

Funktion zur Konstruktion des Voronoi-Diagramms für eine Punktwolke.

```
def make_voronoi_diagram(cloud, verbose=False):
    """Compute the Voronoi-diagram for the cloud of points,
    assumedly given as n x 2 array. We assume that the points
    are _pairwise disjoint_!"""
    # Als erstes: Punktwolke, lexikographisch sortiert
    # "von oben nach unten, dann von rechts nach links":
    cop = cloud[np.lexsort(cloud.T)::-1]

    # Aus Gründen, die später klar werden, geben wir noch
    # einen "ganz weit unten" liegenden Punkt dazu:
    MINUSINFINITY = -10**9
    cop = np.append(cop, np.array([[0,MINUSINFINITY]], dtype=float), axis=0)

    # Die beachline ist gegeben durch Liste der INDICES der Brennpunkte
    # (können sich wiederholen!), zu Beginn besteht sie also nur
    # aus den ersten Brennpunkten mit Index 0 und 1:
    beachline = DoublyLinkedList()
    if cop[0][1] == cop[1][1]:
        # Die ersten Brennpunkte liegen auf derselben Höhe,
        # der zweite (Index 1) liegt links vom ersten (Index 0):
        beachline.push(1)
        beachline.push(0) # ... "von rechts nach links sortiert!"
    else:
        # Der zweite Brennpunkt liegt unterhalb, die durch ihn
        # bestimmte Parabel "durchstößt von unten" die durch den
        # ersten Brennpunkt bestimmte Parabel:
        beachline.push(0)
        beachline.push(1)
        beachline.push(0)

    def str_beachline():
        return "("+".".join(beachline.apply(lambda x : str(x.contents)))+")"

    # Wir bauen das Voronoi-Diagramm auf als ein Dictionary,
    # dessen keys (geordnete) Paare von Indices von Punkten
    # sind, deren Streckensymmetralen Kanten des Diagramms
    # enthalten, und dessen values die Punkte (oder: der Punkt)
    # sind, die diese Kanten begrenzen.
    voronoi_diagram = dict()
    # Hilfsfunktion: Sorge dafür, dass ein Tupel (i,j)
    # geordnet wird (also i<j gilt).

    def edge_key(i, j):
        """Key for edge bordering two cells with centers
        cloud[i], cloud[j] in the Voronoi diagram (i.e.:
        edge is part of the bisector of line (cloud[i], cloud[j])).
        """
        return (i,j) if i<j else (j,i)

    # Arbeite nun der Reihe nach die "site-events" ab, die durch
    # die restlichen Punkte der cloud of points (cop) gegeben sind:
    # Durch die lexikographische Sortierung entspricht das einem
    # "Herabsinken der Leitlinie von oben nach unten":
    old_site_event_height = cop[1][1]
    if verbose:
        print(f"Initial site height: {old_site_event_height}")
    for point_index,point in enumerate(cop[2:], start=2):
        site_event_height = point[1]
        if verbose:
            print(f"\tNew site-event {point_index}: {point}?")
            print(str_beachline())
        # Als erstes müssen wir aber schauen, ob es circle events
        # _oberhalb_ dieser site_event_height_ gibt; dazu muss
        # die beachline zumindest Länge 3 haben:
        if beachline.length > 2:
            left = beachline.end["left"]
            middle = left.neighbour["right"]
            right = middle.neighbour["right"]
            while right:
                # Indices der Brennpunkte:
                li, mi, ri = [
                    node.contents for node in [left, middle, right]
                ]
