Point-in-Polygon Test

Report for the CEGE0096: 1st Assignment

Anastasia Kopytina
Student Number: 16005485
anastasia.kopytina.20@ucl.ac.uk
https://github.com/ana-kop/assignment-1

Introduction

The software developed as part of this assignment allows user to classify the location of specified point(s) of interest (POIs) with regard to a specified polygon. Two versions of the program were developed: main_from_file.py takes POIs for testing from a file, and main_from_user.py takes a POI for testing from user input. In both cases, the program returns the location of the POI(s) as on the 'inside' , 'outside' or 'boundary' of the polygon, and then plots the polygon and the POI(s) (and also the rays from the Ray Casting Algorithm (RCA) if specified). I completed all parts of the assignment, and also did some additional testing of the program by making a new polygon and more input points (outlined in the Additional features section).

The limitations of the software are:
- the *contains()* method may not handle all special cases correctly on its own (without the *boundary()* method);
- both programmes only accept inputs from .csv files;
- the programs were not optimised for speed and take a long time to execute.

The programs were developed by writing some code from scratch using the lecture materials provided as part of the CEGE0096 module, and also by adapting some open-source code found in online communities - the links are included in the References section of this report.

It took around 1.5 weeks to develop a fully operational version of the code for both programs, and another week to debug and improve the code. The hardest parts of the programs to develop were:
- a function for getting inputs from .csv files without using any modules;
- an adaptation of the ray casting and boundary methods that locate all special cases correctly.

Project Description

The programs were developed using Python 3.8 in the Pycharm IDE. Both programs are presented as part of the same Pycharm project and can therefore be conveniently executed in this IDE. However, both programs can also be used from the command line.

Both programs require import of the OrderedDict from the collections module as well as matplotlib library for the Plotter class, and the sys module for the Polygon class. The imports are already specified in the code.

Both programs can be run by executing all of the program code as it is in Pycharm, as long as:
- the input file for polygon (and for the POIs in case of program main_from_file.py) remains in the same directory as the program file;
- the names of the input files remain unchanged ('polygon.csv' and 'input.csv').

If either the file path or the file name have been changed, then they need to be updated in the *main()* function in the last few lines of the program. Furthermore, in case of the main_from_file.py, user can specify whether they want to display a plot of results, and whether this plot should include the RCA rays. This can be specified by editing the variables *display_result* and *display_result_with_rays* in the *main()* function (by default the *display_result* is True, and *display_result_with_rays* is False).

```
if __name__ == "__main__":
    polygon_file = "polygon.csv"
    input_file = "input.csv"
    main(polygon_file, input_file, display_result=True,
display_result_with_rays=False)
```

The output file for program main_from_file.py will also be created in the same directory where the program file main_from_file.py is located.

For program main_from_user.py, after it is executed, the user has to indicate that they would like to locate a POI by inputting ‘yes’, and then input x-coordinate and y-coordinate of the POI when prompted. The program will then test the POI and return result as well as a plot.

```
/Users/ak/opt/anaconda3/envs/geospatial/bin/python /Users/ak/Desktop/Project/
main_from_user.py
--Read polygon.csv
--Insert point information
Would you like to check location of a point? Please enter yes or no: yes
Please enter the X-coordinate of your point: 1.5
Please enter the Y-coordinate of your point: 2.0
--Categorize point
Your point is on the outside of the polygon.
--Plot polygon and points
IMPORTANT: Please close the plot window to continue
Would you like to check location of another point? Please enter yes or no: no

Process finished with exit code 0
```

The user can keep testing more POIs by first closing the plot of the previous POI and then typing ‘yes’ when asked whether they would like to locate another point. If the user inputs ‘no’ as the answer to this question, the program will terminate and would have to be restarted if user would like to test another POI.

