# CS 498 TC ✧ Computational Geometry ✧ Spring 2021

# Homework 4

Due Monday, April 12, 2020 at 8pm

---

1. Suppose you are given a set of $n$ halfplanes that define a convex polygon $P$ in the plane. Show that each of the following objects inside $P$ can be found in $O(n)$ expected time using linear programming. ("Axis-aligned" means all are edges either horizontal or vertical.)

   (a) The largest axis-aligned square inside $P$.

   (b) The axis-aligned rectangle inside $P$ with maximum perimeter.

   (c) Two interior-disjoint axis-aligned squares inside $P$ with maximum total perimeter.

   (d) The largest circle inside $P$.

2. In some applications of linear programming, we want to optimize several different objective functions for exactly the same set of constraints. Given a set $H$ of $n$ $d$-dimensional linear constraints (halfspaces in $\mathbb{R}^d$), we would like to preprocess $H$ into a data structure that supports *linear programming queries*. Given an arbitrary objective vector $c \in \mathbb{R}^d$, a linear programming query reports the solution to the linear program $\max\{c \cdot x \mid x \in \bigcap H\}$.

   (a) Describe an algorithm to answer ***two***-dimensional linear-programming queries in $O(\log n)$ time. How much space does your data structure use? How much preprocessing time do you need construct it?

   (b) Describe an algorithm ***three***-dimensional linear-programming queries in $O(\log n)$ time. How much space does your data structure use? How much preprocessing time do you need construct it?

   In both problems, assume that the feasible polyhedron $\bigcap H$ is both non-empty and bounded, and assume as usual that the input is in general position. Don't build your data structures or preprocessing algorithms from scratch; combine tools that we've already developed!

   *[Hint: Consider the standard projective duality $(a, b, c) \leftrightharpoons z = ax + by - c$. What is the dual of a linear-programming query?]*

3. Given points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ in the plane, the ***linear regression problem*** asks for real numbers $a$ and $b$ such that the line $y = ax + b$ fits the points as closely as possible, according to some criterion. The most common fit criterion is the ***$L_2$ error***, defined as follows:

$$\varepsilon_2(a, b) = \sum_{i=1}^{n} (y_i - ax_i - b)^2.$$

   Finding the line with minimum $L_2$ error is commonly known as *(ordinary) least squares* regression.

   But there are several other ways of measuring how well a line fits a set of points, *some* of which can be optimized via linear programming.

(a) The $L_1$ ***error*** (or *total absolute deviation*) of the line $y = ax + b$ is the sum of the vertical distances from the given points to the line:

$$\varepsilon_1(a,b) = \sum_{i=1}^{n} \left| y_i - ax_i - b \right|.$$

Describe a linear program whose solution encodes the line minimizing this error metric. (Your LP will *not* have fixed dimension.)

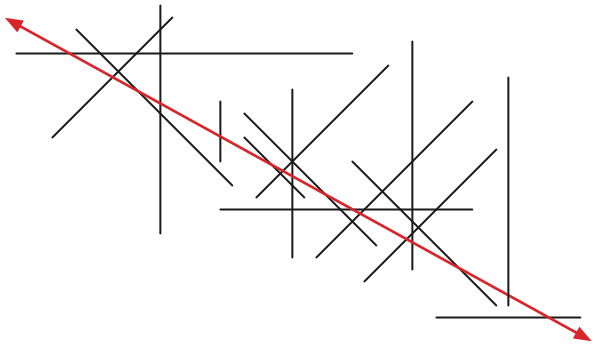
(a′) **Extra credit:** Describe a polynomial-time algorithm to compute the line minimizing this error metric. *[Hint: What does your linear program tell you about the structure of the optimal solution?]*

(b) The $L_\infty$ ***error*** (or *maximum absolute deviation*) of the line $y = ax + b$ is the maximum vertical distance from any given point to the line::

$$\varepsilon_\infty(a,b) = \max_{i=1}^{n} \left| y_i - ax_i - b \right|.$$

Describe a linear-time algorithm to find the line minimizing this error metric.

4. Let $S$ be a set of $n$ line segments in the plane, each parallel to one of $k$ different directions. Segments in $S$ may or may not intersect. A *stabbing line* for $S$ is a line that intersects every segment in $S$. Describe an algorithm to determine whether $S$ has a stabbing line. For full credit, your algorithm should run in $O(kn)$ time.[1] *[Hint: As a warmup, consider the case where $k = 1$ and all segments in $S$ are horizontal.]*



A line (red) stabbing segments with four directions.

[1] In fact, this problem can be solved in $O(n \log n)$ time even when $k = n$—that is, with no restriction on segment directions—but the algorithm is considerably more complicated.