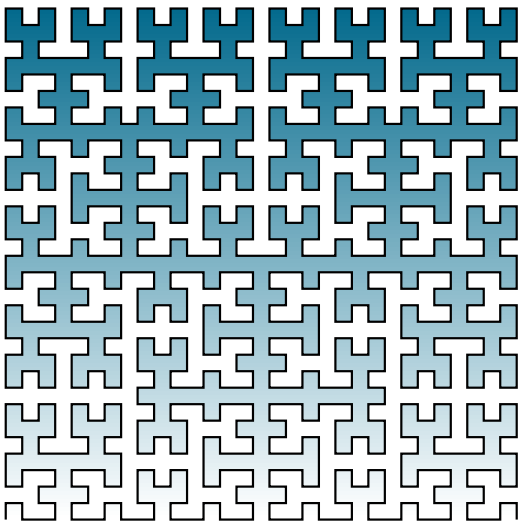
# Computational Geometry in Python

Francisco Blanco-Silva

University of South Carolina



# A QUICK POLL

## LET'S COUNT HANDS

- Are you a frequent `python` user? (at least once a week?)

# A QUICK POLL

## LET'S COUNT HANDS

- Are you a frequent `python` user? (at least once a week?)
- Have you ever used **interactive** `python` (a.k.a. `ipython`)?

# A QUICK POLL

## LET'S COUNT HANDS

- Are you a frequent `python` user? (at least once a week?)
- Have you ever used **interactive** `python` (a.k.a. `ipython`)?
- Have you ever used an `ipython` notebook?

# A QUICK POLL

## LET'S COUNT HANDS

- Are you a frequent `python` user? (at least once a week?)
- Have you ever used **interactive** `python` (a.k.a. `ipython`)?
- Have you ever used an `ipython` notebook?
- Do you use any of the routines/functions/classes in the `numpy` library?

# A QUICK POLL

## LET'S COUNT HANDS

- Are you a frequent `python` user? (at least once a week?)
- Have you ever used **interactive** `python` (a.k.a. `ipython`)?
- Have you ever used an `ipython` notebook?
- Do you use any of the routines/functions/classes in the `numpy` library?
- Have you ever used any of the `scipy` modules for your scientific computing?

# A QUICK POLL

## LET'S COUNT HANDS

- Are you a frequent `python` user? (at least once a week?)
- Have you ever used **interactive** `python` (a.k.a. `ipython`)?
- Have you ever used an `ipython` notebook?
- Do you use any of the routines/functions/classes in the `numpy` library?
- Have you ever used any of the `scipy` modules for your scientific computing?
- Have you ever used the library `pandas` for your data analysis?

# A QUICK POLL

## LET'S COUNT HANDS

- Are you a frequent `python` user? (at least once a week?)
- Have you ever used **interactive** `python` (a.k.a. `ipython`)?
- Have you ever used an `ipython` notebook?
- Do you use any of the routines/functions/classes in the `numpy` library?
- Have you ever used any of the `scipy` modules for your scientific computing?
- Have you ever used the library `pandas` for your data analysis?
- Symbolic computations with the `sympy` libraries?

# A QUICK POLL

## LET'S COUNT HANDS

- Are you a frequent `python` user? (at least once a week?)
- Have you ever used **interactive** `python` (a.k.a. `ipython`)?
- Have you ever used an `ipython` notebook?
- Do you use any of the routines/functions/classes in the `numpy` library?
- Have you ever used any of the `scipy` modules for your scientific computing?
- Have you ever used the library `pandas` for your data analysis?
- Symbolic computations with the `sympy` libraries?
- Are your diagrams generated with `matplotlib`, `plotly`, or `mayavi`?

# A QUICK POLL

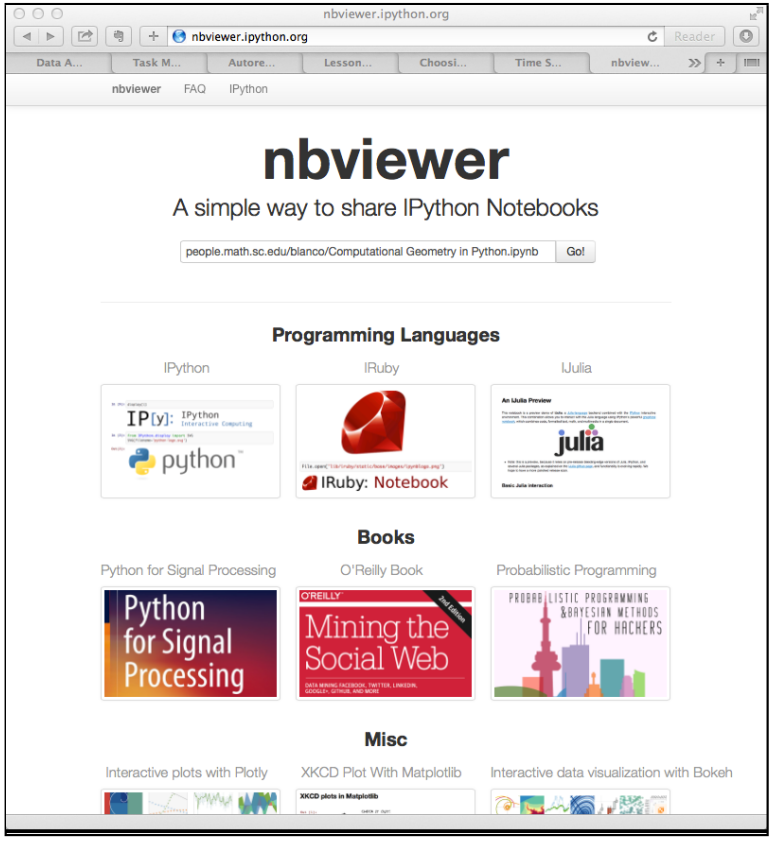
## LET'S COUNT HANDS

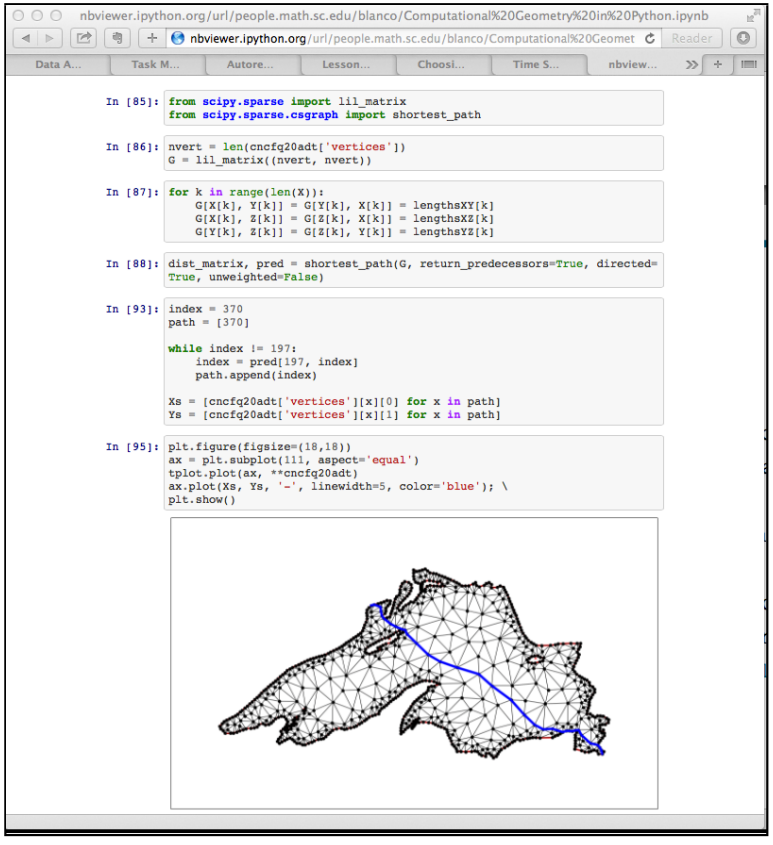
- Are you a frequent `python` user? (at least once a week?)
- Have you ever used **interactive** `python` (a.k.a. `ipython`)?
- Have you ever used an `ipython` notebook?
- Do you use any of the routines/functions/classes in the `numpy` library?
- Have you ever used any of the `scipy` modules for your scientific computing?
- Have you ever used the library `pandas` for your data analysis?
- Symbolic computations with the `sympy` libraries?
- Are your diagrams generated with `matplotlib`, `plotly`, or `mayavi`?
- Do you use any of the `scipy` toolkits?

# Computational Geometry in Python

The contents of this presentation are a modified and reduced version of chapter 6 of the upcoming book Mastering Scipy. They may be followed simultaneously as an `ipython` notebook at `nbviewer.ipython.org`, by requesting:

`people.math.sc.edu/blanco/Computational Geometry in Python.ipynb`



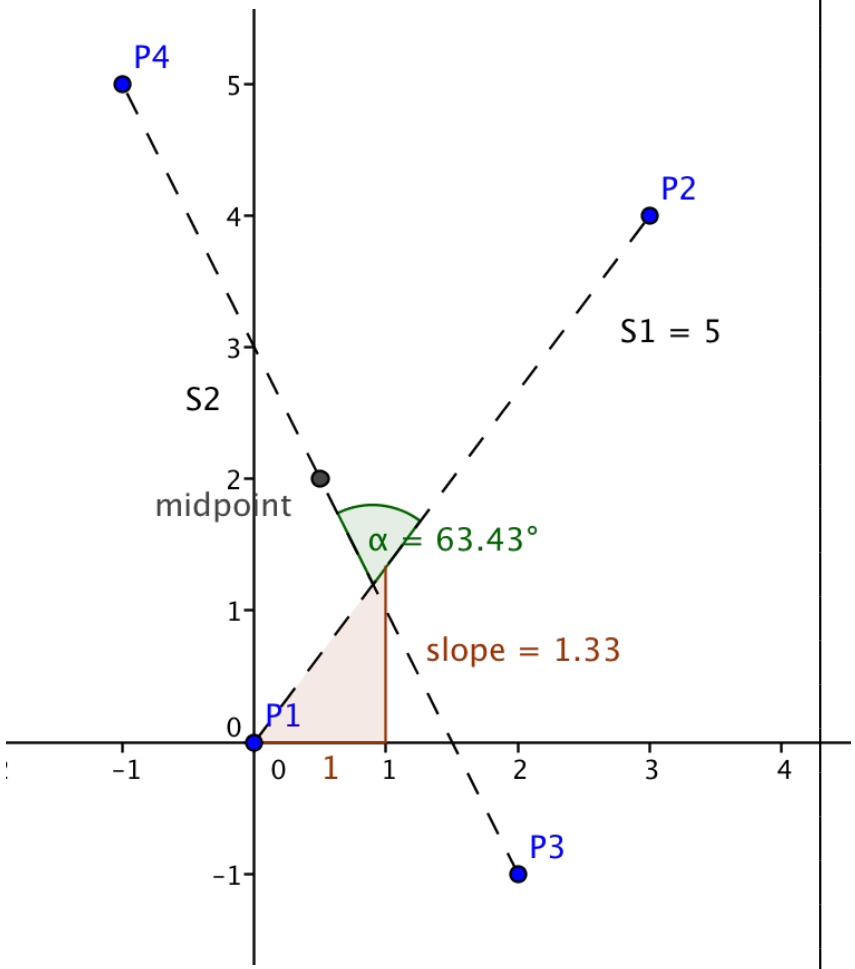
# Computational Geometry in Python

- Plane Geometry
  - Points, Segments
  - Lines
  - Circles
  - Triangles
  - Curves
  - Affine Transformations
- Combinatorial Computational Geometry
  - Static Problems
    - Convex Hulls
    - Voronoi Diagrams
    - Triangulations
    - Shortest Paths
  - Geometric Query Problems
    - Point Location
    - Nearest Neighbor
    - Range Searching
  - Dynamic Problems
- Numerical Computational Geometry
  - Bézier Curves

# Plane Geometry

## Basic objects, properties, methods

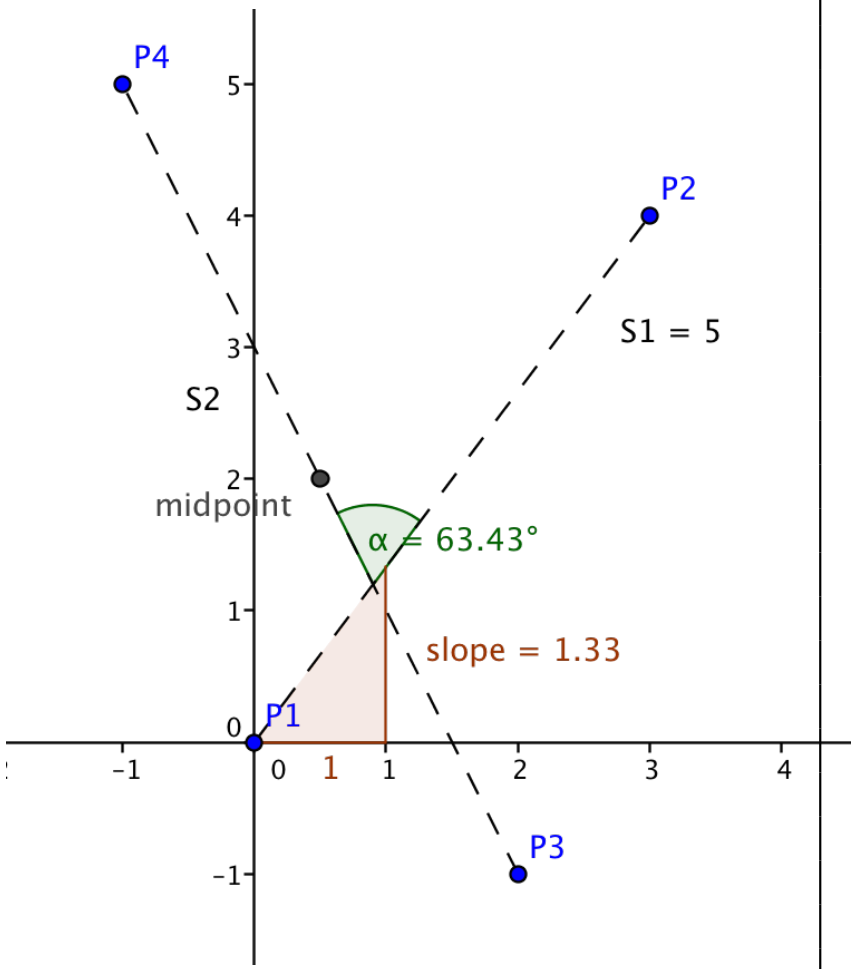
```
1  # Preamble
2  from sympy.geometry import *
3  # Creation of some points
4  P1 = Point(0, 0)
5  P2 = Point(3, 4)
6  P3 = Point(2, -1)
7  P4 = Point(-1, 5)
8  # Creation of some segments
9  S1 = Segment(P1, P2)
10 S2 = Segment(P3, P4)
11 # Let's ask some questions
12 >>> Point.is_collinear(P1, P2, P3)
13 False
14 >>> S1.length
15 5
16 >>> S2.midpoint
17 Point(1/2, 2)
18 >>> S1.slope
19 4/3
20 >>> S1.intersection(S2)
21 [Point(9/10, 6/5)]
22 >>> Segment.angle_between(S1, S2)
23 acos(-sqrt(5)/5)
24 >>> S1.contains(P3)
25 False
```



# Plane Geometry

## Basic objects, properties, methods

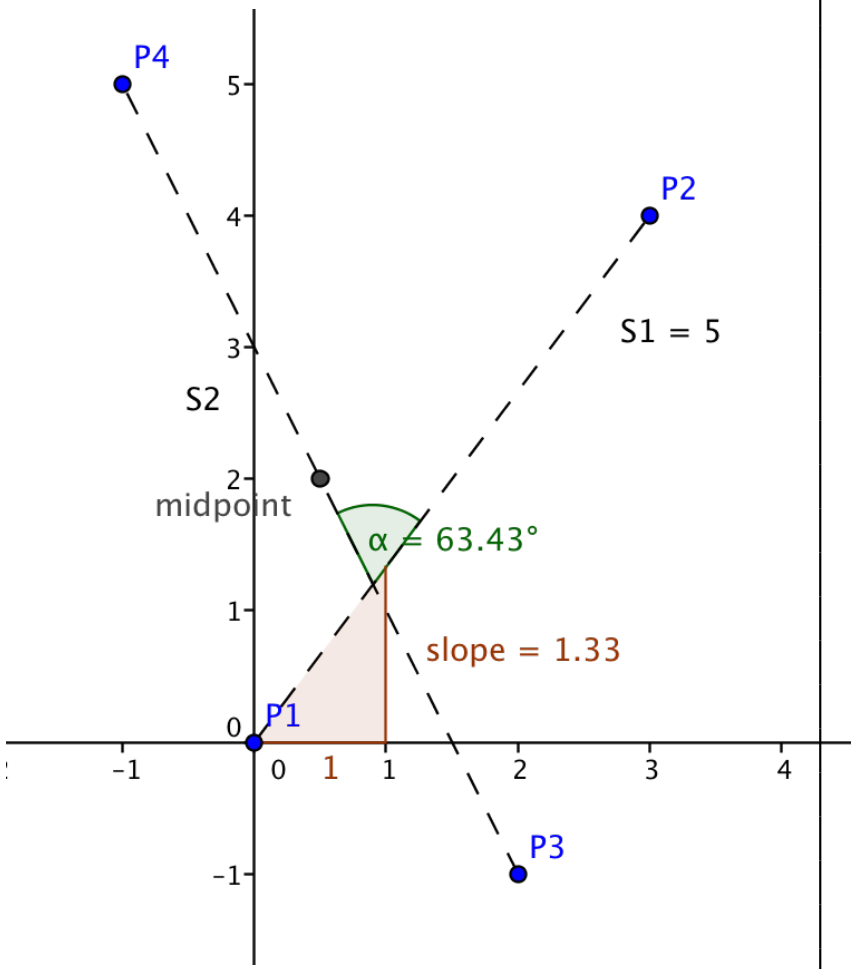
```
1  # Preamble
2  from sympy.geometry import *
3  # Creation of some points
4  P1 = Point(0, 0)
5  P2 = Point(3, 4)
6  P3 = Point(2, -1)
7  P4 = Point(-1, 5)
8  # Creation of some segments
9  S1 = Segment(P1, P2)
10 S2 = Segment(P3, P4)
11 # Let's ask some questions
12 >>> Point.is_collinear(P1, P2, P3)
13 False
14 >>> S1.length
15 5
16 >>> S2.midpoint
17 Point(1/2, 2)
18 >>> S1.slope
19 4/3
20 >>> S1.intersection(S2)
21 [Point(9/10, 6/5)]
22 >>> Segment.angle_between(S1, S2)
23 acos(-sqrt(5)/5)
24 >>> S1.contains(P3)
25 False
```



# Plane Geometry

## Basic objects, properties, methods

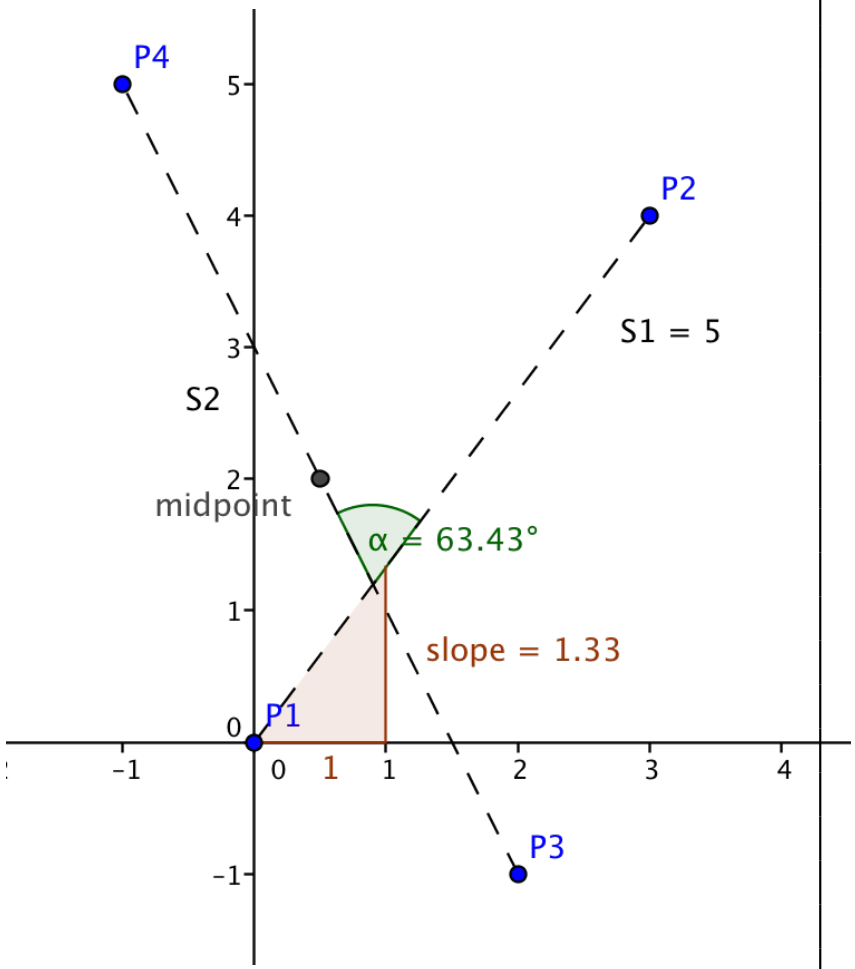
```
1  # Preamble
2  from sympy.geometry import *
3  # Creation of some points
4  P1 = Point(0, 0)
5  P2 = Point(3, 4)
6  P3 = Point(2, -1)
7  P4 = Point(-1, 5)
8  # Creation of some segments
9  S1 = Segment(P1, P2)
10 S2 = Segment(P3, P4)
11 # Let's ask some questions
12 >>> Point.is_collinear(P1, P2, P3)
13 False
14 >>> S1.length
15 5
16 >>> S2.midpoint
17 Point(1/2, 2)
18 >>> S1.slope
19 4/3
20 >>> S1.intersection(S2)
21 [Point(9/10, 6/5)]
22 >>> Segment.angle_between(S1, S2)
23 acos(-sqrt(5)/5)
24 >>> S1.contains(P3)
25 False
```



