Домашнее задание №9

Задание 1:

Найти экстремумы функции $f(x_1, \dots, x_n) = \sum_{i=1}^{n} x_i^2$ на множестве $\sum_{i=1}^{n} x_i^4 \leq 1$, то есть решить задачу

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

Решить аналитически и проверить при помощи оптимизатора в Python.

Решение:

Составим функцию Лагранжа:

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

Найдем производные:

$\frac{\partial L}{\partial x_i} = 2\lambda_0 x_i + 4\lambda_1 x_i^3$

$$\begin{cases} 2\lambda_0 x_i + 4\lambda_1 x_i^3 = 0, i = \overline{1, n} \\ \lambda_1(x_1^4 + \dots + x_n^4 - 1) = 0 \end{cases}$$

Рассмотрим первый случай $\lambda_0 = 0$:

$$\begin{cases} 4\lambda_1 x_i^3 = 0, i = \overline{1, n} \\ \lambda_1(x_1^4 + \dots + x_n^4 - 1) = 0 \end{cases}$$

В данном случае есть два варианта:

- $\lambda_1 = 0$: в таком случае весь вектор λ будет нулевой
- $\lambda_1 \neq 0$: в таком случае система будет несовместной

$\Rightarrow \lambda_0 \neq 0$

Рассмотрим второй случай $\lambda_0 \neq 0$:

Пусть $\lambda_1 = 2$, тогда:

$$\begin{cases} x_i + \lambda_1 x_i^3 = 0, i = \overline{1, n} \\ \lambda_1(x_1^4 + \dots + x_n^4 - 1) = 0 \end{cases}$$

$$\begin{cases} x_i(1 + \lambda_1 x_i^2) = 0, i = \overline{1, n} \\ \lambda_1(x_1^4 + \dots + x_n^4 - 1) = 0 \end{cases}$$

Из первого уравнения $\Rightarrow x_i = 0$ или $1 + \lambda_1 x_i^2 = 0$

1. $x_i = 0$

В таком случае из второго уравнения $\Rightarrow \lambda_1 = 0 \Rightarrow a = (0, 0, \dots, 0)$

$f(a) = 0$

2. $1 + \lambda_1 x_i^2 = 0$

$x_i^2 = -\frac{1}{\lambda_i} \Rightarrow \lambda_1 > 0$

из второго уравнения
$\Rightarrow (\frac{1}{\lambda_1^2} + \dots + \frac{1}{\lambda_1^2} - 1)\lambda_1 = 0 \Rightarrow \frac{n}{\lambda_1^2} - 1 = 0 \Rightarrow \lambda_1 = -\sqrt{n} \Rightarrow x_i = \pm\frac{1}{\sqrt[4]{n}} \Rightarrow b = (\pm\frac{1}{\sqrt[4]{n}}, \pm\frac{1}{\sqrt[4]{n}}$

$f(b) = \frac{n}{\sqrt{n}} = \sqrt{n}$

Итого $2^n + 1$ точек экстремума. Все точки b - точки максимума, при этом значения функции в точках b одинаковое и равно $\sqrt{n}$. Точка a - точка минимума. Значение функции в точке a равно 0

Проверка с помощью оптимизатора на Python

Зададим нашу функцию и ограничения

In [ ]:
```
from scipy.optimize import minimize

def target_function(x):
    # Функция
    return (sum(i**2 for i in x))

# Ограничение
constraints = ({'type': 'eq', 'fun': lambda x: sum(i**4 for i in x) - 1})
```

Предположим что $n = 6$ и найдем минимум функции

In [ ]:
```
# Начальное значение для переменных
n = 6
initial_guess = [0.0 for _ in range(n)] #, 0.0, 0.0

# Проведем оптимизацию для поиска минимума с ограничением
result_min = minimize(target_function, initial_guess, method='trust-constr', constraints

# Выведем результаты минимизации
print("Минимум функции:", result_min.fun)
print("Аргумент минимума:", result_min.x)
```

```
Минимум функции: 0.0
Аргумент минимума: [0. 0. 0. 0. 0. 0.]
```

Ответ совпал. Дальше найдем максимум функции. Для этого найдем минимум функции $-f(x)$

In [ ]:
```
def target_function(x):
    # Функция
    return -1*(sum(i**2 for i in x))
```

