Problem Set 1 – Delaunay triangulations

1 Triangulations in the plane

Let $\mathcal{P} \subset \mathbb{R}^2$ be a finite set of points in the plane containing at least three points, not all collinear.

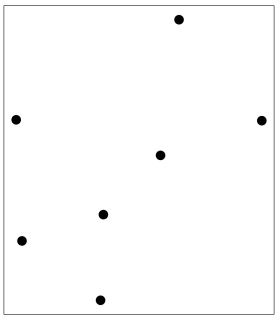


Figure 1: A point set $\mathcal{P}$

Definition 1.1. A **triangulation** of $\mathcal{P}$ is a maximal planar straight-line graph with vertex set $\mathcal{P}$.

The term "maximal" indicates that no additional edges can be added without violating planarity, i.e., without causing edge intersections. Thus, every face in this graph must be triangular; otherwise, it would be possible to insert at least one diagonal edge into a non-triangular face.

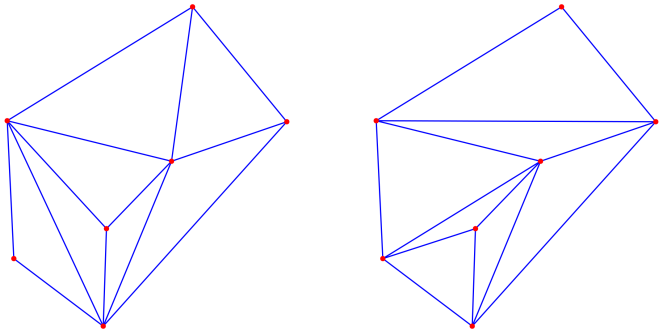


Figure 2: Two triangulations of the point set of fig. 1 with $v = 7, v_b = 5$.

A triangulation of $\mathcal{P}$ also determines its *convex hull:* all the outer edges, i.e., edges incident to one triangle only. Let $v = |\mathcal{P}|$ denote the total number of vertices, and let v_b be the number of vertices on the convex hull. The Euler characteristic of a planar graph is $\chi = 2$ and so the number of faces f, edges e and vertices v satisfy

$$\chi = 2 = v - e + (f + 1), \tag{1}$$

where the +1 accounts for the outer face (the rest of the plane). This yields, for a triangulation,[1]

$$f = 2v - v_b - 2 \tag{2}$$
$$e = 3v - v_b - 3 \tag{3}$$

So, all triangulations of a point set $\mathcal{P}$ have the same number of edges and triangles.

[1] *Prove it!*

1.1 The set of all triangulations of a point set

Triangulations are not unique. An elementary operation called the **flip** transforms one triangulation into another: Select two adjacent triangles that share an edge and consider this edge as a diagonal of the resulting quadrilateral. If the quadrilateral is convex, the edge can be replaced with the other diagonal to perform the flip.

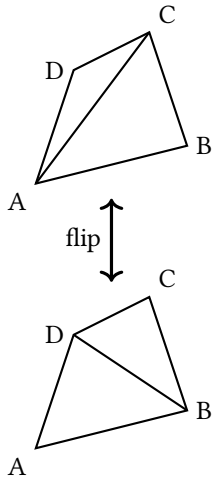
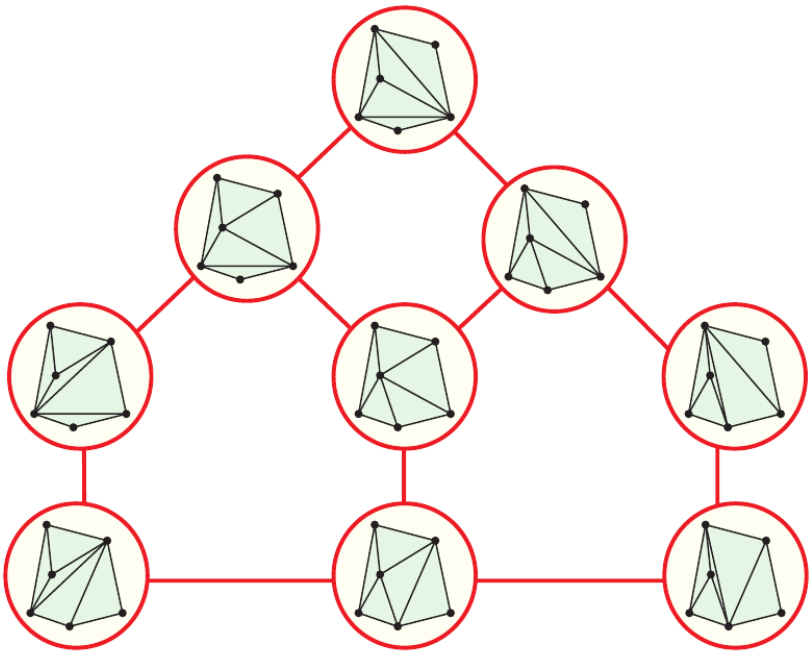


