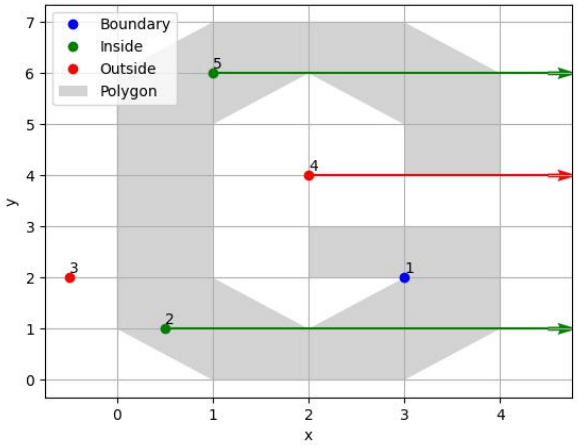
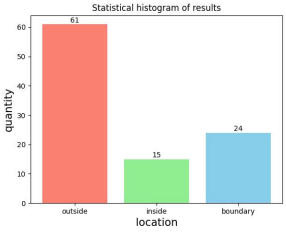
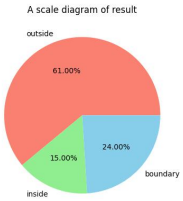
Point-in-Polygon Test

Report for the CEGE0096: 1st Assignment







WENXIN XU
21157882
Uceswx0@ucl.ac.uk

Introduction

Objective

Various problems can arise in the process of determining the relationship between points and graphs, such as the determination of large sets of input points and complex graphical areas, which can make it difficult to reach efficient conclusions. Therefor, we propose to write this software in an attempt to make the judgement easier.

The purpose of this software is to use the computer language to determine the location relationship of points and given graphical area, and eventually to draw a picture to for a visual presentation. I have finished all the required functions used the required format, including reading in the surface data, reading in the point data, categorizing the points, plotting the resulting figure and writing error handling. In addition I have added some additional features, including allowing the user to enter multiple points in main_from_user.py, adding rays, arrows, and the ability to label point numbers for plot function in file_from_user.py, and showing graphically in the additional file additional_feature.py, which shows graphs of the data calculation statistics in main_from_file.py, which function is described in Task 11. Additional Features.

Limitation

The main limitation of the documentation are that the RCA part of the code is too cumbersome, and the program requires the user to input the points as a csv file in a specific format, which can lead to reading errors caused by the csv format not matching the settings. Although the program can do the job of adding points id to the plot section, the main code in main_from_file.py does not use this part of the code, because the plot with the id is too ugly, which may cause difficulties for users who want to get the point positions visually from the plot, as they have to read the results through the output.csv file.

Program Development

The software makes judgement for two different types of input points, one is using a csv file as input point, the other is a manual input by the user to obtain the points during the running of the software, other than that the underlying logic is similar in both programs. The main program includes reading the graphics area file polygon.csv, reading the input point data (input.csv or input by user), categorizing points, writing the result file output.csv (main_from_user.py does not have this part) and drawing the result graphics.

Development process

It took me about 5 days to successfully complete all the functions of the programming task in my assignment. The process of writing the program is in the following order: reading the assignment requirements, software architecture, writing the program, debugging the program, checking the pep8 format, checking comments, and writing the additional part.

Project Description

The software is written in python 3.9, where the plotting function calls the matplotlib library. In order to use the matplotlib library properly, the version of the library should be greater than 3.3.3.

The software should put with a polygon.csv file and plotter.py file in the same folder as the main function to be called in. In addition, for main_from_file.py, the input.csv file should be put in the same folder, after running the program will generate an output.csv file in the same folder to store the results of these points and pop up a window to plot the graph. For main_from_user.py, you need to manually enter the x and y coordinates of the points, you can enter more than one point, and at the end of the run you will get the results of the points in the interactive interface, and get a pop-up window to with graph.

Software

Task 1. The MBR Algorithm

This program handles the MBR function as a class, with the following flowchart and procedures.

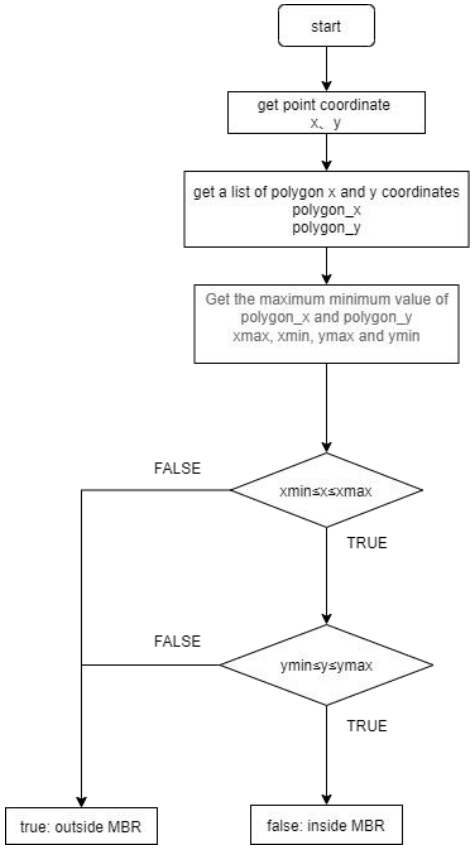


Figure 1 The flow chart of the MBR

Because there will multiple judgement ‘Determine if the value is between the two values’ subsequently, the program creates class max_min, in which class use def judge_boundary to judgement case that includes the equal, and def judge_location is the case that does not include the equal sign, eventually return true or false. class MBR and class CRA are both subclass of class max_min.

