Homework 7

Abhigyan Chattopadhyay
ME19B001

October 29, 2019

Digitized Plotter

October 28, 2019

1 Homework 7 - Q1

We're going to plot some data from a CSV file that was obtained by digitization of this image taken from [1], using WebPlotDigitizer.

We're using matplotlib to plot the graph, and pandas to load the csv file into a python-compatible data structure, called DataFrame.

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

The below DataFrame, df, will hold the data in the CSV file...

[2]:
```
df = pd.read_csv("Digitized_Dataset.csv")
```

...until we extract its columns into two lists, x and y.

[3]:
```
x = df.iloc[:,0].tolist()
y = df.iloc[:,1].tolist()
```

Now, the MatPlotLib Magic!

(I've simply used the numpy `arange()` function, as I wanted to learn to make a range with float increments in Python)

The idea is that the complete plot will be a close replica of the original image. Hence, we start with plotting the data in the CSV file. The second plot has a – linestyle, which means it will be a dashed line.

Next we add the two plots we've generated so that they get plotted in the same graph, and not as two separate graphs.

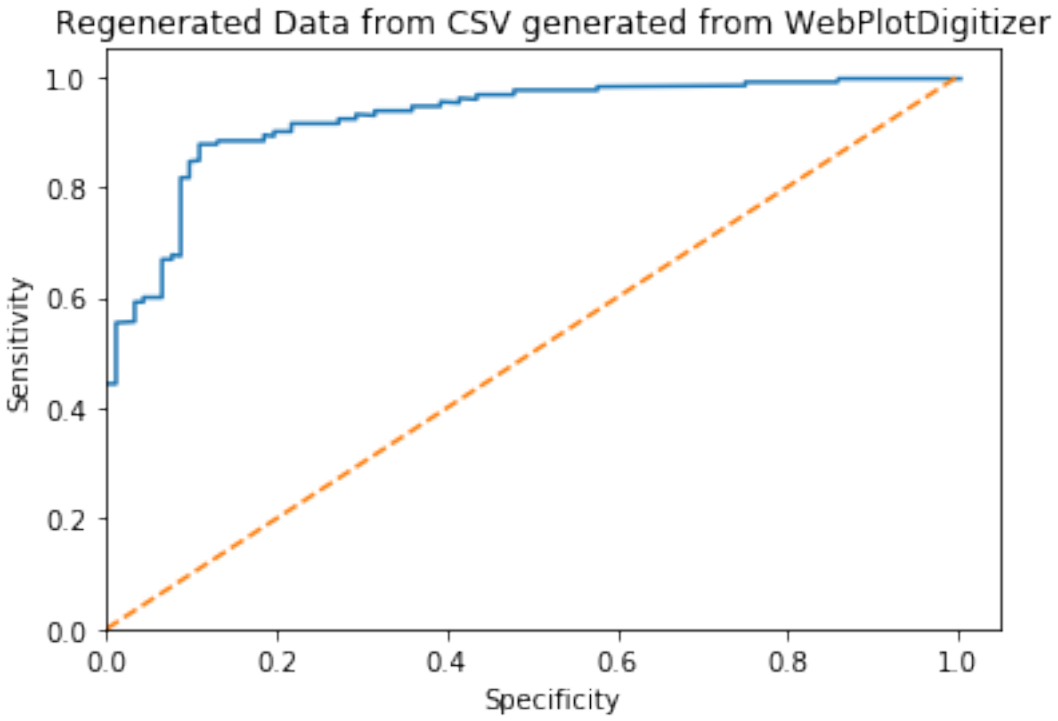
We want the plot to be limited to the range of the original plot, hence, we use the `plt.axis()` command, which specifies the x and y ranges.

Now, for the last part: Axis Labels and Graph Title!

`plt.xlabel()`, `plt.ylabel()`, and `plt.title()` take care of those for us, so yay! We've finished creating our plot!

[4]:
```
r = plt.plot(x,y,label = "Regenerated from CSV")
s = plt.plot(np.arange(0,1.1,0.1),np.arange(0,1.1,0.1), linestyle = '--')
z = r+s
plt.axis([0,1.05,0,1.05])
plt.xlabel("Specificity")
plt.ylabel("Sensitivity")
plt.title("Regenerated Data from CSV generated from WebPlotDigitizer")
```

