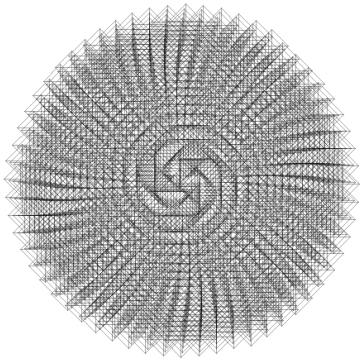
1. Patterns in nature



"The scientist does not study nature because it is useful; he studies it because he delights in it, and he delights in it because it is beautiful." (Henri Poincaré)

There are many examples of *natural facts* that can be described in mathematical terms. Nice examples are the shape of snowflakes, the *fractal geometry* of romanesco broccoli or how self-similarity rules the growth of plants.

R is a tool for doing serious analysis, but not everything in life is serious. Life is also funny, and R can be used to have fun and to do beautiful things. Its graphical power can be used to produce artistic images like the one that illustrates this section, which is inspired by how plants arrange their leaves. This fact is called *phyllotaxis* and will serve as the basis of this project.

In this notebook, we are using the `ggplot2` package. Apart from having fun, we will learn many important features of it that will be useful not only to do art but also to represent data in real-life problems. Let's start by loading the library.

In [292]:

```
# This sets plot images in the whole this notebook to a nice size.
options(repr.plot.width = 4, repr.plot.height = 4)

# Loading in the ggplot2 package
library('ggplot2')
```

In [293]:

```
library(testthat)
library(IRkernel.testthat)

run_tests({
    test_that("Test that ggplot2 is loaded", {
    expect_true( "package:ggplot2" %in% search(),
        info = "The ggplot2 package should be loaded using library().")
    })
})
```

```
1/1 tests passed
```

2. Warming up: drawing points on a circle

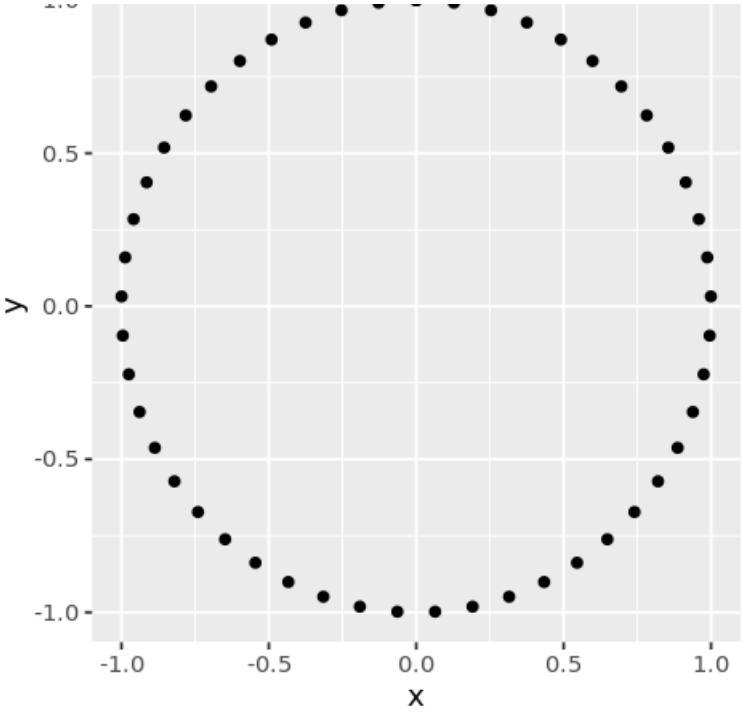
There are many ways to represent data with `ggplot2` : from simple scatter plots to more complex violin plots. The functions that start with `geom_` define the type of plot. In this notebook, we will only work with `geom_point()` which plots points in two dimensions. We'll need a dataset with two variables; let's call them `x` and `y` .

We'll start by drawing 50 points on a circle of radius 1. As every `(x, y)` point should be in the unit circle, it follows that $x^2 + y^2 = 1$. We can get this using the *super famous* Pythagorean trigonometric identity which states that $\sin^2(\theta) + \cos^2(\theta) = 1$ for any real number θ.

