Open in Colab
(https://colab.research.google.com/github/MMiknich/opt_2_newhope/blob/master/HW1_Melkumov_Michael.ipynb)

Homework 1. *Мелкумов Михаил*

- Домашку присылать в виде `.pdf` файла на адрес `homework@merkulov.top`. ## 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. $r_k = \begin{cases} \frac{1}{k^k}, & \text{if } k \text{ is even} \\ \frac{1}{k^{2k}}, & \text{if } k \text{ is odd} \end{cases}$

РЕШЕНИЕ ЕСТЬ

Let's follow [those defenitions (https://fmin.xyz/docs/methods/Methods/#speed-of-convergence)](https://fmin.xyz/docs/methods/Methods/#speed-of-convergence) 1) $r_k = \left\{(0.707)^k\right\}_{k=1}^{\infty}$

- $||r_k - r_*||_2 = 0.707^k \leq Cq^k \rightarrow$ so that's `linear` speed

2) $r_k = \left\{(0.707)^{2^k}\right\}_{k=1}^{\infty}$

- $||r_k - r_*||_2 = 0.707^{2^k} \leq Cq^{2^k} \rightarrow$ so that's `quadratic` speed

3) $r_k = \left\{\dfrac{1}{k^2}\right\}_{k=1}^{\infty}$

- $||r_k - r_*||_2 = k^{-2} \leq Ck^{-2} \rightarrow$ so that's `sublinear` speed

4) $r_k = \left\{\dfrac{1}{k!}\right\}_{k=1}^{\infty}$

- We know, that: $||\frac{1}{k!} - 0||_2 \geq \frac{1}{k^k} = e^{-k\ln k} \geq e^{-2^k \ln c} = (\frac{1}{c})^{2^k} = q^{2^k}$ т.к. $x\ln x$ медленней растет чем $c2^k \rightarrow$ so that's `quadratic` speed

5) $r_k = \begin{cases} \frac{1}{k}, & \text{if } k \text{ is even} \\ \frac{1}{k^2}, & \text{if } k \text{ is odd} \end{cases}$

- It's same as 3rd, well at least it's not faster
- Blah blah blah $||x_k - x^*|| \geq Ck^{-1} \rightarrow$ so that's `sublinear` speed

6)$r_k = \begin{cases} \frac{1}{k^k}, & \text{if } k \text{ is even} \\ \frac{1}{k^{2k}}, & \text{if } k \text{ is odd} \end{cases}$

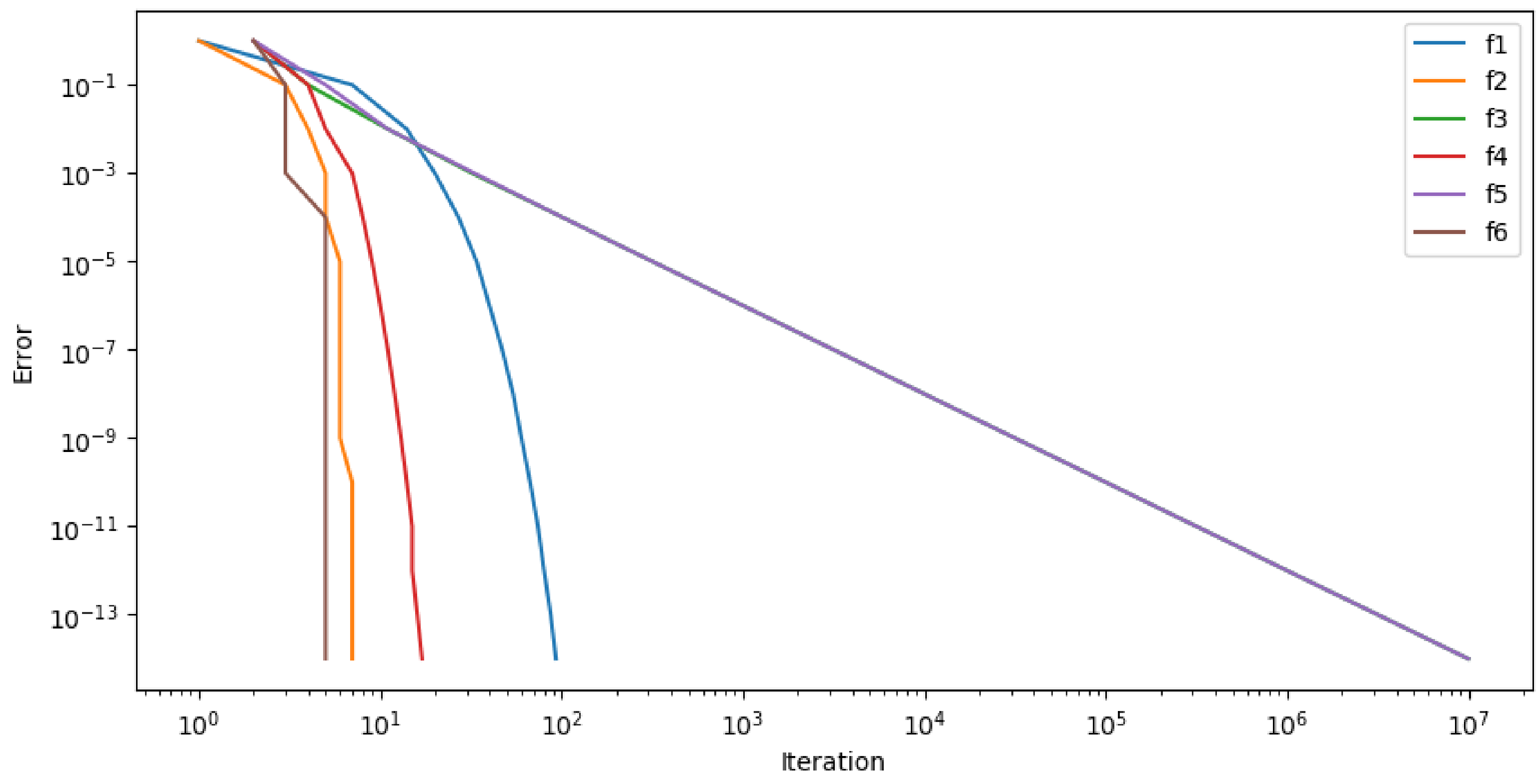
- this->solution == 3).solution
- idkn anything faster than `quadratic` , so it's `quadratic`