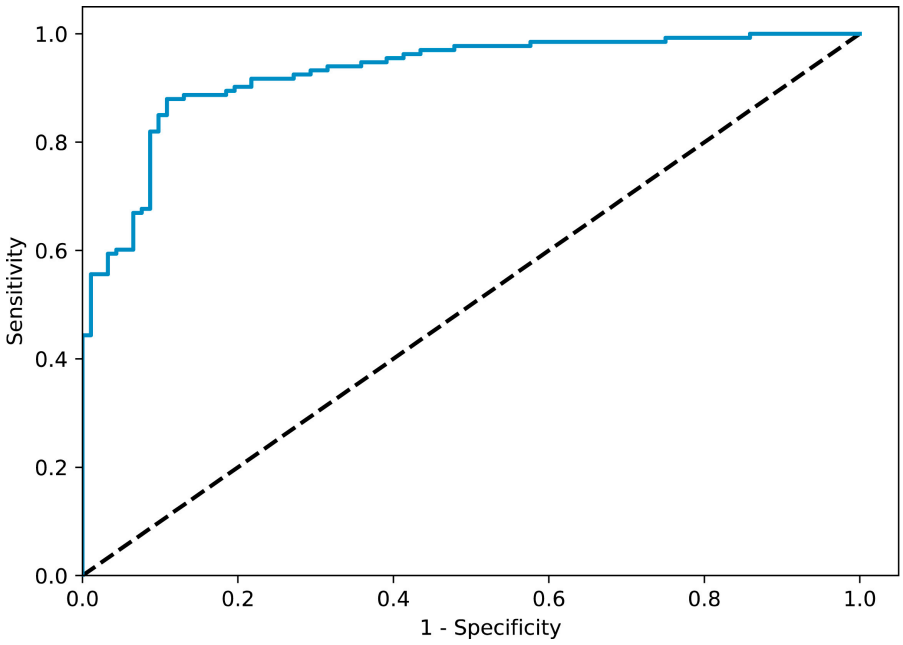
[4]:
```
Text(0.5, 1.0, 'Regenerated Data from CSV generated from WebPlotDigitizer')
```



Now the original image is also given below for reference.

[5]:
```
from IPython.display import Image
Image(filename='Line_plot.jpg')
```

[5]:



[1]: Hekler, Achim, et al. ``Superior skin cancer classification by the combination of human and artificial intelligence.'' European Journal of Cancer 120 (2019): 114-121.

Parametric

October 28, 2019

1 Homework 7 - Q4

We're going to plot some serious Parametric Equations now...

Get ready to behold the beauty of some math (Don't cringe!)

I've used Wikipedia extensively for this one, especially for the Lissajous Figures, Hypotrochoid and Epicycloid.

Among imports, we're mostly going to need `matplotlib` for plotting, `numpy` to generate float ranges, using the `arange()` function I learnt last time, and the regular `math` package to use `cos` and `sin` functions and `pi`'s value...

