Computational Geometry Software

1. Computational Geometry
   - problems and software
   - getting started
2. Solving Geometric Problems
   - convex hulls
   - Voronoi diagrams
   - Delaunay triangulations
3. the Lake Superior Polygon
   - convex hull and Delaunay triangulation
   - improved Delaunay triangulations

MCS 507 Lecture 30
Mathematical, Statistical and Scientific Software
Jan Verschelde, 30 October 2023

Computational Geometry Software

1. Computational Geometry
   - problems and software
   - getting started
2. Solving Geometric Problems
   - convex hulls
   - Voronoi diagrams
   - Delaunay triangulations
3. the Lake Superior Polygon
   - convex hull and Delaunay triangulation
   - improved Delaunay triangulations

computational geometry

Computational geometry studies algorithms for geometric problems.

We distinguish three classes of problems:

- **Static :** compute an object from given inputs.
  Examples: convex hulls, triangulations, Voronoi diagrams.
- **Query :** given a search space and a query,
  find objects in the search space that satisfy the query.
- **Dynamic :** similar problems as static and query,
  but the input changes and objects are inserted and deleted.

Applications : finite element methods, robot motion planning, etc.

computational geometry software

We follow *Mastering SciPy* by Francisco J. Blanco-Silva, Packt Publishing 2015.

We introduce three software packages:

- The `spatial` package of `scipy` for spatial algorithms and data structures: queries, Delaunay triangulations, convex hulls, and Voronoi diagrams.
- Mayavi: 3D scientific data visualization and plotting, provides `mlab`, for 3D plotting in Python.
- Triangle: a mesh generator by Jonathan Richard Shewchuck, used through the wrapper written by Dzhelil Rufat, available at `http://dzhelil.info/triangle/index.html`.

The Computational Geometry Algorithms Library

http://www.cgal.org

CGAL is a software project that provides easy access and reliable geometric algorithms in the form of a C++ library. CGAL provides an extensive collection of examples.

Its development is funded by academia and companies, dual licensing:

- open source: Lesser GPL (LGPL) and GPL;
- a commercial license from the GeometryFactory.

Installation:

- CGAL is available via package managers.
- There is a Python interface, install with pip or conda.

CGAL Made More Accessible, by Nir Goren, Efi Fogel, and Dan Halperin, `arxiv:2202.13889v2 [cs.CG] 7 Jun 2023`.

Computational Geometry Software

1. Computational Geometry
   - problems and software
   - getting started
2. Solving Geometric Problems
   - convex hulls
   - Voronoi diagrams
   - Delaunay triangulations
3. the Lake Superior Polygon
   - convex hull and Delaunay triangulation
   - improved Delaunay triangulations

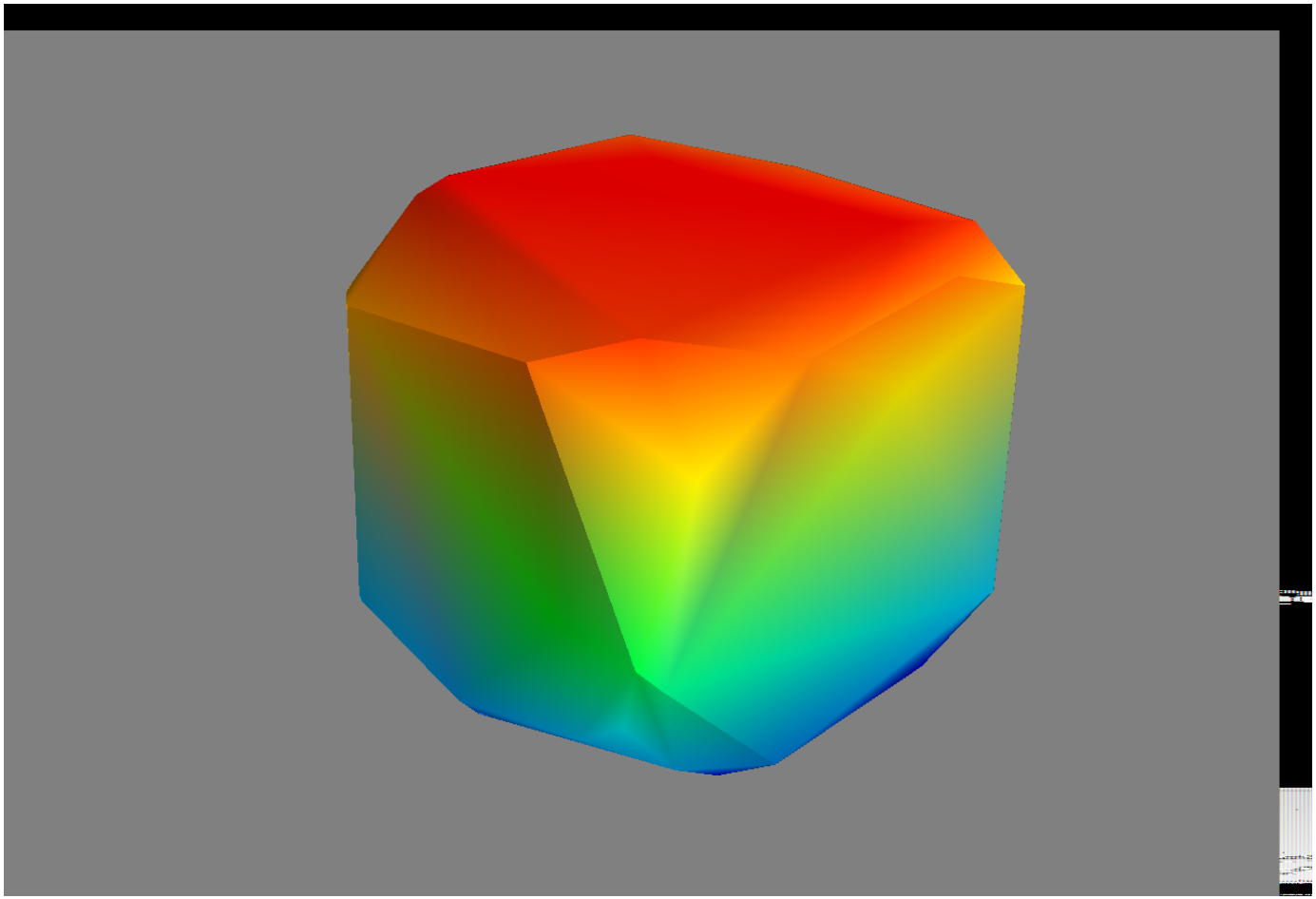
plotting the convex hull of random points

The script below shows the output of the ConvexHull of scipy.spatial on a random set of 320 points in 3-space using Mayavi.

```
import numpy as np
from scipy.spatial import ConvexHull
from mayavi import mlab

points = np.random.rand(320, 3)
hull = ConvexHull(points)
X = hull.points[:, 0]
Y = hull.points[:, 1]
Z = hull.points[:, 2]
mlab.triangular_mesh(X, Y, Z, hull.simplices)
mlab.show()
```

the plot of a convex hull in 3-space



Clicking on the picture allows for a change of view point.

