Week 12: More plotting

Intended learning outcomes

By the end of this class, you will be able to:

- draw and save parametrically defined curves and scatter plots;
- define graphical elements using functions.

1 Parametric plotting

Last week we saw how to plot a function $y = f(x)$ using NumPy arrays and Matplotlib. We defined a NumPy array of x-values, `x_values`, and defined the array of y-values using `f(x_values)`, so the graph could be plotted with `plt.plot(x_values, f(x_values))`. Although slightly counterintuitive to start with, it makes sense to call the array of x-values just `x` so that the plot command becomes just `plt.plot(x, f(x))`.

You will now try plotting some parametrically defined curves. Recall that the points on the unit circle are given parametrically by

$$(\cos(\theta), \sin(\theta)) \quad \text{for } \theta \in [0, 2\pi].$$

To get Python to plot this, type in and run the following program:

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  theta = np.linspace(0, 2*np.pi, 100)
5  plt.plot(np.cos(theta), np.sin(theta))
6  plt.gca().set_aspect("equal")
7  plt.show()
```

You should see a circle.

Line 4 generates an array of 100 values for θ between 0 and 2π. Remember that `theta` will be an array of values, rather than a single value.
Line 5 provides the `plt.plot()` function with the appropriate arrays of x-values and y-values.
Line 6 makes sure that the scaling on the x- and y-axes is the same. Comment out this line with `#` and see how the circle gets deformed.
Line 7 displays the current figure.

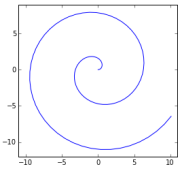
The circle looks a little cramped inside the axes, so we can tell Python to let the axes to be a little bigger. Add the following line before `plt.show()`.

```
plt.axis([-1.2, 1.2, -1.2, 1.2])
```

What are the four numbers doing there? Change them to find out.

Exercise 12.1.

1. Plot an ellipse.
2. The Archimedean spiral has the parametric form $(\theta\cos(\theta), \theta\sin(\theta))$. As accurately as possible, get Python to draw this picture:



2 Varying parameters with `for` loops

Plotting can be combined with `for` loops to various effects. For example, we can consider the parametric equation of a curve:

$$x = \tfrac{3}{2}\cos(\theta) - \cos(k\theta), \quad y = \tfrac{3}{2}\sin(\theta) - \sin(k\theta), \quad \text{for } \theta \in [0, 2\pi],$$

where $k \in \{2, 3, 4, \dots\}$ is a constant.

```
import numpy as np
import matplotlib.pyplot as plt

theta = np.linspace(0, 2*np.pi, 200)
for k in range(2, 7):
    plt.plot(1.5*np.cos(theta) - np.cos(k*theta),
             1.5*np.sin(theta) - np.sin(k*theta))
    plt.gca().set_aspect("equal")
    plt.axis([-3, 3, -3, 3])
    plt.show()
```

You can plot the graphs all on the same axes if you unindent the last three lines, thus removing them from the `for` loop.

You can build up repetitive elements in a picture using loops. Suppose that you want to draw lots of spokes in a wheel. This means that you want to draw a line from $(0, 0)$ to $(\cos(\phi), \sin(\phi))$ for various values of ϕ. Recall that to use `plt.plot()` to draw a line from `(x0, y0)` to `(x1, y1)`, you need to provide x-values `[x0, x1]` and y-values `[y0, y1]`. We will plot N spokes by taking $\phi = 2\pi i/N$ for $i = 0, 1, \dots, N-1$.

```
import numpy as np
import matplotlib.pyplot as plt

NO_OF_SPOKES = 17
for i in range(NO_OF_SPOKES):
    phi = i * 2 * np.pi / NO_OF_SPOKES
    plt.plot([0, np.cos(phi)], [0, np.sin(phi)], color="red")

plt.gca().set_aspect("equal")
plt.axis([-1.1, 1.1, -1.1, 1.1])
plt.show()
```

Exercise 12.2. Draw the outline of a chess board, starting with the following program.