Figure 3: A flip

The flip graph

A nice way to visualize all triangulations of a point set $\mathcal{P}$ is the flip graph: The vertices are the triangulations of $\mathcal{P}$, and two triangulations T_1 and T_2 are connected by an edge if one can be obtained from the other by a flip.



[2] From S.L. Devadoss, J. O'Rourke, *"Discrete and Computational Geometry: An Introduction"*, Princeton University Press, 2011, p. 67

Figure 4: An example[2] of the flip graph of a point set $\mathcal{P}$.

Properties of the flip graph

- The flip graph of a triangulation is triangle-free: We will never come back to the same triangulation with exactly three flips.
- The flip graph of a triangulation is connected[1]. *We will always be able to reach any triangulation from any other one by a sequence of flips.*

[1] Ch. L. Lawson, *"Transforming triangulations"*, Discrete Mathematics 3 (1972), 365–372. Study the proof!

1.2 *Delaunay triangulations*

Triangulations have many applications in computer graphics, computational geometry, and physics. They are used to approximate surfaces and generate meshes for the numerical solution of partial differential equations. One special, nice triangulation preferred in many applications is the Delaunay triangulation.

- The Delaunay triangulation has the defining property that no point lies inside the circumcircle of any triangle. (Of course the points of the triangle lie on the circumcircle.)
- It maximizes the minimum angle of any triangle in the triangulation. So, it avoids skinny triangles and thus improves numerical stability.
- For any point $p \in \mathcal{P}$ the closest neighbour[2] $q \in \mathcal{P}$ is connected by an edge to p. In other words, the Delaunay triangulation includes the nearest neighbour graph of $\mathcal{P}$ as a subgraph.
- A Delaunay triangulation always exists.

[2] The relation 'being closest neighbour' is not symmetric!

1.2.1 Local Delaunay property and uniqueness

Definition 1.2. Let T be a triangulation of a point set $\mathcal{P}$. An edge $(p, q) \in T$ is locally Delaunay if

(i) it is an outer edge belonging to the convex hull of $\mathcal{P}$, or

(ii) it belongs to triangles $\triangle pqr$ and $\triangle pqs$, and point s is not inside the circumcircle of triangle $\triangle pqr$.

Theorem 1.3 (Delaunay Lemma). *(i) If every edge of T is locally Delaunay, then T is a Delaunay triangulation.*

(ii) If an edge (p, q) incident to the triangles $\triangle pqr$ and $\triangle pqs$ is not locally Delaunay, then (i) the quadrilateral formed by p, r, q, s is convex and (ii) the flipped edge (r, s) is locally Delaunay.

(iii) We can use edge flips to convert an arbitrary triangulation to a Delaunay triangulation (Lawson's flip algorithm).

(iv) If the four points p, q, r, s lie on a circle, both edges (pq) and (r, s) are locally Delaunay. Then more than one Delaunay triangulation exists.

2 Algorithms and Implementation

Your task is to write a Julia program that constructs a Delaunay triangulation. The program should take a list of 2D points $\mathcal{P}$ as input and output a list of edges providing a Delaunay triangulation. It should also visualize the triangulation using a plotting library.