Computational Geometry Software

1. Computational Geometry
   - problems and software
   - getting started
2. Solving Geometric Problems
   - convex hulls
   - Voronoi diagrams
   - Delaunay triangulations
3. the Lake Superior Polygon
   - convex hull and Delaunay triangulation
   - improved Delaunay triangulations

the convex hull problem

Problem statement:

> Input: a set S of points.
> Output: the smallest convex set that contains S.

We distinguish between the planar and the spatial case.

- The convex hull in the plane consists of vertices and edges.
- In three dimensions, we have vertices, edges, and simplices.

The `ConvexHull` of scipy.spatial computes the convex hull with the Qhull library, see `http://www.qhull.org`.

C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa.
The Quickhull algorithm for convex hulls.
ACM Trans. Math. Softw. 22(4):469-483, 1996.

the convex hull of 20 random points in the plane

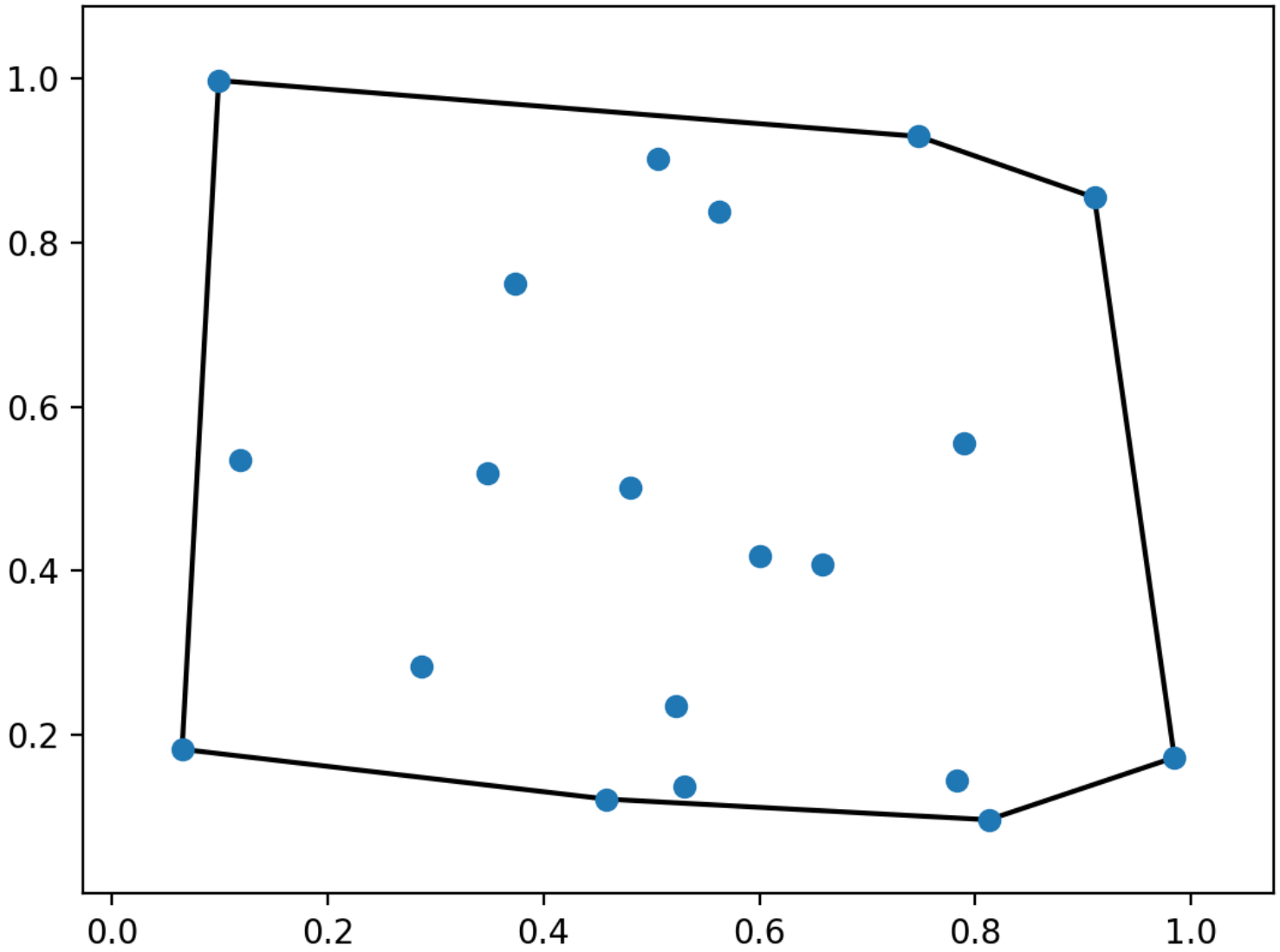
```
import numpy as np
from scipy.spatial import ConvexHull
from scipy.spatial import convex_hull_plot_2d
from matplotlib import pyplot as plt

points = np.random.rand(20, 2)
hull = ConvexHull(points)
print('The vertices :')
print(hull.vertices)
print('The edges :')
for simplex in hull.simplices:
    print(simplex)
convex_hull_plot_2d(hull)
plt.show()
```

output of the script

```
The vertices :
[ 3  5  6 12  1 18 16]
The edges :
[3 5]
[18  1]
[12  1]
[16  3]
[16 18]
[6 5]
[ 6 12]
```

the plot of a convex hull in the plane



Euler's formula

Euler's formula states: #vertices $-$ #edges $+$ #faces $= 2$.