This section is to create a class about the Minimum Bounding Rectangle(MBR), class MBR(max_min), this class is set as a subclass of class max_min because it makes the judgement content of class max_min several times.

And for (1) def get_x_y by inputting a list containing a list of point coordinates value returns a list of the list of x, y points.

As for (2) def mbr_result, which requires the input of a list of polygon and input point coordinates so as to determine whether the point is within the MBR, returning FALSE if it is in MBR and TRUE if it is not in MBR.

This is achieved by calling self.get_x_y to get the set of points of polygon_x, polygon_y, and get the maximum and minimum values of x and y, i.e. the coordinates of the points of the MBR. Subsequently, calling def judge_boundary to determine whether the x and y of the points are within the boundaries of the MBR (xmax, xmin, ymax, ymin), which Completes the smallest rectangle determination.

The statement calls (2) def mbr_result in MBR via the main function def main(), and saves the result as 'outside' when the point is not inside MBR.

```
# Determine whether the point is outside the minimum Bounding Rectangle (MBR).
class MBR(max_min):
    def __init__(self) -> None:
        super().__init__()

(1) def get_x_y(self, value):
        x = []
        y = []
        for i in range(len(value)):
            i_value = value[i]
            x.append(i_value[0])
            y.append(i_value[1])
        return x, y

(2) def mbr_result(self, polygon_value, point):

        polygon_x, polygon_y = self.get_x_y(polygon_value)
```

```
# The range of MBR.
xmin = min(polygon_x)
xmax = max(polygon_x)
ymin = min(polygon_y)
ymax = max(polygon_y)

x = point[0]
y = point[1]

if self.judge_boundary(xmax, xmin, x) == False:
    return True
elif self.judge_boundary(ymax, ymin, y) == False:
    return True
else:
    return False
```

Task 2. The RCA Algorithm

The main function statement def main() enters the RCA judgment after judging that the point is within the MBR. The flowchart and program are as follows.

The judgement part of the RCA program is performed by class RCA(max_min), which inherits class max_min and returns the result 'boundary', 'outside' or 'inside'.

This program (1) def line_judge is the main program of class RCA(max_min) , the other functions in class RCA will be described in turn in subsequent paragraphs.

In def line_judge, the count is given a value of 0 and the result 'not know'. As the ray of a point may just cross the intersection of lines or coincide with a line with k=0, whether to count or not needs to be determined by judging the relationship between the two line segments of the intersection with line or the relationship between the lines before and after the judge line with ray when k=0. So the program obtains a sequential list of the coordinates of three consecutive boundary points of the graph line_1_list, line_2_list and line_3_list, which is achieved by calling def polygon_line in class Polygon to achieve this

The second line, line_2_list, is used as the reference line. The loop traverses all the lines for counting judgement. In the loop, y_1_1 work as the y coordinate of the point before the first line (line_1_list), x_2_1 and x_2_2 are the x coordinates of the points before and after the second line (line_2_list). y_2_1, y_2_2 and y_3_2 are used in a similar way in this program.