# Plane Geometry

## Basic objects, properties, methods

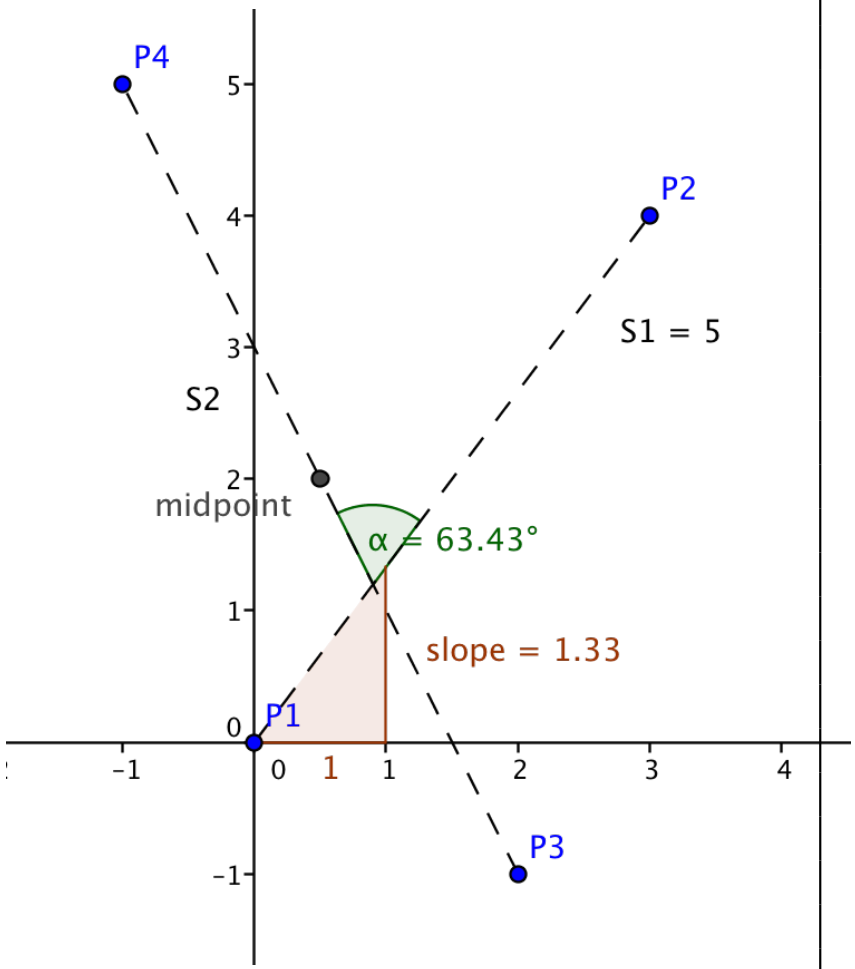
```
1  # Preamble
2  from sympy.geometry import *
3  # Creation of some points
4  P1 = Point(0, 0)
5  P2 = Point(3, 4)
6  P3 = Point(2, -1)
7  P4 = Point(-1, 5)
8  # Creation of some segments
9  S1 = Segment(P1, P2)
10 S2 = Segment(P3, P4)
11 # Let's ask some questions
12 >>> Point.is_collinear(P1, P2, P3)
13 False
14 >>> S1.length
15 5
16 >>> S2.midpoint
17 Point(1/2, 2)
18 >>> S1.slope
19 4/3
20 >>> S1.intersection(S2)
21 [Point(9/10, 6/5)]
22 >>> Segment.angle_between(S1, S2)
23 acos(-sqrt(5)/5)
24 >>> S1.contains(P3)
25 False
```



# Plane Geometry

## Basic objects, properties, methods

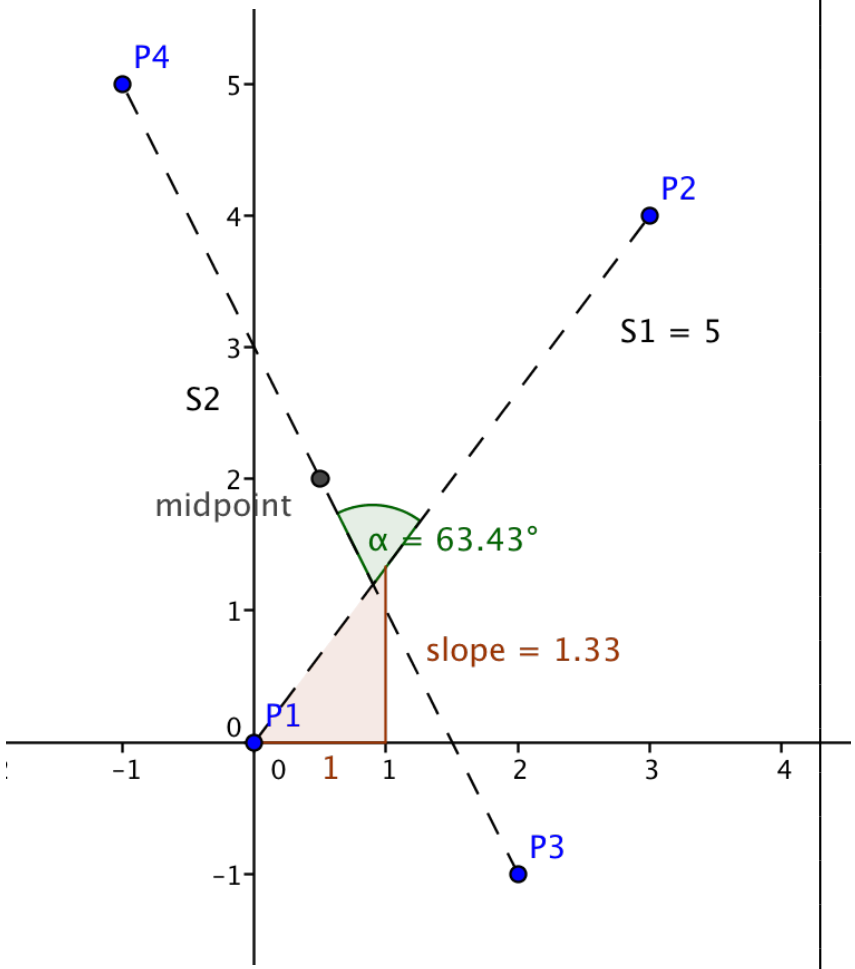
```
1  # Preamble
2  from sympy.geometry import *
3  # Creation of some points
4  P1 = Point(0, 0)
5  P2 = Point(3, 4)
6  P3 = Point(2, -1)
7  P4 = Point(-1, 5)
8  # Creation of some segments
9  S1 = Segment(P1, P2)
10 S2 = Segment(P3, P4)
11 # Let's ask some questions
12 >>> Point.is_collinear(P1, P2, P3)
13 False
14 >>> S1.length
15 5
16 >>> S2.midpoint
17 Point(1/2, 2)
18 >>> S1.slope
19 4/3
20 >>> S1.intersection(S2)
21 [Point(9/10, 6/5)]
22 >>> Segment.angle_between(S1, S2)
23 acos(-sqrt(5)/5)
24 >>> S1.contains(P3)
25 False
```



# Plane Geometry

## Basic objects, properties, methods

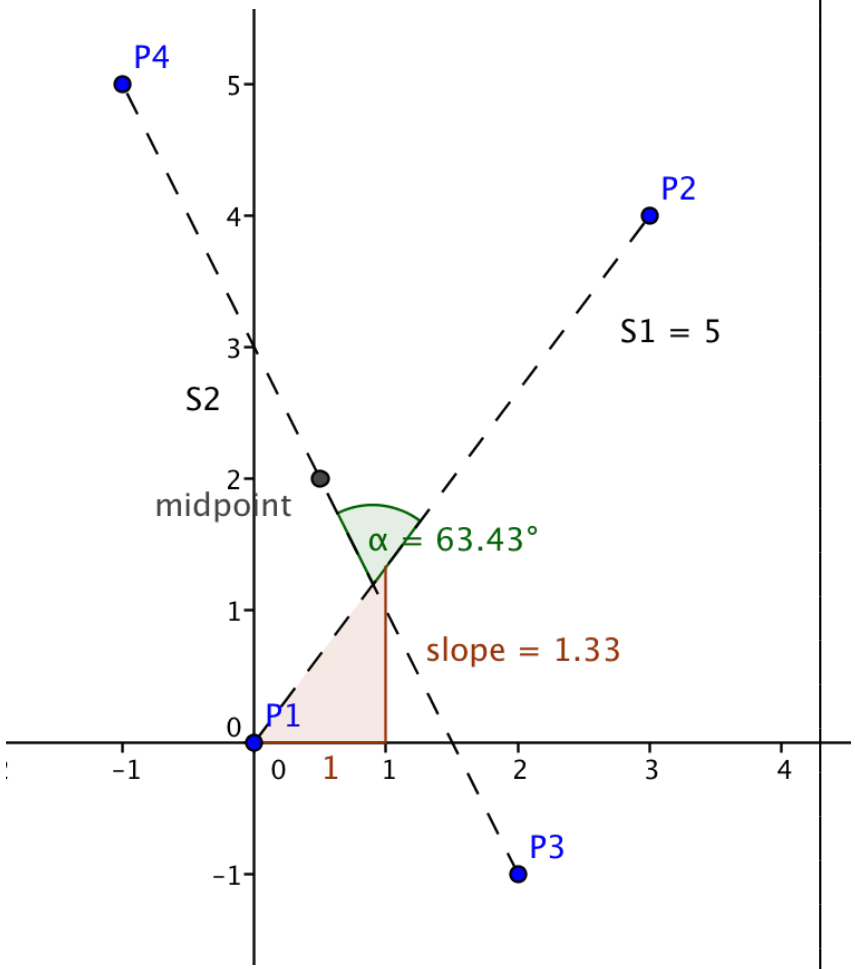
```
# Preamble
from sympy.geometry import *
# Creation of some points
P1 = Point(0, 0)
P2 = Point(3, 4)
P3 = Point(2, -1)
P4 = Point(-1, 5)
# Creation of some segments
S1 = Segment(P1, P2)
S2 = Segment(P3, P4)
# Let's ask some questions
>>> Point.is_collinear(P1, P2, P3)
False
>>> S1.length
5
>>> S2.midpoint
Point(1/2, 2)
>>> S1.slope
4/3
>>> S1.intersection(S2)
[Point(9/10, 6/5)]
>>> Segment.angle_between(S1, S2)
acos(-sqrt(5)/5)
>>> S1.contains(P3)
False
```



# Plane Geometry

## Basic objects, properties, methods

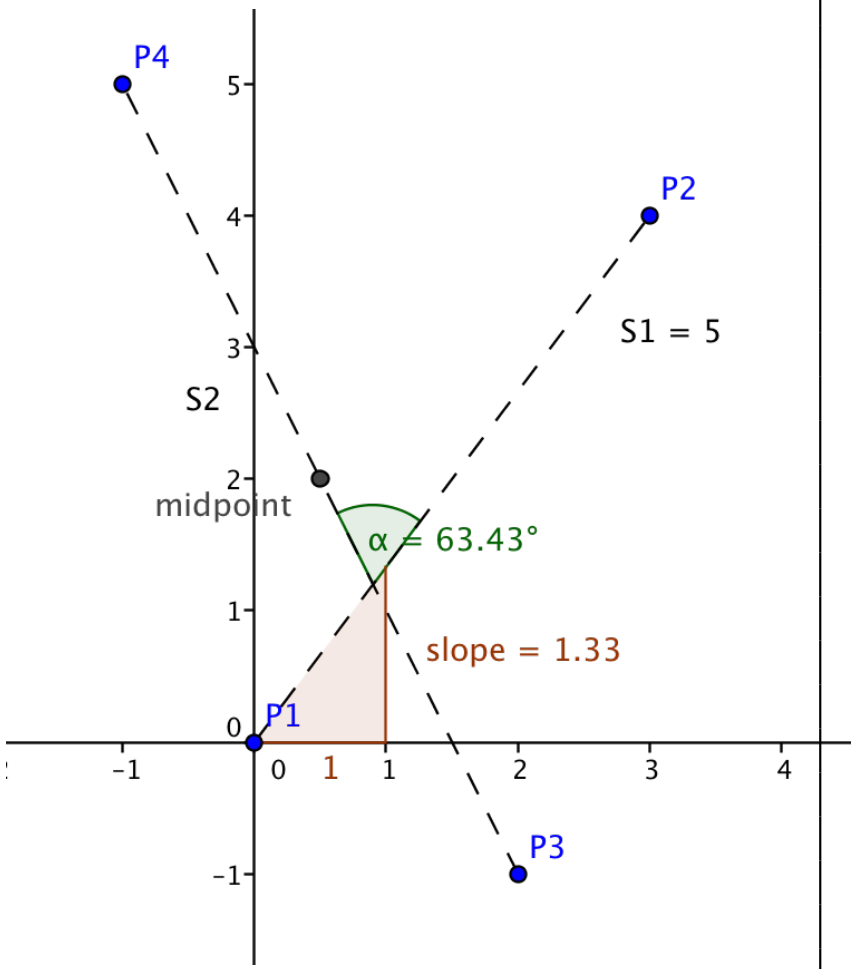
```
1  # Preamble
2  from sympy.geometry import *
3  # Creation of some points
4  P1 = Point(0, 0)
5  P2 = Point(3, 4)
6  P3 = Point(2, -1)
7  P4 = Point(-1, 5)
8  # Creation of some segments
9  S1 = Segment(P1, P2)
10 S2 = Segment(P3, P4)
11 # Let's ask some questions
12 >>> Point.is_collinear(P1, P2, P3)
13 False
14 >>> S1.length
15 5
16 >>> S2.midpoint
17 Point(1/2, 2)
18 >>> S1.slope
19 4/3
20 >>> S1.intersection(S2)
21 [Point(9/10, 6/5)]
22 >>> Segment.angle_between(S1, S2)
23 acos(-sqrt(5)/5)
24 >>> S1.contains(P3)
25 False
```



# Plane Geometry

## Basic objects, properties, methods

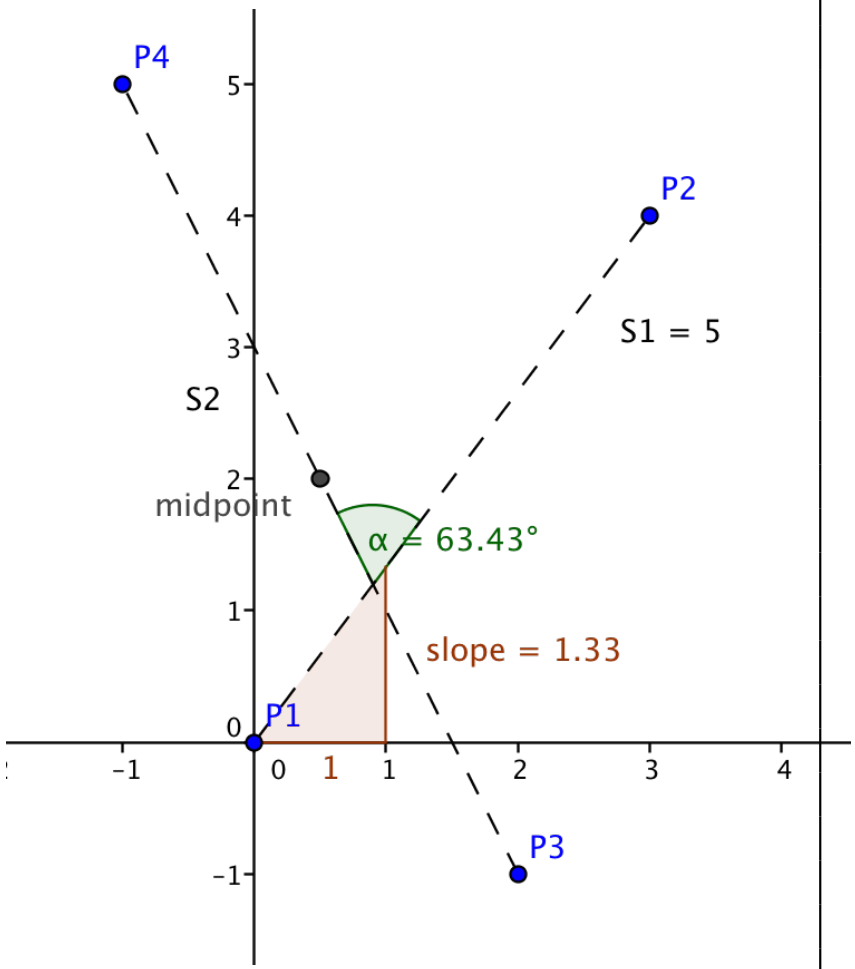
```
1  # Preamble
2  from sympy.geometry import *
3  # Creation of some points
4  P1 = Point(0, 0)
5  P2 = Point(3, 4)
6  P3 = Point(2, -1)
7  P4 = Point(-1, 5)
8  # Creation of some segments
9  S1 = Segment(P1, P2)
10 S2 = Segment(P3, P4)
11 # Let's ask some questions
12 >>> Point.is_collinear(P1, P2, P3)
13 False
14 >>> S1.length
15 5
16 >>> S2.midpoint
17 Point(1/2, 2)
18 >>> S1.slope
19 4/3
20 >>> S1.intersection(S2)
21 [Point(9/10, 6/5)]
22 >>> Segment.angle_between(S1, S2)
23 acos(-sqrt(5)/5)
24 >>> S1.contains(P3)
25 False
```



