Programming 2023/24 — Assignment 2

December 14th 2023

1 About the assignment

You will write Python functions to make various calculations and draw various plots related to the geometry of circles in the plane.

To support your programming, you will also need to do some mathematics.

Your mark for the assignment will be based on both the programming and mathematical parts. This is a programming module and so the programming is the main part of this assignment. Nonetheless, where there are relevant mathematical tasks, the idea is that your solution to a programming task should be based on them. Therefore, a student who submits only the programming part without corresponding maths may receive a mark of zero for any tasks for which a mathematical part was requested. Tasks where some maths is expected have "(Math)" after the task number.

It's important that you test your functions before submitting them. No check will be made at the point of submission to the Moodle server. Therefore, pay close attention to examples given in relation to the earlier tasks and make sure that your functions work for those.

1.1 Rules

The assignment should be your own work without help from others. You may access all Python documentation that was permitted for the first assignment (see the information on Ultra). You may also view web pages referenced below (just click on a link that is formatted as underlined green text) and may look up online or in books the meaning of technical terms used in the assignment. You should not seek further assistance from books or online sources.

You may import the `math` and `numpy` modules or individual items from those modules. You may also make the usual import of the `pyplot` part of `matplotlib` as `plt` and this is **assumed in examples given below**. Other imports are not allowed, specifically not other parts of `matplotlib` such as `matplotlib.patches` and `matplotlib.collections`.

When plotting, it is expected that you will use only `plt.plot` in the functions you submit as your solutions. You should not use other parts of `matplotlib`. However, when testing your code you will clearly want to use `plt.show`. Moreover, your circles will then tend to look more like circles if you also include `plt.gca().set_aspect('equal', 'box')` before `plt.show()` when testing.

1.2 Submission of answers

Your submission will consist of **two files**:

- A single file of Python code that contains only imports, function definitions and comments. This should be submitted to the Moodle server. Functions may call each other and you may include additional functions to support the ones required as solutions to the assignment. Feel free to include comments and to leave commented-out code or commented-out print statements as examples of how your functions would be used. Do not call `plt.show()` in functions that are submitted as solutions to tasks.

- A single PDF file of your mathematical work, clearly identifying to which task each part of your file relates. Please note, this work should be easy to follow and should contain text to explain the maths you are doing. If you choose this submission to be a scan of handwritten pages, be sure that this is legible. This should be submitted to Gradescope having followed the link from Ultra.

The two submission points will open shortly after the release of the assignment, but please do get started working on this assignment before then if possible.

The deadline for the assignment will be 18:00 on Wednesday, January 17th.

1.3 Difficulty of the tasks and time required

The tasks are ordered in a way that seems natural for the material and this also is intended to be in order of increasing challenge.

Tasks 1 to 7 are deliberately clearly sign-posted. Tasks 8 to 10 require increasing independence and tasks 11 to 13 are intended to challenge.

Only you can decide how much time to spend on the assignment between now and the deadline. However, I do not expect you to spend days on this assignment or to make yourself miserable trying to answer questions for which you are not ready. The more challenging tasks are provided to stimulate and provide interest for students who have been enjoying programming.

1.4 Marking

I have not decided a mark scheme yet and will not finalise the number of marks per task until have done most of the marking. However, you will certainly be able to gain at least sixty percent of the marks on tasks 1 to 10 alone. Again, therefore do not spend an unreasonable amount of time on the later tasks if, after making a serious attempt at them, you think you are unable to do them.

Some marks will be reserved for the quality of plots produced, for example for sensible use of colour to distinguish different objects appearing in a plot. Marks may also be reserved for quality of code. In particular, there is a certain hierarchy to some of the tasks where a function written for an earlier task would naturally be called for a later task and doing so may be rewarded in marking.

2 The assignment

On practical sheet 3, you saw that a point in the plane may be represented in Python as a list of two numbers: for example `[-0.3, 1.2]` represents the point $(-0.3, 1.2)$. This representation will be used throughout what follows.

2.1 Working with colours

In what follows, you will need to write functions to add circles and line segments to matplotlib plots in specified colours. A colour can be specified in several different ways described in the matplotlib tutorial on colours. The two easiest are by name (see the "xkcd color survey" link from the tutorial) or as an RGB colour by providing a list of three numbers each between 0 and 1 (the amounts of red, green and blue light to include in the colour). If you don't know what an RGB colour is, see the Wikipedia article about RGB colours.

Here is an example of how to write a function to draw something in a specified colour:

```
def draw_segment(xy1, xy2, thecolour):
    plt.plot([xy1[0], xy2[0]], [xy1[1], xy2[1]], color=thecolour)

draw_segment([0,0], [1,1], 'pink')
draw_segment([-1,3], [2, -2], (1,.5,0)) # Orange line
draw_segment([-2,0], [2, 0.5], (0,0,1)) # Blue line
plt.show()
```

The example shows the definition and use of a function to add the line segment connecting points `xy1` and `xy2` to a matplotlib plot where the colour of the segment is specified by the `thecolour` argument to the function. The `thecolour` argument to the function is passed on to `plt.plot` using the named argument `color` that can be included in a call to `plt.plot`.

