AUTOMOTIVE
ARTIFICIAL
INTELLIGENCE

Coding Challenge 1: Lane Width Evaluation and Continuity Analysis

(DIFFICULTY: EASY)

You are given a piecewise polynomial (assume a polynomial one is also given) description of lane widths along a road segment (similar to OpenDRIVE).

HINT:

Each <width> entry defines a polynomial of the form:

$$w(s)=a+b\cdot(s-sOffset)$$
$$+c\cdot(s-sOffset)2+d\cdot(s-sOffset)3$$

valid for all positions “**s**" greater than or equal to its **sOffset** until the next entry begins.

Your task is to implement a geometry-based evaluation and continuity analysis routine for these polynomials.

Tasks

Part A — Lane Width Evaluation

- Implement a function getLaneWidth(s) that computes the width of the lane at any position “**s**”.

- Use the polynomial whose sOffset is the largest value less than or equal to “**s**”.
- Ensure your method handles edge cases (e.g., “**s**” smaller than the first offset, duplicate offsets).

Part B — Continuity Check Between Segments

- Given two consecutive width functions (e.g., one starting at sOffset=0.0 and the next at sOffset=20.0), check for positional continuity (C0) by evaluating the width at the junction from both polynomials. Report if the difference exceeds a tolerance ε.
- Check for approximate directional continuity (≈C1) by comparing the derivatives of the two polynomials at the junction. Report if the angular difference between slopes exceeds a threshold (in degrees).
- Handle degenerate cases where the derivative is zero or undefined.

Required Plots

1. **Plot 1 — Width Profile:** Plot the lane width function across the full road length. Clearly mark sOffset boundaries and annotate widths at those points.
2. **Plot 2 — Continuity Visualization:** For a junction between two width functions, overlay both polynomial curves near the boundary. Annotate the positional gap (ε) and angular difference (θ) at the junction.

What to Submit

- Source code implementing getLaneWidth(s) and the continuity checks.
- Image files for the plots (e.g., width_profile.png, continuity.png).
- A brief README (≤150 words) describing your distance metric, chosen tolerances, and assumptions.

Constraints and Notes

- Keep the implementation straightforward and readable. Any language is fine. Any plotting library is fine.
- Avoid lengthy explanations—concise comments are sufficient.

Sample Data

<width sOffset="0.0" a="3.0" b="0.1" c="0.0" d="0.0"/>

<width sOffset="20.0" a="5.0" b="-0.05" c="0.001" d="0.0"/>

Expected Output

getLaneWidth(0.0) → 3.0

getLaneWidth(10.0) → 4.0 (approx.)

getLaneWidth(20.0) → 5.0

getLaneWidth(25.0) → 4.76 (approx.)

C0 (positional): Difference = 0.0 (continuous)

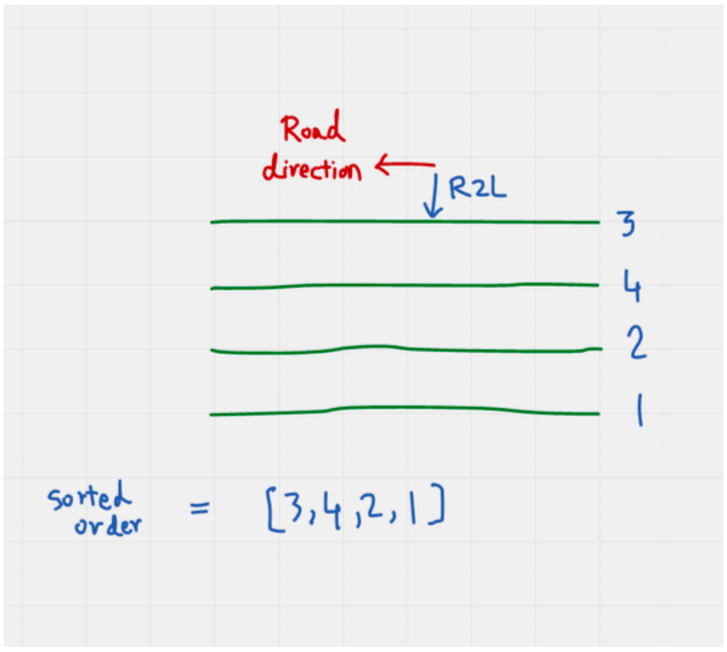
≈ C1 (directional): Angle difference ≈ 8.1° (within tolerance if ε = 10°)

Coding Challenge 2: RIGHT-TO-LEFT LANES Sorting

You are given a set of road lanes, each represented as a straight line segment in 2D space. Each lane is defined by two points (x0, y0) and (x1, y1) such that the direction of travel on the lane goes from (x0, y0) → (x1, y1). Each lane also has a unique string ID.