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
import math
```

1.1 Parabola

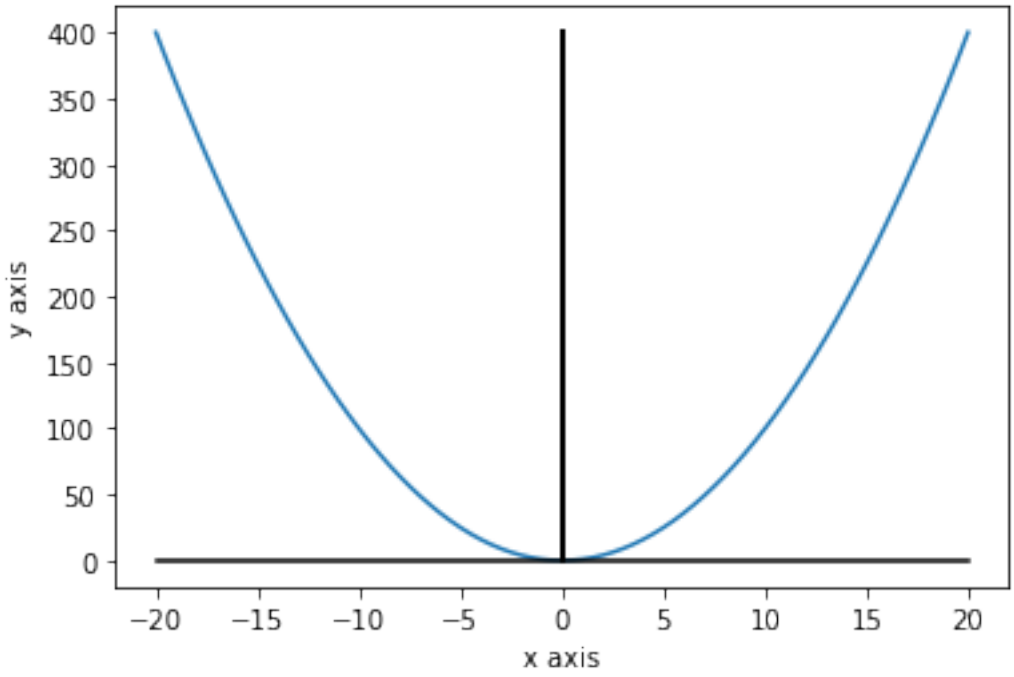
This one's really simple, all we're going to do is produce a set of values of x, and then create our y using the cool `map` function and the amazing `lambda` operator available in Python.

[2]:
```
x = np.arange(-20,20.5,0.5)
y = list(map(lambda t:t*t, x))
```

Next, we're going to plot the Parabola. Just to make it look better, I'm also plotting the values of x and y while keeping the other one fixed as zero, just to make it look like an x-axis and a y-axis.

[3]:
```
z = plt.plot(x,y,label='Parabola, Parametrized in terms of x')
xaxis = plt.plot(x,len(x)*[0], color='black')
yaxis = plt.plot(len(y)*[0],y,color='black')
plt.xlabel('x axis')
plt.ylabel('y axis')
```

[3]:
```
Text(0, 0.5, 'y axis')
```



1.2 Circle

This one is where we're first going to use the cos() and sin() functions.

Now, we're going to create a list of values of θ from 0 to 2π, using the `arange` function again.

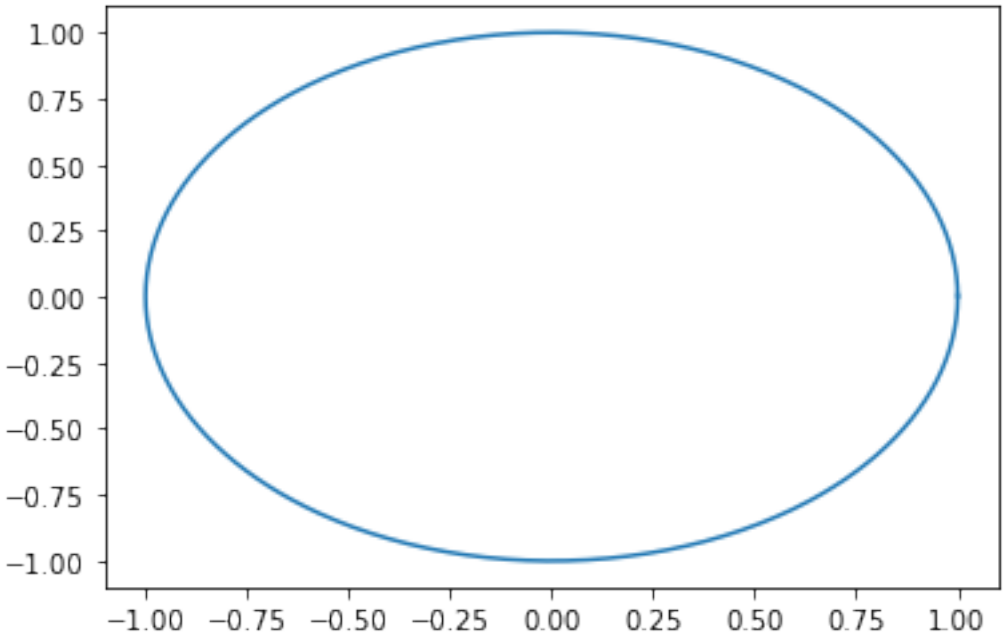
[4]:
```
t = np.arange(0,2*math.pi+0.01, 0.01)
```

We need to use the `plt.axis('equal')` command to make a circle look like an actual circle, as `matplotlib` likes to squeeze and stretch our data when we don't use it.

See the output before and after:

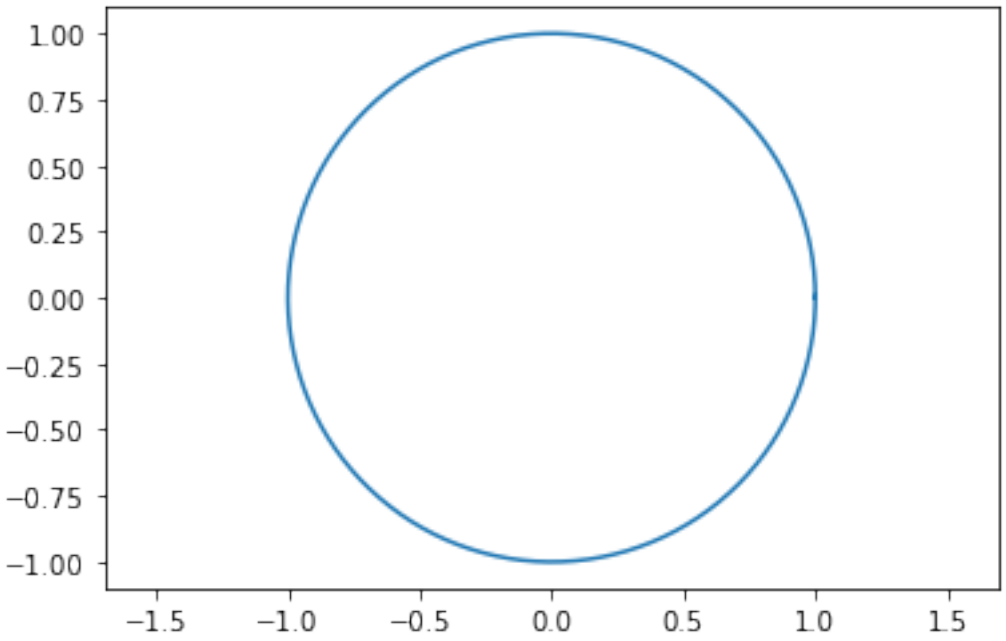
[5]:
```
plt.plot(list(map(lambda x:math.cos(x),t)),list(map(lambda x:math.sin(x),t)))
print("Before:")
```