Software

Task 1. The MBR Algorithm

The Minimum Binding Rectangle (MBR) algorithm was written from scratch. Firstly, the minimum and the maximum of the x-coordinates of the polygon vertices are found, as well as the minimum and the maximum of the y-coordinates of the polygon vertices. This is done in the *main()* function, outside of the Polygon class.

```
polygon_xs = polygon.x_vertices()
polygon_ys = polygon.y_vertices()

min_x = min(polygon_xs)
max_x = max(polygon_xs)
min_y = min(polygon_ys)
max_y = max(polygon_ys)
```

The resulting 4 points then make up the Minimum Binding Rectangle:

- `(min_x, min_y)`
- `(min_x, max_y)`
- `(max_x, max_y)`
- `(max_x, min_y)`

The *mbr_contains()* method for the Polygon class takes a POI and the MBR vertices, and returns True if the point is inside the MBR. It does this by checking if the x-coordinate of the POI is between the minimum and maximum x-coordinates of the polygon, and the y-coordinate of the POI is between the minimum and maximum y-coordinates of the polygon.

```
def mbr_contains(self, point, min_x, max_x, min_y, max_y):
        is_inside_mbr = False
        if (min_x <= point.x <= max_x) and (min_y <= point.y <= max_y):
            is_inside_mbr = True
        return is_inside_mbr
```

Task 2. The RCA Algorithm

The RCA algorithm for this assignment was adapted from the Computer Geekery blog [1] and the rosettacode.org [3] .

The RCA algorithm for the programs is presented as a *contains()* method for the Polygon class, and it takes a point object as parameter. The method uses *sys.float_info.max* to act as infinity if division by 0 is involved. Variable *_eps* is used to represent a very small number (0.00001) for the case when the POI is at the same height as vertex. A variable *is_inside_polygon* is used with a boolean value False initially.

```
def contains(self, point):
       _huge = sys.float_info.max
       _eps = 0.00001
       is_inside_polygon = False
```

Next, a for-loop is initiated which will go through every edge of the polygon (a list of polygon edges is created by the *edge()* method of the Polygon class) and checks whether a horizontal ray drawn from the POI intersects this edge.

The end points of the edge are named A and B (displayed with small letter rather than capitalised in the code). Before checking whether the ray intersects the edge, it is important to make sure that point A is lower than point B.

```
for edge in self.edges:
    a, b = edge[0], edge[1]
    if a.y > b.y:
        a, b = b, a
```

Then the loop checks that POI does not have the same y-coordinate as on of the edge points. If it does, then the y-coordinate of POI is increased by a small number *_eps* = *0.00001*.

```
    if point.y == a.y or point.y == b.y:
       point.y += _eps
```

The overall logic of the for-loop goes as follows. We initially start on the outside of the polygon as the number of edges intersected is zero and hence the point is so far considered outside the polygon. The for-loop then starts by taking an edge from the list of edges and checking if this edge gets intersected by the ray drawn from the POI (this checking process is described in following paragraphs). If it is found that the ray does intersect the edge being considered then the variable *inside* is reset to True, as we now have an odd number of edges intersected (one edge).

The loop then restarts and checks the next edge. If it finds that the ray intersects this edge as well, the variable *inside* now gets reset back to False as we have an even number of edges intersected; the loop then starts again for the next edge. The loop then continues until all edges of the polygon have been checked. In the end, the method returns the value of *inside* that the for-loop ends up on after checking the last edge, and this value represents whether the POI is inside the polygon or outside.

The following checks are used to analyse whether the horizontal ray intersects a particular edge. Firstly, if the x-coordinate of POI is greater than the maximum x-coordinate of the edge, or if POI lies above point B or below point A, then the horizontal ray does not intersect the edge.

```
if point.y > b.y or point.y < a.y or point.x > max(a.x, b.x):
    continue
```

Then, if the x-coordinate of POI is smaller than the minimum x-coordinate of the edge, then the horizontal ray intersects the edge.

```
if point.x < min(a.x, b.x):
    is_inside_polygon = not is_inside_polygon
    continue
```

Then, if neither of the conditions above are satisfied, we find the slope of the edge and the slope of the line between POI and A. In both cases, if the calculation produces a zero division error, then edge is vertical and its slope is set equal to infinity for the purposes of this code.

```
try:
    slope_edge = (b.y - a.y) / (b.x - a.x)
except ZeroDivisionError:
    slope_edge = _huge

try:
    slope_point = (point.y - a.y) / (point.x - a.x)
except ZeroDivisionError:
    slope_point = _huge
```

Finally, if the slope of the line between POI and A is greater than or equal to slope of the edge then the horizontal ray intersects this edge.

```
if slope_point >= slope_edge:
    is_inside_polygon = not is_inside_polygon
    continue
```

Task 3. The Categorisation of Special Cases

One type of special case is the **boundary case**. For this case, a *boundary()* method is defined in the Polygon class. This method was adapted from an answer on stackoverflow.com [4].