Let us verify this formula for some random points.

```
import numpy as np
from scipy.spatial import ConvexHull

points = np.random.rand(320, 3)
hull = ConvexHull(points)
nv = hull.vertices.size
ns = hull.simplices.shape[0]
print('number of vertices :', nv)
print('number of simplices :', ns)
```

computation of all edges

```
edges = []
for simplex in hull.simplices:
    x, y, z = simplex
    edge = ((x, y) if x < y else (y, x))
    if edge not in edges: edges.append(edge)
    edge = ((y, z) if y < z else (z, y))
    if edge not in edges: edges.append(edge)
    edge = ((z, x) if z < x else (x, z))
    if edge not in edges: edges.append(edge)
ne = len(edges)
print('number of edges :', ne);
print('nv - ne + ns :', nv - ne + ns)
```

Computational Geometry Software

1. Computational Geometry
   - problems and software
   - getting started
2. Solving Geometric Problems
   - convex hulls
   - Voronoi diagrams
   - Delaunay triangulations
3. the Lake Superior Polygon
   - convex hull and Delaunay triangulation
   - improved Delaunay triangulations

Voronoi diagrams

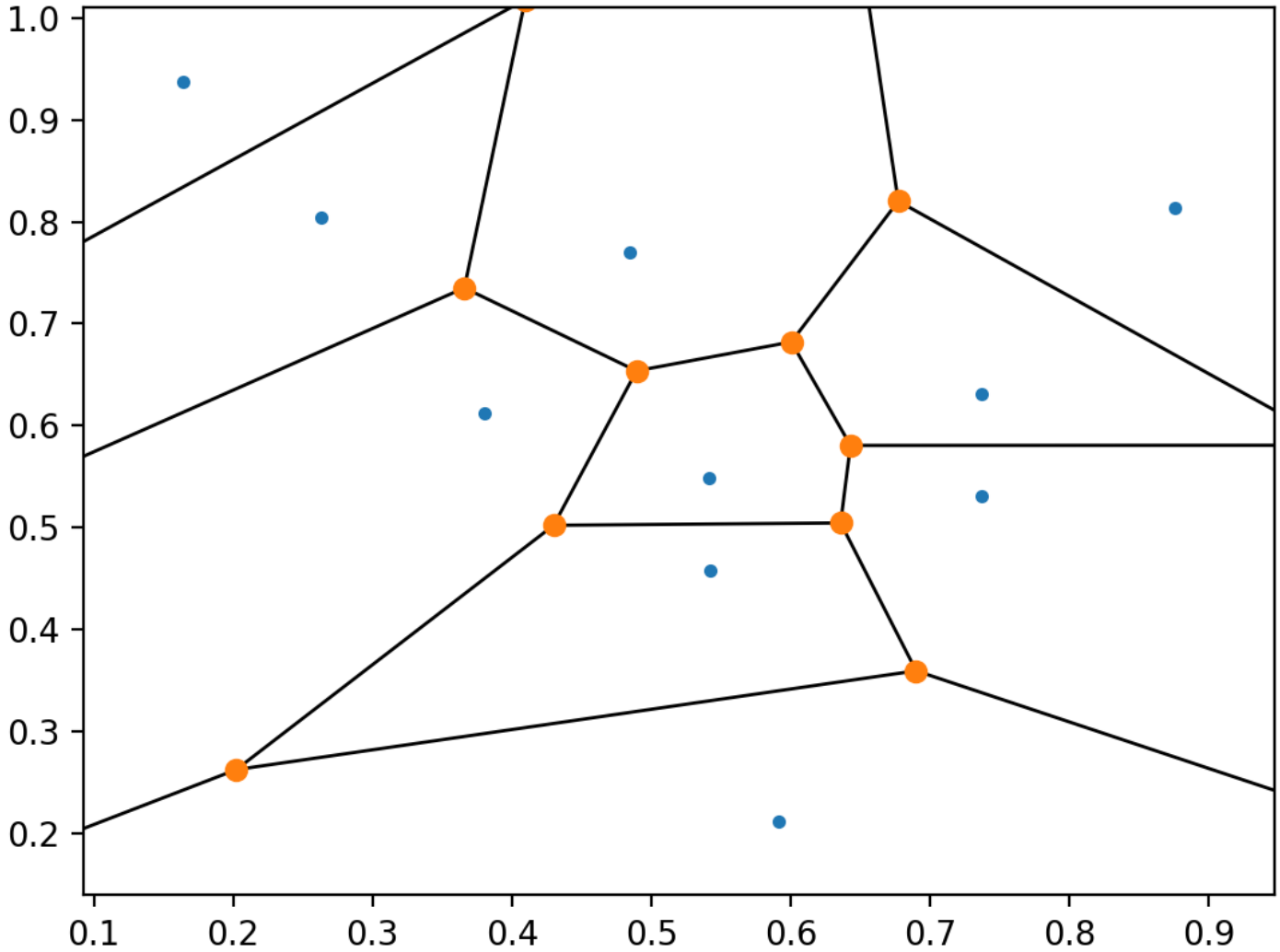
Problem statement:

Input: a set S of points (sites) in the plane.
Output: a partition of the plane in cells, so that all points in the same cell are closest to the same site.

```
import numpy as np
from scipy.spatial import Voronoi, voronoi_plot_2d
from matplotlib import pyplot as plt

points = np.random.rand(10, 2)
vor = Voronoi(points)
voronoi_plot_2d(vor)
plt.show()
```

the Voronoi diagram of 10 random sites



the attributes of a Voronoi diagram

```
vor = Voronoi(points)
print('the input points :')
print(vor.points)
print('the vertices :')
print(vor.vertices)
print('the ridge vertices :')
print(vor.ridge_vertices)
print('the ridge points :')
print(vor.ridge_points)
print('the cells :')
for region in vor.regions:
    print(region)
print('the point_region :')
print(vor.point_region)
```

Computational Geometry Software

1. Computational Geometry
   - problems and software
   - getting started
2. Solving Geometric Problems
   - convex hulls
   - Voronoi diagrams
   - Delaunay triangulations
3. the Lake Superior Polygon
   - convex hull and Delaunay triangulation
   - improved Delaunay triangulations