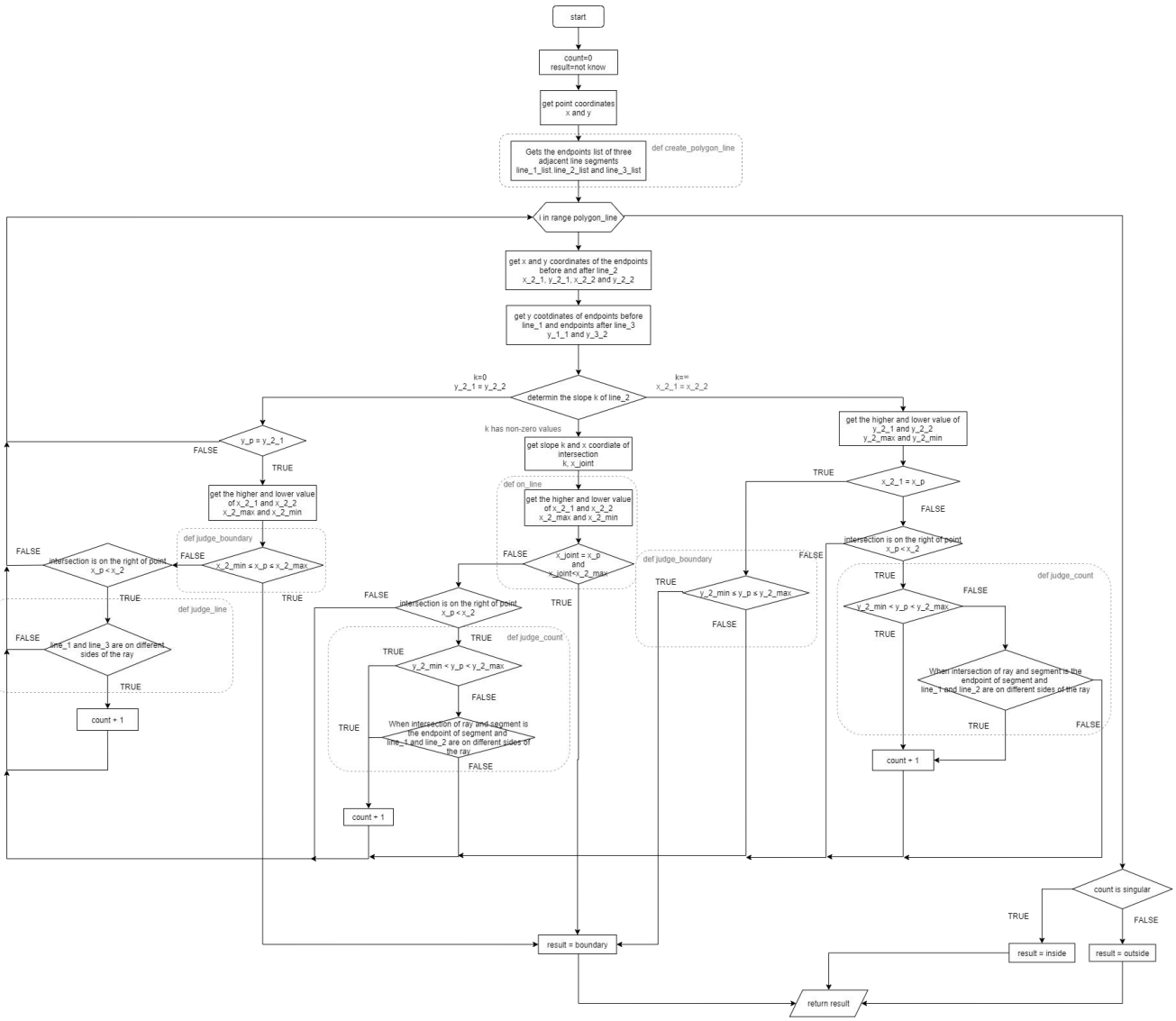


Figure 2 The flow chart of the RCA

The data processing that needs to be applied has been completed, after which we make a classification judgment of the type of line segment, including three types of slope k = 0, k does not exist and k exists non-zero number. For each case, it is necessary to first determine whether the point is on the boundary and then to make next judgement, if it is on the boundary it should return result = 'boundary' and terminate the loop (break) of the line segment without further counting judgement.

For k = 0 (y_2_1 == y_2_2), both the boundary and counting cases need to satisfy that the ray coincides with the line segment (y_p == y_2_1). When the above conditions are met, the first judgement is whether the point is on the boundary, and the counting judgement is made if it is not on the boundary. The case of counting is when the first point (y_1_1) of the previous line segment (line_1_list) and the end point (y_3_2) of the next line segment (line_3_list) are on

different sides of the ray ((4) def judge_line) and the line segment is to the right of the point (x_p < x_2_min), otherwise this line break is not counted without an intersection point (continue).

For k does not exist (x_2_1 == x_2_2), a boundary judgement is made, if not on the boundary, for the count judgement there are two conditions, first is the ray of the point and the line segment have an intersection (def judge_count) , and second condition is the point is on the left of the line segment (x_p < x_2_1), if the condition is met count the number (count += 1), otherwise the current loop is skipped to the next line segment (continue).

For the case k exists a non-zero number, the slope k and the intersection of the ray and the line segment x_joint are first calculated, then make boundary judgement, which is achieved by calling (5) def on_line. The case of not being on the boundary and counting is that the ray of the point and the line segment have an intersection ((3) def judge_count) and the point is to the left of the line segment (x_p < x_2_1), otherwise the loop is skipped to judge the next line segment (continue).

When all the line segments have been looped, we get the count of the points on the boundary (result == 'boundary') and the points not on the boundary (result == 'not know'). For point not on the boundary, when the count is odd (count % 2 == 1) the result is 'inside' and when it is even (count % 2 == 0) the result is 'outside'.