                # Einen eindeutigen Schnittpunkt der Streckensymmetralen
                # von (li, mi) und (mi, ri) gibt es natürlich nur, wenn
                # li != ri ist.
                if li != ri:
                    # Berechne den Schnittpunkt der Streckensymmetralen und
                    # die Höhe, auf der das entsprechende circle-event
                    # passiert:
                    circle_event_height, center = gives_circle_event(
```

```

        cop[li], cop[mi], cop[ri]
    )

    if (
        (not circle_event_height is None)
        and
        (old_site_event_height >
         circle_event_height >=
         site_event_height
        )
    ):
        # Dieses circle-event müssen wir nun berücksicht-
        # igen: Kanten des Voronoi-Diagramms "codieren"
        # wir als das (geordnete) Paar der Indices der
        # Punkte, deren Streckensymmetrale die Kante
        # enthält.
        for key in [
            edge_key(li, mi),
            edge_key(mi, ri),
            edge_key(li, ri)
        ]:
            if key not in voronoi_diagram:
                # Diese Kante hat (vorläufig) nur einen
                # Endpunkt:
                voronoi_diagram[key] = [center]
            else:
                # Diese Kante hat einen zweiten Endpunkt:
                voronoi_diagram[key].append(center)
        # Der "mittlere" Punkt wird nun aus der beachline
        # entfernt:
        middle.detach()
        # Nächsten Durchlauf der while-Schleife vorbereiten
        # (Knoten "middle" wurde ja grade entfernt):
        middle = left
        if verbose:
            print(f"circle_event h = {li,mi,ri} : {center}")
            print(str_beachline())
        # Nächsten Durchlauf der while-Schleife vorbereiten:
        left = middle
        middle = right
        right = middle.neighbour["right"]

# Ok: Alle circle-events, die oberhalb des aktuell betrachteten
# site-events auftreten, wurden behandelt: Wir suchen jetzt die
# Stelle, wo der neue Brennpunkt in die beachline eingefügt werden
# soll - aber NUR, wenn wir nicht unseren ganz oben eingefügten
# "zusätzlichen" Punkt erreicht haben, denn dann sind wir fertig
# (denn alle circle-events haben wir ja grade erledigt):
if site_event_height <= MINUSINFINITY:
    return voronoi_diagram
# Implicit else: Wir haben noch einen "regulären" Punkt, dessen
# site-event wir nun abarbeiten müssen.
x_coordinate = point[0]
left_node = beachline.end["left"]
site_event_done = False
while (right_node := left_node.neighbour["right"]):
    # Die Indices der Brennpunkte ...
    left_focus_index = left_node.contents
    right_focus_index = right_node.contents
    # ... und die Brennpunkte selbst:
    left_focus = cop[left_focus_index]
    right_focus = cop[right_focus_index]
    # The arc with the _lower_ focus "cuts from below";
    # this determines which of the two points of intersections
    # we should choose:
    left_or_right = 0 if left_focus[1] >= right_focus[1] else 1
    # Die x-Koordinate des Schnittpunkts
    intersection_x = parabolic_intersections(
        left_focus,
        right_focus,
        site_event_height
    )[left_or_right]
    if x_coordinate > intersection_x:
        left_node = right_node
        continue
    elif x_coordinate == intersection_x:
        # Der neu einzufügende Brennpunkt liegt "genau zwischen"
        # dem linken und dem rechten Parabelbogen:
        left_node.squeeze_in(point_index, "right")
        site_event_done = True
        break
    else: # D.h., x_coordinate < intersection_x:

```

```
# Der neu einzufügende Brennpunkt "zerschneidet" den  
# linken Parabelbogen, der nun also _zwei_ Parabelbögen  
# zur beachline beisteuert:
```

```
left_node.squeeze_in(point_index, "right")  
right_node.squeeze_in(left_focus_index, "left")  
site_event_done = True  
break
```

```
# Nicht vergessen: Nächsten Durchlauf der while-Schleife  
# vorbereiten!
```

```
left_node = right_node
```

```
if not site_event_done:
```

```
# Der neu einzufügende Brennpunkt "zerschneidet"  
# den letzten Parabelbogen:  
last_focus_index = beachline.end["right"].contents  
beachline.push(point_index)  
beachline.push(last_focus_index)
```

```
if verbose:
```

```
    print(str_beachline())
```

```
old_site_event_height = site_event_height
```

```
return voronoi_diagram
```