Задание 1

$$F(X) = \sum_{i=1}^{N} x_i^2 \to extr$$

$$\sum_{i=1}^{N} x_i^4 - 1 \le 0$$

$$\frac{N}{\lambda_1^2} = 1$$

$$\lambda_1^2 = N$$

$$\lambda_1 = \pm\sqrt{\sqrt{N}}$$

Условия Куна-Таккера

$$\frac{\partial L}{\partial x} = 0 \; ; \quad \lambda_j \psi_j(x) = 0 \; ; \; j \in 1,k$$

$$\lambda_j \ge 0 \; ; \; j \in 1,k$$

$$L(\lambda_0, X, \lambda) = \lambda_0 \sum_{i=1}^{N} x_i^2 + \lambda_1 \left(\sum_{i=1}^{N} x_i^4 - 1\right)$$

$$\begin{cases} 2\lambda_0 x_1 + 4\lambda_1 x_1^3 = 0 \\ \dots \\ 2\lambda_0 x_n + 4\lambda_1 x_n^3 = 0 \\ \lambda_1 \left(\sum_{i=1}^{N} x_i^4 - 1\right) = 0 \end{cases}$$

При $\lambda_1 \neq 0$ сист. несовместна

$\exists \lambda_1 = 0$ и $\lambda_0 = 2$, тогда получим нулевой вектор

$$\exists \sum_{i=1}^{N} x_i^4 - 1 = 0 \text{ и } \lambda_1 \neq 0 \to$$

$$\begin{cases} \lambda_1 x_1^3 - x_1 = 0 \\ \dots \\ \lambda_1 x_n^3 - x_n = 0 \\ \sum_{i=1}^{N} x_i^4 = 1 \end{cases}$$

Решением этой системы будет 2^N точки вида $\left(\pm\sqrt{\frac{\sqrt{N}}{N}}; \dots \pm\sqrt{\frac{\sqrt{N}}{N}}\right)$ и точки вида $\left(\pm\sqrt{\frac{\sqrt{N-K}}{N-K}} \dots \pm\sqrt{\frac{\sqrt{N-K}}{N-K}}\right)$ где $0 < K < N$.

Проверим достаточное усл. экстремума.

Для вектора $(0 \dots 0)$ м. Гессе имеет вид $\begin{pmatrix} 2 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 2 \end{pmatrix}$, очевидно, что она положительна определена, следовательно $(0 \dots 0)$ - точка минимума

Точки вида $\left(\pm\sqrt{\frac{\sqrt{N-k}}{N-k}} \dots \pm\sqrt{\frac{\sqrt{N-k}}{N-k}}\right)$ не явл. точками экстремума, так как не лежат на границе.

Точки вида $\left(\pm\sqrt{\frac{\sqrt{N}}{N}} \dots \pm\sqrt{\frac{\sqrt{N}}{N}}\right)$ являются точками экстремума.

$$f\left(\pm\sqrt{\frac{\sqrt{N-k}}{N-k}} \dots \pm\sqrt{\frac{\sqrt{N-k}}{N-k}}\right) = (N-k)\cdot\frac{\sqrt{N-k}}{N-k} = \sqrt{N-k};$$

$0 < k < N$

$$f\left(\pm\sqrt{\frac{\sqrt{N}}{N}} \dots \pm\sqrt{\frac{\sqrt{N}}{N}}\right) = \sqrt{N}; \qquad \sqrt{N} > \sqrt{N-k}$$

Ответ: $(0\dots0)$ - т-ка минимума

$\left(\pm\sqrt{\frac{\sqrt{N}}{N}} \dots \pm\sqrt{\frac{\sqrt{N}}{N}}\right)$ - т-ка максимума.