```
Before:
```



[6]:
```
plt.axis('equal')
plt.plot(list(map(lambda x:math.cos(x),t)),list(map(lambda x:math.sin(x),t)))
print("After:")
```

```
After:
```

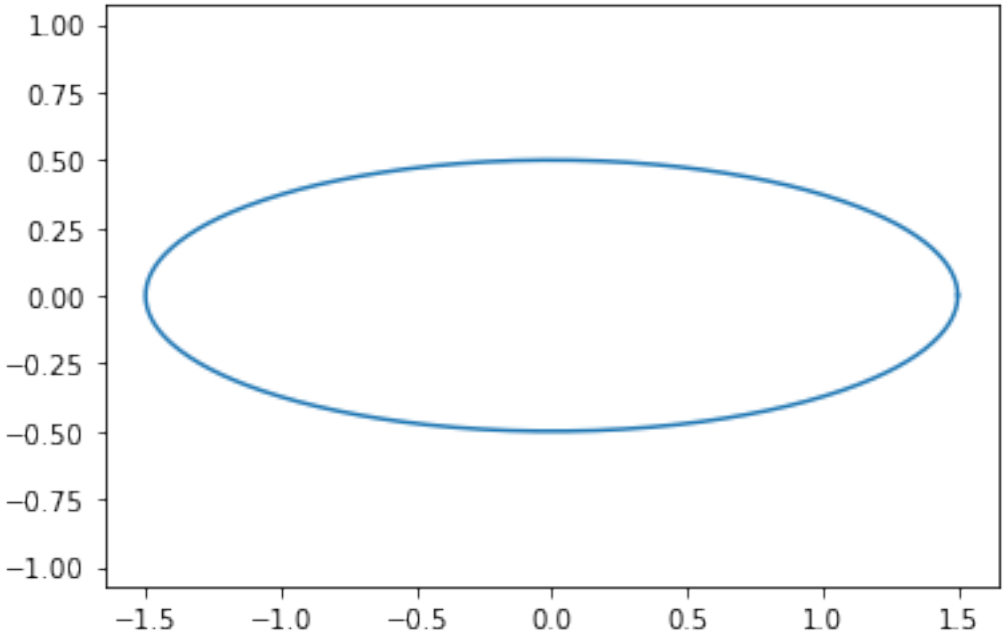


1.3 Ellipse

Pretty much the same thing as a circle... Honestly, I just copied the code, and scaled x and y :)

[7]:
```
plt.axis('equal')
plt.plot(list(map(lambda x:1.5*math.cos(x),t)),list(map(lambda x:0.5*math.
↪sin(x),t)))
```

[7]:
```
[<matplotlib.lines.Line2D at 0x7f0d4b5267f0>]
```



1.4 Lissajous Curves:

This is where the challenge slightly increases. At least in terms of understanding...

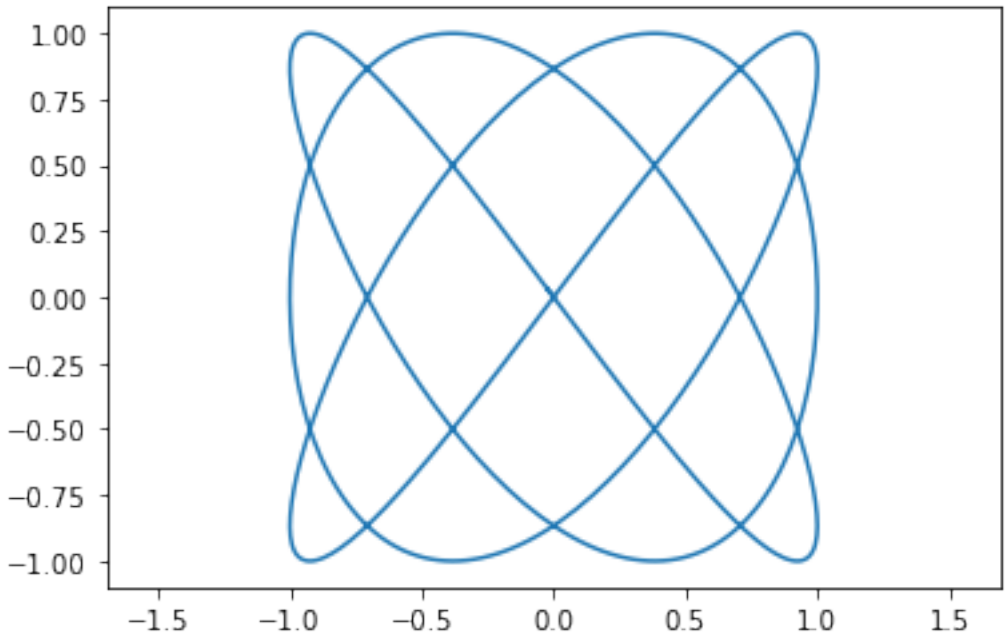
A Lissajous Curve is a curve that is of the following form:

$$x = A\cos(at + \delta), y = B\sin(bt)$$

Now, in my case, I took $A = 1, B = 1, \delta = \frac{\pi}{2}, a = 3$ and $b = 4$

[8]:
```
plt.axis('equal')
plt.plot(list(map(lambda x:math.cos(3*x+math.pi/2),t)),list(map(lambda x:math.
↪sin(4*x),t)))
```

[8]: [<matplotlib.lines.Line2D at 0x7f0d4b493898>]



1.5 Hypotrochoid

The equation of a Hypotrochoid is as follows:

$$x(t) = (R-r)\cos(t) + d\cos\left(\frac{R-r}{r}t\right), y = (R-r)\sin(t) - d\sin\left(\frac{R-r}{r}t\right)$$

I've chosen $R = 15$, $r = 7$ and $d = 4$.

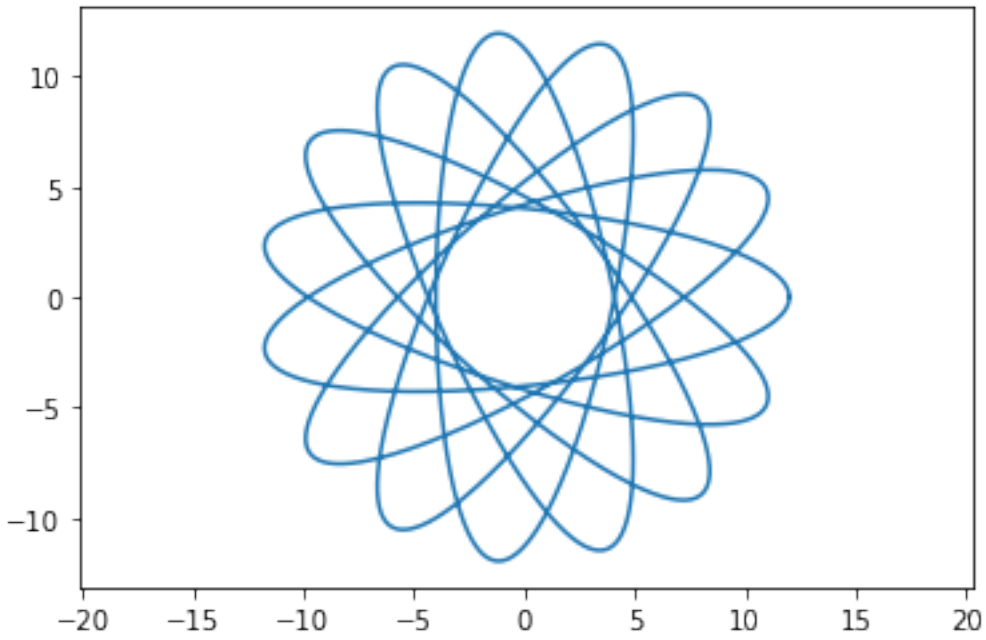
(It's a little better if it can be visualised while moving, for which I recommend: https://www.desmos.com/calculator/3plby3pgqv)

(Note that I've taken t to range from 0 to 30π, since the range of the parameter is given by:

$$[0, 2\pi \times \frac{LCM(r,R)}{R}] = [0, 14\pi]$$

[9]:
```
plt.axis('equal')
R = 15
r = 7
d = 4
u = np.arange(0,14*math.pi+0.01,0.01)
plt.plot(list(map(lambda x: (R-r)*math.cos(x)+d*math.cos((R-r)/r *␣