The main program's implementation of RCA calls (1) def line_judge in class RCA via the main function def main(), get the relationships of point and polygon ('boundary', 'inside' and 'outside').

```
# Use the ray casting algorithm (RCA) to determine whether the point is in the graph and
return the result.
class RCA(max_min):
    def __init__(self) -> None:
        super().__init__()

    # RCA main program.
(1) def line_judge(self, polygon_value, point):

        count = 0
        result = 'not know'

        x_p = point[0]
        y_p = point[1]

        # Gets the endpoints list of three adjacent line segments by class Polygon.
        polygon = Polygon()
        line_1_list = polygon.polygon_lines(1, polygon_value)
        line_2_list = polygon.polygon_lines(2, polygon_value)
```

```
        line_3_list = polygon.polygon_lines(3, polygon_value)


        # Use line 2 to determine the number of times the ray passes through the polygon and
the boundary point.
    ① for i in range(len(polygon_value)):
            y_1_1 = line_1_list[i][0][1]
            x_2_1 = line_2_list[i][0][0]
            y_2_1 = line_2_list[i][0][1]
            x_2_2 = line_2_list[i][1][0]
            y_2_2 = line_2_list[i][1][1]
            y_3_2 = line_3_list[i][1][1]


            # The slope of the line segment (k) is 0.
            if y_2_1 == y_2_2:
                # Continue to judge when the point is on a straight line, otherwise jump to
the end of the loop (continue).
                if y_p == y_2_1:
                    x_2_max = max(x_2_2, x_2_1)
                    x_2_min = min(x_2_2, x_2_1)
                    # Return 'boundary' when the point is on the line segment and break out of
the loop.
                    if self.judge_boundary(x_2_max, x_2_min, x_p):
                        result = 'boundary'
                        break
                    # Line 1 and line 3 are counted on both sides of the ray, otherwise (line
1 and line 3 are on the same side of the ray) they are not counted.
                    elif self.judge_line(y_p, y_1_1, y_3_2) and x_p < x_2_min:
                        count += 1
                    else:
                        continue
                else:
                    continue


            # k does not exist (infinity).
            elif x_2_1 == x_2_2:
                # Return 'boundary' when the point is on the line segment and break out of the
loop.
                if x_p == x_2_1:
                    y_2_max = max(y_2_2, y_2_1)
                    y_2_min = min(y_2_2, y_2_1)
                    if self.judge_boundary(y_2_max, y_2_min, y_p):
```

```
                    result = 'boundary'
                    break
                else:
                    continue
            # If there is ray intersection to the right of point, count.
            else:
                if self.judge_count(y_1_1, y_2_2, y_2_1, y_p) == True and x_p < x_2_1:
                    count += 1
                else:
                    continue

        # There is k non-zero.
        else:
            k = (y_2_2-y_2_1)/(x_2_2-x_2_1)
            x_joint = (y_p - y_2_1)*(1/k)+x_2_1
            # Return 'boundary' when the point is on the line segment and break out of the
loop.
            if self.on_line(x_joint, x_2_1, x_2_2, x_p):
                result = 'boundary'
                break
            # If there is ray intersection to the right of point, count.
            else:
                if self.judge_count(y_1_1, y_2_2, y_2_1, y_p) == True and x_p < x_joint:
                    count += 1
                else:
                    continue

    # Determine the position of the point and assign the value
    if result == 'boundary':
        m = 'boundary'
    else:
        if count % 2 == 0:
            m = 'outside'
        elif count % 2 == 1:
            m = 'inside'
        else:
            m = 'not know'
    return m
```

Other functions and descriptions in `class RCA` are given below.

Function (2)def judge_count is to determine if the ray of a point has an intersection with the boundary of figure by calling def judge_location and def judge_line, if has return TRUE.

Function (3) def judge_line is to determine whether to count number, when the ray coincides with the endpoint of a line segment, counting when the two lines are on the same side of the ray (TRUE) and not counting when the two lines are on different sides of the ray (FALSE).

Function (4) def on_line is a function used to determine if a point is on a line segment when the slope is not equal to 0 i.e. result = 'boundary' condition.

Finally, the class polygon called in the main program (1) def def line_judge to create the line list is described in the Task 5. Object-Oriented Programming.

```
class RCA(max_min):
    --omit--
（1）def line_judge(self, polygon_value, point):
        --omit--


    # Determine whether there is an intersection between point ray and line segment that needs
to be counted.
（2）def judge_count(self, y_1_1, y_2_2, y_2_1, y_p):


        y_2_max = max(y_2_2, y_2_1)
        y_2_min = min(y_2_2, y_2_1)
        y_joint = y_p
        # The ray is in the middle of the line segment, have a intersection, return true.
        if self.judge_location(y_2_max, y_2_min, y_joint):
            return True
        # The ray passes through the end of the line segment to judge the position of the two
lines. The same side is not counted(True), the different side is counted(False).
        elif y_joint == y_2_1:
            if self.judge_line(y_joint, y_1_1, y_2_2):
                return True
            else:
                return False
        else:
            return False


    # The ray passes through the end of the line segment to judge the position of the two
lines. The same side is not counted(True), the different side is counted(False).
（3）def judge_line(self, y_joint, y_1_1, y_2_2):
        if y_joint < y_1_1 and y_joint > y_2_2:
```

```
            return True
        elif y_joint > y_1_1 and y_joint < y_2_2:
            return True
        else:
            return False

    # Determine whether a point is on a line segment.
() def on_line(self, x, x_2_1, x_2_2, x_p):
        if x == x_p:
            x_max = max(x_2_2, x_2_1)
            x_min = min(x_2_2, x_2_1)
            if self.judge_boundary(x_max, x_min, x):
                return True
            else:
                return False
        else:
            return False
```

Task 3. The Categorisation of Special Cases

The program code for the point classification in the main program is as follows.

