# CSC279 HW3

Hanzhang Yin

Oct/2/2023

### Collaborator

Chenxi Xu, Yekai Pan, Yiling Zou, Boyi Zhang

## Question 10

### (PART A)

**Parametrizing the Segment pq:**
Let $p = (x_0, y_0)$ and $q = (x_1, y_1)$ be the endpoints of the segment $pq$. Any point $x(t)$ on $pq$ can be expressed as:

$$x(t) = (1-t)p + tq = ((1-t)x_0 + tx_1, (1-t)y_0 + ty_1),\ t \in [0,1]$$

**Dual Lines Corresponding to Points on pq:**
The dual line $\hat{x}(t)$ Corresponding to $x(t)$ is:

$$y = A(t)x - B(t)$$

where:

$$A(t) = (1-t)x_0 + tx_1$$
$$B(t) = (1-t)y_0 + ty_1$$

Now we can express rewrite the equation of $\hat{x}(t)$ as:

$$y = [x_0 + t(x_1 - x_0)]x - [y_0 + t(y_1 - y_0)]$$
$$= x_0 x - y_0 + t[(x_1 - x_0)x - (y_1 - y_0)]$$

Let:

$$D = (x_1 - x_0)x - (y_1 - y_0)$$

Then:

$$y = x_0 x - y_0 + tD$$

For a fixed $x$, $y$ varies linearly with $t$ from $y = x_0 x - y_0$ (when $t = 0$) to $y = x_1 x - y_1$ (when $t = 1$). The Union of all dual lines $\hat{x}(t)$ for $t \in [0,1]$ is the set of all points $(x, y)$ in the plane satisfying:

$$\min(y_0, y_1) \le y - x \cdot \min(x_0, x_1) \le \max(y_0, y_1)$$

Equivilantly, by eliminating parameter $t$, we can get similar inequality:
We aim to eliminate the parameter $t$ from the parametric equations to describe the union of dual lines $\hat{x}(t)$.
Solving for $t$:

$$t = \frac{y - x_0 x + y_0}{D}$$

**Note:** The sign of $D$ affects the inequality direction.
If $D > 0$, then

$$0 \leq \frac{y - x_0 x + y_0}{D} \leq 1 \Rightarrow 0 \leq y - x_0 x + y_0 \leq D$$

If $D < 0$, then

$$0 \geq \frac{y - x_0 x + y_0}{D} \geq 1 \Rightarrow D \leq y - x_0 x + y_0 \leq 0$$

Both cases can be unified by the product inequality:

$$(y - x_0 x + y_0)(y - x_1 x + y_1) \leq 0$$

This inequality describes the region between the lines $y = x_0 x - y_0$ and $y = x_1 x - y_1$, where the expressions $y - x_0 x + y_0$ and $y - x_1 x + y_1$ have opposite signs or are zero.
Hence, the union of all dual lines is:

$$(y - x_0 x + y_0)(y - x_1 x + y_1) \leq 0$$

This inequality describes all points $(x, y)$ that lie between $y = x_0 x - y_0$, $y = x_1 x - y_1$. which forms a region called a "double wedge".

## (PART B)

```
Input:
    S = { s_i | i = 1 to n }
output:
    l_max # Line has max. intersections point with other sections

Algorithm FindMaxIntersectingLine(S):
    Initialize an empty list L_dual_lines.

    For each segement s_i in S do:
        Let p_i = (x_{0i}, y_{0i}) and q_i = (x_{1i}, y_{1i})
        Compute the dual lines:
            L_{p_i}: y = x_{0i} x - y_{0i}
            L_{q_i}: y = x_{1i} x - y_{01}
        Add L_{p_i} and L_{q_i} to L_dual_lines.

    Construct the arrangement A of the lines in L_dual_lines:
        # TIME COMPLEXITY: O(n^2logn)
        Use the line sweep algorithm to compute the A.
        Store the faces, edges, and vertices of the A.

    Initialize count c_f = 0 for a starting PLANE f_0 at INFINITY

    Traverse arrangement A to label each face with the number
    of double wedges covering it:
        # TIME COMPLEXITY: O(n^2)
        For each edge e in A:
            Determine which double wedges have boundaries along e
            For each face f adjacent to e:
                *c_{f'} is the count of the adjacent face before crossing e
                If crossing e enters a double wedge W_i, then c_f = c_{f'} + 1
                If crossing e exits a double wedge W_i, then c_f = c_{f'} - 1

    Keep track of the face f_max with the maximum count c_max during traversal

    Let (a_max, b_max) be a point inside face f_max.

    Compute the line l_max in the primal plane corresponding to (a_max, b_max):
        l_max: y = a_max x - b_max

    Return l_max
```

**A more general description of the Algorithm:**
For this question, we convert each segment's endpoints into dual lines in the dual plane. The segments in the original plane correspond to **double wedges** (regions between two dual lines) in the dual plane. Our task becomes finding

a point in the dual plane covered by the maximum number of double wedges, which corresponds to a line in the original plane intersecting the most segments. Using the **line-sweeping paradigm**, we sweep a vertical line across the dual plane, tracking how many double wedges are active as we cross intersections between dual lines. For each intersection, we update the count by adding or removing wedges, and we maintain the maximum count throughout the sweep. The point where the count is highest gives the optimal line in the original plane.