 ↪x),u)),list(map(lambda x: (R-r)*math.sin(x)-d*math.sin((R-r)/r * x),u)))
```

[9]: [<matplotlib.lines.Line2D at 0x7f0d4b47fb70>]



1.6 Epicycloid

The equation of an Epicycloid is as follows:

$$x(t) = (R+r)\cos(t) - r\cos\left(\frac{R+r}{r}t\right), y = (R+r)\sin(t) - r\sin\left(\frac{R+r}{r}t\right)$$

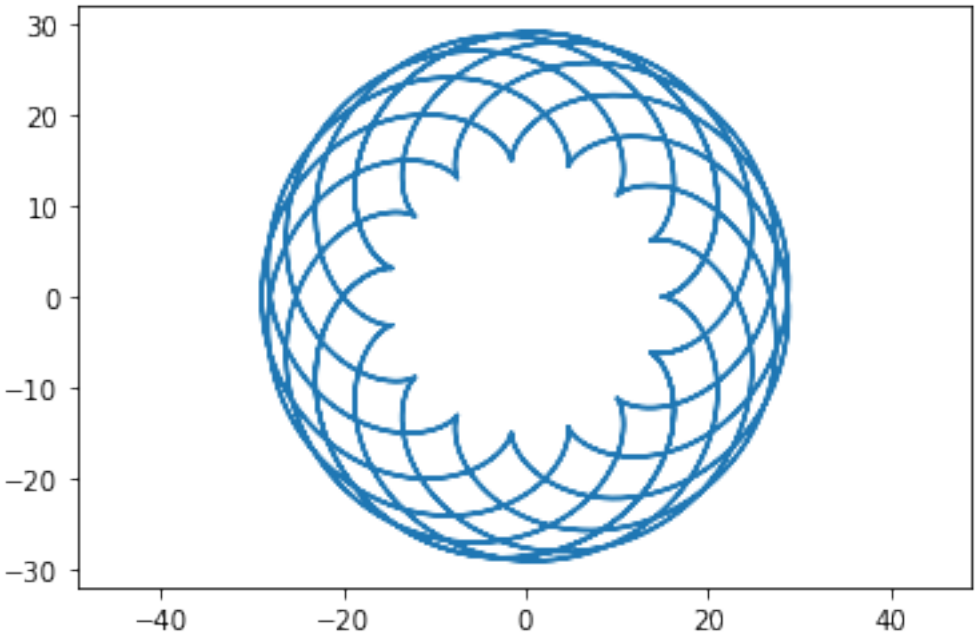
I've chosen $R = 15$ and $r = 7$.

(Note that I've taken t to range from 0 to 30π, since the range of the parameter is given by:

$$[0, 2\pi \times \frac{LCM(r,R)}{r}] = [0, 30\pi]$$

[10]:
```
plt.axis('equal')
R = 15
r = 7
u = np.arange(0,30*math.pi+0.01,0.01)
plt.plot(list(map(lambda x: (R+r)*math.cos(x)-r*math.cos((R+r)/r *␣
↪x),u)),list(map(lambda x: (R+r)*math.sin(x)-r*math.sin((R+r)/r * x),u)))
```

[10]: [<matplotlib.lines.Line2D at 0x7f0d4b408da0>]


30
20
10
0
−10
−20
−30
−40
−20
0
20
40

Envelope Finder

October 29, 2019

1 Homework 7 - Q5

Now, we're going to try to apply the finite difference method and plot some stuff to show that the envelope of tangents drawn at each point is going to give us the original function's outline.

This time round, we're going to use only `matplotlib`, as the maths is simple enough to be done without the use of any other packages.

[1]:
```
import matplotlib.pyplot as plt
```

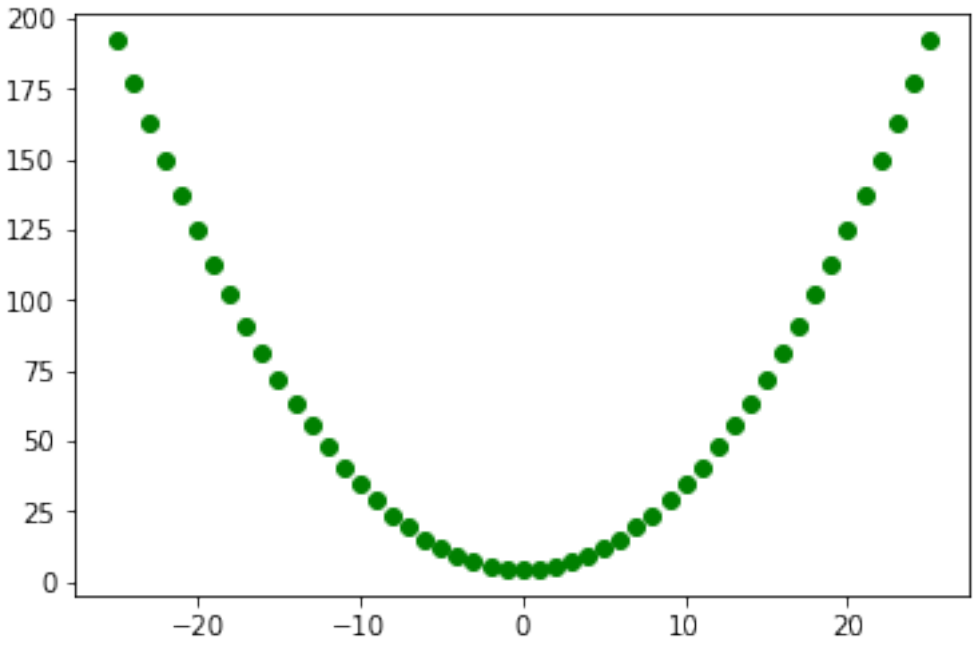
Generating our list of points to be from $x \in (-25, 25)$ to the corresponding y's such that: $y = 0.3x^2 + 4.5$

[2]:
```
pts=[list(range(-25,26)),list(map(lambda t:0.3*t*t+4.5,list(range(-25,26))))]
```

Now, plotting the x and y points as green filled circles

[3]:
```
plt.plot(pts[0],pts[1],'go')
```

[3]:
```
[<matplotlib.lines.Line2D at 0x7f16e59d8780>]
```



