FlatSurf Demo 1

Define the base field.

```
K.<sqrt2> = NumberField(x**2 - 2, embedding=1.414)
```

Note that *sqrt2* now stores the actual square root of 2.

```
RR(sqrt2)
```

```
1.41421356237309
```

Constructing the regular octagon:

```
from flatsurf.geometry.polygon import Polygons
```

The following is the parent for polygons with coordinates in the field K.

Remarks:

- The first vertex of our polygons is always the origin.
- All our polygons are convex.

```
Polygons(K)
```

```
polygons with coordinates in Number Field in sqrt2 with defining
polynomial x^2 - 2
```

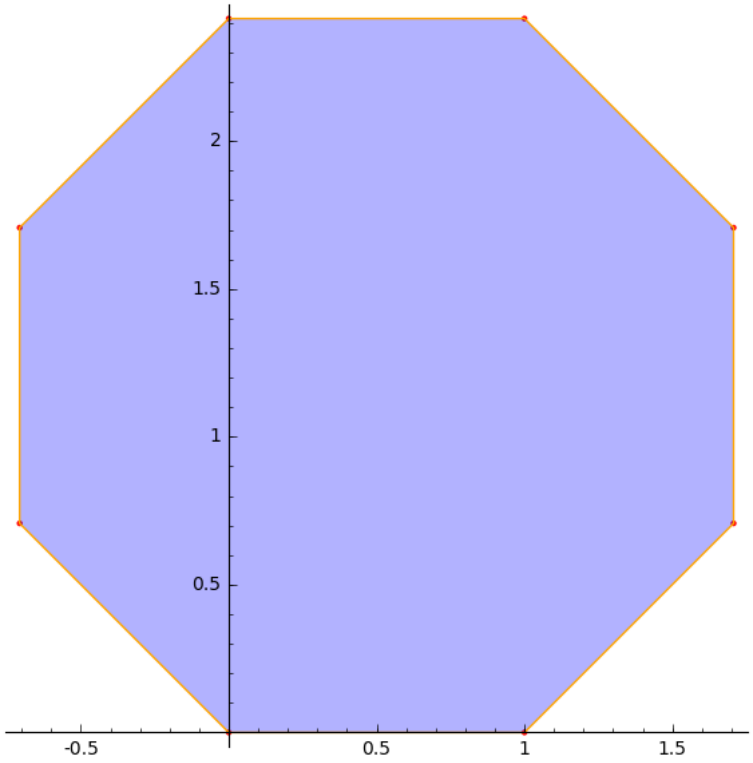
To construct a polygon, use the parent to build the parent. Passing a list of edge vectors will produce the polygon. The edge vectors must sum to zero.

```
p = Polygons(K)([(1,0),
(sqrt2/2, sqrt2/2),
(0, 1),
(-sqrt2/2, sqrt2/2),
(-1,0),
(-sqrt2/2, -sqrt2/2),
(0, -1),
(sqrt2/2, -sqrt2/2)
])

print(p)
```

```
Polygon: (0, 0), (1, 0), (1/2*sqrt2 + 1, 1/2*sqrt2), (1/2*sqrt2 + 1,
1/2*sqrt2 + 1), (1, sqrt2 + 1), (0, sqrt2 + 1), (-1/2*sqrt2, 1/2*sqrt2 +
1), (-1/2*sqrt2, 1/2*sqrt2)
```

```
p.plot()
```



Defining a translation surface

A translation surface is a gluing of one or more polygons with edge identifications glued by translation.

We will construct the octagon with opposite edges identified. Edges of our polygons are indexed by $[0, 1, 2, \ldots, n-1]$ where n is the number of sides of the polygon. An edge of a polygon is indexed by a pair (p, e) where p is a polygon index and e is an edge index.

The gluings are specified as a list of pairs of edges of polygons. For example we want to the bottom edge $(0, 0)$ to the top edge $(0, 4)$.

```
from flatsurf import *
```

Define a surface built from polygons with vertices in K^2.

```
surface = Surface_list(K)
```

We add the polygon p to the surface. It gets a label, in this case 0.

```
surface.add_polygon(p)
```

```
0
```

We now glue the edges together. The line below says "glue edge e of polygon 0 to edge e+4 of polygon 0."

```
for e in range(4):
    surface.change_edge_gluing(0,e,0,e+4)
```

We want to view the surface we built as a translation surface.

```
s=TranslationSurface(surface)
```

Run some tests to make sure the surface is okay.

```
TestSuite(s).run(verbose=True)
```

