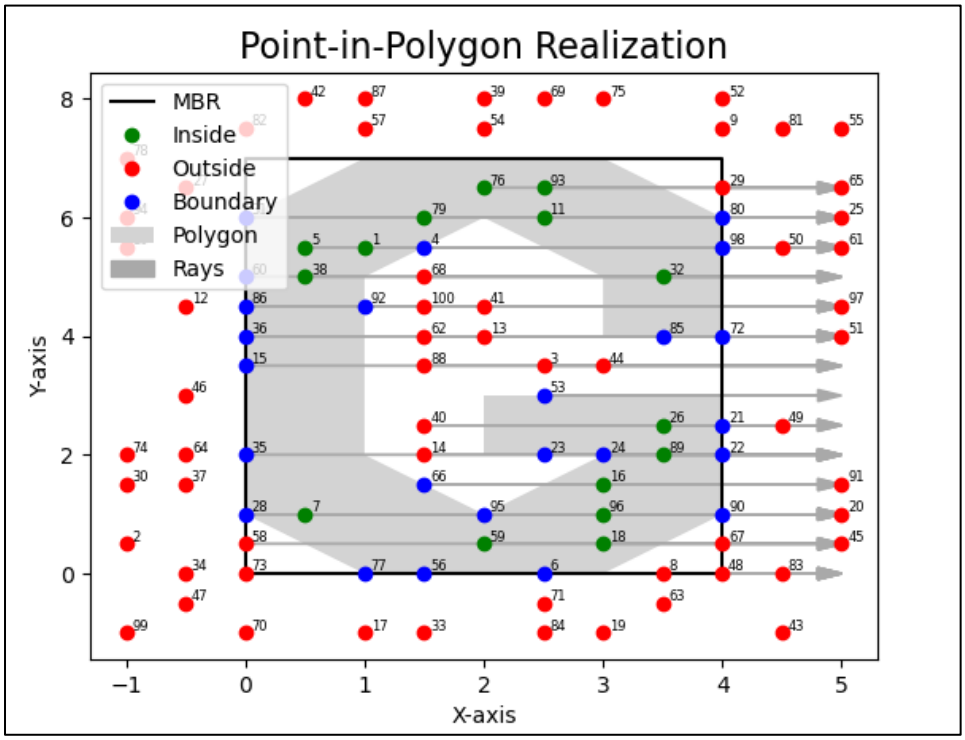
Point-in-Polygon Test

Report for the CEGE0096: 1st Assignment



Nina Moiseeva

21167320

ucesois@ucl.ac.uk

Introduction

As a result of an Assignment 1 for CEGE 0096: Geospatial Programming a program for Point-in-Polygon testing was developed. Point-in-Polygon testing consists of 3 main parts:

- The Minimum Bounding Rectangle Algorithm;
- The Point-in-Polygon Algorithm;
- Regarding of Special Cases of the Point-in-Polygon Algorithm.

The software was developed with the usage of PyCharm with Python 3.8. compilator. During the software development, coding was performed at the beginning without the use of object-oriented programming and code was transformed into the object-oriented conception only after debugging all the functions. The software was developed during 14.11.2021-22.11.2021.

Ready to use software contains 2 .py files to run depending on the user's goals:

- main_from_file.py – main program, which works with predefined files;
- main_from_user.py – program, which works with user input files;
- plotter.py – program for Plotting

The achieved results of Assignment 1 for CEGE 0096: Geospatial Programming are flagged in the table 1 (the fact of work performance is noted, and not an assessment of their quality and completeness).

Table 1. – The achieved results of Assignment 1.

n	Task Description	Status
1	The successful implementation of MBR	Done
2	The successful implementation of RCA	Done
3	The successful categorisation of special cases	Done
4	Write a report about the project	Done
5	Make your code Object-oriented	Done
6	Make regular commits on the GitHub repository	Done
7	Comment your code clearly so that it can be understood by others	Done
8	Apply PEP8 style	Done
9	Plotting of points, polygons, and ray with appropriate axis labels and annotations	Done
10	Incorporate some simple error handling functionality	Done
11	Creativity marks are available under certain conditions for adding features that have not been specified	Attempt done

Project Description

Programming Project solves the following tasks for the realization of Point-in-Polygon test:

- Reading a csv with input polygon vertexes coordinates
- Reading a csv with input point coordinates
- Realization of a Minimum Bounding Rectangle Algorithm
- Implementation of the algorithm for finding points belonging to the boundaries of the polygon
- Implementation of the Ray Casting Algorithm with regarding of special cases
- Launching classification for each point in a dictionary
- Creating an output csv
- Visualization of all objects on the graph

For software usage Limitations are as follows:

- PyCharm or Anaconda Prompt;
- Python compilator 3.8 with matplotlib library;

To use a software user need to follow the several steps:

- Download a provided Archive: point-in-polygon-test-ucesois-master.zip and
- Unzip Archive in a folder;
- If user prefer to launch a file with predefined input files, he need to run main_from_file.py;
- If user prefer to launch a file with his own input files, he need to run main_from_user.py;
- To run a choden file user can choose between PyCharm or Anaconda Prompt, but PyCharm is preferred option.

To run a chosen file in PyCharm user have to:

- Open PyCharm;
- Press File – Open File from Project and define a path to an unzipped project folder (point-in-polygon-test-ucesois-master.zip);
- With Edit Configuration choose which file to run - main_from_file.py or main_from_user.py;
- Press a Run button.

To run a chosen file in Anaconda Prompt user have to:

- Open Anaconda Prompt;
- Activate appropriate working environment;
- Type a name of a file - main_from_file.py or main_from_user.py and press Enter;

Notice, main_from_user.py will ask user about the way he wants to define an input points and will requires a polygon vertexes in csv format.

Notice, in Anaconda Prompt, it is necessary to use quotation marks for defining a path to a file.