Предположим что в этот раз $n = 15$ и найдем минимум для этой функции

In [ ]:
```
# Начальное значение для переменных
n = 15
initial_guess = [0.0 for _ in range(n)] #, 0.0, 0.0
```

```
# Проведем оптимизацию для поиска минимума с ограничением
result_min = minimize(target_function, initial_guess, method='trust-constr', constraints

# Выведем результаты минимизации
print("Минимум функции:", result_min.fun)
print("Аргумент минимума:", result_min.x)
```

```
Минимум функции: -3.8729833462074175
Аргумент минимума: [0.50813275 0.50813275 0.50813275 0.50813275 0.50813274 0.50813275
 0.50813275 0.50813275 0.50813275 0.50813275 0.50813275 0.50813275
 0.50813274 0.50813275 0.50813275]
```

Проверим ответы:

Исходя из аналитического решения, следует что максимум функции равен $\sqrt{n}$. В данном случе $n = 15 \Rightarrow \sqrt{n} = 3.87298$ Аргументы для точки максимума равны следующим значениям $b = (\pm\frac{1}{\sqrt[4]{n}}, \pm\frac{1}{\sqrt[4]{n}}, \dots, \pm\frac{1}{\sqrt[4]{n}})$; $\frac{1}{\sqrt[4]{n}} = 0.508133$ при $n = 15$

Ответ снова совпал.

Задание 2:

1. Решить аналитически и проверить при помощи оптимизатора в Python.
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.

Решить задачу коммивояжёра методом ветвей и границ:

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

Решение методом ветвей и границ:

Шаг 1

Город	A	B	C	D	E	d_i
A	M	4	5	7	5	4
B	8	M	5	6	6	5
C	3	5	M	9	6	3
D	3	5	6	M	2	2

Город	A	B	C	D	E	d_i
E	6	2	3	8	M	2

Шаг 2

Город	A	B	C	D	E
A	M	0	1	3	1
B	3	M	0	1	1
C	0	2	M	6	3
D	1	3	4	M	0
E	4	0	1	6	M
d_j	0	0	0	1	0

Шаг 3

Город	A	B	C	D	E
A	M	0	1	2	1
B	3	M	0	0	1
C	0	2	M	5	3
D	1	3	4	M	0
E	4	0	1	5	M

$H_0 = 4 + 5 + 3 + 2 + 2 + 0 + 0 + 0 + 1 + 0 = 17$

Город	A	B	C	D	E
A	M	0(1)	1	2	1
B	3	M	0(1)	0(2)	1
C	**0(3)**	2	M	5	3
D	1	3	4	M	0(2)
E	4	0(1)	1	5	M

Наибольшая сумма для ребра C-A (3) $\Rightarrow$ разбиваем на два множества C-A и C-*A*

Шаг 4

Исключение ребра C-A

Город	A	B	C	D	E	d_i
A	M	0	1	2	1	0
B	3	M	0	0	1	0
C	M	2	M	5	3	2
D	1	3	4	M	0	0
E	4	0	1	5	M	0
d_j	1	0	0	0	0	

$H_1^* = 17 + 3 = 20$

Включение ребра C-A

Город	**B**	**C**	**D**	**E**	d_i
A	0	M	2	1	0
B	M	0	0	1	0
D	3	4	M	0	0
E	0	1	5	M	0
d_j	0	0	0	0	

$H_1 = 17 + 0 = 17$

Так как 17 < 20 $\Rightarrow$ включаем в маршрут ребро C-A

Шаг 5

Город	**B**	**C**	**D**	**E**
A	0(1)	M	2	1
B	M	0(1)	0(2)	1
D	3	4	M	0(4)
E	0(1)	1	5	M

Наибольшая сумма для ребра D-E (4) $\Rightarrow$ разбиваем на два множества D-E и D-*E*

Шаг 6

Исключение ребра D-E

Город	**B**	**C**	**D**	**E**	d_i
A	0	M	2	1	0
B	M	0	0	1	0
D	3	4	M	M	3
E	0	1	5	M	0
d_j	0	0	0	1	

$H_2^* = 17 + 4 = 21$

Включение ребра D-E

Город	**B**	**C**	**D**	d_i
A	0	M	2	0
B	M	0	0	0
E	0	1	M	0
d_j	0	0	0	

$H_2 = 17 + 0 = 17$

Так как 17 < 21 $\Rightarrow$ включаем в маршрут ребро D-E

Шаг 7

Город	B	C	D
A	0(2)	M	2
B	M	0(1)	0(2)
E	0(1)	1	M

Наибольшая сумма для ребра B-D (2) $\Rightarrow$ разбиваем на два множества B-D и B-*D*

