Computational topology
Homework 2 (due: December 6th 2020)

Each problem is worth a certain amount of points. The point assignment should somewhat reflect the difficulty of the problem. Some problems are theoretical, others require you also submit the code (that conforms to the requirements given in the problem description). You may choose which problems to solve, **15 points is equal to 100%**.

You have to submit your solutions before the deadline as **one .zip file** to the appropriate mailbox at https://ucilnica.fri.uni-lj.si/course/view.php?id=111 (near the top of the page).

This .zip file should contain:

1. **a namesurname.pdf file written in LaTeX** and containing the solutions to the theoretical problems you have chosen as well as solutions and explanations for the programming problems (also make sure you sign your name on the top of the first page),
2. **.py files** containing the code (one for each of the programming problems you have chosen).

1 Theoretical problems

1. **(3 points) Triangulations.**

   Let S be a set of n points in the plane.

   (a) Show that there are at most $2^{\frac{n(n-1)}{2}}$ triangulations of S.

   (b) The degree of a point in a triangulation T is the number of edges in T, incident to that point. Construct a set S such that all possible triangulations of S have a point of degree $n-1$.

   (c) If not all points in S are collinear, then any triangulation T of S has at most $3n-3$ edges. Use this fact to prove that any triangulation T of S has a point of degree 5 or less.

2. **(1 point) Vietoris-Rips Complex and Čech Complex.**

   Let $S = \{(0,0),(2,0),(1,0.5),(1,1.5)\} \subset \mathbb{R}^2$.

   (a) Build the Vietoris-Rips complex $\mathrm{VR}_{2\epsilon}(S)$ and the Čech complex $\check{\mathrm{C}}_{\epsilon}(S)$ for $\epsilon = 0.8$.

   (b) Build the Vietoris-Rips complex $\mathrm{VR}_{2\epsilon}(S)$ and the Čech complex $\check{\mathrm{C}}_{\epsilon}(S)$ for $\epsilon = 1$.

   In each case list all the simplices, determine its dimension and find the Euler characteristic.

3. **(1 point) Voronoi diagram.** Find a configuration of n points ($n > 3$) in the plane such that their Voronoi diagram will have a cell with $n-1$ vertices.

4. **(3 points) Chessboard Complex.**

   The *chessboard complex* of a $m \times n$ chessboard is a simplicial complex $\Delta_{m\times n}$. The vertices of $\Delta_{m\times n}$ correspond to the squares of the chessboard. Simplices of $\Delta_{m\times n}$ correspond to non-taking placements of rooks (ie. placements where no two rooks are in the same column or in the same row).

   (a) Show that the chessboard complex $\Delta_{3\times 2}$ is homeomorphic to the circle S^1. (Hint: Show that the simplicial complex you obtain is a 1-dimensional connected manifold with no boundary.)

(b) Show that the chessboard complex $\Delta_{4\times 3}$ is homeomorphic to a torus $S^1 \times S^1$. (Show that the complex you obtain is an orientable connected 2-dimensional manifold without boundary with Euler characteristic 0. Alternatively, you can try to construct an explicit homeomorphism.)

(c) Show that the chessboard complex $\Delta_{n\times(n-1)}$ is a manifold for all n.

5. **(1 point) Vietoris-Rips Complex and Čech Complex in d_∞ metric.**

   Prove that if we define the Vietoris-Rips and the Čech Complex using d_∞ metric instead of d_2, then the complexes we obtain for any given ϵ are the same, ie. $\check{C}_\epsilon = \mathrm{VR}_{2\epsilon}$.

2 Programming problems

6. **(2 points) Line sweep triangulation**

   Given a cloud of points S, write a function `triangulate(S, vertical = True)` that returns the list of edges E and triangles T obtained by the line sweep algorithm. By default the line should be vertical, and if the optional parameter is set to False the algorithm should use a horizontal line instead.

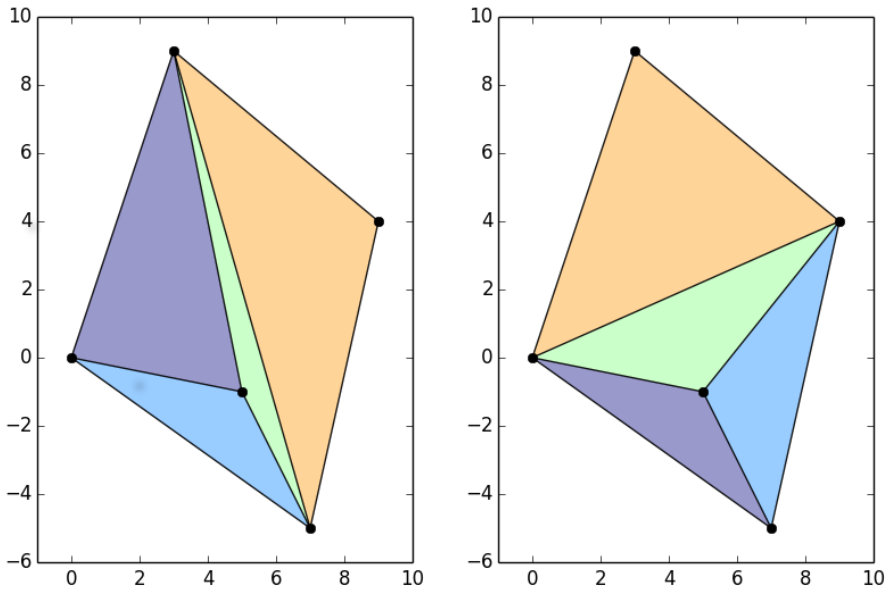
   Submit a file named `linesweeptriangulation.py` that also includes a function `generify(S)` which adds a small amount of noise to the input points before constructing the triangulation to ensure they are in general position.

