Stanford University

AA222/CS361: Engineering Design Optimization
Spring 2023

Prof. Mykel J. Kochenderfer • NVIDIA Auditorium • email: *mykel@stanford.edu*

PROJECT 1: UNCONSTRAINED OPTIMIZATION Due date: April 14, 2023 (5:00pm PT)

The purpose of this project is for students to code their own unconstrained optimization strategies. This project assumes students have either Julia1.7+ or Python3.6+ installed on their computer and have familiarized themselves with the basics of running scripts in these languages. Installation instructions, a project tutorial, and programming resources can be found on the Project 0 web page: `https://aa222.stanford.edu/projects/project-0/`.

1 Project Overview

This project involves a programming competition where you can implement any unconstrained optimization algorithm in Julia/Python. The submissions that get closest to the global optimum value (within the allotted function and gradient evaluations), win!

You will be implementing a function `optimize` that minimizes a function with a limited number of evaluations. The rules are as follows:

1. We provide you a function $f(x)$, its gradient $g(x)$, and a number of allowed evaluations n.
2. Each call to `f` counts as one evaluation, and each call to `g` counts as two evaluations.
3. The only external libraries allowed for your implementation of `optimize` are `numpy` in Python and `Statistics` in Julia. In addition to those, you may use any of the standard libraries of either language.
4. You can use different optimization strategies for each problem, since we pass you a string `prob` in the call to `optimize`.
5. You can base your algorithm on those found in the book or online, but you must give credit. The code you submit must be typed by you (please do not copy/paste).
6. Although you may discuss your algorithm with others, you must not share code.
7. You cannot return hard-coded minima. If we find that your code is doing so, you will get a 0 for that problem.

2 Project Instructions

The following instructions assume that the `AA222Project1` folder is your working directory.

2.1 Choose a programming language

Pick either Julia or Python as a programming language. Depending on your choice, go to `language.txt` and change `notalanguage` to either `julia` or `python`.

2.2 Complete the required code

If you chose Julia, go to `project1_jl/project1.jl` and complete the function `optimize`. If you chose Python, go to `project1_py/project1.py` and complete the function `optimize`. To get full credit on a given problem, your implementation must outperform random search on 55% of random seeds.

2.3 Test your completed code

If you chose Julia test your completed code by running:

```
julia --color=yes localtest.jl
```

(see the file for more details) If you chose Python, test your completed code by running:

```
python3 localtest.py
```

You should see `Pass: optimize does better than random search on [problem].` for all the simple problems.

2.4 Prepare your `README.pdf`

In addition to the programming aspect, you are also required to submit (also on Gradescope) a PDF writeup, worth 50% of the assignment. It should contain the following information:

A description of the method(s) you chose. A plot showing the path for Rosenbrock's function with the objective contours and the path taken by your algorithm from three different starting points of your choice. Convergence plots for the three simple functions (Rosenbrock's function, Himmelblau's function, and Powell's function). Each plot should have the iterations on the x-axis and the function value on the y-axis. You can select a few initial points to start from (1-3) and plot them on top of one another. Note: the starter code for plotting is not provided.

2.5 Create the code submission

Create your `project1` submission by running

```
bash ./make_submission.sh
```

from the `starter_code` directory if you are on a Mac/Linux. On windows, run the following from a GitBash terminal (see Project 0 for download instructions):

```
bash ./make_submission_gitbash.sh
```

2.6 Submit on Gradescope

1. Submit the created zip file (`project1.zip`) on Gradescope/AA222/Project1
2. Submit your `README.pdf` on Gradescope/AA222/Project1 Writeup

3 Objective Functions

Three of the five objective functions are explicitly listed below and included in the starter code. Your algorithm will be tested and ranked on two additional secret functions, which are described below. Note: We are only revealing these problem descriptions to spark your interest. Knowledge of the problem domains is not necessary to be able to complete this project, and they can be solved with algorithms covered in this class.

1. Rosenbrock's function (the minimum is in a parabolic valley):
   - $f = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$
   - $x^* = (1, 1)^\top, \quad f^* = 0$
   - Max number of evaluations = 20
2. Himmelblau's function (multiple optima):
   - $f = (x_1^2 + x_2 - 11)^2 + (x + y^2 - 7)^2$

- $x^* = \begin{cases} (3,2)^\top \\ (-2.805, 3.131)^\top \\ (-3.779, -3.283)^\top \\ (3.584, -1.848)^\top \end{cases}, \quad f^* = 0$
- Max number of evaluations $= 40$

3. Powell's function (the Hessian matrix is singular at the optimum):
    - $f = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4$
    - $x^* = (0,0,0,0)^\top, \quad f^* = 0$
    - Max number of evaluations $= 100$
4. **Secret 1**:
    - In this problem, we are given synthetic driving data and fit parameters of a highway driving model.
    - The input $x \in \mathbb{R}^3$ holds the logarithm of three driving model parameters
    - Max number of evaluations $= 30$
5. **Secret 2**:
    - This one is just a secret!
    - This is a two-dimensional problem
    - Max number of evaluations $= 60$

4 Frequently Asked Questions

My strategy involves randomness. Are the scores averaged with different random seeds?

Yes! The scores are averaged over 500 runs with different seeds.

Where are the Hessians?

We are not providing the Hessian function for you to use. But feel free to estimate it with calls to `f` and `g`. The cost would depend on the number of calls to `f` and `g` you end up making. Note that you are not allowed to implement and use the analytical Hessians for problems.

How many submission can we make?

Unlimited!

Can we exceed the max number of evaluations when making the plots for the README?

Yes! In python, you can get around the assertion error by calling the `problem.nolimit()` method to allow infinite evaluations.

How long does the autograder take to grade?

It shouldn't take more than 5 minutes to grade. If your submission times-out during grading, please contact us on Ed.

How are leaderboard scores computed?

All of the problems are designed to have an optimal value of 0. The closer you are to 0, the closer you are to winning! The total score is the sum of all 5 problems (all of the problems are weighted the same).

Can I write code outside of the optimize function?

Yes, you can organize your code however you want as long as, at the end of the day, optimize works as described. Note that if you decide to create additional files, please make sure to import/include (python/julia respectively) them in your `project1` file, or else they won't be available to the autograder.

Can I change the starter code files?

Yes, they are not used in the autograder, so you have complete ownership over them.

Do I have to manually keep track of how many times the functions have been called?

Nope, we've decided to be very generous and provide a method for doing just that.

In Julia: `count(f)` will return how many times the function `f` has been called. For convenience, `count(f, g)` will give `count(f) + 2*count(g)`. Example:

```
function optimize(f, g, x0, n, prob)
    while count(f, g) < n
        # ... do some optimization
    end
    return x_best
end
```

In Python: the optimize function has an additional input argument `count`, which takes no arguments and evaluates to `f + 2g`. Example:

```
def optimize(f, g, x0, n, count, prob):
    while count() < n:
        # ... do some optimization
    return x_best
```

My README plots require an optimization path, but `optimize` only returns the optimum itself!

optimize is geared towards the autograder's evaluation of your method, so it only returns the final point. However, if you're coding with the README in mind (which is a good idea!), you may want to design your code in a way that is conducive to both requirements by writing methods thats collect the optimization history. See the following julia example:

```
function optimize(f, g, x0, n, prob)
    if prob == "simple1"
        x_history = some_method(f, g, x0, n)
    else
        x_history = some_other_method(f, g, x0, n)
    end
    return last(x_history)
end
```