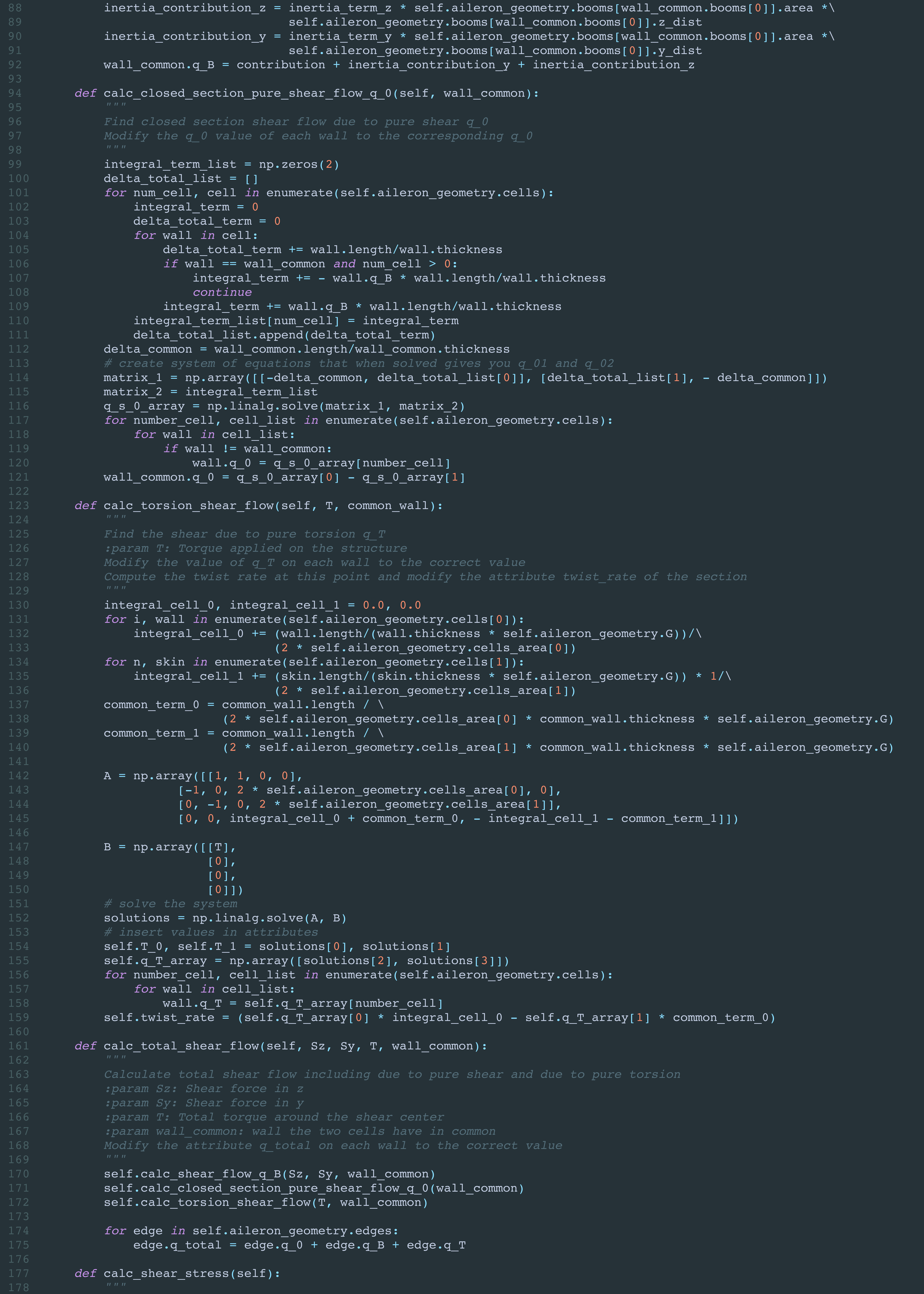
DiscreteSection.py

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


C_a = 0.547
l_a = 2.771
x_1 = 0.153
x_2 = 1.281
x_3 = 2.681
x_a = 0.28
h_a = 0.225
t_sk = 0.0011
t_sp = 0.0029
t_st = 0.0012
h_st = 0.015
w_st = 0.02
n_st = 17
d_1 = 0.1103
d_3 = 0.1642
theta = 26
P = 9170
q = 4530