Шаг 8

Исключение ребра B-D

Город	B	C	D	d_i
A	0	M	2	0
B	M	0	M	0
E	0	1	M	0
d_j	0	0	2	

$H_3^* = 17 + 2 = 19$

Включение ребра B-D

Город	B	C	d_i
A	0	M	0
E	0	1	0
d_j	0	1	

$H_3 = 17 + 1 = 18$

Так как 18 < 19 $\Rightarrow$ включаем в маршрут ребро B-D

Шаг 7

Город	B	C
A	0	M
E	0(1)	1

Наибольшая сумма для ребра E-B (1) $\Rightarrow$ разбиваем на два множества E-B и E-*B*

Шаг 8

Исключение ребра E-B

Город	B	C	d_i
A	0	M	0
E	M	1	0
d_j	0	1	

$H_4^* = 18 + 1 = 19$

Включение ребра E-B

Город	С
A	M

$H_3 = 17 + \infty = \infty$

Так как 19 < $\infty \Rightarrow$ исключаем в маршрут ребро E-B

$\Rightarrow$ включаем в маршрут оставшиеся ребра A-B и E-C $\Rightarrow H = 18$

Итоговый маршрут

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

Проверка с помощью ORTools

In [ ]:

```
from ortools.constraint_solver import routing_enums_pb2
from ortools.constraint_solver import pywrapcp

def create_data_model():
    """Stores the data for the problem."""
    data = {}
    data["distance_matrix"] = [
        [0, 4.0, 5.0, 7.0, 5.0],
        [8.0, 0, 5.0, 6.0, 6.0],
        [3, 5, 0, 9, 6.0],
        [3.0, 5.0, 6.0, 0, 2.0],
        [6.0, 2.0, 3.0, 8.0, 0]
    ]
    data["num_vehicles"] = 1
    data["depot"] = 0
    return data

data = create_data_model()
manager = pywrapcp.RoutingIndexManager(
    len(data["distance_matrix"]), data["num_vehicles"], data["depot"]
)
routing = pywrapcp.RoutingModel(manager)

def distance_callback(from_index, to_index):
    """Returns the distance between the two nodes."""
    # Convert from routing variable Index to distance matrix NodeIndex.
    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)

routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)

search_parameters = pywrapcp.DefaultRoutingSearchParameters()
search_parameters.first_solution_strategy = (
    routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC
)

def print_solution(manager, routing, solution):
    """Prints solution on console."""
    print(f"Objective: {solution.ObjectiveValue()} miles")
    index = routing.Start(0)
    plan_output = "Route for vehicle 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, 0)
    plan_output += f" {manager.IndexToNode(index)}\n"
    print(plan_output)
    plan_output += f"Route distance: {route_distance}miles\n"
```

In [ ]:

```
solution = routing.SolveWithParameters(search_parameters)
if solution:
    print_solution(manager, routing, solution)
```

```
Objective: 18 miles
Route for vehicle 0:
 0 -> 1 -> 3 -> 4 -> 2 -> 0
```

Результаты совпали, следовательно решение можно считать верным

Реализация муравьиного алгоритма

Реализация была взята из следующего репозитория https://github.com/ppoffice/ant-colony-tsp