Software

Task 1. The MBR Algorithm

The implementation of the Minimum Bounding Rectangle algorithm begins straight after loading the input data and generating global variables. The algorithm determines the minimum and maximum from the list of X and Y coordinates of the input polygon.

Next, the algorithm goes through all the input points, comparing them with the boundaries of the Minimum Bounding Rectangle extent and writes result to the dictionary of the points, those points are marked as ´outside´.

The code block is provided below (data input code block, if necessary, is provided in Task 10 part).

Code Block 1.1. – Code block for the Minimum Bounding Rectangle algorithm.

```
# Realization of a Minimum Bounding Rectangle Algorithm
def mbr(self):
    # defining minimums and maximums according to lists of X and
Y coordinates of a polygon vertexes
    self.x_max = max(self.x_pol_list)
    self.x_min = min(self.x_pol_list)
    self.y_max = max(self.y_pol_list)
    self.y_min = min(self.y_pol_list)

    # Iterating through points to exclude those lying outside
the extent of a Minimum Bounding Rectangle
    for id in self.dict_pt.keys():
        x_pt = self.dict_pt[id]['coordinates'][0]
        y_pt = self.dict_pt[id]['coordinates'][1]
        if x_pt < self.x_min or x_pt > self.x_max or y_pt <
self.y_min or y_pt > self.y_max:
            self.dict_pt[id]['category'] = 'outside'
```

Task 2. The RCA Algorithm

The implementation of Ray Casting Algorithm consists of 3 methods:

- Boundary - algorithm for finding points belonging to the boundaries of the polygon;
- Intersect - Implementation of the Ray Casting Algorithm with regarding of special cases;
- Classify - Launching classification for each point in a dictionary.

It is important to mention, that the regarding of special cases is performed inside each of the methods, so there is no separate method in the prepared code for highlighting only special cases. The use of the return operator for counting is also found in both methods: Boundary and Intersect.

All these 3 methods operate with a list of points sp_points[] to record the results of special cases analysis.

The final classification is launched by the Classify method, which processes only points that are not lying on the boundary and are not outside the minimum bounding rectangle. A parity counter is also built into this method.

The code blocks with all the appropriate comments are provided below.

Code Block 2.1. – Code block for the Boundary method.

```
# Implementation of the algorithm for finding points belonging
to the boundaries of the polygon
def boundary(self, x, y, id_pt=1):
    for id_pol in self.dict_pol.keys():
        # Exception for the closure point
        if id_pol == len(self.dict_pol.keys()):
            break
        # Setting polygon segments
        segment_1 = [self.dict_pol[id_pol], self.dict_pol[id_pol
+ 1]]
        point = [x, y]
        x1 = segment_1[0][0]
        x2 = segment_1[1][0]
        y1 = segment_1[0][1]
        y2 = segment_1[1][1]
        # Processing a case when the denominator is equal to
zero, comparison of Y values
        if x1 == x2 and point[0] != x1:
            continue
        if x1 == x2 and point[0] == x1:
            if y1 <= y2:
                if y1 <= point[1] <= y2:
                    self.dict_pt[id_pt]['category'] = 'boundary'
                    continue
                else:
                    continue
            elif y1 >= y2:
                if y2 <= point[1] <= y1:
                    self.dict_pt[id_pt]['category'] = 'boundary'
                    continue
                else:
                    continue
            else:
                continue
        # Finding line coefficients
        a = (y1 - y2) / (x1 - x2)
        b = y1 - ((y1 - y2) / (x1 - x2)) * x1
        # Calculation of whether a point belongs to the border
segments, recording the result
```

```
        if point[1] == a * point[0] + b:
            if x1 <= x2 and y1 <= y2:
                if x1 <= point[0] <= x2 and y1 <= point[1] <=
y2:
                    self.dict_pt[id_pt]['category'] = 'boundary'
            elif x1 >= x2 and y1 <= y2:
                if x2 <= point[0] <= x1 and y1 <= point[1] <=
y2:
                    self.dict_pt[id_pt]['category'] = 'boundary'
            elif x1 <= x2 and y1 >= y2:
                if x1 <= point[0] <= x2 and y2 <= point[1] <=
y1:
                    self.dict_pt[id_pt]['category'] = 'boundary'
            elif x1 >= x2 and y1 >= y2:
                if x2 <= point[0] <= x1 and y2 <= point[1] <=
y1:
                    self.dict_pt[id_pt]['category'] = 'boundary'
    return
```

