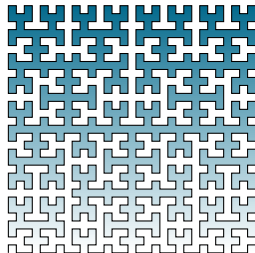


# Computational Geometry in Python

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?

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](http://nbviewer.ipython.org), by requesting:

The screenshot shows the nbviewer website interface. At the top, there's a navigation bar with links like 'Data A...', 'Task M...', 'Admin...', 'Lesson...', 'Choo...', 'Time S...', 'nbview...', and 'Reader'. Below this, the main heading is 'nbviewer' with the subtitle 'A simple way to share IPython Notebooks'. A search bar contains the text 'people.math.sc.edu/blanco/Computational Geometry in Python.ipynb' and a 'Go!' button. Under the heading 'Programming Languages', there are three columns: 'Python', 'IRuby', and 'Julia'. The 'Python' column is highlighted, showing a preview of an IPython notebook titled 'Python for Signal Processing' by O'Reilly Book. The 'IRuby' column shows a preview of an IRuby Notebook titled 'Mining the Social Web'. The 'Julia' column shows a preview of a Julia notebook titled 'PROBABILISTIC PROGRAMMING: ADVANCED METHODS FOR HACKERS'.

```

nbviewer-ipython.org?url=https://math.sc.edu/blanco/ComputationalK20GeometryK20DyOython.ipyyn
nbviewer-ipython.org?url=https://math.sc.edu/blanco/ComputationalK20GeometryK20DyOython.ipyyn
Data A. Task M. Autore. Lessons. Cheest. Time S. nbviewer. 39
In [85]: from scipy.sparse import lil_matrix
         from scipy.sparse.csrgraph import shortest_path

In [86]: nverts = len(concfig20adt['vertices'])
         G = lil_matrix((nverts, nverts))

In [87]: for k in range(len(X1)):
         G[X1[k], Y[k]] = G[Y[k], X[k]] = lengthXY[k]
         G[X1[k], X1[k]] = G[Y[k], Y[k]] = lengthXY[k]
         G[Y[k], X[k]] = G[X[k], Y[k]] = lengthXY[k]

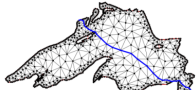
In [88]: dist_matrix, pred = shortest_path(G, return_predecessors=True, directed=False)

In [89]: index = 370
         path = [370]
         while index != 197:
             index = pred[index], index]
             path.append(index)

         Xn = [concfig20adt['vertices'][x][0] for x in path]
         Yn = [concfig20adt['vertices'][x][1] for x in path]

In [90]: plt.figure(figsize=(10,10))
         ax = plt.subplot(111, aspect='equal')
         tcolor=plt.cm._cm.cmap2color
         ax.plot(Xn, Yn, "-", linewidth=3, color='blue') \
             plt.show()

```



# 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

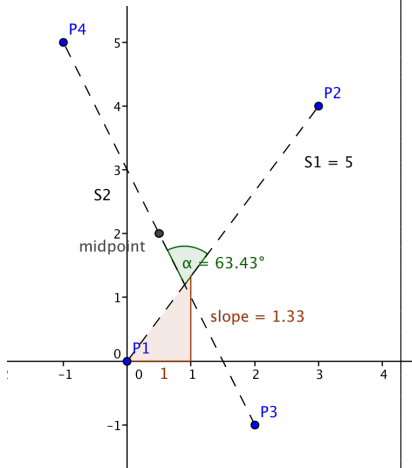
# PLANE GEOMETRY

## 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

```



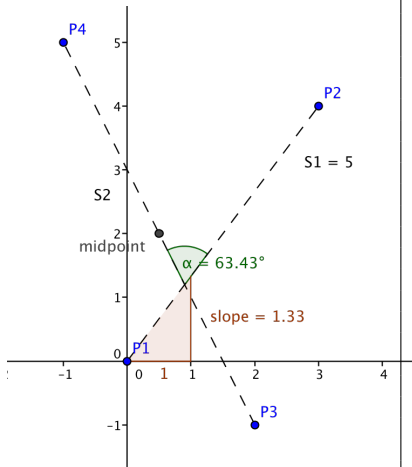
# PLANE GEOMETRY

## 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

```



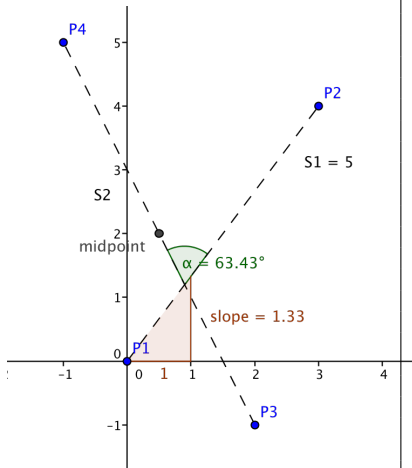
# PLANE GEOMETRY

## 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

```



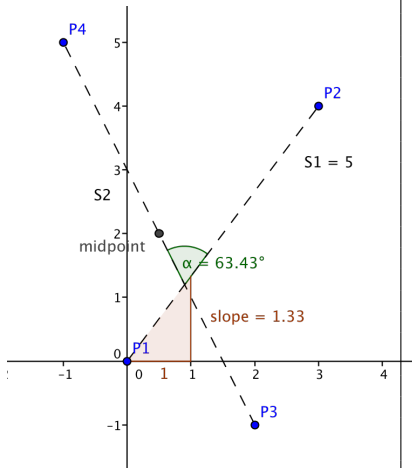
# PLANE GEOMETRY

## 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

