1 Code journal: Regular Polygon Spiral

21.07.2020, Mariana Ávalos Arce

1.1 What is constant and what is not?

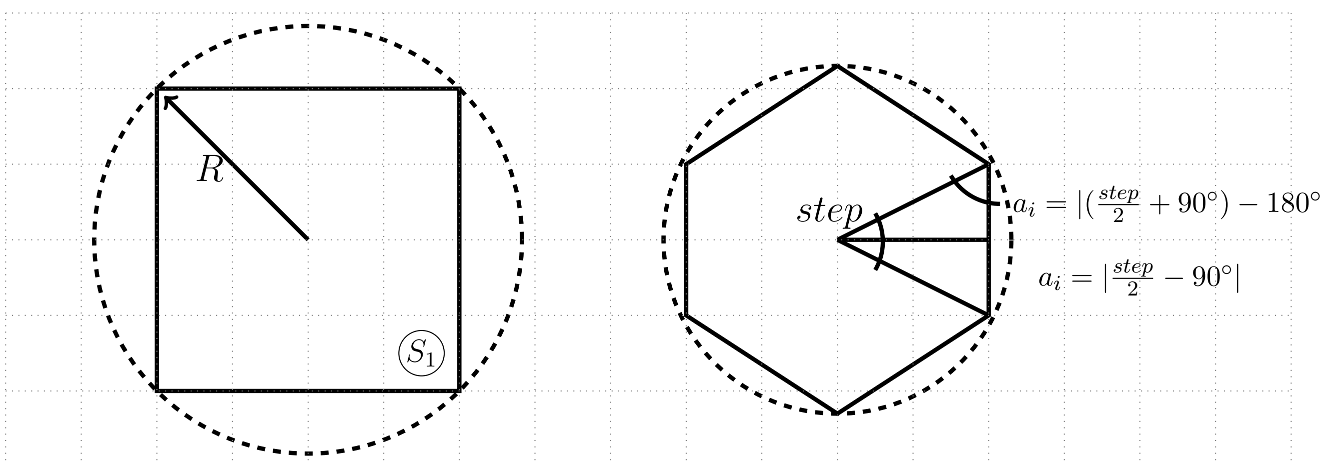
A **Geometric Spiral** is basically the well-known image where a regular polygon rotates and gets bigger and bigger as the image is filled. The image might look deceitful at times, which made me doubt: are both the angle and the size increasing, or just one of them?

1.2 A regular polygon

The first thing to build the image is to create the central **regular polygon**. To build any regular polygon, the whole 360 degrees must be divided by the number of sides in the desired polygon. During this report, the main example will be seen with a **square**, but it works for any type of regular polygon. Thus,

$$step_{angle} = \frac{360}{sides} = \frac{360}{4} = 90^\circ \tag{1}$$

Which tells us that every 90 degrees a polygon vertex is positioned. Thus, we just need a custom radius or polygon size to draw square 1 (S_1) inside a circumference of radius R (or size), following the ideas below. A simplified code for the mentioned concept is also shown below.




```
acc_angle = 0.0; acc_radius = radius
last_point = [0.0, 0.0]; first_point = [0.0, 0.0]
    for i in range(num_sides):
    angle = i * step_angle - (step_angle / 2.0) - acc_angle
    x = fig_center[0] + (acc_radius) * math.cos(angle * PI / 180.0)
    y = fig_center[1] + (acc_radius) * math.sin(angle * PI / 180.0)
    if i == 0:
            first_point = [x, y]
    if i > 0:
            ln.line([x, y], last_point, thick, color, img)
    if i == num_sides - 1:
            ln.line([x, y], first_point, thick, color, img)
    last_point = [x, y]
```

1.3 One polygon leads to another

The above mentioned algorithm results in the first polygon at the center of the image, which will be also the smallest. The following iterations of polygons in an external loop answer the first question: what is constant, the angle or the radius increase in each polygon? If we look below, the emerging lines are the Δa or the slight change in the rotation angle of each polygon, and we see that **each polygon follows a constant angle increase** when compared to the previous one.