Code Block 2.1. – Code block for the Intersect method.

```
# Implementation of the Ray Casting Algorithm with regarding of
special cases
def intersect(self, segment_1, segment_2):
    # Setting rays from points
    x1 = segment_1[0][0]
    x2 = segment_1[1][0]
    y1 = segment_1[0][1]
    y2 = segment_1[1][1]
    # Finding line coefficients
    a = (y1 - y2) / (x1 - x2)
    b = y1 - ((y1 - y2) / (x1 - x2)) * x1
    # Setting polygon segments
    x3 = segment_2[0][0]
    x4 = segment_2[1][0]
    y3 = segment_2[0][1]
    y4 = segment_2[1][1]
    # Finding the intersection point for the case when the
denominator is equal to zero
    if x3 == x4:
        x_int = x3
        y_int = a * x3 + b
    # Finding the intersection point for other cases
    else:
        c = (y3 - y4) / (x3 - x4)
        d = y3 - ((y3 - y4) / (x3 - x4)) * x3
        # Considering the case when the ray runs parallel to the
segment of the polygon and not coincides with it
        if c == 0 and y1 != y3:
            return 0
```

```
        # Considering the case when the ray runs parallel to the
segment of the polygon and coincides with it
        if c == 0 and y1 == y3:
            # Foresight the case when coincident polygon vertex
has the first identifier
            id3 = self.pol_points.index([x3, y3])
            id4 = self.pol_points.index([x4, y4])
            prev_pt = self.pol_points[id3 - 1]
            next_pt = self.pol_points[id4 + 1]
            if id3 == 0:
                prev_pt = self.pol_points[-2]
            elif id3 == len(self.pol_points) - 1:
                next_pt = self.pol_points[1]
            # Both coincident vertexes are written to the list
of exclusion points to avoid reconsideration
            if x3 > x1:
                self.sp_points.append([x3, y3])
                self.sp_points.append([x4, y4])
                if (prev_pt[1] >= y1 and next_pt[1] >= y1) or
(prev_pt[1] <= y1 and next_pt[1] <= y1):
                    return 0
                else:
                    return 1
            else:
                return 0
        # Finding the intersection point
        x_int = (d - b) / (a - c)
        y_int = a * ((d - b) / (a - c)) + b
    # Considering the case when an intersection point is not
contained in the list of polygon vertexes
    # Standard counting case
    if not [x_int, y_int] in self.pol_points:
        if x3 <= x4 and y3 <= y4:
            if x3 <= x_int <= x4 and y3 <= y_int <= y4 and x1 <=
x_int <= x2:
                return 1
            else:
                return 0
        elif x3 >= x4 and y3 <= y4:
            if x4 <= x_int <= x3 and y3 <= y_int <= y4 and x1 <=
x_int <= x2:
                return 1
            else:
                return 0
        elif x3 <= x4 and y3 >= y4:
            if x3 <= x_int <= x4 and y4 <= y_int <= y3 and x1 <=
x_int <= x2:
                return 1
            else:
                return 0
```

```
        elif x3 >= x4 and y3 >= y4:
            if x4 <= x_int <= x3 and y4 <= y_int <= y3 and x1 <=
x_int <= x2:
                return 1
            else:
                return 0
        else:
            return 0
    # Considering the case when an intersection point is
contained in the list of polygon vertexes
    # Counting with regarding of exceptions
    else:
        if [x_int, y_int] != [x3, y3] and [x_int, y_int] != [x4,
y4]:
            return 0
        if [x_int, y_int] in self.sp_points:
            return 0
        sp_id = self.pol_points.index([x_int, y_int])
        sp_bef = self.pol_points[sp_id - 1]
        sp_af = self.pol_points[sp_id + 1]
        # Checking which side of the ray both polygon segments
lie on
        if x1 <= x_int <= x2:
            if (sp_bef[1] >= y_int and sp_af[1] >= y_int) or
(sp_bef[1] <= y_int and sp_af[1] <= y_int):
                self.sp_points.append([x_int, y_int])
                return 0
            else:
                self.sp_points.append([x_int, y_int])
                return 1
        else:
            return 0
```

Code Block 2.3. – Code block for the Classify method.

```
# Launching classification for each point in a dictionary
def classify(self):
    for id_pt in self.dict_pt.keys():
        # Calling a boundary method
        self.boundary(self.dict_pt[id_pt]['coordinates'][0],
self.dict_pt[id_pt]['coordinates'][1], id_pt)
        count = 0
        self.sp_points = []
        # Skipping points, which were already classified
        if self.dict_pt[id_pt]['category'] == 'outside' or
self.dict_pt[id_pt]['category'] == 'boundary':
            continue
        # Setting rays from points
        segment_1 = [self.dict_pt[id_pt]['coordinates'],
[self.x_max + 1, self.dict_pt[id_pt]['coordinates'][1]]]
```

```
        for id_pol in self.dict_pol.keys():
            # Exception for the closure point
            if id_pol == len(self.dict_pol.keys()):
                break
            # Setting polygon segments
            segment_2 = [self.dict_pol[id_pol],
self.dict_pol[id_pol + 1]]
            # Setting counter and performing an evenness and odd
check (parity check)
            count += self.intersect(segment_1, segment_2)
            if count % 2 == 0:
                self.dict_pt[id_pt]['category'] = 'outside'
            else:
                self.dict_pt[id_pt]['category'] = 'inside'
```

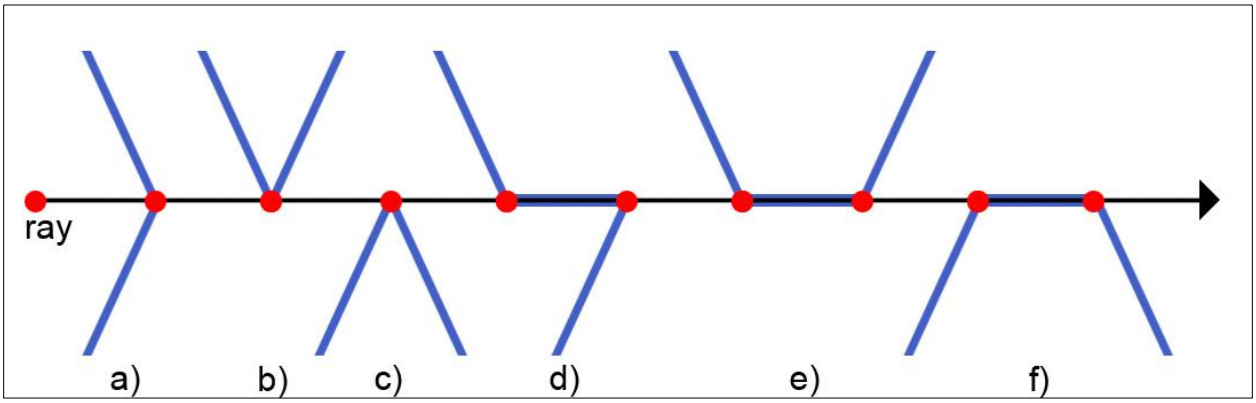
Task 3. The Categorisation of Special Cases

During the assignment all the occurring and expected Special cases were solved. The explanation of their solvation is provided in Task 2 code block. The algorithm regards and solves the follows special cases:

- Boundary case (point lies in a boundary);
- Ray intersect a polygon vertex;
- Ray coincident with a polygon segment;
- Coincident vertex is a first in a list of polygon vertexes.

The parity switching rules are illustrated in the Figure 3.1. Figure 3.2. illustrates polygons that were additionally created and tested during the development of the algorithm.