```





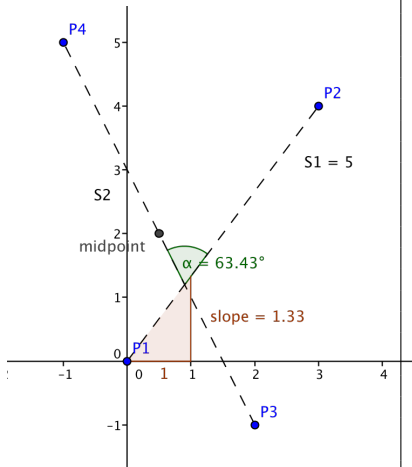
# PLANE GEOMETRY

## 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

```



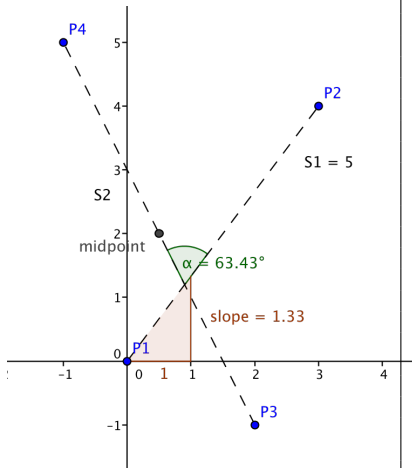
# PLANE GEOMETRY

## BASIC OBJECTS, PROPERTIES, METHODS

```

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

```



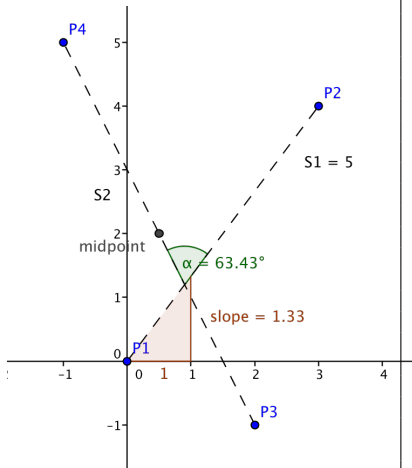
# PLANE GEOMETRY

## 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

```



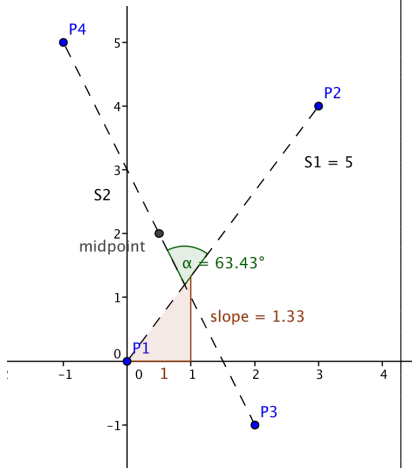
# PLANE GEOMETRY

## 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

```



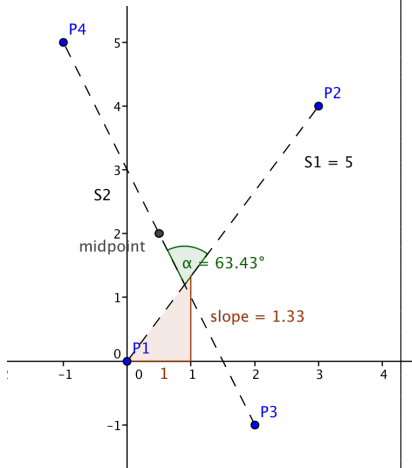
# PLANE GEOMETRY

## 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

```



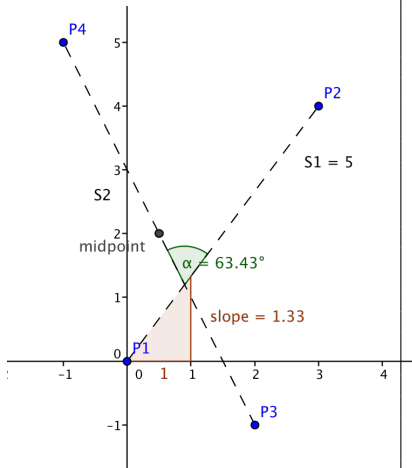
# PLANE GEOMETRY

## 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

```



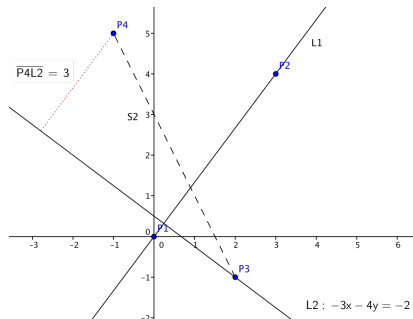
# PLANE GEOMETRY

## BASIC OBJECTS, PROPERTIES, METHODS

```

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

```



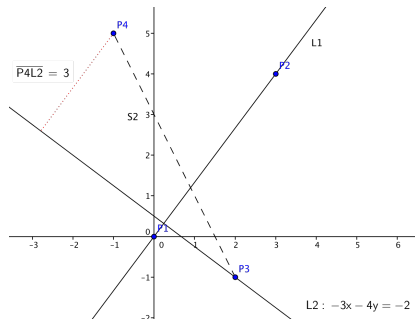
# PLANE GEOMETRY

## 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

```





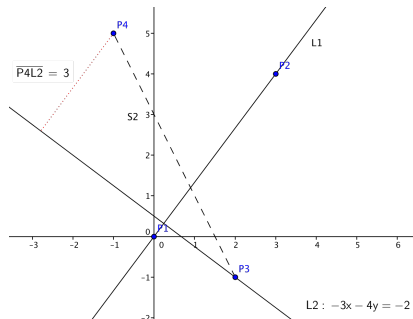
# PLANE GEOMETRY

## 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

```



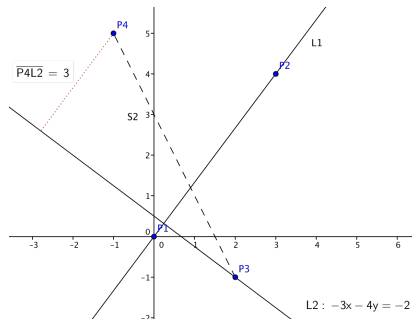
# PLANE GEOMETRY

## BASIC OBJECTS, PROPERTIES, METHODS