The categorize points section stores the results as a dictionary type, with the point order as the key corresponding to the value of the point result, so (1) first creates the dictionary type and corresponding id value to 'not know'. Then (2) for each point call class MBR and class RCA, the former reassigning the dictionary value as 'outside' if it is a true, otherwise calling the latter to return the dictionary worthy result directly. When the loop end each point is judged and classified.

```
def main():
    --omit--

    # categorize points
    print('categorize points')

    # Data classification is initialized to 'not know', dictionary type.
(1) result_point = dict()
    for i in range(len(input_value)):
        result_point[i+1] = 'not know'
```

```
    # Get categorize result.
(2) for i in range(len(input_value)):
        rca_judge = RCA()
        mbr_judge = MBR()

        # The point out MBR, result is outside.
        if mbr_judge.mbr_result(polygon_value, input_value[i]):
            result_point[i+1] = 'outside'
        # The point in MBR, RCA judgment, return the result.
        else:
            result_point[i +
                         1] = rca_judge.line_judge(polygon_value, input_value[i])
```

Task 5. Object-Oriented Programming

I completed the content of the requested OOP, and the code is written in two main parts, the writing of class geometry and its subclass, and the implementation of MBR and RCA. Also although the reading and writing out of csv can be written as classes, I did not write def csv as a class in the documentation in order to show my mastery of the content of the called functions.

As the contents of the MBR and RCA have already been described above, a brief description of these two classes is given here first. First the value judgement class max_min as a parent class and then completing the writing of class MBR and class RCA as subclass.

For class geometry, the documentation is divided into three parts: points, lines and polygon. Where (1) class Point are used primarily in combination with (4) def csv to read a list of points and polygon stored in csv. (2) class Line and (3) class Polygon are combined to obtain a given graph of three adjacent line segments of the point list, by considering the value of the given input to determine for the first few line segments column surface line segment list of acquisition, and the use of (2) class Line to obtain a list of line segments, the final use in class RCA, see the code in Task 2. The RCA Algorithm for details. The code for class geometry and its subclasses and the code for def csv are as follows.

In addition, during the drawing process I changed and called the written plot code plotter.py given by the teacher, and for the second file main_from_user.py and additional feature.py I also called MBR, RCA and Point from the first file main_from_file.py to simplify the repetition of code. writing.

```
    # class geometry and it subclasses.
    class geometry:
        def __init__(self, name):
            self.__name = name
```

```
        def get_name(self):
            return self.__name
```

（1）
```
class Point(geometry):
        def __init__(self, name, x, y):
            super().__init__(name)
            self.__x = x
            self.__y = y

        def get_point(self):
            self.__point = [self.__x, self.__y]
            return self.__point
```

（2）
```
class Line(geometry):
    def __init__(self):
        pass

    def lines(self, point1, point2):
        self.__point1 = point1
        self.__point2 = point2
        self.line = [point1, point2]
        return self.line
```

（3）
```
class Polygon(geometry):
    def __init__(self):
        pass

    def polygon_lines(self, m, value):
        polygon_line_point = []
        line = Line()

        # line 1
        if m == 1:
            for i in range(len(value)-1):
                point_1 = value[i]
                point_2 = value[i+1]
                polygon_line_point.append(line.lines(point_1, point_2))
```

```
            polygon_line_point.append(line.lines(point_2, value[0]))

        # line 2
        elif m == 2:
            for i in range(len(value)-2):
                point_1 = value[i+1]
                point_2 = value[i+2]
                polygon_line_point.append(line.lines(point_1, point_2))
            polygon_line_point.append(line.lines(point_2, value[0]))
            polygon_line_point.append(line.lines(value[0], value[1]))

        # line 3
        elif m == 3:
            for i in range(len(value)-3):
                point_1 = value[i+2]
                point_2 = value[i+3]
                polygon_line_point.append(line.lines(point_1, point_2))
            polygon_line_point.append(line.lines(point_2, value[0]))
            polygon_line_point.append(line.lines(value[0], value[1]))
            polygon_line_point.append(line.lines(value[1], value[2]))

        else:
            print('error')
            raise

        return polygon_line_point

    ############################################################
    # read csv file, and save as list
    # can be save as class
(4) def csv(path):
        with open(path, 'r') as f:
            f_value = list()

            for n in f.readlines()[1:]:
                row = n.strip("\n").split(',')
                point = Point(int(row[0]), float(row[1]), float(row[2]))
                f_value.append(point.get_point())
```

```
    f.close()
    return f_value
```

Task 6. On the Use of Git and GitHub

I use Git and GitHub as a cloud for downloading, upload and view past changes as shared files, although I know I can create branches for some additional operations, I don't do this due to the submission requirement to submit in given directory.

At the start of the job I configured Git and the SSH key in GitHub, which simplified the process of entering the account and password for each fetch and push. Then used GitHub to cloned and downloaded given files (it also can use git pull) to the given folder.

In subsequent commits, the Git repository was associated with the local folder (git remote add <name > <url>), then added files to the local repository (git add <file>) and committed changes (git commit -m ' '), and finally submitted these files to remote repository (git push <name> master) and checked in GitHub to see if the upload is successful. The specific upload information can be seen in log out module later.