Figure 3.1. Parity switching rules for special cases: a) – switches ones, b) – without switching, c) – switches twice, d) – switches ones, e) – without switching, f) – switches twice



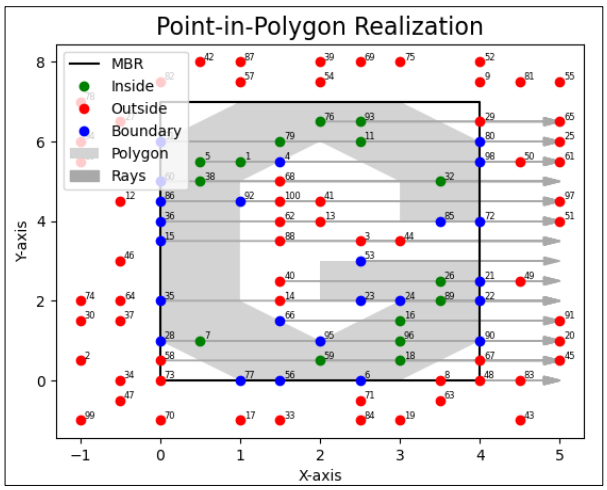


Figure 3.2. a) Basic Input case

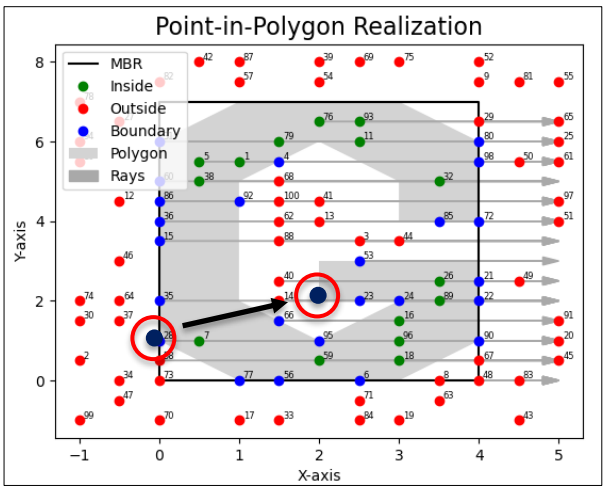


Figure 3.2. b) Testing on modified input: first point of a polygon shifted to a special case with a coincident segment

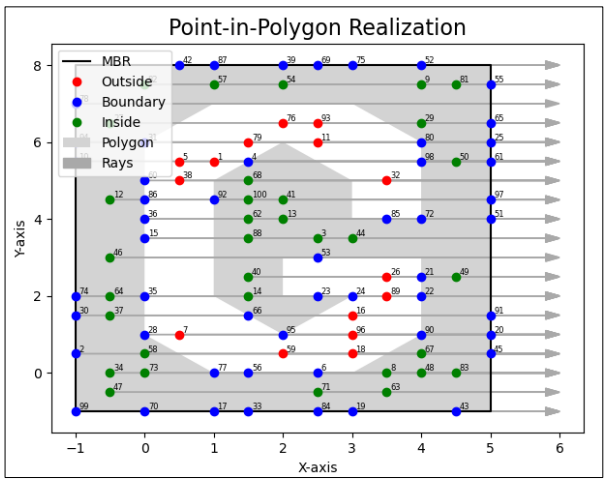


Figure 3.2. c) Testing on inverted polygon (with simulated hole)

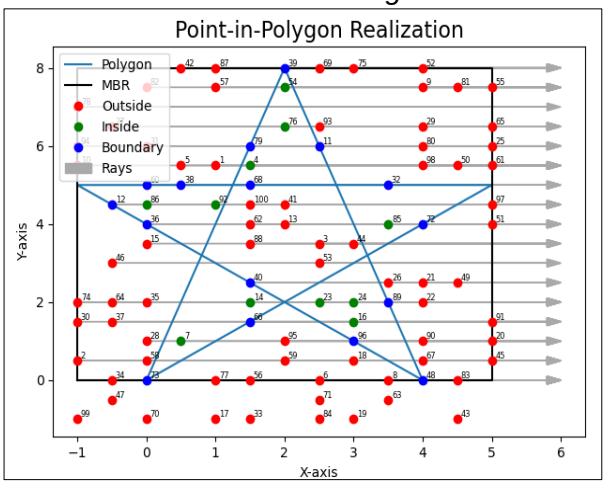


Figure 3.2. d) Testing on self-intersecting twisted polygon (classic twisted star)

Task 5. Object-Oriented Programming

For implementing an Object-Oriented Programming I defined a class, called Algorithm, with 15 global attributes, their list and explanation is provided in a code block below.

Code Block 5.1. – Attributes of a class, called Algorithm.

```
class Algorithm:
    # Defining global attributes
    def __init__(self):
        # List of exclusion points for classifying special cases
        self.sp_points = []
        # Dictionary of polygon vertexes, lists to operate with
polygon vertexes
        self.dict_pol = {}
        self.x_pol_list = []
```

```
        self.y_pol_list = []
        self.id_pol_list = []
        self.pol_points = []
        # Dictionary of input points, lists to operate with
input points
        self.dict_pt = {}
        self.x_pt_list = []
        self.y_pt_list = []
        self.id_pt_list = []
        # Spatial extent of a Minimum Bounding Rectangle
        self.x_max = 0
        self.x_min = 0
        self.y_max = 0
        self.y_min = 0
        # Dictionary for program output
        self.output_pt = {}
```

Within a class, called Algorithm, I defined 8 methods, which my code accesses, the main function also accesses them. These 8 methods are responsible for the main steps of the Point-in-Polygon Algorithm implementation (4 methods), and also they are responsible for supportive procedures: downloading the input data, writing the output data, creating a plot (also 4 methods). The methods and their attributes are given in the code block below.