2.2 Intial plotting exercise

Task 1a (Math): Consider a line with the equation $y = mx + c$. You may assume m has finite value. Then consider a line segment of length l, with the leftmost point of the segment being the point $(x_0, mx_0 + c)$. Write down the coordinates of the of the rightmost point of the line segment in terms of m,c,x_0 and l.

Task 1b (Python):Write a Python function `draw_line(m,c,x0,ell)` that draws a segment of that line of length `ell`, with the leftmost point of the segment having the x value `x0`.

Here is an example of how it could be used (you may test your function using this example):

```
draw_line(1,0,0,2**0.5)
plt.show()
```

As mentioned previously, your function `draw_line(m,c,x0,ell)` should not include `plt.show()` in the function definition itself.

2.3 Drawing a circle in the plane

2.3.1 The unit circle (centered at the origin)

The unit circle in the plane is the circle of radius 1 centered at the origin. Points (x, y) on the unit circle satisfy the equation $x^2 + y^2 = 1$ and we can compute pairs of points using the equations $x = \cos\theta$ and $y = \sin\theta$ where θ ranges from 0 to 2π.

Task 2 (Python): Write a function `draw_unit_circle(thecolour)` that adds the unit circle to a matplotlib plot with the colour of the circle specified by the `thecolour` argument. Here is an example of how it could be used (you may test your function using this example):

```
draw_unit_circle('red')
plt.gca().set_aspect('equal', 'box')
plt.show()
```

Again, your function `draw_unit_circle(thecolour)` should not include `plt.show()` in the function definition itself.

2.3.2 A general circle and ellipse

Consider the circle of radius r centered at point (X, Y) in the plane.

The centre of the circle is a point in the plane and can therefore be represented as a list of two numbers. A circle may therefore be represented as list with two elements: the first element is the center of the circle and the second element is the radius of the circle. For example `[ [-0.3, 1.2], 2.3 ]` represents the circle of radius 2.3 centered at $(-0.3, 1.2)$. **In this question, and subsequent questions, you should not use** $plt.Circle$**. You should be able to make a circular plot using just** $plt.plot$**.** Remember how we plotted graphs in the practical sheets, and in particular that you can get the appearance of a curve by joining togehter enough points that are close enough to each other.

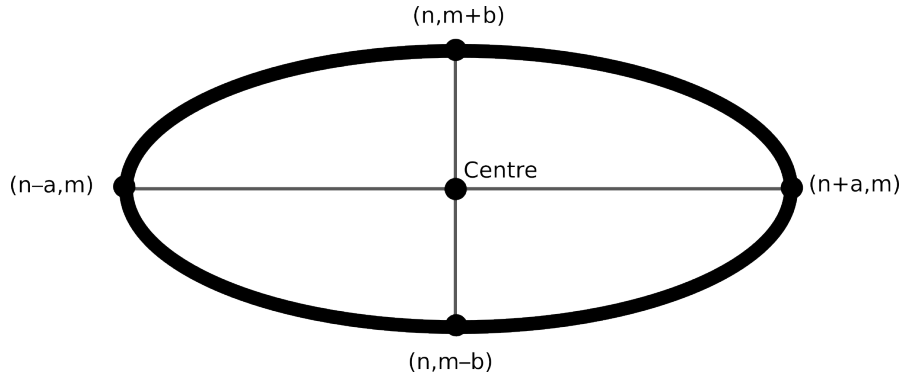
Task 3a (Math): Write down the equations for points on the general circle corresponding to equations given in section 2.3.1 for the unit circle centered at the origin.

Task 3b (Python): Write a function `draw_circle(circ, thecolour)` that adds the circle represented by `circ` to a matplotlib plot, drawn in colour `thecolour`. Here is an example showing how the function might be used:

```
circ1 = [ [2,1], 0.5 ]
circ2 = [ [-0.5,1.6], 1 ]
draw_circle(circ1, 'red')
draw_circle(circ2, [0,0,1])
plt.gca().set_aspect('equal', 'box')
plt.show()
```

Task 4 (Python): Test your function by writing a Python program that draws six circles each of radius $1/2$ with centres given by $(\cos(j\pi/3), \sin(j\pi/3))$ for $j = 0, 1, \ldots, 5$. When submitting your solution, this program should be in the form of a function `show_six_circles_example()`. Note that this function has no arguments (like the functions you submitted for quiz 8).

