boom.py

```python
import numpy as np
import math
import helpers
import edges

class Boom():
    def __init__(self, number, coordinates, stringer_area, neutral_axis):
        """
        Initialise instance of boom for structural idealisation.
        :param coordinates: coordinates (z, y) of boom location. Origin is taken at hinge point.
        :param adjacents: list of adjacent booms. List of number_booms length where each element is a list that contains
        details about each adjacent boom. Each element contains [boom number, thickness of edge, length of edge].
        :param stringer_area: Area of the stringers. If there are no stringers in the place of the boom, set to 0.0.
        :param neutral_axis: line of the neutral axis. Format: (A, B, C) where the neutral axis: Ax + By + C = 0
        """
        self.neutral_axis = neutral_axis
        self.coordinates = coordinates
        self.adjacents = []
        self.stringer_area = stringer_area
        self.dist_neutral_axis = 0.0
        self.area = 0.0
        self.z_dist = 0.0
        self.y_dist = 0.0
        self.number = number
        self.dist_origin_coordinates = 0.0
        self.bending_stress = None

    def calc_distance_neutral_axis(self):
        """
        calculate and update distance from boom to neutral axis
        """
        self.dist_neutral_axis = helpers.distance_point_line(self.coordinates, self.neutral_axis)

    def calc_dist_origin_coordinates(self):
        """
        calculate the distance from the boom to the origin of coordinates. This is useful to fid the new coordinates
        after a rotation
        update value for each boom
        """
        self.dist_origin_coordinates = (self.coordinates[0] ** 2 + self.coordinates[1] ** 2) **0.5

    def update_coordinates(self, theta):
        rotation_matrix = np.array([[np.cos(theta), -np.sin(theta)],
                                    [np.sin(theta), np.cos(theta)]])
        new_coords = np.dot(rotation_matrix, np.asarray(self.coordinates))
        self.coordinates = new_coords

    def calc_y_dist(self, aileron_geometry):
        """
        :param aileron_geometry: geometry of the cross-section, we need the centroid from it
        update distance from boom to centroid in y-direction
        """
        self.y_dist = self.coordinates[1] - aileron_geometry.centroid[1]

    def calc_z_dist(self, aileron_geometry):
        """
        :param aileron_geometry: geometry of the cross-section, we need the centroid from it
        update distance from boom to centroid in z-direction
        """
        self.z_dist = self.coordinates[0] - aileron_geometry.centroid[0]

    def calculate_area(self, aileron_geometry):
        """
        Calculate area of boom following formula 20.1 of Megson
        :param aileron_geometry: instance of class Geometry describing the geometrical properties of the cross-section
        update area of boom
        """
        boom_area = self.stringer_area
        self.calc_distance_neutral_axis()
        for adjacent_edge in self.adjacents:
            if adjacent_edge.booms[0] != self.number:
                boom = adjacent_edge.booms[0]
            else:
                boom = adjacent_edge.booms[1]
            boom_obj = aileron_geometry.booms[boom]
            boom_obj.calc_distance_neutral_axis()
            t = adjacent_edge.thickness     # thickness of link
            l = adjacent_edge.length        # length of link
            if boom_obj.coordinates[0] == self.coordinates[0] and boom_obj.coordinates[1] == - self.coordinates[1]:
                ratio = -1
            else:
                if abs(self.coordinates[1]) < 0.001:
                    continue
                else:
                    ratio = boom_obj.dist_neutral_axis / self.dist_neutral_axis
            boom_area += (t * l)/6.0 * (2 + ratio)
        self.area = boom_area

    def calc_bending_stress(self, Mz, My, aileron_geometry):
```

```
        """
        Calculates bending stresses at given point (z, y) in the particular section of the aileron
        :param Mz: Moment distribution at given point in x
        :param My: Moment distribution at given point in x
        update Bending stress at given point in the cross-section at given point in x direction
        """
        moment_contribution = (Mz * self.y_dist) / aileron_geometry.Izz + (My * self.z_dist) / aileron_geometry.Iyy
        self.bending_stress = moment_contribution


```