The *boundary()* method takes a POI, and checks if the POI lies on the polygon boundary. A variable *on_boundary* is used with a boolean value False originally. A for-loop is initiated which will go through every edge of the polygon (a list of polygon edges is created by the *edge()* method of the Polygon class) and check whether the POI lies on this edge.

```
def boundary(self, point):
      on_boundary = False

      for edge in self.edges:
        a, b = edge[0], edge[1]
```

First, we calculate the distance between A and B, between A and POI, and between B and POI using the formula:

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

```
        distance_a_b = ((a.x - b.x) ** 2 + (a.y - b.y) ** 2) ** (1 / 2)
        distance_a_point = ((a.x - point.x) ** 2 + (a.y - point.y) ** 2) ** (1 / 2)
        distance_b_point = ((point.x - b.x) ** 2 + (point.y - b.y) ** 2) ** (1 / 2)
```

Then, we sum the distances between A and POI, and between B and POI. We also round the *sum_point_dist* and *distance_a_b* to 12 decimal places, since Python sometimes rounds the square roots inconsistently, which leads to misclassification of points.

```
        sum_point_dist = distance_a_point + distance_b_point
        distance_a_b = round(distance_a_b, 12)
        sum_point_dist = round(sum_point_dist, 12)
```

Then, we check whether these two distances are equal. If they are, then the POI must lie on that edge, and the method returns True.

```
        if sum_point_dist == distance_a_b:
            on_boundary = True
    return on_boundary
```

If the POI does not lie on the edge, then together, the POI and the points A and B must form a triangle, and the length of one side of the triangle (AB) is always smaller than the sum of the lengths of the other 2 sides (A-POI and B-POI), and this is why this method works.

Another case is when the y-coordinate of the POI is equal to the y-coordinate of one of the endpoints of the edge. This is dealt with by adding a small number, _eps = 0.0001, to the y-coordinate of the POI.

```
if point.y == a.y or point.y == b.y:
    point.y += _eps
```

In the case when a ray passes through a vertex, the *contains()* method may return inconsistent result, however, the *boundary()* method will always classify these points as on the boundary and therefore the final result will be correct.

Task 5. Object-Oriented Programming

Three classes are defined for this assignment: Plotter, Point and Polygon. Plotter class includes methods that allow drawing of polygon, points, rays and MBR.

A Point class object 3 requires attributes: point name (used for point id provided in the input files), x-coordinate, and y-coordinate. In the program main_from_user.py, the Point class does not have a name attribute. Point class also contains methods to get each of these 3 attributes.

A Polygon class object 1 requires attribute - a list of points in clockwise order. Apart from the *mbr_contains(), contains()* and *boundary()* methods outlined in previous sections, it includes the following methods:

- *edges():* returns a list of tuples, where each tuple contains 2 endpoints of a polygon edge
- *x_vertices(), y_vertices():* return a list of x- and y-coordinates of the polygon vertices respectively

The rest of the code in these programs makes use of these 3 classes and also defines a function *read_points_from_file(file_path, list_for_points)* for reading points from a .csv file, and the *main()* function.

Task 6. On the Use of Git and GitHub

A private GitHub repository named **assignment-1** was created. In order to enable Git for this project in Pycharm, the following steps have been taken, from the command line:

```
conda activate geospatial
cd desktop
cd assignment_1
git init
git add .
git remote add origin https://github.com/ana-kop/assignment-1
```

The last 2 steps uploaded all the files in the assignment_1 folder into the GitHub repository. In order to then make commits, Pycharm GUI was used.

There was a total of 24 commits, with each commit usually including one or two major changes to the code. The comments on the commits feature the date commit was made and what part of the code has been changed or added. (Please note the comments on the first few commits may not follow this logic).