Delaunay triangulations

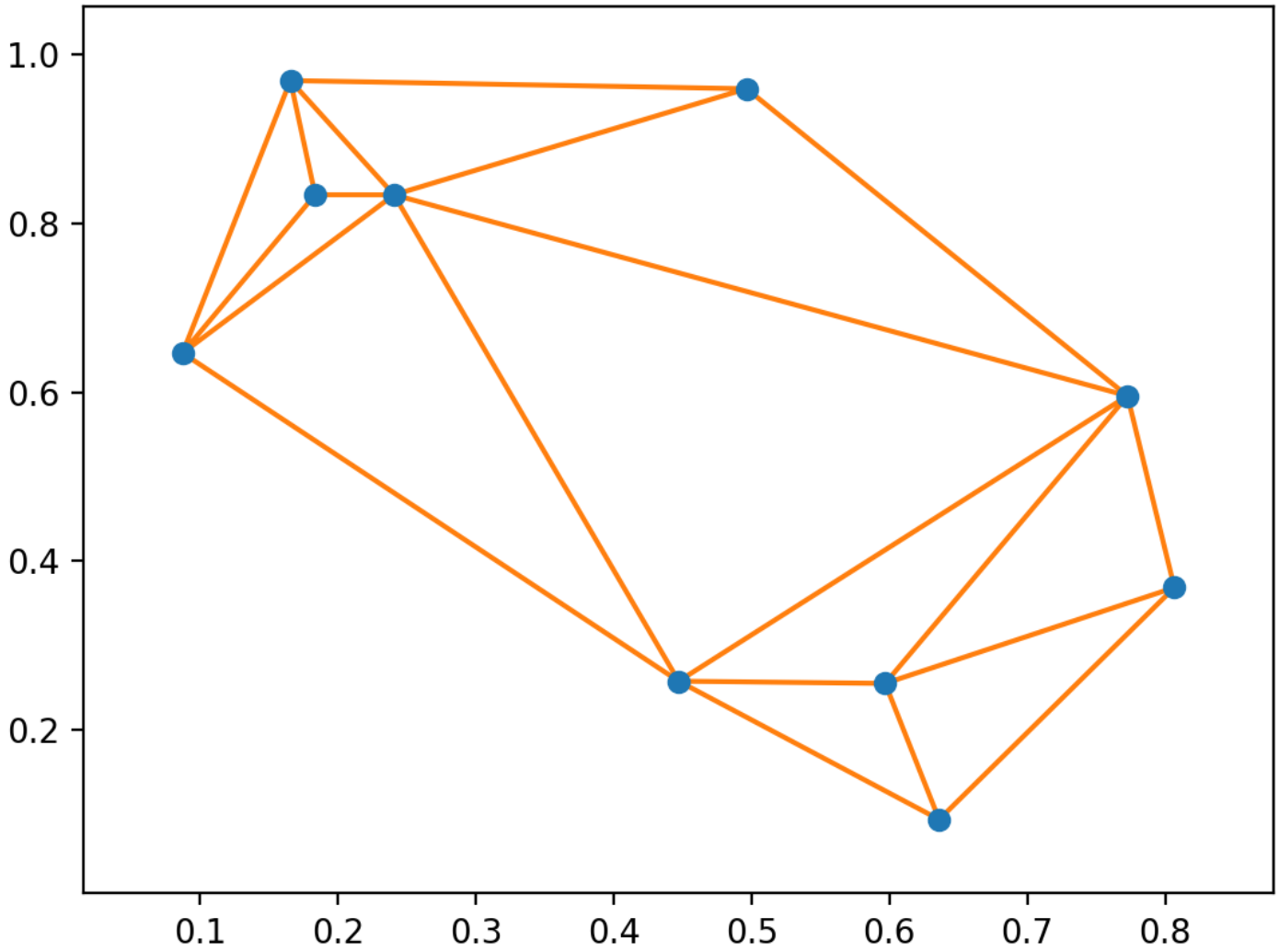
Problem statement:

Input: a set S of points in the plane.
Output: the triangulation that maximizes the minimal angle over all triangulations of S.

```
import numpy as np
from scipy.spatial import Delaunay, delaunay_plot_2d
from matplotlib import pyplot as plt

points = np.random.rand(10, 2)
tri = Delaunay(points)
delaunay_plot_2d(tri)
plt.show()
```

the Delaunay triangulation of 10 random points



some attributes of a Delaunay triangulation

```
points = np.random.rand(10, 2)
tri = Delaunay(points)
print('the input points :')
print(tri.points)
print('indices of the simplices :')
for simplex in tri.simplices:
    print(simplex)
print('indices to neighbor simplices :')
for neighbor in tri.neighbors:
    print(neighbor)
print('the equations :')
for equation in tri.equations:
    print(equation)
```