In [294]:

```
t <- seq(0, 2*pi, length.out = 50)
x <- sin(t)
y <- cos(t)
df <- data.frame(t, x, y)

# Make a scatter plot of points in a circle
p <- ggplot(df, aes(x, y))
p +
geom_point()
```



In [295]:

```
run_tests({
    test_that("Check that a geom_point plot was plotted.", {
    expect_true( "GeomPoint" %in% class( last_plot()$layers[[1]]$geom ) ,
        info = "Add geom_point() to produce a scatter plot.")
    })
})
```

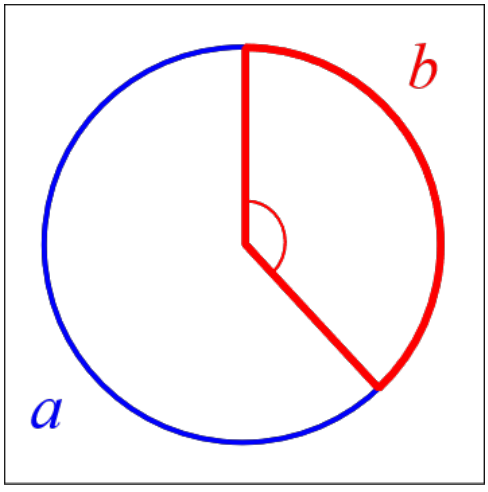
```
1/1 tests passed
```

3. Make it harmonious with the Golden Angle

Plants arrange their leaves in spirals. A spiral is a curve which starts from the origin and *moves away* from this point as it revolves around it. In the plot above all our points are at the same distance from the origin. A simple way to arrange them in a spiral is to multiply `x` and `y` by a factor which increases for each point. We *could* use `t` as that factor, as it meets these conditions, but we will do something more *harmonious*. We will use the Golden Angle:

Golden Angle = $\pi(3 - \sqrt{5})$

This number is inspired by the Golden Ratio, one of the most famous numbers in the history of mathematics. Imagine that you have a circumference and you break up it into two arcs with lengths `a` and `b`, with `a>b` (an arc is a portion of the circumference). The angle that *breaks* the circle so that `a/b=(a+b)/a` is called the Golden Angle. In other words: the Golden Angle *breaks up* a circle so that the ratio of the big arc to the little arc is the Golden Ratio. This image (from Wikipedia) illustrates the previous definition:



The Golden Angle is the angle subtended by the smaller (red) arc. Both the Golden Ratio and the Golden Angle appear in

The Golden Angle is the angle subtended by the smaller (red) arc. Both the Golden Ratio and the Golden Angle appear in unexpected places in nature. Apart of flower petals and plant leaves, you'll find them in seed heads, pine cones, sunflower seeds, shells, spiral galaxies, hurricanes, etc.

It's time to *spiralize*!

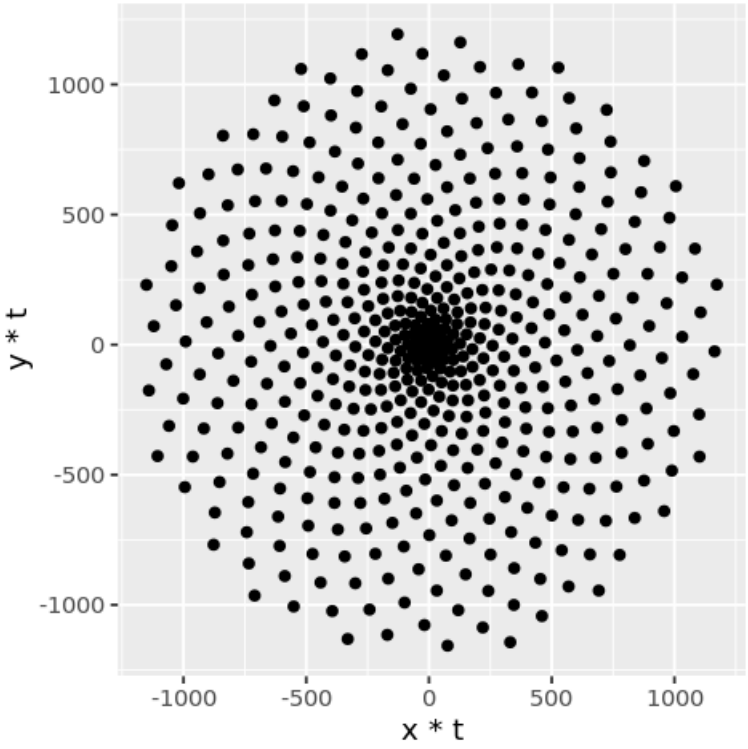
In [296]:

```
# Defining the number of points
points <- 500

# Defining the Golden Angle
angle <- pi*(3 - sqrt(5))

t <- (1:points) * angle
x <- sin(t)
y <-cos(t)
df <- data.frame(t, x, y)

# Make a scatter plot of points in a spiral
p <- ggplot(df, aes(x*t, y*t))
p +
 geom_point()
```



