Optimization Algorithms
Assignment 1

Joaquim Ortiz-Haro & Jung-Su Ha & Marc Toussaint
Learning & Intelligent Systems Lab, TU Berlin
Marchstr. 23, 10587 Berlin, Germany

Winter 2021/22

The first coding assignment includes three tasks:

a) Implement a nonlinear solver for unconstrained optimization fulfilling certain requirements (see below). (70 %)

b) Implement a mathematical program with a scalar cost function (15 %)

c) Implement a mathematical program with a sums of squares cost function (15 %)

Deadline: Monday 6.12.2021 at 11.55 am.

To work on the assignment, **update the submodule `optimization_algorithms` to the latest version:**

```
cd optimization_algorithms
git pull origin main
cd ..
git add optimization_algorithms
git commit -m "update submodule"
```

Note: If you haven't completed assignment 0, please check the README of https://git.tu-berlin.de/lis-public/oa-workspace on how to setup your working environment, and provide the https link of your repository in the questionnaire in ISIS.

1 Solver for unconstrained optimization

Implement a solver for unconstrained optimization problems of the form:

$$\min_{x \in \mathbb{R}^n} f(x) + \phi(x)^T \phi(x) \ , \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ and $\phi : \mathbb{R}^n \to \mathbb{R}^m$ are nonlinear functions (not necessarily convex). Your solver needs to fulfill the following requirements:

- On a purely quadratic cost f, it requires less than 10 queries to converge, independent of the dimensionality n
- On a linear least squares problem (where $\phi(x) = Ax+b$), it requires less than 10 queries to converge, independent of the dimensionality n
- It converges also on non-convex cost functions (if unimodal and downhill leads to convergence)
- The precision of the returned solution is $\|x - x^*\| \leq .001$, where x is the convergence point of the solver and x^* is the optimum of the problem (we only test on problems with unique optimum).

In addition, your solver should aim to

- minimize the number of queries (in each test problem, we will compare against our unconstrained solver)

- use a maximum of 1000 queries to the mathematical program and limit on the compute time (problem dependent).

The tests included in the assignment code are to help you to test the requirements.
The optimization problem is represented with an object of the class `MathematicalProgram`. Before starting the optimization, you can use the method `getFeatureTypes` to know which entries correspond to $f(x)$ and which entries correspond to $\phi(x)$. Here is some help to interpret the features and their types when querying the mathematical program:

```
problem = MathematicalProgram()
types = problem.getFeatureTypes()
index_f = [i for i, x in enumerate(types) if x == OT.f]  #get all features of type f
assert( len(index_f) <= 1 ) # at most, only one term of type OT.f
index_r = [i for i, x in enumerate(types) if x == OT.sos] #get all sum-of-square features
phi, J =  problem.evaluate(x)
#example to compute the overall cost:
c = 0
if len(index_f) > 0 :
  c += phi[index_f][0]
if len(index_r) > 0 :
  c += phi[index_r].T @ phi[index_r]
```

Getting started: As also detailed in the README, you should first copy the assignment code to your workspace folder:

```
cd oa-workspace
cp -R optimization_algorithms/assignments/a1_unconstrained_solver .
cd a1_unconstrained_solver
```

Your working path `a1_unconstrained_solver` now contains the template `solution.py` that you edit, and `test.py` to run a default test.
In this exercise, you need to implement the `solve` method of the `SolverUnconstrained` class in `solution.py`. The class is derived from `NLPSolver`, which you can check to get an overview of the interface.
Whenever you want to test your solver, run `python3 test.py`.

2 Mathematical Program with a general cost function

Implement a mathematical program to model the following problem:

The problem is to find the optimal placement $x \in \mathbb{R}^2$ of an antenna that maximizes the sum of potentials to the surrounding m villages at locations $p_j \in \mathbb{R}^2,\ j = 1, ..., m$, weighted by their population w_j,

$$\min_{x \in \mathbb{R}^2} -\sum_j w_j e^{-||x-p_j||^2} \ . \tag{2}$$

The initialization sample is the geometric center of the villages $\frac{1}{m}\sum p_j$.

Getting started: Copy your working path:

```
cd oa-workspace
cp -R optimization_algorithms/assignments/a1_antenna_placement .
cd a1_antenna_placement
```

Implement the missing methods of the `AntenaPlacement` class in `solution.py`. This class is derived from `MathematicalProgram`, which defines our abstract interface to mathematical programs. Check the methods inside `a1_antenna_placement/test.py` and the code skeleton of `AntennaPlacement` to get an idea of the python interface. Whenever you want to test your mathematical program, run `python3 test.py`.

3 Mathematical Program with a Sum of squares cost

Implement a mathematical program to model the following problem:

The problem is to find the configuration $q \in \mathbb{R}^3$ of a 3-DOF planar robot that reaches the desired position $p^* \in \mathbb{R}^2$ with a tool. The robot has 3 degrees of freedom $q \in \mathbb{R}^3$, and the position of the tool is defined by the function $p(q) = (p_1(q), p_2(q))$ where

$$p_1(q) = \cos(q_1) + \frac{1}{2}\cos(q_1 + q_2) + \frac{1}{3}\cos(q_1 + q_2 + q_3) \tag{3}$$

$$p_2(q) = \sin(q_1) + \frac{1}{2}\sin(q_1 + q_2) + \frac{1}{3}\sin(q_1 + q_2 + q_3) \tag{4}$$

$$\tag{5}$$

We add a small regularization with respect to a default configuration $q_0 \in \mathbb{R}^3$, with the term $\lambda\|q - q_0\|^2,\ \lambda \in \mathbb{R}_+$. The optimization objective is:

$$\min_{q \in \mathbb{R}^3} \|p(q) - p^*\|^2 + \lambda\|q - q_0\|^2 \tag{6}$$

$p^*,\ \lambda$ and q_0 are given. The initialization sample is q_0.

Note: the implementation should only contains terms of type sum of squares.

Getting started: Copy your working path

```
cd oa-workspace
cp -R optimization_algorithms/assignments/a1_robot_tool .
cd a1_robot_tool
```

Implement the missing methods of the `RobotTool` class in `solution.py`. This class is derived from `MathematicalProgram`, which defines our abstract interface to mathematical programs. Check the methods inside `a1_robot_tool/test.py` and the code skeleton of `RobotTool` to get an idea of the python interface. Whenever you want to test your solver, run `python3 test.py`.