```

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

```



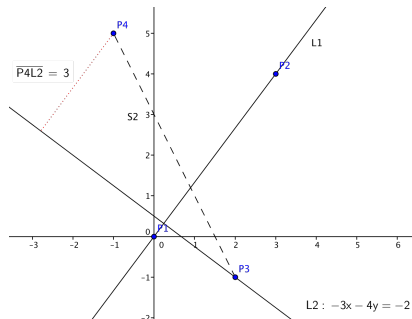
# PLANE GEOMETRY

## 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

```



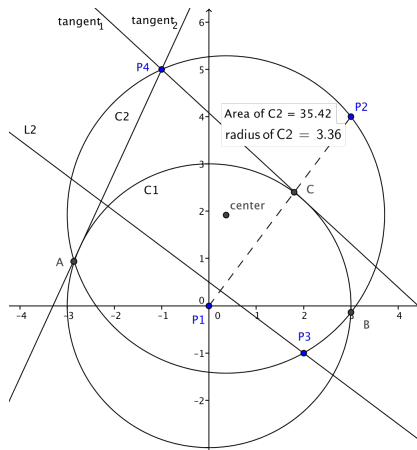
# PLANE GEOMETRY

## BASIC OBJECTS, PROPERTIES, METHODS

```

1  # Creation of some circles
2  C1 = Circle(P1, 3)
3  C2 = Circle(P2, P3, P4)
4  # Equations
5  >>> C2.equation()
6  (x - 5/14)**2 + ...
7  # Let's ask some questions
8  >>> C2.area
9  1105*pi/98
10 >>> C2.radius
11 sqrt(2210)/14
12 >>> C2.center
13 Point(5/14, 27/14)
14 >>> C2.circumference
15 sqrt(2210)*pi/7
16 >>> C1.tangent_lines(P4)
17 [Line ...,
18  Line ...]
19 # Intersections with other objects
20 >>> C2.intersection(C1)
21 [Point(55/754 + ...,
22  Point(55/754 - ...)]
23 >>> C2.intersection(S1)
24 [Point(3, 4)]
25 >>> C2.is_tangent(L2)
26 False

```



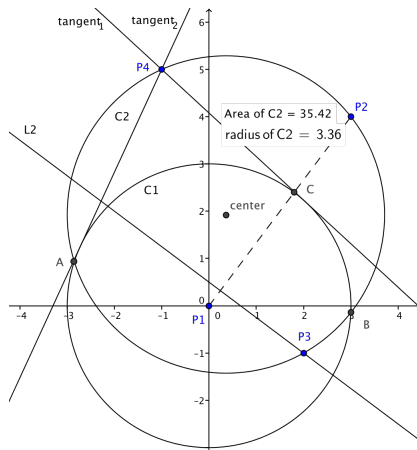
# PLANE GEOMETRY

## BASIC OBJECTS, PROPERTIES, METHODS

```

1  # Creation of some circles
2  C1 = Circle(P1, 3)
3  C2 = Circle(P2, P3, P4)
4  # Equations
5  >>> C2.equation()
6  (x - 5/14)**2 + ...
7  # Let's ask some questions
8  >>> C2.area
9  1105*pi/98
10 >>> C2.radius
11 sqrt(2210)/14
12 >>> C2.center
13 Point(5/14, 27/14)
14 >>> C2.circumference
15 sqrt(2210)*pi/7
16 >>> C1.tangent_lines(P4)
17 [Line ...,
18  Line ...]
19 # Intersections with other objects
20 >>> C2.intersection(C1)
21 [Point(55/754 + ...,
22  Point(55/754 - ...)]
23 >>> C2.intersection(S1)
24 [Point(3, 4)]
25 >>> C2.is_tangent(L2)
26 False

```



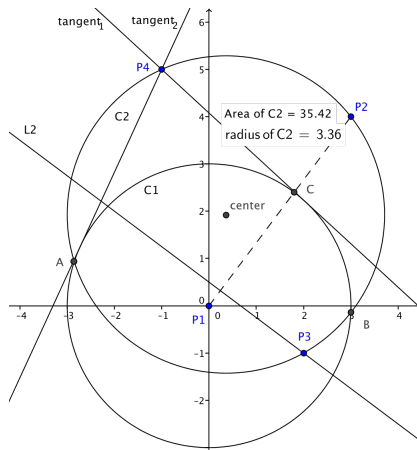
# PLANE GEOMETRY

## BASIC OBJECTS, PROPERTIES, METHODS

```

1  # Creation of some circles
2  C1 = Circle(P1, 3)
3  C2 = Circle(P2, P3, P4)
4  # Equations
5  >>> C2.equation()
6  (x - 5/14)**2 + ...
7  # Let's ask some questions
8  >>> C2.area
9  1105*pi/98
10 >>> C2.radius
11 sqrt(2210)/14
12 >>> C2.center
13 Point(5/14, 27/14)
14 >>> C2.circumference
15 sqrt(2210)*pi/7
16 >>> C1.tangent_lines(P4)
17 [Line ...,
18  Line ...]
19 # Intersections with other objects
20 >>> C2.intersection(C1)
21 [Point(55/754 + ...,
22  Point(55/754 - ...)]
23 >>> C2.intersection(S1)
24 [Point(3, 4)]
25 >>> C2.is_tangent(L2)
26 False