1.1 Using the Finite Difference Method

In principle:

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x - \Delta x)}{2\Delta x}$$

Hence, we can say that, approximately:

$$f'(x) = \frac{f(x + 1) - f(x - 1)}{2}$$

Now, what we're doing is we're evaluating the value of f at two points, the one ahead and the one behind the point at which we want to calculate the derivative. Now, we'll find these values and store them in a list of values (basically, this becomes our m array).

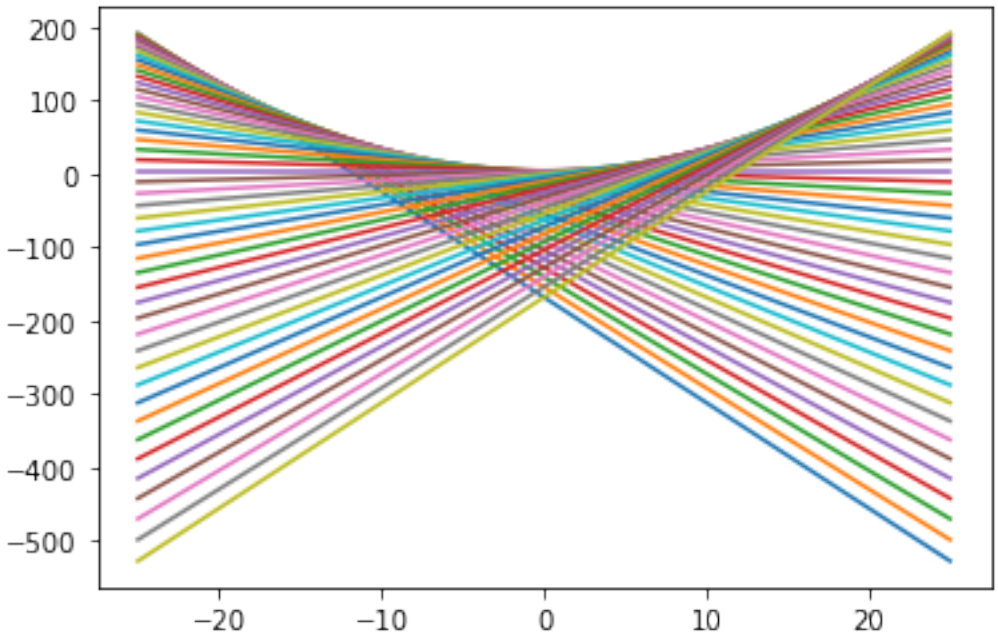
Next, we use some simple coordinate geometry to find the value of c for a given m, using the concept of a straight line. This is done taking the fact that $y = mx + c$, and hence $c = y - mx$.

Thus, at each point, we can evaluate the c_i value by taking the value of $y_i - m_i x_i$

[4]:
```
mc = [[],[]]
for i in range(1,50):
    m = ((pts[1][i+1]-pts[1][i-1])/(2))
    mc[0].append(m)
    mc[1].append(pts[1][i]-m*pts[0][i])
```

Now, plotting all the lines we've just created over the same x-array, again, using the `map` and `lambda` functions we have in Python.

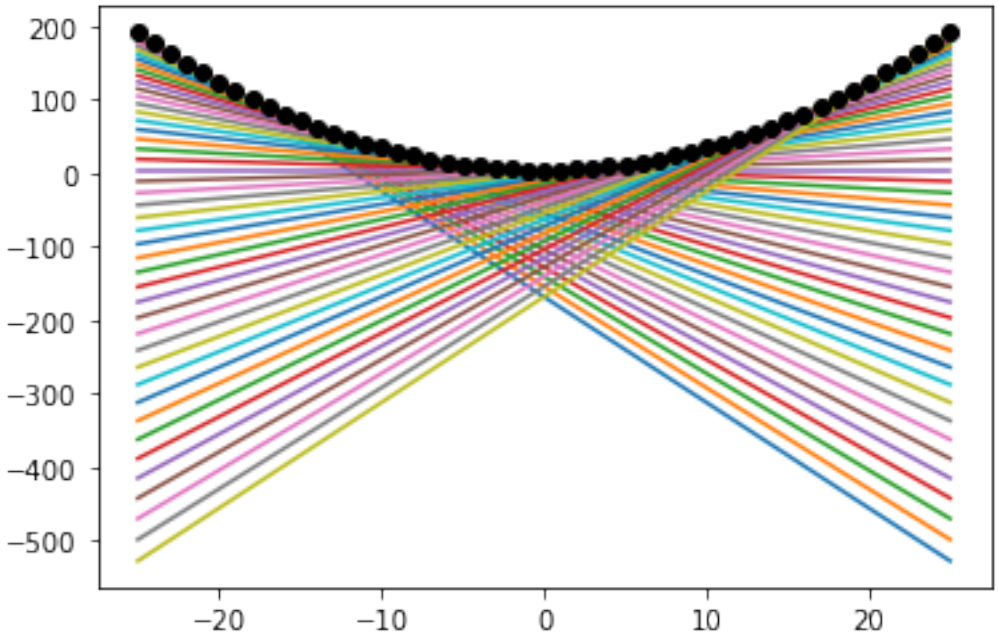
[5]:
```
for i in range(len(mc[0])):
    plt.plot(pts[0],list(map(lambda t: mc[0][i]*t+mc[1][i],pts[0])))
```



Now, superposing the above plot with the original plot of x and y to show that it's actually the envelope of the function we have chosen.

[6]:
```
for i in range(len(mc[0])):
    plt.plot(pts[0],list(map(lambda t: mc[0][i]*t+mc[1][i],pts[0])))
plt.plot(pts[0],pts[1],'o',color='black')
```

[6]:
```
[<matplotlib.lines.Line2D at 0x7f16e5844a20>]
```



Just showing the points (x, y) taken:

[7]:
```
for i in range(len(pts[0])):
    print(pts[0][i],",",pts[1][i])
```