Code Block 5.2. – Objects of a class, called Algorithm.

```
# Reading a csv with input polygon vertexes coordinates
def read_poly(self, in_poly):
# Reading a csv with input point coordinates
def read_poi(self, in_poi):
# Realization of a Minimum Bounding Rectangle Algorithm
def mbr(self):
# Implementation of the algorithm for finding points belonging
to the boundaries of the polygon
def boundary(self, x, y, id_pt=1):
# Implementation of the Ray Casting Algorithm with regarding of
special cases
def intersect(self, segment_1, segment_2):
# Launching classification for each point in a dictionary
def classify(self):
# Creating an output csv
def write_output(self, out_path):
# Visualization of all objects on the graph
def plot(self):
```

The main function of a main_from_file.py have got 3 parameters: input points, input polygons and output file. With these 3 parameters main function accesses the specified objects as shown in the code block below.

Code Block 5.3. – Referring to methods within a main function.

```
def main(in_poly='polygon.csv', in_poi='input.csv',
out_path='output.csv'):
    pip = Algorithm()

    # Referring to methods
    try:
        pip.read_poly(in_poly)
        pip.read_poi(in_poi)
        pip.mbr()
        pip.classify()
        pip.write_output(out_path)
        pip.plot()
    except Exception:
        return
```

Task 6. On the Use of Git and GitHub

For project implementation I used a GitHub repository, provided for a work via a link:

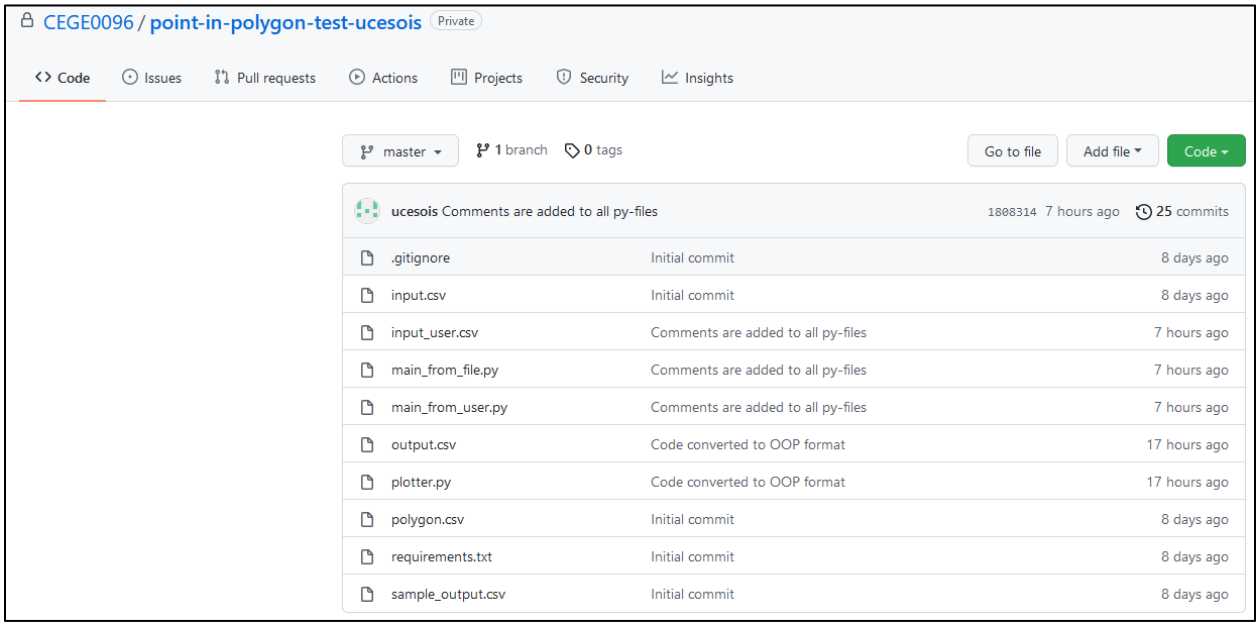
https://github.com/CEGE0096/point-in-polygon-test-ucesois

At the beginning of my work on a project, I activate a git clone function via Anaconda Prompt to create a copy of a project on my local computer.

After that I worked with a project with PyCharm. I added a remote access to a provided repository. To save my work and to prevent it from unexpected changes or crush – I made regular commits with the usage of Git / Push and Git / Commit commands in a GUI of PyCharm. My initial commit was made on 15.11.2021. Git log of my project, created via Anaconda Prompt, is provided in a current report, in a Git Log section.

Mi final .zip archive for submission was created with the usage of GitHub. State of project in GitHub repository (for 22.11.2021) is shown on Figure 6.1.

Figure 6.1. – State of project in GitHub repository (for 22.11.2021).



Task 9. Plotting

I used Plotting class during the project development as an assistant for finding errors in classification as an express tool for output data visualization.