```



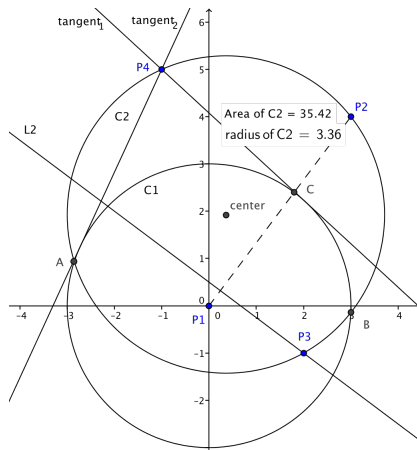
# PLANE GEOMETRY

## BASIC OBJECTS, PROPERTIES, METHODS

```

1  # Creation of some circles
2  C1 = Circle(P1, 3)
3  C2 = Circle(P2, P3, P4)
4  # Equations
5  >>> C2.equation()
6  (x - 5/14)**2 + ...
7  # Let's ask some questions
8  >>> C2.area
9  1105*pi/98
10 >>> C2.radius
11 sqrt(2210)/14
12 >>> C2.center
13 Point(55/754, 27/14)
14 >>> C2.circumference
15 sqrt(2210)*pi/7
16 >>> C1.tangent_lines(P4)
17 [Line ...,
18  Line ...]
19 # Intersections with other objects
20 >>> C2.intersection(C1)
21 [Point(55/754 + ...,
22  Point(55/754 - ...)]
23 >>> C2.intersection(S1)
24 [Point(3, 4)]
25 >>> C2.is_tangent(L2)
26 False

```



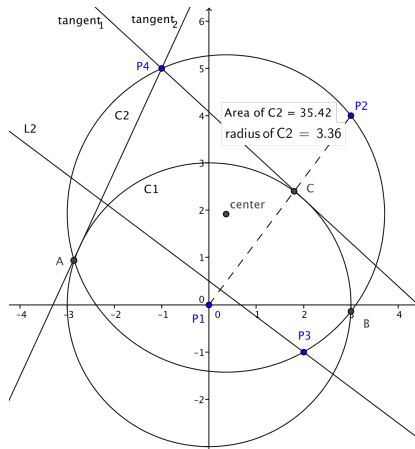
# PLANE GEOMETRY

## BASIC OBJECTS, PROPERTIES, METHODS

```

1  # Creation of some circles
2  C1 = Circle(P1, 3)
3  C2 = Circle(P2, P3, P4)
4  # Equations
5  >>> C2.equation()
6  (x - 5/14)**2 + ...
7  # Let's ask some questions
8  >>> C2.area
9  1105*pi/98
10 >>> C2.radius
11 sqrt(2210)/14
12 >>> C2.center
13 Point(5/14, 27/14)
14 >>> C2.circumference
15 sqrt(2210)*pi/7
16 >>> C1.tangent_lines(P4)
17 [Line ...,
18  Line ...]
19 # Intersections with other objects
20 >>> C2.intersection(C1)
21 [Point(55/754 + ...,
22  Point(55/754 - ...)]
23 >>> C2.intersection(S1)
24 [Point(3, 4)]
25 >>> C2.is_tangent(L2)
26 False

```





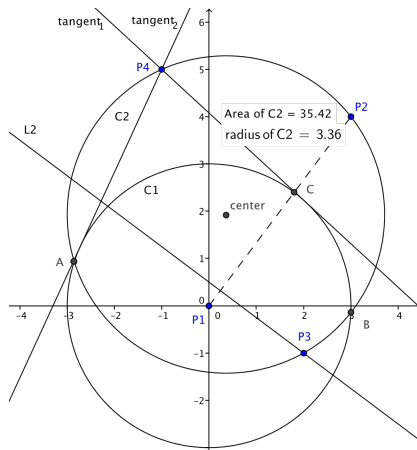
# PLANE GEOMETRY

## BASIC OBJECTS, PROPERTIES, METHODS

```

1  # Creation of some circles
2  C1 = Circle(P1, 3)
3  C2 = Circle(P2, P3, P4)
4  # Equations
5  >>> C2.equation()
6  (x - 5/14)**2 + ...
7  # Let's ask some questions
8  >>> C2.area
9  1105*pi/98
10 >>> C2.radius
11 sqrt(2210)/14
12 >>> C2.center
13 Point(5/14, 27/14)
14 >>> C2.circumference
15 sqrt(2210)*pi/7
16 >>> C1.tangent_lines(P4)
17 [Line ...,
18  Line ...]
19 # Intersections with other objects
20 >>> C2.intersection(C1)
21 [Point(55/754 + ...,
22  Point(55/754 - ...)]
23 >>> C2.intersection(S1)
24 [Point(3, 4)]
25 >>> C2.is_tangent(L2)
26 False

```



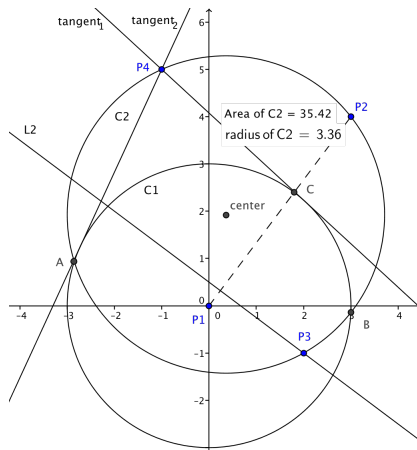
# PLANE GEOMETRY

## BASIC OBJECTS, PROPERTIES, METHODS

```

1  # Creation of some circles
2  C1 = Circle(P1, 3)
3  C2 = Circle(P2, P3, P4)
4  # Equations
5  >>> C2.equation()
6  (x - 5/14)**2 + ...
7  # Let's ask some questions
8  >>> C2.area
9  1105*pi/98
10 >>> C2.radius
11 sqrt(2210)/14
12 >>> C2.center
13 Point(5/14, 27/14)
14 >>> C2.circumference
15 sqrt(2210)*pi/7
16 >>> C1.tangent_lines(P4)
17 [Line ...,
18  Line ...]
19 # Intersections with other objects
20 >>> C2.intersection(C1)
21 [Point(55/754 + ...,
22  Point(55/754 - ...)]
23 >>> C2.intersection(S1)
24 [Point(3, 4)]
25 >>> C2.is_tangent(L2)
26 False

