Assignment 4

Implement the following three algorithms described below. Each algorithm is worth up to five points. Solutions must be submitted by **24.5.2026**. Use the link on e-ucilnica to turn in your work. The report must be in `.pdf` format, with the provided `.py` file for code submission.

Continuous Optimization

This assignment is an introduction to continuous optimization using functions available in the `pymoo` Python package. This is also a warm-up for Assignment 5. By completing this assignment, you should have the groundwork for starting the next assignment.

The assignment consists of finding values at specific points in two-dimensional space for three functions, namely **Rastrigin**, **Sphere**, and **Ackley**. You need to implement three basic search algorithms, namely **grid search**, **random search**, and **first descent local optimization**.

Each algorithm must be implemented as a function in Python. You should submit a PDF report with the required information and your source code as a separate file for the algorithms.

This section describes the algorithms you are implementing. The appendix includes an example of how to run the selected functions in Python.

Grid Search

Implement a grid search algorithm to evaluate the three selected two-dimensional functions on a discrete grid of points with a grid size of 1, within the specified bounds for each function. The point $(0, 0)$ should **always** be included in the search. See the appendix or the provided Python code to see how to find the default bounds for each function.

For each of the selected functions, the report should include:

a) Number of points tested.

b) Coordinates and objective values for the minimum and maximum found.

Random Search

Implement a random search function that searches the two-dimensional space uniformly randomly within the specified bounds for each function.

For each of the selected functions, the report should include:

a) Mean objective value found over 1000 calls.

b) Coordinates and objective value for the minimum found.

Local Search

Implement a local search using **first descent**. This means that you move to the next solution as soon as the first neighbor you find is better than the current solution, instead of checking all neighbors and moving to the best one.

Let the algorithm run for a maximum of 1000 iterations with a neighborhood size of 100. Define a neighbor of a solution (x, y) as:

$$(x \pm \text{rand}(0.1),\ y \pm \text{rand}(0.1)),$$

where rand(0.1) returns a uniformly random number from 0 to 0.1. The initial solution, from which you start the search, should be generated uniformly randomly inside the bounds of the function.

The algorithm should stop after 1000 iterations or if it gets stuck in a local optimum. This means that none of the 100 generated neighbors are a better solution.

For each of the selected functions, run the algorithm 10 times and report:

a) Best coordinates and objective value found over the 10 runs.

b) Mean objective value found over the 10 runs.

c) For each run, report the local minimum found, the number of iterations before reaching the local minimum, and the number of calls to the objective function.

Appendix

Python Example

```python
import numpy as np
import pymoo.problems as problem
import pymoo.visualization.fitness_landscape as pymooviz

# Import 3 problems
p1 = [problem.get_problem("rastrigin", n_var=2), "Rastrigin"]
p2 = [problem.get_problem("sphere", n_var=2), "Sphere"]
p3 = [problem.get_problem("ackley", n_var=2), "Ackley"]

# Visualization of the selected problems
for p in [p1, p2, p3]:
    pymooviz.FitnessLandscape(
        p[0],
        angle=(45, 45),
        _type="surface"
    ).show()

point = np.array([
    [1, 0.5],
    [0, 0]
])

# Evaluating each function at specific points and printing bounds
for p in [p1, p2, p3]:
    print(p[1], ":")
    print(p[0].evaluate(point))
    print("Lower and upper bounds:")
    print(p[0].xl)
    print(p[0].xu)
    print(" ")
```