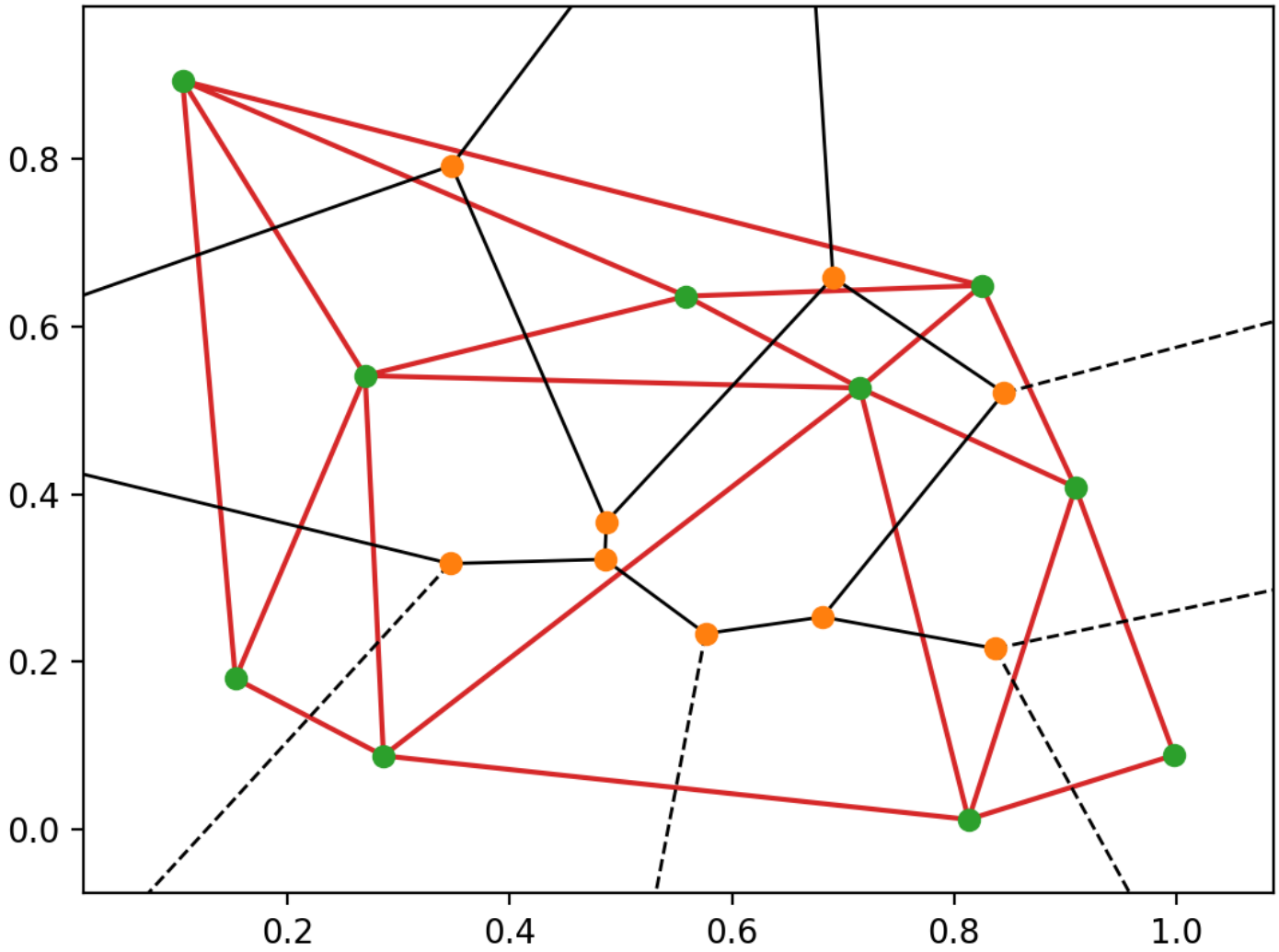
duality

The dual of a Voronoi diagram is a Delaunay triangulation and the dual of a Delaunay triangulation is a Voronoi diagram.

```
import numpy as np
from scipy.spatial import Voronoi, voronoi_plot_2d
from scipy.spatial import Delaunay, delaunay_plot_2d
from matplotlib import pyplot as plt

points = np.random.rand(10, 2)
vor = Voronoi(points)
tri = Delaunay(points)
fig, axs = plt.subplots()
voronoi_plot_2d(vor, ax=axs)
delaunay_plot_2d(tri, ax=axs)
plt.show()
```

Voronoi diagram and Delaunay triangulation



Computational Geometry Software

1. Computational Geometry
   - problems and software
   - getting started
2. Solving Geometric Problems
   - convex hulls
   - Voronoi diagrams
   - Delaunay triangulations
3. the Lake Superior Polygon
   - convex hull and Delaunay triangulation
   - improved Delaunay triangulations

the Lake Superior Polygon

Following the *Mastering Scipy* book of Francisco J. Blanco-Silva, Packt Publishing 2015, we examine the Lake Superior polygon, available for download at
`https://github.com/blancosilva/Mastering-Scipy/blob/master/chapter6/superior.poly`.

This polygon has 7 holes (for the islands), 518 vertices, and 518 edges.

The .poly format comes from the Triangle software, available at
`http://www.cs.cmu.edu/~quake/triangle.html`

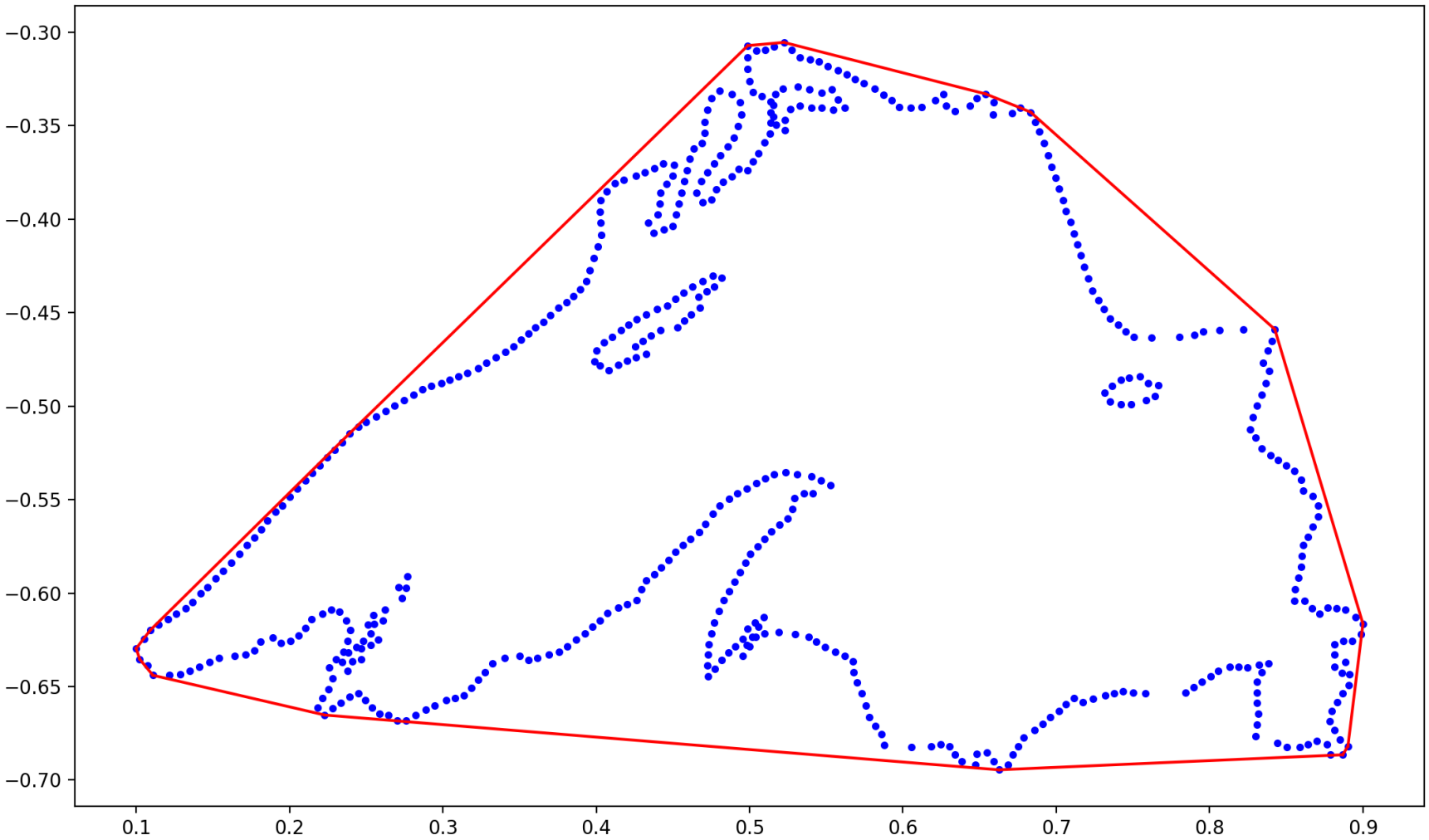
We use the `read_poly()` function of *Mastering SciPy*.