```



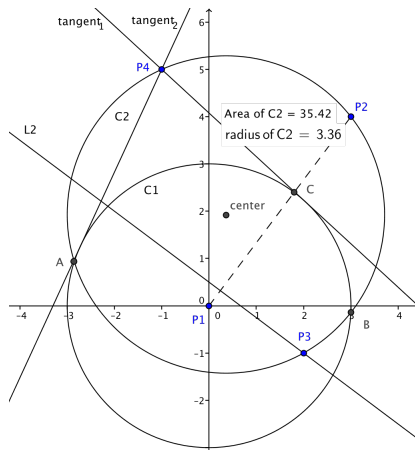
# PLANE GEOMETRY

## BASIC OBJECTS, PROPERTIES, METHODS

```

1  # Creation of some circles
2  C1 = Circle(P1, 3)
3  C2 = Circle(P2, P3, P4)
4  # Equations
5  >>> C2.equation()
6  (x - 5/14)**2 + ...
7  # Let's ask some questions
8  >>> C2.area
9  1105*pi/98
10 >>> C2.radius
11 sqrt(2210)/14
12 >>> C2.center
13 Point(5/14, 27/14)
14 >>> C2.circumference
15 sqrt(2210)*pi/7
16 >>> C1.tangent_lines(P4)
17 [Line ...,
18  Line ...]
19 # Intersections with other objects
20 >>> C2.intersection(C1)
21 [Point(55/754 + ...,
22  Point(55/754 - ...)]
23 >>> C2.intersection(S1)
24 [Point(3, 4)]
25 >>> C2.is_tangent(L2)
26 False

```



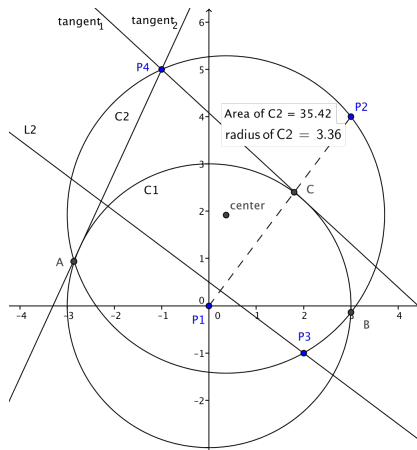
# PLANE GEOMETRY

## BASIC OBJECTS, PROPERTIES, METHODS

```

1  # Creation of some circles
2  C1 = Circle(P1, 3)
3  C2 = Circle(P2, P3, P4)
4  # Equations
5  >>> C2.equation()
6  (x - 5/14)**2 + ...
7  # Let's ask some questions
8  >>> C2.area
9  1105*pi/98
10 >>> C2.radius
11 sqrt(2210)/14
12 >>> C2.center
13 Point(5/14, 27/14)
14 >>> C2.circumference
15 sqrt(2210)*pi/7
16 >>> C1.tangent_lines(P4)
17 [Line ...,
18  Line ...]
19 # Intersections with other objects
20 >>> C2.intersection(C1)
21 [Point(55/754 + ...,
22  Point(55/754 - ...)]
23 >>> C2.intersection(S1)
24 [Point(3, 4)]
25 >>> C2.is_tangent(L2)
26 False

```



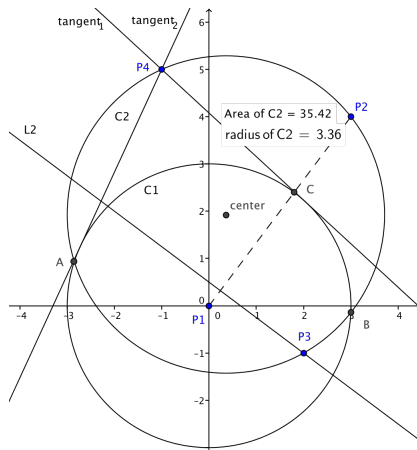
# PLANE GEOMETRY

## BASIC OBJECTS, PROPERTIES, METHODS

```

1  # Creation of some circles
2  C1 = Circle(P1, 3)
3  C2 = Circle(P2, P3, P4)
4  # Equations
5  >>> C2.equation()
6  (x - 5/14)**2 + ...
7  # Let's ask some questions
8  >>> C2.area
9  1105*pi/98
10 >>> C2.radius
11 sqrt(2210)/14
12 >>> C2.center
13 Point(5/14, 27/14)
14 >>> C2.circumference
15 sqrt(2210)*pi/7
16 >>> C1.tangent_lines(P4)
17 [Line ...,
18  Line ...]
19 # Intersections with other objects
20 >>> C2.intersection(C1)
21 [Point(55/754 + ...,
22  Point(55/754 - ...)]
23 >>> C2.intersection(S1)
24 [Point(3, 4)]
25 >>> C2.is_tangent(L2)
26 False

```



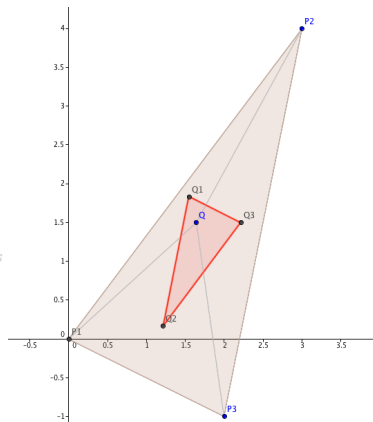
# PLANE GEOMETRY

## 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

```



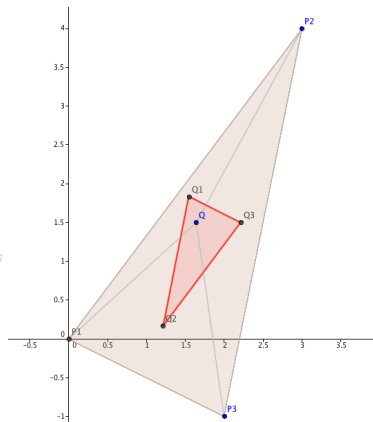
# PLANE GEOMETRY

## 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