Task 9. Plotting

The plotter class that was provided for this assignment is used to plot the results of the points classification in both programs. The original Plotter class was appended with an *add_mbr()* method which works very similarly to the *add_polygon()* method, but doesn't fill in the resulting polygon but instead plots its outline with a dashed blue line.

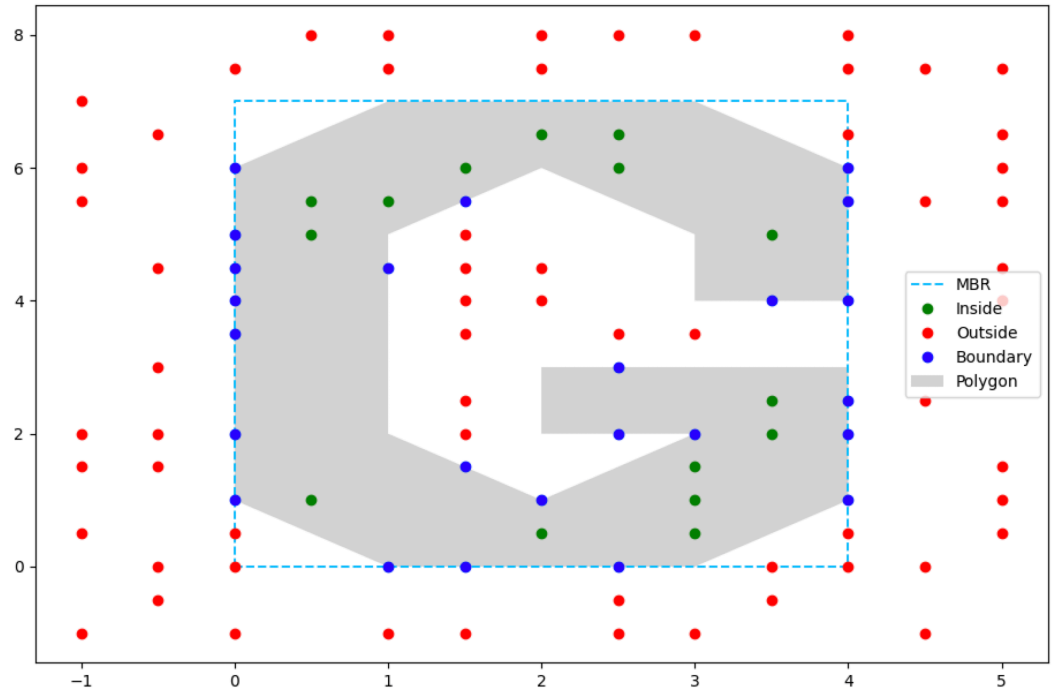
Furthermore, an *add_ray()* method was added, which plots a horizontal line from a POI.

```
def add_ray(self, x, y, max_x_input):
    xs = [x, max_x_input + 1]
    ys = [y, y]
    plt.plot(xs, ys, label="Ray", linewidth=0.4,  color="black")
```

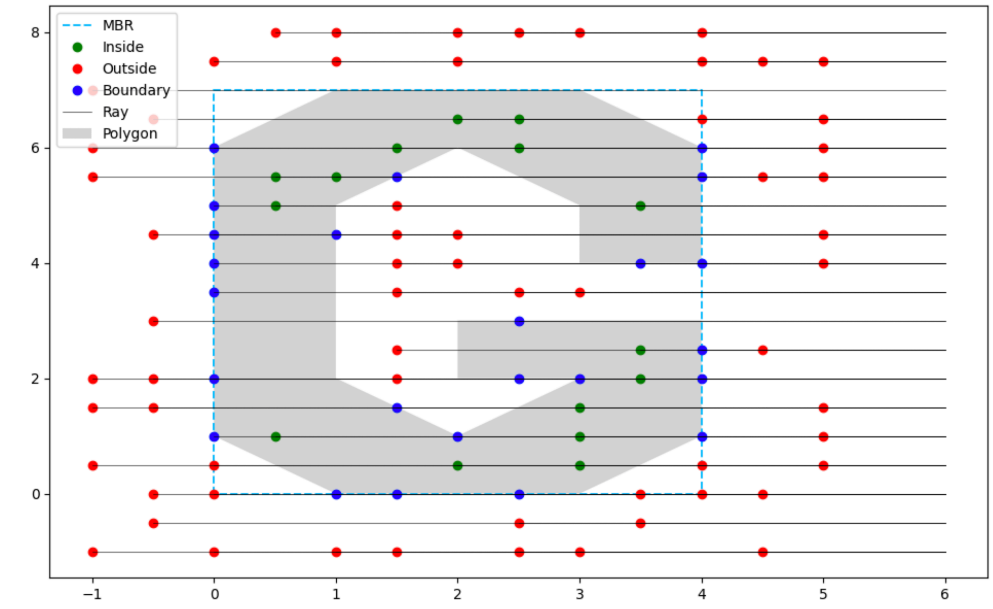
The user of the program main_from_file.py can decide whether they would like to plot the ray by changing the variable *display_result_with_rays* to True in the *main()* function. In case of main_from_user.py the ray is always added to the plot.