# Question 11

```
Input:
    Simple polygon P given as a list of vertices [v1, v2, ..., vn] in order
Output:
    Whether there exists a line l such that P is monotone w.r.t. l

Algorithm DetermineMonotonicity(P):
    BadIntervals = empty_list()
    # TIME COMPLEXITY: O(n)
    For i from i to n:
        v_prev = P[(i - 2) mod n]
        v = P[i - 1]
        v_next = P[i mod n]
        Compute vectors e1 = v - v_prev
        Compute vectors e2 = v_next - v
        Compute cross product cp = e1.x * e2.y - e1.y * e2.x
        If cp < 0 (vertex is concave):
            Compute angles a1 = atan2(e1.y, e1.x) mod 2PI
            Compute angles a2 = atan2(e2.y, e2.x) mod 2PI

            Let Interval = [a2, a1] if a1 > a2 else [a2, a1 + 2PI]
            Normalize Interval to [0, 2PI)

            Add Interval to BadIntervals

    # TIME COMPLEXITY: O(nlogn)
    Sort BadIntervals by their start angles
    Merge overlapping intervals in BadIntervals to get a list of disjoint intervals

    If the merged intervals cover [0, 2PI):
        return FALSE

    return TRUE
```

**A more general description of the Algorithm:**
For each vertex $i$, determine if it is concave using the **counterclockwise (ccw)** test. If the ccw sign at the vertex differs from the majority, mark it as concave. For each concave vertex, compute two vectors:

$$c_0 = v(i) - v(i-1), \quad c_1 = v(i+1) - v(i).$$

Calculate their angles $a_0$ and $a_1$ using the `atan2` function, normalized to $[0, 2\pi)$:

$$a_0 = (\text{atan2}(c_0.y, c_0.x) + 2\pi) \bmod 2\pi, \quad a_1 = (\text{atan2}(c_1.y, c_1.x) + 2\pi) \bmod 2\pi.$$

Store the interval $[a_1, a_0]$ in a list.
After processing all vertices, sort the intervals and merge any overlapping ones.

If the merged intervals cover the full $2\pi$ range, the polygon is **not monotone**. Otherwise, it is **monotone**.

## Question 12

```
Input:
    RedPoints = [ (x1, y1), (x2, y2), ..., (xn, yn) ]
    BluePoints = [ (x1', y1'), (x2', y2'), ..., (xn', yn') ]
Output:
    Coefficients (a, b, c) defining the parabola y = a x^2 + b x + c
    or report "No solution exists" if impossible

Algorithm FindSeparatingParabola(RedPoints, BluePoints):
    Initialize an empty list Constraints = []

    # NOTE: e is a small positive number
    # TIME COMPLEXITY: O(n)
    For each red point (x_i, y_i) in RedPoints do:
        Make inequality:
            a (x_i^2) + b (x_i) + c - y_i < 0
        Convert to standard LP form (inequalities with <=):
            a (x_i^2) + b (x_i) + c - y_i <= -e
        Add this to Constraints

    # TIME COMPLEXITY: O(n)
    For each blue point (x_j, y_j) in RedPoints do:
        Make inequality:
            a x_j^2 + b x_j + c - y_j > 0
        Convert to standard LP form (inequalities with <=):
            -a x_j^2 - b x_j - c + y_j <= -e
        Add this to Constraints

    ObjectiveFunction: Minimize 0

    # TIME COMPLEXITY: O(n)
    Solution = LinearProgramming(Constraints, ObjectiveFunction)

    If Solution is feasible then:
       Output "Parabola found with coefficients:"
       Output "a =", Solution.a
       Output "b =", Solution.b
       Output "c =", Solution.c

    Output "No solution exists"
```

## Question 13

For this question, we use an approximation to replace $L_2$ norm in order to use LP.

$$
\begin{cases}
r \geq 0 \\
w \geq 0 \\
\forall i \in \{1, \ldots, n\}, \forall j \in \{1, \ldots, k\} : \\
\quad (p_{i,x} - c_x) \cos\theta_j + (p_{i,y} - c_y) \sin\theta_j \geq r \cos\left(\frac{\pi}{k}\right) \\
\quad (p_{i,x} - c_x) \cos\theta_j + (p_{i,y} - c_y) \sin\theta_j \leq (r + w) \sec\left(\frac{\pi}{k}\right)
\end{cases}
$$

### Variables

- $c_x, c_y$: Coordinates of the center $c$ of the annulus
- $r \geq 0$: Inner Radius
- $w \geq 0$: Width of the annulus ($w = R - r$)

### Objective Function:

$$\min w$$

### Constraint:

1. $r \geq 0, \ w \geq 0$
2. To approximate $L_1$ norm, we pick $k$ directions. For a regular $k$-gon inscribed in the unit circle, define angles:
$$\theta_j = \frac{2\pi j}{k}, \quad j = 1, 2, \ldots, k$$
3. Compute unit vectors in these directions:
$$u_j = (\cos\theta_j, \sin\theta_j)$$
4. For each point $p_i = (p_{i,x}, p_{i,y})$ and each direction $u_j$:
   - Inner Boundary Constraint:
   $$(p_{i,x} - c_x) \cos\theta_j + (p_{i,y} - c_y) \sin\theta_j \geq r \cos\left(\frac{\pi}{k}\right)$$
   - Outer Boundary Constraint:
   $$(p_{i,x} - c_x) \cos\theta_j + (p_{i,y} - c_y) \sin\theta_j \leq (r + w) \sec\left(\frac{\pi}{k}\right)$$

   The factors $\cos\left(\frac{\pi}{k}\right)$ and $\sec\left(\frac{\pi}{k}\right)$ adjust for the approximation error due to replacing the circle with a regular $k$-gon.