   Plot the points, edges and triangles in the plane.

   Sample input:

   ```
   S = [(0, 0), (3, 9), (5, -1), (9, 4), (7, -5)]
   ```

   Sample output:



   Make up two more test cases consisting of at least 100 points and include the resulting triangulations (with both vertical and horizontal line sweeps) in your report.

7. **(3 points) Delauney triangulation**

Your file `delauney.py` should contain a function `optimize(T)`, which takes as input any triangulation and optimizes it to produce the Delauney triangulation of the underlying set of points. Do this by implementing the edge-flip algorithm. Plot both the initial triangulation and the resulting Delauney triangulation. You can include the function `generify(S)` which adds a small amount of noise to the input points before constructing the triangulation to ensure they are in general position.

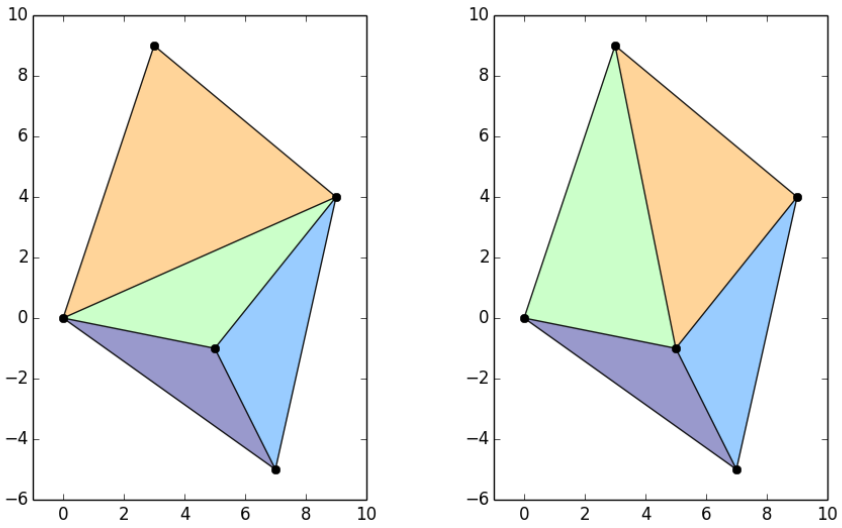
Plot the Dealuney triangulation in the plane.

Sample input:

```
[((0, 0), (5, -1), (7, -5)), ((5, -1), (7, -5), (9, 4)),
 ((0, 0), (5, -1), (9,  4)), ((0,  0), (3,  9), (9, 4))]
```

Sample output:

```
[((0,  0), (5, -1), (7, -5)), ((5, -1), (7, -5), (9, 4)),
 ((5, -1), (3,  9), (0,  0)), ((5, -1), (3,  9), (9, 4))]
```



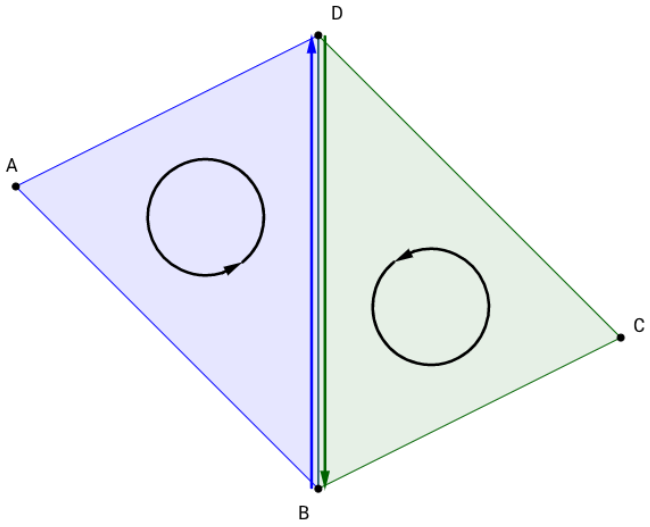
Make up at least two more test cases consisting of at least 100 points and include plots of original and resulting Delaunay triangulations in your report.

8. **(4 points) Orientation of surfaces**

Your file `orientable.py` should contain a function `orientableQ(T)`, which returns `True` if a 2-manifold given by its triangulation T is orientable and `False` otherwise. Also include a function `orientable(T)`, which returns the list of oriented triangles if the 2-manifold is orientable and `None` otherwise.

Test your function on triangulations of a torus, a Klein bottle, a sphere, a cylinder and a Moebius band (you can use the triangulations found on the internet, or you can construct your own).

A 2-manifold is orientable if you can choose the orientations of all its triangles consistently. Two triangles that share an edge are consisently oriented if they induce opposite orientations on the common edge (see figure).



Sample input:

```
M = [(1, 2, 3), (2, 3, 4), (3, 4, 5), (4, 5, 6), (2, 5, 6), (1, 2, 6)]
```

Sample output:

```
This surface is not orientable!
Oriented triangles:
None
```

Sample input:

```
S2 = [(1, 2, 3), (1, 2, 4), (1, 3, 4), (2, 3, 4)]
```

Sample output:

```
This surface is orientable.
Oriented triangles:
[(1, 2, 3), (1, 4, 2), (1, 3, 4), (2, 4, 3)]
```