class DiscreteSection:
    def __init__(self, neutral_axis, aileron_geometry):
        """
        :param neutral_axis: neutral axis of the cross section
        :param aileron_geometry: instance of the class Geometry containing information on the cross section's geometry
        """
        self.neutral_axis = neutral_axis
        self.bending_stress = None
        self.bending_deflection = None
        self.aileron_geometry = aileron_geometry
        self.twist_rate = None
        self.T_1 = None
        self.T_0 = None
        self.q_T_array = None

    def calc_shear_flow_q_B(self, Sz, Sy, wall_common):
        """
        Calculate the open section shear flow q_B by the traditional method: make an imaginary cut.
        :param Sz: Shear force in z
        :param Sy: Shear force in y
        :param wall_common: wall that both cells have in common
        Modifies the value of q_B of each wall to the correct value.
        """
        Izz = self.aileron_geometry.Izz
        Iyy = self.aileron_geometry.Iyy
        Izy = self.aileron_geometry.Izy

        inertia_term_z = - (Sz * Izz - Sy * Izy) / (Izz * Iyy - Izy ** 2)
        inertia_term_y = - (Sy * Iyy - Sz * Izy) / (Izz * Iyy - Izy ** 2)
        number_cells = len(self.aileron_geometry.cells)
        for num_cell in range(number_cells):
            # make the cut at the first wall of each cell
            wall_cut = self.aileron_geometry.cells[num_cell][0]
            wall_cut.q_B = 0.0
            accumulation_y, accumulation_z = 0.0, 0.0
            for n, wall in enumerate(self.aileron_geometry.cells[num_cell]):
                # only calculate the qB if it is not the wall that you have already cut or the common wall
                if wall == wall_cut:
                    continue
                if wall == wall_common and num_cell > 0:
                    accumulation_y += self.aileron_geometry.booms[wall.booms[1]].area * \
                                  self.aileron_geometry.booms[wall.booms[1]].y_dist
                    accumulation_z += self.aileron_geometry.booms[wall.booms[1]].area * \
                                      self.aileron_geometry.booms[wall.booms[1]].z_dist
                    continue
                accumulation_y += self.aileron_geometry.booms[wall.booms[0]].area * \
                                  self.aileron_geometry.booms[wall.booms[0]].y_dist
                accumulation_z += self.aileron_geometry.booms[wall.booms[0]].area * \
                                  self.aileron_geometry.booms[wall.booms[0]].z_dist
                if wall == wall_common:
                    continue
                wall.q_B = (inertia_term_z * accumulation_z + inertia_term_y * accumulation_y)

        # find shear flow on web
        contribution = 0.0
        for adjacent_to_common_wall in self.aileron_geometry.booms[wall_common.booms[0]].adjacents:
            if adjacent_to_common_wall != wall_common:
                if adjacent_to_common_wall in self.aileron_geometry.cells[0]:
                    contribution += adjacent_to_common_wall.q_B
                else:
                    contribution += -adjacent_to_common_wall.q_B
```



```python
        inertia_contribution_z = inertia_term_z * self.aileron_geometry.booms[wall_common.booms[0]].area *\
                                 self.aileron_geometry.booms[wall_common.booms[0]].z_dist
        inertia_contribution_y = inertia_term_y * self.aileron_geometry.booms[wall_common.booms[0]].area *\
                                 self.aileron_geometry.booms[wall_common.booms[0]].y_dist
        wall_common.q_B = contribution + inertia_contribution_y + inertia_contribution_z

