

Optimization Algorithms

Coding Assignment 1

Joaquim Ortiz-Haro & Marc Toussaint

Learning & Intelligent Systems Lab, TU Berlin

Marchstr. 23, 10587 Berlin, Germany

Winter 2022/23

The first coding assignment includes three tasks:

- a) Implement a gradient descent solver with backtracking linear search (70 %)
- b) Implement a nonlinear program with a quadratic cost function (15 %)
- c) Implement a nonlinear program with a nonlinear cost function (15 %)

Soft deadline: Wednesday 09.11.2022 at 11.55 pm.

Hard deadline: Tuesday 15.11.2022 at 11.55 pm.

In the first assignment, the code of the students will be evaluated two times: once on the soft and once on the hard deadline. After the soft deadline, we will publish the preliminary results, and you will have the opportunity to resubmit.

The goal of the soft deadline is to check that you have followed the instructions in the README correctly, that you have submitted correctly and that we are able to run your code. After the soft deadline, you can contact us if you have any problem with the setup or the submission and need some help.

The hard deadline is strict. After the final evaluation, we cannot reevaluate the code or attend individual requests.

1 Gradient Descent Solver with Backtracking Line Search

The first task is to implement a solver that does gradient descent steps $x \leftarrow x - \alpha \nabla f(x)$, where the step size α at each iteration is chosen adaptively using backtracking line search (see slide 5 of Lecture Slides - Part 1: Line Search).

You have to modify the function `solve` in `assignments/a1_gradient_descent/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 (e.g. how to get the starting point x_0 , and how to query the function and gradient of the nonlinear program).

The precision of the returned solution x has to be $\|x - x^*\| < 0.001$, where x^* is the true local optima. Your solver should aim to minimize the number of function and gradient queries required to solve each problem. We will compare against our own implementation. Moreover, in the evaluation we will set a limit of 1000 queries to the `NLP` and a time limit.

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/a1_gradient_descent
python3 test.py
```

You can also modify `assignments/a1_gradient_descent/play.py` to evaluate and check your implementation.

```
cd assignments/a1_gradient_descent
python3 play.py
```

Once you are done, stage, commit and push `assignments/a1_gradient_descent/solution.py`.

2 Quadratic Cost

Implement the optimization problem (1) using the NLP interface:

$$\min_{x \in \mathbb{R}^n} x^\top C^\top C x, \quad (1)$$

for a given matrix $C \in \mathbb{R}^{m \times n}$ and $x \in \mathbb{R}^n$. You have to modify the missing methods in `assignments/a1_quadratic_function/solution.py`. Comments in the file provide more detailed instructions.

You can test and run your code with:

```
cd assignments/a1_quadratic_function
python3 test.py
python3 play.py
```

Once you are done, stage, commit and push `assignments/a1_quadratic_function/solution.py`.

3 Nonlinear Cost

Implement the optimization problem (2) using the NLP interface

$$\min_{x \in \mathbb{R}^n} \frac{1}{\|Cx\|^2}, \quad (2)$$

for a given matrix $C \in \mathbb{R}^{m \times n}$ and $x \in \mathbb{R}^n$, where $\|\cdot\|$ is the 2-norm. You have to modify the missing methods in `assignments/a1_nonlinear_function/solution.py`. Comments in the file provide more detailed instructions. You can test and run your code with:

```
cd assignments/a1_nonlinear_function
python3 test.py
python3 play.py
```

Once you are done, stage, commit and push `assignments/a1_nonlinear_function/solution.py`.