plotting the convex hull

```
from read_poly import read_poly
from matplotlib import pyplot as plt
from scipy.spatial import ConvexHull, convex_hull_plot_2d

lake_superior = read_poly('superior.poly')
points = lake_superior['vertices']
print('the number of vertices : ', end='')
print(points.size)
hull = ConvexHull(points)
plt.plot(points[:,0], points[:,1], 'b.')
for simplex in hull.simplices:
    plt.plot(points[simplex, 0], points[simplex, 1], 'r-')
plt.show()
```

the convex hull

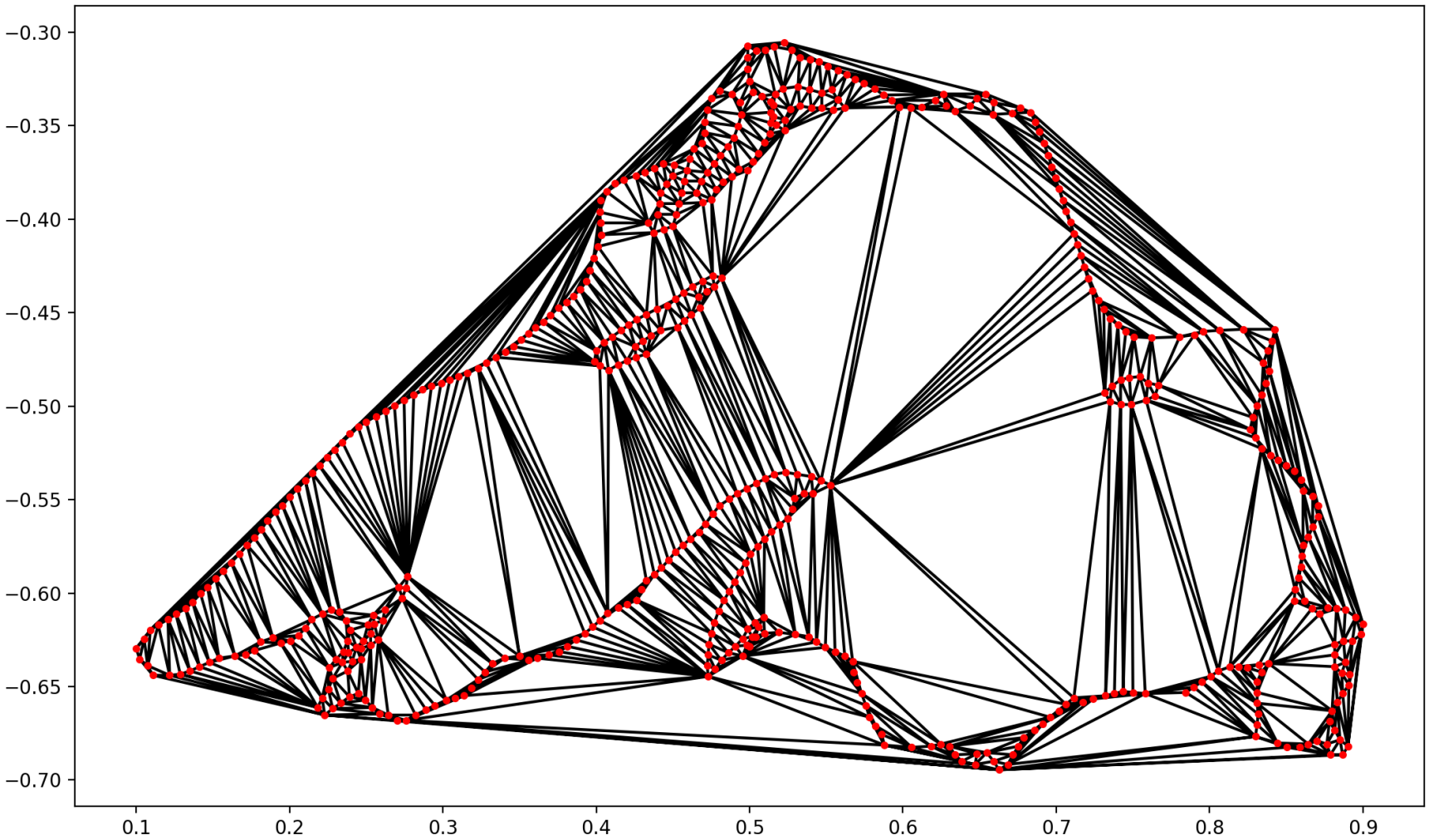


computing the Delaunay triangulation

```
from read_poly import read_poly
from matplotlib import pyplot as plt
from scipy.spatial import Delaunay

lake_superior = read_poly('superior.poly')
points = lake_superior['vertices']
print('the number of vertices : ', end='')
print(points.size)
tri = Delaunay(points)
plt.triplot(points[:,0], points[:,1],
    tri.simplices, 'k-')
plt.plot(points[:,0], points[:,1], 'r.')
plt.show()
```

the Delaunay triangulation



Computational Geometry Software

1. Computational Geometry
   - problems and software
   - getting started
2. Solving Geometric Problems
   - convex hulls
   - Voronoi diagrams
   - Delaunay triangulations
3. the Lake Superior Polygon
   - convex hull and Delaunay triangulation
   - improved Delaunay triangulations

prize winning software

The scripts in the following slides contain code of the *Mastering SciPy* book of Francisco J. Blanco-Silva, using the software Triangle.

Jonathan Richard Shewchuk, ***Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator***, in *Applied Computational Geometry: Towards Geometric Engineering*, edited by Ming C. Lin and Dinesh Manocha, volume 1148 of Lecture Notes in Computer Science, pages 203-222, Springer-Verlag, Berlin, May 1996.

Received the 2003 James H. Wilkinson Prize for Numerical Software.

Usable in Python through wrappers written by Dzhelil Rufat, available via `pip install triangle`.

