Optimization Algorithms
Coding Assignment 2

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

Winter 2021/22

The second coding assignment includes three tasks:

a) Implement a nonlinear solver for constrained optimization using the Log Barrier method, fulfilling certain requirements (see below). (40 %)

b) Implement a nonlinear solver for constrained optimization using the Augmented Lagrangian algorithm fulfilling certain requirements (see below). (40 %)

c) Implement a mathematical program to model a linear dynamical system with quadratic cost (20 %)

Deadline: Monday 31.01.2022 at 11.55 pm.
NOTE: We recommend to use your unconstrained solver from assignment 1 to solve the inner augmented lagrangian and log barrier problems. **Caution!** The solution to each task should be self-contained in the corresponding `solution.py`, so you will have to copy the code of the unconstrained solver inside the new `solution.py`.

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 or assignment 1, 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: Log Barrier method

Implement a solver for constrained optimization problems of the form (1), using the Log Barrier method.

$$\min_{x\in\mathbb{R}^n} f(x) + \phi(x)^\top\phi(x) \ , \qquad \text{s.t } g(x) \le 0 \ , \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$, $\phi : \mathbb{R}^n \to \mathbb{R}^d$ and $g : \mathbb{R}^n \to \mathbb{R}^m$ are nonlinear functions (not necessarily convex). You can make the following assumptions:

- The staring point for the optimization is guaranted to be feasible, $g(x_0) \le 0$.
- The initial value of the log-barrier parameter $\mu = 1$.
- In the inner problem (optimization of the log barrier for fixed μ), you can approximate the Hessian of the constraint as zero, $\nabla^2 g_i = 0$. (But do not in general assume $\nabla^2 f(x) = 0$ and account for the least squares terms.)

Your solver needs to fulfill the following requirements:

- At the end of each outer iteration, x is feasible, i.e. $g(x) \le 0$. (We call outer iterations where you update μ.)
- The precision of the returned solution is $\|x - x^*\| \le .001$, 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).
- The returned solution x is feasible, i.e. $g(x) \le 0$.

In addition, your solver should aim to

- minimize the number of queries (in each test problem, we will compare against our unconstrained solver), and
- use a maximum of 10000 queries and limit 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 $f(x)$, $\phi(x)$ or $g(x)$.

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/a2_interior_point .
cd a2_interior_point
```

Your working path `a2_interior_point` 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 `SolverInteriorPoint` 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 Solver: Augmented Lagrangian

Implement a solver for constrained optimization problems of the form (2), using the Augmented Lagrangian method.

$$\min_{x \in \mathbb{R}^n} f(x) + \phi(x)^\top \phi(x) \ , \tag{2}$$

$$\text{s.t } g(x) \leq 0 \ , \tag{3}$$

$$h(x) = 0 \ . \tag{4}$$

where $f : \mathbb{R}^n \to \mathbb{R}$, $\phi : \mathbb{R}^n \to \mathbb{R}^d$, $g(x) : \mathbb{R}^n \to \mathbb{R}^m$ and $h(x) : \mathbb{R}^n \to \mathbb{R}^l$ are nonlinear functions (not necessarily convex). You can make the following assumptions:

- (No assumption on the starting point x_0 – it can be either feasible or infeasible.)
- The initial value of the penalty parameter is $\mu = 1$, and of the Lagrange multipliers $\lambda = 0, \kappa = 0$.
- In the inner problem, you can approximate the Hessian of the constraints as zero, $\nabla^2 g_i = 0, \nabla^2 h_i = 0$. (But do not in general assume $\nabla^2 f(x) = 0$, account for the least squares terms, and the overall Hessian of the Augmented Lagrangian.)

Your solver needs to fulfill the following requirements:

- 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).
- The returned solution is at most $\epsilon = 0.001$ infeasible, i.e. $\max(g(x), |h(x)|) \leq \epsilon$.

In addition, your solver should aim to

- minimize the number of queries (in each test problem, we will compare against our unconstrained solver), and
- use a maximum of 10000 queries to and limit 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 $f(x), \phi(x), g(x)$ and $h(x)$.

```
problem = MathematicalProgram()
types = problem.getFeatureTypes()
f_index = [i for i in range(len(types)) if types[i]==OT.f]
sos_index = [i for i in range(len(types)) if types[i]==OT.sos]
eq_index = [i for i in range(len(types)) if types[i]==OT.eq]
ineq_index = [i for i in range(len(types)) if types[i]==OT.ineq]
```

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/a2_augmented_lagrangian .
cd a2_augmented_lagrangian
```

Your working path `a2_augmented_lagrangian` 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 `SolverAugmentedLagrangian` 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`.

3 Mathematical Program: Optimal Control

Implement a mathematical program to model an optimal control problem with linear dynamics and quadratic cost, known as Linear Quadratic Regulator (LQR) with terminal constraints:

$$\begin{aligned}
&\min \frac{1}{2}\sum_{0}^{K-1} u(k)^\top R u(k) + \frac{1}{2}\sum_{1}^{K} y(k)^\top Q y(k), \qquad (5)\\
\text{s.t}\quad & y(1) = Bu(0),\\
& y(k+1) = A\, y(k) + B\, u(k)\ , \quad k = 1, \dots, K-1,\\
& y(K) = y_f.
\end{aligned}$$

where the variables are the states $y(k) \in \mathbb{R}^n,\ k = 1, \dots, K$ and the controls $u(k) \in \mathbb{R}^n,\ k = 0, \dots, K-1$.

Note: The optimization variable should be: $\mathbf{x} \in \mathbb{R}^{2nK} = [u(0), y(1), u(1), y(2), ..., u(K-1), y(K)]$. The implementation should only contain one cost objective of type `OT.f` and any number of objectives of type `OT.eq`

The input parameters of the mathematical program are $Q, R \in \mathbb{R}^{n\times n}$ (with $Q^\top = Q, R^\top = R$), $A, B \in \mathbb{R}^{n\times n}$, $y_f \in \mathbb{R}^n$ and $K \in \mathbb{N}$. The dimension of the optimization variable and the number of equality constraints depend on n and K. To get started, you should first copy the assignment code to your workspace folder:

```
cd oa-workspace
cp -R optimization_algorithms/assignments/a2_control .
cd a2_control
```

Your working path `a2_control` 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 methods of the `LQR` class in `solution.py`. The class is derived from `MathematicalProgram`, which you can check to get an overview of the interface.

As an example, you will find the solution to the exercise *Minimum fuel optimal control* (exercise sheet 7) in the folder `mathematical_programs/minimal_fuel.py`. Whenever you want to test your program, run `python3 test.py`.