In [297]:

```
run_tests({
    test_that("points are 500.", {
    expect_equal(points, 500,
        info = "There should be 500 points.")
    })

    test_that("angle is golden.", {
    expect_equal(angle, pi*(3-sqrt(5)),
        info = "angle should be set to the Golden Angel. Check the hint!")
    })
})
```

```
2/2 tests passed
```

4. Remove everything unnecessary

Apart from data, a plot includes many other components that define its final appearance. Our previous plot contains:

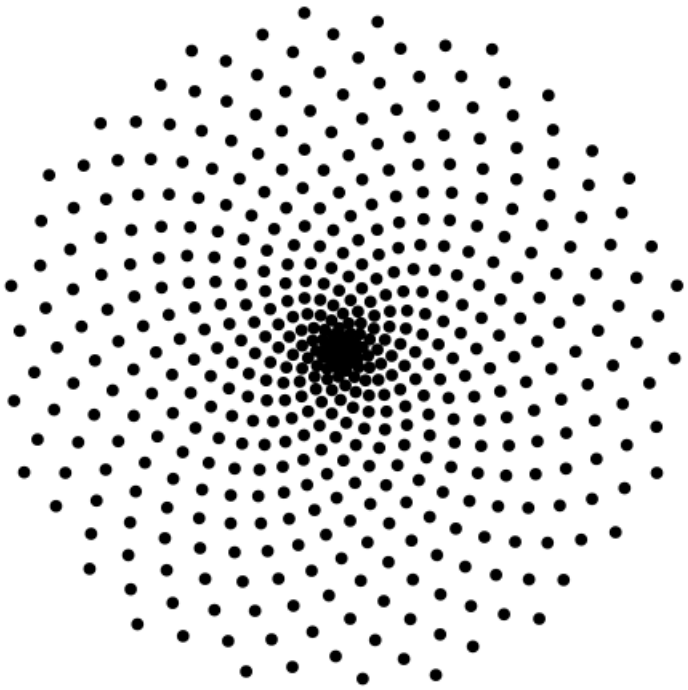
- a **background** filled with grey color
- a **grid** of horizontal and vertical white lines
- **ticks** along the axis
- a **title** on each axis
- **text** along axes to label marks

Art does not get along with most of these elements, so it's time to move to action.

In [298]:

```
df <- data.frame(t, x, y)

# Make a scatter plot of points in a spiral
p <- ggplot(df, aes(x*t, y*t))
p + geom_point() +
theme(legend.position="none",
        panel.background = element_rect(fill = 'white'),
        panel.grid=element_blank(),
        axis.ticks=element_blank(),
        axis.title=element_blank(),
        axis.text=element_blank() )
```



In [299]:

```
# Maybe just check if one of the this are blank,
# seems too much to check them all...

run_tests({
    test_that("Background is white.", {
    expect_equal(last_plot()$theme$panel.background$fill, "white",
        info = "The background should be white.")
    })
    test_that("Grid is removed.", {
    expect_true("element_blank" %in% class(last_plot()$theme$panel.grid),
        info = "The grid lines should be removed.")
    })
})
```

```
2/2 tests passed
```

