Оптимизация ДЗ 2.

1)

$$\begin{cases} x_1^2 + \ldots + x_n^2 \to extr, \\ x_1^4 + \ldots + x_n^4 - 1 \leq 0. \end{cases}$$

Решаем по методу Лагранжа:

$$L(x_1, \ldots, x_n, \lambda) = \sum_{i=1}^{n} x_i^2 + \lambda \cdot \left(\sum_{i=1}^{n} x_i^4 - 1\right) =$$

$$= \lambda_0 (x_1^2 + \ldots + x_n^2) + \lambda_1 (x_1^4 + \ldots + x_n^4 - 1)$$

Частные производные:

$$\begin{cases} \frac{\partial L}{\partial x_i} = 2x_i + \lambda \cdot 4x_i^3 = 0 \\ \frac{\partial L}{\partial \lambda} = \sum_{i=1}^{n} x_i^4 - 1 = 0 \end{cases} \Rightarrow \begin{cases} x_i (1 + 2\lambda x_i^2) = 0 \\ \sum_{i=1}^{n} (x_i^2)^2 - 1 \leq 0 \end{cases}$$

$$\Downarrow$$

$$\begin{cases} x_i = 0 \quad (\text{т.к. } -1 \leq 0 \text{ - подх}) \\ x_i^2 = -\frac{1}{2\lambda} \end{cases}$$

$\lambda = \pm \frac{\sqrt{n}}{2}$, $\quad \frac{\sqrt{n}}{2}$ - не подх. $\left(x_i^2 = -\frac{1}{\sqrt{n}}\right)$

$\quad -\frac{\sqrt{n}}{2}$ - подх. $\left(x_i^2 = \frac{1}{\sqrt{n}}\right)$

Теперь найдём значение функции в точке:

$$f(x_1, \dots, x_n) = \sum_{i=1}^{n} x_i^2 = n \cdot \frac{1}{\sqrt{n}} = \sqrt{n}$$

$$f(x_1, \dots, x_n) = \sum_{i=1}^{n} 0 = 0$$

$$\Delta = -\begin{vmatrix} 0 & f'_{x_i} \\ f'_{x_i} & L''_{x_i, x_i} \end{vmatrix} = -\begin{vmatrix} 0 & 4x_i^3 \\ 4x_i^3 & 2+12\lambda x_i^2 \end{vmatrix} = 16x_i^6$$

$$\Delta_1 = 16\left(\frac{1}{\sqrt{n}}\right)^3 = \frac{16}{n\sqrt{n}} > 0 \Rightarrow$$ локальный минимум

$$\Delta_2 = 16 \cdot 0^6 = 0 \Rightarrow$$ не даёт допустимого решения

2

1)

$$\begin{pmatrix} \infty & 4 & 5 & 7 & 5 \\ 8 & \infty & 5 & 6 & 6 \\ 3 & 5 & \infty & 9 & 6 \\ 3 & 5 & 6 & \infty & 2 \\ 6 & 2 & 3 & 8 & \infty \end{pmatrix}$$

2)

∞	0	1	3	1
3	∞	0	1	1
0	2	∞	6	3
1	3	4	∞	0
4	0	1	6	∞
0	0	0	1	0

3)

∞	0	1	2	1
3	∞	0	0	1
0	2	∞	5	3
1	3	4	∞	0
4	0	1	5	∞

$H_0 = 17$

ление.

4)

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

5)

∞	0	1	2	1	0
3	∞	0	0	1	0
∞	2	∞	5	3	2
1	3	4	∞	0	0
4	0	1	6	∞	0
1	0	0	0	0	

$H_1 = 20$

0	∞	2	1	0
∞	0	0	1	0
3	4	∞	0	0
0	1	5	∞	0
0	0	0	0	

$H_1 = 17$

6)

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

7)

0	∞	2	1	0
∞	0	0	1	0
3	4	∞	∞	3
0	1	5	∞	0
0	0	0	1	

$H_2 = 21$

0	∞	2	0
∞	0	0	0
0	1	∞	0
0	0	0	

$H_2 = 17$

8) $\begin{matrix} 0(2) & \infty & 2 \\ \infty & 0(1) & 0(2) \\ 0(1) & 1 & \infty \end{matrix}$

9)
$$\begin{array}{ccc|c} 0 & \infty & 2 & 0 \\ \infty & 0 & \infty & 0 \\ 0 & 1 & \infty & 0 \\ \hline 0 & 0 & 2 & \end{array}$$

$H_3 = 19$

$$\begin{array}{cc|c} 0 & \infty & 0 \\ 0 & 1 & \\ \hline 0 & 1 & 0 \end{array}$$

$H_3 = 18$

10) $\begin{matrix} 0 & \infty \\ 0(1) & 1 \end{matrix}$

11)
$$\begin{array}{cc|c} 0 & \infty & 0 \\ \infty & 1 & 0 \\ \hline 0 & 1 & \end{array}$$

$H_4 = 19$

$\Rightarrow H_4 = \infty$

Ответ: $C \to A \to B \to D \to E \to C$

$3 \to 1 \to 2 \to 4 \to 5 \to 3$

```
from scipy.optimize import minimize
import numpy as np
```

Задание 1.