Use only the following packages: (i) a plot package like `Plots` or `CairoMakie` and (ii) the `LinearAlgebra` package.

Remember that a Julia `Vector` has a stack interface, i.e. supports pushing and popping elements to/from the top of the stack.

There exist various algorithms to compute Delaunay triangulations. We propose a quite simple one:

(i) Create an arbitrary triangulation for the points $\mathcal{P}$, then

(ii) flip each edge that is not locally Delaunay until all edges are locally Delaunay (Lawson's flip algorithm).

For this, we need some small helper functions for standard tasks in computational geometry.

(i) Create your own data types for 2D points and edges. Ensure that `Edge(p,q)` and `Edge(q,p)` are equal, e.g. by using an inner constructor sorting the nodes.

(ii) Create a point set $\mathcal{P}$ using random numbers, e.g., `P = rand(30, 2)`. This will ensure that the points are in general position, i.e., no three points are collinear.

(iii) Write a function `intersect(e::Edge, f::Edge)` which returns `true` iff the two edges intersect, i.e. the two line segments share a common point. For our purposes they do not intersect if they share a node.

(iv) Write a function `incident_triangles{e::Edge}` which takes an edge (p, q) and returns a list of points r_i such that (p, q, s_i) is a triangle. This list will have length 1 for outer edges and length 2 for inner edges. Be careful! The existence of edges (p, s) and (q, s) does not imply that (p, q, s) is a triangle (p, q, s) in the sense of a triangulation; see Fig. 5.

(v) Write a function `in_circumcircle(p, q, r, s)` which returns `true` iff the point s is inside the circumcircle of the triangle $\triangle pqr$.

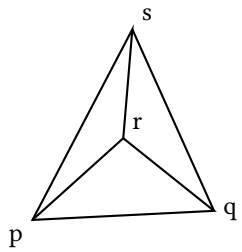


Figure 5: $\triangle pqs$ is not a triangle of the triangulation.

Our first algorithm computes a triangulation for a point set $\mathcal{P}$:

> Sort the points according to x-coordinates. The first three points determine a triangle. Consider the next point p in the ordered set and connect it with all previously considered points $\{p_1, \ldots, p_k\}$ which are visible to p. Continue this process until all points have been considered. Here, a point p_i is visible to p if the planned new edge (p, p_i) does not intersect any already existing edges.

Algorithm 1: Incremental Triangulation

Data: A list of 2D points $P = \{p_1, p_2, \dots, p_n\}, n \geq 3$
Result: A list (stack) of edges E

Sort P by x-coordinates in ascending order;
Initialize empty edge list $E \leftarrow \emptyset$;

Take first 3 points p_1, p_2, p_3 from sorted list;
push *triangle edges* (p_1, p_2), (p_2, p_3), (p_3, p_1) to E;

for $i = 4$ **to** n **do**
    for $j = 1$ **to** $i - 1$ **do**
        $edge \leftarrow (p_i, p_j)$;
        if *edge does not* **intersect** *any existing edge in* E **then**
            push *edge to* E;
        end
    end
end
return E;

The idea of Lawson's flip algorithm is simply to go through all edges and flip those that are not locally Delaunay. The only point to take into account is that after flipping the diagonal of a quadrilateral, we have to re-check the four edges of the quadrilateral. This is done by using a stack to keep track of the edges that need to be checked.

Algorithm 2: Lawson's Flip Algorithm for Delaunay Triangulation

Data: A triangulation (edges list) T
Result: A Delaunay triangulation T'

stack $S \leftarrow$ **copy** (T);
while *not* **isEmpty** S **do**
    $e \leftarrow$ **pop** *from* S;
    if $e = (p, q)$ *is not locally Delaunay in quadrilateral* (p, r, q, s) **then**
        flip *edge* e **in** T;
        foreach *edge* $e' \in \{(p, r), (r, q), (q, s), (s, p)\}$ **do**
            push e' *to* S;
        end
    end
end
return modified T;