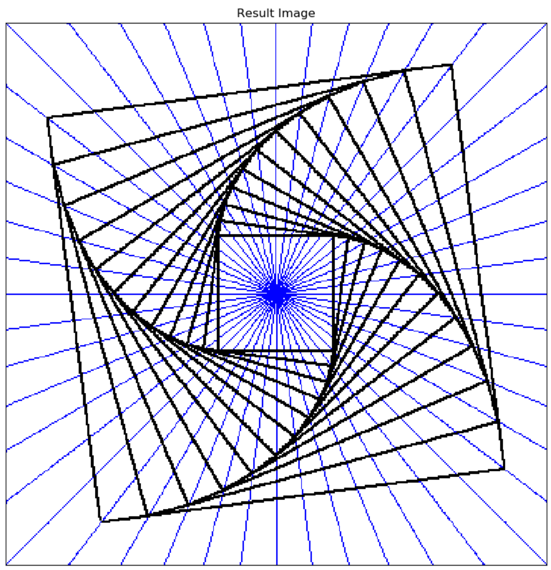
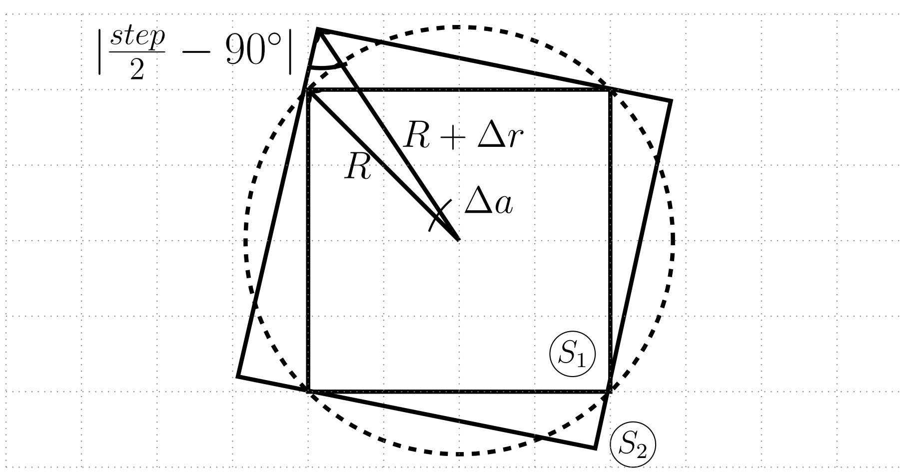


Figure 1: Test showing constant angle

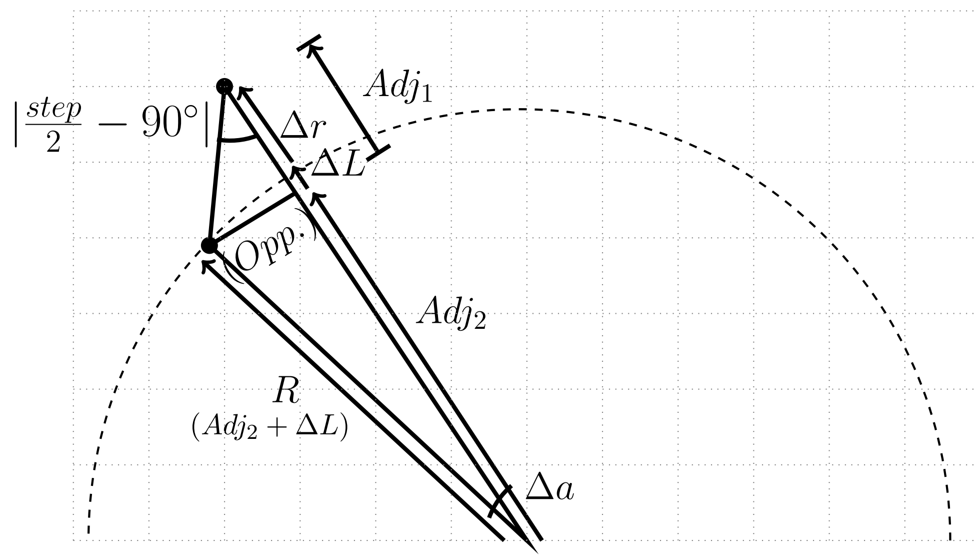
The test image also shows that each of the emerging lines slide the internal polygon angles in half, which is crucial to calculate the **increase in radius** (Δr) of each polygon so that its sides exactly touch the last polygon's sides. This increase in radius will then be **larger** in every polygon drawn. To explain this, an example of a two-square spiral is drawn in the diagram below.