The final plots for the main_from_file.py program are presented below.

```
if __name__ == "__main__":
    polygon_file = "polygon.csv"
    input_file = "input.csv"
 main(polygon_file, input_file, display_result=True,
display_result_with_rays=False)
```

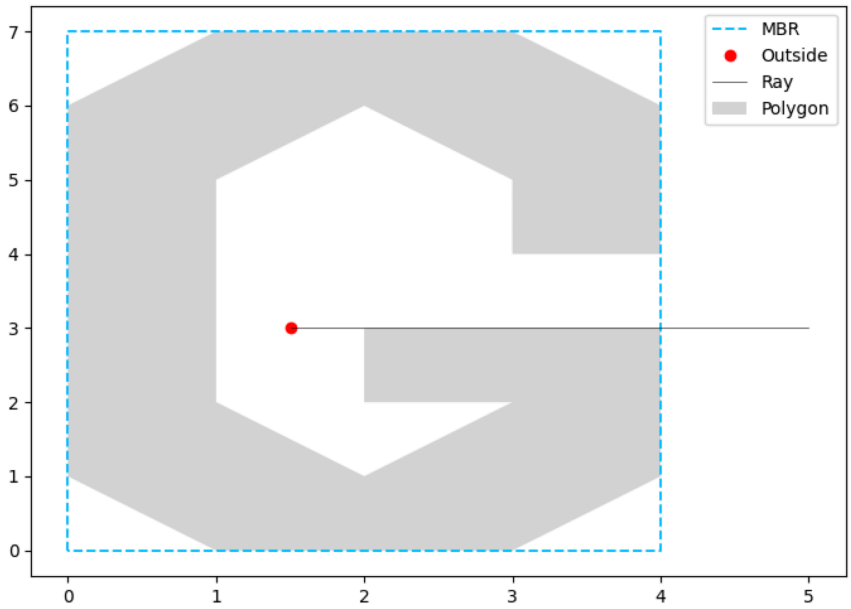


```
if __name__ == "__main__":
    polygon_file = "polygon.csv"
    input_file = "input.csv"
    main(polygon_file, input_file, display_result=True,
display_result_with_rays=True)
```



An example of a plot produced by the main_from_user.py program is presented below.

```
if __name__ == "__main__":
    polygon_file = "polygon.csv"
    input_file = "input.csv"
    main(polygon_file, input_file)
```



Task 10. Error Handling

The *contains()* methods in both programs includes conditions that allow the method to handle division by zero error when calculating the slope of a vertical edge.

```
except ZeroDivisionError:
    slope_edge = _huge
```

Further error handling functionality is included into the main_from_user.py program. This error handling functionality is needed because the program takes input from user, and the user may not input what is expected by the program.

If user types something other than ‘yes’ or ‘no’ when asked if they would like to check location of a point:

```
if user_decision != "yes" and user_decision != "no":
    user_decision = input("Your answer is not recognised. Would you like to check
location of a point? Please enter yes or no:  “)
```

If the user input for the coordinates of the point is something other than a number:

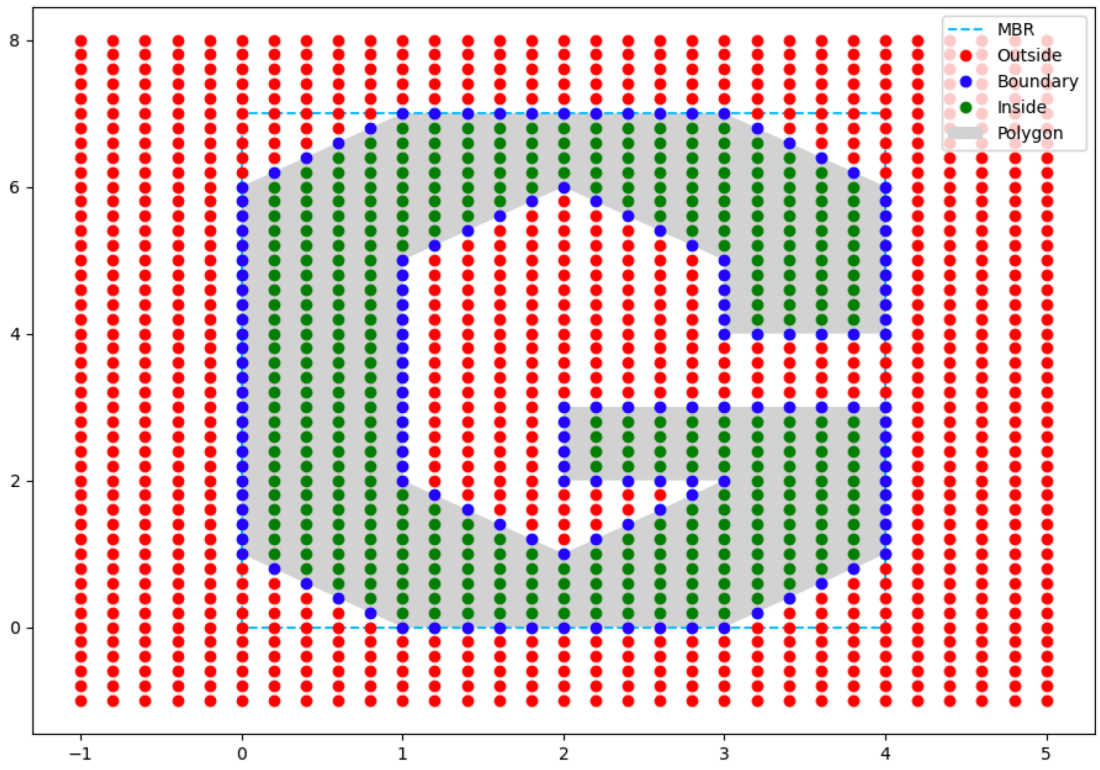
```
try:
    x = float(input("Please enter the X-coordinate of your point: "))
except ValueError:
    x = float(input("Please input a NUMBER. Please enter the X-coordinate of your
point: "))

try:
    y = float(input("Please enter the Y-coordinate of your point: "))
except ValueError:
    y = float(input("Please input a NUMBER. Please enter the Y-coordinate of your
point: “))
```

Task 11. Additional Features

Some additional testing of the program main_from_file.py was done by making a new polygon and new input points. The code to make more points can be found in the file more_input_gen.py (provided in the zip folder and GitHub repository). This code produces a file more_input.csv; when this file is used as input into main_from_file.py instead input.csv, it produces the following result:

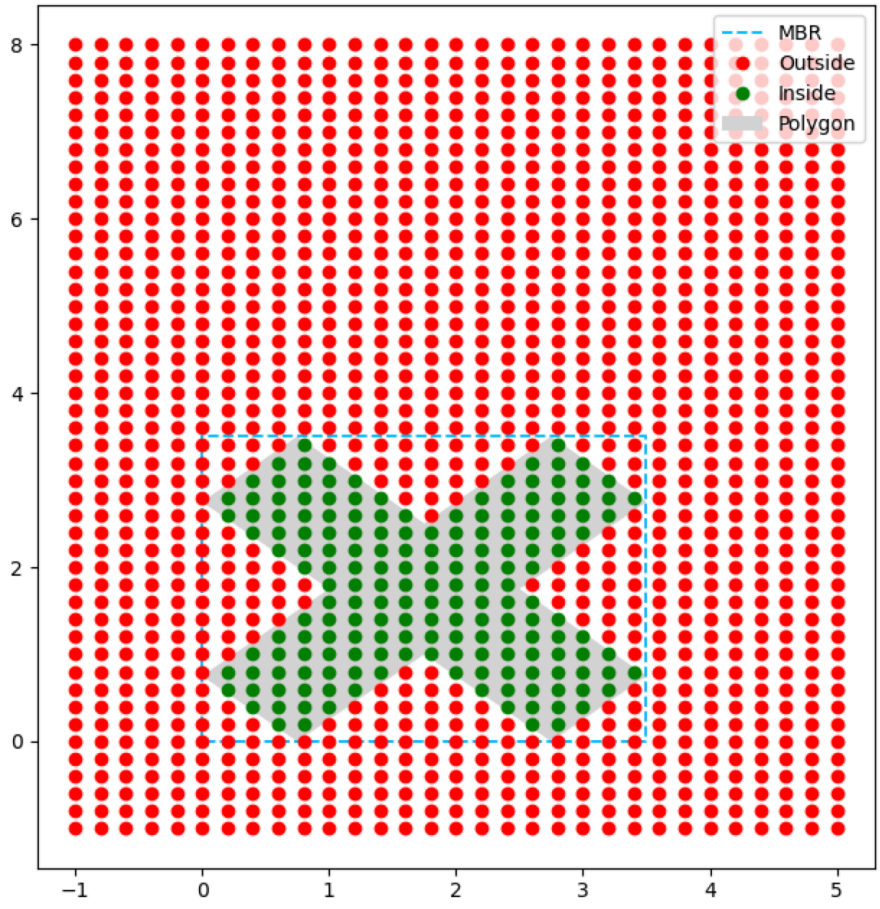
```
if __name__ == "__main__":
    polygon_file = "polygon.csv"
    input_file = “more_input.csv"
    main(polygon_file, input_file, display_result=True,
display_result_with_rays=False)
```



This additional testing allowed improvements to be made to the *boundary()* method. This method sometimes failed to identify points on the boundary due to inconsistent rounding of the decimals when calculating square roots.

Furthermore, another polygon file, polygon_x.csv was also generated manually to allow more testing. This again confirmed that the program works as needed.

```
if __name__ == "__main__":
    polygon_file = “polygon_x.csv"
    input_file = “more_input.csv”
    main(polygon_file, input_file, display_result=True,
display_result_with_rays=False)
```



References

1. Lemons, Phillip, Ray casting algorithm: http://philliplemons.com/posts/ray-casting-algorithm;
2. Lipani, Aldo, Course materials for CEGE0096;
3. Rosettacode.org, Ray casting algorithm, Python: https://rosettacode.org/wiki/Ray-casting_algorithm;
4. Stackoverflow, answer by Jules on 29/11/2008, Determining if a point is between two other points on a line segment: https://stackoverflow.com/questions/328107/how-can-you-determine-a-point-is-between-two-other-points-on-a-line-segment

Git Log

```
commit f24ae8e172bd73d8db64d6426187e9b80d1621d4
Author: ana-kop <ana.kop20@gmail.com>
Date:   Mon Nov 16 17:25:31 2020 +0000
    16/11: last commit!


commit 82ac5630ee8600dde2cabb40e749e1ee1beeeb5e
Author: ana-kop <ana.kop20@gmail.com>
Date:   Mon Nov 16 15:21:04 2020 +0000
    16/11: final edits #2


commit 48b72fe8e16f6d1df0373791eef4616f4824c499
Author: ana-kop <ana.kop20@gmail.com>
Date:   Sun Nov 15 18:07:29 2020 +0000
    15/11: extra polygon for testing the code

commit 21fbb19d98d9e03f60cc07aeea9b3982ff652e96
Author: ana-kop <ana.kop20@gmail.com>
Date:   Sun Nov 15 18:07:09 2020 +0000
    15/11: extra input points for checking

commit 86b982c268f4423df2e8bf211fc73cef2679d2bb
Author: ana-kop <ana.kop20@gmail.com>
Date:   Sun Nov 15 18:06:17 2020 +0000
    15/11: some final edits

commit f04673b47a772d46cfecf5307d6739b40afa177f
Author: ana-kop <ana.kop20@gmail.com>
Date:   Sun Nov 15 12:30:37 2020 +0000
    15/11: finished docstrings

commit a763bcfb7e3d3514342dc2236da553cec4b0dcc7
Author: ana-kop <ana.kop20@gmail.com>
```