5. A bit of makeup: size, color and transparency

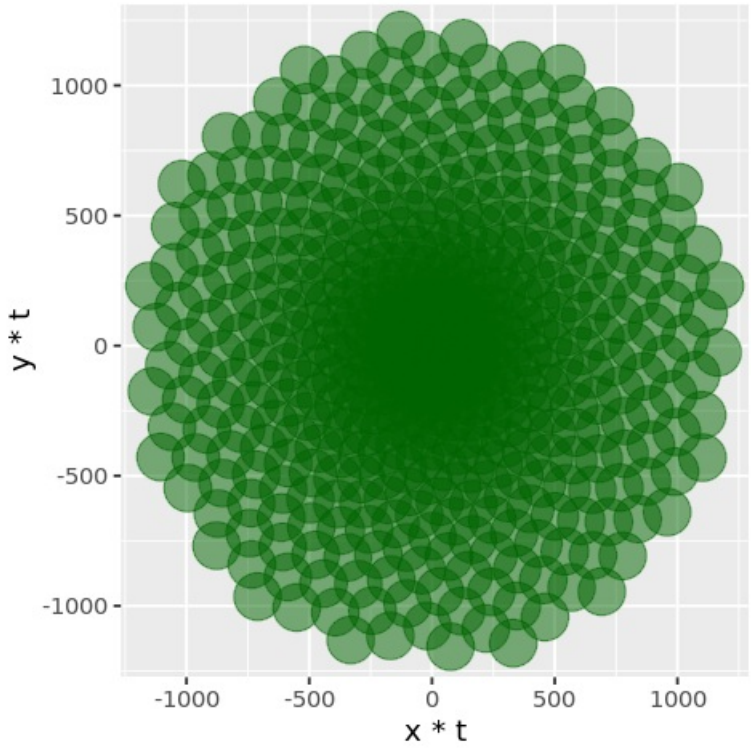
Our drawing is starting to look like a plant, but we can better. By changing color, transparency (also called *alpha*), and size of the points, the image will become more appealing.

In [300]:

```
# Change the code from Task 4 to modify the
# size, transparency, and color of the points
p <- ggplot(df, aes(x*t, y*t))
p + geom_point(size = 8, alpha = 0.5, color = 'darkgreen')
head(df)
```

A data.frame: 6 x 3

t	x	y
<dbl>	<dbl>	<dbl>
2.399963	0.6754903	-0.73736888
4.799926	-0.9961710	0.08742572
7.199890	0.7936008	0.60843886
9.599853	-0.1741820	-0.98471349
11.999816	-0.5367281	0.84375529
14.399779	0.9657151	-0.25960430



In [301]:

```
run_tests({
    test_that("Point size equal to 8.", {
    expect_equal(last_plot()$layers[[1]]$aes_params$size, 8,
        info = "size should be set 8.")
    })
    test_that("alpha equal to 0.5.", {
    expect_equal(last_plot()$layers[[1]]$aes_params$alpha, 0.5,
        info = "alpha should be set 0.5.")
    })
})
```

```
2/2 tests passed
```