Task 9. Plotting

I have added some of the statements in plotter.py to implement more drawing functionality such as ray, arrow, and point code. Since the drawing elements in main_from_file.py are too many extra elements that make the drawing area difficult to see, so main_from_file.py uses a basic graphical presentation instead of showing rays and id. This part of the functionality is more representative in main_from_user.py and I will use the code in main_from_user.py as an example to illustrate the code. Figure 3 shows an example of the results of the application of the plot module in main_from_user.py

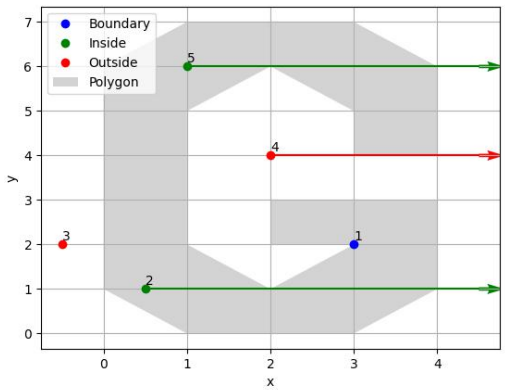


Figure 3 Schematic of the main_from_user.py plotting results

Firstly, the document will describe that the code changed in plotter.py is (in grey background), after which the imports to the plot part of the code in the main program will be described.

In class Plotter (1) def add_point, name=None is added to the assignment in order to display the point number, and name is referenced in the file using plt.text. When the user does not want to use this part of the function, the name is not assigned when def add_point is called.

To add the ability to draw rays, write (2) def add_ray in class Plotter and use try except to make it possible to draw rays and arrows only when the point is on boundary and inside MBR , otherwise it is skipped. The rays drawn 'outside' are red and 'inside' are green.

In class Plotter (1) def show adds the axis names (x, y) for the x and y axes and adds the grid line plt.grid.

```
from collections import OrderedDict
import matplotlib
import matplotlib.pyplot as plt
--omit--
class Plotter:
  --omit--
```

```
(1) def add_point(self, x, y, kind=None, name=None):
        plt.text(x, y+0.1, name)
        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')
```

```
(2) def add_ray(self, x, y, x_max, kind=None):
        try:
            if kind == 'outside':
                plt.plot([x, x_max+0.5], [y, y], 'r')
                plt.quiver(x_max+0.5, y, 1, 0, color='r')
            elif kind == 'inside':
                plt.plot([x, x_max+0.5], [y, y], 'g')
                plt.quiver(x_max+0.5, y, 1, 0, color='g')
        except:
            pass
```

```
(3) 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.xlabel('x')
        plt.ylabel('y')
        plt.grid()
        plt.show()
```

The following is the code for the main program that calls the class Plotter function.

Use (1) from plotter import Plotter to call the written file Plotter.py in the program. plot the polygon graph ((2) def add_polygon), (3) for points that result in 'outside' or 'inside' within the MBR rectangle, plot the ray with the arrow and show it's id, and then use (4) plotter.show() to call the window to display the graph.

```
(1) from plotter import Plotter

def main():
    --omit--

    #plot
    print('\nplot polygon and points')
    plotter = Plotter()

    # plot polygon
(2) polygon_x, polygon_y = mbr_judge.get_x_y(polygon_value)
    plotter.add_polygon(polygon_x, polygon_y)

    # plot ray for point inside MBR and not 'boundary'
(3) for i in range(len(input_value)):
        # plot ray for point inside MBR and not 'boundary'
        if mbr_judge.mbr_result(polygon_value, input_value[i]) == False:
            x_max = max(polygon_x)
            plotter.add_ray(
                input_value[i][0], input_value[i][1], x_max, result_point[i+1])
        plotter.add_point(input_value[i][0],
                          input_value[i][1], result_point[i+1], i+1)
```

```
(4) plotter.show()
```

Task 10. Error Handling

The class polygon mentioned in task5. Object-Oriented Programming above adds an else: raise statement to the judgement of the lines, which is used to return an error for unintended values.

In addition, the other error handling in this program is mainly used in the input and output of main_from_file.py and main_from_user.py. The code descriptions will use the input of main_from_user.py and the output of main_from_file.py as examples, in which the point information input of main_from_user.py has been changed to a multi-point input.

The description and code are as follows.

In the input code of main_from_user.py when the polygon.csv file is read in error the statement 'Make sure pologon.csv is in the same folder.' will appear and the program will pause, when the user changes it correctly and clicks on any of key the program will continue reading.

In the process of reading points, when the user makes a typing error for the first time, the screen will show 'Data error, please re-enter.' and 'If you want to stop typing, press Enter.', when the user makes a typing error for the second time, the program will stop the input module and continue with the categorizing point.

The next paragraph is a description of the output in main_from_user.py.

