Computational topology

Urša Zrimšek

Theoretical problems

Exploring different metrics

We have three different metrics, defined on $\mathbb{R}^2$:

$$\alpha(x,y) = \begin{cases} 0 & \text{if } x = y \\ \sqrt{x_1^2 + x_2^2} + \sqrt{y_1^2 + y_2^2} & \text{otherwise} \end{cases},$$

$$\beta(x,y) = \begin{cases} \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2} & \text{if } x_1 y_2 = x_2 y_1 \\ \sqrt{x_1^2 + x_2^2} + \sqrt{y_1^2 + y_2^2} & \text{otherwise} \end{cases},$$

$$\gamma(x,y) = \begin{cases} |y_2 - x_2| & \text{if } x_1 = y_1 \\ |x_2| + |y_1 - x_1| + |y_2| & \text{otherwise} \end{cases}.$$

To simplify the above equations we defined points x and y as $x = (x_1, x_2), y = (y_1, y_2)$.

First thing we explored was the distance between points $a = (1,2)$, $b = (2,4)$ and $c = (2,-1)$ in all three metrics:

$$\begin{array}{lll} \alpha(a,b) = 3\sqrt{5} & \beta(a,b) = \sqrt{5} & \gamma(a,b) = 7 \\ \alpha(a,c) = 2\sqrt{5} & \beta(a,c) = 2\sqrt{5} & \gamma(a,c) = 4 \\ \alpha(c,b) = 3\sqrt{5} & \beta(c,b) = 3\sqrt{5} & \gamma(c,b) = 5. \end{array}$$

Next we drew three open balls in each metric. The results can be seen on figures 1, 2 and 3. We notice that the center is always contained in the balls, that's because the distance between center and itself is always equal to 0, which is smaller than any radius the ball can have.

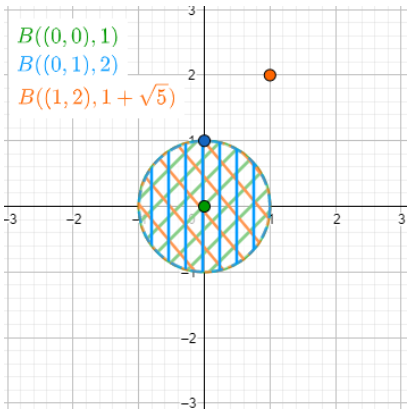

Figure 1. **Three different open balls in metric α.**

Discrete metric

The discrete metric on space X is defined as $d : X \times X \to \mathbb{R}$,

$$d(x,y) = \begin{cases} 0 & x = y \\ 1 & \text{otherwise.} \end{cases}$$

If $X = \mathbb{N}$, the following sets are described as:

$$B(1, \frac{1}{2}) = \{1\},\ B(2,1) = \{2\},\ \overline{B}(3, \frac{1}{2}) = \{3\},\ \overline{B}(4,1) = \{3,4,5\}.$$

In such metric, every three pairwise distinct integers represent equilateral triangle, because the distances between them are always the same – equal to 1.

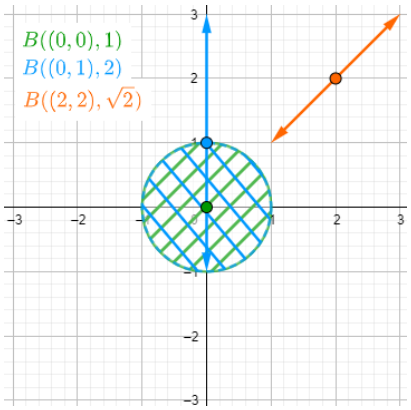

Figure 2. **Three different open balls in metric β.**

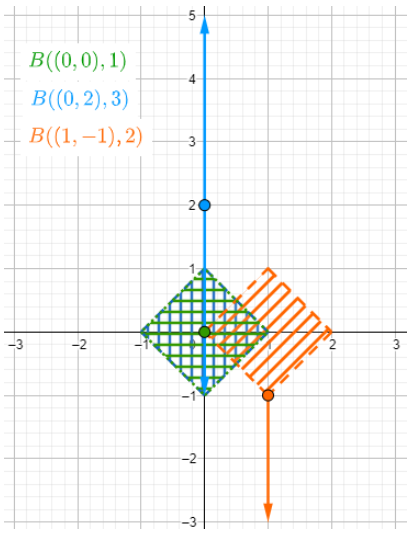

Figure 3. **Three different open balls in metric γ.**

Homeomorphic spaces

Let $X_n = S^{n-1} \times [0,1] \subset \mathbb{R}^{n+1}$ and $Y_n = \{(x_1, \ldots, x_n) \in \mathbb{R}^n; 1 \leq x_1^2 + \cdots + x_n^2 \leq 4\}$. On figures 4 and 5 we can see this spaces for $n = 1$ and $n = 2$.