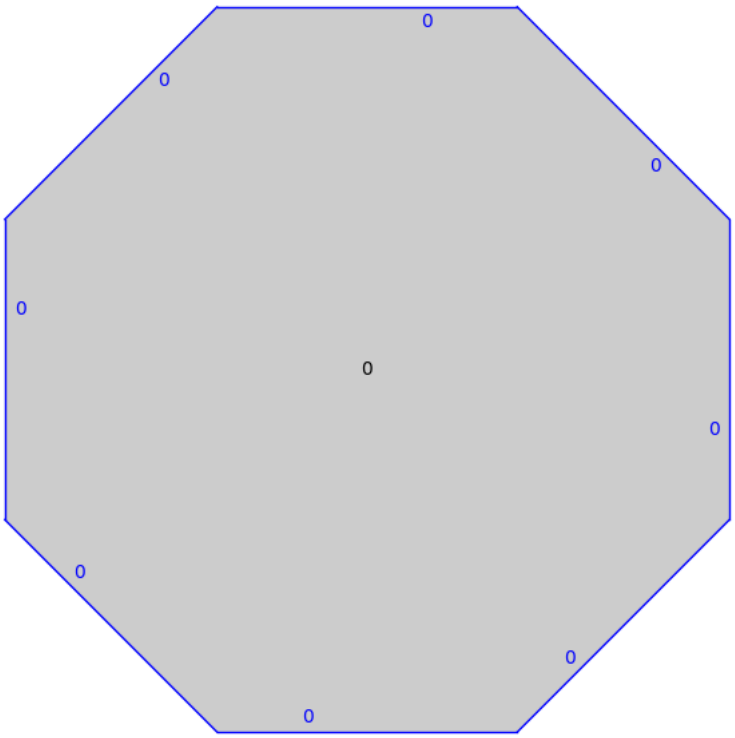
```
    running ._test_base_label() . . . pass
    running ._test_base_ring() . . . pass
    running ._test_category() . . . pass
    running ._test_edge_matrix() . . . pass
    running ._test_gluings() . . . pass
    running ._test_not_implemented_methods() . . . pass
    running ._test_override() . . . pass
    running ._test_pickling() . . . pass
    running ._test_polygons() . . . pass
```

Graphical surfaces:

A graphical surface is a version of a surface which stores some extra data that can be used to draw the surface.

```
gs=s.graphical_surface()
```

```
gs.plot()
```



Straight-line flow

Construct the tangent vector in polygon 0 based at (0,0) pointed pointed in direction (1,2).

```
v=s.tangent_vector(0,(0,0),(1,2))
print(v)
```

```
    SimilaritySurfaceTangentVector in polygon 0 based at (0, 0) with vector
```

```
(1, 2)
```

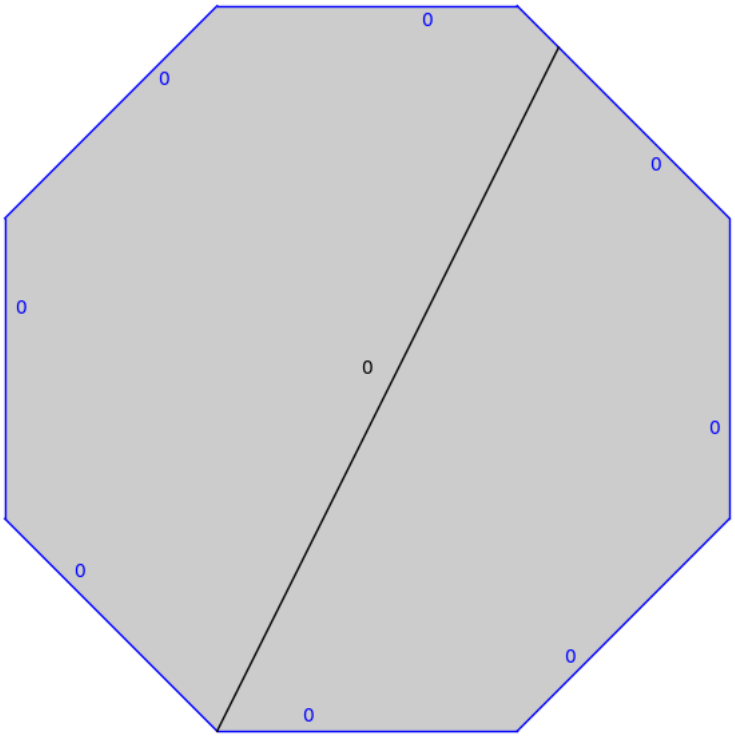
You can convert the tangent vector to a StraightLineTrajectory, which is a finite list of intersections of the straight-line trajectory with the polygons defining the surface.

```
traj=v.straight_line_trajectory()
```

```
print traj
```

```
Straight line trajectory made of 1 segments from (0, 0) in polygon 0 to
(1/3*sqrt2 + 2/3, 2/3*sqrt2 + 4/3) in polygon 0
```

```
gs.plot()+traj.graphical_trajectory(gs).plot()
```