Main_from_user.py

```
def main():
    # read polygon.csv as list
    print('read polygon.csv')
    try:
        polygon_path = '.\\polygon.csv'
        polygon_value = csv(polygon_path)
    except:
        print('Make sure pologon.csv is in the same folder.')
        # pause, it would be better to use os.system("pause") here.
        input()
        polygon_path = '.\\polygon.csv'
        polygon_value = csv(polygon_path)

    # insert point as list
    print('\nInsert point information')

    # A loop that allows the user to keep entering parameters
```

```
    # allow user put multiple points
    a = 1
    input_value = []
    while a == 1:
        try:
            x = float(input('x coordinate: '))
            y = float(input('y coordinate: '))
        except:
            print('Data error, please re-enter.')
            print('If you want to stop typing, press Enter.')
            try:
                x = float(input('x coordinate: '))
                y = float(input('y coordinate: '))
            except:
                break

        input_value.append([x, y])
```

When outputting output.csv in main_from_file.py, the output is done by calling the def output function, which sets the output mode to 'x' for fear of overwriting the user's useful file with the output content. On first output failure it will display 'Check whether output.csv exists in the same folder' and program will pause, waiting for the user to modify and enter any information before the program resumes output.

Main_from_file.py

```
def main():
    --omit--

    # write output.csv
    print('write output.csv')
    try:
        output(input_value, result_point)
    except:
        print('Check whether output.csv exists in the same folder')
        input()

        output(input_value, result_point)
```

Task 11. Additional Features

As the user may want to enter multiple point coordinates in main_from_user.py, the program is modified to have a multi-point input mode, as described in (Task 10. Error Handling).

For readability of the output graph, the program modifies the given Plotter.py and plots in main_from_user.py a ray with an arrow, a point number and changes the plotting format, as described in (Task 9. Plotting)

The above two points are illustrated functions, which is stored in main_from_file.py, and will not be repeated here.

In addition, considering the user's judgement of the relative graphical position of points subsequent to the great possibility of data statistics and graphical display. However, the main program only generates a csv containing multi-point information and a graph for easy visualization, which is not easy to quickly count statistics and display the results. The program uses the input points and polygon in main_from_file.py as a reference in the additional feature.py file, using the added statistical bar and pie charts for plotting. The bar charts provide a good display of statistical quantity information, and the pie charts generate a proportion of the results, and these charts facilitate the information statistics and simplify the process of visualizing the data for possible subsequent reporting.

Figures 4 and 5 are schematic diagrams of the results of the run, and the exact code of the program is as follows

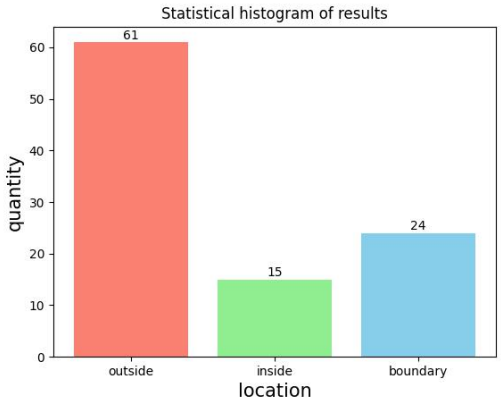


Figure 4 Statistical histogram of results

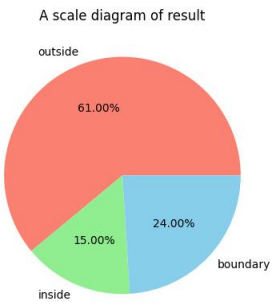


Figure 5 Scale diagram of result

The first step of the class Figure (1) def count carries out the information statistics of the point position results and finally returns the statistical values. (2) def show_bar is the content of the drawing of the bar chart, ① in the color will be different results of the bar drawn into different colors, ② in the bar chart above to add the count value. (3) def show_pie is the plotting procedure for pie charts displaying data proportions retaining two decimal places and giving a specific colour to keep in line with the bar chart.

```
# bar graph and pie graph, adivice use in main_from_file.
class Figure():
    def __init__(self) -> None:
        Pass

# count the result
(1) def count(self, input_value, result_point):
        out = 0
        ins = 0
        bou = 0
        for i in range(len(input_value)):
            if result_point[i+1] == 'outside':
                out += 1
            elif result_point[i+1] == 'inside':
                ins += 1
            elif result_point[i+1] == 'boundary':
                bou += 1
        return out, ins, bou

# plot bar figure
(2) def show_bar(self, x, y):
    ①  plt.bar(x=x, height=y, color=["salmon", 'lightgreen', 'skyblue'])

    ②  for i in range(len(y)):
            plt.text(x=x[i], y=y[i]+2.5, s=str(y[i]), ha='center', va='top')

        plt.title("Statistical histogram of results")
        plt.xlabel("location", fontsize=15)
        plt.ylabel("quantity", fontsize=15)
        plt.show()
```

```
# plot pie figure
(3) def show_pie(self, x, y):
        plt.pie(x=y, labels=x, autopct='%1.2f%%', colors=[
                "salmon", 'lightgreen', 'skyblue'])
        plt.title('A scale diagram of result')
        plt.show()
```

