Fractals - Choas Game

Raphaël Morsomme

November 15, 2018

Contents

```r
library(tidyverse)
```

1 Introduction

In this script, I implement a recursive method to generate fractals, which is called the *chaos game*. I decided to write this script after watching a tutorial video by Numberphile on the topic. I found the chaos game fascinating and wanted to implement it on `R Studio`. I also wanted to generalize the method presented in the tutorial to other types of fractals.

1.1 Fractals

Fractals are complex mathematical objects. In this script, it is sufficient to understand them as geometric figures whose parts are reduced-size copies of the whole. That is, given a fractal, if we zoom in on any of its parts, we find the exact same patterns as in the original figure, no matter how small the part is. One of the most famous fractals in mathematics is the Sierpinski Triangle (see Figure 1.).

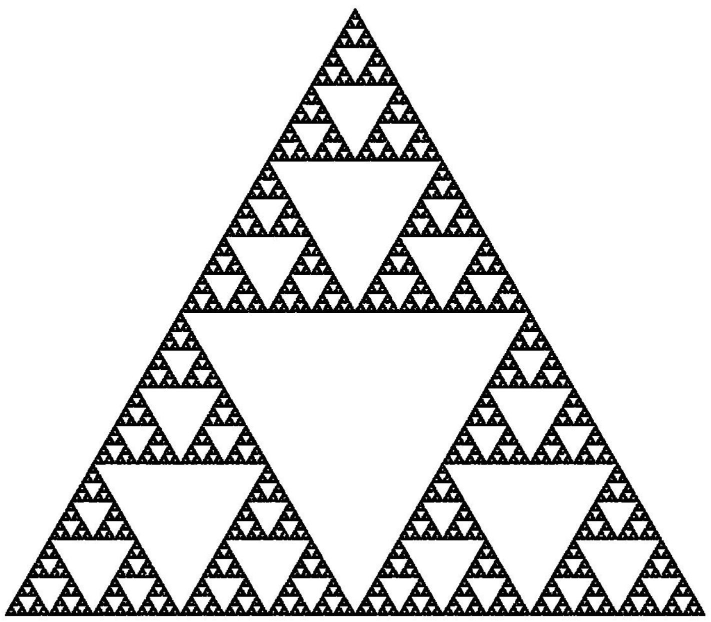


Figure 1: Sierpinski Gasket

1.2 The Choas Game

The 8-minute tutorial *Chaos Game* by Numberphile[1] does a much better job at explaining the choas game than I could possibly do, so I recommend the reader to simply watch it to understand its mechanism. Depending on the desired fractal, the method slightly changes. For the Sierpinski Gasket, it follows 5 steps:

1. Take three points in a plane. These will be the vertices of the final fractal.
2. Choose a point in the plane and draw it. This is the current point.
3. Randomly choose a vertex. This is the the chosen vertex.

[1]The tutorial video is freely available at https://www.youtube.com/watch?v=kbKtFN71Lfs.

4. Find the point half distance between the current point and the chosen vertex. Draw the point. This point is now the current point.
5. Repeat from step 3.

1.3 Outline of Script

I kick off this script with a well-know fractal: Sierpinski Gasket. I write a function that generates it and then experiment with the parameters to vary the shape of and the pattern in the obtained fractal. I then generalize the function to generate fractals with any number of vertices before finishing the script with an adaptation of the chaos game that generates Barnsley Fern, another beautiful fractal.

2 Sierpinski Gasket and Other Triangular Fractals

2.1 The Function

We design a function `generate_sg` that uses the chaos game method to generate a Sierpinski Gasket. The function has the following arguments:

- `n` determines the number of iterations. A large `n` produces a figure with a sharper pattern.
- `v1`, `v2` and `v3` determine the coordinates of the three vertices of the fractal. Their default values produce a equilateral triangle.
- `p` determines the location of the new point on the line segment between the previous point and the chosen vertex. Note that `p` must be comprised between `0` and `1`; small values indicates that the new point is close to the selected vertex, while values close to `1` indicate that the new point is close to the previous one.
- `initial_point` indicates the location of the initial point.
- `title` is the title of the graph. If let to `NULL`, the figure has no title.
- `subtitle` is the subtitle of the graph. If let to `NULL`, the figure has no subtitle.

