```python
import numpy as np
import math
from helpers import *


class Geometry:
    def __init__(self, number_booms, booms, edges, cells_area, G):
        """
        :param number_booms: Number of booms in the cross section
        :param booms: list of Boom class instances containing all booms in the cross section
        :param edges: list of Edge class instances containing all edges in the cross section
        :param cells_area: list containing the areas of the cells (for multicell problems)
        :param G: shear modulus of the material
        """
        self.number_booms = number_booms
        self.booms = booms
        self.edges = edges
        self.cells = []
        self.boom_areas = np.zeros(self.number_booms)
        self.centroid = np.zeros(2)
        self.neutral_axis = ()  # in the form A, B, C : neutral axis line Az + By + C = 0
        self.z_dists = np.zeros(self.number_booms)
        self.y_dists = np.zeros(self.number_booms)
        self.Iyy = 0.0
        self.Izz = 0.0
        self.Izy = 0.0
        self.shear_center = 0.0
        self.cells_area = cells_area
        self.G = G


    def construct_geometry(self):
        """
        modify all Boom objects. For each object, modify their attribute "adjacents" to include a list of all the edges
        that contain the boom.
        """
        for element in self.booms:
            for edge in self.edges:
                if element.number in edge.booms:
                    element.adjacents.append(edge)

    def get_areas(self):
        """
        Update values in self.boom_areas to include the areas of the booms.
        """
        for i, boom in enumerate(self.booms):
            self.boom_areas[i] = boom.area

    def calc_centroid(self):
        """
        Calculate centroid position (z, y) taking as origin of coordinates the hinge point.
        Set self.centroid to calculated coordinates.
        """
        sum_y = 0.0
        sum_z = 0.0
        for boom in self.booms:
            sum_y += boom.area * boom.coordinates[1]
            sum_z += boom.area * boom.coordinates[0]

        self.centroid[1] = sum_y/sum(self.boom_areas)
        self.centroid[0] = sum_z/sum(self.boom_areas)

    def calc_y_dists(self):
        """
        Calculates the distance in y-direction from each boom to the centroid and store in a list y_dists.
        Modifies self.y_dists[]
        """
        for i, boom in enumerate(self.booms):
            self.y_dists[i] = boom.coordinates[1] - self.centroid[1]

    def calc_z_dists(self):
        """
        Calculates the distance in z-direction from each boom to the centroid and store in a list z_dists.
        Modifies self.z_dists[]
        """
        for i, boom in enumerate(self.booms):
            self.z_dists[i] = boom.coordinates[0] - self.centroid[0]

    def moment_inertia_Izz(self):
        """
        Calculates moment of inertia in z using Izz = Sigma(Bi * yi^2)
        Updates self.Izz to moment of inertia.
        """
        self.calc_y_dists()
        for i, area in enumerate(self.boom_areas):
            self.Izz += area * self.y_dists[i] ** 2

    def moment_inertia_Iyy(self):
```

```
        """
        Calculates moment of inertia in y using Izz = Sigma(Bi * zi^2)
        Updates self.Iyy to moment of inertia.
        """
        self.calc_z_dists()
        for n, area in enumerate(self.boom_areas):
            self.Iyy += area * self.z_dists[n] ** 2

    def plot_edges(self):
        """
        Plot the booms (numbered) and the edges uniting them.
        This plot is used to verify that the booms and edges created correspond to the correct geometry.
        """
        coordinates = []
        for element in self.booms:
            coordinates.append(element.coordinates)
        zs = []
        ys = []
        n = range(len(coordinates))
        for boom_coord in coordinates:
            zs.append(boom_coord[0])
            ys.append(boom_coord[1])
        for wall in self.edges:
            z_positions = [self.booms[wall.booms[0]].coordinates[0], self.booms[wall.booms[1]].coordinates[0]]
            y_positions = [self.booms[wall.booms[0]].coordinates[1], self.booms[wall.booms[1]].coordinates[1]]
            plt.plot(z_positions, y_positions, color='black')
        plt.scatter(zs, ys)
        for i, txt in enumerate(n):
            plt.annotate(txt, (zs[i], ys[i]))
        plt.show()
```