# Plane Geometry

## Basic objects, properties, methods

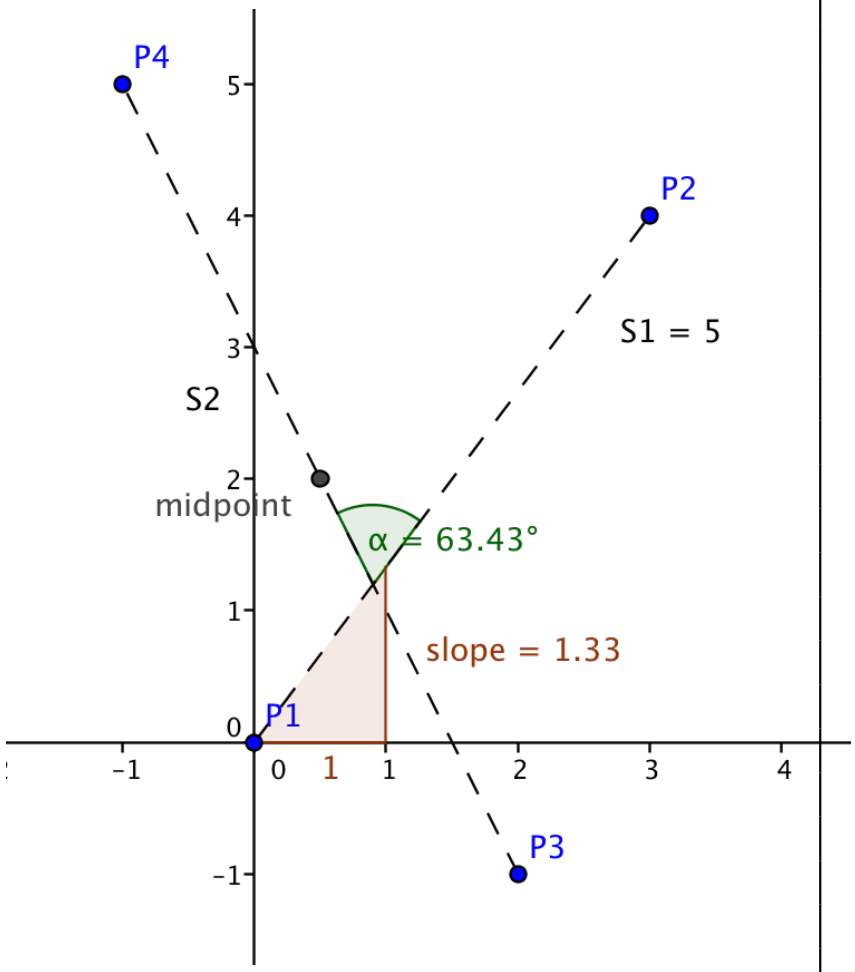
```
# Preamble
from sympy.geometry import *
# Creation of some points
P1 = Point(0, 0)
P2 = Point(3, 4)
P3 = Point(2, -1)
P4 = Point(-1, 5)
# Creation of some segments
S1 = Segment(P1, P2)
S2 = Segment(P3, P4)
# Let's ask some questions
>>> Point.is_collinear(P1, P2, P3)
False
>>> S1.length
5
>>> S2.midpoint
Point(1/2, 2)
>>> S1.slope
4/3
>>> S1.intersection(S2)
[Point(9/10, 6/5)]
>>> Segment.angle_between(S1, S2)
acos(-sqrt(5)/5)
>>> S1.contains(P3)
False
```



# Plane Geometry

## Basic objects, properties, methods

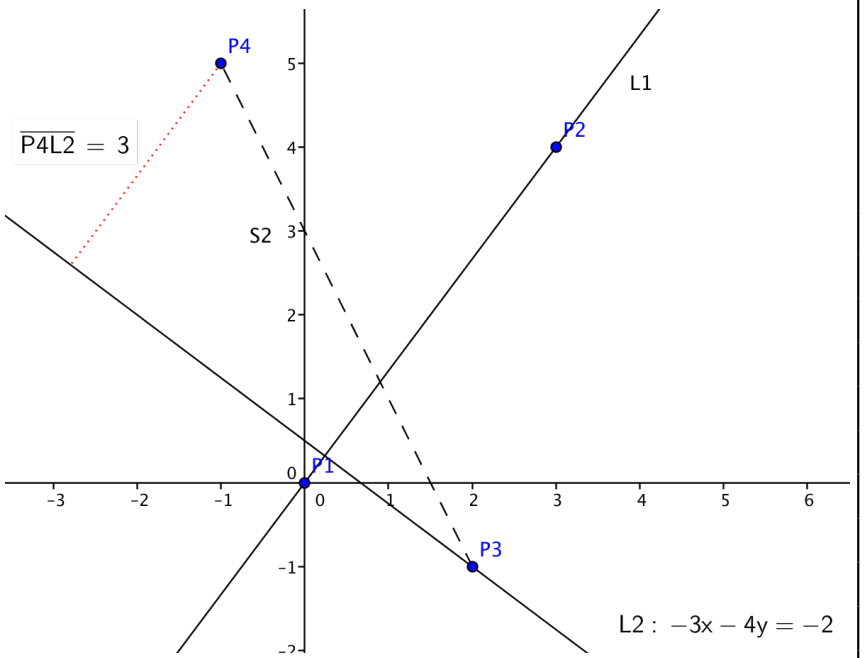
```
1  # Preamble
2  from sympy.geometry import *
3  # Creation of some points
4  P1 = Point(0, 0)
5  P2 = Point(3, 4)
6  P3 = Point(2, -1)
7  P4 = Point(-1, 5)
8  # Creation of some segments
9  S1 = Segment(P1, P2)
10 S2 = Segment(P3, P4)
11 # Let's ask some questions
12 >>> Point.is_collinear(P1, P2, P3)
13 False
14 >>> S1.length
15 5
16 >>> S2.midpoint
17 Point(1/2, 2)
18 >>> S1.slope
19 4/3
20 >>> S1.intersection(S2)
21 [Point(9/10, 6/5)]
22 >>> Segment.angle_between(S1, S2)
23 acos(-sqrt(5)/5)
24 >>> S1.contains(P3)
25 False
```



# Plane Geometry

## Basic Objects, properties, methods

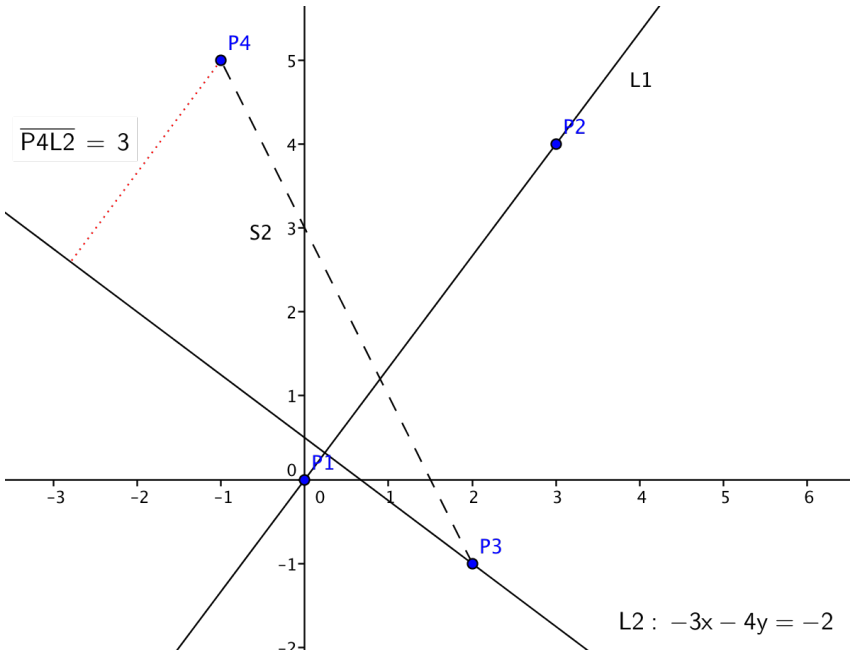
```
1  # Creation of some lines
2  L1 = Line(P1, P2)
3  L2 = L1.perpendicular_line(P3)
4  # Equations of lines
5  >>> L2.arbitrary_point()
6  Point(4*t + 2, -3*t - 1)
7  >>> L2.equation()
8  3*x + 4*y - 2
9  # Let's ask some questions
10 >>> L2.contains(P4)
11 False
12 >>> L2.distance(P4)
13 3
14 >>> L1.is_parallel(S2)
15 False
```



# Plane Geometry

## Basic Objects, properties, methods

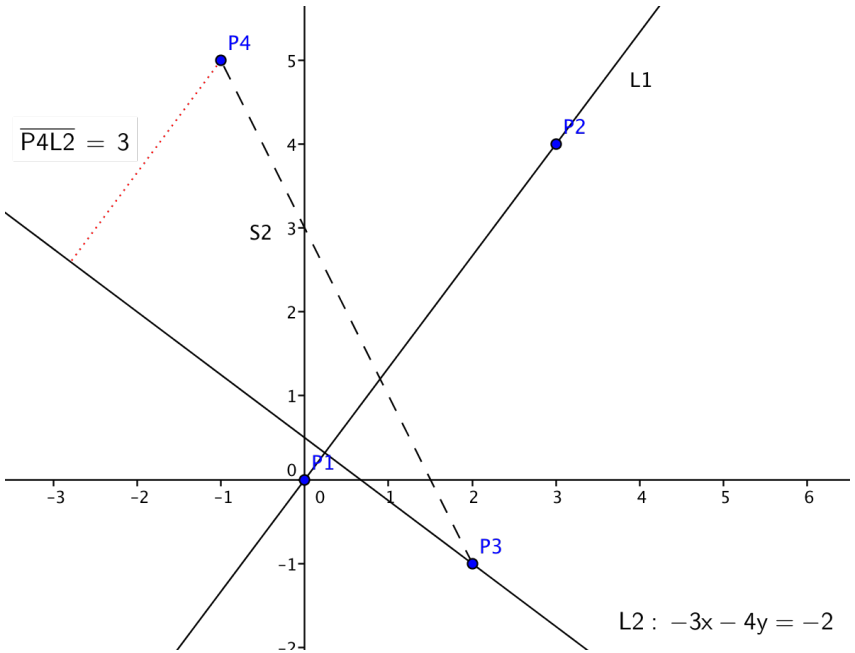
```
1  # Creation of some lines
2  L1 = Line(P1, P2)
3  L2 = L1.perpendicular_line(P3)
4  # Equations of lines
5  >>> L2.arbitrary_point()
6  Point(4*t + 2, -3*t - 1)
7  >>> L2.equation()
8  3*x + 4*y - 2
9  # Let's ask some questions
10 >>> L2.contains(P4)
11 False
12 >>> L2.distance(P4)
13 3
14 >>> L1.is_parallel(S2)
15 False
```



# Plane Geometry

## Basic Objects, properties, methods

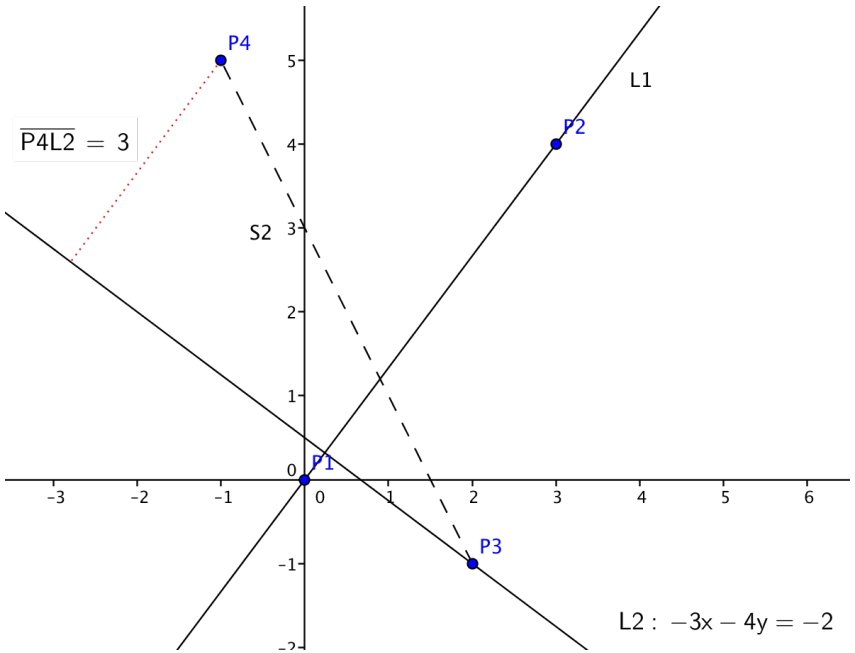
```
1  # Creation of some lines
2  L1 = Line(P1, P2)
3  L2 = L1.perpendicular_line(P3)
4  # Equations of lines
5  >>> L2.arbitrary_point()
6  Point(4*t + 2, -3*t - 1)
7  >>> L2.equation()
8  3*x + 4*y - 2
9  # Let's ask some questions
10 >>> L2.contains(P4)
11 False
12 >>> L2.distance(P4)
13 3
14 >>> L1.is_parallel(S2)
15 False
```



# Plane Geometry

## Basic Objects, properties, methods

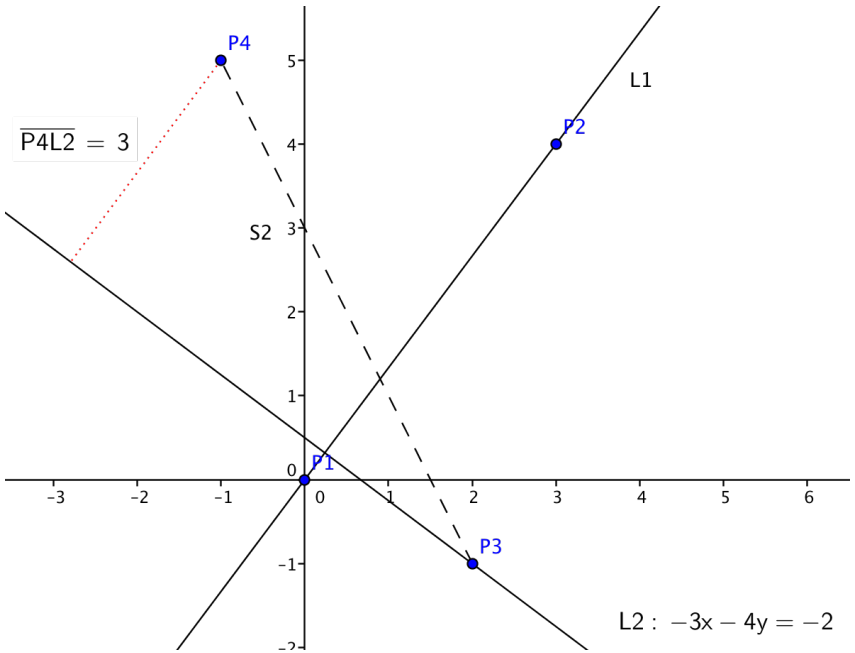
```
# Creation of some lines
L1 = Line(P1, P2)
L2 = L1.perpendicular_line(P3)
# Equations of lines
>>> L2.arbitrary_point()
Point(4*t + 2, -3*t - 1)
>>> L2.equation()
3*x + 4*y - 2
# Let's ask some questions
>>> L2.contains(P4)
False
>>> L2.distance(P4)
3
>>> L1.is_parallel(S2)
False
```



# Plane Geometry

## Basic Objects, properties, methods

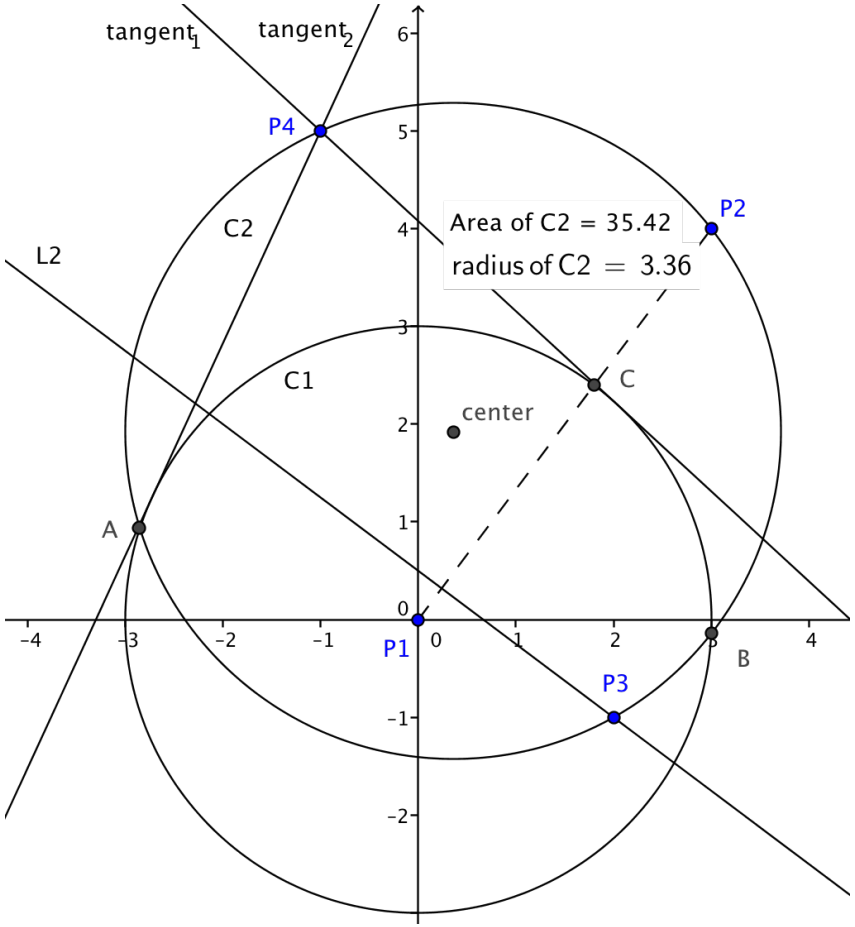
```
# Creation of some lines
L1 = Line(P1, P2)
L2 = L1.perpendicular_line(P3)
# Equations of lines
>>> L2.arbitrary_point()
Point(4*t + 2, -3*t - 1)
>>> L2.equation()
3*x + 4*y - 2
# Let's ask some questions
>>> L2.contains(P4)
False
>>> L2.distance(P4)
3
>>> L1.is_parallel(S2)
False
```



# Plane Geometry

## Basic Objects, properties, methods

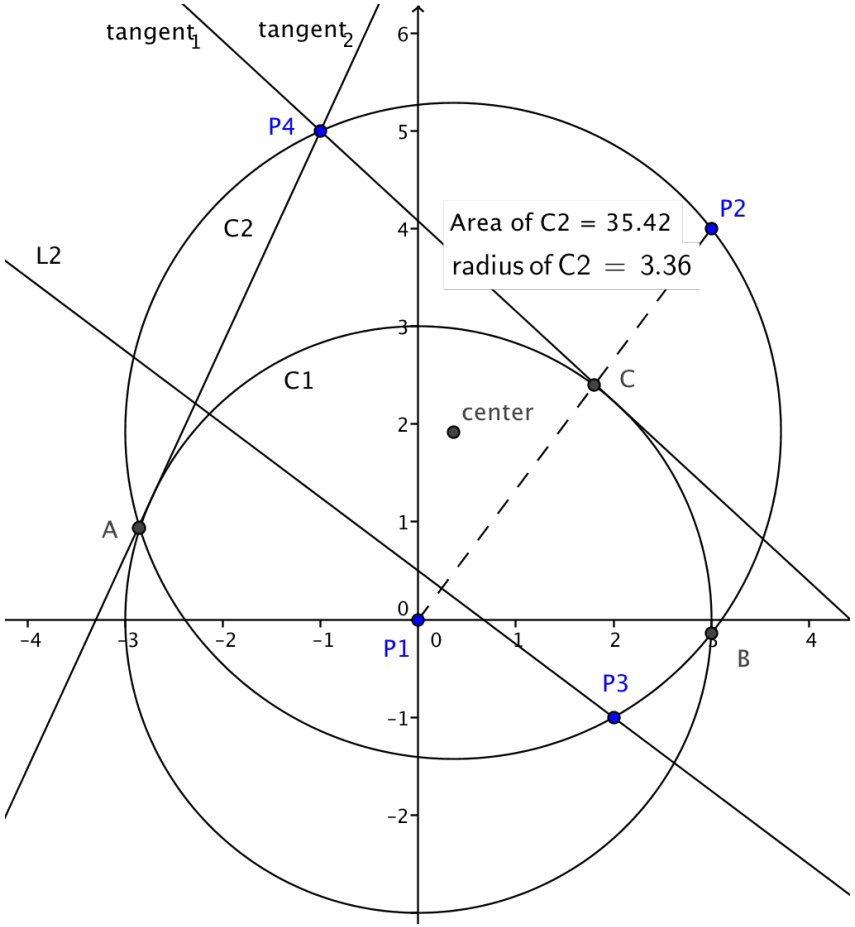
```
# Creation of some circles
C1 = Circle(P1, 3)
C2 = Circle(P2, P3, P4)
# Equations
>>> C2.equation()
(x - 5/14)**2 + ...
# Let's ask some questions
>>> C2.area
1105*pi/98
>>> C2.radius
sqrt(2210)/14
>>> C2.center
Point(5/14, 27/14)
>>> C2.circumference
sqrt(2210)*pi/7
>>> C1.tangent_lines(P4)
[Line ...,
 Line ...]
# Intersections with other objects
>>> C2.intersection(C1)
[Point(55/754 + ...
 Point(55/754 - ...)]
>>> C2.intersection(S1)
[Point(3, 4)]
>>> C2.is_tangent(L2)
False
```