In [1]:
```python
#!pip install ipympl #it's for colab
%matplotlib inline
import numpy as np
import math
from scipy import optimize
from matplotlib import pyplot as plt
plt.rcParams['figure.dpi'] = 100
```

In [2]:
```python
a = np.ones((2, 2))
print(a)
b = np.array([1, 2])
print(a @ b)
def convergence_speed(f, lim, epsilon):
    N = 1
    while abs(f(N) - lim) >= epsilon:
        N+=1
    return N
def f1(k):
    return 0.707**k
def f2(k):
    return 0.707**(2**k)
def f3(k):
    return 1.0/(k**2)
def f4(k):
    return 1.0/(math.factorial(k))
def f5(k):
    if k%2==0:
        return 1.0/k
    else:
        return 1.0/(k**2)
def f6(k):
    if k%2==0:
        return 1.0/(k**k)
    else:
        return 1.0/(k**(2**k))
```

```
[[1. 1.]
 [1. 1.]]
[3. 3.]
```

In [3]:
```
x = [1, 1e-1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8, 1e-9, 1e-10, 1e-11, 1e-12, 1e-13, 1e-14]
#x1 = list(range(0,len(x)))

plt.figure(figsize=(10, 5))

y = []
for i in x:
    y.append(convergence_speed(f1, 0, i))
plt.plot(y, x, label = "f1")

y = []
for i in x:
    y.append(convergence_speed(f2, 0, i))
plt.plot(y, x, label = "f2")

y = []
for i in x:
    y.append(convergence_speed(f3, 0, i))
plt.plot(y, x, label = "f3")

y = []
for i in x:
    y.append(convergence_speed(f4, 0, i))
plt.plot(y, x, label = "f4")

y = []
for i in x:
    y.append(convergence_speed(f5, 0, i))
plt.plot(y, x, label = "f5")


y = []
for i in x:
    y.append(convergence_speed(f6, 0, i))
plt.plot(y, x, label = "f6")
plt.ylabel("Error")
plt.xlabel("Iteration")
plt.legend()
plt.xscale("log")
plt.yscale("log")
plt.style.use("fivethirtyeight")
plt.show()
```



