EECS 498: Introduction to Algorithmic Robotics
Fall 2020
Homework Assignment #2
Due October 7th at 2:59pm

Rules:

1. **All homework must be done individually, but you are encouraged to post questions on Piazza.**
2. No late homework will be accepted.
3. The goal of this homework is to develop your understanding of optimization methods. You should use python to implement solutions. You may not use any other language, only python will be accepted.
4. Submit your python files in a `zip` along with a `pdf` of your answers to Gradescope. Do not paste your code into your `pdf`.
5. Remember that copying-and-pasting code from other sources is not allowed.

Questions

1. (5 points) Is a single point convex? Use the definition of convexity to justify your answer.
2. (10 points) Is the function $f(x) = (|2 - 5x| + 2x + 8e^{(-4x)}) - 1$ convex? Use the principles of composition and the common convex functions shown in the lecture to justify your answer.
3. (10 points) For the following equation:

   $$f(x) = e^{(0.5x+2)} + e^{(-0.5x-0.5)} + 4x$$

   Use the Symmetric Difference Quotient to compute the numerical gradient at $x = 5$ for $h = 0.001$, $h = 0.0001$, $h = 0.00001$, and $h = 0.000001$. Compute the gradient analytically and evaluate it at $x = 5$. Which numerical gradients are larger than the analytical gradient and which are smaller? Which h value gives the closest numerical gradient?
4. (10 points) Rewrite the following optimization problem in standard form ($x = [x_1, x_2]^T$):

   $$\begin{aligned} \underset{x}{\text{maximize}} \quad & -2x_2 + 3x_1 - 1 \\ \text{subject to} \quad & x_2 \geq -3, \\ & x_1 - 2 \geq x_1 - 5x_2, \\ & -x_2 + 2.3 = x_1, \\ & -x_1 + 2 + 4x_2 \leq 4x_1. \end{aligned}$$

5. (15 points) Consider $f(x) = \max\{3x^2 - 2, 2x - 1\}$. At what value(s) of x will the subdifferential $\delta f(x)$ contain more than one subgradient? What is $\delta f(x)$ at each such x value?

6. A linear program is defined as:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & c^T x \\ \text{subject to} \quad & Gx \leq h, \\ & Ax = b. \end{aligned}$$

   a. (10 points) Write down the Lagrange dual function for this problem and define the variables.

   b. (5 points) Write down the dual problem for this LP.

   c. (5 points) Suppose you solved the dual problem and obtained an optimal value d^*. Assuming the primal is feasible and bounded, how does d^* relate to the solution of the primal problem p^*? Explain why.

Software

1. Install `Matplotlib`, which is a plotting package for python. Open a terminal and run the following command:

   ```
   sudo apt-get install python-matplotlib
   ```

   Documentation is here.

2. Download `HW2files.zip`. Included in `HW2files.zip` is a file called `plotter.py`. This shows a simple example of how to plot a function using `matplotlib`. Run this script and verify that it produces a plot. You can use the code in this script to help you debug your code in the implementation section.

3. Install cvxopt. Open a terminal and run the following commands:

   ```
   sudo apt-get install python-pip
   pip install cvxopt
   ```

   Documentation is here.

4. Included in `HW2files.zip` is a file `lp.py`. It implements the linear programming example shown here. Make sure this code runs and produces a solution.

Implementation

1. Descent Methods: Here you will implement two descent methods and compare them.

   a. (5 points) Implement backtracking line search for functions of the form $f(x) : \mathbb{R} \to \mathbb{R}$. Set $\alpha = 0.1$ and $\beta = 0.6$. Submit your code as `backtracking.py` in your `zip` file.

   b. (10 points) Implement the Gradient Descent algorithm. Use your backtracking line search implementation (with the same α and β as above) to compute the step length. Use $\epsilon = 0.0001$. Submit your code as `gradientdescent.py` in your `zip` file.

c. (15 points) Implement the Newton's Method algorithm. Use your backtracking line search implementation (with the same α and β as above) to compute the step length. Use $\epsilon = 0.0001$. Submit your code as `newtonsmethod.py` in your `zip` file.

d. (20 points) Run Gradient Descent and Newton's method on the following problem, starting at $x^{(0)} = 5$:

$$\underset{x}{\text{minimize}}\ f(x) = e^{(0.5x+1)} + e^{(-0.5x-0.5)} + 5x$$

Generate the following plots (this can be done using the `matplotlib` library). Include these plots in your `pdf` and the code to generate them as `plot_descents.py` in your `zip`. Remember to label the axes.

i. A plot showing the objective function over the interval $[-10, 10]$ (black) and the sequence of points generated by Gradient Descent (red) and Newton's Method (magenta).

ii. A plot showing the $f(x^{(i)})$ vs. i for Gradient Descent (red) and Newton's Method (magenta).

Explain which algorithm performed better in this example in terms of number of iterations and why.

2. Stochastic Gradient Descent: Open the `SGDtest.py` file included in `HW2files.zip`. Here you'll see that a function `fsum(x)` has been defined as a sum of functions `fi(x,i)` for $i = 1, ..., $`nFunctions`. The first and second derivatives of `fsum` and `fi` are also given.

a. (10 points) Implement the Stochastic Gradient Descent (SGD) algorithm as a separate file `sgd.py` (submit this in your `zip` file). You can call your algorithm from `SGDtest.py` to test it. Set the parameters in the following way:

- Set the step size $t = 1$. It's OK to use this large step size for this problem because ∇f_i is very small.
- Run the algorithm for 1000 iterations (this is the termination condition).
- Choose one random ∇f_i per iteration.
- Start at $x^{(0)} = -5$.