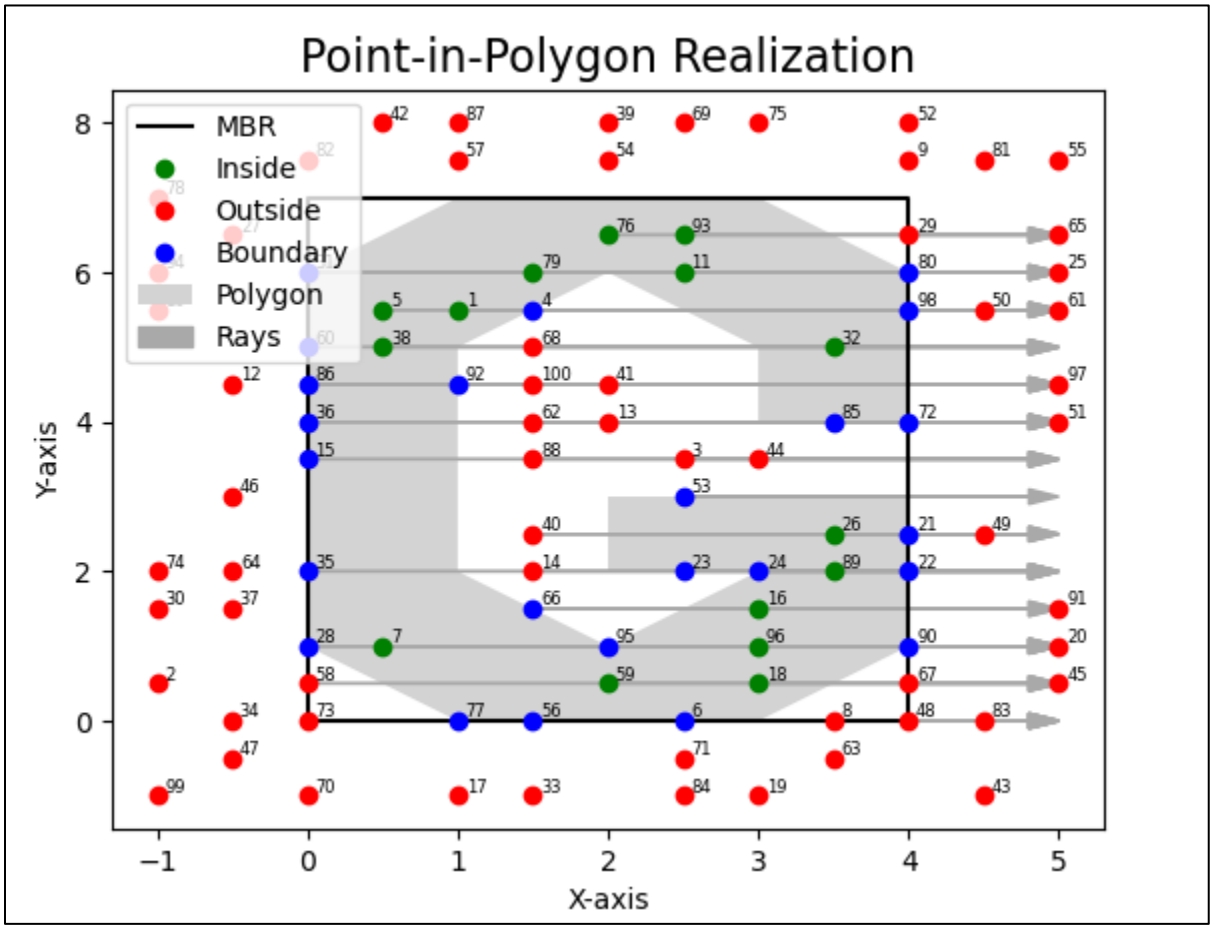
My main function in both main_from_file.py and main_from_user.py access plotting class and visualize the results of an Algorithm.

I additionally added the following in Plotting:

- label the attributes of the point identifiers;
- added the visualization of the Minimum Bounding Rectangle;
- added the visualization of the rays (for points inside the Minimum Bounding Rectangle);
- modified object visualization styles and graph labels (heading and axes).

My Plotting result for predefined points and polygon is shown on Figure 9.1. Code block for plotting with comments is provided below.

Figure 9.1. – Plotting result for predefined points and polygon.



Code Block 9.1. – Modified Plotter class.

```
class Plotter:

    def __init__(self):
        plt.figure()

    # Creating a method for displaying a polygon, configuring
display
    def add_polygon(self, xs, ys):
        plt.fill(xs, ys, 'lightgray', label='Polygon')

    # Creating a method for displaying a Minimum Bounding
Rectangle, configuring display
    def add_mbr(self, x_max, x_min, y_max, y_min):
        plt.plot([x_min, x_min, x_max, x_max, x_min], [y_min,
y_max, y_max, y_min, y_min], 'black', label='MBR')

    # Creating a method for displaying Rays (used in Ray
Casting) as arrows, configuring display
    def add_arrow(self, x, y, dx, dy):
```

```
        plt.arrow(x, y, dx, dy, color='darkgray',
linewidth=None, head_width=0.2, head_length=0.2,
                  length_includes_head=True, label='Rays')

    # Creating a method for displaying points (both in
classified and unclassified state), configuring display
    def add_point(self, x, y, kind=None, annotation=None):
        plt.annotate(annotation, (x, y), fontsize=6, xytext=(x +
0.05, y + 0.05))
        if kind == 'outside':
            plt.plot(x, y, 'ro', label='Outside')
        elif kind == 'boundary':
            plt.plot(x, y, 'bo', label='Boundary')
        elif kind == 'inside':
            plt.plot(x, y, 'go', label='Inside')
        else:
            plt.plot(x, y, 'ko', label='Unclassified')

    # General visualization settings, adding axis caption and
heading
    def show(self):
        handles, labels = plt.gca().get_legend_handles_labels()
        by_label = OrderedDict(zip(labels, handles))
        plt.legend(by_label.values(), by_label.keys())
        plt.title('Point-in-Polygon Realization', fontsize=16)
        plt.xlabel('X-axis')
        plt.ylabel('Y-axis')
        plt.show()
```

Code Block 9.2. – Modified Plotter class calling.

```
def plot(self):
    plotter = Plotter()
    # Plotting of a testing polygon
    plotter.add_polygon(self.x_pol_list, self.y_pol_list)
    # Plotting of a Minimum Bounding Rectangle
    plotter.add_mbr(self.x_max, self.x_min, self.y_max,
self.y_min)
    for p in range(len(self.x_pt_list)):
        # Plotting of a Rays, build during the Ray Casing
Algorithm implementation as arrows
        if self.x_min <= self.x_pt_list[p] <= self.x_max and
self.y_min <= self.y_pt_list[p] <= self.y_max:
            plotter.add_arrow(self.x_pt_list[p],
self.y_pt_list[p], self.x_max + 1 - self.x_pt_list[p], 0)
        # Plotting of classified points with their ID labels
        plotter.add_point(self.x_pt_list[p], self.y_pt_list[p],
self.output_pt[p + 1], p + 1)
    plotter.show()
```

Task 10. Error Handling

I implemented the error handling functionality for the methods of uploading input files in csv format (both for input polygon vertexes list and input points), as well as for generating an output file with Algorithm results in csv format.