Line search

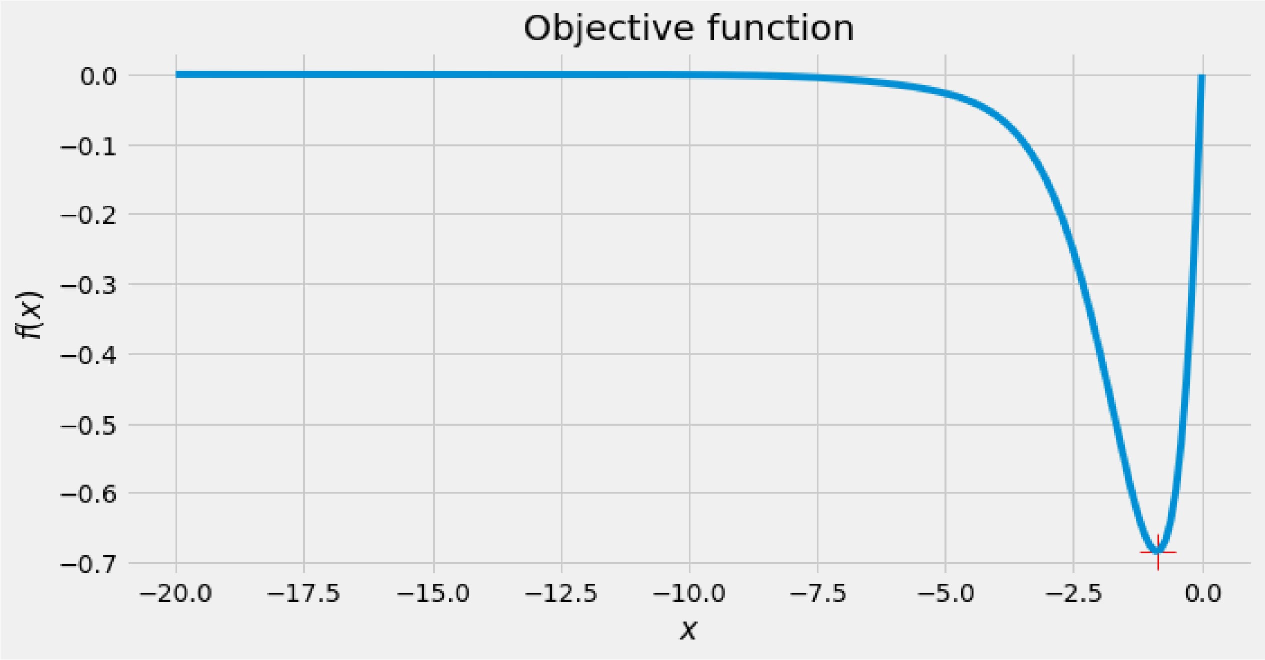
Problem 2

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 [4]:
```
def f(x):
    return  (x + np.sin(x))*np.exp(x)

x_star = optimize.minimize_scalar(f, bounds=((-20, 0)))
x = np.linspace(-20,0, 200)
plt.figure(figsize=(10, 5))
plt.plot(x_star["x"], x_star["fun"], "r+", ms=20)
plt.plot(x, f(x))
plt.title('Objective function')
plt.xlabel(f'$x$')
plt.ylabel(f'$f(x)$')
plt.style.use("fivethirtyeight")
plt.show()

print(x_star["x"], x_star["fun"])
```




```
-0.874357863611476 -0.6847142088028847
```

In [ ]:

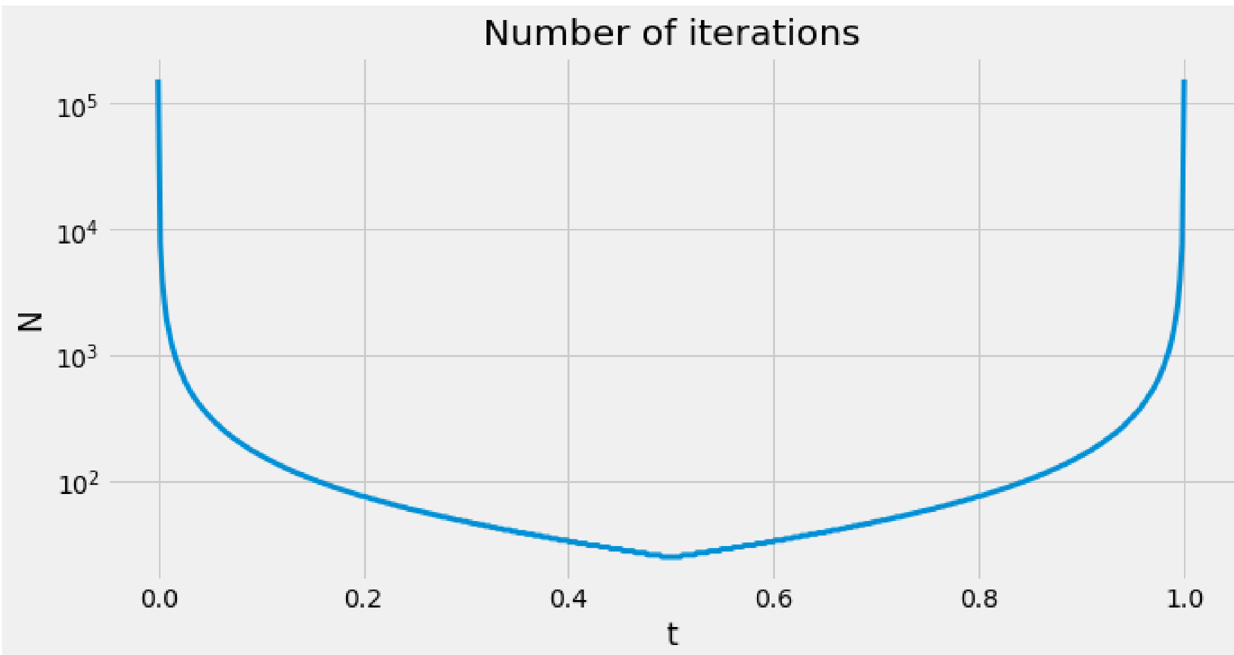
```
def x_t(a, b, t):
    return a + t*(b-a), b + t*(a-b)

def iter_procces(f, a, b, t, epsilon):
    N = 0
    if t == 0.5:
        return di_iter_procces(f, a, b, epsilon)
    while(abs(a-b)>=epsilon):
        N+=1
        x1, x2 = x_t(a, b, t)
        if(x1 > x2):
            x1, x2 = x2, x1
        f1 = f(x1)
        f2 = f(x2)

        if(f1 == f2):
            a = x1
            b = x2

        elif(f1 < f2):
            b = x2
        else:
            a = x1
    return a, b, N
def di_iter_procces(f, a, b, epsilon):
    N = 0
    c = (a+b)/2.0
    while(abs(a-b)>=epsilon):
        N+=1
        a, b, c = di_iter(f, a ,b, c)
    return a, b, N
def di_iter(f, a, b, c):
    x = (a+c)/2.0
    y = (c+b)/2.0
    if f(x)>f(c) and f(y)>f(c):
        return x, y, c
    elif f(x)>f(c) and f(y) < f(c):
        return c, b, y
    elif f(x)<f(c) and f(y)>f(c):
        return a, c, x
    else:
        raise NameError("Unimodality problem")
```

In [ ]:

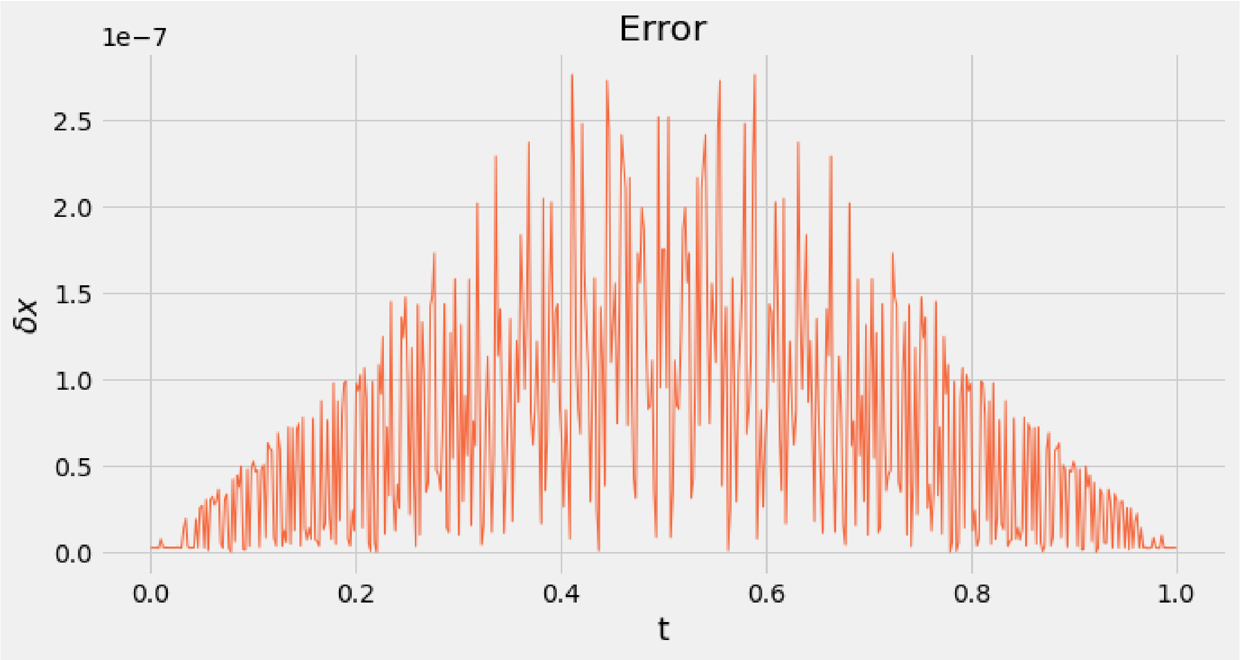
```
a = -20.0
b = 0.0
epsilon = 10.0**-6
x = np.linspace(0.0001,0.9999, 500)
err = np.zeros(x.shape)
y = np.zeros(x.shape)
for i in range(0, x.shape[0]):
    at, bt, y[i] = iter_procces(f, a, b, x[i], epsilon)
    err[i] = abs(x_star["x"] - (at + bt)/2.0)
```

In [7]:

```
plt.figure(figsize=(10, 5))
plt.plot(x, y, linewidth=3)
plt.yscale('log')
plt.title("Number of iterations")
plt.xlabel("t")
plt.ylabel("N")
plt.style.use("fivethirtyeight")
plt.show()
```



In [8]:
```
plt.figure(figsize=(10, 5))
plt.plot(x, err, linewidth = 1, color = "#f5683d")
plt.title('Error')
plt.xlabel("t")
plt.ylabel("$\delta x$")
plt.style.use("fivethirtyeight")
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 [16]:

```
%%time
%matplotlib widget

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

np.random.seed(42)

#Generating random clasters of abonents
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=(10,8))
    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_min), np.random.random(N_stations)*(y_max - 
y_min)
        population.append(stations)
    return population
```

```python
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_base = 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) for 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_profit_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_stations))
        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_sta
tions)*(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]))
```

```python
    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 = []
    N = 0
    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, population)

        # 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)]

        print(N)
        if(N%199==0):
            fig = make_subplots(rows=1, cols=2, subplot_titles=("Topology of the best 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[generation][:, 1], mode='markers', name='st
ations', 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()
        N+=1
    fig.write_html("test.html")
    return costs_evolution, abonents, best_creatures

## My new functions
def define_box(x):
  x1, x2 = x
  return [int(x1*10), int(x2*10)]

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

```
CPU times: user 14min 42s, sys: 372 ms, total: 14min 42s
Wall time: 14min 44s
```