Решить аналитически и проверить при помощи оптимизатора в Python. Оптимизатор можно использовать на своё усмотрение.

```
def objective(x):
    return np.sum(x**2)

# Определение ограничения
def constraint_eq(x):
    return np.sum(x**4) - 1

x0 = np.full(10, 0**(1/4))
opt_method = 'SLSQP'
res = minimize(objective, x0, method=opt_method, constraints={'type':
'eq', 'fun': constraint_eq})

print("Минимальное значение функции:", res.fun)
print("Точка минимума:", res.x)
```

```
Минимальное значение функции: 0.0
Точка минимума: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

```
initial_guess = np.array([10**(1/4)]*10)

method = 'SLSQP'

result = minimize(objective, initial_guess, method=method,
constraints={'type': 'eq', 'fun': constraint_eq})

print("Минимальное значение функции:", result.fun)
print("Точка минимума:", result.x)
```

```
Минимальное значение функции: 3.1622776601542704
Точка минимума: [0.56234425 0.56234425 0.56234425 0.56234425
0.56234042 0.56234042
 0.56234039 0.56234039 0.5623427  0.56233191]
```

Заметно, что аналитическое решение было корректным

Задача 2.

1. Решить аналитически и проверить при помощи оптимизатора в Python. Оптимизатор можно использовать на своё усмотрение (например, ORTools).
2. Также дополнительно помимо оптимизатора использовать какой-нибудь метаэвристический алгоритм (имитация отжига / квантовый отжиг / муравьиный алгоритм / генетический алгоритм) для проверки результатов.

3. Дать оценку устойчивости метаэвристики в зависимости от начальной точки и от количества итераций.

```
import random
import numpy as np
from numpy.random import choice
from ortools.constraint_solver import routing_enums_pb2, pywrapcp

class ACO_Solver:
    def __init__(self, cost_matrix, ants, elite_ants, iterations,
evaporation, alpha=1, beta=1):
        self.cost_matrix = cost_matrix
        self.pheromones = np.ones_like(cost_matrix) / len(cost_matrix)
        self.ants = ants
        self.elite_ants = elite_ants
        self.iterations = iterations
        self.evaporation = evaporation
        self.alpha = alpha
        self.beta = beta
        self.total_nodes = len(cost_matrix)
        self.nodes_list = range(self.total_nodes)

    def optimize(self):
        best_route = None
        best_length = float("inf")
        for _ in range(self.iterations):
            generated_routes = self.construct_routes()
            self.enhance_pheromones(generated_routes)
            current_best = min(generated_routes, key=lambda x: x[1])
            if current_best[1] < best_length:
                best_route, best_length = current_best
            self.pheromones *= self.evaporation
        return best_route, best_length

    def construct_routes(self):
        routes = []
        for _ in range(self.ants):
            tour = self.generate_tour(0)
            routes.append((tour, self.evaluate_route(tour)))
        return routes

    def generate_tour(self, start):
        path = []
        visited = {start}
        prev_node = start
        for _ in range(self.total_nodes - 1):
            next_node = self.pick_next_node(prev_node, visited)
            path.append((prev_node, next_node))
            prev_node = next_node
            visited.add(next_node)
```

```
        path.append((prev_node, start))
        return path

    def pick_next_node(self, current, visited):
        local_pheromones = np.copy(self.pheromones[current])
        local_pheromones[list(visited)] = 0
        weights = (local_pheromones ** self.alpha) * (1.0 /
self.cost_matrix[current]) ** self.beta
        weights /= weights.sum()
        return choice(self.nodes_list, 1, p=weights)[0]

    def evaluate_route(self, path):
        return sum(self.cost_matrix[i][j] for i, j in path)

    def enhance_pheromones(self, routes):
        top_routes = sorted(routes, key=lambda x: x[1])
[:self.elite_ants]
        for route, length in top_routes:
            for i, j in route:
                self.pheromones[i][j] += 1.0 / length


def setup_problem():
    params = {}
    params["cost_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],
    ]
    params["vehicles"] = 1
    params["start"] = 0
    return params


def display_result(params, manager, model, solution):
    print(f"Optimal cost: {solution.ObjectiveValue()}")
    for vid in range(params["vehicles"]):
        idx = model.Start(vid)
        route_plan = f"Vehicle {vid} follows route:\n"
        while not model.IsEnd(idx):
            route_plan += f" {manager.IndexToNode(idx) + 1} ->"
            idx = solution.Value(model.NextVar(idx))
        route_plan += f" {manager.IndexToNode(idx) + 1}\n"
        print(route_plan)


def solve_vrp():
    params = setup_problem()
```

```
    manager = pywrapcp.RoutingIndexManager(
        len(params["cost_matrix"]), params["vehicles"],
params["start"]
    )
    model = pywrapcp.RoutingModel(manager)

    def distance_function(i, j):
        return params["cost_matrix"][manager.IndexToNode(i)]
[manager.IndexToNode(j)]

    transit_callback_idx =
model.RegisterTransitCallback(distance_function)
    model.SetArcCostEvaluatorOfAllVehicles(transit_callback_idx)
    model.AddDimension(transit_callback_idx, 0, 3000, True,
"Distance")
    search_config = pywrapcp.DefaultRoutingSearchParameters()
    search_config.first_solution_strategy =
routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC
    solution = model.SolveWithParameters(search_config)
    if solution:
        display_result(params, manager, model, solution)


if __name__ == "__main__":
    solve_vrp()
```

```
Optimal cost: 18
Vehicle 0 follows route:
 1 -> 2 -> 4 -> 5 -> 3 -> 1
```

Результаты совпали с аналитическим решением, что подтверждает корректность метода.

Стоимость маршрута зависит от начальной температуры: чем она ближе к минимальному значению, тем выше вероятность получения некорректного решения.