```



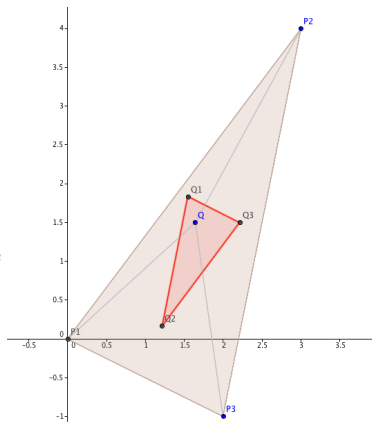
# PLANE GEOMETRY

## 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

```





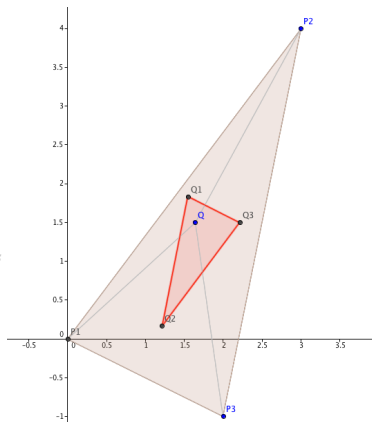
# PLANE GEOMETRY

## 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

```



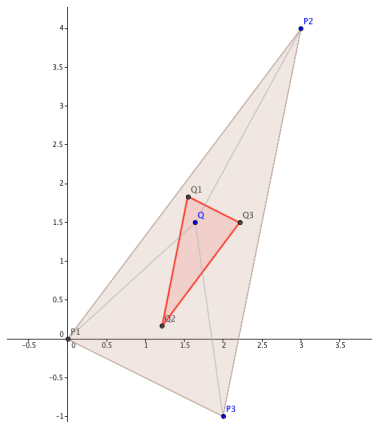
# PLANE GEOMETRY

## 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

```



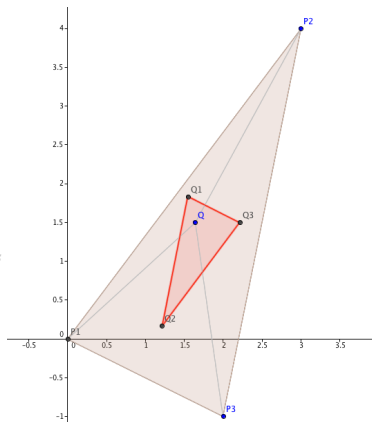
# PLANE GEOMETRY

## 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