# Plane Geometry

## Basic Objects, properties, methods

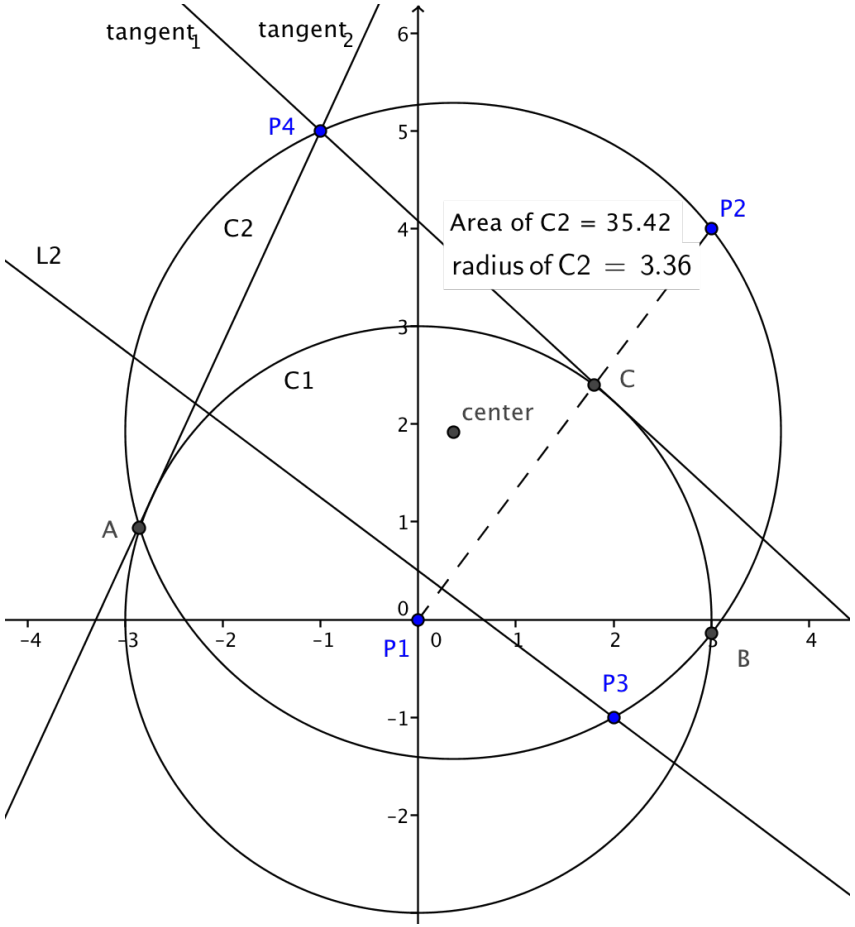
```
# Creation of some circles
C1 = Circle(P1, 3)
C2 = Circle(P2, P3, P4)
# Equations
>>> C2.equation()
(x - 5/14)**2 + ...
# Let's ask some questions
>>> C2.area
1105*pi/98
>>> C2.radius
sqrt(2210)/14
>>> C2.center
Point(5/14, 27/14)
>>> C2.circumference
sqrt(2210)*pi/7
>>> C1.tangent_lines(P4)
[Line ...,
 Line ...]
# Intersections with other objects
>>> C2.intersection(C1)
[Point(55/754 + ...
 Point(55/754 - ...)]
>>> C2.intersection(S1)
[Point(3, 4)]
>>> C2.is_tangent(L2)
False
```



# Plane Geometry

## Basic Objects, properties, methods

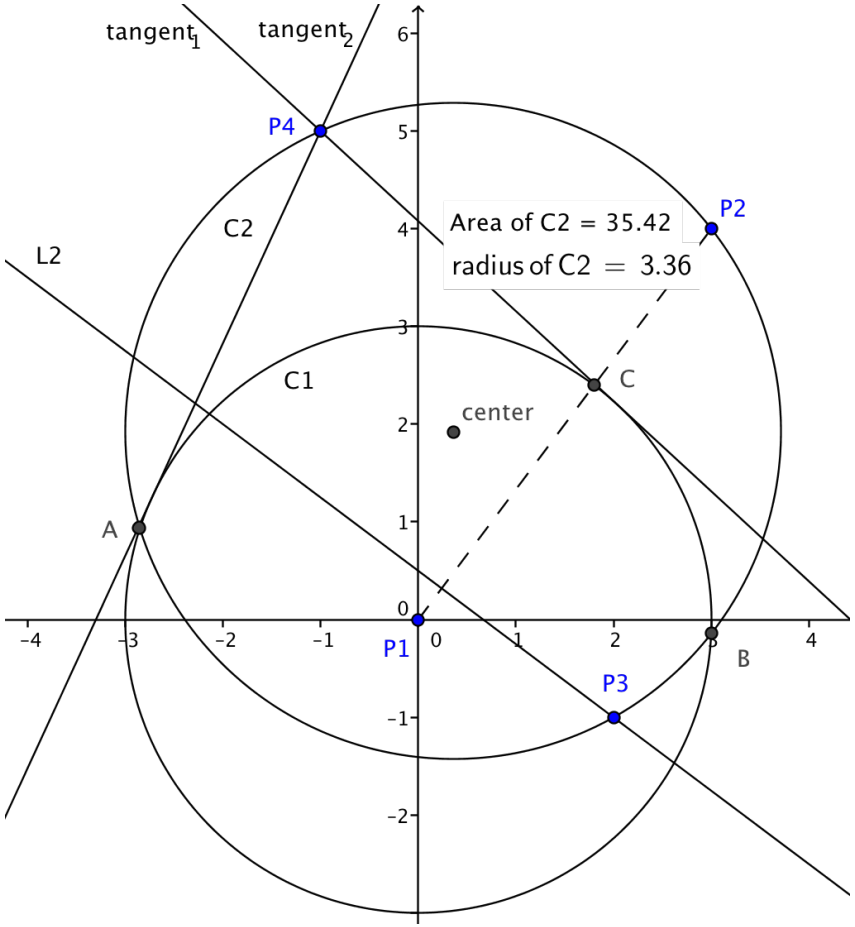
```
# Creation of some circles
C1 = Circle(P1, 3)
C2 = Circle(P2, P3, P4)
# Equations
>>> C2.equation()
(x - 5/14)**2 + ...
# Let's ask some questions
>>> C2.area
1105*pi/98
>>> C2.radius
sqrt(2210)/14
>>> C2.center
Point(5/14, 27/14)
>>> C2.circumference
sqrt(2210)*pi/7
>>> C1.tangent_lines(P4)
[Line ...,
 Line ...]
# Intersections with other objects
>>> C2.intersection(C1)
[Point(55/754 + ...
 Point(55/754 - ...)]
>>> C2.intersection(S1)
[Point(3, 4)]
>>> C2.is_tangent(L2)
False
```



# Plane Geometry

## Basic Objects, properties, methods

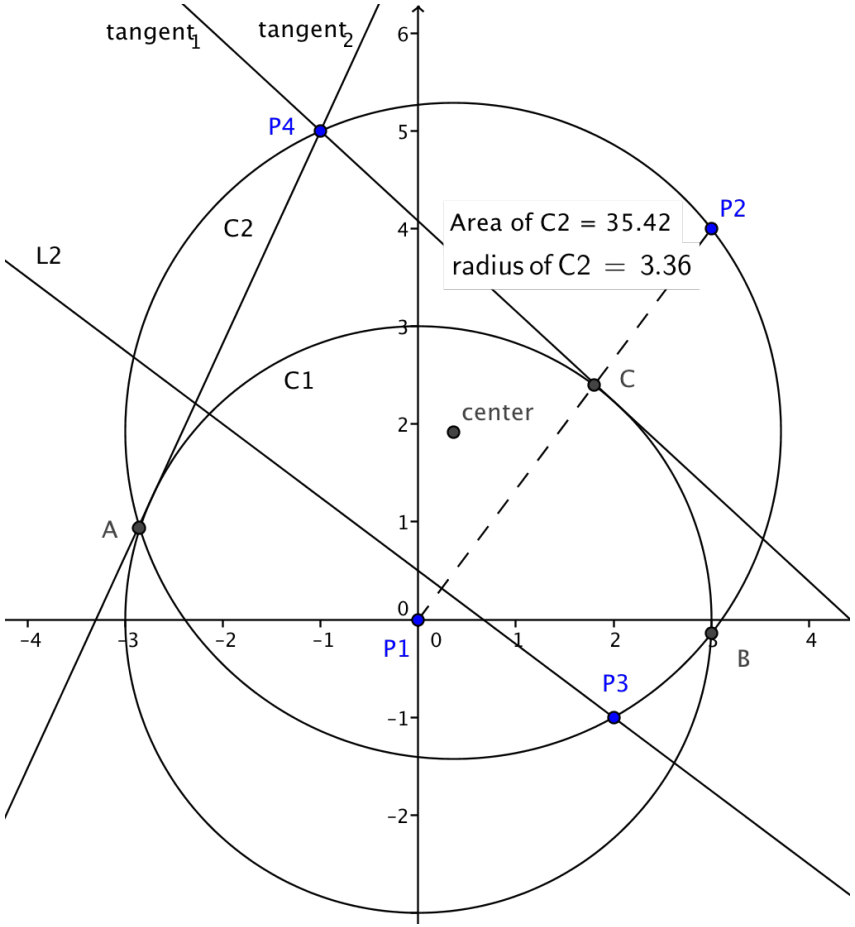
```
# Creation of some circles
C1 = Circle(P1, 3)
C2 = Circle(P2, P3, P4)
# Equations
>>> C2.equation()
(x - 5/14)**2 + ...
# Let's ask some questions
>>> C2.area
1105*pi/98
>>> C2.radius
sqrt(2210)/14
>>> C2.center
Point(5/14, 27/14)
>>> C2.circumference
sqrt(2210)*pi/7
>>> C1.tangent_lines(P4)
[Line ...,
 Line ...]
# Intersections with other objects
>>> C2.intersection(C1)
[Point(55/754 + ...
 Point(55/754 - ...)]
>>> C2.intersection(S1)
[Point(3, 4)]
>>> C2.is_tangent(L2)
False
```



# Plane Geometry

## Basic Objects, properties, methods

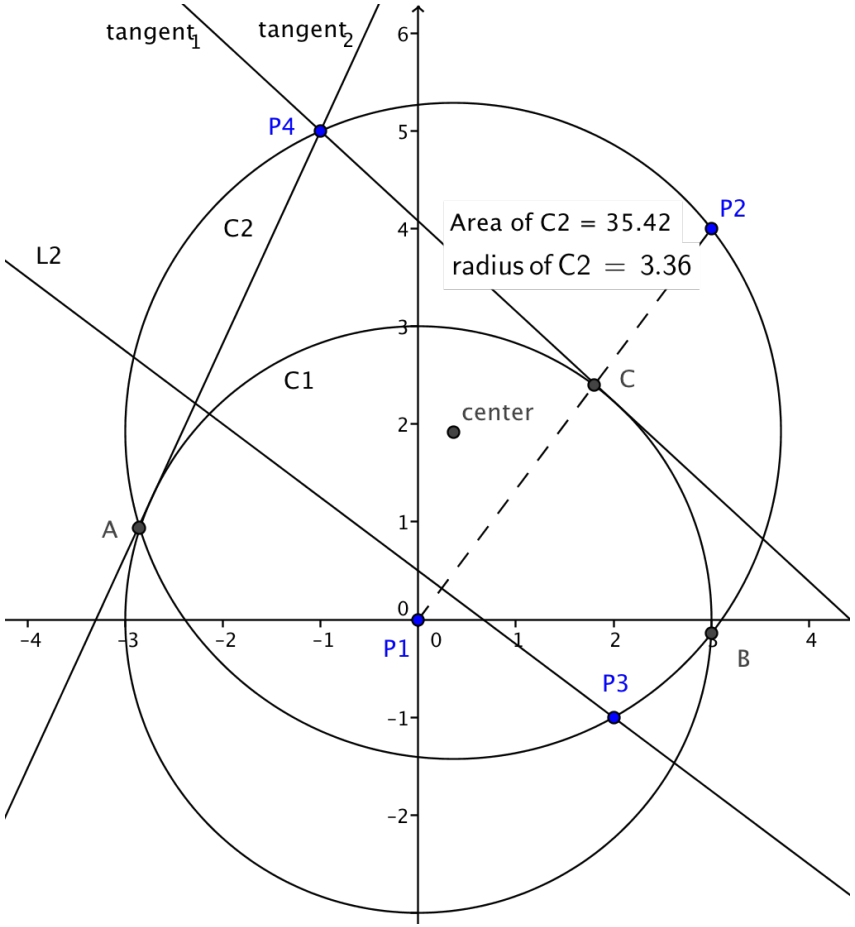
```
# Creation of some circles
C1 = Circle(P1, 3)
C2 = Circle(P2, P3, P4)
# Equations
>>> C2.equation()
(x - 5/14)**2 + ...
# Let's ask some questions
>>> C2.area
1105*pi/98
>>> C2.radius
sqrt(2210)/14
>>> C2.center
Point(5/14, 27/14)
>>> C2.circumference
sqrt(2210)*pi/7
>>> C1.tangent_lines(P4)
[Line ...,
 Line ...]
# Intersections with other objects
>>> C2.intersection(C1)
[Point(55/754 + ...
 Point(55/754 - ...)]
>>> C2.intersection(S1)
[Point(3, 4)]
>>> C2.is_tangent(L2)
False
```



# Plane Geometry

## Basic Objects, properties, methods

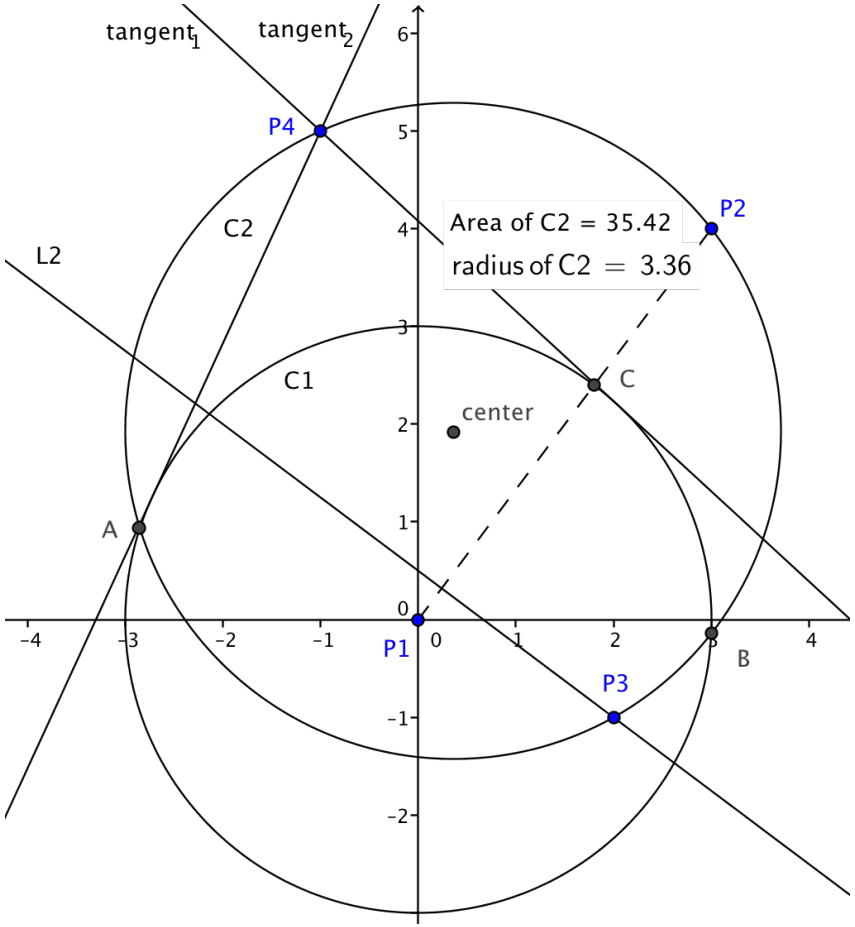
```
# Creation of some circles
C1 = Circle(P1, 3)
C2 = Circle(P2, P3, P4)
# Equations
>>> C2.equation()
(x - 5/14)**2 + ...
# Let's ask some questions
>>> C2.area
1105*pi/98
>>> C2.radius
sqrt(2210)/14
>>> C2.center
Point(5/14, 27/14)
>>> C2.circumference
sqrt(2210)*pi/7
>>> C1.tangent_lines(P4)
[Line ...,
 Line ...]
# Intersections with other objects
>>> C2.intersection(C1)
[Point(55/754 + ...
 Point(55/754 - ...)]
>>> C2.intersection(S1)
[Point(3, 4)]
>>> C2.is_tangent(L2)
False
```



# Plane Geometry

## Basic Objects, properties, methods

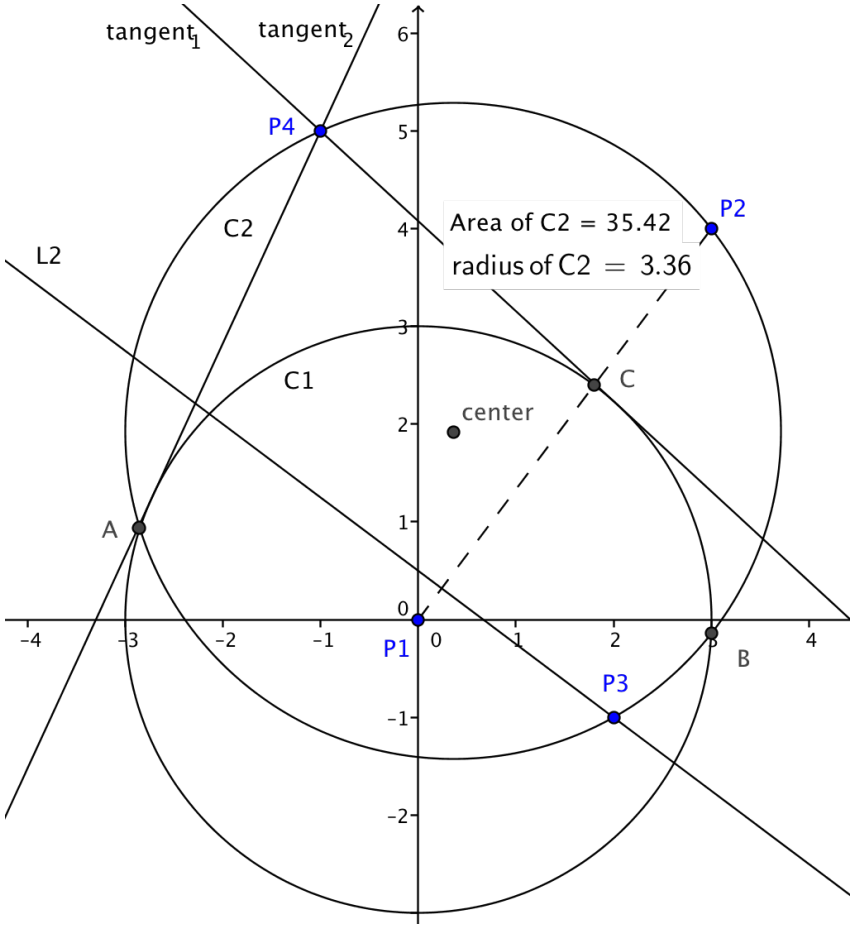
```
# Creation of some circles
C1 = Circle(P1, 3)
C2 = Circle(P2, P3, P4)
# Equations
>>> C2.equation()
(x - 5/14)**2 + ...
# Let's ask some questions
>>> C2.area
1105*pi/98
>>> C2.radius
sqrt(2210)/14
>>> C2.center
Point(5/14, 27/14)
>>> C2.circumference
sqrt(2210)*pi/7
>>> C1.tangent_lines(P4)
[Line ...,
 Line ...]
# Intersections with other objects
>>> C2.intersection(C1)
[Point(55/754 + ...
 Point(55/754 - ...)]
>>> C2.intersection(S1)
[Point(3, 4)]
>>> C2.is_tangent(L2)
False
```