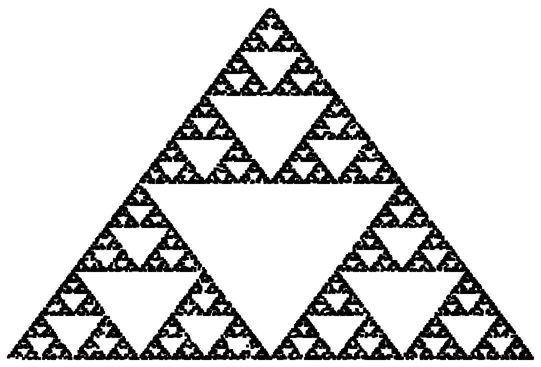
```r
generate_sg <- function(n = 1e4, v1 = c(0,0), v2 = c(1, 0), v3 = c(0.5, sqrt(3)/2),
                        p = 0.5, initial_point = v1, title = NULL, subtitle = NULL){

  points <- data.frame(x = NA, y = NA)

  point_previous <- initial_point

  for(i in 1:n){

    vertex         <- sample(list(v1, v2, v3), size = 1)[[1]]
    point_new      <- p       * point_previous +
                      (1 - p) * vertex

    points[i, ]    <- point_new

    point_previous <- point_new

  }

  g <- ggplot(points, aes(x, y)) +
    geom_point(shape = ".") +
    labs(title = title, subtitle = subtitle) +
    theme_void() +
    theme(plot.title    = element_text(hjust = 0.5),
          plot.subtitle = element_text(hjust = 0.5))

  ggsave(paste("Triangular Fractal with p =", p, ".jpeg"),
         width = 4, height = 2 * sqrt(3), path = "Plots")

  return(g)

}
```

The loop of `generate_sg` can be simplified in the following way to speed up the function.

```r
generate_sg <- function(n = 1e4, v1 = c(0,0), v2 = c(1, 0), v3 = c(0.5, sqrt(3)/2),
                        p = 0.5, initial_point = v1, title = NULL, subtitle = NULL){

  points <- data.frame(x = NA, y = NA)

  point <- initial_point

  for(i in 1:n){

    point       <- p       * point +
                  (1 - p) * sample(list(v1, v2, v3), size = 1)[[1]]

    points[i, ] <- point

  }

  g <- ggplot(points, aes(x, y)) +
    geom_point(shape = ".") +
    labs(title = title, subtitle = subtitle) +
    theme_void() +
    theme(plot.title    = element_text(hjust = 0.5),
          plot.subtitle = element_text(hjust = 0.5))

  ggsave(paste("Triangular Fractal with p =", p, ".jpeg"),
         width = 4, height = 2 * sqrt(3), path = "Plots")

  return(g)

}
```

Running the function with its arguments left to their defaults values generates the Sierpinski Gasket.

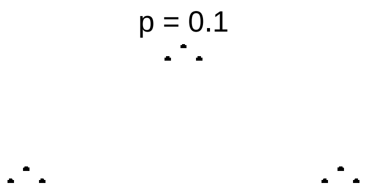
```r
generate_sg()
```

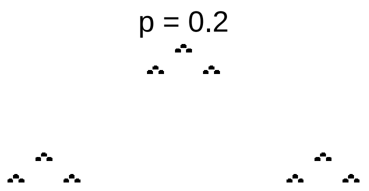


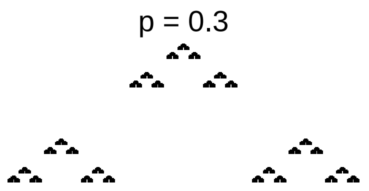
2.2 Other Triangular Fractals

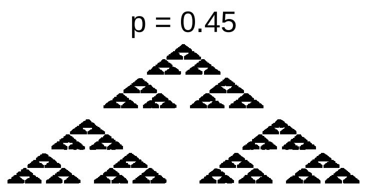
We can design triangular fractals with various shapes and patterns by varying the values of the parameters of the function `generate_sg`. First, by changing the value of `p`, we generate fractals with different patterns.