Date: Sun Nov 15 12:18:25 2020 +0000
    15/11: adding docstrings

commit b31425594526f44d2490bc89fcfea9dca3cd6c3a
Author: ana-kop <ana.kop20@gmail.com>
Date: Sat Nov 14 17:29:29 2020 +0000
    14/11: minor edits

commit f67062c40aa530a532d41e54b5db46c58dbf34af
Author: ana-kop <ana.kop20@gmail.com>
Date: Sat Nov 14 09:43:48 2020 +0000
    14/11: added comments

commit cf6e932c77daabc9776e38b4b05c516f67dd3c43
Author: ana-kop <ana.kop20@gmail.com>
Date: Thu Nov 12 17:57:27 2020 +0000
    12/11: optimising the code

commit ce222a6aebdf36ffe4ea81eb1262163770f57f11
Author: ana-kop <ana.kop20@gmail.com>
Date: Thu Nov 12 17:56:38 2020 +0000
    12/11: debugging and improving efficiency

commit d53eb7ee03550cb0ba2925ba4f52b160e5cde656
Author: ana-kop <ana.kop20@gmail.com>
Date: Wed Nov 11 20:25:12 2020 +0000
    11/11: points dictionary + read file function

commit 2d6f766ee37611ff48e93b361884648288b8a4ba
Author: ana-kop <ana.kop20@gmail.com>
Date: Wed Nov 11 09:14:11 2020 +0000
    11/11: added plotter + error handling

commit 23c5bc91253814189e519fcfc5487dbd1ac71da4
Author: ana-kop <ana.kop20@gmail.com>
Date: Wed Nov 11 09:04:43 2020 +0000
    11/1 : fully operational version of program 2

commit 97b85c664ae34d0b3560b2dd2459fd08ff58436d
Author: ana-kop <ana.kop20@gmail.com>
Date: Tue Nov 10 14:57:16 2020 +0000
    10/11: write output into file (fully operational version)

commit 36ada75561d1ff118e3b66ef7fcbb5a770a4e7a2
Author: ana-kop <ana.kop20@gmail.com>
Date: Tue Nov 10 07:48:31 2020 +0000
    10/11: updates to boundary method + open file

commit 7e06dff567479631182730e39db0887cdc7bc0ae
Author: ana-kop <ana.kop20@gmail.com>
Date: Mon Nov 9 20:02:31 2020 +0000

```
    09/11: input and output files

commit 4bf8e00f2d2b690a429235b4fbca1a2ffca1b6e5
Author: ana-kop <ana.kop20@gmail.com>
Date:   Mon Nov 9 08:53:33 2020 +0000
    09/11: classifying points

commit c45c6ef0712eb571483da55fe0f187e0dc5584e3
Author: ana-kop <ana.kop20@gmail.com>
Date:   Sun Nov 8 09:28:04 2020 +0000
    08/11: drawing polygon and MBR

commit 874632c8f5f52955e41ab7804dada2d0f0e5aef9
Author: ana-kop <ana.kop20@gmail.com>
Date:   Sat Nov 7 11:36:56 2020 +0000
    07/11: polygon class

commit 2890a2ce61398fe40e3383f1f7e528d6cfc15a2b
Author: ana-kop <ana.kop20@gmail.com>
Date:   Sat Nov 7 11:14:33 2020 +0000
    07/11: improved plotter

commit eb9fbbbd0f37684764ecd43065fd24ffda19278b
Author: ana-kop <ana.kop20@gmail.com>
Date:   Fri Nov 6 15:36:41 2020 +0000
    1: file for program 2

commit 868aaa5660675b443d15bf60c9a570d543ecd5b9
Author: ana-kop <ana.kop20@gmail.com>
Date:   Fri Nov 6 15:29:34 2020 +0000
    2: Point class

commit bf56a177a05fab02e2a826c58c715cb6bbe390f8
Author: ana-kop <ana.kop20@gmail.com>
Date:   Fri Nov 6 15:20:50 2020 +0000
    1
```