Задание 1

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
from scipy import optimize
```

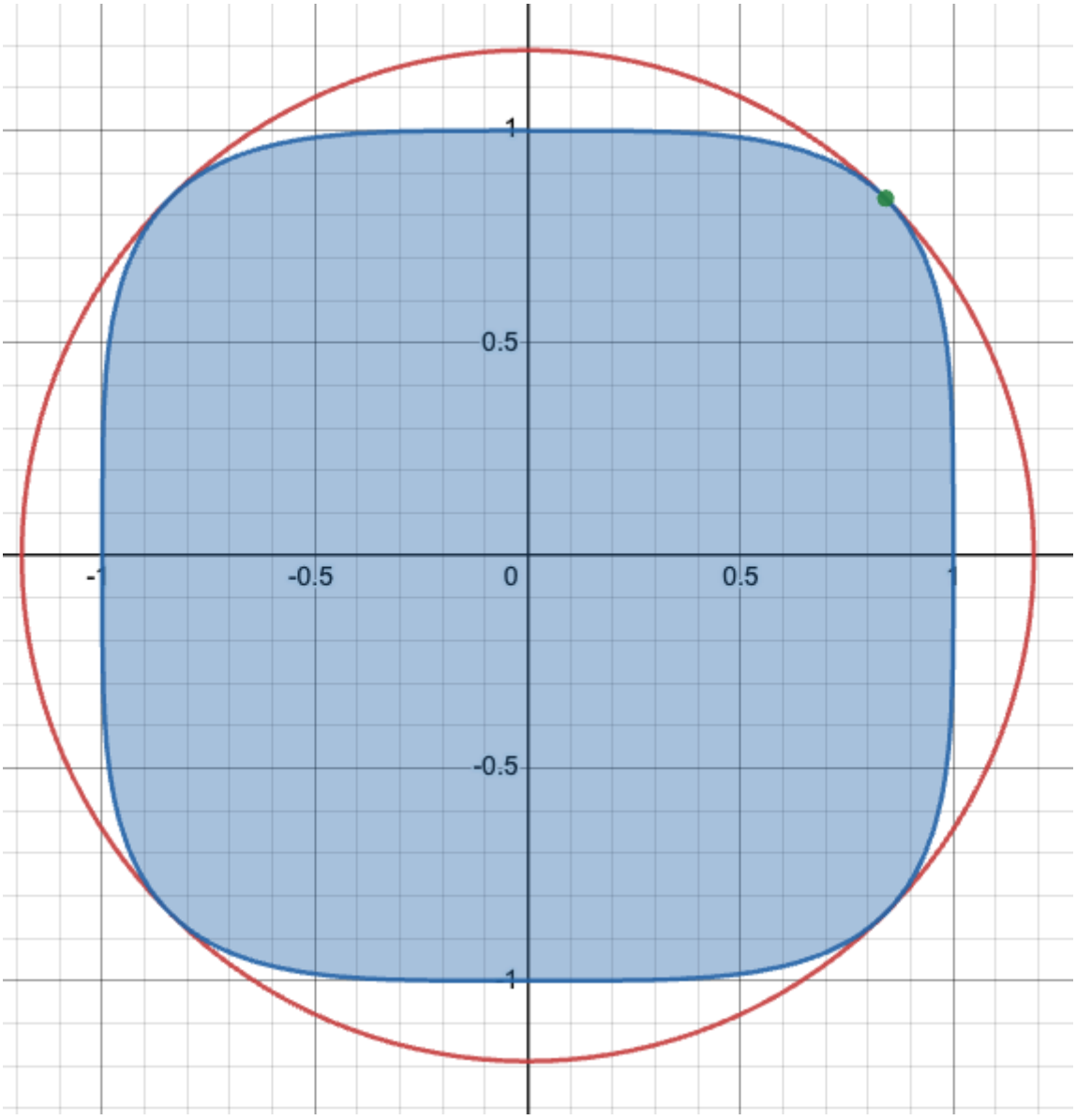
Покажем для случая двух переменных

In [ ]:
```
fun = lambda x: x[0]**2 + x[1]**2
cons = ({'type': 'ineq', 'fun': lambda x:  x[0]**4 + x[1]**4 - 1})
res = optimize.minimize(fun, (10, 0))
(round(res.fun), res.x)
```

Out[ ]:
```
(0, array([-4.14270251e-08,  0.00000000e+00]))
```

In [ ]:
```
fun = lambda x: -(x[0]**2) -(x[1]**2)
cons = ({'type': 'eq', 'fun': lambda x:  x[0]**4 + x[1]**4 - 1})
res = optimize.minimize(fun, (1, 1), constraints=cons)
(-res.fun, res.x)
```

Out[ ]:
```
(1.414213562384965, array([0.84089643, 0.8408964 ]))
```



	0	1	2	3	4	
0	M	4	5	7	5	
1	8	M	5	6	6	
2	3	5	M	9	6	
3	3	5	6	M	2	
4	6	2	3	8	M	

	0	1	2	3	4	
0	M	4	5	7	5	4
1	8	M	5	6	6	5
2	3	5	M	9	6	3
3	3	5	6	M	2	2
4	6	2	3	8	M	2

	0	1	2	3	4	
0	M	0	1	3	1	4
1	3	M	0	1	1	5
2	0	2	M	6	3	3
3	1	3	4	M	0	2