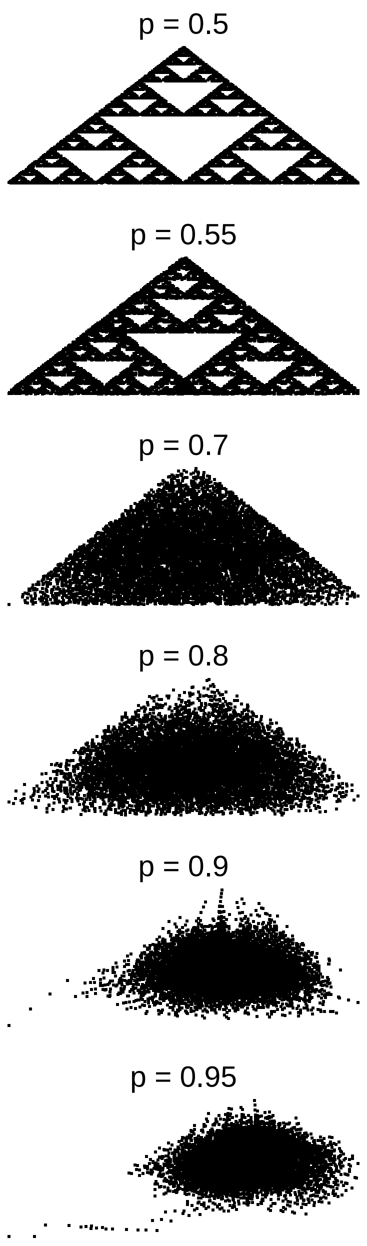
```r
for(p in c(0.1, 0.2, 0.3, 0.45, 0.5, 0.55, 0.7, 0.8, 0.9, 0.95)){

  print(generate_sg(p = p, subtitle = paste("p =", p)))

}
```







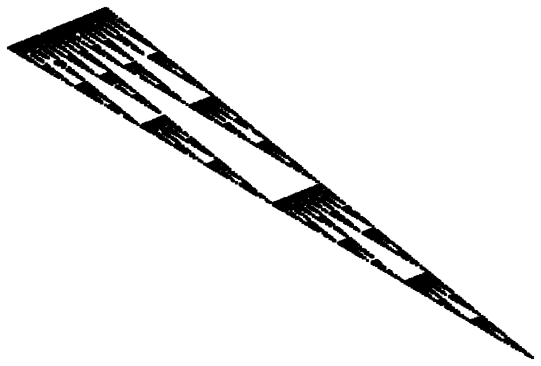




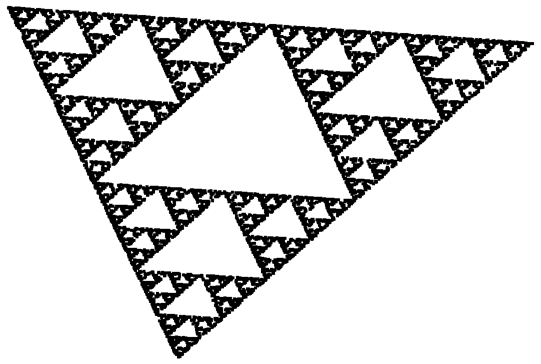
It seems that for `p` superior to `0.5`, larger values give figures that are more chaotic.

Then, we can also randomly determine the location of the vertices to obtain different triangular figures.

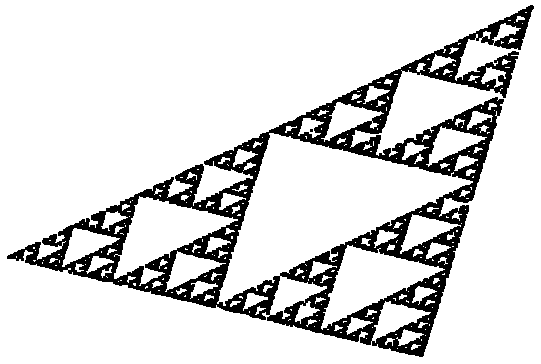
```r
set.seed(123)
generate_sg(v1 = runif(2), v2 = runif(2), v3 = runif(2))
```



```r
set.seed(124)
generate_sg(v1 = runif(2), v2 = runif(2), v3 = runif(2))
```



```r
set.seed(125)
generate_sg(v1 = runif(2), v2 = runif(2), v3 = runif(2))
```