    def calc_closed_section_pure_shear_flow_q_0(self, wall_common):
        """
        Find closed section shear flow due to pure shear q_0
        Modify the q_0 value of each wall to the corresponding q_0
        """
        integral_term_list = np.zeros(2)
        delta_total_list = []
        for num_cell, cell in enumerate(self.aileron_geometry.cells):
            integral_term = 0
            delta_total_term = 0
            for wall in cell:
                delta_total_term += wall.length/wall.thickness
                if wall == wall_common and num_cell > 0:
                    integral_term += - wall.q_B * wall.length/wall.thickness
                    continue
                integral_term += wall.q_B * wall.length/wall.thickness
            integral_term_list[num_cell] = integral_term
            delta_total_list.append(delta_total_term)
        delta_common = wall_common.length/wall_common.thickness
        # create system of equations that when solved gives you q_01 and q_02
        matrix_1 = np.array([[-delta_common, delta_total_list[0]], [delta_total_list[1], - delta_common]])
        matrix_2 = integral_term_list
        q_s_0_array = np.linalg.solve(matrix_1, matrix_2)
        for number_cell, cell_list in enumerate(self.aileron_geometry.cells):
            for wall in cell_list:
                if wall != wall_common:
                    wall.q_0 = q_s_0_array[number_cell]
        wall_common.q_0 = q_s_0_array[0] - q_s_0_array[1]

    def calc_torsion_shear_flow(self, T, common_wall):
        """
        Find the shear due to pure torsion q_T
        :param T: Torque applied on the structure
        Modify the value of q_T on each wall to the correct value
        Compute the twist rate at this point and modify the attribute twist_rate of the section
        """
        integral_cell_0, integral_cell_1 = 0.0, 0.0
        for i, wall in enumerate(self.aileron_geometry.cells[0]):
            integral_cell_0 += (wall.length/(wall.thickness * self.aileron_geometry.G))/\
                               (2 * self.aileron_geometry.cells_area[0])
        for n, skin in enumerate(self.aileron_geometry.cells[1]):
            integral_cell_1 += (skin.length/(skin.thickness * self.aileron_geometry.G)) * 1/\
                               (2 * self.aileron_geometry.cells_area[1])
        common_term_0 = common_wall.length / \
                        (2 * self.aileron_geometry.cells_area[0] * common_wall.thickness * self.aileron_geometry.G)
        common_term_1 = common_wall.length / \
                        (2 * self.aileron_geometry.cells_area[1] * common_wall.thickness * self.aileron_geometry.G)

        A = np.array([[1, 1, 0, 0],
                  [-1, 0, 2 * self.aileron_geometry.cells_area[0], 0],
                  [0, -1, 0, 2 * self.aileron_geometry.cells_area[1]],
                  [0, 0, integral_cell_0 + common_term_0, - integral_cell_1 - common_term_1]])

        B = np.array([[T],
                      [0],
                      [0],
                      [0]])
        # solve the system
        solutions = np.linalg.solve(A, B)
        # insert values in attributes
        self.T_0, self.T_1 = solutions[0], solutions[1]
        self.q_T_array = np.array([solutions[2], solutions[3]])
        for number_cell, cell_list in enumerate(self.aileron_geometry.cells):
            for wall in cell_list:
                wall.q_T = self.q_T_array[number_cell]
        self.twist_rate = (self.q_T_array[0] * integral_cell_0 - self.q_T_array[1] * common_term_0)

    def calc_total_shear_flow(self, Sz, Sy, T, wall_common):
        """
        Calculate total shear flow including due to pure shear and due to pure torsion
        :param Sz: Shear force in z
        :param Sy: Shear force in y
        :param T: Total torque around the shear center
        :param wall_common: wall the two cells have in common
        Modify the attribute q_total on each wall to the correct value
        """
        self.calc_shear_flow_q_B(Sz, Sy, wall_common)
        self.calc_closed_section_pure_shear_flow_q_0(wall_common)
        self.calc_torsion_shear_flow(T, wall_common)

        for edge in self.aileron_geometry.edges:
            edge.q_total = edge.q_0 + edge.q_B + edge.q_T

    def calc_shear_stress(self):
        """
        Calculate shear stress on each wall
        IMPORTANT: this function must be called AFTER self.calc_total_shear_flow()
        Modify attribute shear_stress on each wall to the correct value
        """
        for wall in self.aileron_geometry.edges:
            wall.shear_stress = wall.q_total / wall.thickness

```