1.4 Predicting the next size

In order to calculate the exact change in radius *for the next polygon* so that the sides touch each other exactly, we must take on account that we know the value of two crucial angles: the custom

Δa and the angle $|\frac{step}{2} - 90°|$. If we take the triangle and amplify it a bit as in the following diagram, we get a more evident discovery to know the value of Δr, and with this, the next (further) point will be given now that the new accumulated radius gets increased with the new value; allowing the radius value of the next polygon to be known at the *end* of each iteration.



Thus, taking into account some trigonometry (all angles must be converted to radians),

$$\text{Opp.} = \text{R} \times sin(\Delta a), \tag{2}$$

Which leads to

$$\frac{tan(|\frac{step}{2} - 90°|)}{1} = \frac{Opp.}{Adj_1}, \tag{3}$$

$$\frac{Adj_1}{Opp.} = \frac{1}{tan(|\frac{step}{2} - 90°|)} \tag{4}$$

$$\boxed{Adj_1 = \frac{Opp.}{tan(|\frac{step}{2} - 90°|)} = \frac{\text{R} \times sin(\Delta a)}{tan(|\frac{step}{2} - 90°|)}} \tag{5}$$

But then, we also need to notice

$$Adj_2 = \text{R} \times cos(\Delta a), \tag{6}$$

$$\Delta L = |\text{R} - Adj_2|, \tag{7}$$

So that we can finally get

$$\boxed{\Delta r = |Adj_1 - \Delta L| = \left|\left(\frac{\text{R} \times sin(\Delta a)}{tan(|\frac{step}{2} - 90°|)}\right) - |\text{R}(1 - cos(\Delta a)|\right|} \tag{8}$$

We now can complete the code with Δr (`delta_radius`) as the following:

```
for j in range(num_iter):
    last_point = [0.0, 0.0]; first_point = [0.0, 0.0]
    for i in range(num_sides):
        angle = i * step_angle - (step_angle / 2.0) - acc_angle
        x = fig_center[0] + (acc_radius) * math.cos(angle * PI / 180.0)
        y = fig_center[1] + (acc_radius) * math.sin(angle * PI / 180.0)
        if i == 0:
            first_point = [x, y]
        if i > 0:
            ln.line([x, y], last_point, thick, color, img)
        if i == num_sides - 1:
            ln.line([x, y], first_point, thick, color, img)
        last_point = [x, y]
    angle2 = abs((step_angle / 2.0 - 90.0))
    adj1 = (acc_radius * math.sin(delta_angle * PI / 180.0)) / (math.tan(angle2 * PI / 180.0))
    adj2 = acc_radius * math.cos(delta_angle * PI / 180.0); delta_L = abs(acc_radius - adj2)
    acc_angle += delta_angle
    delta_radius = abs(adj1 - delta_L)
    acc_radius += delta_radius
```

This code creates a number of iterations where each makes a polygon slightly rotated by a custom `delta angle` that remains **constant during all** the iterations. Nevertheless, each polygon should **grow its radius** or size, but not by a constant amount. The increase in radius of the polygons should obey the relation in Eq.(5) and, more precisely, accumulate the radius with Eq.(8) at the end of every polygon drawing. The result is the image shown below.