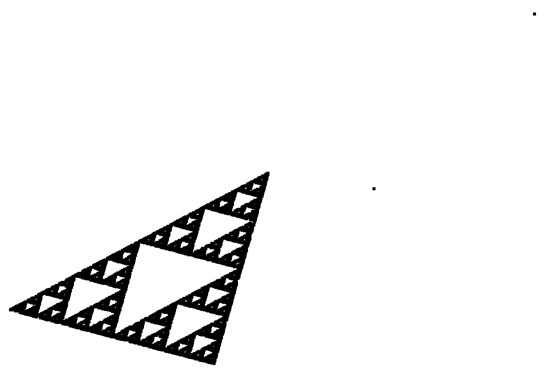


Note that increasing the number of iterations generates a figure with a sharper pattern, but takes more time to run.

```r
set.seed(125)
#generate_sg(v1 = runif(2), v2 = runif(2), v3 = runif(2),
#            n = 1e5)
```

Also note that changing the location of the initial point does not significantly alter the outcome: the sequence of points rapidly follows the regular pattern.

```r
set.seed(125)
generate_sg(v1 = runif(2), v2 = runif(2), v3 = runif(2),
           initial_point = c(3, 3))
```



3 General Formula

3.1 The Function

One could also be interested in generating fractals with more vertices. The function `generate_fractal` is a generalization of the function `generate_sg` that can generate fractals with any number of vertices. It has the following arguments:

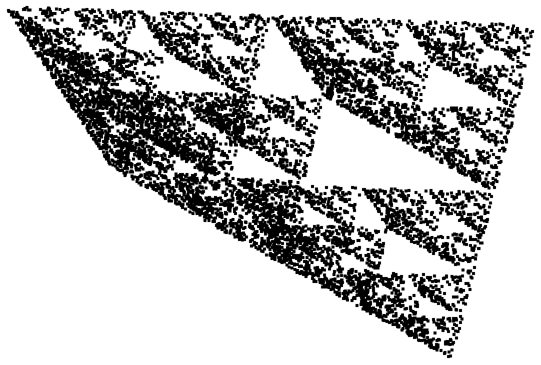
- `n` determines the number of iterations.
- `k` determines the number of vertices of the fractal.
- `x` and `y` determine the x- and y-coordinates of the fractal's vertices. If the arguments `x` and `y` are left to `NULL`, then the coordinates are randomly generated.
- `p` determines the location of the new point on the line segment between the previous point and the chosen vertex. Note that `p` must be comprised between `0` and `1`; small values indicates that the new point will be close to the selected vertex, while values close to `1` indicate that the new point will be close to the previous one.
- `title` is the title of the graph. If let to `NULL`, the figure has no title.
- `subtitle` is the subtitle of the graph. If let to `NULL`, the figure has no subtitle.

```
generate_fractal <- function(n = 1e4, k = 4, x = NULL, y = NULL, p = 0.5, title = NULL, subtitle = NULL

  if(is.null(x)){ x <- runif(k)}
  if(is.null(y)){ y <- runif(k)}

  points <- data.frame(x = NA, y = NA)

  point <- c(x[1], y[1])

  for(i in 1:n){

    m      <- sample(1 : length(x), size = 1)
    vertex <- c(x[m], y[m])
    point  <- p       * point +
              (1 - p) * vertex
    points[i, ] <- point

  }

  g <- ggplot(points, aes(x, y)) +
    geom_point(shape = ".") +
    labs(title = title, subtitle = subtitle) +
    theme_void() +
    theme(plot.title = element_text(hjust = 0.5),
          plot.subtitle = element_text(hjust = 0.5))


  ggsave(paste("k =", k, "and p =", p, ".jpeg"), width = 4, height = 4, path = "Plots")

  return(g)

}
```

3.2 Experimenting

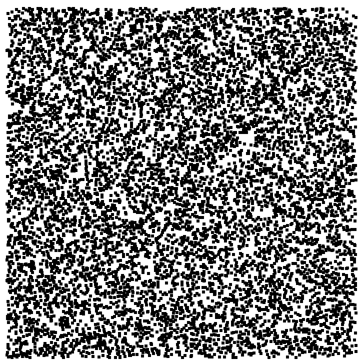
Running the function with its parameters left to their default value generates a quadrilateral figure with its vertices randomly located.

```
set.seed(123)
generate_fractal()
```



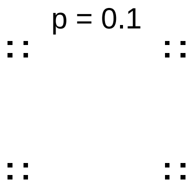
By fixing the vertices, we can generate a square fractal.