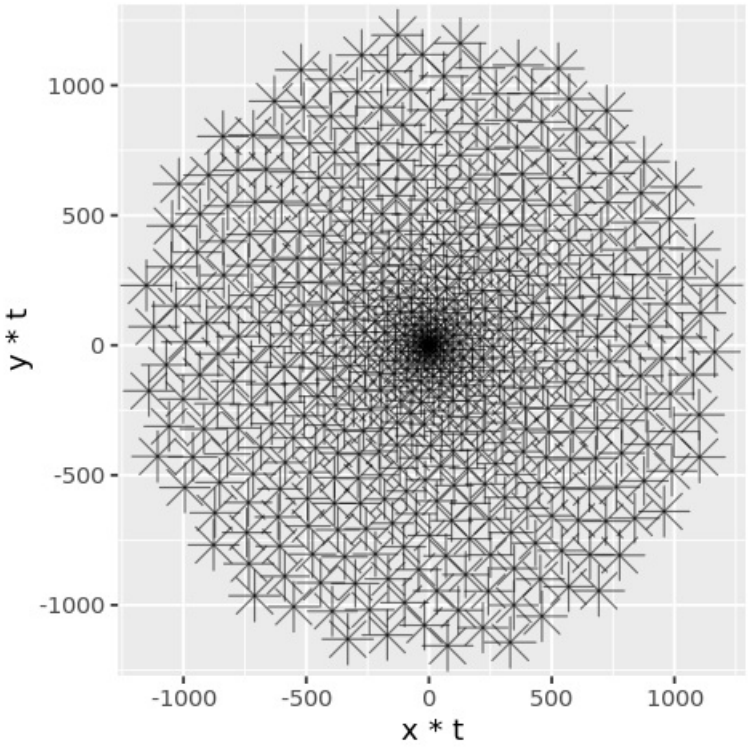
6. Play with aesthetics: the dandelion

6. Play with aesthetics: the dandelion

Until now, all points have the same appearance (`size` , `color` , `shape` , and `alpha`). Sometimes we will want to make the appearance of the points dependent on a variable in the dataset. Now we will make the size variable. We will also change the shape of the points. Although we won't be able to blow on it, the resulting image should remind you of a dandelion.

In [302]:

```
# Copy the code from Task 5 and modify the
# color, size, and shape of the points
p <- ggplot(df, aes(x*t, y*t))
p + geom_point(aes(size = t, alpha = 0.5), shape = 8) + #or shape = "*"
theme(legend.position = "none")
```



In [303]:

```
run_tests({
    test_that("Map size of points to t.", {
    expect_equal(last_plot()$labels$size, "t",
        info = "Map size of points to t. Check the hint!")
    })
    test_that("point shape is asterisk.", {
    expect_equal(last_plot()$layers[[1]]$aes_params$shape, 8,
        info = "Change the shape of all points to asterisks.")
    })
    test_that("Legend is removed.", {
    expect_equal(last_plot()$theme$legend.position, "none",
        info = "Remove the legend from the plot.")
    })
})
```

```
3/3 tests passed
```

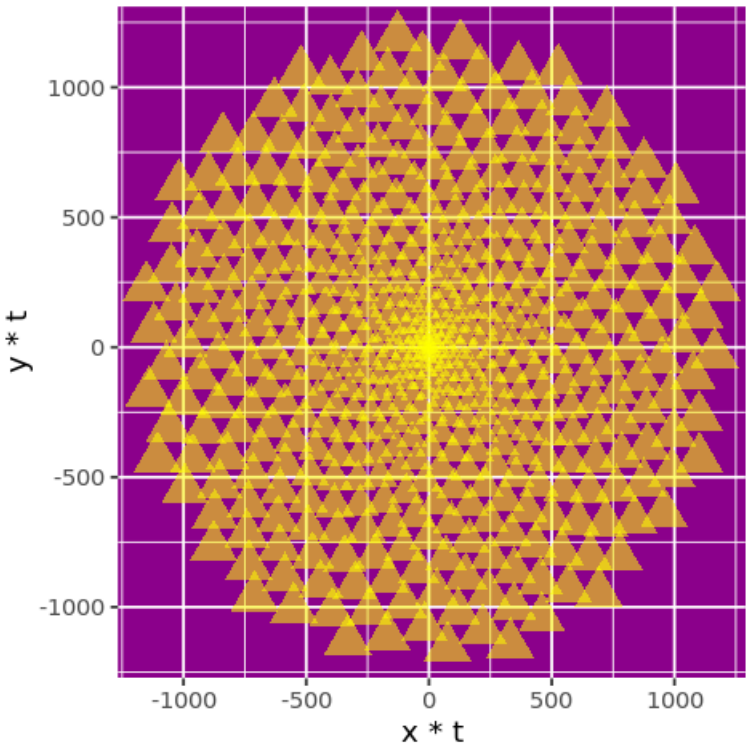
7. Put all it together: the sunflower

Plants not only use the Golden Angle to arrange leaves. The Golden Angle is also found in the arrangement of sunflower seeds. We don't need anything new to draw a sunflower; we just need to combine some of the things we already know.