```
import matplotlib.pyplot as plt

plt.plot([0, 0], [0, 8], color="blue")
plt.plot([1, 1], [0, 8], color="blue")
plt.plot([0, 8], [0, 0], color="blue")
plt.plot([0, 8], [1, 1], color="blue")

plt.gca().set_aspect("equal")
plt.axis([-1, 9, -1, 9])
plt.show()
```

Make sure you know why it is drawing each of the lines correctly. You could continue in this way, writing out a total of eighteen (why eighteen?) `plt.plot()` commands to draw an eight-by-eight block of squares. Instead, use just two `plt.plot()` commands and either one or two `for` loops to draw an eight-by-eight block of squares.

3 Defining graphical elements using functions

If you are doing the same thing, such as drawing a circle, in different situations then it makes sense to define and re-use a function for this. Below, the function `draw_circle()` takes two parameters: a list with two elements for the coordinates of the centre, and a number for the radius. Recall that the parametric form for a circle with centre (x, y) and radius r is $(r\cos(\theta) + x, r\sin(\theta) + y)$.

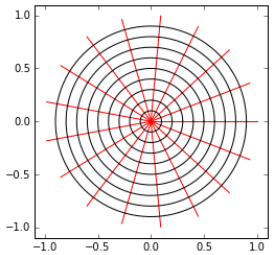
```
import numpy as np
import matplotlib.pyplot as plt


def draw_circle(centre, radius):
    """Draw a black circle with specified centre and radius."""
    theta = np.linspace(0, 2*np.pi, 100)
    plt.plot(radius*np.cos(theta) + centre[0],
             radius*np.sin(theta) + centre[1],
             color="black")


draw_circle([0, 0], 5)
draw_circle([0, 0], 1)
draw_circle([3, 0], 2)
draw_circle([-3, 0], 2)
draw_circle([0, 3], 2)
draw_circle([0, -3], 2)

plt.gca().set_aspect("equal")
plt.show()
```

Exercise 12.3. Add the `draw_circle()` function to the spoke drawing program above and add a **`for`** loop to draw nine circles to produce this spider's web:



4 Saving your graphs as files

You might wish to save your plots so you can include them in webpages or LaTeX documents. Replace `plt.show()` with the following to save a plot as a `png` file.

```
plt.savefig("picture.png")
```

This is saved in the working directory. Open that directory and view the graph. If using Google Colab, click on the 'Files' icon to the left of the window to find where the file has been saved; to save the file to your Google Drive, click on the 'Mount Drive' icon and follow the on-screen instructions.

Exercise 12.4. Alter one of your programs so that it saves the picture as a file. Find the file and make sure you can view it.

For different purposes you should use different types of graphic files: for a webpage, save as either a `png` or an `svg` file; for a LaTeX document, save as a `pdf` or `eps` file.

Python will automatically detect which format to save as based on the extension you give the filename, such as `.png` or `.svg`.

To reduce the amount of padding round the graph, use the following variant:

```
plt.savefig("picture.png", bbox_inches="tight")
```

5 Scatter plots

To draw points, not lines, the command `plt.scatter(x, y, c=color, s=size)` can be used. This command takes four lists or Numpy arrays as arguments: `x` and `y` store the coordinates of the points to plot, while `colour` and `size` are lists whose ith entries store the colour and size of the ith point.

The command `np.random.rand(N)` produces a Numpy array of length `N` whose entries are random numbers between 0 and 1, so the following code produces two arrays, `x` and `y`, with entries between -1 and 1. It colours the ith point, (x_i, y_i), red if $(x_i)^{2/3} + (y_i)^{2/3} < 1$ and blue otherwise.

```
import matplotlib.pyplot as plt
import numpy as np

N = 20
x = 2*np.random.rand(N) - 1
y = 2*np.random.rand(N) - 1
size = [5] * N
colour = ['blue'] * N
for i in range(N):
    if (x[i]**2)**(1/3) + (y[i]**2)**(1/3) < 1:
        colour[i] = ('red')

plt.scatter(x, y, c=colour, s=size)
plt.axis([-1.1, 1.1, -1.1, 1.1])
plt.gca().set_aspect("equal")
plt.show()
```

Run this a couple of times to see if you can spot the pattern that is formed. Now change the number of points from 20 to 200 and repeat. Then try for 2000 and 20000 points. The outline of the resulting figure is called an *astroid* curve.

Exercise 12.5. Alter the code to plot blue dots with size 10 and red with size 1.

Homework

1. Finish the sheet.
2. (*Quick review.*) What will the following programs output? Decide then try.

   (i)
   ```
   import numpy as np
   x = np.linspace(0, 1, 5)
   print(x)
   print(20*x)
   ```

   (ii)
   ```
   import matplotlib.pyplot as plt
   plt.plot([0, 0, 1, 1], [1, 0, 0, 1], color='red')
   plt.axis([-1, 2, -1, 2])
   plt.show()
   ```

3. Revise for Friday's Python test.
4. (*Optional.*) Write a program which draws 20 random stars, something like as shown to the right.