Assignment 4

Implement the following three algorithms described below. Each algorithm is worth up to five points. Solutions must be submitted by 15.5.2023. Use the link on e-ucilnica to turn in your work. The report must be in .pdf format and .R or .py for code submission.

Continuous optimization

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

The assignment consists of finding values at specific points in 2D space of three functions, namely **Rosenbrock**, **Ackley** and **Rastrigin**. 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 R or Python. You should submit pdf report with required information and your source code for the algorithms.

This section describes the algorithms you are implementing. The second sections has instruction on how to run the selected functions in R, and the last sections includes example on how to run the same package inside Python.

Grid search

Implement a grid search algorithm to evaluate the three selected 2D 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 next section for an example on how to find 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 2D 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**, which 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 the 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±rand(0.1), y±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.

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.

Installing and using the *smoof* package

To install and load the *smoof* package run the following two lines.

```
install.packages("smoof")
library(smoof)
```

The package *smoof* is now available in R. This package include a collection of test functions for continuous optimization. To use the functions we first need to create functions before we can use them.

Creating and running test functions

You need to create test functions first. The *smoof* package creates functions with predefined bounds and dimensions. Each functions has a know global optimum (minimum).

Lets make the selected three functions.

```
fun1 <- makeRosenbrockFunction(2)
fun2 <- makeAckleyFunction(2)
fun3 <- makeRastriginFunction(2)
```

This code generates two dimensional functions. If you want more dimensions just change the first parameter.

Running the functions. Lets find the value of the Rosenbrock function at point (1,2).

```
fun1(c(1,2))

## [1] 100
```

By using print() on the function you can see its basic info.

```
print(fun1)

## Single-objective function
## Name: 2-d Rosenbrock Function
## Description: no description
## Tags: single-objective, continuous, differentiable,
non-separable, scalable, multimodal
## Noisy: FALSE
## Minimize: TRUE
## Constraints: TRUE
## Number of parameters: 2
##            Type len Def   Constr Req Tunable Trafo
## x numericvector   2   - -5 to 10   -    TRUE     -
## Global optimum objective value of 0.0000 at
##   x1 x2
## 1  1  1
```

For this assignment pay attention to **Constr** value which represents the bounds of the search space.

We can easily optimize these functions using simulated annealing from *GenSA* package.

```
library(GenSA)
output <- GenSA(fn = fun1,
                lower = c(-5, -5),
                upper = c(10, 10),
                control=list(verbose=FALSE, max.time = 10))
output$par

## [1] 1 1

output$value

## [1] 3.759141e-20
```

You can also easily plot the functions using function plot3D() from package *plot3D*. The plots are only available for two dimensional functions. The plots of the three selected functions are available in appendix.

Running *smoof* in Python

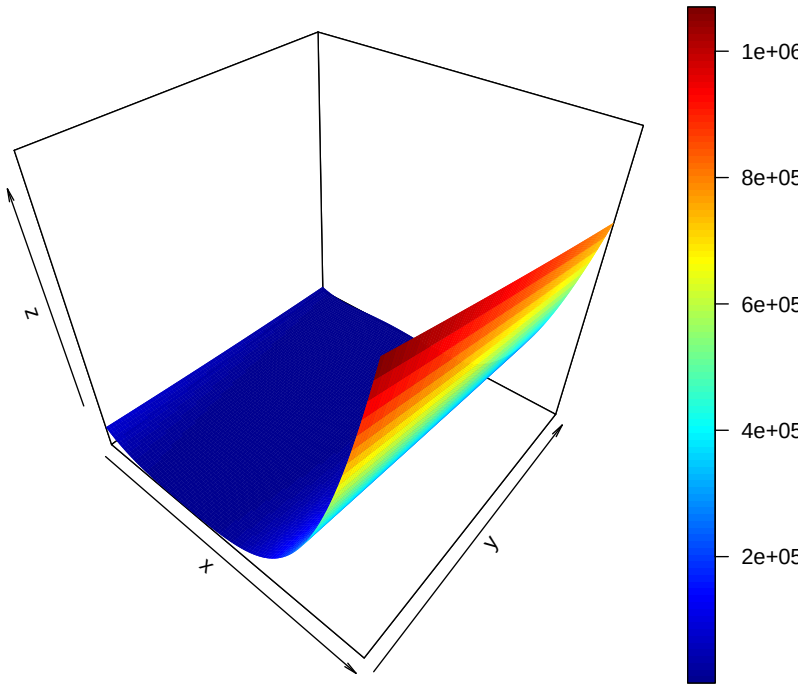
In this part we will embed the *smoof* package from R into Python using *rpy2* module. You can install it using "pip install rpy2". This packages allows you to run R code from python and requires R to be installed.

If you have *numpy* and *smoof* modules you can run the following example that shows the basic functionality similar to the previous section.

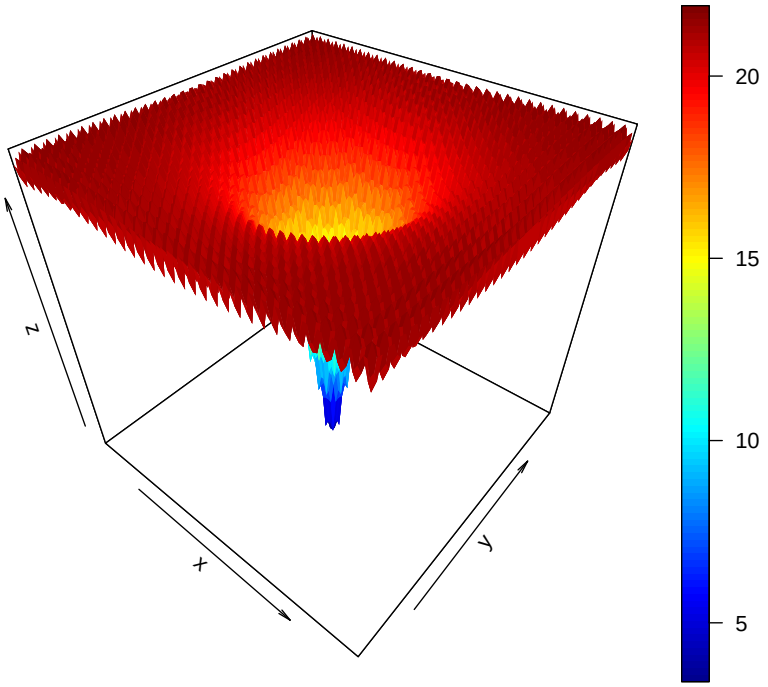
```
import numpy as np
from rpy2.robjects import numpy2ri
from rpy2.robjects.packages import importr

# Activate the automatic conversion from numpy to R arrays
numpy2ri.activate()

# Import the 'smoof' package
smoof = importr('smoof')

# Define the Rosenbrock function using smoof
rosenbrock = smoof.makeRosenbrockFunction(dimensions=2)

# Evaluate the function at a specific point
point = np.array([1, 2])
result = rosenbrock(point)
print("Rosenbrock function value at", point, ":", result[0])

# Print the basic information of the created function
print(rosenbrock)
```

Appendix - plots

```
plot3D(fun1)
```



```
plot3D(fun2)
```



```
plot3D(fun3)
```