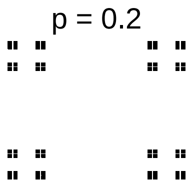
```
generate_fractal(x = c(0,0,1,1), y = c(0,1,0,1))
```

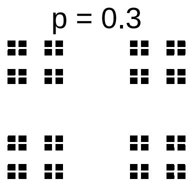


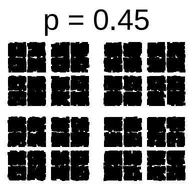
Surprisingly, unlike the previous quadrilateral figure, the square fractal does not contain any clear pattern. Yet, once again, by varying the value of `p` we can change the patterns in the obtained figure.

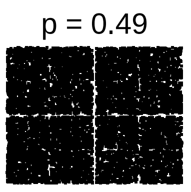
```
for(p in c(0.1, 0.2, 0.3, 0.45, 0.49, 0.5, 0.55, 0.7, 0.9)){

  print(generate_fractal(x = c(0,0,1,1), y = c(0,1,0,1),
                         p = p, subtitle = paste("p =", p)))

}
```

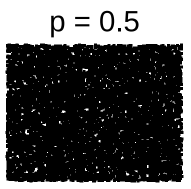


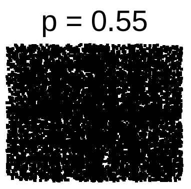


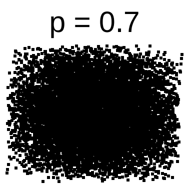


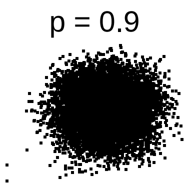












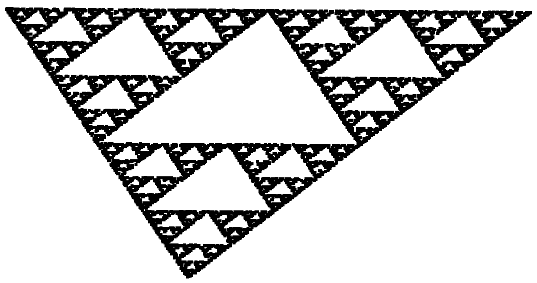
It seems that for square fractals, values of `p` equal or superior to `0.5` generates figures with no apparent pattern.

Changing the value of the argument `k` will generate fractals with different numbers of vertices.

```r
ks    <- c(3  , 4   , 5  , 7   )
seeds <- c(1  , 12  , 121, 241 )
ps    <- c(0.5, 0.45, 0.4, 0.35)

for(i in 1:length(ks)){

  k    <- ks[i]
  seed <- seeds[i]
  p    <- ps[i]

  set.seed(seed)

  print(generate_fractal(k = k, p = p,
                         title    = paste("Fractal with", k, "Vertices"),
                         subtitle = paste("p =", p)))

}
```

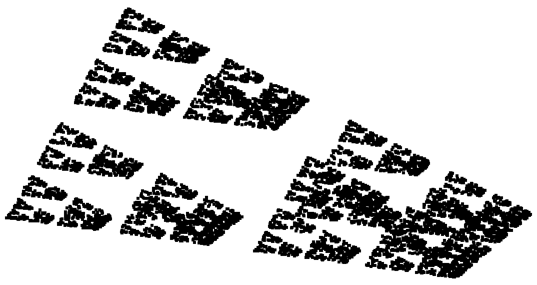
Fractal with 3 Vertices

p = 0.5



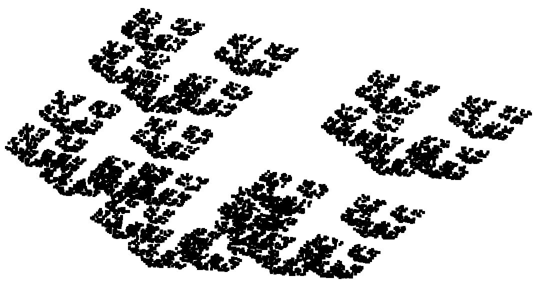
Fractal with 4 Vertices

p = 0.45



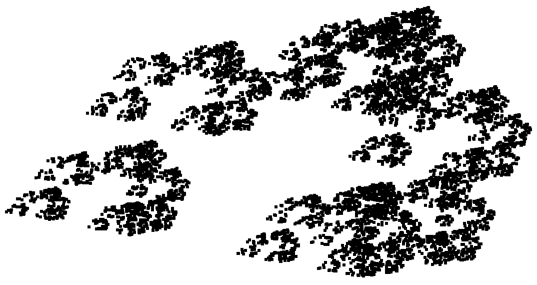
Fractal with 5 Vertices

p = 0.4



Fractal with 7 Vertices

p = 0.35



4 Bonus: Barnsley Fern

I want to conclude this script with an adaptation of the chaos game that generates the so-called Barnsley Fern (in my opinion one of the most beautiful fractals there is.). Remark on Figure 2. that each leaf of the fern is a fern itself.