# Plane Geometry

## Basic Objects, properties, methods

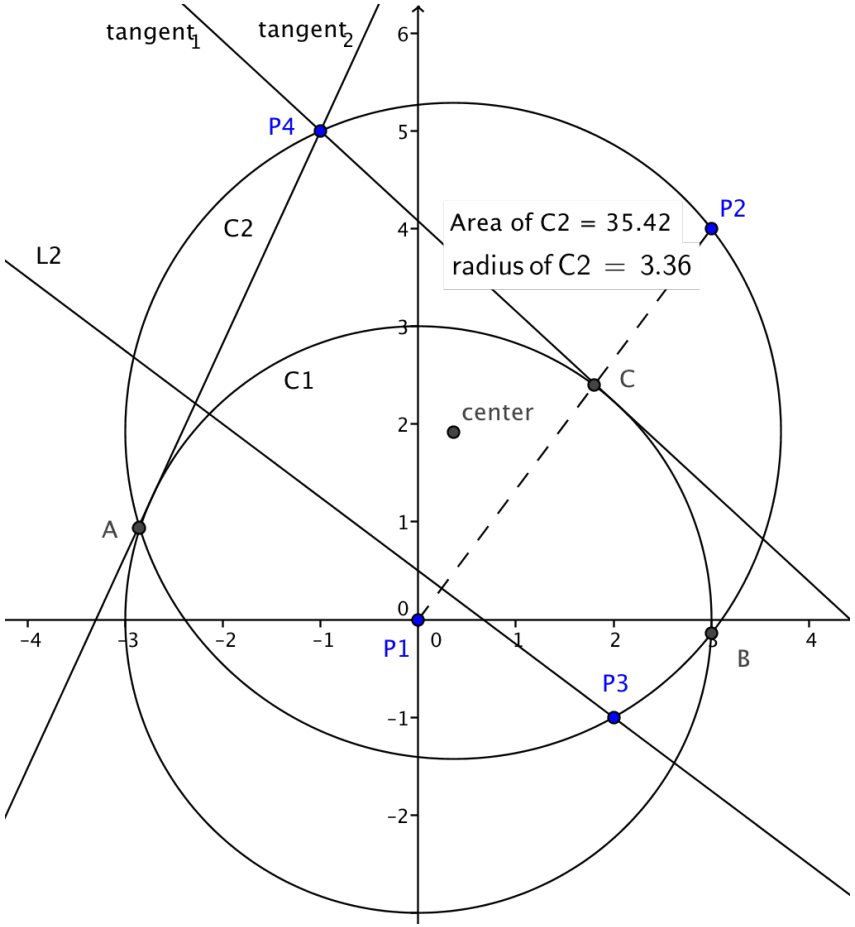
```
# Creation of some circles
C1 = Circle(P1, 3)
C2 = Circle(P2, P3, P4)
# Equations
>>> C2.equation()
(x - 5/14)**2 + ...
# Let's ask some questions
>>> C2.area
1105*pi/98
>>> C2.radius
sqrt(2210)/14
>>> C2.center
Point(5/14, 27/14)
>>> C2.circumference
sqrt(2210)*pi/7
>>> C1.tangent_lines(P4)
[Line ...,
 Line ...]
# Intersections with other objects
>>> C2.intersection(C1)
[Point(55/754 + ...
 Point(55/754 - ...)]
>>> C2.intersection(S1)
[Point(3, 4)]
>>> C2.is_tangent(L2)
False
```



# Plane Geometry

## Basic Objects, properties, methods

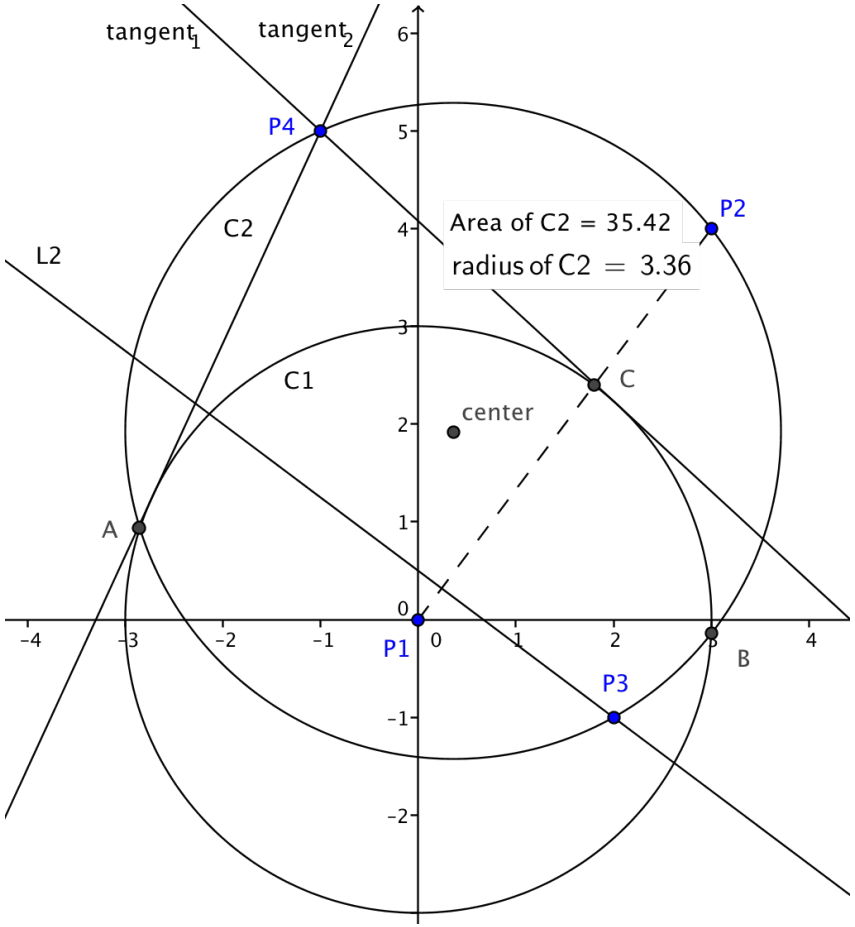
```
# Creation of some circles
C1 = Circle(P1, 3)
C2 = Circle(P2, P3, P4)
# Equations
>>> C2.equation()
(x - 5/14)**2 + ...
# Let's ask some questions
>>> C2.area
1105*pi/98
>>> C2.radius
sqrt(2210)/14
>>> C2.center
Point(5/14, 27/14)
>>> C2.circumference
sqrt(2210)*pi/7
>>> C1.tangent_lines(P4)
[Line ...,
 Line ...]
# Intersections with other objects
>>> C2.intersection(C1)
[Point(55/754 + ...
 Point(55/754 - ...)]
>>> C2.intersection(S1)
[Point(3, 4)]
>>> C2.is_tangent(L2)
False
```



# Plane Geometry

## Basic Objects, properties, methods

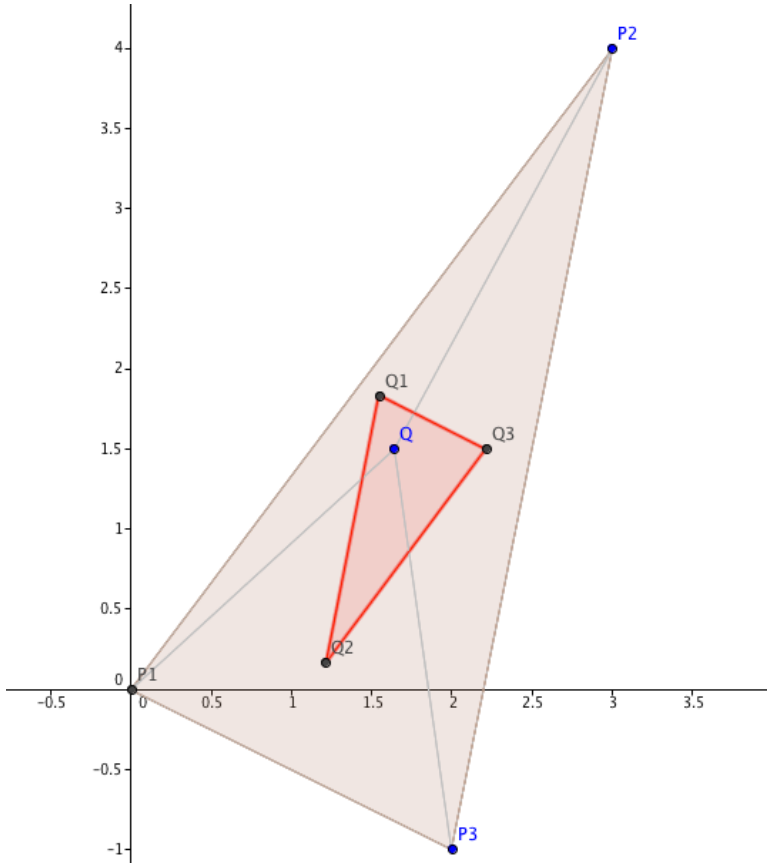
```
# Creation of some circles
C1 = Circle(P1, 3)
C2 = Circle(P2, P3, P4)
# Equations
>>> C2.equation()
(x - 5/14)**2 + ...
# Let's ask some questions
>>> C2.area
1105*pi/98
>>> C2.radius
sqrt(2210)/14
>>> C2.center
Point(5/14, 27/14)
>>> C2.circumference
sqrt(2210)*pi/7
>>> C1.tangent_lines(P4)
[Line ...,
 Line ...]
# Intersections with other objects
>>> C2.intersection(C1)
[Point(55/754 + ...
 Point(55/754 - ...)]
>>> C2.intersection(S1)
[Point(3, 4)]
>>> C2.is_tangent(L2)
False
```



# Plane Geometry

## Basic Objects, properties, methods

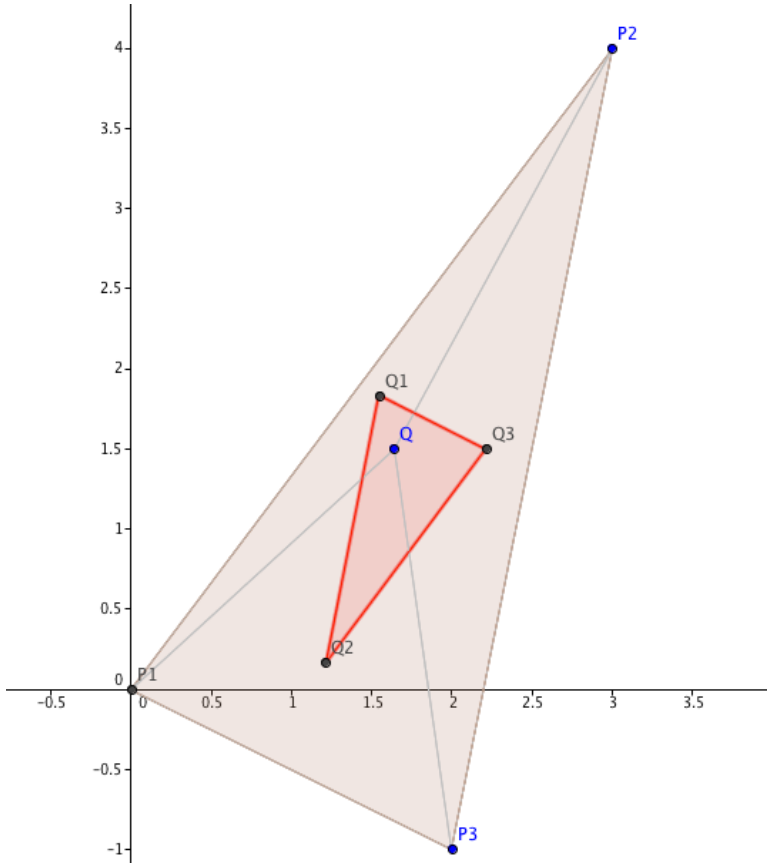
```
1  # Creation of a triangle
2  T = Triangle(P1, P2, P3)
3  # Pick ANY point in the plane
4  from sympy import var
5  var('x y')
6  Q = Point(x, y)
7  # Construct three smaller triangles
8  # with a common vertex at Q
9  T1 = Triangle(P1, P2, Q)
10 T2 = Triangle(P1, P3, Q)
11 T3 = Triangle(P2, P3, Q)
12 # Find the centroid of those triangles
13 Q1 = T1.centroid
14 Q2 = T2.centroid
15 Q3 = T3.centroid
16 # Construct a triangle through centroids
17 TQ = Triangle(Q1, Q2, Q3)
18 # Show coordinates of vertices
19 >> TQ.vertices
20 (Point(x/3 + 1, y/3 + 4/3),
21  Point(x/3 + 2/3, y/3 - 1/3),
22  Point(x/3 + 5/3, y/3 + 1))
23 # T and TQ look similar... are they?
24 >>> TQ.is_similar(T)
25 True
```



# Plane Geometry

## Basic Objects, properties, methods

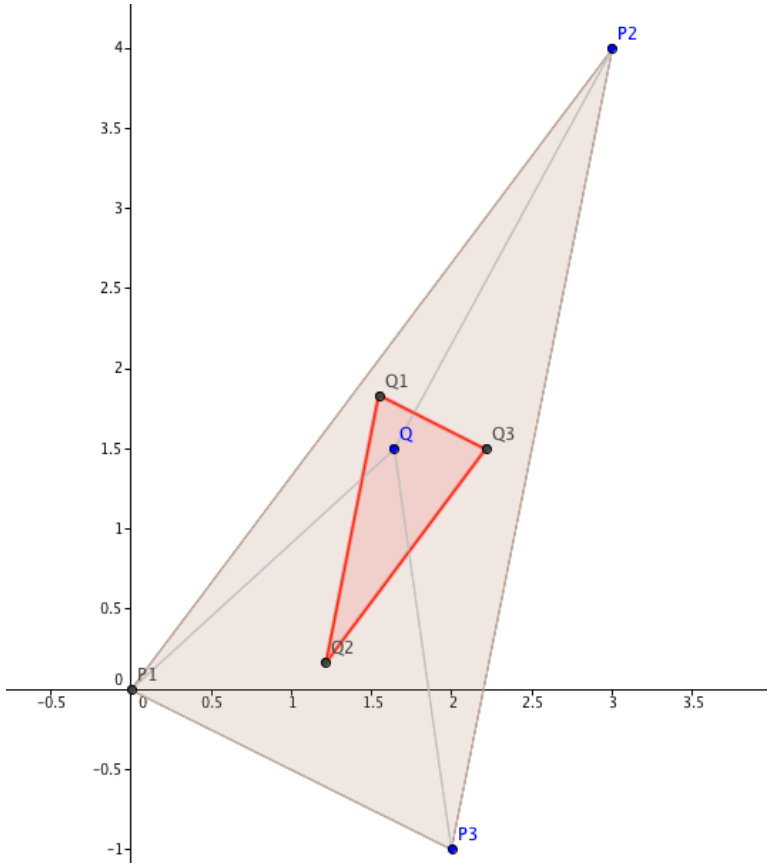
```
1  # Creation of a triangle
2  T = Triangle(P1, P2, P3)
3  # Pick ANY point in the plane
4  from sympy import var
5  var('x y')
6  Q = Point(x, y)
7  # Construct three smaller triangles
8  # with a common vertex at Q
9  T1 = Triangle(P1, P2, Q)
10 T2 = Triangle(P1, P3, Q)
11 T3 = Triangle(P2, P3, Q)
12 # Find the centroid of those triangles
13 Q1 = T1.centroid
14 Q2 = T2.centroid
15 Q3 = T3.centroid
16 # Construct a triangle through centroids
17 TQ = Triangle(Q1, Q2, Q3)
18 # Show coordinates of vertices
19 >> TQ.vertices
20 (Point(x/3 + 1, y/3 + 4/3),
21  Point(x/3 + 2/3, y/3 - 1/3),
22  Point(x/3 + 5/3, y/3 + 1))
23 # T and TQ look similar... are they?
24 >>> TQ.is_similar(T)
25 True
```



# Plane Geometry

## Basic Objects, properties, methods

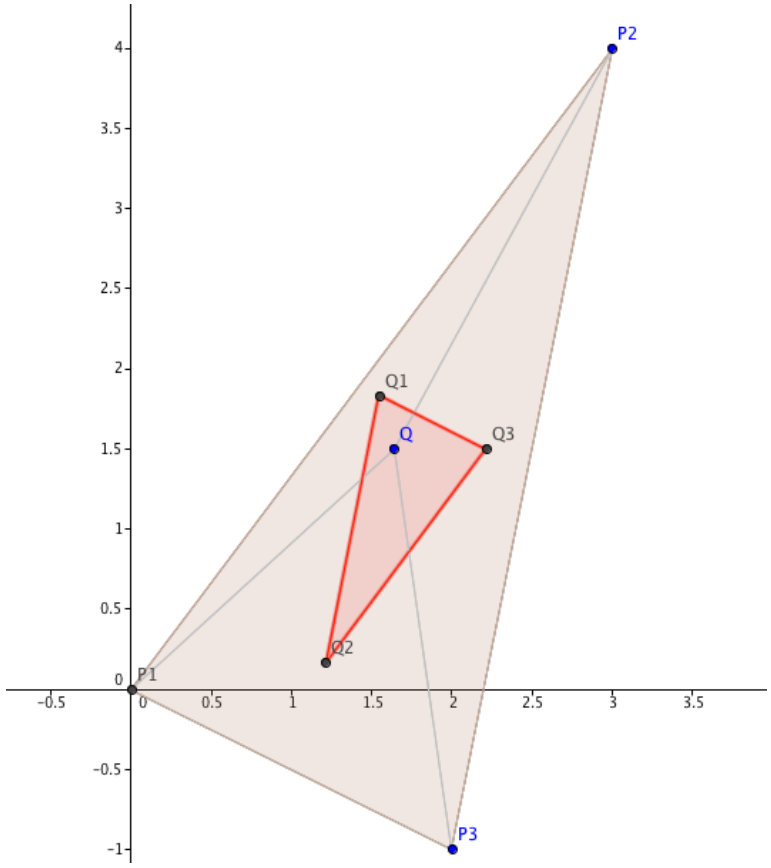
```
1  # Creation of a triangle
2  T = Triangle(P1, P2, P3)
3  # Pick ANY point in the plane
4  from sympy import var
5  var('x y')
6  Q = Point(x, y)
7  # Construct three smaller triangles
8  # with a common vertex at Q
9  T1 = Triangle(P1, P2, Q)
10 T2 = Triangle(P1, P3, Q)
11 T3 = Triangle(P2, P3, Q)
12 # Find the centroid of those triangles
13 Q1 = T1.centroid
14 Q2 = T2.centroid
15 Q3 = T3.centroid
16 # Construct a triangle through centroids
17 TQ = Triangle(Q1, Q2, Q3)
18 # Show coordinates of vertices
19 >> TQ.vertices
20 (Point(x/3 + 1, y/3 + 4/3),
21  Point(x/3 + 2/3, y/3 - 1/3),
22  Point(x/3 + 5/3, y/3 + 1))
23 # T and TQ look similar... are they?
24 >>> TQ.is_similar(T)
25 True
```



# Plane Geometry

## Basic Objects, properties, methods

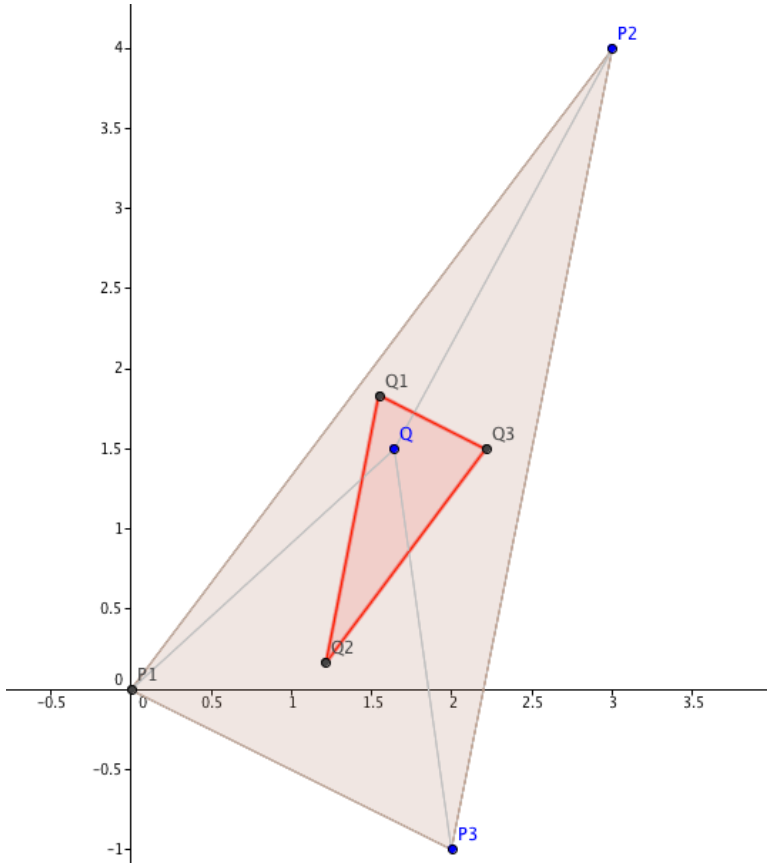
```
1  # Creation of a triangle
2  T = Triangle(P1, P2, P3)
3  # Pick ANY point in the plane
4  from sympy import var
5  var('x y')
6  Q = Point(x, y)
7  # Construct three smaller triangles
8  # with a common vertex at Q
9  T1 = Triangle(P1, P2, Q)
10 T2 = Triangle(P1, P3, Q)
11 T3 = Triangle(P2, P3, Q)
12 # Find the centroid of those triangles
13 Q1 = T1.centroid
14 Q2 = T2.centroid
15 Q3 = T3.centroid
16 # Construct a triangle through centroids
17 TQ = Triangle(Q1, Q2, Q3)
18 # Show coordinates of vertices
19 >> TQ.vertices
20 (Point(x/3 + 1, y/3 + 4/3),
21  Point(x/3 + 2/3, y/3 - 1/3),
22  Point(x/3 + 5/3, y/3 + 1))
23 # T and TQ look similar... are they?
24 >>> TQ.is_similar(T)
25 True
```