We want to show that this spaces are homeomorphic. We will do that by finding a homomorphism between each two spaces. For the ones we plotted, we can easily see what our mapping should do. For $n = 1$ we want to "lay" the lines $(\pm 1, 0) + t(0,1);\ t \in [0,1]$ onto the intervals $[\pm 1, \pm 2]$. This can be done with mapping $f_1((x,y)) = x(1+y)$. Similarly for $n = 2$, we want to lay the band onto the donut, which means that we want each line $(x,y,0) + t(0,0,1);\ t \in [0,1]$ to map onto $(x,y) + t\frac{(x,y)}{\|(x,y)\|};\ t \in [0,1]$. This can be done with mapping $f_2((x,y,z)) = (x,y) \cdot (1+z)$. Now we can notice the pattern, and assume that the desired mapping is

$$f_n((x_1, \ldots, x_n)) = (1 + x_n) \cdot (x_1, \ldots, x_{n-1}).$$

To prove that this is a homeomorphism, we will also need $g_n : Y_n \to X_n$ .We define $g_n(y) = (\frac{y_1}{\|y\|}, \ldots, \frac{y_{n-1}}{\|y\|}, \|y\| - 1)$. First we need to check that $\forall x : f_n(x) \in Y_n$ and $\forall y : g_n(y) \in X_n$.

Since we know that $(x_1, \ldots, x_{n-1})$ lies on a sphere so it's length is equal to 1, and that $x_n \in [0, 1]$, we can calculate

$$\|f_n(x)\| = (1 + x_n)^2(x_1^2 + \cdots + x_{n-1}^2) = (1 + x_n)^2 \in [1, 4],$$

which proves that $f_n(x) \in Y_n$. For the second part it is obvious that first $n - 1$ coordinates lie on a sphere and that the last component is between 0 and 1, so it is an element of X_n. Now we will check that $f \circ g = id_Y$ and $g \circ f = id_X$:

$$f(g(y)) = (1 + \|y\| - 1)(\frac{y_1}{\|y\|}, \ldots, \frac{y_{n-1}}{\|y\|}) = y,$$

$$g(f(x)) = (\frac{x_1(1 + x_n)}{\|1 + x_n\|}, \ldots, \frac{x_{n-1}(1 + x_n)}{\|1 + x_n\|}, \|1 + x_n\| - 1) = x.$$

With this, we proved that $\forall n : X_n \cong Y_n$.

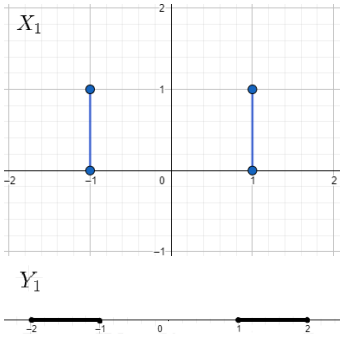

Figure 4. **Homeomorphic spaces for $n = 1$: X_1 on top, Y_1 on the bottom.**

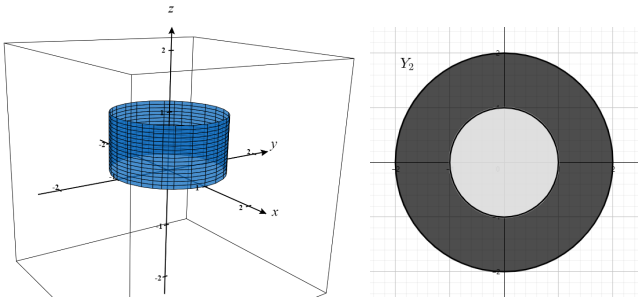

Figure 5. **Homeomorphic spaces for $n = 2$: X_2 on the left, Y_2 on the right.**

Programming problems

Deciding connectivity

In this task we were searching for components of graphs, given with lists of vertices and edges. We performed a depth first search, tagging all the vertices we visited with current component, and raising the component number when we needed to jump on a new untagged edge. To easily test the correctness of our approach, we draw the graphs and colored the components returned by our algorithm. The results can be seen on figure 6.