Task 5a (Math): Consider an ellipse in the $x - y$ plane as described in the diagram below, with co-vertices (points on the ellipse closest to the centre) labelled $(n, m + b)$ and $(n, m - b)$, and vertices (points on the ellipse furthest away from the centre) labelled $(n - a, m)$ and $(n + a, m)$.



Write an equation for the ellispe in terms of x, y, n, m, a and b.

Task 5b (Python): Write a function `draw_ellipse(n,m,a,b)` that adds the a plot of an ellipse described by the parameters n, m, a, and b that were defined in the mathematical part of the question.

2.3.3 Adding a tangent line

Given any circle and a point (x, y) on the circle, there is a unique line through the point that is a tangent to the circle.

Task 6a (Math): for the point (x, y) on the circle with radius r centered at (X, Y), find the equation of the tangent line through the point. Hint: the tangent line is orthogonal to the diameter of the circle passing through the point (x, y).

Task 6b (Python): Write a function `draw_tangent(circ, xy, d, thecolour)`. Here `circ` represents a circle, `xy` a point (x, y) that should lie on the circle and `d` the length d of the tangent line segment to draw.

(a) The function first checks whether or not the point `xy` lies on the circle represented by `circ`:

- if you know about Python "exceptions", your function should raise and exception if the point does not lie on the circle;
- otherwise (you don't know about Python "exceptions"), your function should return `False` without drawing anything if the point does not lie on the circle.

In checking whether or not the point lies on the circle, you should bear in mind the limitations of floating point arithmetic.

(b) When the point lies on the circle, the function draws a tangent line segment of length d (in the same units as used to specify the size and position of the circle) in the specified colour. The point of contact with the circle should be the midpoint of the line segment.

This function should not draw the circle, just the tangent line.

Task 7 (Python): Test your functions produced so far by writing a Python program that produces a plot showing the circle of radius 2 centered at $(-1, 3)$ and the tangent line segment of length 4 to the circle at the point $(0.2, 4.6)$. When submitting your solution, this program should be in the form of a function `show_tangent_example()`.

2.4 Intersecting circles and radical axes

If two nonconcentric circles in the plane intersect, they do so either at one point (they are tangent to each other) or at two distinct points. In the latter case, the line through those two points is known as the radical axis.

2.4.1 Circles with centres on the x-axis

Consider two circles of radius r and R respectively and with corresponding centres $(0, 0)$ and $(c, 0)$.

Task 8a (Math): Determine a condition on r, R, and c that determines whether or not the circles intersect at two points. For situations in which they do so intersect, determine the coordinates of the points at which they intersect.

Task 8b (Python): Write a function `xcircles_and_radicalaxis(r, R, c, thecolours)` that adds the two circles to a matplotlib plot. In cases where they intersect at two distinct points, it should also draw the segment of the radical axis connecting the points of intersection. `thecolours` is a list of three colours to be used for the two circles and the radical axis segment.

2.4.2 General circles

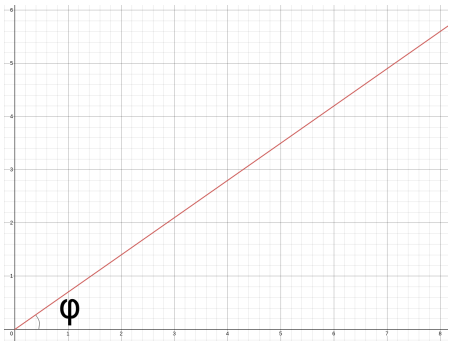
Task 9a (Math): Generalise task 8a to circles with arbitrary centres in the plane. You can do so in more than one way: brute force solution of the more general equations or by thinking about translating and rotating the plane to take advantage of your mathematics in section 2.4.1.

Task 9b (Python): Write a function `gencircles_and_radicalaxis(circ1, circ2, thecolours)` that adds two general circles represented by `circ1` and `circ2` to a matplotlib plot and adds the radical axis segment where appropriate. As in task 8b, `thecolours` specifies the colours to be used.

2.5 Pinball

The following material explores aspects of geometry that might be considered relevant to pinball with circular bumpers. But don't worry: you don't need to know anything about pinball in order to do this problem!

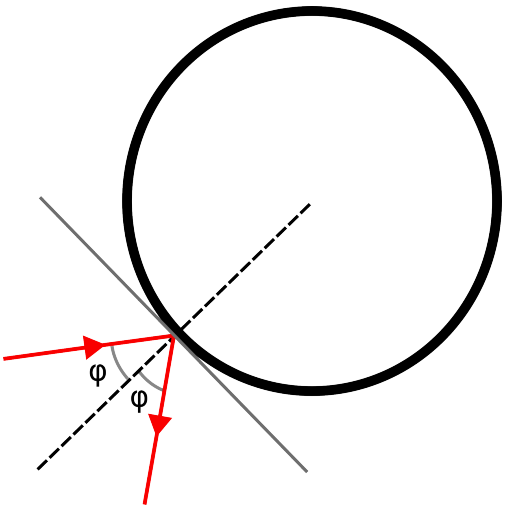
We will assume that a pinball of neglible size is launched from the origin at an angle ϕ from the positive x-axis, as demonstrated in the diagram below:

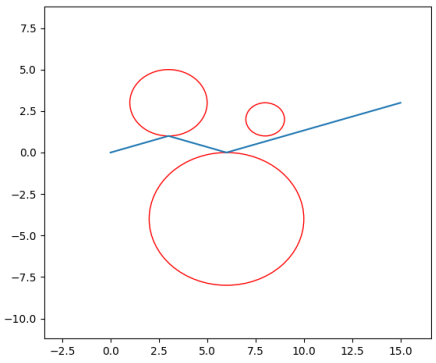


Bumpers on the pinball table may be thought of as circles in the plane. We can construct a track representing the motion of the pinball follows.

- We move in a straight line until we hit a circle.
- When we hit the circle, we bounce and again move in a straight line unless we hit another circle, in which case we bounce again.
- Each bounce is such that the angles of incidence and reflection are equal with respect to the normal to tangent line at the point where we hit the circle.

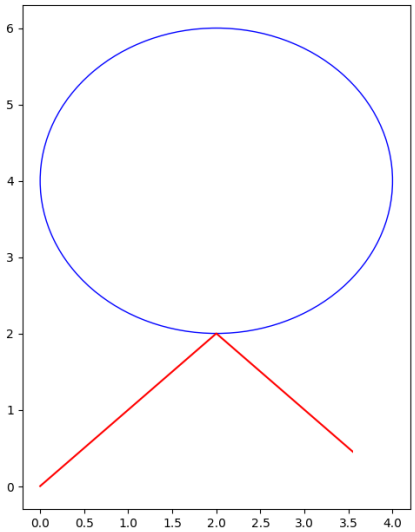
The basic idea is illustrated below (not drawn to scale). The detail of a bounce is shown in the left panel of the picture (the two angles labelled ϕ are the same). The panel on the right shows the track of a pinball on a table with three circular bumpers.





Task 10a (Math): Consider a track of a pinball starting at the orgin with an angle ϕ show that the track of the pinball is a fragment of the line with equation $x \sin\phi - y\cos\phi = 0$. Now consider the pinball track hitting a circular bumper with centre (x_0, y_0) and radius r (you may assume that the circle does not intersect or touch the origin). Consider the track representing the motion of the pinball after it bounces, and write down the equation of the line that it is a fragment of.

Task 10b (Python): Write a function `single_bumper(theta, circ, ell, circ_col, line_col)` that adds a circle of colour `circ_col` that is defined by list `circ` (as in section 2.3.2) and a track of length `ell` representing the motion of a pinball. If the track hits the circle, it should bounce. For example, `single_bumper(pi/4,[[2,4],2],5,'blue','red')` should add a plot that when displayed using `plt.show()` looks like:



Challenges on next page.

Challenges (likely to involve Math as well as Python):

- **Task 11**: Consider a situation where a pinball is fired from the orgin at angle θ on a table with two circular bumpers. Write a Python function `double_hit(circ1,circ2)` that takes as its arguments `circ1` and `circ2` – which describe two circular bumpers in the same notation as in section 2.3.2. If the two circles intersect or touch, the function should raise an exception or return `False`. Otherwise, the function should return a two list of elements which define the endpoints of the range of angles θ in which the pinball can fired so that it first hits the circle described by `circ1` and then the circle described by `circ2`. If this is not possible in the given geometry, the function should return the special Python value `None`.

- **Task 12**:Generalise the function in section 2.5 so that it can add a plot with an arbitary number of bumpers. Your function should be of the form `multi_bumper(theta, circs, l, circ_col, line_col)`, where `circs` is a list of lists defining the cicle.

  A valid `circs` would be `[[[3,3],2],[[6,-3],3],[[8,2],1]]`, which would describe three circles – one of centre $(3, 3)$ and radius 2, one of centre $(6, -3)$ and radius 3 and one of centre $(8, 2)$ and radius 1. If any of the circles touch or intersect, the function should not add a plot, but instead raise an exception or return the value `False`.

- **Task 13**: Further generalise the function in the previous task such that it can now plot elliptical as well as circular bumpers. The function should be of the format `multi_bumper2(theta, bumpers, l, circ_col, line_col)`, where `bumpers` is a list of lists defining the circular and ellitical bumpers in section 2.3.2. For example to define a circle of centre $(x0, y0)$ and radius r and an ellipse with vertices $(n - a, m)$ and $(n + a, m)$ and co-vertices $(n, m + b)$ and $(n, m - b)$ we could write `[[[x0,y0],r],[n,m,a,b]]` or `[[n,m,a,b],[[x0,y0],r]]`.