# Plane Geometry

## Basic Objects, properties, methods

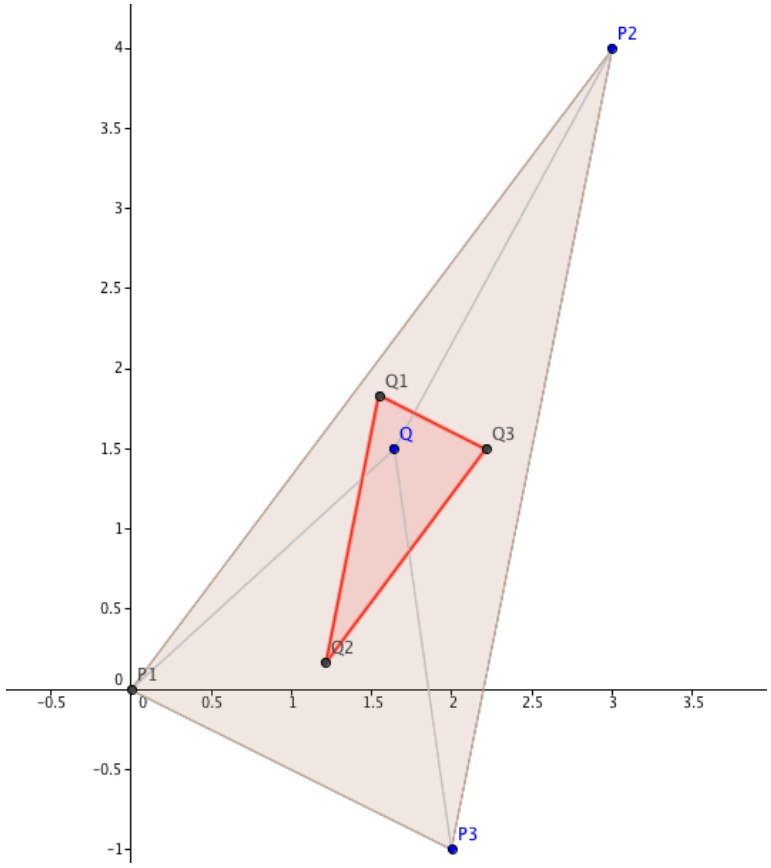
```
1  # Creation of a triangle
2  T = Triangle(P1, P2, P3)
3  # Pick ANY point in the plane
4  from sympy import var
5  var('x y')
6  Q = Point(x, y)
7  # Construct three smaller triangles
8  # with a common vertex at Q
9  T1 = Triangle(P1, P2, Q)
10 T2 = Triangle(P1, P3, Q)
11 T3 = Triangle(P2, P3, Q)
12 # Find the centroid of those triangles
13 Q1 = T1.centroid
14 Q2 = T2.centroid
15 Q3 = T3.centroid
16 # Construct a triangle through centroids
17 TQ = Triangle(Q1, Q2, Q3)
18 # Show coordinates of vertices
19 >> TQ.vertices
20 (Point(x/3 + 1, y/3 + 4/3),
21  Point(x/3 + 2/3, y/3 - 1/3),
22  Point(x/3 + 5/3, y/3 + 1))
23 # T and TQ look similar... are they?
24 >>> TQ.is_similar(T)
25 True
```



# Plane Geometry

## Basic Objects, properties, methods

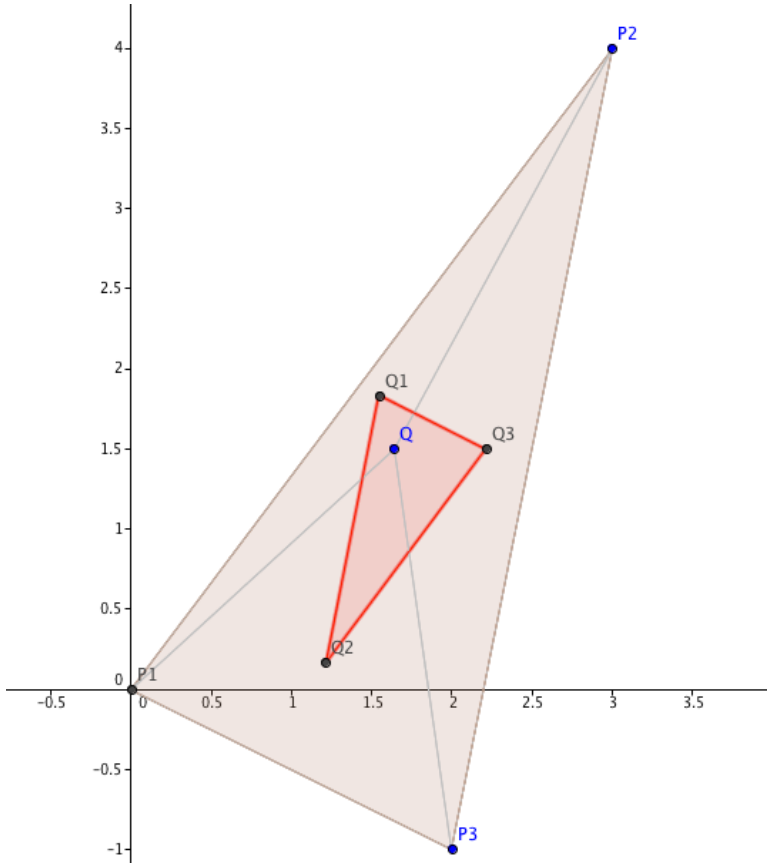
```
1  # Creation of a triangle
2  T = Triangle(P1, P2, P3)
3  # Pick ANY point in the plane
4  from sympy import var
5  var('x y')
6  Q = Point(x, y)
7  # Construct three smaller triangles
8  # with a common vertex at Q
9  T1 = Triangle(P1, P2, Q)
10 T2 = Triangle(P1, P3, Q)
11 T3 = Triangle(P2, P3, Q)
12 # Find the centroid of those triangles
13 Q1 = T1.centroid
14 Q2 = T2.centroid
15 Q3 = T3.centroid
16 # Construct a triangle through centroids
17 TQ = Triangle(Q1, Q2, Q3)
18 # Show coordinates of vertices
19 >> TQ.vertices
20 (Point(x/3 + 1, y/3 + 4/3),
21  Point(x/3 + 2/3, y/3 - 1/3),
22  Point(x/3 + 5/3, y/3 + 1))
23 # T and TQ look similar... are they?
24 >>> TQ.is_similar(T)
25 True
```



# Plane Geometry

## Basic Objects, properties, methods

```
1  # Creation of a triangle
2  T = Triangle(P1, P2, P3)
3  # Pick ANY point in the plane
4  from sympy import var
5  var('x y')
6  Q = Point(x, y)
7  # Construct three smaller triangles
8  # with a common vertex at Q
9  T1 = Triangle(P1, P2, Q)
10 T2 = Triangle(P1, P3, Q)
11 T3 = Triangle(P2, P3, Q)
12 # Find the centroid of those triangles
13 Q1 = T1.centroid
14 Q2 = T2.centroid
15 Q3 = T3.centroid
16 # Construct a triangle through centroids
17 TQ = Triangle(Q1, Q2, Q3)
18 # Show coordinates of vertices
19 >> TQ.vertices
20 (Point(x/3 + 1, y/3 + 4/3),
21  Point(x/3 + 2/3, y/3 - 1/3),
22  Point(x/3 + 5/3, y/3 + 1))
23 # T and TQ look similar... are they?
24 >>> TQ.is_similar(T)
25 True
```



# Combinatorial Computational Geometry

## Static Problems—Convex Hulls

```
# Preamble
import numpy as np
from scipy.spatial import ConvexHull
import matplotlib.pyplot as plt
%matplotlib inline
# Read poly file: vertices | segments | holes
lake_superior = read_poly("superior.poly")
vertices_ls = lake_superior['vertices']
# Compute Convex Hull of polygon
>>> %time hull = ConvexHull(vertices_ls)
CPU time: user 936 us, sys: 490 us, total: 1.43 ms
Wall time: 871 us
# Generate diagram
plt.plot(vertices_ls[:,0], vertices_ls[:,1], 'b.')
for simplex in hull.simplices:
    plt.plot(vertices_ls[simplex, 0], vertices_ls[simplex, 1], 'r-')
plt.show()
```

# Combinatorial Computational Geometry

## Static Problems—Convex Hulls

```
# Preamble
import numpy as np
from scipy.spatial import ConvexHull
import matplotlib.pyplot as plt
%matplotlib inline
# Read poly file: vertices | segments | holes
lake_superior = read_poly("superior.poly")
vertices_ls = lake_superior['vertices']
# Compute Convex Hull of polygon
>>> %time hull = ConvexHull(vertices_ls)
CPU time: user 936 us, sys: 490 us, total: 1.43 ms
Wall time: 871 us
# Generate diagram
plt.plot(vertices_ls[:,0], vertices_ls[:,1], 'b.')
for simplex in hull.simplices:
    plt.plot(vertices_ls[simplex, 0], vertices_ls[simplex, 1], 'r-')
plt.show()
```

# Combinatorial Computational Geometry

## Static Problems—Convex Hulls

```
# Preamble
import numpy as np
from scipy.spatial import ConvexHull
import matplotlib.pyplot as plt
%matplotlib inline
# Read poly file: vertices | segments | holes
lake_superior = read_poly("superior.poly")
vertices_ls = lake_superior['vertices']
# Compute Convex Hull of polygon
>>> %time hull = ConvexHull(vertices_ls)
CPU time: user 936 us, sys: 490 us, total: 1.43 ms
Wall time: 871 us
# Generate diagram
plt.plot(vertices_ls[:,0], vertices_ls[:,1], 'b.')
for simplex in hull.simplices:
    plt.plot(vertices_ls[simplex, 0], vertices_ls[simplex, 1], 'r-')
plt.show()
```

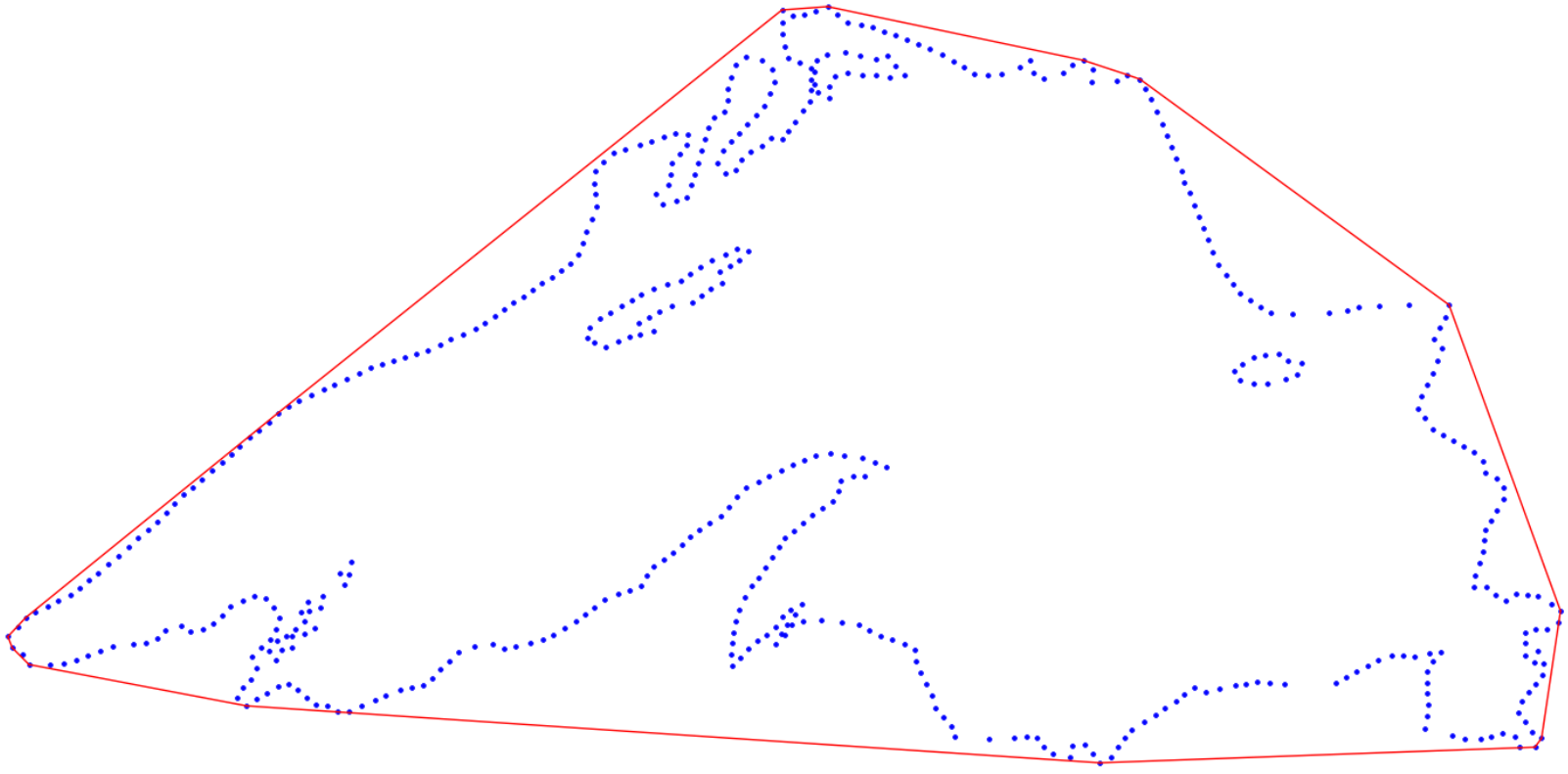
# Combinatorial Computational Geometry

## Static Problems—Convex Hulls

```
# Preamble
import numpy as np
from scipy.spatial import ConvexHull
import matplotlib.pyplot as plt
%matplotlib inline
# Read poly file: vertices | segments | holes
lake_superior = read_poly("superior.poly")
vertices_ls = lake_superior['vertices']
# Compute Convex Hull of polygon
>>> %time hull = ConvexHull(vertices_ls)
CPU time: user 936 us, sys: 490 us, total: 1.43 ms
Wall time: 871 us
# Generate diagram
plt.plot(vertices_ls[:,0], vertices_ls[:,1], 'b.')
for simplex in hull.simplices:
    plt.plot(vertices_ls[simplex, 0], vertices_ls[simplex, 1], 'r-')
plt.show()
```

# Combinatorial Computational Geometry

## Static Problems—Convex Hulls



# Combinatorial Computational Geometry

## Static Problems—Voronoi Diagrams

```
# Preamble
from scipy.spatial import Voronoi, voronoi_plot_2d
# Compute Voronoi Diagram of polygon
vor = Voronoi(vertices_ls)
# Generate diagram (close-up)
ax = plt.subplot(111,aspect='equal')
plt.xlim( 0.45,  0.50)
plt.ylim(-0.40, -0.35)
voronoi_plot_2d(vor, ax=ax)
plt.show()
```

# Combinatorial Computational Geometry

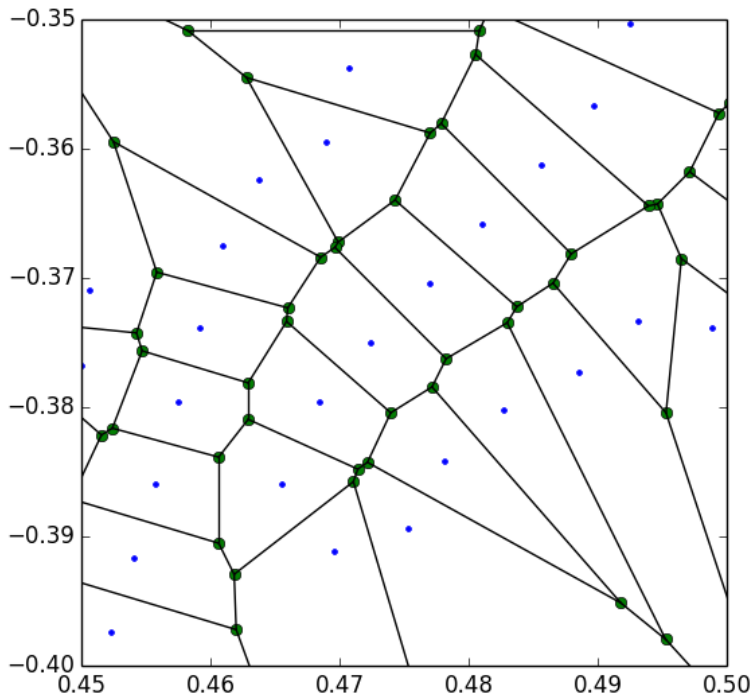
## Static Problems—Voronoi Diagrams

```
# Preamble
from scipy.spatial import Voronoi, voronoi_plot_2d
# Compute Voronoi Diagram of polygon
vor = Voronoi(vertices_ls)
# Generate diagram (close-up)
ax = plt.subplot(111,aspect='equal')
plt.xlim( 0.45,  0.50)
plt.ylim(-0.40, -0.35)
voronoi_plot_2d(vor, ax=ax)
plt.show()
```

# Combinatorial Computational Geometry

## Static Problems—Voronoi Diagrams

```
1  # Preamble
2  from scipy.spatial import Voronoi, voronoi_plot_2d
3  # Compute Voronoi Diagram of polygon
4  vor = Voronoi(vertices_ls)
5  # Generate diagram (close-up)
6  ax = plt.subplot(111,aspect='equal')
7  plt.xlim( 0.45,  0.50)
8  plt.ylim(-0.40, -0.35)
9  voronoi_plot_2d(vor, ax=ax)
10 plt.show()
```



# Combinatorial Computational Geometry

## Static Problems—Triangulations

```
# Preamble --- Delaunay Triangulation
from scipy.spatial import Delaunay, delaunay_plot_2d
# Compute Delaunay triangulation of VERTICES
tri = Delaunay(vertices_ls)
# Generate diagram
ax = plt.subplot(111, aspect='equal')
delaunay_plot_2d(tri, ax=ax)
plt.show()
```

# Combinatorial Computational Geometry

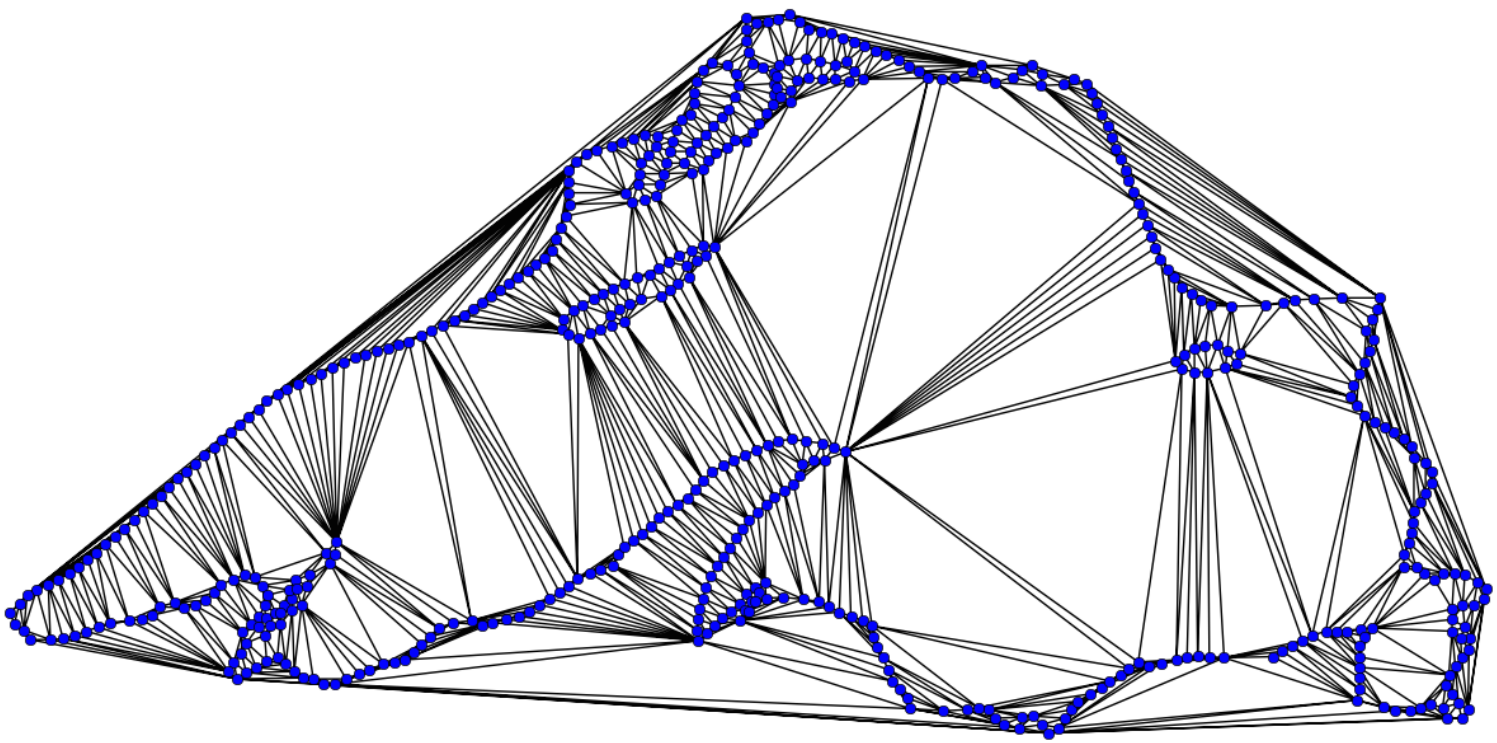
## Static Problems—Triangulations

```
# Preamble --- Delaunay Triangulation
from scipy.spatial import Delaunay, delaunay_plot_2d
# Compute Delaunay triangulation of VERTICES
tri = Delaunay(vertices_ls)
# Generate diagram
ax = plt.subplot(111, aspect='equal')
delaunay_plot_2d(tri, ax=ax)
plt.show()
```