I implemented the error handling functionality with the usage of Try / Except construction, also with the usage of If / Return construction.

Parts of the code with comments, where error handling is implemented, are provided below.

Code Block 10.1. – Error handling for input of polygon vertexes in csv.

```
# Reading a csv with input polygon vertexes coordinates
def read_poly(self, in_poly):
    # For error handling a try / except construction is
implemented
    try:
        # File opening for reading
        with open(in_poly, 'r') as pol:
            next(pol)
            # Formatting of input data, adding of formatted data
into dictionary and lists
            for input_pol in pol:
                input_pol = input_pol.replace('\n', '')
                input_pol = input_pol.split(',')
                id_pol = int(input_pol[0])
                x_pol = float(input_pol[1])
                y_pol = float(input_pol[2])
                self.dict_pol[id_pol] = [x_pol, y_pol]
                self.x_pol_list.append(x_pol)
                self.y_pol_list.append(y_pol)
                self.id_pol_list.append(id_pol)
                self.pol_points.append([x_pol, y_pol])
    # Predefined file reading errors
    except OSError:
        print('Could not open or read file: ' + in_poly)
        return
    except ValueError:
        print('Input polygon data is not in appropriate
format!')
        return
    except SyntaxError or NameError or PermissionError:
        print('Provide a correct path to a file!')
        return
    # Checking the object is a polygon (at least a triangle)
    if len(self.x_pol_list) < 4:
        print('This is not a polygon!')
        return
    # Checking the closure condition
```

```
    if self.pol_points[0] != self.pol_points[-1]:
        print('This is not a polygon (close the polygon)!')
        return
    # Checking the unique identifiers of points
    if len(self.id_pol_list) != len(set(self.id_pol_list)):
        print('IDs of points have to be unique!')
        return
```

Code Block 10.2. – Error handling for input of points in csv.

```
# Reading a csv with input point coordinates
def read_poi(self, in_poi):
    # For error handling a try / except construction is
implemented
    try:
        # File opening for reading
        with open(in_poi, 'r') as pt:
            next(pt)
            # Formatting of input data, adding of formatted data
into dictionary and lists
            for input_pt in pt:
                input_pt = input_pt.replace('\n', '')
                input_pt = input_pt.split(',')
                id_pt = int(input_pt[0])
                x_pt = float(input_pt[1])
                y_pt = float(input_pt[2])
                self.dict_pt[id_pt] = {'coordinates': [x_pt,
y_pt], 'category': ''}
                self.x_pt_list.append(x_pt)
                self.y_pt_list.append(y_pt)
                self.id_pt_list.append(id_pt)
    # Predefined file reading errors
    except OSError:
        print('Could not open or read file: ' + in_poi)
        return
    except ValueError:
        print('Input point data is not in appropriate format!')
        return
    except SyntaxError or NameError or PermissionError:
        print('Provide a correct path to a file!')
        return
    # Checking the presence of at least one point
    if len(self.x_pt_list) < 1:
        print('File is empty!')
        return
    # Checking the unique identifiers of points
    if len(self.id_pt_list) != len(set(self.id_pt_list)):
        print('IDs of input points have to be unique!')
        return
```

Code Block 10.3. – Error handling for output in csv.

```
# For error handling a try / except construction is implemented
try:
    # Writing an output csv file from a dictionary
    with open(out_path, 'w') as out:
        # Formatting the headings
        out.write('id,category\n')
        print('Results of Point-in-Polygon Algorithm:')
        # Formatting the output data and providing a print of
results for input points
        for id in self.output_pt.keys():
            out.write(str(id) + ',' + self.output_pt[id] + '\n')
            print('Point ' + str(id) + ' is ' +
self.output_pt[id])
# Error handling to check a provided path to a file
except SyntaxError or NameError or PermissionError:
    print('Provide a correct path to a file!')
    return
```

Task 11. Additional Features

Small additional feature was added to a developed program. The user was provided with the ability to enter points for testing in two different ways:

1. the User can set the path to the input csv file with points for testing;
2. the User can independently enter the dots from the console.

In order to choose the input method for a User, a dialog box with a built-in error handling function is provided in main_from_user.py file. To facilitate the code, main_from_user.py operates with the variables of the main function of the main_from_file.py. Part of the code with the User options is provided below.

Code Block 11.1. – User options and main_from_user.py code.

```
# Defining main function
def main():
    # Dialogue with a User for choosing point entering from csv
file or from a keyboard (as a pair of coordinates)
    answer = 0
    while answer != str(1) and answer != str(2):
        answer = str(input('How do you want to provide points?
Type 1 for csv / Type 2 for keyboard input: '))
        if answer == str(1):
            in_poi = input('Insert a path to points file: ')
        elif answer == str(2):
            x_user = input('Insert X coordinate: ')
            y_user = input('Insert Y coordinate: ')
            with open('input_user.csv', 'w') as out:
```

```
                out.write('id,x,y\n')
                out.write('1,' + str(x_user) + ',' +
str(y_user))
                in_poi = 'input_user.csv'
        else:
            print('Provide an answer as 1 / 2! ')

    # Entering the path to the csv file with polygon vertexes
    in_poly = input('Insert a path to polygon file: ')

    # Entering the path to the output algorithm results
    out_path = input('Insert where to write the output: ')

    # Launching the main algorithm
    algorithm(in_poly=in_poly, in_poi=in_poi, out_path=out_path)


if __name__ == '__main__':
    main()
```

Git Log

Git log for my project is provided below. Commits start with the most relevant by date and drop down to the least relevant.