```



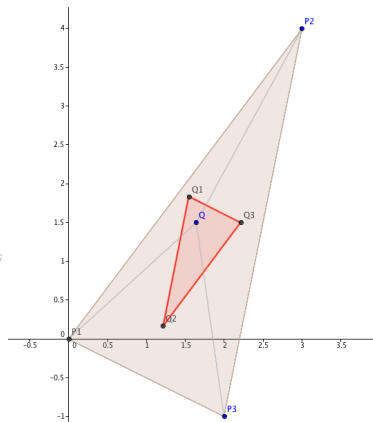
# PLANE GEOMETRY

## 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

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

# COMBINATORIAL COMPUTATIONAL GEOMETRY

## STATIC PROBLEMS—CONVEX HULLS

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

# COMBINATORIAL COMPUTATIONAL GEOMETRY

## STATIC PROBLEMS—CONVEX HULLS

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

# COMBINATORIAL COMPUTATIONAL GEOMETRY

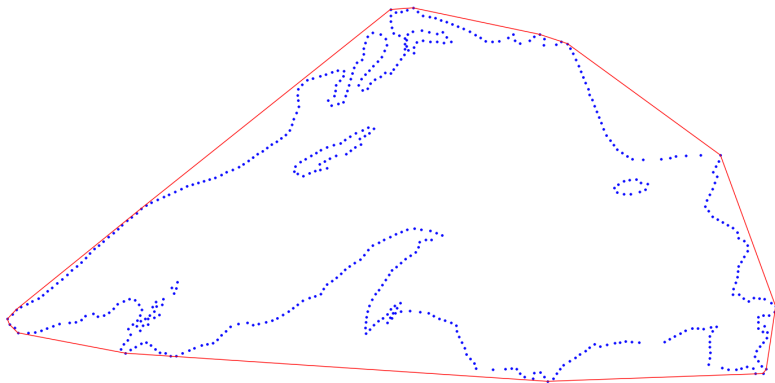
## STATIC PROBLEMS—CONVEX HULLS

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



# COMBINATORIAL COMPUTATIONAL GEOMETRY

## STATIC PROBLEMS—CONVEX HULLS



# 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

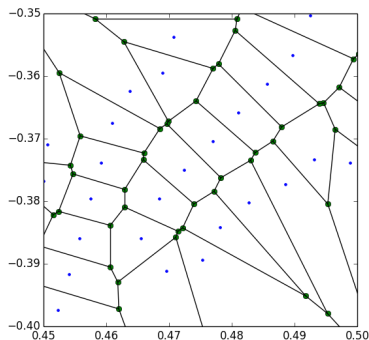
## STATIC PROBLEMS—VORONOI DIAGRAMS

```
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—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

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

# COMBINATORIAL COMPUTATIONAL GEOMETRY

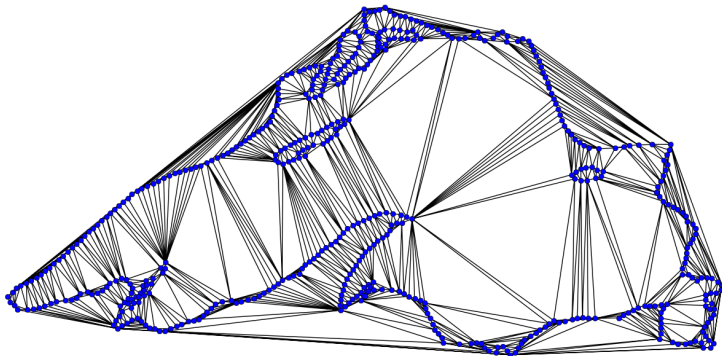
## STATIC PROBLEMS—TRIANGULATIONS

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

# COMBINATORIAL COMPUTATIONAL GEOMETRY

## STATIC PROBLEMS—TRIANGULATIONS

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



# COMBINATORIAL COMPUTATIONAL GEOMETRY

## STATIC PROBLEMS—TRIANGULATIONS

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



# COMBINATORIAL COMPUTATIONAL GEOMETRY

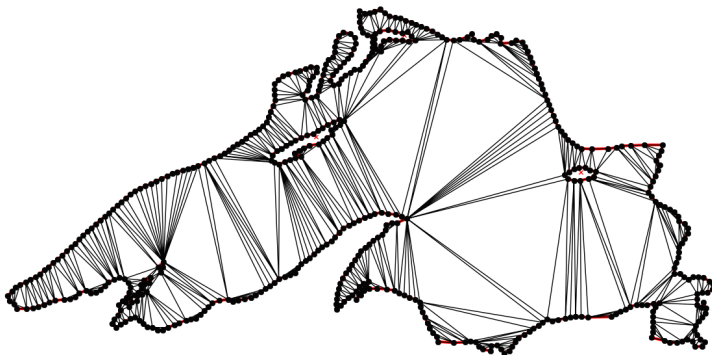
## STATIC PROBLEMS—TRIANGULATIONS

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

# COMBINATORIAL COMPUTATIONAL GEOMETRY

## STATIC PROBLEMS—TRIANGULATIONS

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



# COMBINATORIAL COMPUTATIONAL GEOMETRY

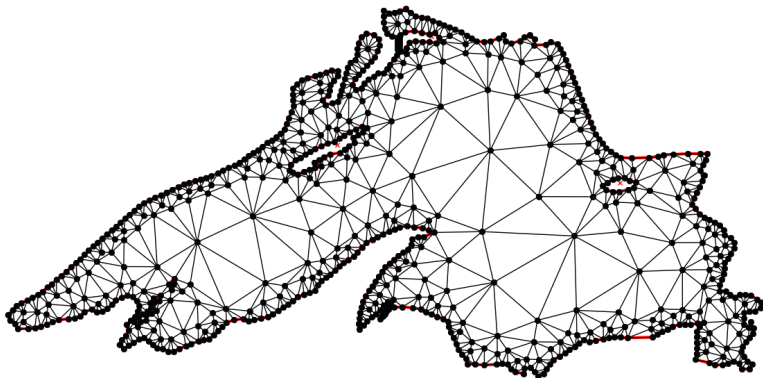
## STATIC PROBLEMS—TRIANGULATIONS

```
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

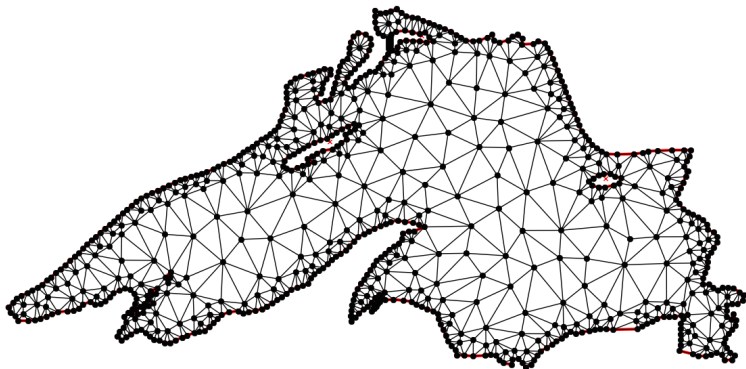
```
1 # Constrained Conforming Delaunay Triangulation
2 # Impose minimum angles of 20 degrees (whenever possible)
3 cncfq20dt = triangulate(lake_superior, 'pq20D')
4 # Generate diagram
5 ax = plt.subplot(111, aspect='equal')
6 tplot.plot(ax, **cncfq20dt)
7 plt.show()
```



