Activity 1

The System of ODEs

$$\frac{dx_1}{dt} = -\frac{\Gamma_2(y_1 - y_2)}{d_{12}^2}$$

$$\frac{dy_1}{dt} = \frac{\Gamma_2(x_1 - x_2)}{d_{12}^2}$$

$$\frac{dx_2}{dt} = -\frac{\Gamma_1(y_2 - y_1)}{d_{12}^2}$$
$$\frac{dy_2}{dt} = \frac{\Gamma_1(x_2 - x_1)}{d_{12}^2}$$

Solving these

It is clear to see (HINT):
$$\Gamma_1\frac{dx_1}{dt} = -\Gamma_2\frac{dx_2}{dt}$$
$$\Gamma_1\frac{dy_1}{dt} = -\Gamma_2\frac{dy_2}{dt}$$

Integrate both sides with respect to t, to get:

1. $\Gamma_1 x_1 = -\Gamma_2 x_2 + \alpha$
2. $\Gamma_1 y_1 = -\Gamma_2 y_2 + \beta$

We can now see, we have:
$$x_1 - x_2 = -\left(1 + \frac{\Gamma_2}{\Gamma_1}\right)x_2 + \frac{\alpha}{\Gamma_1} = -\left(\frac{\Gamma_1 + \Gamma_2}{\Gamma_1}\right)x_2 + \frac{\alpha}{\Gamma_1}$$

Similarly:
$$y_1 - y_2 = -\left(\frac{\Gamma_1 + \Gamma_2}{\Gamma_1}\right)y_2 + \frac{\beta}{\Gamma_1}$$

We can use these observations to now solve for (x_2, y_2). Thankfully, the d_{12} term is eliminated when we try to derive $\frac{dy_2}{dx_2}$. We have the following (HINT):

$$\frac{dy_2}{dx_2} = \frac{\frac{dy_2}{dt}}{\frac{dx_2}{dt}} = -\frac{x_1 - x_2}{y_1 - y_2} = -\frac{-(\Gamma_1 + \Gamma_2)x_2 + \alpha}{-(\Gamma_1 + \Gamma_2)y_2 + \beta}$$

We rearrange this, and then integrate with respect to x_2 on both sides:

$$\int -(\Gamma_1 + \Gamma_2)y_2 + \beta dy_2 = -\int -(\Gamma_1 + \Gamma_2)x_2 + \alpha dx_2$$
$$-\tfrac{1}{2}(\Gamma_1 + \Gamma_2)y_2^2 + \beta y_2 + C = -\left[-\tfrac{1}{2}(\Gamma_1 + \Gamma_2)x_2^2 + \alpha x_2\right]$$

Rearranging, we finally get:

3. $\frac{1}{2}(\Gamma_1 + \Gamma_2)(x_2^2 + y_2^2) - \alpha x_2 - \beta y_2 = C$

We can now spot that $\frac{dy_1}{dx_1} = \frac{dy_2}{dx_2}$, and that the relationships between (x_1, y_1) and (x_2, y_2) have a symmetrical form:
$\Gamma_2 x_2 = -\Gamma_1 x_1 + \alpha$
$\Gamma_2 y_2 = -\Gamma_1 y_1 + \beta$

Hence, we can just swap 1 by 2 in our final equation above to get:

4. $\frac{1}{2}(\Gamma_1 + \Gamma_2)(x_1^2 + y_1^2) - \alpha x_1 - \beta y_1 = D$

What's left now is to find the values of the constants.
Substituting our initial conditions in, we get:

$\alpha := \Gamma_1 a_1 + \Gamma_2 a_2$
$\beta := \Gamma_1 b_1 + \Gamma_2 b_2$
$C := \frac{1}{2}(\Gamma_1 + \Gamma_2)(a_2^2 + b_2^2) - \alpha a_2 - \beta b_2$
$D := \frac{1}{2}(\Gamma_1 + \Gamma_2)(a_1^2 + b_1^2) - \alpha a_1 - \beta b_1$

You can simplify these a little if you'd like, but it's not necessary.

What are the cases?

If $\Gamma_1 = -\Gamma_2$ then the squared terms disappear, and we get two lines, just with different y-intercepts, C and D (hence **parallel lines**).

If $\Gamma_1 = \Gamma_2$, we use 1 and 2 to write $x_2^2 + x_1^2$ and substitute it into 3. We make some more substitutions, and can follow through to get $C = D$. Since all the other coefficients are the same, we can conclude that the **circles traversed are the same**.

If $\Gamma_1 \neq \Gamma_2$, we just have **different circles**. These are always concentric because α, β and $\Gamma_1 + \Gamma_2$ are shared by both as coefficients.

Activity 2

```
def getInducedVelocity(self, otherPos):
        """
        Get the velocity contribution that this vortex
        induces at otherPos, as a tuple.
        """
        otherX, otherY = otherPos
        selfX, selfY = self.pos
        distSquared = (otherX - selfX) ** 2 + (otherY - selfY) ** 2
        if distSquared == 0:
                return (0, 0)
        else:
                return (
                        -self.circulation * (otherY - selfY) / distSquared,
                        self.circulation * (otherX - selfX) / distSquared,
                )

def computeVelocity(self, vortexArray):
        """
        Compute and set the velocity of this vortex by combining
        the contributions from all surrounding vortices.
        """
        self.velocity = (0, 0)
        for otherVortex in vortexArray:
                self.velocity = self.velocity + np.array(
                        otherVortex.getInducedVelocity(self.pos)
                )
```

```python
def move(self, timePeriod):
        """
        Move this vortex over the specified time
        period, updating its position.
        """
        self.pos = self.pos + timePeriod * np.array(self.velocity)
```

Things that are likely to go wrong:

- Not dealing with the case when the distance is 0 in `getInducedVelocity` (or alternatively, not excluding the vortex in question from the velocity calculation in `computeVelocity`).
- Not using numpy arrays to add tuples pointwise.
- The computation can be many times slower depending on how the `distanceSquared` is computed e.g. `np.array(self.pos)**2` is inefficient for some reason.
- Potentially not using `getInducedVelocity` in `computeVelocity`?
- I may not have communicated well enough the role of `timePeriod` in `move`.

Finished?

```python
def add_rule(self, char, string):
        """
        Add a new rule, specifying what a given
        character should be replaced with.
        """
        self.rules[char] = string

def update(self):
        """
        Apply all the rules to generate a new string.
        """
        new_string = ""
        for char in self.current_string:
                if char in self.rules.keys():
                        new_string += self.rules[char]
                else:
                        new_string += char

        self.current_string = new_string

def print(self):
        """
        Convert the current string into Turtle instructions,
        and draw the result.
        """
        for char in self.current_string:
                if char == "+":
                        self.turtle.right(self.angle_change)
                        self.turtle.forward(self.speed)
                elif char == "-":
                        self.turtle.left(self.angle_change)
                        self.turtle.forward(self.speed)
```