(geospatial) D:\NINA\UCL\CEGE0096\Assignment_1\UCESOIS>git log
commit 180831460777fcd1920278f952cbfba84482ba0e (HEAD -> master, origin/master,
commit 180831460777fcd1920278f952cbfba84482ba0e (HEAD -> master, origin/master, origin/HEAD)
Author: ucesois <ucesois@ucl.ac.uk>
Date: Mon Nov 22 13:37:54 2021 +0000

    Comments are added to all py-files

commit 8a883e10b3444ba22544c5315e07aeea52e77a4e
Author: ucesois <ucesois@ucl.ac.uk>
Date: Mon Nov 22 08:08:00 2021 +0000

    Mistake in user input file is corrected

commit 114ef2b306c45690f8aace05d7f1844335f0c6d2
Author: ucesois <ucesois@ucl.ac.uk>
Date: Mon Nov 22 06:17:31 2021 +0000

    Mistakes in user input file are corrected

commit 3e043e2e8ef4e6125b00c383c9c702f70ba0a08f
Author: ucesois <ucesois@ucl.ac.uk>

Date: Mon Nov 22 04:09:39 2021 +0000

    For user 2 possibilities of entering points are added

commit 64c9d1b96caa5dabbb77d27941410eb25d433ac0
Author: ucesois <ucesois@ucl.ac.uk>
Date: Mon Nov 22 04:03:32 2021 +0000

    Code converted to OOP format

commit 33b72a0dfec2e139847402da90c36ccce62ca6c0
Author: ucesois <ucesois@ucl.ac.uk>
Date: Sun Nov 21 18:16:18 2021 +0000

    Error handling for output is added

commit a6b7dd84bdb4a4921670982247c9c3a56987f130
Author: ucesois <ucesois@ucl.ac.uk>
Date: Sun Nov 21 01:58:08 2021 +0000

    Output file with corrected name is added

commit d5f3c22f49fbce0668c351d55e89adeea6d54582
Author: ucesois <ucesois@ucl.ac.uk>
Date: Sun Nov 21 01:56:26 2021 +0000

    Lines of code for algorithm checking and debugging are deleted

commit 698cd9cb231acc3b76c939fff18e511398cec560
Author: ucesois <ucesois@ucl.ac.uk>
Date: Sat Nov 20 19:34:20 2021 +0000

    Error handling is provided. First version of main_from_user is provided

commit ef4480733b26ad2aa242b71bc320d9448f049d61
Author: ucesois <ucesois@ucl.ac.uk>
Date: Sat Nov 20 06:00:53 2021 +0000

    Equation entries are simplified. Link for testing of alternative polygons is provided

commit 17e061cb79b0ce2de80e34c6090d54d49010e578
Author: ucesois <ucesois@ucl.ac.uk>
Date: Sat Nov 20 03:25:40 2021 +0000

    Plotting of points, polygons, and rays with appropriate axis labels and annotations is complete

commit ac0be37db0fa684bd560baf2f067a7d689939c0f
Author: ucesois <ucesois@ucl.ac.uk>
Date: Sat Nov 20 00:01:09 2021 +0000

    output_from_user.csv is added

commit ccb06b8e092ed62c05f21e46a9b170b3d90b0380
Author: ucesois <ucesois@ucl.ac.uk>

Date: Fri Nov 19 23:59:29 2021 +0000

    Creation of output csv is added

commit b2f4457a01fc37a2156b12547246ae2083041a3a
Author: ucesois <ucesois@ucl.ac.uk>
Date: Fri Nov 19 21:52:28 2021 +0000

    Code placed into a main function. Error for console launch is fixed

commit 50446f6a5b3e3b39104b60e43ec065bca95668fe
Author: ucesois <ucesois@ucl.ac.uk>
Date: Fri Nov 19 05:06:10 2021 +0000

    Special cases are solved

commit 277d4f216ffc94b1bf9a92d19b83609530dc8964
Author: ucesois <ucesois@ucl.ac.uk>
Date: Fri Nov 19 02:39:42 2021 +0000

    Mistake with parallel lines is corrected

commit d4204c426b54cf8e1678c07d4f89e853e99f6c0e
Author: ucesois <ucesois@ucl.ac.uk>
Date: Fri Nov 19 02:29:17 2021 +0000

    Intersection function is corrected, Algorithm for boundary points is added

commit c24f779f1f826f8d6ad986e45857da88b2fffd9f
Author: ucesois <ucesois@ucl.ac.uk>
Date: Wed Nov 17 22:07:05 2021 +0000

    Cosmetic changes

commit 7b9652a215c8700c7dc49b3c43900967161d9edb
Author: ucesois <ucesois@ucl.ac.uk>
Date: Wed Nov 17 21:59:55 2021 +0000

    Intersection function is defined

commit 7293394cf9fce2f30751bb4f78796777bcb20add
Author: ucesois <ucesois@ucl.ac.uk>
Date: Wed Nov 17 20:25:58 2021 +0000

    Minimum bounding rectangle algorithm is implemented

commit 25b000c6d2496dd408e5879ba3fc62b6d29f5c82
Author: ucesois <ucesois@ucl.ac.uk>
Date: Wed Nov 17 02:31:43 2021 +0000

    Input objects plottered, Min-max x and y for polygon is found

commit e2cf93cb33a2a45970adcf0f797e8c1b40d1781c
Author: ucesois <ucesois@ucl.ac.uk>

Date: Wed Nov 17 02:15:43 2021 +0000

    Reading of a list of x, y coordinates (for points and polygon csv) is completed

commit ac56366f350d8155ba26a74865fd732cfdf253ce
Author: ucesois <ucesois@ucl.ac.uk>
Date: Tue Nov 16 20:45:51 2021 +0000

    A description of the general sequence of actions has been added in the comments

commit b00b5efd842d487b3ce5eed07bd63c5b06e4640f
Author: ucesois <ucesois@ucl.ac.uk>
Date: Mon Nov 15 12:20:04 2021 +0000

    Update for testing commits in Git (several comments are added for testing)

commit a4cacfb532bed21fec699ea0768a94d75fd0c46b
ub.com>
Date: Mon Nov 15 02:22:34 2021 +0000

    Initial commit