Homework 6
APPM 4720/5720 Scientific Machine Learning, Fall 2024

Due date: Monday, Oct. 7 '24, before midnight, via Gradescope

Instructor: Prof. Becker
Revision date: 10/2/2024

Theme: A Toy Problem in Constrained Optimization

Instructions Collaboration with your fellow students is OK and in fact recommended, although direct copying is not allowed. The internet **is allowed for basic tasks** (e.g., looking up definitions on wikipedia, looking at documentation, looking at basic tutorials) but it is not permissible to search for solutions to the exact problem or to *post* requests for help on forums such as `http://math.stackexchange.com/`.

Setup Consider the constrained optimization problem

$$\min_{\boldsymbol{w}} \frac{1}{2}\|A\boldsymbol{w} - \boldsymbol{y}^{(1)}\|_2^2 \qquad \text{subject to} \quad B\boldsymbol{w} = \boldsymbol{y}^{(2)} \tag{P}$$

for $\boldsymbol{w} \in \mathbb{R}^d$, $A \in \mathbb{R}^{n_1 \times d}$ and $B \in \mathbb{R}^{n_2 \times d}$.

Problem 1: **Students in 5720 only** The problem (P) is simple enough that one can find a closed-form expression for it (involving matrix inverses). $\boxed{\text{Derive such a closed form expression.}}$ *Hint: you could try either a change of variables, or solving the KKT equations*

Problem 2: Let $d = 20$, $n_1 = 33$ (we can allow A to be over-determined) and $n_2 = 12$ (we want B to be under-determined... why?), and choose A and B to be realizations of iid standard normal matrices. Let $\boldsymbol{w}^{\text{signal}}$ be the all ones vector, and define $\boldsymbol{y}^{(1)} = A\boldsymbol{w}^{\text{signal}} + \sigma\boldsymbol{z}$ for $\sigma = 0.1$ and $\boldsymbol{z}$ iid standard normal, and define $\boldsymbol{y}^{(2)} = B\boldsymbol{w}^{\text{signal}}$.

Numerically find the solution to (P) using these parameters, via an optimization software package. Find a solution with as much accuracy as possible, i.e., 10^{-10} ballpark. $\boxed{\text{Turn in code showing your work.}}$

Suggestion: for Python, try the `cvxpy` package (pre-installed on colab, or install it yourself via `conda install -c conda-forge cvxpy`). This is actually an optimization framework that includes several bundled solvers. The default `SCS` solver is not super high accuracy, so I'd suggest something like `ECOS` for this problem.

Suggestion: for 5720 students, use this solver to check your answer from Problem 1.

Problem 3: Subproblems

a) If we solve (P) via the **penalty method** (with a given penalty parameter μ), write down the subproblem, and $\boxed{\text{turn in code showing how to solve the subproblem}}$.

   Note: In general one would solve the subproblem with an iterative method like gradient descent, but in this case you can solve it with linear algebra techniques.

b) If we solve (P) via the **Augmented Lagrangian** method (with a given penalty parameter μ and dual variable $\boldsymbol{\nu}$), write down the subproblem, and $\boxed{\text{turn in code showing how to solve the subproblem}}$.

   Note: Again, generally this would be iterative, but here it can be via linear algebra.

Problem 4: Since we're using linear algebra techniques for the penalty method, there's no need to "warm-start" the solver, hence the penalty method is as simple as solving the penalty method subproblem for a large value of μ. $\boxed{\text{Plot the error as a function of } \mu}$ for a reasonable range of μ values, where the error is the Euclidean norm of the difference between the **penalty method** solution $\boldsymbol{w}^{(\mu)}$ and the true solution from Problem 2.

Comment: If using an iterative method, then large values of μ might require many iterations. Since we're using a linear algebra method, can you think of any downsides of using a large μ?

Problem 5: Building on your code from part 3(b), write code that uses the **Augmented Lagrangian** method to solve (P), and turn in $\boxed{\text{two plots}}$: the first plot is the error vs μ (as in Problem 4) but using the Augmented Lagrangian method, and the second plot is the error vs the number of Augmented Lagrangian *iterations*.

Comment: You'll want your Augmented Lagrangian code to stop iterating after it reaches some kind of tolerance criterion.

Problem 6: **Students in 5720 only** Make a larger problem, with $d = 1000$, $n_1 = 2000$ and $n_2 = 100$. Find the true solution to (P) using the methods from Problem 1 or 2, and then try to find a solution using both the penalty method and the Augmented Lagrangian. Can you get a solution using these methods that has error (Euclidean norm of the different $\boldsymbol{w}$ solutions) less than 10^{-12}?

Going further: Because this is a toy problem and the solution is linear in the data, one could improve the linear algebra using *iterative refinement* ideas. No need to do this for the homework.