Optimization Algorithms
Coding Assignment 3 and 4

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

Winter 2023/24

Note: This year we have merged the last two coding assignments into a single exercise. When computing the final grade for the coding assignments, assignment 3 and 4 are counted independently. That is, the final score will be $(a_1 + a_2 + a_3 + a_4)/4$, where a_i is the score of assignment i.

The third coding assignment has a single task:

a) Implement a nonlinear solver for constrained optimization using the Augmented Lagrangian algorithm (100 %)

The fourth coding assignment has three tasks:

a) Compute the gradient of the solution of a parametric Quadratic Program (50 %)

b) Evaluate the impact of the initial step size and decay rate in stochastic gradient descent (25 %)

c) Implement the ADAM algorithm for stochastic optimization (25 %)

Deadline: Thursday 08.02.2024 at 11.55 am.

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:joaquimortizdeharo/optimization_algorithms_w23.git (fetch)
origin        git@git.tu-berlin.de:joaquimortizdeharo/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.pdf`.

Now, you can merge our repository with your fork. If you have only modified the files in `assignments/a1_###`, `assignments/a2_###` 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_###`, `assignments/a2_###` are exactly the same as in our repository.

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

Coding Assignment 3

1 Solver: Augmented Lagrangian Method

Implement a solver for constrained optimization problems of the form (1), using the Augmented Lagrangian algorithm.

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \; & f(x) + ||r(x)||^2 \;, \\ \text{s.t. } & g(x) \leq 0 \;, \\ & h(x) = 0 \,, \end{aligned} \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$, $r : \mathbb{R}^n \to \mathbb{R}^{m_r}$, $g : \mathbb{R}^n \to \mathbb{R}^{m_g}$ $h : \mathbb{R}^n \to \mathbb{R}^{m_h}$ are nonlinear functions (not necessarily convex). You can make the following assumptions:

- The initial value of the penalty parameter is 10.
- The initial value of the Lagrange multipliers is 0.
- In the inner problem (optimization of the Augmented Lagrangian for fixed penalties and multipliers), you can approximate the Hessian of the constraint as zero, i.e. $\nabla^2 g_i = 0, \nabla^2 h_i = 0$. Do not assume that the Hessian of the cost or the squared penalty is zero or the identity matrix.

We recommend an incremental approach: first use gradient descent with line search to minimize the Augmented Lagrangian function – this should already pass some tests (you can modify the variable `FACTOR` in `test.py`). Once this is working, replace gradient descent with a newton/gauss-newton method.

Your solver needs to fulfill the following requirements:

- The cost of the returned solution is $f(x_{\text{out}})+||r(x_{\text{out}})||^2 \leq f(x^*)+||r(x^*)||^2+0.001$, 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 up to tolerance 0.01. That is,

$$\max_i g_i(x_{\text{out}}) \leq 0.01 \,, \quad \max_i \; |\, h_i(x_{\text{out}}) \,| \leq 0.01 \,.$$

- Use few queries to the NLP. The maximum number will be problem-dependent, based on the iterations of our implementation. Using the number of queries as termination criteria in your solver will not be considered a valid solution.

You have to modify the function `solve` in `assignments/a3_auglag/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/a3_auglag
python3 test.py
```

You can also modify `assignments/a3_auglag/play.py` to evaluate and check your implementation.

```
cd assignments/a3_auglag
python3 play.py
```

Once you are done, stage, commit and push `assignments/a3_auglag/solution.py`.

NOTE: your code should be self-contained in `assignments/a3_auglag/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.

Coding Assignment 4

2 Differentiable Optimization

Implement the optimization problem

$$\min_{x \in \mathbb{R}^2} \; c^T y^*(x) \; , \tag{2}$$

where $y^*(x)$ is the solution of the parametric Quadratic program (QP) (3)

$$\min_{y \in \mathbb{R}^2} ||y - x||^2 \text{ s.t. } Ay \leq b \; , \tag{3}$$

and $c \in \mathbb{R}^2$, $A \in \mathbb{R}^{4\times 2}$, and $b \in \mathbb{R}^4$ are given input parameters.
Hints and suggestions:

- For an input x, you could get $y^*(x)$ solving the QP (3) using an algorithm for constrained optimization, e.g. Augmented Lagrangian. To simplify the exercise, solving the QP is not required. Instead, you can query an oracle that provides the mapping $x \mapsto (y^*(x), \lambda^*(x))$ for a limited set of x. λ^* are the Lagrange multipliers of (3) at the optimum y^*.
- You should compute the gradient $\frac{d}{dx} y^*(x)$ using the optimality KKT conditions and the implicit function theorem. Assume that $y^*(x)$ is differentiable, i.e. the constraint activity does not switch.
- You should only implement the cost function and the gradient (but not the Hessian) of (2), using a single feature of type `OT.f`.

You will find some comments in `solution.py` to guide you in the implementation. Run and test your code with

```
cd assignments/a4_diff_opt
python3 test.py
python3 play.py
```

Once you are done, stage, commit and push `assignments/a4_diff_opt/solution.py`.

3 Solver: Stochastic Gradient Descent

Implement stochastic gradient descent $x' = x_k - \alpha_k \nabla f_i(x_k)$ using a learning rate with decay,

$$\alpha_k = \frac{\alpha_0}{(1 + \alpha_0 \lambda k)}, \tag{4}$$

where α_0, and λ are user-defined constants.
The input of the `solve` function is an object of a new class called `NLP_stochastic`, that represents a stochastic optimization problem of the form,

$$\min_x f(x) = \frac{1}{N} \sum_{i=1}^{N} f_i(x) \tag{5}$$

Importantly, you can only query the function value and gradient of one index at a time, i.e. $f_i(x), \nabla f_i(x)$ for some chosen i. `NLP_stochastic.evaluate_i` has two inputs: `x` (the variable) and `i` (the index of the cost function you want to query), and returns the value and gradient of $f_i(x)$. `NLP_stochastic.getNumSamples()` returns the value of N.

In your implementation, you need to choose an appropriate α_0 and λ, and a strategy/ordering to choose which index to query next. The solver should fulfill the following requirements:

- The number of calls to `NLP_stochastic.evaluate_i(x,i)` is limited to 10.000. The solver should return a solution before reaching the maximum number of queries. Using the number of queries as termination criteria is a valid solution.
- The returned solution x^* is close to the real optimum, with $||x^* - x_{\text{opt}}|| \leq 0.05$.

```
cd assignments/a4_sgd
python3 test.py
python3 play.py
```

Once you are done, stage, commit and push `assignments/a4_sgd/solution.py`.

4 Solver: Adam

Implement the ADAM Solver (see Lecture Slides - `Stochastic Gradient Descent`) to solve stochastic optimization problems of the form:

$$\min_x f(x) = \frac{1}{N} \sum_{i=1}^{N} f_i(x) \tag{6}$$

The input of the `solve` function is an object of a new class called `NLP_stochastic` (see Section 2).
The solver should fulfill the following requirements:

- The number of calls to `NLP_stochastic.evaluate_i(x,i)` is limited to 10.000. The solver should return a solution before reaching the maximum number of queries. Using the number of queries as termination criteria is a valid solution.
- The returned solution x^* is close to the real optimum, with $||x^* - x_{\text{opt}}|| \leq 0.05$.

```
cd assignments/a4_adam
python3 test.py
python3 play.py
```

Once you are done, stage, commit and push `assignments/a4_adam/solution.py`.