In [304]:

```
# Copy the code from Task 6 and modify the color and
# shape of the points, and the background color
```

```
p <- ggplot(df, aes(x*t, y*t))
p + geom_point(aes(size = t, alpha = 0.5), shape = 17, color = 'yellow') +
theme(legend.position = "none", panel.background = element_rect(fill = "darkmagenta"))
```



In [305]:

```
run_tests({
    test_that("point shape is filled triangles.", {
        expect_equal(last_plot()$layers[[1]]$aes_params$shape, 17,
                     info = "Change the shape of all points to filled triangles. Check the hint.")
    })
    test_that("The triangles are yellow", {
        expect_equal(last_plot()$layers[[1]]$aes_params$colour, "yellow",
                     info = "The triangles are not yellow. Check the hint.")
    })
    test_that("The background is dark magenta", {
        expect_equal(last_plot()$theme$panel.background$fill, "darkmagenta",
         info = "The background is not dark magenta. Check the hint.")
    })
})
```

```
3/3 tests passed
```

8. What if you modify the angle?

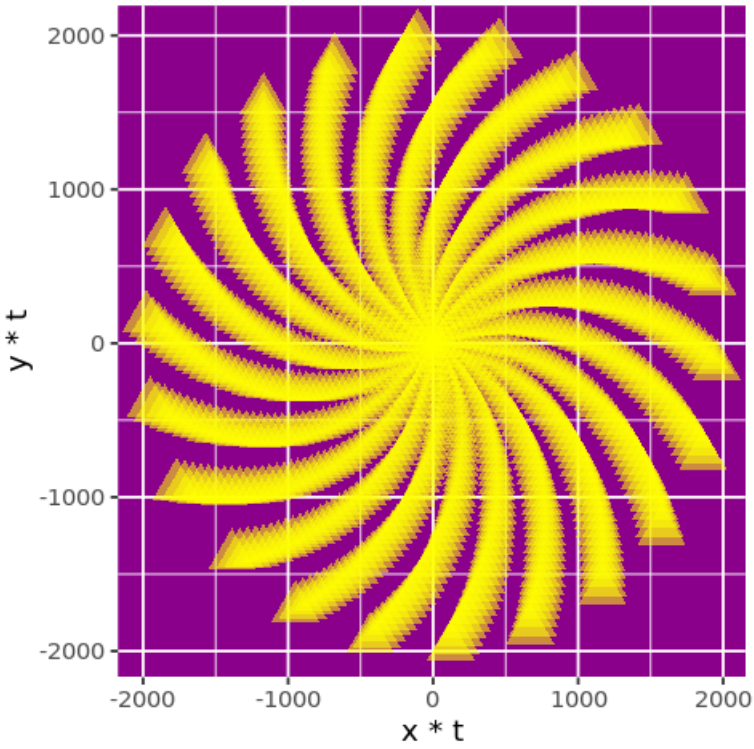
These patterns are very sensitive to the angle between the points that form the spiral. Small changes to the angle can generate very different images. Let's look at an example of that.

In [306]:

```
# Change the value of the angle
angle <- 2.0
points <- 1000

t <- (1:points)*angle
x <- sin(t)
y <- cos(t)
df <- data.frame(t, x, y)

# Copy the plotting code from Task 7
p <- ggplot(df, aes(x*t, y*t))
p + geom_point(aes(size = t, alpha = 0.5), shape = 17, color = 'yellow') +
theme(legend.position = "none", panel.background = element_rect(fill = "darkmagenta"))
```

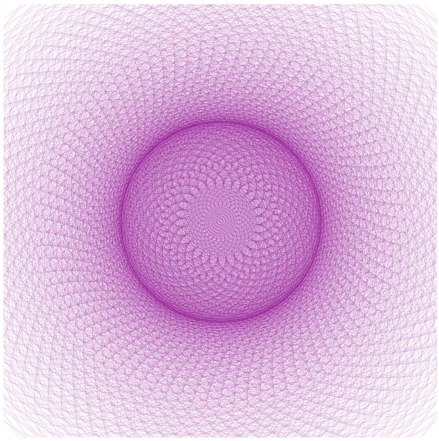


In [307]:

```
run_tests({
    test_that("angle is 2.", {
    expect_equal(angle, 2,
        info = "angle should be equal to 2")
    })
})
```

```
1/1 tests passed
```

9. All together now: imaginary flowers



The techniques we've used so far allow us to create an *infinite* number of patterns inspired by nature: the only limit is our imaginations. But making art has also been a fun excuse to learn to use `ggplot2`. All the tricks we have seen in this notebook are useful when plotting *real* data too.

