    go.Scatter(x=test_abonents[:, 0], y=test_abonents[:, 1], mode='markers', sho
wlegend=False,
               marker=dict(size=5, color='blue')),
    row=1, col=2
)
fig.show()
```

In [73]:

```
def test_algoritms(abonents, N_creatures, N_generations, mut1_list, mut2_list, m
ut3_list, mut4_list, n_tests=3):
    results = []
    start_population = [random_solution(abonents, N_creatures) for i in range(n_
tests)]
    N = -1

    for m1 in mut1_list:
        for m2 in mut2_list:
            for m3 in mut3_list:
                for m4 in mut4_list:

                    N += 1

                    cost_min = []
                    cost_mean = []

                    for i in range(n_tests):

                        print("Iteration number ", N, "-", i)
                        my_population = deepcopy(start_population[i])

                        for gen in range(N_generations):
                            my_population                     = my_mutation(my_pop
ulation, m1, m2, m3, m4)
                            my_population                     = my_breed(my_popula
tion)
                            my_population, my_best_creature = my_selection(abone
nts, my_population, N_creatures)

                        res = evaluate_generation(abonents, my_population)
                        cost_min.append(np.min(res))
                        cost_mean.append(np.mean(res))

                    cost_min = np.array(cost_min)
                    cost_mean = np.array(cost_mean)

                    results.append([m1, m2, m3, m4, np.mean(cost_min), np.std(co
st_min), np.mean(cost_mean), np.std(cost_mean)])

    return results
```

In [ ]:

```
mut1_list = list(np.linspace(0.2, 0.7, 4))
mut2_list = list(np.linspace(0.2, 0.6, 4))
mut3_list = list(np.linspace(0.01, 0.2, 6))
mut4_list = list(np.linspace(0.3, 0.5, 3))

results = test_algoritms(test_abonents, 100, 10, mut1_list, mut2_list, mut3_list
, mut4_list, 3)
#print(results)
```

In [96]:

```
results_ = deepcopy(results)
```

In [ ]:

```
results = np.array(results)
print(results.shape)
idx = np.argsort(results[:,4])[:10]
idx2 = np.argsort(results[:,6])[:10]
print(idx)
print(idx2)
```

In [ ]:

```
import pandas as pd
frame = pd.DataFrame(results)
frame.columns = ['mut_rate_generation', 'mut_rate_creature', 'mut_degree', 'new_
creatures',
                 'best_creature_mean', 'best_creature_std', 'mean_creature_mean'
, 'mean_creature_std']
frame.iloc[idx]
```

Gradient descent

Problem 4

Говорят, что функция принадлежит классу $f \in C_L^{k,p}(Q)$, если она k раз непрерывно дифференцируема на Q, а p-ая производная имеет константу Липшица L.

$$\|\nabla^p f(x) - \nabla^p f(y)\| \leq L\|x - y\|, \qquad \forall x, y \in Q$$

Чаще всего используются $C_L^{1,1}, C_L^{2,2}$ на $\mathbb{R}^n$. Заметим, что:

- $p \leq k$
- Если $q \geq k$, то $C_L^{q,p} \subseteq C_L^{k,p}$. Чем выше порядок производной, тем сильнее ограничение (меньшее количество функций принадлежат классу)

Докажите, что функция принадлежит к классу $C_L^{2,1} \subseteq C_L^{1,1}$ тогда и только тогда, когда $\forall x \in \mathbb{R}^n$:

$$\|\nabla^2 f(x)\| \leq L$$

Докажите так же, что последнее условие можно без ограничения общности переписать в виде:

$$-LI_n \preceq \nabla^2 f(x) \preceq LI_n$$

Примечание: по умолчанию для векторов используется Евклидова норма, а для матриц - спектральная

Решение:

1. Сначала заметим, что если функция дважды непрерывно дифференцируема, то ее смешанные производные не зависят от порядка дифференцирования, то есть гессиан $\nabla^2 f(x)$ является симметричной матрицей в любой точке.

   В случае симметричной матрицы спектральная норма имеет вид:

$$\left\|\nabla^2 f(x)\right\| = \max_i |\lambda_i|$$

   Пользуясь тем, что матрица положительно полуопределена тогда и только тогда, когда все е е собственные числа неотрицательны, получаем

$$\left\|\nabla^2 f(x)\right\| \leq L \qquad \Longleftrightarrow \qquad -LI \preceq \nabla^2 f(x) \preceq LI$$

1. Пусть $\|\nabla^2 f(x)\| \leq L$. Для любых x, y по формуле Тейлора:
$$\nabla f(y) = \nabla f(x) + \nabla^2 f(\xi)(y-x), \qquad \xi \in [x, y]$$
   Тогда
$$\left\|\nabla f(x) - \nabla f(y)\right\| = \left\|\nabla^2 f(\xi)(y-x)\right\| \leq \left\|\nabla^2 f(\xi)\right\| \cdot \|x-y\| \leq L\|x-y\| \qquad \Longrightarrow \qquad f \in C_L^{2,1}$$

1. Допустим, что в какой-то точке $\left\|\nabla^2 f(x)\right\| > L$. Значит, какое-то собственное значение $\nabla^2 f(x)$ по модулю больше L. Для определенности, пусть оно положительно: $\lambda > L$. Пусть h - соответствующий собственный вектор единичной длины. Тогда при $t \neq 0$ по формуле Тейлора:

$$\nabla f(x+th) - \nabla f(x) = \nabla^2 f(x)th + o(t) = \lambda th + t \cdot r(t), \qquad \|r(t)\| = o(1)$$

$$\frac{\left\|\nabla f(x+th) - \nabla f(x)\right\|}{\|x+th-x\|} = \|\lambda h + r(t)\| \rightarrow \lambda > L \qquad \text{при} \qquad t \rightarrow 0$$

   При достаточно малых t неравенство реализуется, поэтому $f \notin C_L^{2,1}$.

Problem 5

Покажите, что с помощью следующих стратегий подбора шага в градиентному спуске:

- Постоянный шаг $h_k = \dfrac{1}{L}$
- Убывающая последовательность $h_k = \dfrac{\alpha_k}{L}, \quad \alpha_k \rightarrow 0$

можно получить оценку убывания функции на итерации вида:

$$f(x_k) - f(x_{k+1}) \geq \frac{\omega}{L}\|\nabla f(x_k)\|^2$$

$\omega > 0$ - некоторая константа, L - константа Липщица градиента функции

Решение:

1. Рассмотрим случай шага $h_k = \dfrac{1}{L}$.

   Для выпуклой функции с L-липшицевым градиентом справедлива оценка
   $$f(y) \le f(x) + \left\langle \nabla f(x), y - x \right\rangle + \frac{L}{2}\|y - x\|^2, \qquad \forall x, y \in \mathbb{R}^n$$

   Подставляя $y = x_{k+1},\ x = x_k$ и учитывая связь $x_{k+1} = x_k - \dfrac{1}{L}\nabla f(x_k)$, получаем
   $$f(x_k) - f(x_{k+1}) \ge \left\langle \nabla f(x_k), x_k - x_{k+1} \right\rangle - \frac{L}{2}\|x_k - x_{k+1}\|^2 = \frac{1}{2L}\left\|\nabla f(x_k)\right\|^2$$

   То есть требуемая оценка выполняется с константой $\omega = \dfrac{1}{2}$.
2. Рассмотрим случай шага $h_k = \dfrac{\alpha_k}{L},\ \ \alpha_k \to 0$.

   Возможно имелось в виду что-то другое, но здесь можно доказать, что такой константы ω не существует.

   Для выпуклой функции с L-липшицевым градиентом также справедлива оценка
   $$f(y) \ge f(x) + \left\langle \nabla f(x), y - x \right\rangle - \frac{L}{2}\|y - x\|^2, \qquad \forall x, y \in \mathbb{R}^n$$

   Подставляя $y = x_{k+1},\ x = x_k$ и учитывая связь $x_{k+1} = x_k - \dfrac{\alpha_k}{L}\nabla f(x_k)$, получаем
   $$f(x_k) - f(x_{k+1}) \le \left\langle \nabla f(x_k), x_k - x_{k+1} \right\rangle + \frac{L}{2}\|x_k - x_{k+1}\|^2 = \frac{2\alpha_k + \alpha_k^2}{2L}\left\|\nabla f(x_k)\right\|^2$$

   Мы хотим, чтобы одновременно было выполнено
   $$\frac{\omega}{L}\left\|\nabla f(x_k)\right\|^2 \le f(x_k) - f(x_{k+1}) \le \frac{2\alpha_k + \alpha_k^2}{2L}\left\|\nabla f(x_k)\right\|^2$$

   Отсюда следует, что
   $$2\omega \le 2\alpha_k + \alpha_k^2 \xrightarrow{k \to \infty} 0 \qquad \Longrightarrow \qquad w = 0$$

   Это противоречие, так как $\omega > 0$.

Problem 6

Рассмотрим функцию двух переменных:

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

где k - некоторый параметр

In [165]:

```
def f_6(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_6(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]])
```

In [166]:

```
%matplotlib inline
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter

def plot_3d_function(x1, x2, f, title, *f_params, minima = None, iterations = No
ne):
    '''
    '''
    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('$x_1$')
    ax.set_ylabel('$x_2$')

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

    if iterations is not None:
```

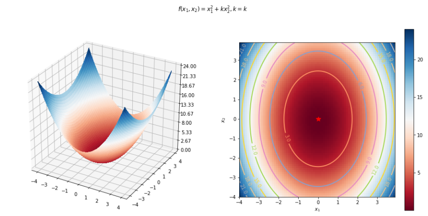
```
        for point in iterations:
            ax.plot(*point, 'go', markersize=3)
        iterations = np.array(iterations).T
        ax.quiver(iterations[0,:-1], iterations[1,:-1], iterations[0,1:]-iterati
ons[0,:-1], iterations[1,1:]-iterations[1,:-1], scale_units='xy', angles='xy', s
cale=1, color='blue')

    plt.show()
```

In [167]:

```
up_lim  = 11
low_lim = -4
x1 = np.arange(low_lim, up_lim, 0.1)
x2 = np.arange(low_lim, up_lim, 0.1)
k=0.5
#title = '$f(x_1, x_2) = x_1^2 + k x_2^2, k = {k}$'

#plot_3d_function(x1, x2, f_6, title, k, minima=[0,0])
```



Для наглядности можете пользоваться кодом отрисовки окружающих картинок

In [204]:

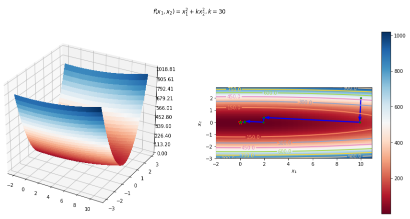
```
from scipy.optimize import minimize_scalar

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_i
ter:
        res = minimize_scalar(lambda alpha: f(x - alpha * df(x, *f_params), *f_p
arams))
        alpha_opt = res.x
        x = x - alpha_opt * df(x, *f_params)
        iterations.append(x)
    #print('Finished with', len(iterations), 'iterations')
    return iterations

x_0 = [10,2]
k = 100
iterations = steepest_descent(x_0, f_6, df_6, k, df_eps = 1e-7)
title = '$f(x_1, x_2) = x_1^2 + k x_2^2, k = 30$'


x1 = np.arange(-2, 11, 0.1)
x2 = np.arange(-3, 3, 0.1)