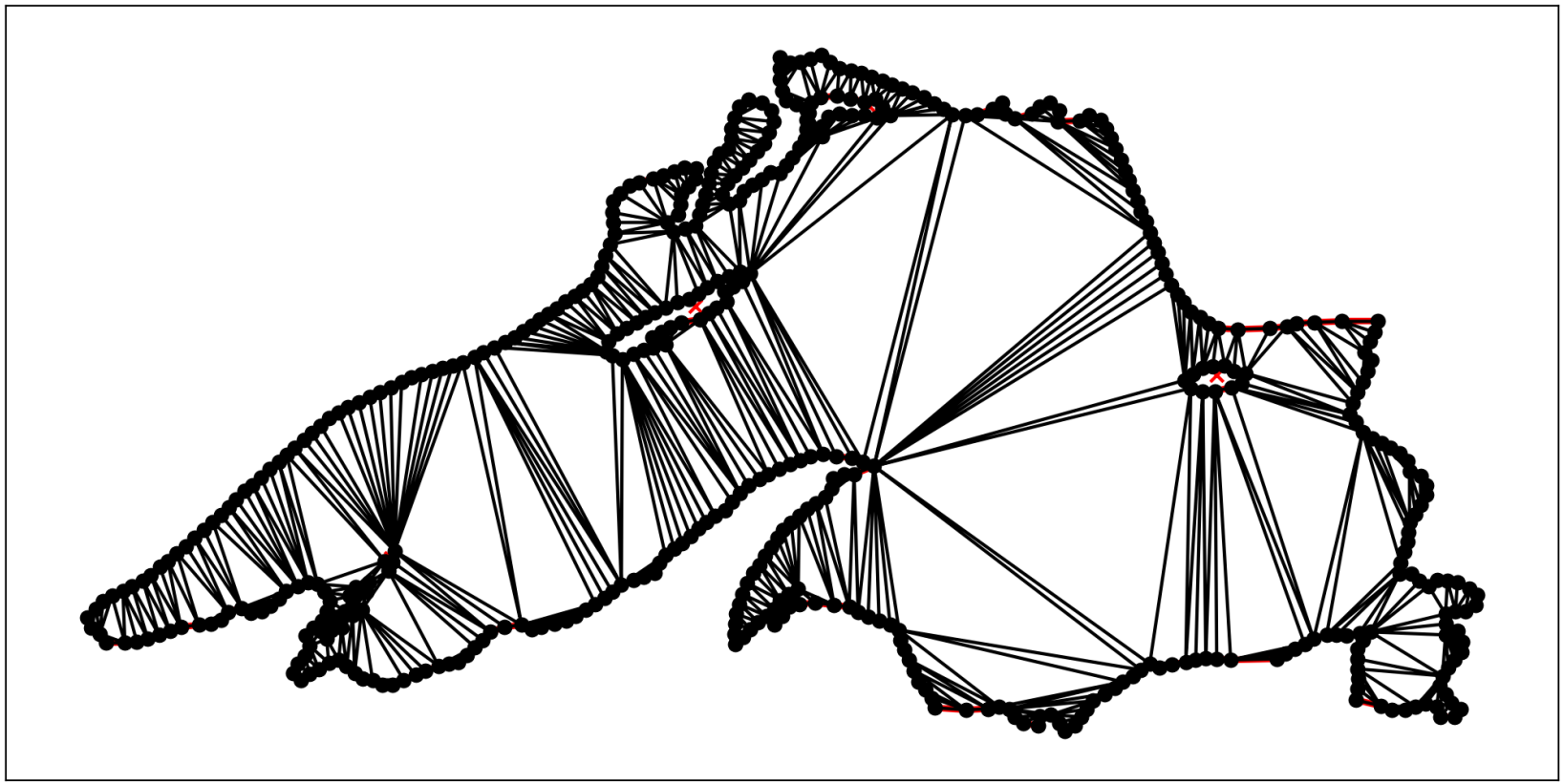
computing a constrained Delaunay triangulation

A constrained Delaunay triangulation (cndt) is computed with the flag `p` to indicate that the source is a planar straight line graph.

```
from read_poly import read_poly
from matplotlib import pyplot as plt
from triangle import triangulate
from triangle import plot as tplot

lake_superior = read_poly('superior.poly')
cndt = triangulate(lake_superior, 'p')
ax = plt.subplot(111)
tplot(ax, **cndt)
plt.show()
```

the constrained Delaunay triangulation



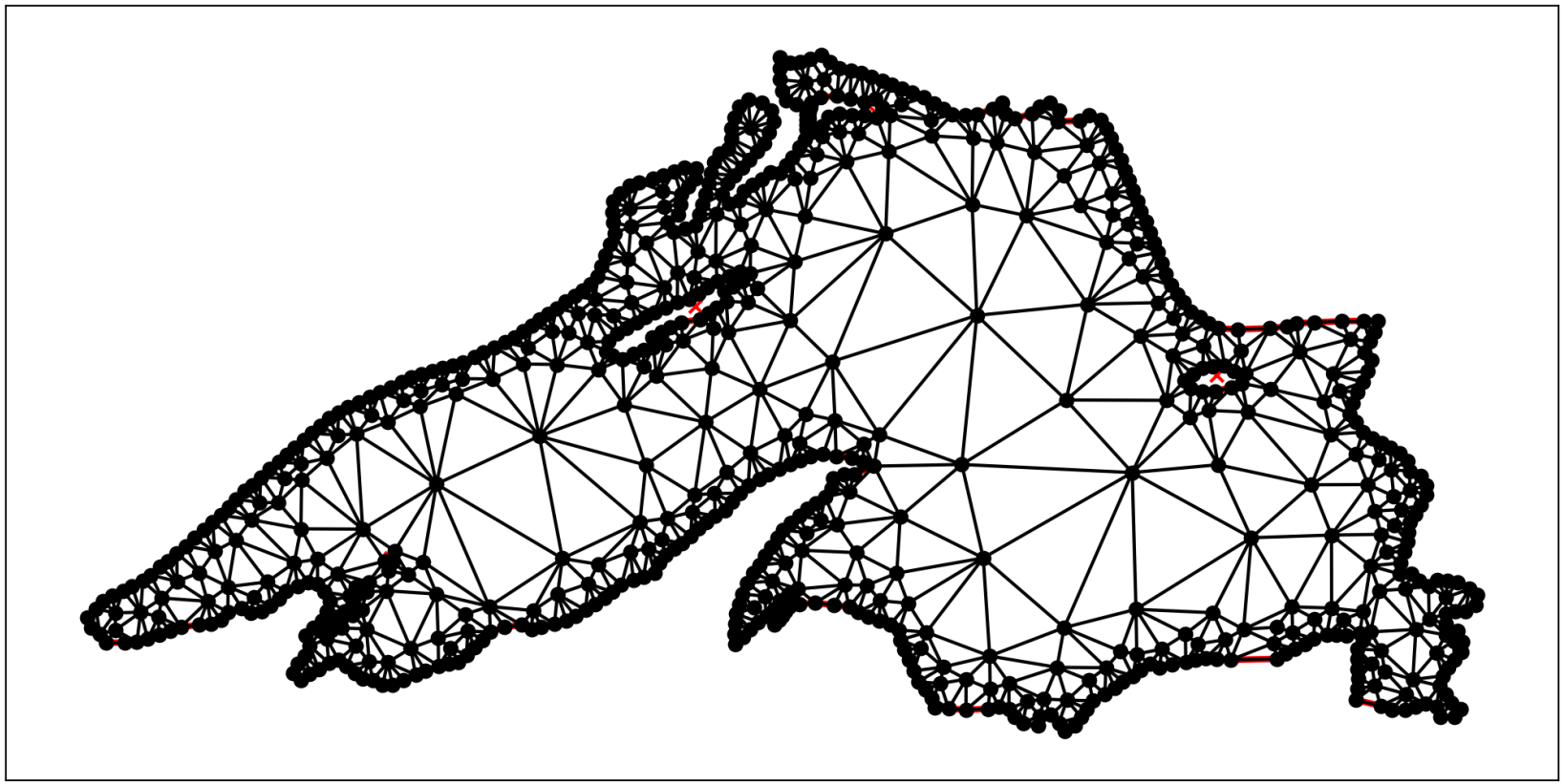
computing a constrained Delaunay triangulation