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

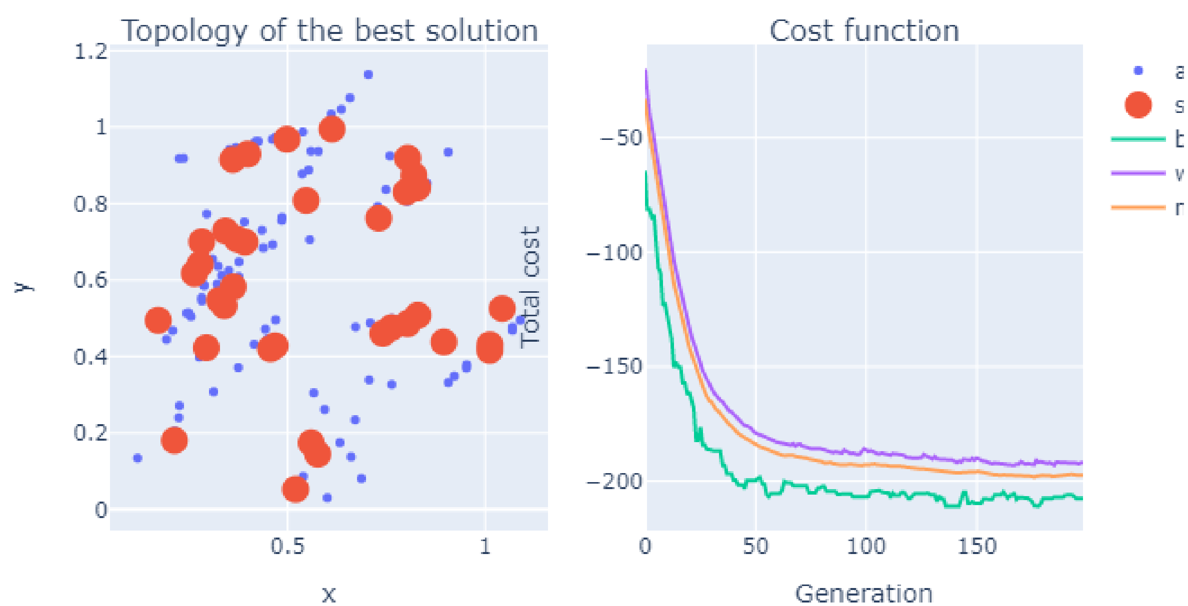
Предлагемые изменения

- 1 Если господь создал мир таким, то я не имею права с ним не соглашаться
- 2 Я прочитал дофига всего я хз что изменять, все мои изменения либо слишком рано скатывались во чтото не оптимальное (другие способы создания ребенка) Либо расходились(игрался с мутациями) Либо считались по 10 минут 1 эпоха (все вместе + жесткая система отбора с вырезанием больший части нации и заселением её гастарбайгастарбайтерами-мутантами)

Было

картинка запуска дефолтной работы алгоритма

Стало



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

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)

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

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

2)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

Solution

1)Разложим градиент по тейлору $\nabla f(y) = \nabla f(x) + \nabla^2 f(x)^T (y - x)$ и подставив нашу константу ограничить ф-цию сверху.

$$\nabla f(y) - \nabla f(x) \leq L(y - x)$$

Из чего следует и Липшицевость

А в обратную сторону: см Гасникова стр.20 формула(1.4) Из липшицевости следует ограниченность собственных значений гессиана (или *матрицы Гессе* как написано у Гасникова), а уже из этого следует то что надо доказать: см след. пункт

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

Спектральная норма говорит нам о том что $\sqrt{max_i \lambda(A^* A)}$. Возьмем наше спектральное разложение матрицы $H - LI_n = VA_\lambda V^{-1}$, где A_λ будет иметь вид

$$A_\lambda = \begin{pmatrix} L - \lambda_1 & 0 & \dots & 0 \\ 0 & L - \lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & L - \lambda_n \end{pmatrix}$$

которая в силу условия должна быть положительно определена, начит кажное соб-сное число должно быть больше нуля, значит $\lambda_i < L$, а значит $\|H\| < L$, т.к. $H = H^*$ и значит $\lambda(H^* H) = \lambda(H^2) = \lambda^2(H)$. Последнее верно т.к.

$$Ax = \lambda x$$

$$A^2 x = \lambda Ax = \lambda^2 x$$

Поменяв знак можно сделать тоже самое с левой частью данного в задаче неравенства

P.S. Спасибо Стифену и его братану Бойду за эту топовую бумажку (https://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf)

Problem 5

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

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

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 Липшица градиента функции

Итак: Разложим по Тейлору функцию

$$f(y) = f(x) + \langle \nabla f(x), y - x \rangle + \frac{1}{2} \langle \nabla^2 f(x)(y - x), y - x \rangle$$

см previous задачу там сказано про ограниченность собств значений *матрицы Гессе*

$$(y) = f(x) + \langle \nabla f(x), y - x \rangle + \frac{L}{2} \|y - x\|_2^2$$

шаг у нас в градиентном спуске такой:

$$x_{x+1} = x_k - \eta \nabla f(x_k)$$

подсталяем

$$f(x_{k+1}) \leq f(x_k) + \eta \langle \nabla f(x_k), \nabla f(x_k) \rangle + \frac{\eta^2 L}{2} \|\nabla f(x_k)\|_2^2$$
$$= f(x_k) - \eta (1 - \frac{L\eta}{2}) \|\nabla f(x_k)\|_2^2$$

Подставляем наши наблы

$$1) f(x_{k+1}) - f(x_k) \leq -\frac{1}{L}(1 - \frac{1}{2}) \|\nabla f(x_k)\|_2^2$$
$$\rightarrow f(x_k) - f(x_{k+1}) \geq \frac{0.5}{L} \|\nabla f(x_k)\|_2^2$$

Теперь другую

$$2) f(x_k) - f(x_{k+1}) \geq \frac{1 - \alpha_k}{L} \|\nabla f(x_k)\|_2^2$$

Problem 6

Рассмотрим функцию двух переменных:

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

где k - некоторый параметр

In [ ]:
```
def f_6(x, k=2):
    x_1, x_2 = x
    return x_1**2 + k*x_2**2

def df_6(x, k=2):
    return np.array([2*x[0], 2*k*x[1]])
```

In [ ]:

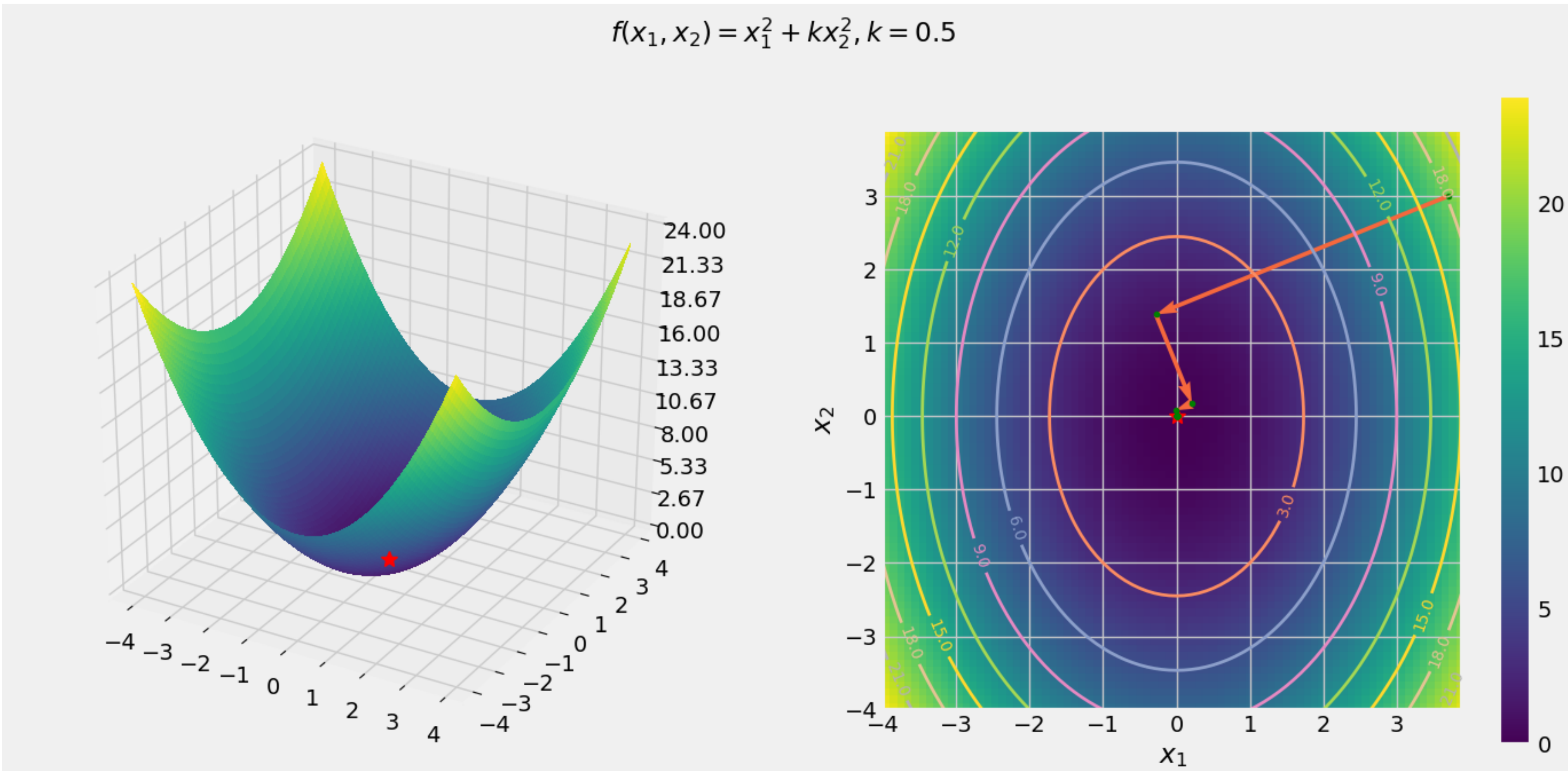
```
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter

def plot_3d_function(x1, x2, f, title, *f_params, minima = None, iterations = 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='viridis' ,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='viridis',  extent=[low_lim_1, up_lim_1, low_lim_2, up_lim_2])
    fig.colorbar(im)
```

```
    cset = ax.contour(x1, x2, Z, linewidths=2, cmap=plt.cm.Set2)
    ax.clabel(cset,inline=True,fmt='%1.1f',fontsize=10)

    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='xy', angles='xy', scale=1, color = "#f5683d")
    plt.style.use("fivethirtyeight")
    plt.show()
```