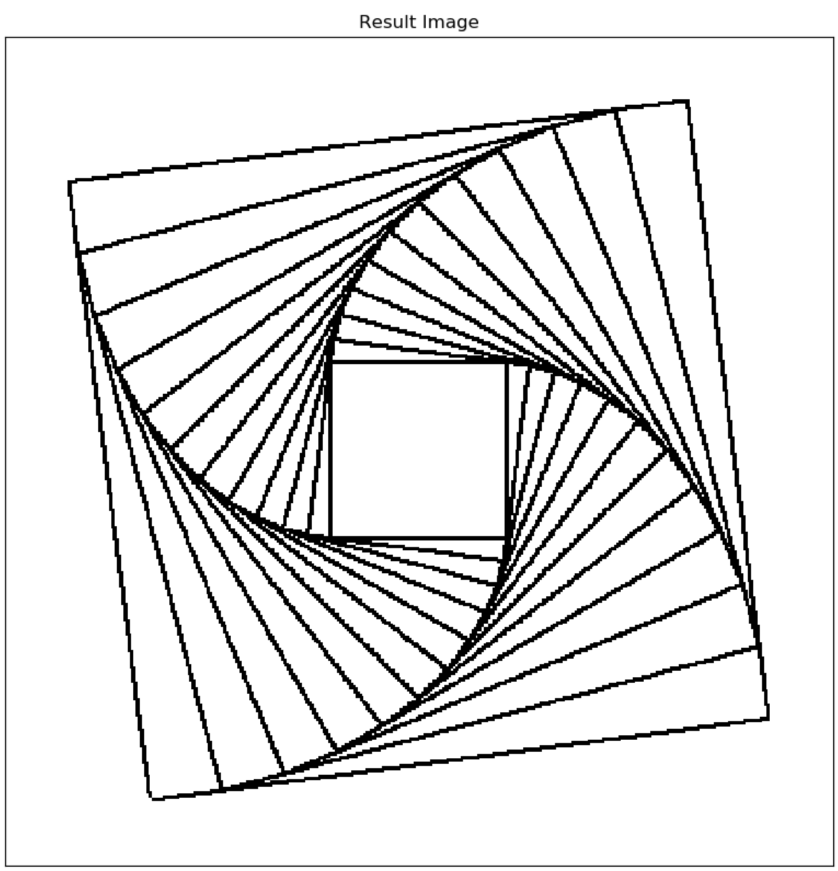


Figure 2: Final Geometric Spiral using Python 3.6.2

1.5 Further results

The program uses `delta_angle` as a variable to store the Δa, but the variable is not fully a custom variable (could be fully custom, though): it depends on the number of sides of the polygon (in this case 4 to form the square) and the number of iterations the user wants. Thus, this variable and others are defined as the following for the program to work in a complete way.

```
img = np.zeros([400, 400,3],dtype=np.uint8); img[:] = (255, 255, 255)
color = [0, 0, 0]; thick = 2
num_sides = 4 # the number of sides could be any
fig_center = [0.0, 0.0]
radius = 60 # in pixels
num_iter = 12
angle_dist = 2
delta_angle = (angle_dist * 180.0 / (num_sides)) / num_iter
acc_angle = 0.0
acc_radius = radius
step_angle = 360.0 / num_sides
```

Further testing of the explained program done with Python 3.6.2 gave the output below. The only parameter changed by the user is the number of iterations.