Shelling disks

A *shelling* is a sequence of all triangles from a triangulation of a polygon P (simple, closed), such that any initial sequence is homeomorphic to a closed disk. Our goal was to find an algorithm that produces one of such possible sequences. If we want that our sequence will be homeomorphic to a disk, it needs to have one component and no holes.

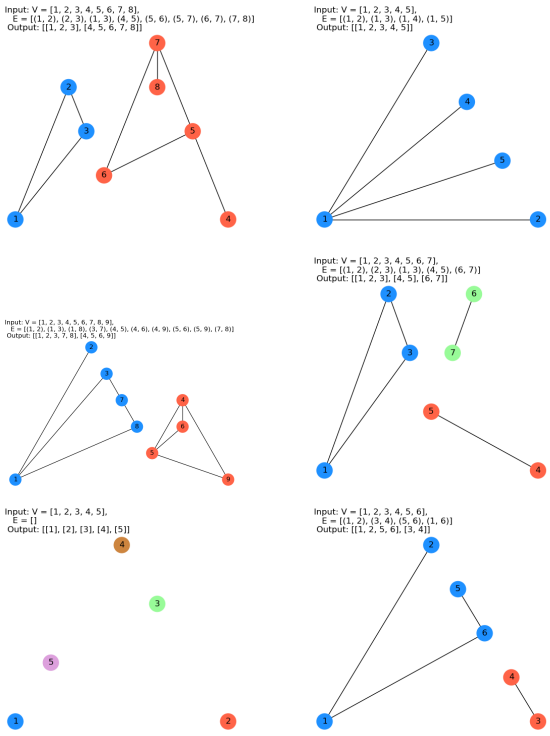

Figure 6. **Test case graphs.** The graphs are built from the input written above, and colored with the output (also written above each graph) of our algorithm. We can see that it works correctly.

A perfect tool to check that is the Euler characteristic χ, that tells us the difference between the number of components and holes. Since we can't make a new component and a hole with addition of one triangle, we can always just check what would be the Euler characteristic with any new triangle, and add it only if it is equal to 1. We can easily compute it by $\chi = F - E + V$, so we only need to check how many new edges and vertices we add with new a triangle. On figure 7 we can see two examples, that were given by inputs

```
T1 = [(1,2,6),(1,5,6),(2,3,7),(2,6,7),(3,4,8),(3,7,8),
(5,6,9),(6,7,11),(6,9,10),(6,10,11),(7,8,12),(7,11,12),
(9,10,13),(10,13,14),(10,11,15),(10,14,15),(11,12,15),
(12,15,16)]
```

and

```
T2 = [(1,2,6),(7,11,12),(3,4,8),(2,3,7),(6,10,11),
(2,6,7),(3,7,8),(5,6,9),(6,7,11),(6,9,10)]
```

The outputs of our algorithm were:

```
[(1,2,6),(1,5,6),(2,6,7),(5,6,9),(6,7,11),(6,9,10),
(6,10,11),(7,11,12),(9,10,13),(10,13,14),(10,11,15),
(10,14,15),(11,12,15),(12,15,16),(2,3,7),(3,7,8),
(7,8,12),(3,4,8)]
```

and

```
[(1,2,6),(2,6,7),(6,7,11),(7,11,12),(2,3,7),
(6,10,11),(3,7,8),(6,9,10),(3,4,8),(5,6,9)].
```

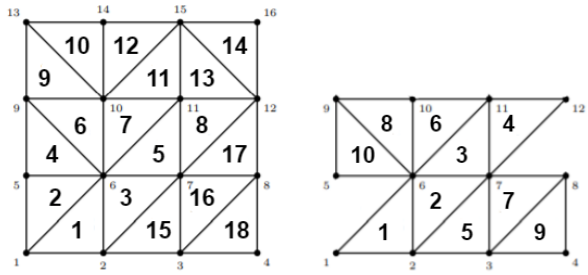

Figure 7. **Shellings of two different triangulations of polygons.** The numbers on the triangles represent the order that our algorithm returned.

Jordan curve theorem

A simple closed curve in the plane is a connected curve with no self-intersections. We will consider a special case of a finite chain of straight line segments closing in a loop to form a simple closed polygonal curve. The Jordan Curve Theorem states that every simple closed curve in $\mathbb{R}^2$ decomposes $\mathbb{R}^2$ into two components, the bounded inside and the unbounded outside. We implemented a function that returns whether point T lies inside the polygonal curve P. To determine this, we counted the number of intersections of an infinite ray starting at T with the segments forming the curve P.