plot_3d_function(x1, x2, f_6, title, k, minima=[0,0], iterations = iterations)
```



Постройте график количества итераций, необходимых для сходимости алгоритма наискорейшего спуска (до выполнения условия $\|\nabla f(x_k)\| \leq \varepsilon = 10^{-7}$) в зависимости от значения k. Рассмотрите интервал $k \in [10^{-3}; 10^3]$ (будет удобно использовать функцию `ks = np.logspace(-3,3)`) и строить график по оси абсцисс в логарифмическом масштабе `plt.semilogx()` или `plt.loglog()` для двойного лог. масштаба.

Сделайте те же графики для функции:

$$f(x) = \ln(1 + e^{x^\top A x}) + \mathbf{1}^\top x$$

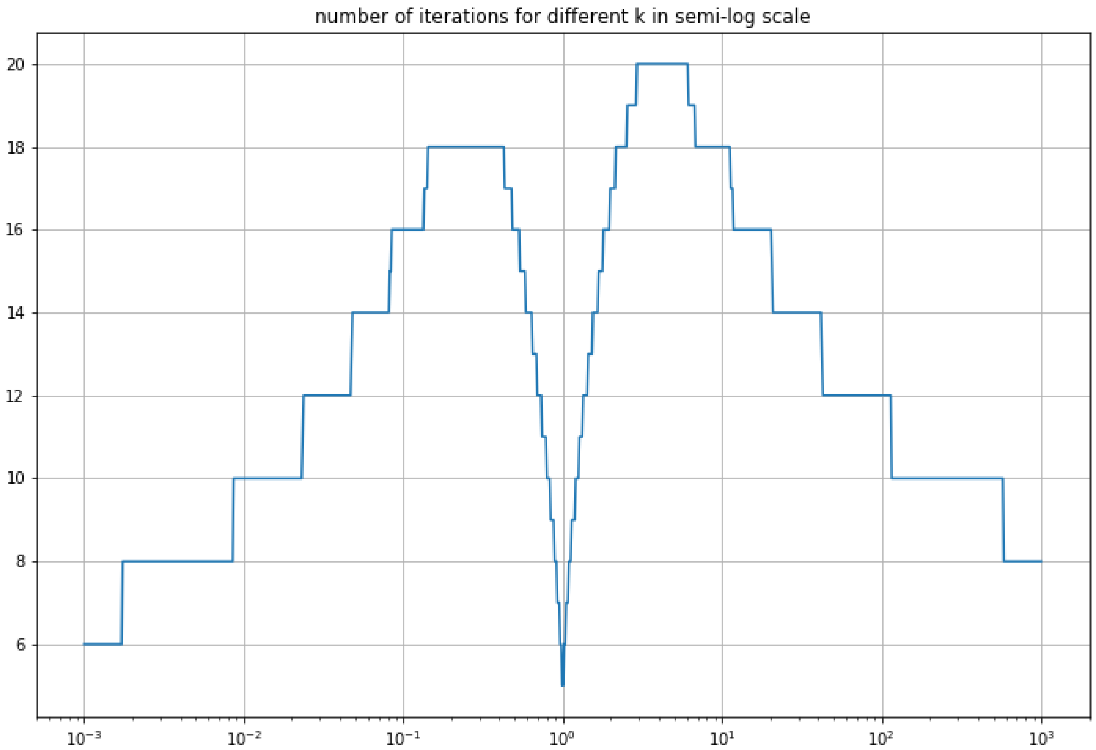
Объясните полученную зависимость.

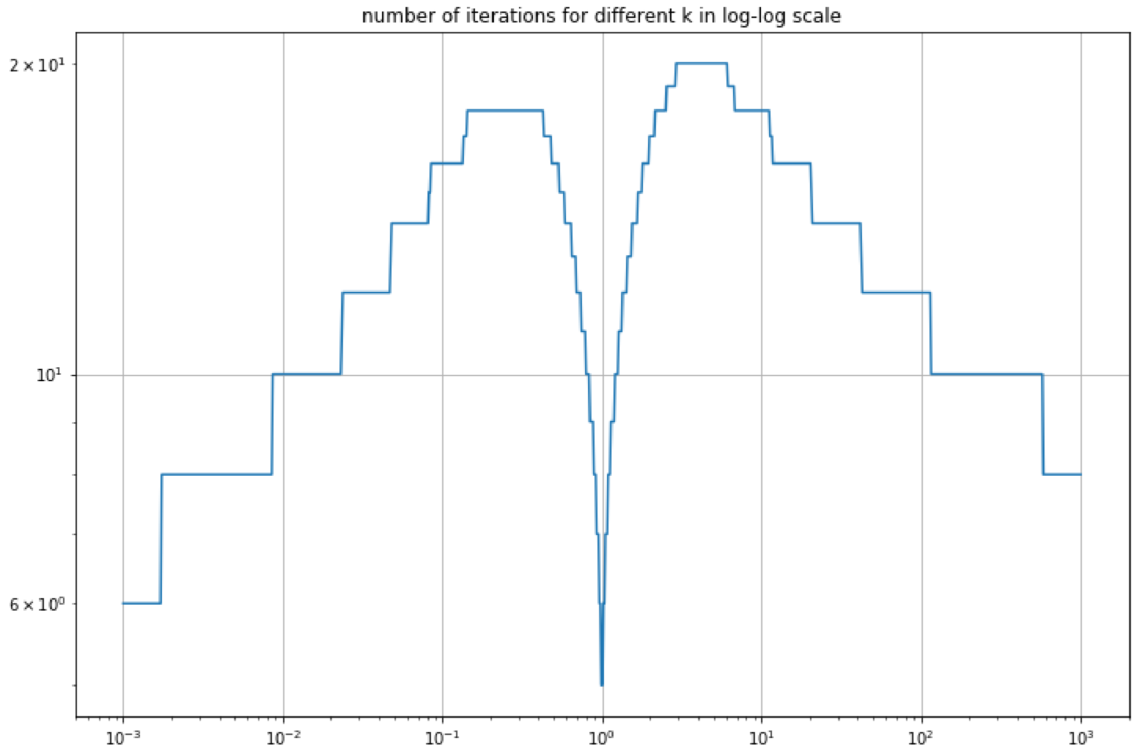
Решение

Первая функция

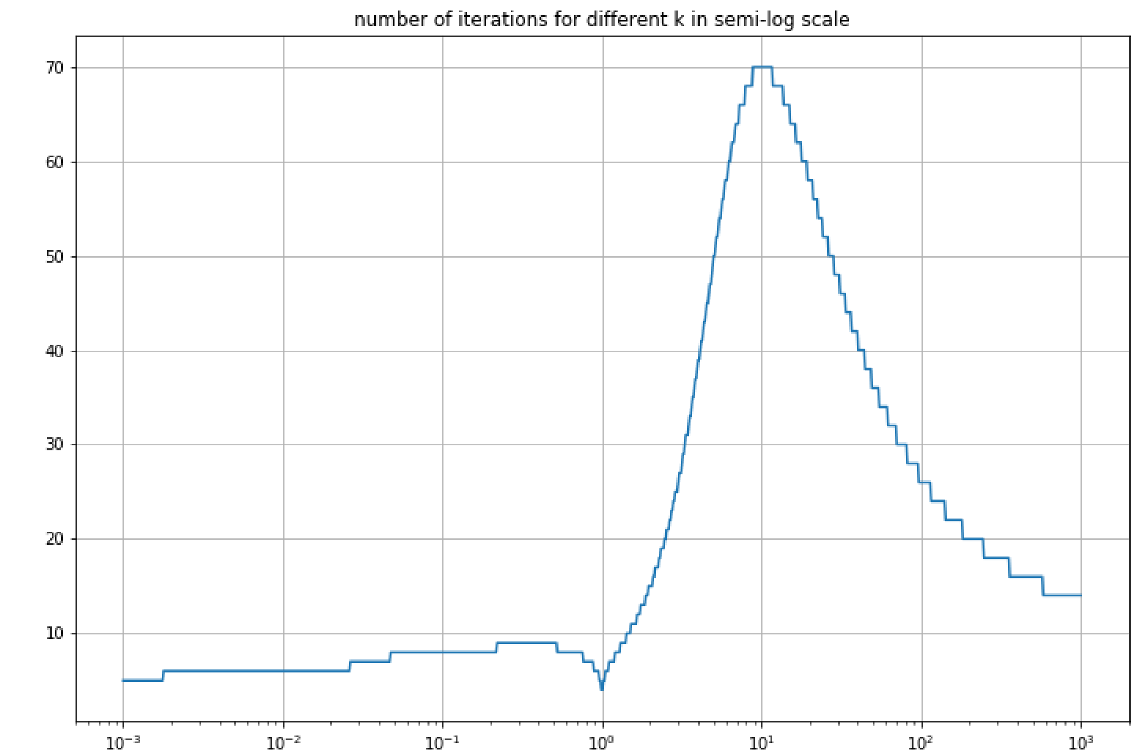
$$f(x_1, x_2) = x_1^2 + kx_2^2$$

Графики числа итераций сильно зависят от начальной точки. Если ее взять на биссектрисе $x_0 = (10, 10)$, то, например, случаи $k = 10$ и $k = 0.1$ будут примерно симметричными, поэтому в логарифмическом масштабе график будет симметричным относительно $k = 1$.

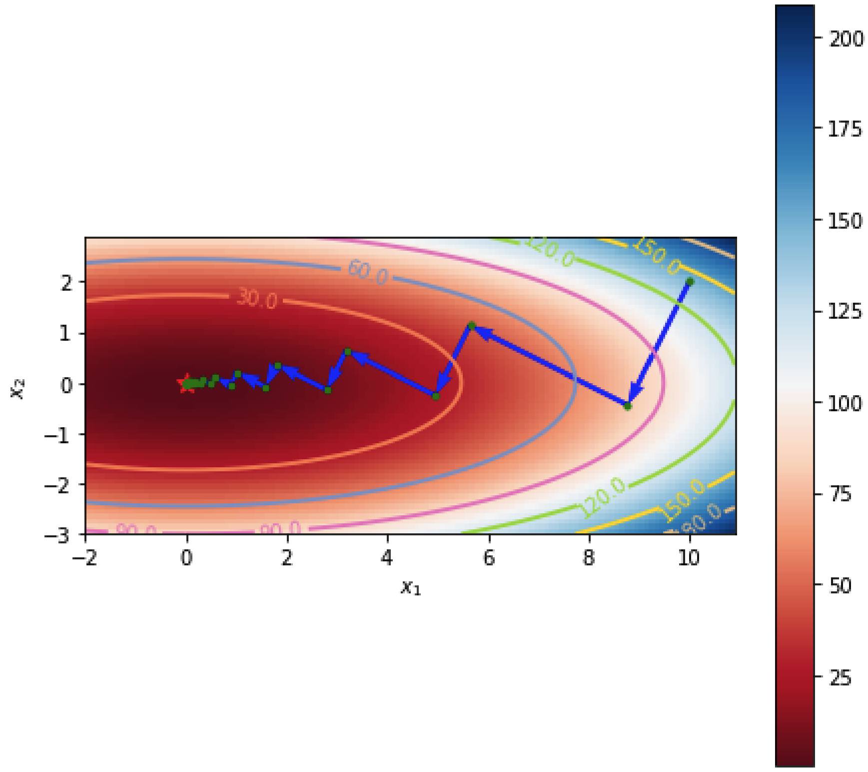




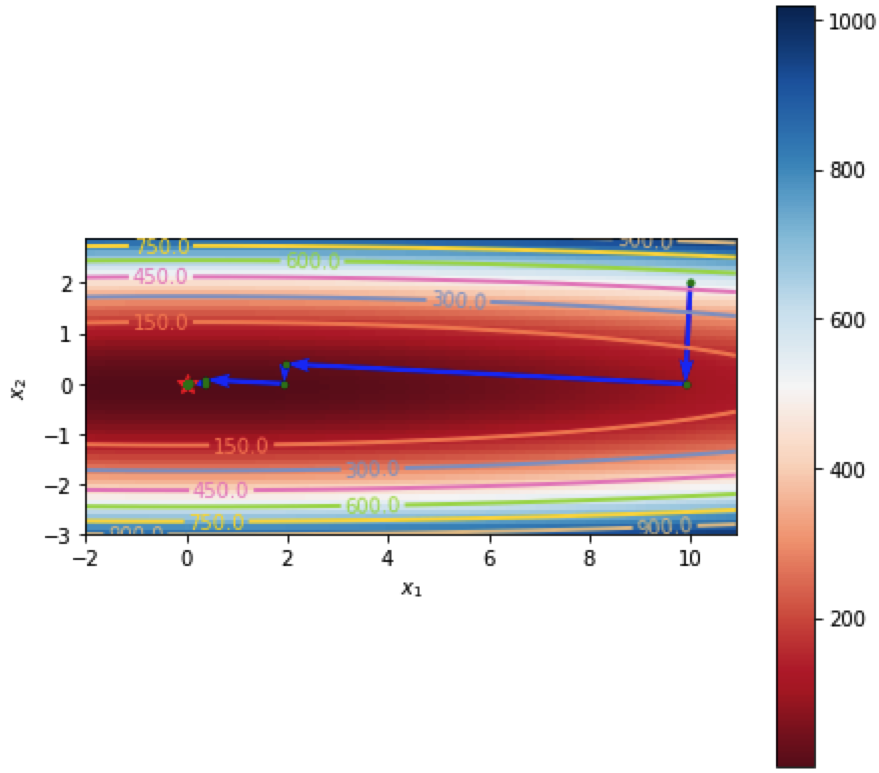