4	4	0	1	6	M	2
	0	0	0	1	0	
H0	17					

	0	1	2	3	4	
0	M	0(1)	1	2	1	1
1	3	M	0(1)	0(2)	1	0
2	0(3)	2	M	5	3	2
3	1	3	4	M	0(2)	1
4	4	0(1)	1	5	M	1
	1	0	1	2	1	

	0	1	2	3	4	
0	M	0	1	2	1	0
1	3	M	0	0	1	0
2	M	2	M	5	3	2
3	1	3	4	M	0	0
4	4	0	1	5	M	0
	1	0	0	0	0	
H(2,0)	20					

	1	**2**	**3**	**4**		
0	0(1)	M	2	1	1	
1	M	0(1)	0(2)	1	0	
3	3	4	M	0(4)	3	
4	0(1)	1	5	M	1	
	0	1	2	1		

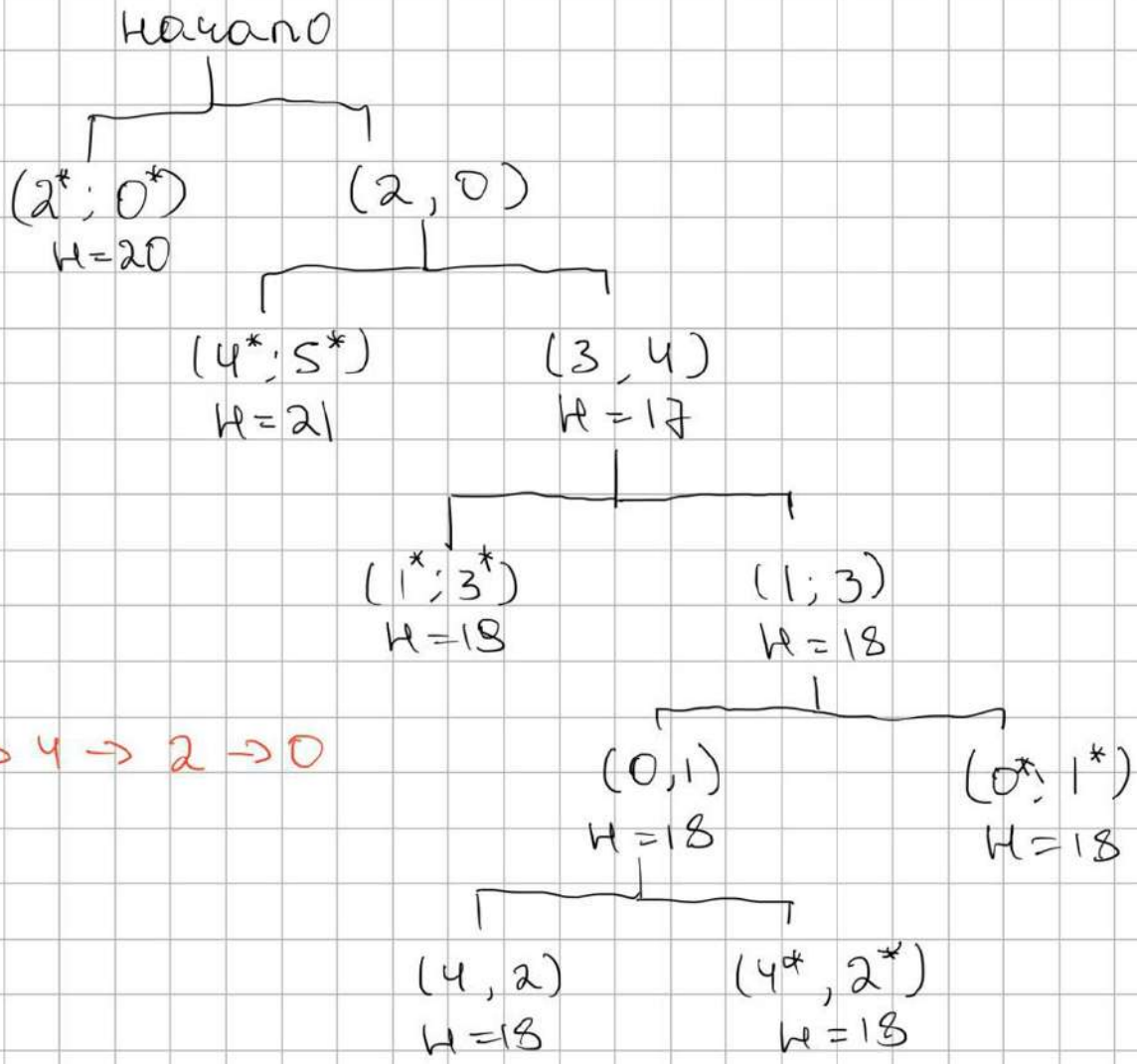
	1	**2**	**3**	**4**		
0	0	1	2	1	0	
1	M	0	0	1	0	
3	3	4	M	M	3	
4	0	1	5	M	0	
	0	0	0	1		
H(3,4)	21					

	1	**2**	**3**			
0	0(2)	M	2	2		
1	M	0(1)	0(2)	0		
4	0(1)	1	M	1		
	0	1	2			

	1	**2**	**3**			
0	0	M	2	0		
1	M	0	M	0		
4	0	1	M	0		
	0	0	2			
H(1,3)	19					

	1	**2**				
0	0	M	0			
4	0	1	0			
	0	1				
H(0,4)	18					

Итоговый ответ	0 -> 1 -> 3 -> 4 -> 2 -> 0					



$0 \to 1 \to 3 \to 4 \to 2 \to 0$

$\sum C_{ij} = 18$

Задание 2

Пункт 2

In [ ]:
```
!python3 -m pip install --user ortools
```

In [ ]:
```
# импорты
from ortools.constraint_solver import pywrapcp
from ortools.constraint_solver import routing_enums_pb2
```

In [ ]:
```
def create_data_model():
    """Задает параметры заадчи"""
    data = {}
    data["distance_matrix"] = [
        [0, 4, 5, 7, 5],
        [8, 0, 5, 6, 6],
        [3, 5, 0, 9, 6],
        [3, 5, 6, 0, 2],
        [6, 2, 3, 8, 0]
    ]
    data["num_vehicles"] = 1
    data["depot"] = 0
    return data
```

In [ ]:
```
# создадим модель маршрутизации
data = create_data_model()
manager = pywrapcp.RoutingIndexManager(
    len(data["distance_matrix"]), data["num_vehicles"], data["depot"]
)
routing = pywrapcp.RoutingModel(manager)
```

In [ ]:
```
def distance_callback(from_index, to_index):
    """Возвращает расстояние между двумя узлами"""
    from_node = manager.IndexToNode(from_index)
    to_node = manager.IndexToNode(to_index)
    return data["distance_matrix"][from_node][to_node]