Task

Your task is to determine the correct order of the lane IDs from right to left, when viewed in the direction of travel along the road.



For example, in the above given figure, there are 4 lanes (given as lines with IDs: 1, 2, 3, 4). The road direction is being depicted with **red arrow**. (Road direction can be found from **any lane** – in the direction of first-point to last-point).
Looking in the road direction, if we sort the lanes (right-to-left), the resulting order is: (3, 4, 2, 1).

Note: The direction of the road can be in any direction. The above diagram is only for illustration purposes. Please make sure that the code is robust and handles the road given in any direction.

Please write detailed thought-process through comments. Try to describe your approach as much as possible. **Having a valid approach will also carry significant marks even if the solution does not work in the end.**

Note: Further instructions can be found in the attached file: lane-sorting-test.py

What to Submit

- Source code implementation of lanes sorting algorithm.
- Image files for the plots (e.g., PNG or SVG) named clearly (e.g., before_sort.png, after_sort.png).
- A very brief README (≤150 words) stating your choices (distance metric, tolerances) and any assumptions.

Coding Challenge 3: Geometry-Based Lane Smoothing and Continuity

(DIFFICULTY: MEDIUM To HARD)

You are given 2D/3D polylines representing lane boundaries in a road map. They may be noisy, kinked, or slightly misaligned at junctions (Assume your own 3D polyline). Your task is to implement a geometry-based smoothing routine that preserves endpoints and a continuity checker between two polylines.

Tasks

Part A — Bounded Smoothing

- Implement a smoothing method for a 2D/3D polyline with a specified number of iterations, preserving the first and last points (endpoints must not move).
- Enforce a maximum geometric deviation from the original polyline (choose and document a reasonable distance metric). If an iteration would exceed the bound, do not apply it.
- Your method should be robust to duplicate points and short segments.

Part B — Continuity Check

- Given two polylines A and B, determine positional continuity (C0) by checking if A's end and B's start are within a tolerance ε.

- Determine approximate directional continuity (≈C1) by comparing the direction of A's last segment and B's first segment against an angle tolerance (in degrees).
- Handle edge cases where direction is undefined (e.g., degenerate segments).

Required Plots

1. Plot 1 — Smoothing Comparison: Overlay the original and smoothed polylines on the same axes. Highlight fixed endpoints. Include a legend and a concise title.
2. Plot 2 — Continuity Visualization: Plot polylines A and B near their join. Draw small arrows for the direction vectors at the join and annotate the endpoint distance and angle (ε and θ) in the title or caption.

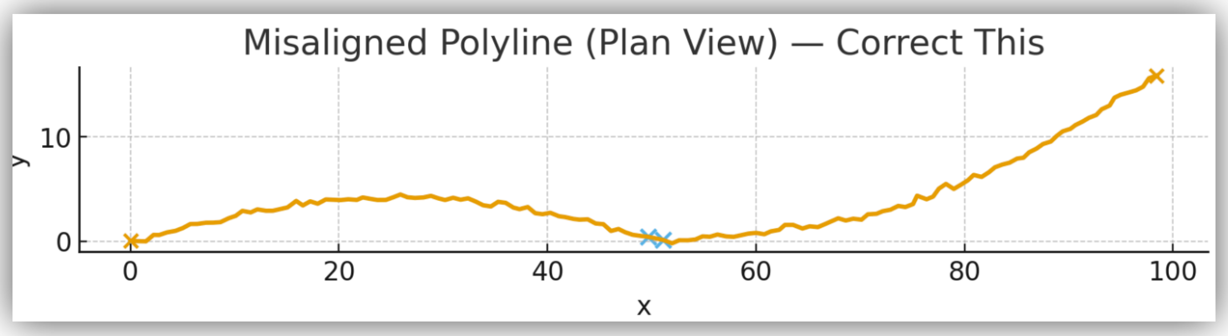
What to Submit

- Source code implementing the smoothing routine and the continuity check.
- Image files for the plots (e.g., PNG or SVG) named clearly (e.g., smoothing.png, continuity.png).
- A very brief README (≤150 words) stating your choices (distance metric, tolerances) and any assumptions.

Constraints and Notes

- Keep the implementation straightforward and readable. Any language is fine. Any plotting library is fine.
- Avoid lengthy explanations—concise comments are sufficient.

Sample Data


Misaligned Polyline (Plan View) — Correct This
y
10
0
0
20
40
60
80
100
x