MATH-UA 252/MA-UY 3204 - Fall 2022 - Homework #1

Problem 1. A linear inequality constraint in 2D has the form:

$$a_1 x_1 + a_2 x_2 \leq b, \qquad \text{or} \qquad a^\top x \leq b, \tag{1}$$

where $a = (a_1, a_2)$ and $x = (x_1, x_2)$.

1. Make up a set of linear inequality constraints in 2D which define a closed and bounded polygon with **at least 6 sides**. What is the minimum number of constraints needed to do this? Write this constraint set using matrix notation. Draw this set and clearly label the constraints.
2. Next, give **two** examples of linear inequality constraints which can be added that are *redundant*—i.e., adding them to the constraint set does not change the polygonal set. Include them in your picture.
3. Give an example of a linear cost function which, when combined with the constraint set, gives a linear program (LP) whose minimizer is one of the vertices of the polygonal domain. Now, choose a different linear cost function such that the minimizer corresponds to an entire *edge* of the polygon. What do you notice? Draw pictures of each of these LPs depicting the level sets of the cost functions and the minimizers. (You do not need to relabel the constraints or include the redundant constraints in these pictures.)
4. How could you modify the constraint set so that your two LPs no longer have an optimum—i.e., the LPs are *unbounded below*, and have no minimizing value for a finite argument. Draw a picture.

Note: you should have **four** drawings—one for Problems 1 and 2, one for Problem 3, and one for Problem 4.

Problem 2. A convex polyhedron in $\mathbb{R}^3$ is a polyhedron P such that for each $x, y \in P$, $(1-\lambda)x + \lambda y \in P$ for all λ such that $0 \leq \lambda \leq 1$. (If you have never heard of a convex set before, take a minute to do an image search and familiarize yourself with what they look like.) One way of specifying a (closed) convex polyhedron is as the set of points which satisfy a system of m linear inequality in three variables:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &\leq b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &\leq b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + a_{m3}x_3 &\leq b_m. \end{aligned} \tag{2}$$

Note that the surface of a polyhedron can be partitioned into three types of objects: *vertices*, *edges*, and *faces* (or *facets*).

Assume that we have a polyhedron P defined this way, and that the system of linear inequality constraints is *minimal*—i.e., there are no redundant constraints. Give a simple brute force algorithm for finding all of the vertices of the polyhedron.

(*Hint:* It may help to start with a polygon in $\mathbb{R}^2$. What is true of a vertex? Use linear algebra.)

Problem 3. Use Python to solve the nonlinear least squares problem:

$$\text{minimize} \qquad \frac{1}{N} \sum_{i=1}^{N} (y_i - f(x_i, c_1, c_2))^2 \tag{3}$$

where:

$$f(x, c_1, c_2, c_3) = c_1 + c_2 e^{c_3 x}. \tag{4}$$

To do this, randomly choose (c_1, c_2, c_3) using `np.random`, pick N, sample N x_i's, and compute y_i by evaluating $f(x_i, c_1, c_2, c_3)$ for each x_i. To actually solve the nonlinear least squares problem, you should write a function that implements Gauss-Newton and use it to find (c_1, c_2, c_3). Next, add a small amount of noise to the observed y_i's. You can use `np.random.randn` times a small constant and solve the problem again. For each problem, make a plot of the predicted function, and evaluate the MMSE (the minimum mean squared error—just the cost function for this problem).

Go through the process described above for several different choices of parameters, different choices of N, and different noise intensities (vary the small parameter multiplying the output from `np.random.randn`). Write a short (~1 paragraph) explanation of your findings.

A word about Problem 5. Let $c^{(n)}$ be the nth iterate of your Gauss-Newton algorithm for computing the coefficients in the previous problem. You will generate a sequence of iterates:

$$c^{(0)}, c^{(1)}, c^{(2)}, \dots \tag{5}$$

This sequence *may not converge*. The choice of $c^{(0)}$ plays an important role. For this problem, what is a good choice of $c^{(0)}$? That is, how can we make an educated guess about the coefficients without solving the problem exactly? The length of each step that you take is also important. We will learn more about this in class later, but for the time being, a good heuristic you can try using is as follows. Assume that your iteration is of the form:

$$c^{(n+1)} = c^{(n)} + \Delta c^{(n)}, \qquad n \geq 0, \tag{6}$$

and let $F^{(n)} = F(c^{(n)})$ be the cost function evaluated at n. If $F^{(n+1)} > F^{(n)}$ ($\Delta c^{(n)}$ isn't a "*descent direction*"—i.e., moving from $c^{(n)}$ to $c^{(n+1)}$ fails to decrease the cost function), then you can try modifying the step length. Replace your iteration with:

$$c^{(n+1)} = c^{(n)} + \alpha^{(n)} \Delta c^{(n)}, \qquad 0 \leq \alpha^{(n)} \leq 1, \qquad n \geq 0. \tag{7}$$

A simple way to calculate the factor $\alpha^{(n)}$ is to iterate:

$$\alpha^{(n)} := \beta \alpha^{(n)}, \qquad 0 < \beta < 1, \qquad \alpha^{(0)} = 1 \tag{8}$$

until $F^{(n+1)} < F^{(n)}$. (For example, pick $\beta = 0.9$, then scale the step length by β repeatedly until we've found a descent direction.)