transit_callback_index = routing.RegisterTransitCallback(distance_callback)
```

In [ ]:
```
def print_solution(manager, routing, solution):
    """Печать решения"""
    print(f"Оптимальное решение: {solution.ObjectiveValue()} км")
    index = routing.Start(0)
    plan_output = "Старт из пункта 0:\n"
    route_distance = 0
    while not routing.IsEnd(index):
        plan_output += f" {manager.IndexToNode(index)} ->"
        previous_index = index
        index = solution.Value(routing.NextVar(index))
        route_distance += routing.GetArcCostForVehicle(previous_index, index
    plan_output += f" {manager.IndexToNode(index)}\n"
    print(plan_output)
    plan_output += f"Route distance: {route_distance}miles\n"
```

In [ ]:
```
# зададим стоимость проезда
routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)
```

In [ ]:
```
# установим параметры поиска, в данном случае – самый кратчайший путь
search_parameters = pywrapcp.DefaultRoutingSearchParameters()
search_parameters.first_solution_strategy = (
    routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC
)
```

In [ ]:
```
# получим решение и напечатаем его
solution = routing.SolveWithParameters(search_parameters)
if solution:
    print_solution(manager, routing, solution)
```

```
Оптимальное решение: 18 км
Старт из пункта 0:
 0 -> 1 -> 3 -> 4 -> 2 -> 0
```

Пункт 3

Реализуем метод имитации отжига

Зададим целевую функцию - наша задача, стартуя из точки 0 обеспечить кратчайший путь, проходящий через всегда города **только один раз**, при этом маршрут должен заканчиваться в точке 0.

Тогда целевая функция имеет вид:

$$f = (\sum_{i=0}^{N-2} dist(A_i, A_{i+1})) + dist(A_{N-1}, A_0) \rightarrow min,$$

где

N - количество городов

$dist(A_i, A_{i+1})$ - расстояние между городами $i \in [0, N-1]$

Встает вопрос: как получать новое состояние? Будем брать два случайных города в маршруте и инвертировать их. Например, у нас есть маршрут `A -> B -> C -> D -> E -> A`, тогда если мы случайным образом выберем два города, пусть это будут `B` и `E`, тогда новый маршрут будет выглядеть следующим образом `A -> E -> C -> D -> B -> A`.

В нашей постановке учитывается, что из любого города можно добраться в любой, иначе пришлось бы проверять новые маршруты на валидность.

In [ ]:
```
import random
import numpy as np
```

In [ ]:
```
# Начальные данные
start = 0 # стартовая точка, она же конечная
dist = [
        [0, 4, 5, 7, 5],
        [8, 0, 5, 6, 6],
        [3, 5, 0, 9, 6],
        [3, 5, 6, 0, 2],
```

```
        [6, 2, 3, 8, 0]
    ] # матрица расстояний
```

In [ ]:
```
def new_path(path: list[int]) -> list[int]:
    """Возвращает новый путь"""
    a, b = random.sample(path[1:-1], k=2)
    a_index = path.index(a)
    b_index = path.index(b)
    path[a_index], path[b_index] = path[b_index], path[a_index]
    return path
```

In [ ]:
```
def path_cost(dist: list[list[int]], path: list[int]) -> int:
    """Целевая функция"""
    total = sum(dist[path[i]][path[i + 1]] for i in range(len(path) - 2))
    return total + dist[path[-2]][path[0]]
```

In [ ]:
```
def simulated_annealing(dist, start, init_temp, end_temp):
    n = len(dist)
    temp_path = [x for x in range(len(dist)) if x != start]
    random.shuffle(temp_path)
    path = [start] + temp_path + [start]

    current_energy = path_cost(dist, path)
    t = init_temp
    k = 0
    while t > end_temp:
        k += 1
        condidtate_path = new_path(path)
        condidtate_energy = path_cost(dist, condidtate_path)
        de = condidtate_energy - current_energy
        if de <= 0:
            path = condidtate_path
            current_energy = condidtate_energy
        else:
            p = np.exp(-de / t)
            if random.random() <= p:
                path = condidtate_path
                current_energy = condidtate_energy

        t = init_temp * 0.1 / k

    return (current_energy, path)
```

In [ ]:
```
res = simulated_annealing(dist, start, 10, 0.0001)
res
```

Out[ ]:
```
(18, [0, 2, 3, 4, 1, 0])
```

Понятно, что алгоритм имитации отжига не является устойчивым относительно количества итераций, продемонстрируем это примером

In [ ]:
```
opt = 18
wrong_ans = 0
n = 100
for i in range(n):
    res = simulated_annealing(dist, start, 10, 0.1)
    if res[0] != opt:
        wrong_ans += 1
print(wrong_ans)
```

Мы сократили число возможных итераций в этом методе и получили, что около 70% решений были не оптимальными. Таким образом, мы видим, что итоговое получается очень неустойчивым в зависимости от количества итераций.

Однако, в исходной реализации мы использовали в качестве начального решения случайно сгенерированный маршрут и независимо от начального решения мы получаем один и тот же результат при одинаковом количестве итераций. Покажем это

In [ ]:
```
opt = 18
wrong_ans = 0
n = 100
for i in range(n):
    res = simulated_annealing(dist, start, 10, 0.0001)
    if res[0] != opt:
        wrong_ans += 1
print(wrong_ans)
```

```
0
```

Число решений, которые не совпадают с оптимальным равно нулю, что и требовалось показать