A constrained conforming Delaunay triangulation (cfdt) is computed with the flag p to indicate that the source is a planar straight line graph and the flag D to enforce Steiner points. Furthermore, we require all triangles to have a minimum angle of at least 20 degrees.

```
from read_poly import read_poly
from matplotlib import pyplot as plt
from triangle import triangulate
from triangle import plot as tplot

lake_superior = read_poly('superior.poly')
cncfq20dt = triangulate(lake_superior, 'pq20D')
ax = plt.subplot(111)
tplot(ax, **cncfq20dt)
plt.show()
```

the conforming Delaunay triangulation



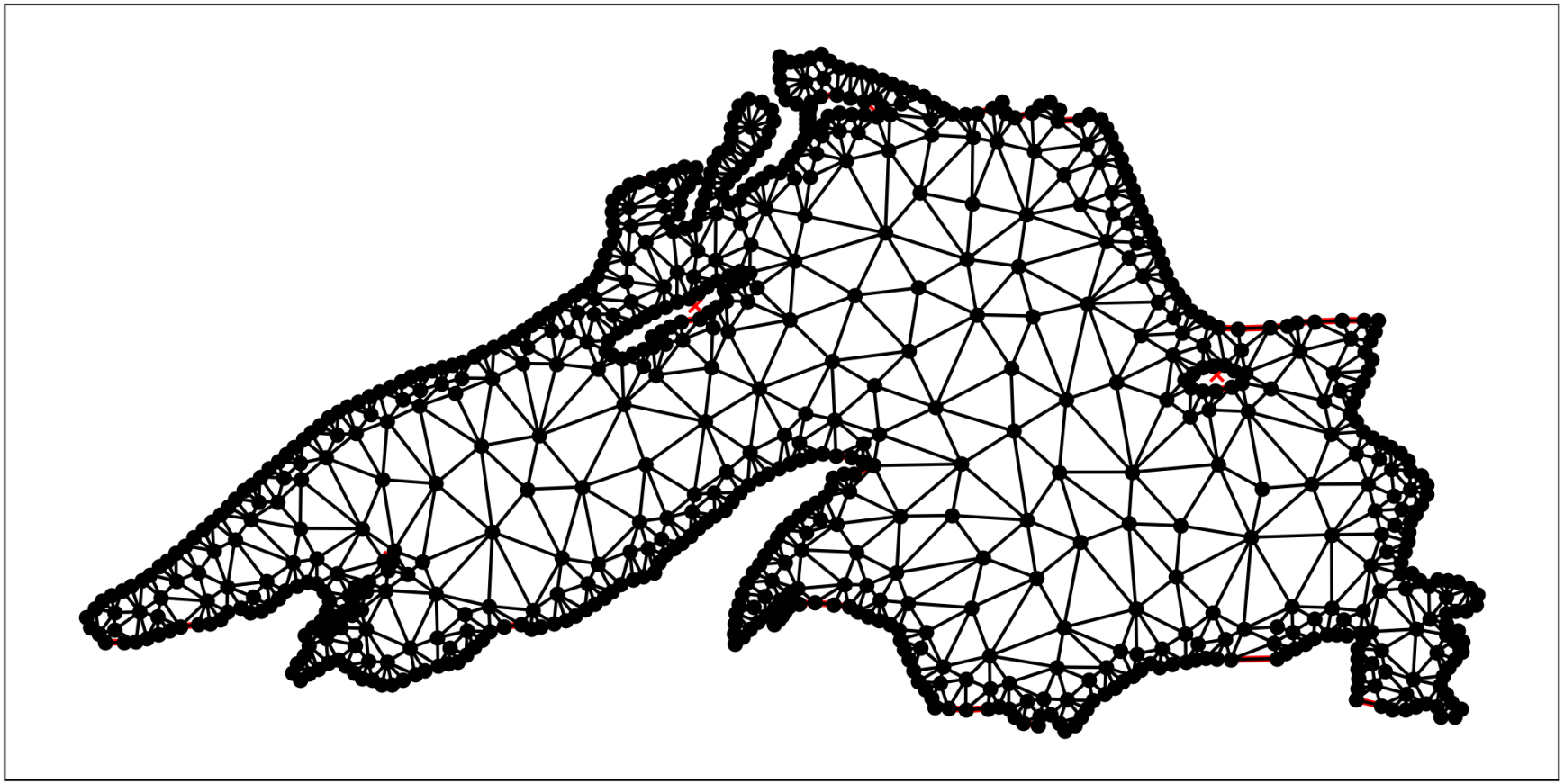
imposing a maximum area on triangles

A constrained conforming Delaunay triangulation (cfdt) is computed with the flag p to indicate that the source is a planar straight line graph and the flag D to enforce Steiner points.
We furthermore require that all triangles have a minimum angle of at least 20 degrees and impose a maximum area on triangles.

```
from read_poly import read_poly
from matplotlib import pyplot as plt
from triangle import triangulate
from triangle import plot as tplot

lake_superior = read_poly('superior.poly')
cncfq20adt = triangulate(lake_superior, 'pq20a.001D')
ax = plt.subplot(111)
tplot(ax, **cncfq20adt)
plt.show()
```

constraints on degrees and areas



Exercises

1. Consider the cube spanned by the points in the list `L`.

   ```
   L = [[0, 0, 0], [1, 0, 0], [0, 1, 0], [1, 1, 0],
        [0, 0, 1], [1, 0, 1], [0, 1, 1], [1, 1, 1]]
   ```

   Apply `ConvexHull` to this list and compute the edges.
   Verify Euler's formula.
   Explain and give an interpretation for the results.

2. Consider five collinear points in the plane,
   compute and plot their Voronoi diagram.
   Describe and explain the plot.

3. A Delaunay triangulation can be defined as the projection of the lower convex hull of the points $(x, y, x^2 + y^2)$.
   Choose a good example to make a drawing to show this definition.