# 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

```
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]
```



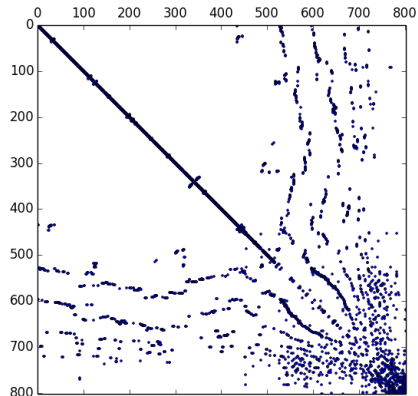
# COMBINATORIAL COMPUTATIONAL GEOMETRY

## STATIC PROBLEMS—SHORTEST PATH

```
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()
```

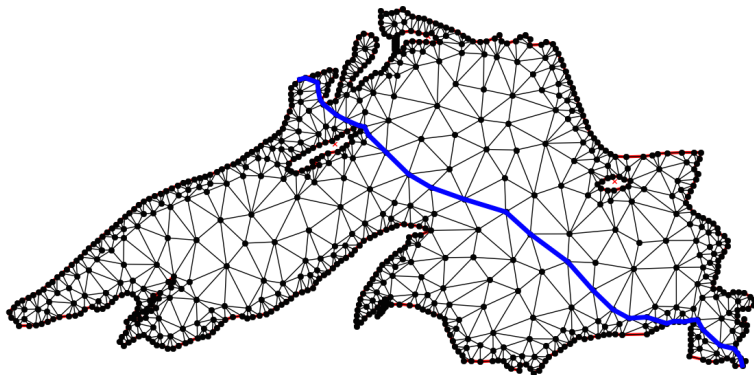
# COMBINATORIAL COMPUTATIONAL GEOMETRY

## 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

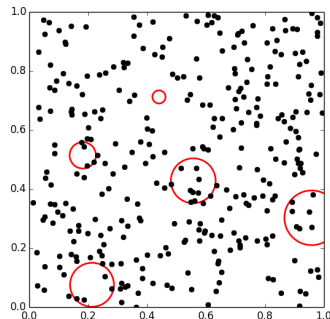
## STATIC PROBLEMS—SHORTEST PATH



# COMBINATORIAL COMPUTATIONAL GEOMETRY

## GEOMETRIC QUERY PROBLEMS—RANGE SEARCHING

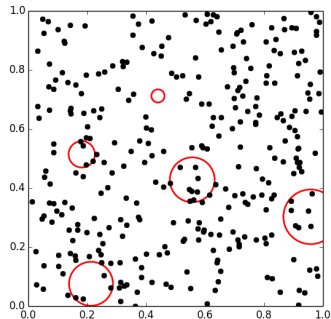
```
1  # Preamble
2  from scipy.spatial import cKDTree
3
4  # input: 320 random points
5  # query: 5 random circles
6  points = np.random.rand(320, 2)
7  range_centers = np.random.rand(5, 2)
8  range_radii = 0.1 * np.random.rand(5)
9
10 # To perform fast searches,
11 # store points in a k-d tree
12 tree = 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
```

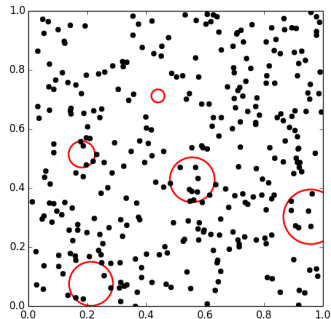




# COMBINATORIAL COMPUTATIONAL GEOMETRY

## GEOMETRIC QUERY PROBLEMS—RANGE SEARCHING

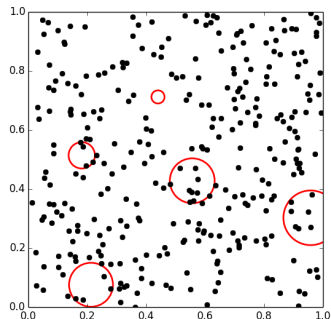
```
1  # Preamble
2  from scipy.spatial import cKDTree
3
4  # input: 320 random points
5  # query: 5 random circles
6  points = np.random.rand(320, 2)
7  range_centers = np.random.rand(5, 2)
8  range_radii = 0.1 * np.random.rand(5)
9
10 # To perform fast searches,
11 # store points in a k-d tree
12 tree = 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

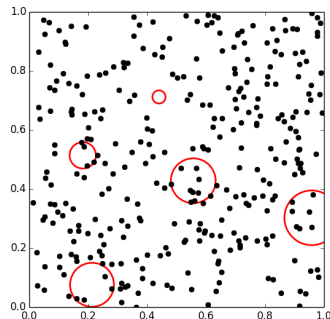
```
1  # Preamble
2  from scipy.spatial import cKDTree
3
4  # input: 320 random points
5  # query: 5 random circles
6  points = np.random.rand(320, 2)
7  range_centers = np.random.rand(5, 2)
8  range_radii = 0.1 * np.random.rand(5)
9
10 # To perform fast searches,
11 # store points in a k-d tree
12 tree = 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, ...

Searching (again?) for the SS Central America | Francisco Blanco-Silva

blancosilva.wordpress.com/2014/08/31/searching-again-for-the-ss-central-amer

Francisco Blanco-Silva... Follow Like Retweet

**Francisco Blanco-Silva**  
Useless math is useful math. It can be used to generate more useless math.

About me Books Curriculum Vitae Research Teaching Problem Solving LaTeX

Home > Applied Mathematics, Data mining, History, Probability, Python, sage, Scientific Computing, Statistics > Searching (again?) for the SS Central America

### Searching (again?) for the SS Central America

August 31, 2014

Go to comments Leave a comment Edit



On Tuesday, September 8th 1857, the steamboat SS Central America left Havana at 9 AM for New York, carrying about 600 passengers and crew members. Inside of this vessel, there was stowed a very precious cargo: a set of manuscripts by John James Audubon, and three tons of gold bars and coins. The manuscripts documented an expedition through the yet uncharted southwestern United States and California, and contained 200 sketches and paintings of its wildlife. The gold, fruit of many years of prospecting and mining during the California Gold Rush, was meant to start anew the lives of many of the passengers aboard.

On the 9th, the vessel ran into a storm which developed into a hurricane. The steamboat endured four hard days at sea, and by Saturday morning the ship was doomed. The captain arranged to have women and children taken off to the brig Marine, which offered them assistance at about noon. In spite of the efforts of the remaining crew and passengers to save the ship, the inevitable happened at about 8 PM that same day. The wreck claimed the lives of 425 men, and carried the valuable cargo to the bottom of the sea.


It was not until late 1990s that technology allowed recovery of shipwrecks at deep sea. But no technology would be of any help without an accurate location of the site. In the following paragraphs we would like to illustrate the power of the `scipy` stack by performing a simple simulation, that ultimately creates a dataset of possible locations for the wreck of the SS Central America, and mines the data to attempt to pinpoint the most probable target.

We simulate several possible paths of the steamboat (say 10,000 randomly generated possibilities), between 7:00 AM on Saturday, and 13 hours later, at 8:00 pm on Sunday. At 7:00 AM on that Saturday the ship's captain, William Harrison, took a celestial fix and verbally relayed

Blanco-Silva's Books

in the news: Mathematics adjacent welcomes visiting research students

blancosilva.wordpress.com



Community Experience Distilled

## Learning SciPy for Numerical and Scientific Computing

A practical tutorial that guarantees fast, accurate, and easy-to-code solutions to your numerical and scientific computing problems with the power of SciPy and Python

Francisco J. Blanco-Silva

**[PACKT]** open source  
PUBLISHING