Optimization Algorithms
Coding Assignment 3

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

Winter 2022/23

The third coding assignment includes three tasks:

a) Implement a nonlinear solver for unconstrained optimization, including least squares terms (25 %)

b) Implement a nonlinear solver for constrained optimization using the Augmented Lagrangian algorithm (55 %)

c) Implement a mathematical program to model a problem with least squares (20 %)

Deadline: Wednesday 18.01.2023 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:joaquimortizdeharo/optimization_algorithms_w22.git (fetch)
origin        git@git.tu-berlin.de:joaquimortizdeharo/optimization_algorithms_w22.git (push)
upstream        https://git.tu-berlin.de/lis-public/optimization_algorithms_w22 (fetch)
upstream        https://git.tu-berlin.de/lis-public/optimization_algorithms_w22 (push)
```

Otherwise, follow the instructions in `coding-readme.pdf`.

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

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

1 Solver: Unconstrained Optimization with Gauss Newton

Implement a solver for unconstrained optimization problems of the form:

$$\min_{x \in \mathbb{R}^n} f(x) + ||r(x)||^2 \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ and $r : \mathbb{R}^n \to \mathbb{R}^m$ are nonlinear functions (not necessarily convex). Your solver can query the value $f(x)$, the gradient $\nabla f(x)$, and the Hessian $\nabla^2 f(x)$ of the term f, and the residuals $r(x)$ and its Jacobian $\frac{d}{dx} r(x)$. It needs to fulfill the following requirements:

- On a quadratic f and linear $r(x)$, it requires less than 20 queries to converge.
- It converges also on non-convex cost functions (if unimodal and downhill leads to convergence).
- The precision of the returned solution is $\|x - x^*\| \leq 0.001$, 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).
- It uses few queries to the NLP. The number of maximum available queries is problem-dependent, based on the iterations of our reference implementation. Using the number of queries as termination criteria in your solver will not be considered a valid solution.

A successful solver should use Newton steps when appropriate, use the Gauss-Newton approximation for the least-squares term, be robust against non convexity, and use backtracking line search. You have to modify the function `solve` in `assignments/a3_gauss_newton/solution.py`. This function takes as input an object of class `NLP` (nonlinear program) and returns a local optima x^*. 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_gauss_newton
python3 test.py
```

You can also modify `assignments/a3_gauss_newton/play.py` to evaluate and check your implementation.

```
cd assignments/a3_gauss_newton
python3 play.py
```

Once you are done, stage, commit and push `assignments/a3_gauss_newton/solution.py`.

2 Solver: Augmented Lagrangian Method

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

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

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

$$h(x) = 0 \tag{3}$$

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 precision of the returned solution is $\|x - x^*\| \leq 0.001$, 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).
- 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^*. 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.

3 Least Squares problem

Implement the optimization problem

$$\min_{q \in \mathbb{R}^4} \|p(q) - p^*\|^2 + \lambda \|q - q_0\|^2 \tag{4}$$

where $p(q) = [p_1, p_2]$ with:

$$p_1(q) = \cos(q_1) + \frac{1}{2}\cos(q_1 + q_2) + \left(\frac{1}{3} + q_4\right)\cos(q_1 + q_2 + q_3),$$
$$p_2(q) = \sin(q_1) + \frac{1}{2}\sin(q_1 + q_2) + \left(\frac{1}{3} + q_4\right)\sin(q_1 + q_2 + q_3),$$

and $p^* \in \mathbb{R}^2,\ \lambda \in \mathbb{R}$ and $q_0 \in \mathbb{R}^4$ are given parameters.
You have to formulate problem (4) as a least squares problem.

$$\min_{q \in \mathbb{R}^4} ||r(q)||^2 \tag{5}$$

with $r : \mathbb{R}^4 \to \mathbb{R}^6$. Your implementation should return the residual features $r(q)$ and the Jacobian $\frac{d}{dq} r(q)$ (not the cost directly!). You have to modify the missing methods in `assignments/a3_robot_tool/solution.py`. Comments

in the file provide more detailed instructions. Computing the Jacobian using finite differences is not allowed.

You can test and run your code with:

```
cd assignments/a3_robot_tool
python3 test.py
python3 play.py
```

Once you are done, stage, commit and push `assignments/a3_robot_tool/solution.py`.