Если начальную точку сдвинуть ближе к оси X_1, чем оси X_2, например $x_0 = (10, 2)$, то получается



Объяснить это можно тем, что при как раз $k = 10$ мы идем практически вдоль большой горизонтальной оси эллипсов (линий уровня функции) маленькими шагами (градиенты направлены к осям под углами, сравнимыми с 45°).

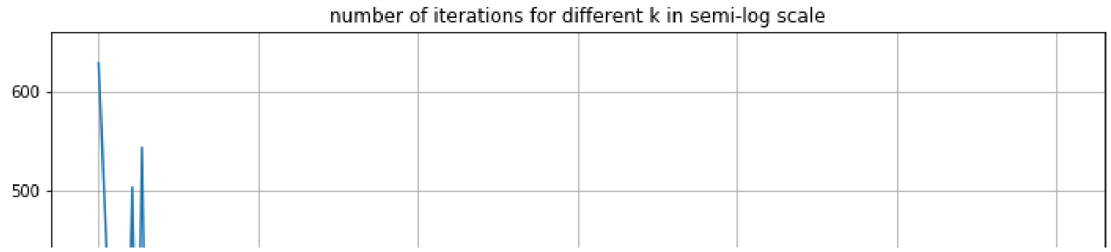


Если брать $k \gg 10$ или $k \ll 10$, то градиенты почти параллельны осям и мы сравнительно быстро сходимся. На графике ниже $k = 100$.



Вторая функция

$$g(x_1, x_2) = \ln\left(1 + e^{x_1^2 + kx_2^2}\right) + x_1 + x_2$$



In [ ]:

```
ks = np.logspace(-3,3,1000)
n_iter = []
x_0 = [10,2]

for k in ks:
    iters = steepest_descent(x_0, f_6, df_6, k, df_eps = 1e-7)
    n_iter.append(len(iters))

fig, ax = plt.subplots(figsize = (12,8))
ax.semilogx(ks, n_iter)
ax.set(title='number of iterations for different k in semi-log scale')
ax.grid()

fig.savefig("t6_right_semilog.png")
plt.show()
```

In [212]:

```
def g_6(x, *f_params):
    if len(f_params) == 0:
        k = 2
    else:
        k = float(f_params[0])
    x_1 = x[0]
    x_2 = x[1]
    return np.log(1 + np.exp(x_1*x_1 + k*x_2*x_2)) + x_1 + x_2

def dg_6(x, *f_params):
    if len(f_params) == 0:
        k = 2
    else:
        k = float(f_params[0])
    x_1 = x[0]
    x_2 = x[1]
    return np.array([(np.exp(x_1*x_1 + k*x_2*x_2)*2*x_1)/(1 + np.exp(x_1*x_1 + k
*x_2*x_2)) + 1,
                     (np.exp(x_1*x_1 + k*x_2*x_2)*2*k*x_2)/(1 + np.exp(x_1*x_1 +
k*x_2*x_2)) + 1])
```

In [ ]:

```
x_0 = [3,3]
k = 0.5
title = '$g(x_1, x_2), k = 2$'


x1 = np.arange(-7, 7, 0.1)
x2 = np.arange(30, 50, 0.1)

plot_3d_function(x1, x2, g_6, title, k)
```

In [ ]:

```
ks = np.logspace(-3,3,200)
n_iter = []
x_0 = [3.7,1]

for k in ks:
    iters = steepest_descent(x_0, g_6, dg_6, k, df_eps = 1e-7)
    n_iter.append(len(iters))
```

In [ ]:

```
n_iter = np.array(n_iter)
idx = n_iter < 995
ks = ks[idx]
n_iter = n_iter[idx]

fig, ax = plt.subplots(figsize = (12,8))
ax.semilogx(ks, n_iter)
ax.set(title='number of iterations for different k in semi-log scale')
ax.grid()

fig.savefig("t6_g_semilog.png")
plt.show()
```