Для наглядности можете пользоваться кодом отрисовки окружающих картинок

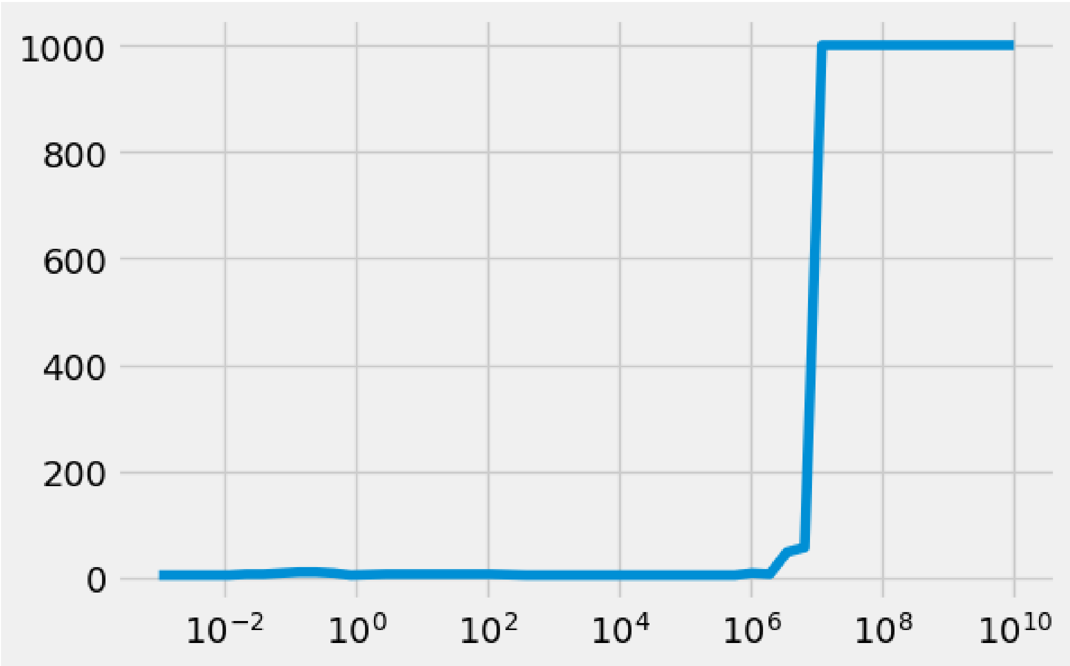
In [ ]:
```
import scipy.optimize

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 = optimize.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)
    return iterations
```

In [ ]:
```
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)
x_0 = [3.7,3]
k = 0.5
iterations = steepest_descent(x_0, f_6, df_6, 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_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.

In [ ]:
```
k_x = np.logspace(-3, 10)
y=[]
for k in k_x:
    y.append(len(steepest_descent([4, 7],f_6, df_6, k)))
plt.semilogx()
#plt.loglog()
plt.plot(k_x, y)
plt.show()
```



Сделайте те же графики для функции:

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

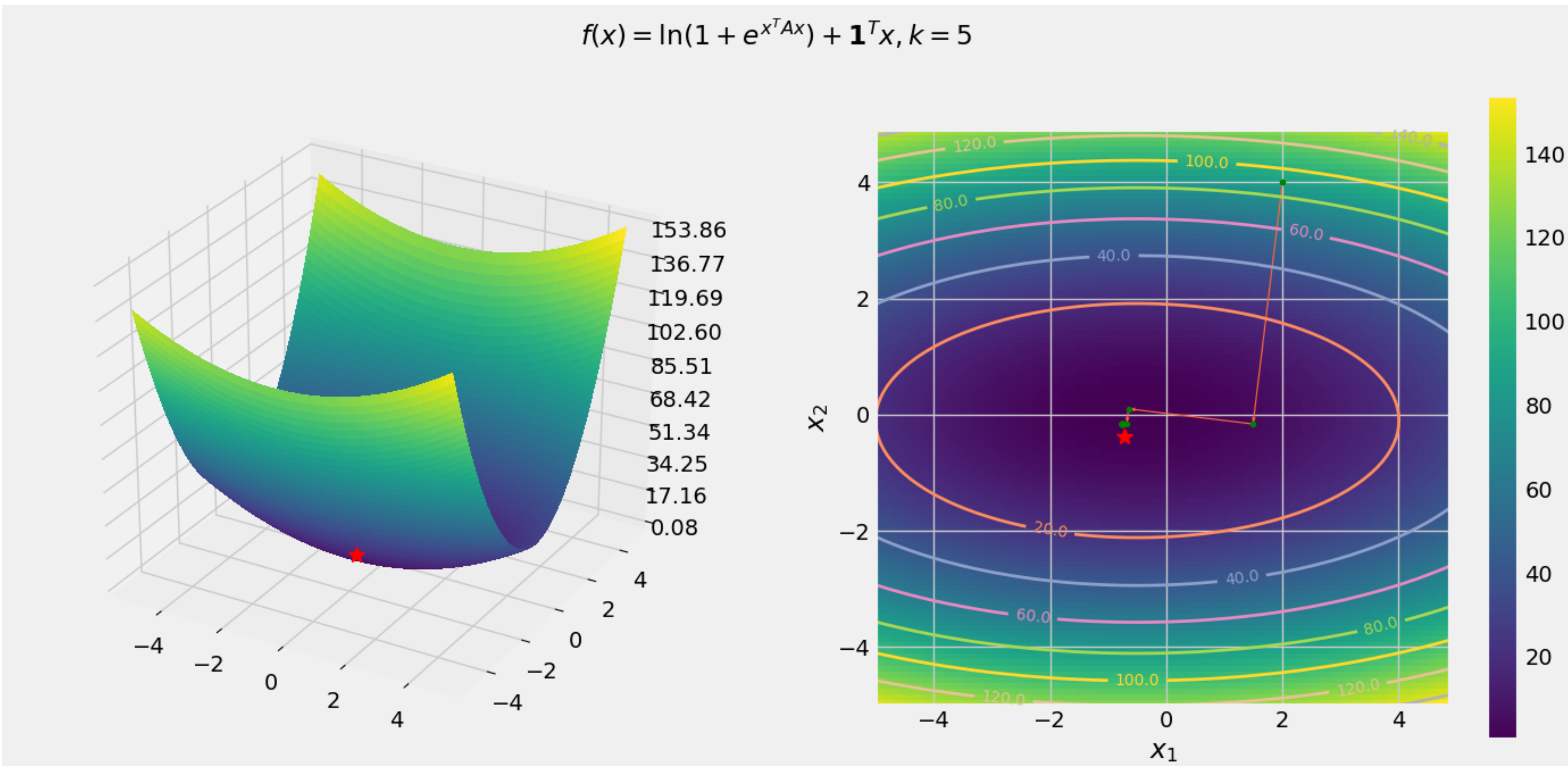
Объясните полученную зависимость.

In [ ]:

```
def f_l(x, k=2): #defining new function
    x1, x2 = x
    return np.log(1+np.exp(x1**2 + k*x2**2)) + x1 + x2

def df_l(x, k=2):
    x1, x2 = x
    A = np.array([[1, 0],[0, k]])
    A1 = np.array([1, 0])
    A2 = np.array([0, k])
    dfdx1 = np.exp(x1**2 + k*x2**2)*2*(x1)/(1+np.exp(x1**2 + k*x2**2)) + 1
    dfdx2 = np.exp(x1**2 + k*x2**2)*2*(k*x2)/(1+np.exp(x1**2 + k*x2**2)) + 1
    return np.array([dfdx1,dfdx2])

def df_l_x(x, k=2):
  return df_l(x, k)[:, 0]

def df_l_y(x, k=2):
  return df_l(x, k)[:, 1]
```

```
In [ ]: up_lim  = 5
        low_lim = -up_lim
        x1 = np.arange(low_lim, up_lim, 0.1)
        x2 = np.arange(low_lim, up_lim, 0.1)
        x_0 = [2,4]
        k = 5
        iterations = np.array(steepest_descent(x_0, f_l, df_l, k, df_eps = 1e-9))

        title = "$f(x) = \ln(1 + e^{x^T A x}) + \mathbf{1}^Tx,$"+ f"$k = {k}$"
        plot_3d_function(x1, x2, f_l, title, k, minima=min['x'], iterations = iterations)
```

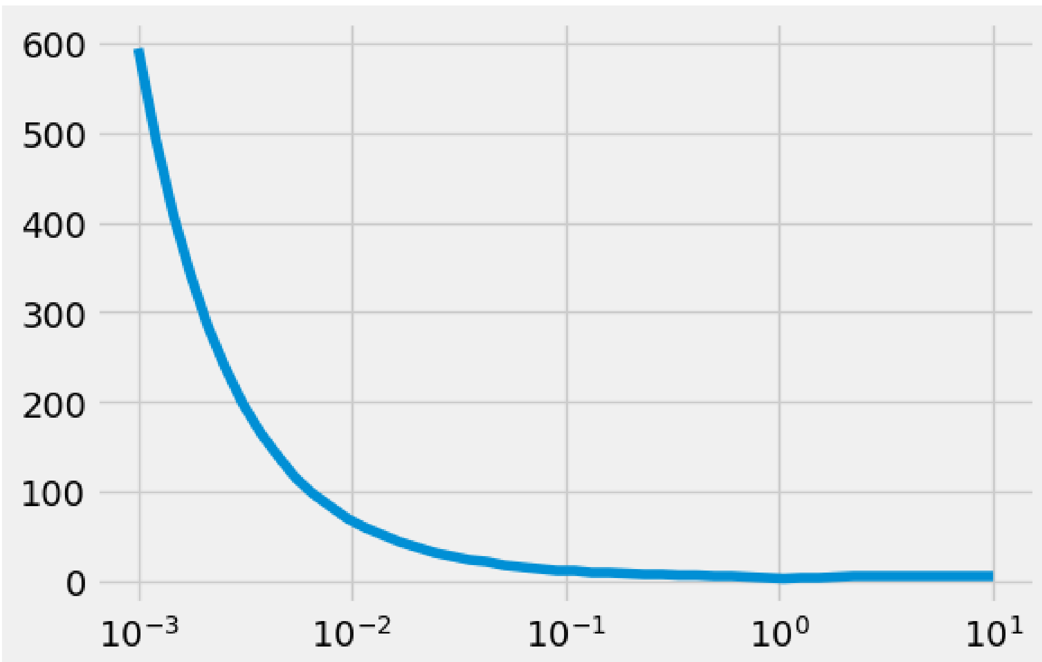
```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: RuntimeWarning: overflow encountered in exp
  This is separate from the ipykernel package so we can avoid doing imports until
/usr/local/lib/python3.6/dist-packages/scipy/optimize/optimize.py:1985: RuntimeWarning: invalid value encountered in double
_scalars
  tmp2 = (x - v) * (fx - fw)
```



In [ ]:

```
k_x = np.logspace(-3, 1)
y=[]
for k in k_x:
    y.append(len(steepest_descent([1.5, 2],f_l, df_l, k)))
plt.semilogx()
#plt.loglog()
plt.plot(k_x, y)
plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: RuntimeWarning: overflow encountered in exp
  This is separate from the ipykernel package so we can avoid doing imports until
/usr/local/lib/python3.6/dist-packages/scipy/optimize/optimize.py:1985: RuntimeWarning: invalid value encountered in double_scalars
  tmp2 = (x - v) * (fx - fw)
/usr/local/lib/python3.6/dist-packages/scipy/optimize/optimize.py:1986: RuntimeWarning: invalid value encountered in double_scalars
  p = (x - v) * tmp2 - (x - w) * tmp1
```



Объяснение поведения:

При маленьком k число обусловленности страдает и из-за машинной ошибки алгоритм не сходится

При большом k просто экспонента выходит за пределы (не обращай внимания на Exeption-ы)