In [ ]:

```
import random
import numpy as np

class Graph(object):
    def __init__(self, cost_matrix: list, rank: int):
        """
        :param cost_matrix:
        :param rank: rank of the cost matrix
        """
        self.matrix = cost_matrix
        self.rank = rank
        # noinspection PyUnusedLocal
        self.pheromone = [[1 / (rank * rank) for j in range(rank)] for i in range(rank)]


class ACO(object):
    def __init__(self, ant_count: int, generations: int, alpha: float, beta: float, rho:
                 strategy: int):
        """
        :param ant_count:
        :param generations:
        :param alpha: relative importance of pheromone
        :param beta: relative importance of heuristic information
        :param rho: pheromone residual coefficient
        :param q: pheromone intensity
        :param strategy: pheromone update strategy. 0 - ant-cycle, 1 - ant-quality, 2 -
        """
        self.Q = q
        self.rho = rho
        self.beta = beta
        self.alpha = alpha
        self.ant_count = ant_count
        self.generations = generations
        self.update_strategy = strategy

    def _update_pheromone(self, graph: Graph, ants: list):
        for i, row in enumerate(graph.pheromone):
            for j, col in enumerate(row):
                graph.pheromone[i][j] *= self.rho
                for ant in ants:
                    graph.pheromone[i][j] += ant.pheromone_delta[i][j]
```

```python
    # noinspection PyProtectedMember
    def solve(self, graph: Graph):
        """
        :param graph:
        """
        best_cost = float('inf')
        best_solution = []
        for gen in range(self.generations):
            # noinspection PyUnusedLocal
            ants = [_Ant(self, graph) for i in range(self.ant_count)]
            for ant in ants:
                for i in range(graph.rank - 1):
                    ant._select_next()
                ant.total_cost += graph.matrix[ant.tabu[-1]][ant.tabu[0]]
                if ant.total_cost < best_cost:
                    best_cost = ant.total_cost
                    best_solution = [] + ant.tabu
                # update pheromone
                ant._update_pheromone_delta()
            self._update_pheromone(graph, ants)
            # print('generation #{}, best cost: {}, path: {}'.format(gen, best_cost, bes
        return best_solution, best_cost


class _Ant(object):
    def __init__(self, aco: ACO, graph: Graph):
        self.colony = aco
        self.graph = graph
        self.total_cost = 0.0
        self.tabu = []  # tabu list
        self.pheromone_delta = []  # the local increase of pheromone
        self.allowed = [i for i in range(graph.rank)]  # nodes which are allowed for the
        self.eta = [[0 if i == j else 1 / graph.matrix[i][j] for j in range(graph.rank)]
                    range(graph.rank)]  # heuristic information
        start = random.randint(0, graph.rank - 1)  # start from any node
        self.tabu.append(start)
        self.current = start
        self.allowed.remove(start)

    def _select_next(self):
        denominator = 0
        for i in self.allowed:
            denominator += self.graph.pheromone[self.current][i] ** self.colony.alpha *

        # noinspection PyUnusedLocal
        probabilities = [0 for i in range(self.graph.rank)]  # probabilities for moving
        for i in range(self.graph.rank):
            try:
                self.allowed.index(i)  # test if allowed list contains i
                probabilities[i] = self.graph.pheromone[self.current][i] ** self.colony.
                    self.eta[self.current][i] ** self.colony.beta / denominator
            except ValueError:
                pass  # do nothing
        # select next node by probability roulette
        selected = 0
        rand = random.random()
        for i, probability in enumerate(probabilities):
            rand -= probability
            if rand <= 0:
                selected = i
                break
        self.allowed.remove(selected)
        self.tabu.append(selected)
        self.total_cost += self.graph.matrix[self.current][selected]
        self.current = selected

    # noinspection PyUnusedLocal
```

```python
    def _update_pheromone_delta(self):
        self.pheromone_delta = [[0 for j in range(self.graph.rank)] for i in range(self.
        for _ in range(1, len(self.tabu)):
            i = self.tabu[_ - 1]
            j = self.tabu[_]
            if self.colony.update_strategy == 1:  # ant-quality system
                self.pheromone_delta[i][j] = self.colony.Q
            elif self.colony.update_strategy == 2:  # ant-density system
                # noinspection PyTypeChecker
                self.pheromone_delta[i][j] = self.colony.Q / self.graph.matrix[i][j]
            else:  # ant-cycle system
                self.pheromone_delta[i][j] = self.colony.Q / self.total_cost
```

In [ ]:
```python
cities = []
points = []
cost_matrix = [
    [0, 4.0, 5.0, 7.0, 5.0],
    [8.0, 0, 5.0, 6.0, 6.0],
    [3, 5, 0, 9, 6.0],
    [3.0, 5.0, 6.0, 0, 2.0],
    [6.0, 2.0, 3.0, 8.0, 0]
]
rank = 5
aco = ACO(10, 100, 1.0, 10.0, 0.5, 10, 2)
graph = Graph(cost_matrix, rank)
path, cost = aco.solve(graph)
print('cost: {}, path: {}'.format(cost, path))
```

```
cost: 18.0, path: [2, 0, 1, 3, 4]
```

Маршрут и стоимость маршрута совпали, следовательно реализация и решение верное.

Оценка устойчивости метаэвристики в зависимости от параметров

Так как в алгоритме начальная точка задается случайно, то вместо её оценки устойчивости рассмотрим такой параметр как число муравьев.

Число муравьев

Число муравьев является важным параметром в муравьином алгоритме

- **Важность числа муравьев**: Увеличение числа муравьев может улучшить исследование пространства решений. Больше муравьев может означать больше возможностей для параллельного исследования различных путей. И наоборот, слишком маленькое число муравьев может ограничить разнообразие исследуемых путей, что может снизить вероятность нахождения оптимального решения.
- **Устойчивость**: Обычно увеличение числа муравьев делает алгоритм менее зависимым от конкретного начального решения и более устойчивым к случайным вариациям.

Количество итераций

- **Важность**: Количество итераций определяет, насколько долго алгоритм может исследовать пространство решений. Большее количество итераций может улучшить вероятность нахождения более оптимального решения.
- **Устойчивость**: Муравьиный алгоритм обычно стабилен при различных значениях количества итераций, но слишком низкое количество итераций может привести к недостаточному исследованию пространства решений.