```
-25 , 192.0
-24 , 177.29999999999998
-23 , 163.2
-22 , 149.7
-21 , 136.79999999999998
-20 , 124.5
-19 , 112.8
-18 , 101.69999999999999
-17 , 91.19999999999999
-16 , 81.3
-15 , 72.0
-14 , 63.300000000000004
-13 , 55.199999999999996
-12 , 47.699999999999996
-11 , 40.8
-10 , 34.5
-9 , 28.799999999999997
-8 , 23.7
-7 , 19.200000000000003
-6 , 15.299999999999999
-5 , 12.0
```

```
-4 , 9.3
-3 , 7.199999999999999
-2 , 5.7
-1 , 4.8
0 , 4.5
1 , 4.8
2 , 5.7
3 , 7.199999999999999
4 , 9.3
5 , 12.0
6 , 15.299999999999999
7 , 19.200000000000003
8 , 23.7
9 , 28.799999999999997
10 , 34.5
11 , 40.8
12 , 47.699999999999996
13 , 55.199999999999996
14 , 63.300000000000004
15 , 72.0
16 , 81.3
17 , 91.19999999999999
18 , 101.69999999999999
19 , 112.8
20 , 124.5
21 , 136.79999999999998
22 , 149.7
23 , 163.2
24 , 177.29999999999998
25 , 192.0
```

Now, showing the values of (m, c) for each point (x, y) (except the first and last ones, as their slope can't be defined without other information).

[8]:
```
for i in range(len(mc[0])):
    print(mc[0][i],",",mc[1][i])
```