Git Log

The statement about git log in git base is copied below.

```
$ git log
commit 0e45257cd8e2070f9c47c2a5af303a986fc3014e (HEAD -> master)
Author: uceswx0 <xwx1724088989@163.com>
Date:   Mon Nov 22 10:51:32 2021 +0000

    change format

commit b931ffa74d63c5146fc86c63b82450ec9a6fcd7a
Author: uceswx0 <xwx1724088989@163.com>
Date:   Mon Nov 22 10:40:42 2021 +0000

    change format

commit b5462dabc7c365d9f9837cb0d26816ede608ecf0
Author: uceswx0 <xwx1724088989@163.com>
Date:   Mon Nov 22 10:37:25 2021 +0000

    change class geo

commit 7d99839f00df6c0809a6515d274c442897442025
Author: uceswx0 <xwx1724088989@163.com>
Date:   Mon Nov 22 10:16:53 2021 +0000

    change read input point
```

```
:...skipping...
commit 0e45257cd8e2070f9c47c2a5af303a986fc3014e (HEAD -> master)
Author: uceswx0 <xwx1724088989@163.com>
Date:   Mon Nov 22 10:51:32 2021 +0000

    change format

commit b931ffa74d63c5146fc86c63b82450ec9a6fcd7a
Author: uceswx0 <xwx1724088989@163.com>
Date:   Mon Nov 22 10:40:42 2021 +0000

    change format

commit b5462dabc7c365d9f9837cb0d26816ede608ecf0
Author: uceswx0 <xwx1724088989@163.com>
Date:   Mon Nov 22 10:37:25 2021 +0000

    change class geo

commit 7d99839f00df6c0809a6515d274c442897442025
Author: uceswx0 <xwx1724088989@163.com>
Date:   Mon Nov 22 10:16:53 2021 +0000

    change read input point

commit de8315088d047b3fff0eba7ba6874f089f226a5d
Author: uceswx0 <xwx1724088989@163.com>
Date:   Mon Nov 22 00:55:46 2021 +0000

    change class point and def csv

commit 7d66fe6091093cfc44707a56ea81bc271743b753
Author: uceswx0 <xwx1724088989@163.com>
Date:   Sun Nov 21 23:42:28 2021 +0000

    change class polygon

commit b1fbb8deada56d3fe743ae9c843ee8ef96062958
Author: uceswx0 <xwx1724088989@163.com>
```

```
Date:   Sat Nov 20 22:47:25 2021 +0000

    change commit and format

commit 55dda40f63a4c304973920099a147c44bfa78d9a
Author: uceswx0 <xwx1724088989@163.com>
Date:   Sat Nov 20 22:46:29 2021 +0000

    modified commit and format

commit c48bb2b6076a47d147dd7ffbf2d17c6fe54ad5a2
Author: uceswx0 <xwx1724088989@163.com>
Date:   Sat Nov 20 17:03:59 2021 +0000

    change commit and format

commit f8d1f1ab0ca531964ea4a9929aea54cb36d0f523
Author: uceswx0 <xwx1724088989@163.com>
Date:   Sat Nov 20 16:12:16 2021 +0000

    Modify the spelling

commit ede4346e094eb20642d1f3c324c4837f18f883f3
Author: uceswx0 <xwx1724088989@163.com>
Date:   Sat Nov 20 14:55:03 2021 +0000

    change plot function

commit e033adb2a8dcdec5f825275fcfce8b6b6e26bd54
Author: uceswx0 <xwx1724088989@163.com>
Date:   Sat Nov 20 14:54:05 2021 +0000

    change commit

commit 6b38cd2387d1cf1257c780c0c00832d01710300a
Author: uceswx0 <xwx1724088989@163.com>
Date:   Sat Nov 20 14:53:27 2021 +0000

    Modify bar function
```

```
commit b278975dd90af29bd033d13e92cca279b2a876bc
Author: uceswx0 <xwx1724088989@163.com>
Date:   Sat Nov 20 03:55:50 2021 +0000

    add ray

commit 9de46dab3be5a35fa0a8fa15e6cca85b7fce1160
Author: uceswx0 <xwx1724088989@163.com>
Date:   Sat Nov 20 03:54:56 2021 +0000

    commit and simplifies feature

commit ef701b14843836fedec43a5d814678fda6811e02
Author: uceswx0 <xwx1724088989@163.com>
Date:   Sat Nov 20 03:53:47 2021 +0000

    modify to multi-input mode

commit 663d4b9bb1e9498763bb7e1aec9d4f8316686875
Author: uceswx0 <xwx1724088989@163.com>
Date:   Sat Nov 20 03:51:39 2021 +0000

    change out put mode

commit 23328f894aa41fd5335555cdcf22f7c49f0d4a15
Author: uceswx0 <xwx1724088989@163.com>
Date:   Fri Nov 19 13:52:57 2021 +0000

    second sub

commit 73cce699660064114bd489db9603f89e905752a5
Author: uceswx0 <xwx1724088989@163.com>
Date:   Fri Nov 19 13:50:19 2021 +0000

    second sub

commit 6fac3e5949f8943901O0ca0afaf1cbcd6fe26354
Author: uceswx0 <xwx1724088989@163.com>
```

```
Date:   Fri Nov 19 12:42:05 2021 +0000

    first sub

commit 29147e5f8b0398151034bd01836f513d328adeda
Author: uceswx0 <xwx1724088989@163.com>
Date:   Fri Nov 19 12:25:04 2021 +0000

    first sub
(END)
```