The image on the left is a simple variation of the previous flower and is in essence very similar to the first figure in which we plotted 50 points in a circle. I hope you've enjoyed the journey between that simple circle and this beautiful flower.

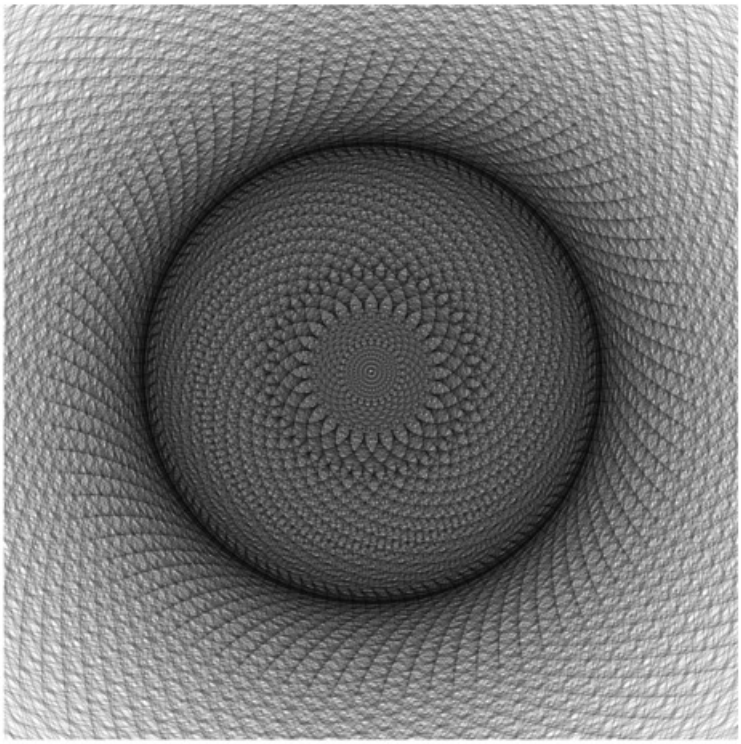
In [308]:

```
# Change the values of angle and points
angle <- 13*pi/180
points <- 2000

t <- (1:points)*angle
x <- sin(t)
y <- cos(t)
df <- data.frame(t, x, y)

# Adjust the plot parameters to create the magenta flower
```

```
# Adjust the plot parameters to create the magenta flower
p <- ggplot(df, aes(x*t, y*t))
p + geom_point(size = 80, alpha = 0.1, shape = 1)+
  theme(legend.position="none",
        panel.background = element_rect(fill = 'white'),
        panel.grid=element_blank(),
        axis.ticks=element_blank(),
        axis.title=element_blank(),
        axis.text=element_blank())
```



In [309]:

```
run_tests({
    test_that("points is equal to 2000.", {
    expect_equal(points, 2000,
        info = "There should be 2000 points.")
    })
    test_that("point shape is empty circle.", {
    expect_equal(last_plot()$layers[[1]]$aes_params$shape, 1,
        info = "Change the shape of all points to empty circles. Check the hint!")
    })
    test_that("alpha is equal 0.1", {
    expect_equal(last_plot()$layers[[1]]$aes_params$alpha, 0.1,
        info = "alpha of points should be 0.1")
    })
    test_that("Background is white.", {
    expect_equal(last_plot()$theme$panel.background$fill, "white",
        info = "The background should be white.")
    })
    test_that("angle is 13*pi/180.", {
    expect_equal(angle, 13*pi/180,
        info = "angle should be set to 13*pi/180.")
    })
})
```

5/5 tests passed