```
-14.400000000000006 , -168.30000000000015
-13.799999999999997 , -154.19999999999993
-13.200000000000003 , -140.7000000000001
-12.599999999999994 , -127.79999999999993
-11.999999999999993 , -115.49999999999986
-11.400000000000006 , -103.80000000000011
-10.800000000000004 , -92.7000000000001
-10.199999999999996 , -82.19999999999993
-9.599999999999994 , -72.29999999999991
-8.999999999999996 , -62.99999999999994
-8.400000000000002 , -54.30000000000002
```

-7.800000000000004 , -46.20000000000007
-7.199999999999999 , -38.699999999999996
-6.599999999999998 , -31.799999999999983
-6.0 , -25.5
-5.4 , -19.800000000000004
-4.799999999999997 , -14.699999999999978
-4.2 , -10.2
-3.6000000000000014 , -6.30000000000001
-2.999999999999999 , -2.9999999999999964
-2.4000000000000004 , -0.3000000000000007
-1.8000000000000003 , 1.799999999999999
-1.1999999999999997 , 3.3000000000000007
-0.6000000000000001 , 4.199999999999999
0.0 , 4.5
0.6000000000000001 , 4.199999999999999
1.1999999999999997 , 3.3000000000000007
1.8000000000000003 , 1.799999999999999
2.4000000000000004 , -0.3000000000000007
2.999999999999999 , -2.9999999999999964
3.6000000000000014 , -6.30000000000001
4.2 , -10.2
4.799999999999997 , -14.699999999999978
5.4 , -19.800000000000004
6.0 , -25.5
6.599999999999998 , -31.799999999999983
7.199999999999999 , -38.699999999999996
7.800000000000004 , -46.20000000000007
8.400000000000002 , -54.30000000000002
8.999999999999996 , -62.99999999999994
9.599999999999994 , -72.29999999999991
10.199999999999996 , -82.19999999999993
10.800000000000004 , -92.7000000000001
11.400000000000006 , -103.80000000000011
11.999999999999993 , -115.49999999999986
12.599999999999994 , -127.79999999999993
13.200000000000003 , -140.7000000000001
13.799999999999997 , -154.19999999999993
14.400000000000006 , -168.30000000000015