The *flow(n)* method constructs the next *n* segments obtained by straight-line flowing forward and intersecting with the provided polygons. It will just stop if a singularity is hit.

```
traj.flow(100)
print(traj)
print("It has length: "+str(traj.combinatorial_length()))
```

```
Straight line trajectory made of 17 segments from (0, 0) in polygon 0 to
(0, sqrt2 + 1) in polygon 0
It has length: 17
```

```
traj.is_saddle_connection()
```

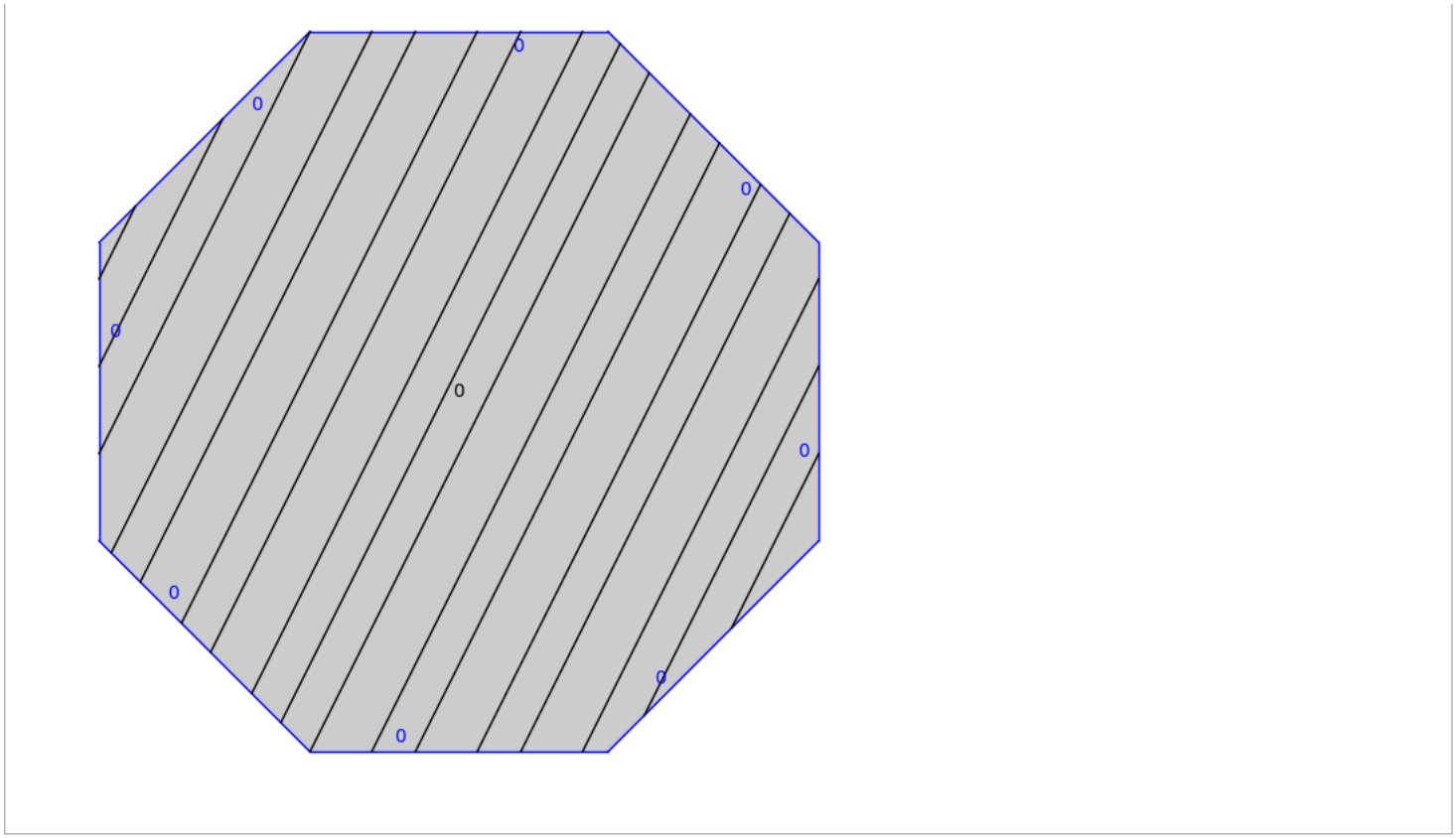
```
True
```

We can draw a picture of this saddle connection by converting it to a GraphicalStraightLineTrajectory.

```
gtraj = traj.graphical_trajectory(gs)
```

```
gs.plot()+gtraj.plot()
```

evaluate


0
0
0
0
0
0
0
0
0
0