# Combinatorial Computational Geometry

## Static Problems—Triangulations

```
# Preamble --- Delaunay Triangulation
from scipy.spatial import Delaunay, delaunay_plot_2d
# Compute Delaunay triangulation of VERTICES
tri = Delaunay(vertices_ls)
# Generate diagram
ax = plt.subplot(111, aspect='equal')
delaunay_plot_2d(tri, ax=ax)
plt.show()
```



# Combinatorial Computational Geometry

## Static Problems—Triangulations

```
# Preamble  --- Constrained Delaunay Triangulation
from triangle import triangulate, plot as tplot
# Compute Constrained Delaunay triangulation of POLYGON
cndt = triangulate(lake_superior, 'p')
# Generate diagram
ax = plt.subplot(111, aspect='equal')
tplot.plot(ax, **cndt)
plt.show()
```

# Combinatorial Computational Geometry

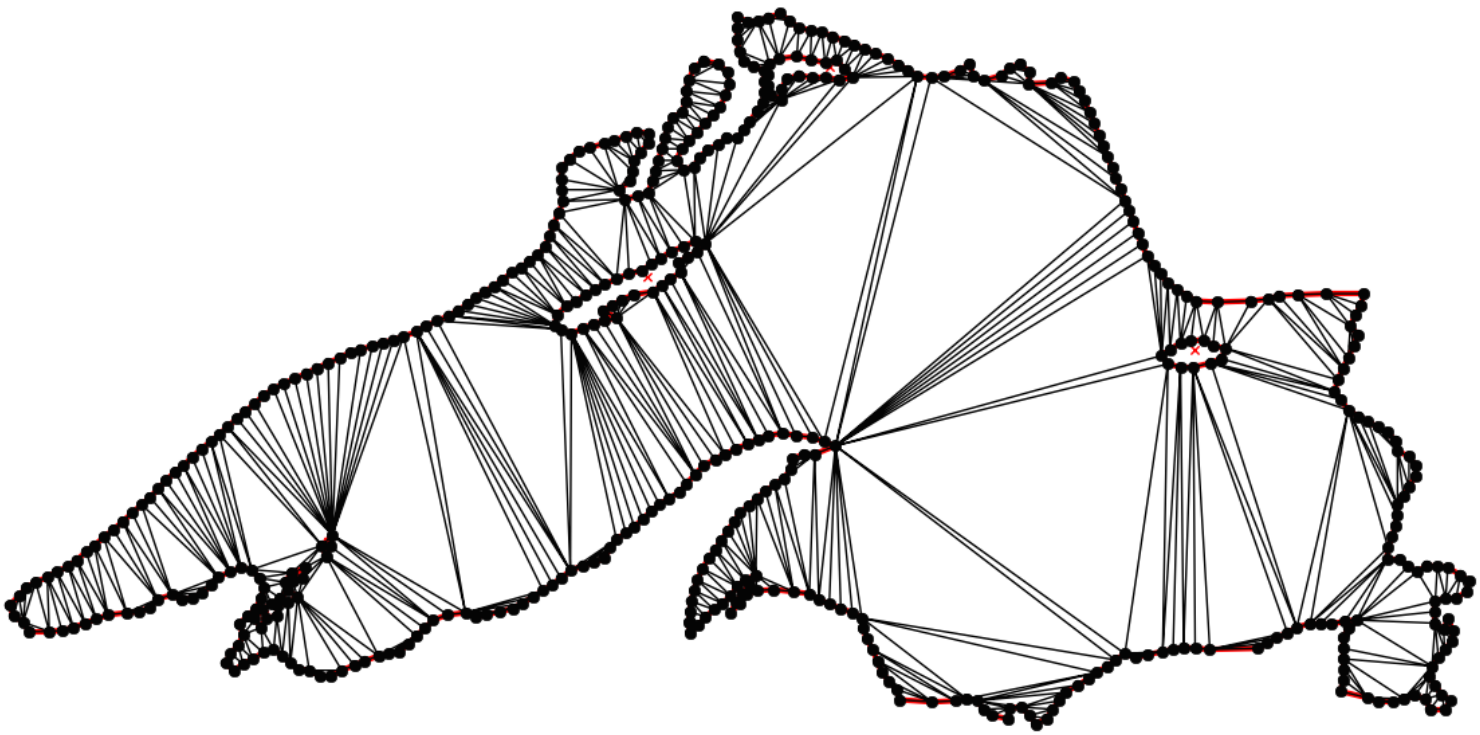
## Static Problems—Triangulations

```
# Preamble  --- Constrained Delaunay Triangulation
from triangle import triangulate, plot as tplot
# Compute Constrained Delaunay triangulation of POLYGON
cndt = triangulate(lake_superior, 'p')
# Generate diagram
ax = plt.subplot(111, aspect='equal')
tplot.plot(ax, **cndt)
plt.show()
```

# Combinatorial Computational Geometry

## Static Problems—Triangulations

```
# Preamble  --- Constrained Delaunay Triangulation
from triangle import triangulate, plot as tplot
# Compute Constrained Delaunay triangulation of POLYGON
cndt = triangulate(lake_superior, 'p')
# Generate diagram
ax = plt.subplot(111, aspect='equal')
tplot.plot(ax, **cndt)
plt.show()
```



# Combinatorial Computational Geometry

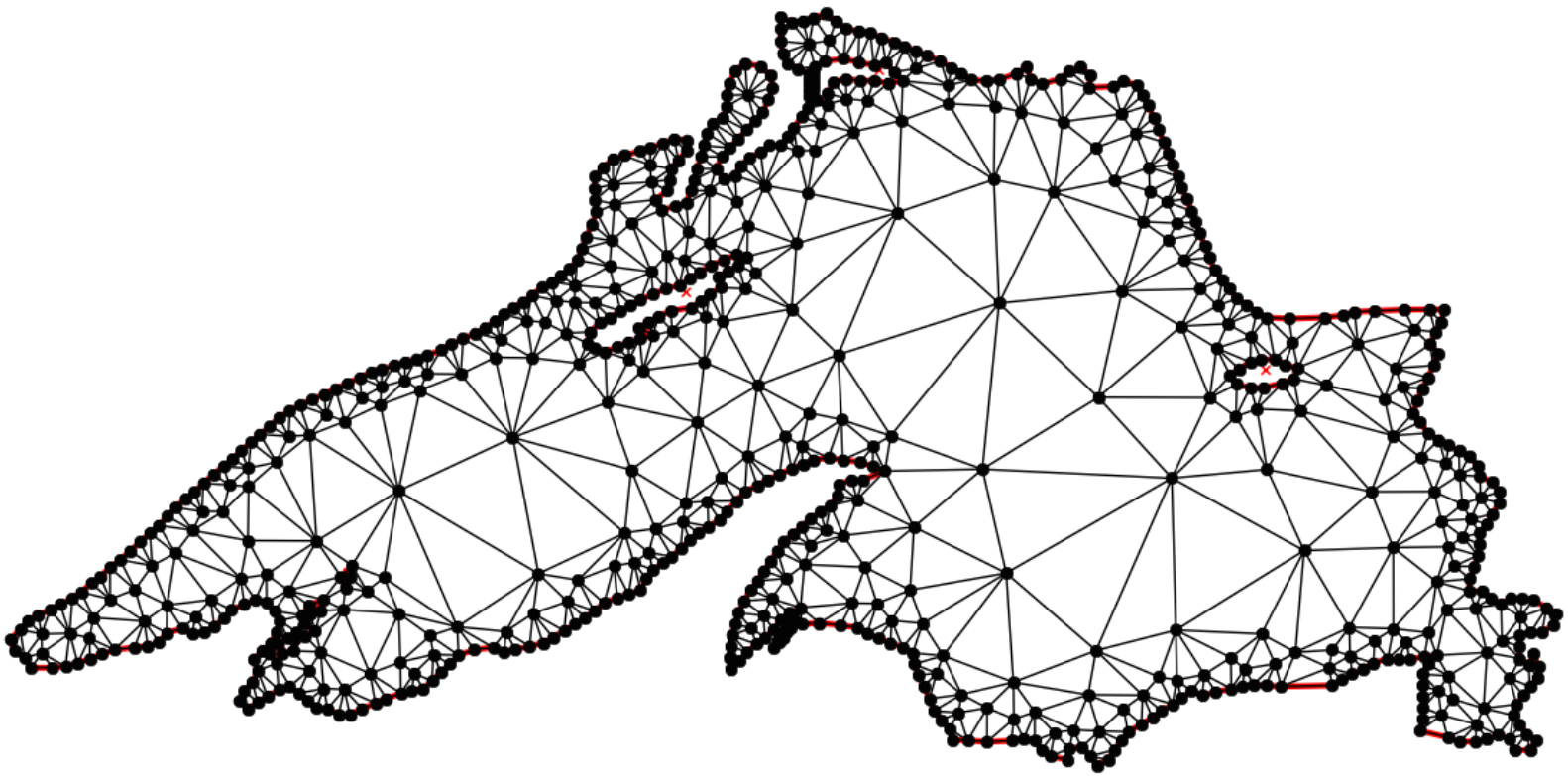
## Static Problems—Triangulations

```
# Constrained Conforming Delaunay Triangulation
# Impose minimum angles of 20 degrees (whenever possible)
cncfq20dt = triangulate(lake_superior, 'pq20D')
# Generate diagram
ax = plt.subplot(111, aspect='equal')
tplot.plot(ax, **cncfq20dt)
plt.show()
```

# Combinatorial Computational Geometry

## Static Problems—Triangulations

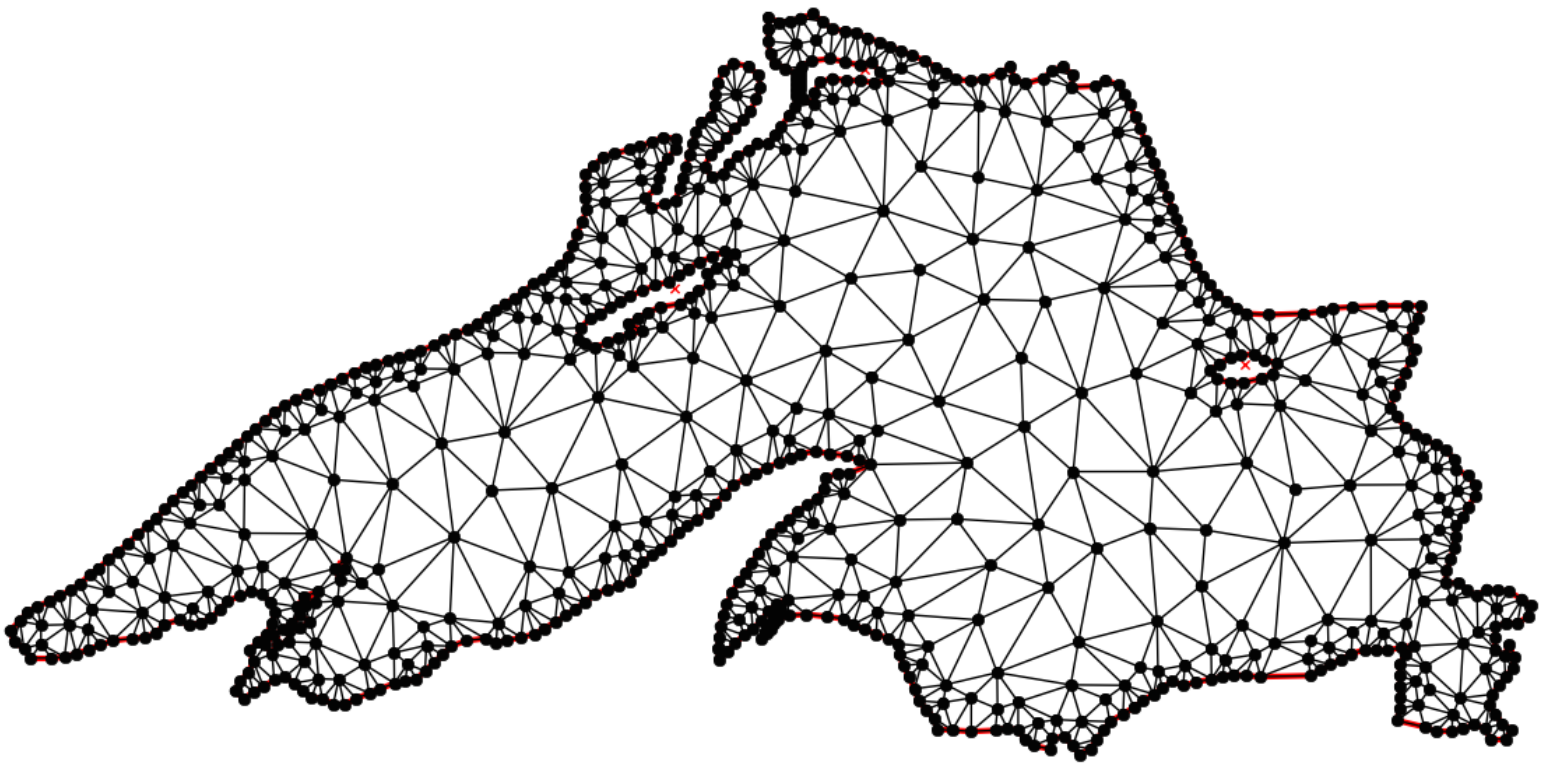
```
1 # Constrained Conforming Delaunay Triangulation
2 # Impose minimum angles of 20 degrees (whenever possible)
3 cncfq20dt = triangulate(lake_superior, 'pq20D')
4 # Generate diagram
5 ax = plt.subplot(111, aspect='equal')
6 tplot.plot(ax, **cncfq20dt)
7 plt.show()
```



# Combinatorial Computational Geometry

## Static Problems—Triangulations

```
1 # Constrained Conforming Delaunay Triangulation
2 # Impose minimum angles of 20 degrees (whenever possible)
3 # Impose also maximum area
4 cncfq20adt = triangulate(lake_superior, 'pq20a.001D')
5 # Generate diagram
6 ax = plt.subplot(111, aspect='equal')
7 tplot.plot(ax, **cncfq20adt)
8 plt.show()
```



# Combinatorial Computational Geometry

## Static Problems—Shortest Path

```
1  # Preamble
2  from scipy.spatial import minkowski_distance
3  from scipy.sparse import lil_matrix
4  from scipy.sparse.csgraph import shortest_path
5
6  # Collect segments from triangles and their corresponding vertices
7  X = cncfq20adt['triangles'][:, 0]
8  Xvert = [cncfq20adt['vertices'][x] for x in X]
9  Y = cncfq20adt['triangles'][:, 1]
10 Yvert = [cncfq20adt['vertices'][y] for y in Y]
11 Z = cncfq20adt['triangles'][:, 2]
12 Zvert = [cncfq20adt['vertices'][z] for z in Z]
13
14 # Compute the lengths of these segments
15 lengthsXY = minkowski_distance(Xvert, Yvert)
16 lengthsXZ = minkowski_distance(Xvert, Zvert)
17 lengthsYZ = minkowski_distance(Yvert, Zvert)
18
19 # Create adjacency matrix
20 nvert = len(cncfq20adt['vertices'])
21 G = lil_matrix((nvert, nvert))
22 for k in range(len(X)):
23     G[X[k], Y[k]] = G[Y[k], X[k]] = lengthsXY[k]
24     G[X[k], Z[k]] = G[Z[k], X[k]] = lengthsXZ[k]
25     G[Y[k], Z[k]] = G[Z[k], Y[k]] = lengthsYZ[k]
```

# Combinatorial Computational Geometry

## Static Problems—Shortest Path

```
# Preamble
from scipy.spatial import minkowski_distance
from scipy.sparse import lil_matrix
from scipy.sparse.csgraph import shortest_path

# Collect segments from triangles and their corresponding vertices
X = cncfq20adt['triangles'][:, 0]
Xvert = [cncfq20adt['vertices'][x] for x in X]
Y = cncfq20adt['triangles'][:, 1]
Yvert = [cncfq20adt['vertices'][y] for y in Y]
Z = cncfq20adt['triangles'][:, 2]
Zvert = [cncfq20adt['vertices'][z] for z in Z]

# Compute the lengths of these segments
lengthsXY = minkowski_distance(Xvert, Yvert)
lengthsXZ = minkowski_distance(Xvert, Zvert)
lengthsYZ = minkowski_distance(Yvert, Zvert)

# Create adjacency matrix
nvert = len(cncfq20adt['vertices'])
G = lil_matrix((nvert, nvert))
for k in range(len(X)):
    G[X[k], Y[k]] = G[Y[k], X[k]] = lengthsXY[k]
    G[X[k], Z[k]] = G[Z[k], X[k]] = lengthsXZ[k]
    G[Y[k], Z[k]] = G[Z[k], Y[k]] = lengthsYZ[k]
```

# Combinatorial Computational Geometry

## Static Problems—Shortest Path

```
1  # Preamble
2  from scipy.spatial import minkowski_distance
3  from scipy.sparse import lil_matrix
4  from scipy.sparse.csgraph import shortest_path
5
6  # Collect segments from triangles and their corresponding vertices
7  X = cncfq20adt['triangles'][:, 0]
8  Xvert = [cncfq20adt['vertices'][x] for x in X]
9  Y = cncfq20adt['triangles'][:, 1]
10 Yvert = [cncfq20adt['vertices'][y] for y in Y]
11 Z = cncfq20adt['triangles'][:, 2]
12 Zvert = [cncfq20adt['vertices'][z] for z in Z]
13
14 # Compute the lengths of these segments
15 lengthsXY = minkowski_distance(Xvert, Yvert)
16 lengthsXZ = minkowski_distance(Xvert, Zvert)
17 lengthsYZ = minkowski_distance(Yvert, Zvert)
18
19 # Create adjacency matrix
20 nvert = len(cncfq20adt['vertices'])
21 G = lil_matrix((nvert, nvert))
22 for k in range(len(X)):
23      G[X[k], Y[k]] = G[Y[k], X[k]] = lengthsXY[k]
24      G[X[k], Z[k]] = G[Z[k], X[k]] = lengthsXZ[k]
25      G[Y[k], Z[k]] = G[Z[k], Y[k]] = lengthsYZ[k]
```