b. (10 points) Create a plot showing `fsum`$(x^{(i)})$ vs. i and insert it in the `pdf` and include the code to generate the plot as `plot_sgd.py` in your `zip`. Note that you should **not** evaluate `fsum` within the SGD loop because this will be very slow. Instead, store the $x^{(i)}$s you produce and evaluate `fsum` after you're done running the algorithm. Is `fsum`$(x^{(i)})$ always decreasing? Explain why or why not.

c. (10 points) Run SGD 30 times and compute the mean and variance of the resulting `fsum`(x^*). Run SGD with 750 iterations 30 times and compare the resulting mean and variance to what you got with 1000 iterations. Explain the results.

d. Now we will compare SGD with 1000 iterations to Gradient Descent and Newton's Method in terms of computation time. Use $x^{(0)} = -5$ and $\epsilon = 0.0001$ for both Gradient Descent and Newton's Method. To time how long an algorithm takes, see the timing code in `SGDtest.py`.

i. (10 points) Put the runtimes you get for the three algorithms in your `pdf`. Explain your results, i.e. why are you getting the order of times you get here.

ii. (5 points) Compare the three algorithms in terms of `fsum`(x^*). Which is the best and which is the worst? Explain why and explain if the difference is significant.

3. Implementing the Barrier Method: Before you start writing code, you will need to figure out some of the equations you will need. The solutions to the following should be included in the `pdf` you submit:

   a. (5 points) Write down the objective function used in the centering problem of the log barrier method (in the general case). Define the variables used in this function.

   b. (5 points) Write down the function for (a) in the case of a linear program. Define the variables used in this function.

   c. (5 points) Write down the derivative of the function for (b).

   d. (5 points) Write down the second derivative of the function for (b).

   e. (5 points) Write down the duality gap for the barrier method with log barrier functions. It should be in terms of `numplanes`, the number of hyperplanes, and `t`, the optimization "force" increment.

   The code: Included in `HW2files.zip` is a file called `barrier.py`. This file sets up, draws, and solves a 2-dimensional LP. The barrier method is heavily commented but some parts are missing. You will need to fill in those parts and run the code. You will need to go through the code carefully to understand what is going on in order to fill in the parts correctly. Please make sure not to change anything in the script except what is marked as `###YOUR CODE HERE###`. When you are finished and you run your code, it should produce an output that looks like Figure 1.

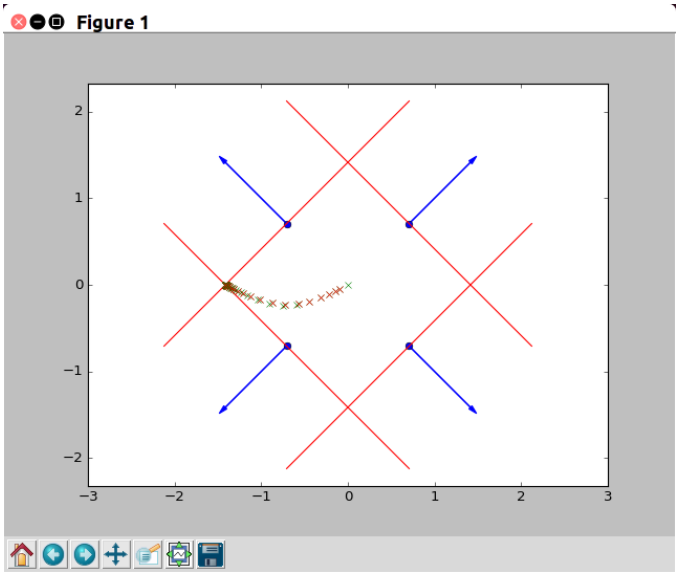

Figure 1: The red lines are the hyperplanes defining the constraints, blue arrows are the normals of these hyperplanes. Green xs are the x values computed during the inner loop iterations (i.e. the Newton iterations). Red xs are the values at every outer-loop iteration.

The script should also print out this message at the end:

The optimal point: (-1.413993, -0.000117)
Total number of outer loop iterations: 28

If you're not getting this solution (within a small tolerance), something is wrong with your implementation.

f. (35 points) Include your `barrier.py` in your `zip` file. We will test your code on a different problem to verify that it is correct (i.e. by changing the a,b, and c matrices). Do not assume the number of hyperplanes is constant. The problem will still be 2-dimensional. If your code does not run or crashes, you will receive 0 points for this part of the problem.

4. Formulating and solving an LP: Consider the robot team renting problem shown in lecture. We would like the robots to work at a construction site, which includes performing five tasks: `heavy lifting`, `materials transport`, `earth moving`, `concrete pouring`, and `brick laying`. There are four robots available. Their names, their price per hour, and their task-specific capabilities (per hour) are shown in the table below. Remember the simplifying assumption that robots can work on all tasks simultaneously.

Name	**Price per hour**	`heavy lifting`	`materials transport`	`earth moving`	`concrete pouring`	`brick laying`
SpiderBot P8	$75	1.6	3.5	0.1	2.3	6.1
Gigantimus Maximus	$128	7.2	2.1	7.1	3.2	0.1
VersaDroid X17	$70	3.7	3.2	2.9	3.4	4.9
HedonismBot	$34	0.1	0.15	0.1	0.15	0.1

For the particular construction project you are interested in, the task requirements are:

Task	**Amount Necessary**
`heavy lifting`	51
`materials transport`	48
`earth moving`	202
`concrete pouring`	120
`brick laying`	229

a. (25 points) Formulate an LP which outputs how long to rent each robot such that you minimize the total cost and get at least the amount of work necessary for each task. Write down your formulation in standard form in your `pdf`.

b. (15 points) Solve this LP using the `cvxopt` solver. Put the answer in the `pdf`. Include your code for solving the LP as a file called `robotrental.py` in your `zip` file.