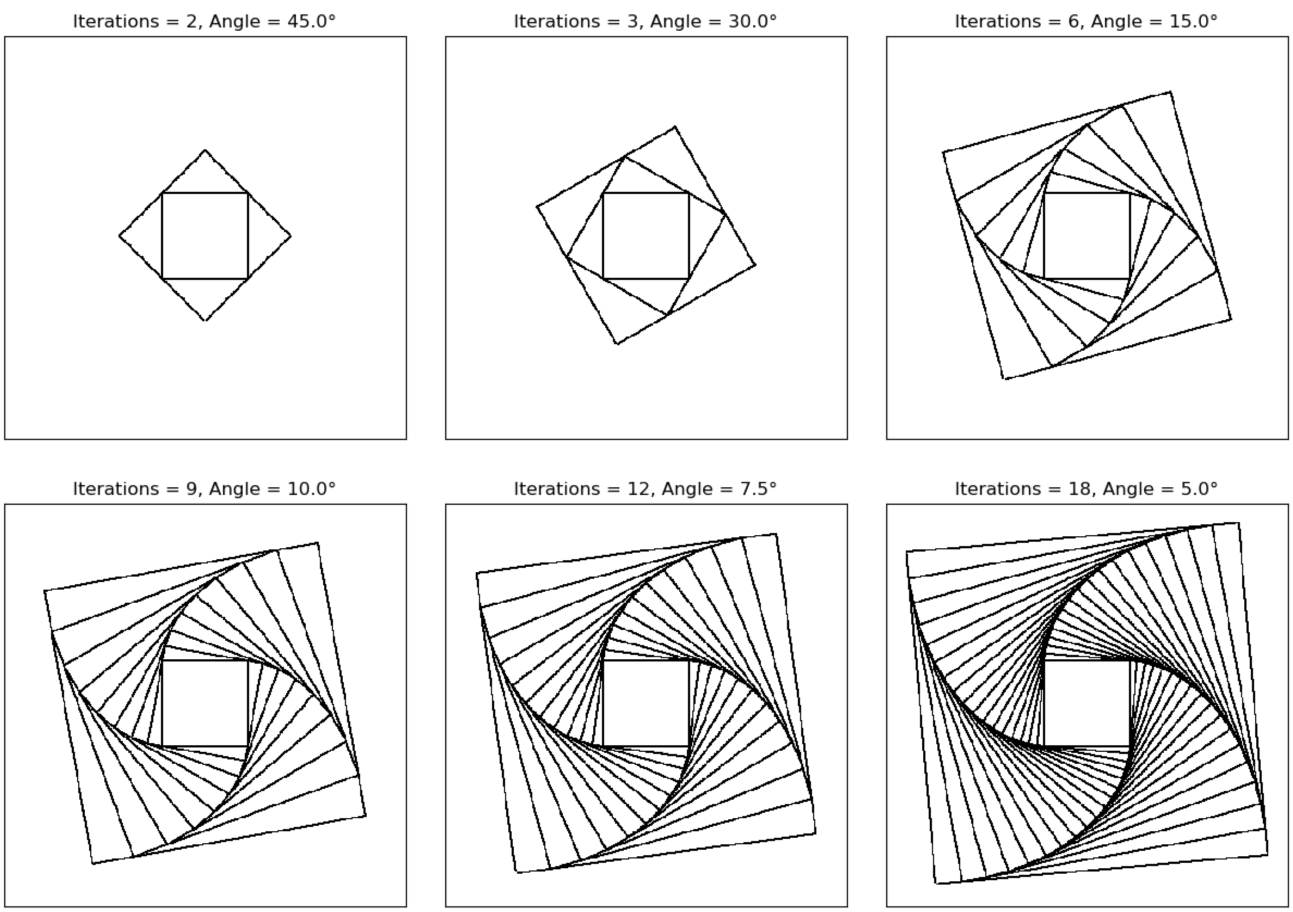


Figure 3: Program tests following Eq.(8)

The relations in the previous section work for any regular polygon, that is, when `num_sides` ≥ 3. Thus, testing the program for an hexagon (6 sides) or triangle shows the output below.