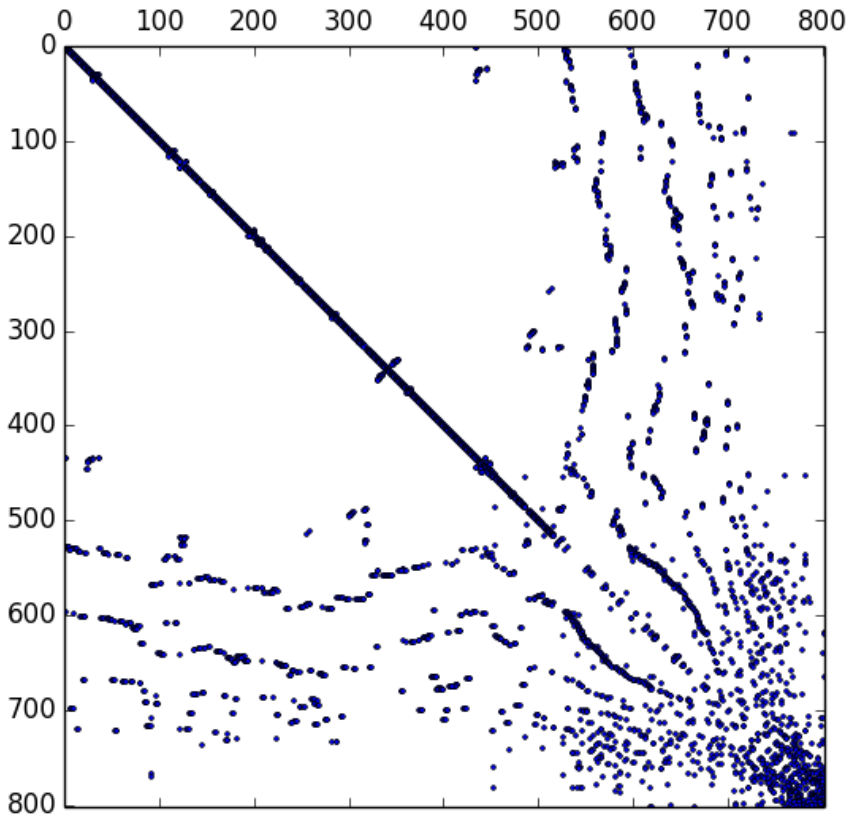
# Combinatorial Computational Geometry

## Static Problems—Shortest Path

```
1  # Preamble
2  from scipy.spatial import minkowski_distance
3  from scipy.sparse import lil_matrix
4  from scipy.sparse.csgraph import shortest_path
5
6  # Collect segments from triangles and their corresponding vertices
7  X = cncfq20adt['triangles'][:, 0]
8  Xvert = [cncfq20adt['vertices'][x] for x in X]
9  Y = cncfq20adt['triangles'][:, 1]
10 Yvert = [cncfq20adt['vertices'][y] for y in Y]
11 Z = cncfq20adt['triangles'][:, 2]
12 Zvert = [cncfq20adt['vertices'][z] for z in Z]
13
14 # Compute the lengths of these segments
15 lengthsXY = minkowski_distance(Xvert, Yvert)
16 lengthsXZ = minkowski_distance(Xvert, Zvert)
17 lengthsYZ = minkowski_distance(Yvert, Zvert)
18
19 # Create adjacency matrix
20 nvert = len(cncfq20adt['vertices'])
21 G = lil_matrix((nvert, nvert))
22 for k in range(len(X)):
23      G[X[k], Y[k]] = G[Y[k], X[k]] = lengthsXY[k]
24      G[X[k], Z[k]] = G[Z[k], X[k]] = lengthsXZ[k]
25      G[Y[k], Z[k]] = G[Z[k], Y[k]] = lengthsYZ[k]
```

# Combinatorial Computational Geometry

## Static Problems—Shortest Path




Visualization of the adjacency matrix $G$

# Combinatorial Computational Geometry

## Static Problems—Shortest Path

```
26 # Initialize the *shortest path* algorithm of Dijkstra
27 dist_matrix, pred = shortest_path(G, return_predecessors=True,
28                                   directed=True, unweighted=False)
29
30 # Compute the path from source vertex (index 370)
31 # to target vertex (index 197)
32 index = 370
33 path = [370]
34
35 while index != 197:
36     index = pred[197, index]
37     path.append(index)
38
39 Xs = [cncfq20adt['vertices'][x][0] for x in path]
40 Ys = [cncfq20adt['vertices'][x][1] for x in path]
41
42 # Generate diagram
43 ax = plt.subplot(111, aspect='equal')
44 tplot.plot(ax, ** cncfq20adt)
45 ax.plot(Xs, Ys, '-', linewidth=5, color='blue')
46 plt.show()
```

# Combinatorial Computational Geometry

## Static Problems—Shortest Path

```
26 # Initialize the *shortest path* algorithm of Dijkstra
27 dist_matrix, pred = shortest_path(G, return_predecessors=True,
28                                   directed=True, unweighted=False)
29
30 # Compute the path from source vertex (index 370)
31 # to target vertex (index 197)
32 index = 370
33 path = [370]
34
35 while index != 197:
36     index = pred[197, index]
37     path.append(index)
38
39 Xs = [cncfq20adt['vertices'][x][0] for x in path]
40 Ys = [cncfq20adt['vertices'][x][1] for x in path]
41
42 # Generate diagram
43 ax = plt.subplot(111, aspect='equal')
44 tplot.plot(ax, ** cncfq20adt)
45 ax.plot(Xs, Ys, '-', linewidth=5, color='blue')
46 plt.show()
```

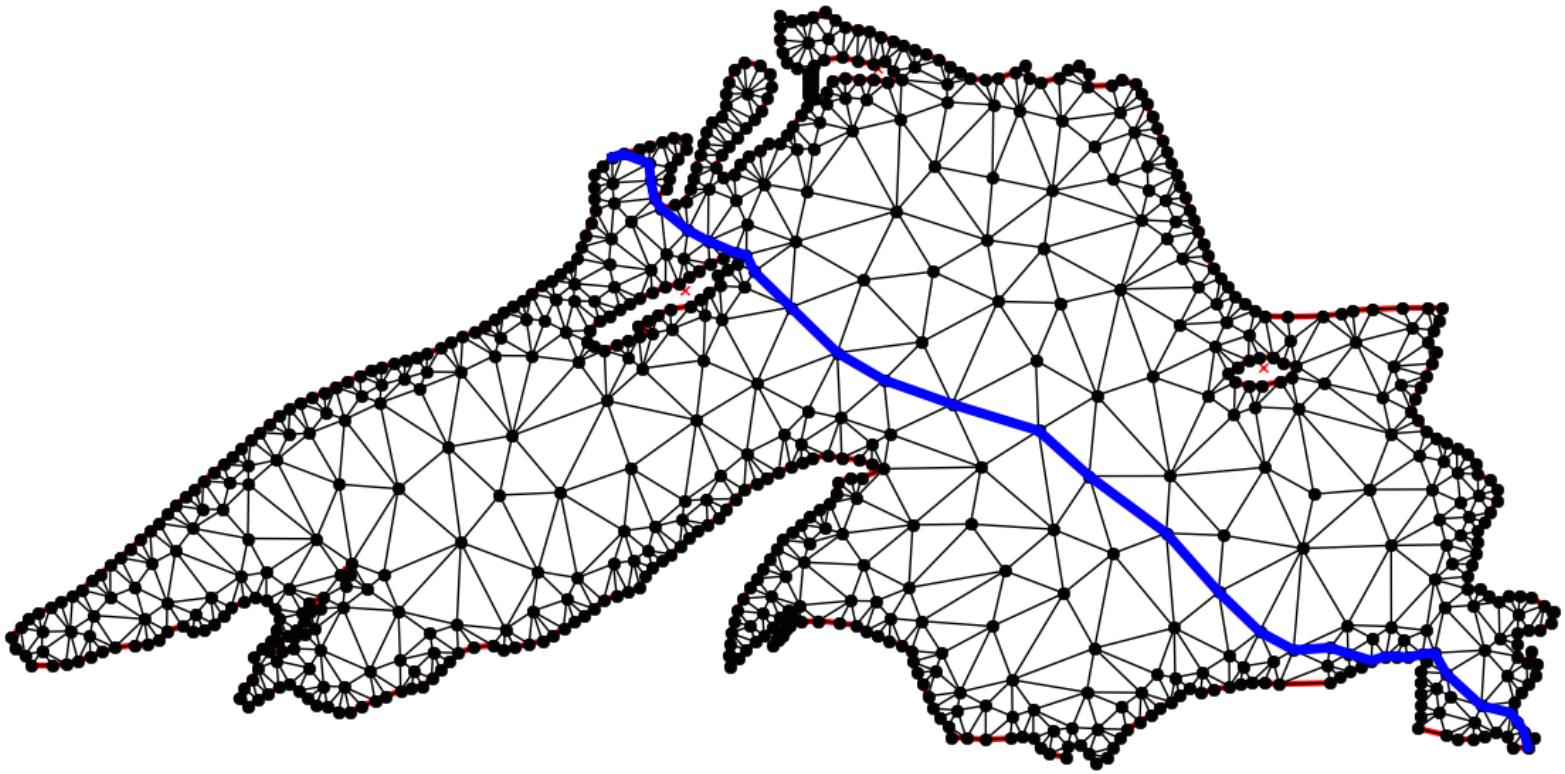
# Combinatorial Computational Geometry

## Static Problems—Shortest Path

```
26 # Initialize the *shortest path* algorithm of Dijkstra
27 dist_matrix, pred = shortest_path(G, return_predecessors=True,
28                                   directed=True, unweighted=False)
29
30 # Compute the path from source vertex (index 370)
31 # to target vertex (index 197)
32 index = 370
33 path = [370]
34
35 while index != 197:
36     index = pred[197, index]
37     path.append(index)
38
39 Xs = [cncfq20adt['vertices'][x][0] for x in path]
40 Ys = [cncfq20adt['vertices'][x][1] for x in path]
41
42 # Generate diagram
43 ax = plt.subplot(111, aspect='equal')
44 tplot.plot(ax, ** cncfq20adt)
45 ax.plot(Xs, Ys, '-', linewidth=5, color='blue')
46 plt.show()
```

# Combinatorial Computational Geometry

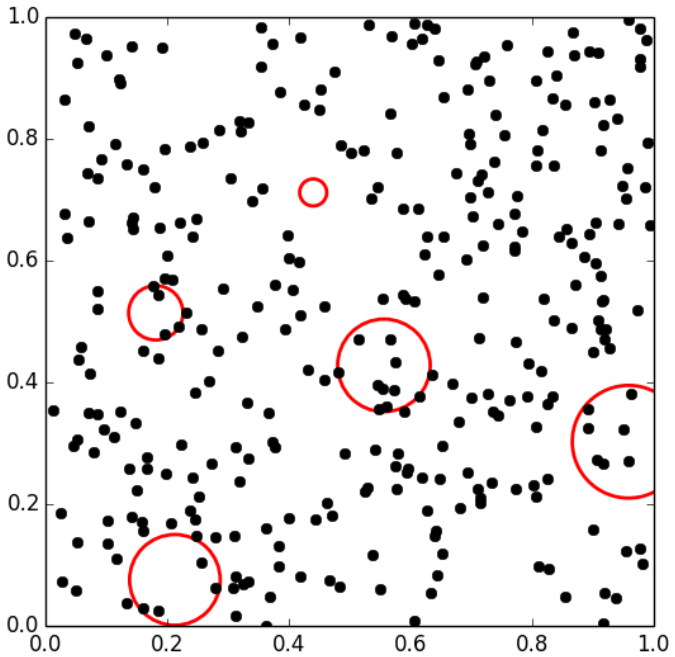
## Static Problems—Shortest Path



# Combinatorial Computational Geometry

## Geometric Query Problems—Range Searching

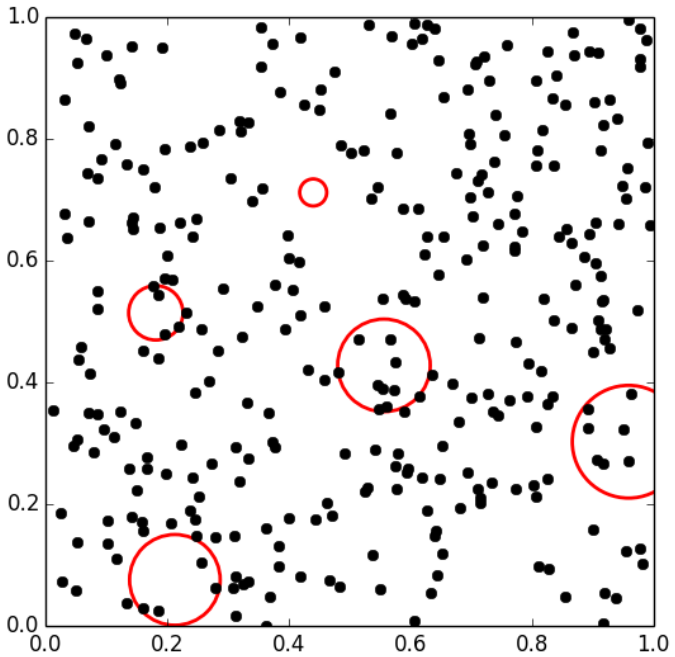
```
1  # Preamble
2  from scipy.spatial import cKDTree
3
4  # input: 320 random points
5  # query: 5 random circles
6  points = np.random.rand(320, 2)
7  range_centers = np.random.rand(5, 2)
8  range_radii = 0.1 * np.random.rand(5)
9
10 # To perform fast searches,
11 # store points in a k-d tree
12 tree = cKDTree(points)
13
14 # For each circle in the query,
15 # look for points inside
16 result = set()
17
18 for k in range(5):
19     center = range_centers[k]
20     radius = range_radii[k]
21     partial_query = tree.query_ball_point(center, radius)
22     result = result.union(set(partial_query))
23
24 print result
```



# Combinatorial Computational Geometry

## Geometric Query Problems—Range Searching

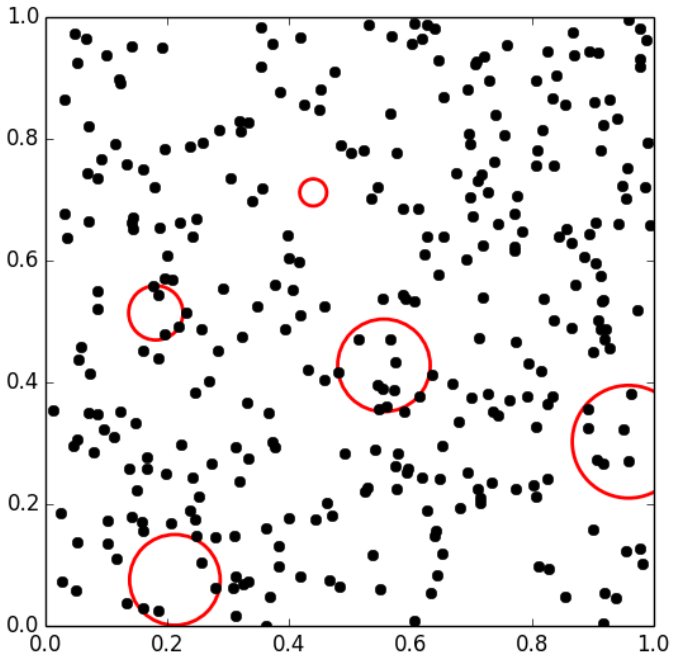
```
1  # Preamble
2  from scipy.spatial import cKDTree
3
4  # input: 320 random points
5  # query: 5 random circles
6  points = np.random.rand(320, 2)
7  range_centers = np.random.rand(5, 2)
8  range_radii = 0.1 * np.random.rand(5)
9
10 # To perform fast searches,
11 # store points in a k-d tree
12 tree = cKDTree(points)
13
14 # For each circle in the query,
15 # look for points inside
16 result = set()
17
18 for k in range(5):
19     center = range_centers[k]
20     radius = range_radii[k]
21     partial_query = tree.query_ball_point(center, radius)
22     result = result.union(set(partial_query))
23
24 print result
```



# Combinatorial Computational Geometry

## Geometric Query Problems—Range Searching

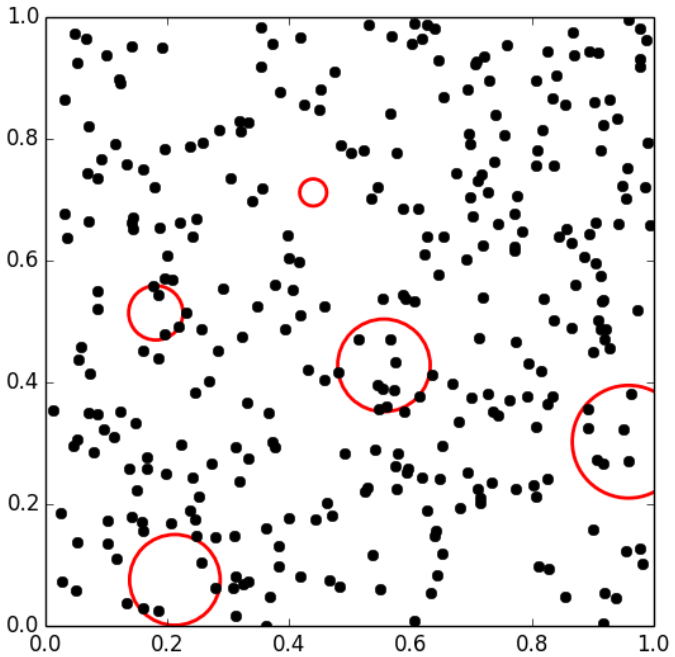
```
1  # Preamble
2  from scipy.spatial import cKDTree
3
4  # input: 320 random points
5  # query: 5 random circles
6  points = np.random.rand(320, 2)
7  range_centers = np.random.rand(5, 2)
8  range_radii = 0.1 * np.random.rand(5)
9
10 # To perform fast searches,
11 # store points in a k-d tree
12 tree = cKDTree(points)
13
14 # For each circle in the query,
15 # look for points inside
16 result = set()
17
18 for k in range(5):
19     center = range_centers[k]
20     radius = range_radii[k]
21     partial_query = tree.query_ball_point(center, radius)
22     result = result.union(set(partial_query))
23
24 print result
```



# Combinatorial Computational Geometry

## Geometric Query Problems—Range Searching

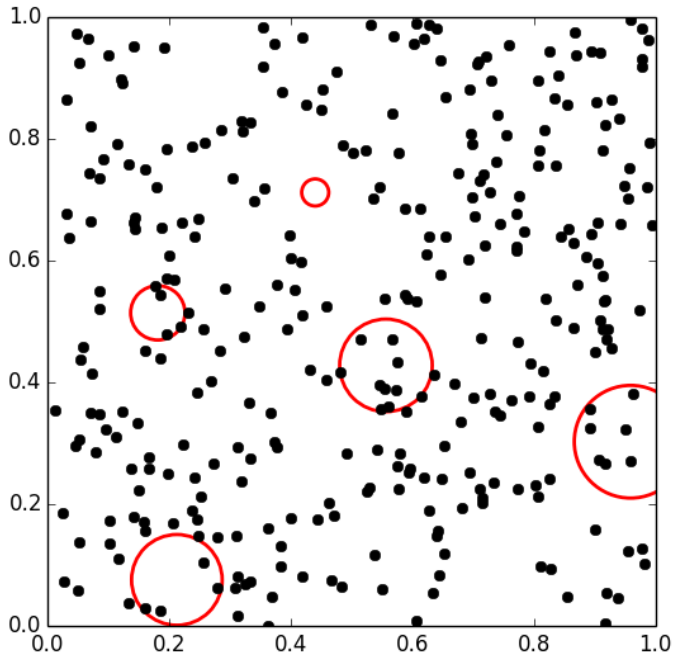
```
1  # Preamble
2  from scipy.spatial import cKDTree
3
4  # input: 320 random points
5  # query: 5 random circles
6  points = np.random.rand(320, 2)
7  range_centers = np.random.rand(5, 2)
8  range_radii = 0.1 * np.random.rand(5)
9
10 # To perform fast searches,
11 # store points in a k-d tree
12 tree = cKDTree(points)
13
14 # For each circle in the query,
15 # look for points inside
16 result = set()
17
18 for k in range(5):
19     center = range_centers[k]
20     radius = range_radii[k]
21     partial_query = tree.query_ball_point(center, radius)
22     result = result.union(set(partial_query))
23
24 print result
```



# Combinatorial Computational Geometry

## Geometric Query Problems—Range Searching

```
1  # Preamble
2  from scipy.spatial import cKDTree
3
4  # input: 320 random points
5  # query: 5 random circles
6  points = np.random.rand(320, 2)
7  range_centers = np.random.rand(5, 2)
8  range_radii = 0.1 * np.random.rand(5)
9
10 # To perform fast searches,
11 # store points in a k-d tree
12 tree = cKDTree(points)
13
14 # For each circle in the query,
15 # look for points inside
16 result = set()
17
18 for k in range(5):
19     center = range_centers[k]
20     radius = range_radii[k]
21     partial_query = tree.query_ball_point(center, radius)
22     result = result.union(set(partial_query))
23
24 print result
```




```
set([126, 43, 266, 77, 78, 273, 107, 270, 25, 283, 229, 38, 39,
104, 299, 44, 239, 49, 53, 56, 168, 60, 190])
```

# FOR MORE INFORMATION, EXAMPLES, IDEAS, . . .


blancosilva.wordpress.com