We first implemented a function that takes two lines, given with a point and direction vector. In our case we used it to calculate the intersection between a line given with $T+t\cdot(0,1)$ and a segment of the polygon given by $P_0+p\cdot(P_1-P_0)$. We were interested in coefficients t and p of the point of intersection. The function first checks if the lines are parallel. In that case, we need to check if the lines are "the same", if they are we return the coefficient p, gotten by $P_0+p\cdot(P_1-P_0)=T$. If lines are different, we return `None` as that means they don't intersect. If lines are not parallel, we return t, p, coefficients of the intersection on each line. We get them by solving equations $T+t\cdot(0,1)=P_0+p\cdot(P_1-P_0)$.

In our main function, we go through all polygon sections and explore the intersections of the ray with them. If the previous function returns `None`, we continue to the next function. If it returns only parameter p, we check weather it's between 0 and 1. That means that the point is on the edge (or vertex) of the polygon, so we return `True`. If p is not in that range, we continue. In most situations the function returns both parameters, p and t. As we only want to look for intersections in positive direction (otherwise their number wouldn't tell us anything), we continue if $t<0$. If $t=0$ it means that the point is on the edge, so we return `True`. Next we look at parameter p: if it's not in range $(0,1]$ we continue, as that means that the intersection is outside the line. Notice that we also left out the case where the intersection is on the first point of the line ($p=0$). That's because we look at each point twice, once when it's the first point in the polygon section, and once when it's the last. If $p\in(0,1)$ we count it as an intersection. If $p=1$ we need to further inspect where the point is, as we can have two different situations, described on figure 8.

We can distinct between the two if we look at the angle of vectors representing the polygon segments coming into and out of the vertex. We calculated the angles $\phi_1,\phi_2\in[-\pi,\pi]$. Because we already inspected parallel segment before, we know that $\phi_1\neq 0$. If $\phi_2\in\{0,\pi\}$, we change it to the angle of the first

Figure 8. **Two different situations of intersection being on vertex.** Here we see a proper intersection on vertex on the right, and improper one on the left. We shouldn't count the one on the left as an intersection.

next unparallel segment. Now we can check whether $\phi_1\cdot\phi_2>0$, and if it is, we have a proper intersection and we count it, otherwise not.

After this whole process, we only check if the number of intersections is even or odd. If it's odd, the point is inside, otherwise it's not. On figure 9 we can see two polygons with three test points. The inputs given to our algorithm and the

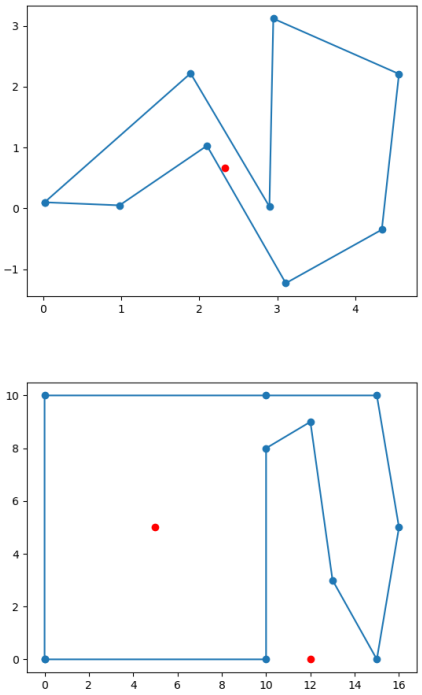

Figure 9. **Two polygons with the points of interest.**

results are:

```
T1 = (2.33, 0.66)
P1 = [(0.02,0.10),(0.98,0.05),(2.10,1.03),(3.11,-1.23),
(4.34,-0.35),(4.56,2.21),(2.95,3.12),(2.90,0.03),
(1.89, 2.22)]
T2 = (5, 5)
T3 = (12, 0)
P2 = [(0,0),(10,0),(10,8),(12,9),(13,3),(15,0),
(16,5),(15,10),(10,10),(0, 10)]
insideQ(P1, T1) = True
insideQ(P2, T2) = True
insideQ(P2, T3) = False
```

From the figures it is clear that the results are correct.