Optimization Algorithms
Coding Assignment 2

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

Winter 2023/24

The second coding assignment includes three tasks:

a) Implement a nonlinear solver for unconstrained optimization (40 %)

b) Implement a nonlinear solver for constrained optimization using the log barrier method (40 %)

c) Implement a mathematical program to model a constrained optimization problem (20 %)

Deadline: Wednesday 13.12.2023 at 11.55 pm.

To work on the assignment, first check that the remote `upstream` is pointing to our repository.

```
git remote -v
```

You should see two remotes, similar to:

```
origin        git@git.tu-berlin.de:jortizdeharo/optimization_algorithms_w23.git (fetch)
origin        git@git.tu-berlin.de:jortizdeharo/optimization_algorithms_w23.git (push)
upstream        https://git.tu-berlin.de/lis-public/optimization_algorithms_w23 (fetch)
upstream        https://git.tu-berlin.de/lis-public/optimization_algorithms_w23 (push)
```

Otherwise, follow the instructions in `readme` in Coding Assignments in ISIS.

Now, you can merge our repository with your fork. If you have only modified the files in `assignments/a1_###`, the merge should be automatic (git will use the recursive strategy and it will ask for a commit message). Otherwise, you have to make sure that, after the merge, all files outside `assignments/a1_###` are exactly the same as in our repository.

```
git fetch upstream
git merge upstream/main
```

1 Solver: Unconstrained Optimization

Implement a solver for unconstrained optimization problems:

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

where $f : \mathbb{R}^n \to \mathbb{R}$ is a nonlinear function (not necessarily convex). Your solver can query the cost function $f(x)$, the gradient $\nabla f(x)$, and the Hessian $\nabla^2 f(x)$. It needs to fulfill the following requirements:

- On a quadratic convex cost f, 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 $f(x_{\text{out}}) - f(x^*) \leq 0.001$, where x_{out} is the convergence point of the solver and x^* is the optimum of the problem (we only test on problems with unique optimum).
- It requires a maximum of 1000 queries to the mathematical program.

A successful solver should use Newton steps when appropriate, be robust against non convexity and use backtracking line search. You have to modify the function `solve` in `assignments/a2_unconstrained/solution.py`. This function takes as input an object of class `NLP` (nonlinear program) and returns a local optima x_{out}. You will find some comments to guide you in the implementation.

The file `test.py` contains some tests to check if the solver is able to optimize some nonlinear programs we have prepared for you. Once you submit your code, we will evaluate the solver with similar problems. When you want to test your solver, run

```
cd assignments/a2_unconstrained
python3 test.py
```

You can also run a single test with, e.g,

```
python3 test.py testSolverUnconstrained.testHole
```

You can also modify `assignments/a2_unconstrained/play.py` to evaluate and check your implementation.

```
cd assignments/a2_unconstrained
python3 play.py
```

Once you are done, stage, commit and push `assignments/a2_unconstrained/solution.py`.

2 Solver: Log Barrier Method

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

$$\begin{aligned} &\min_{x \in \mathbb{R}^n} f(x) \ , \\ &\text{s.t } g(x) \leq 0 \ , \end{aligned} \tag{2}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ 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 guaranteed to be feasible, $g(x_0) \leq 0$.
- The initial value of the log-barrier parameter is $\mu = 1$.

- In the inner problem (optimization of the log barrier for a fixed μ), you can approximate the Hessian of the constraint as zero, i.e. $\nabla^2 g_i = 0$. Do not assume that the Hessian of the cost $f(x)$ or the log barrier $\log(-g_i(x))$ is zero or the identity matrix.

Your solver needs to fulfill the following requirements:

- The precision of the returned solution is $f(x_{\text{out}}) - f(x^*) \leq 0.001$, where x_{out} 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_{out} is feasible, i.e. $g(x_{\text{out}}) \leq 0$.
- Use a maximum of 10000 queries to the NLP.

You have to modify the function `solve` in `assignments/a2_log_barrier/solution.py`. This function takes as input an object of class `NLP` (nonlinear program) and returns a local optima x_{out}. You will find some comments to guide you in the implementation.

The file `test.py` contains some tests to check if the solver is able to optimize some nonlinear programs we have prepared for you. Once you submit your code, we will evaluate the solver with similar problems. When you want to test your solver, run

```
cd assignments/a2_log_barrier
python3 test.py
```

You can also modify `assignments/a2_log_barrier/play.py` to evaluate and check your implementation.

```
cd assignments/a2_log_barrier
python3 play.py
```

Once you are done, stage, commit and push `assignments/a2_log_barrier/solution.py`.

NOTE: your code should be self-contained in `assignments/a2_log_barrier/solution.py`. If you want to reuse some code of other coding assignments, you should copy and paste the code, instead of importing from other files.

3 Constrained Optimization Problem

Implement the optimization problem (3) using the `NLP` interface:

$$\min_{x \in \mathbb{R}^n} -\exp(-(x - x_0)^T D(x - x_0)) \;, \quad \text{s.t. } Ax \leq b \;, \tag{3}$$

where exp is the exponential function, $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m, D \in \mathbb{R}^{n \times n}$ symmetric, and $x_0 \in \mathbb{R}^n$.
You have to modify the missing methods in `assignments/a2_gaussian_ineq/solution.py`. Comments in the file provide more detailed instructions.

You can test and run your code with:

```
cd assignments/a2_gaussian_ineq
python3 test.py
python3 play.py
```

Once you are done, stage, commit and push `assignments/a2_guassian_ineq/solution.py`.