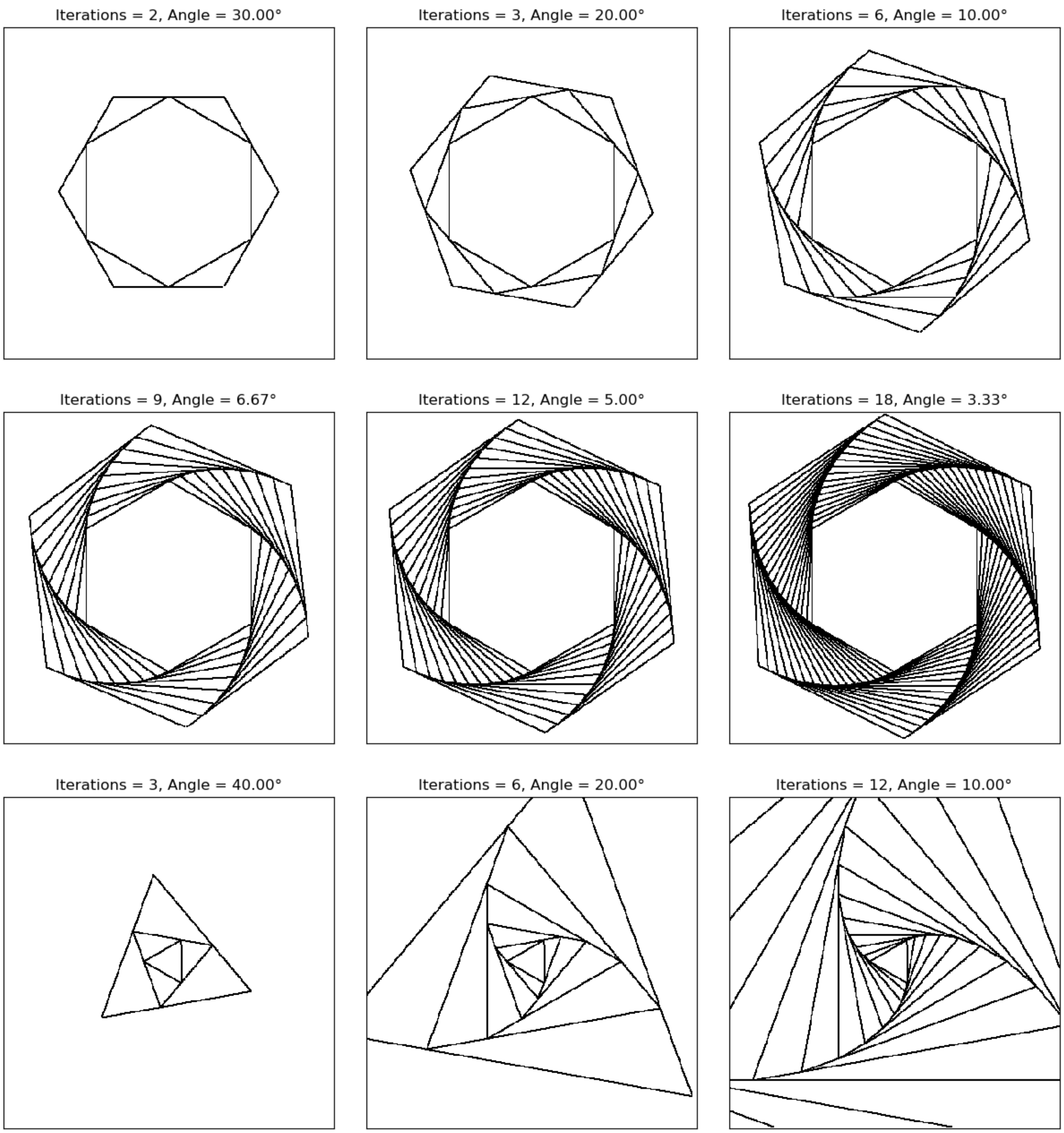


Figure 4: Further tests following Eq.(8)