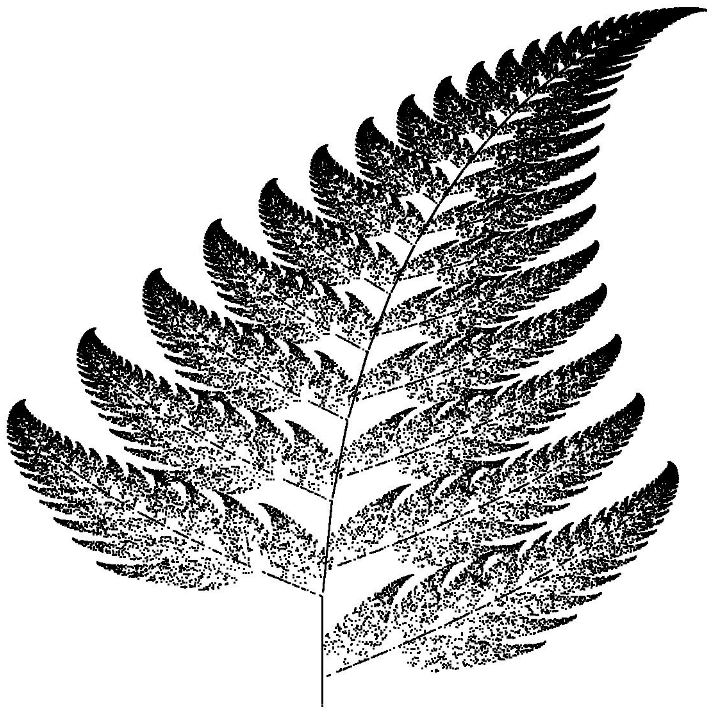


Figure 2: Barnsley Fern

4.1 The Function

The function `generate_bf` uses an adaptation of the chaos game to generate a Barnsley Fern. The underlying iterative mechanism of the function is fundamentally the same as that of the functions `generate_fractal` and `generate_sg`: given a point, we randomly apply to its coordinates a transformation from a given set of transformations to generate the next point. For the function `generate_bf`, we apply one of 4 affine transformations to the point's coordinates. These 4 transformation are captured in the rows of the two matrices `M1` and `M2`[2].

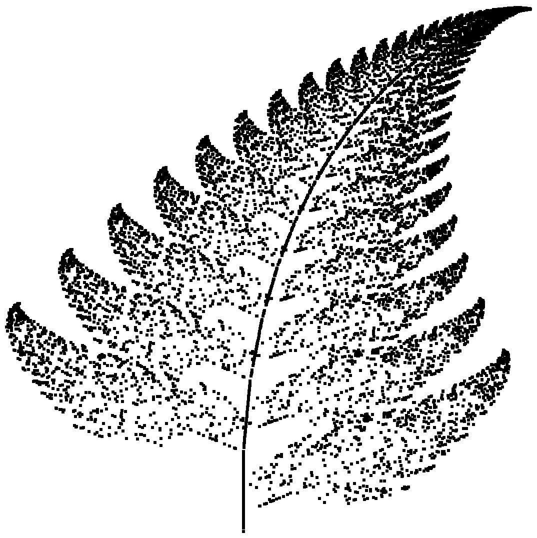
- `n` determines the number of iterations.
- `proba` gives the probability of applying each of the 4 transformations to the point. Each transformation has an associated element on the plot: the stem, the leaves' end, the fern's left-hand side and the fern's right-hand side.
- `title` is the title of the graph. If let to `NULL`, the figure has no title.
- `subtitle` is the subtitle of the graph. If let to `NULL`, the figure has no subtitle.

[2]I got the values for the entries of these matrices from Mr. Barnsley's book `Fractals Everywhere` (p.86, table III.3. *IFS code for a fern*). For clarity, I decided to split the table from Mr. Barnsley's book into two matrices.

```r
generate_bf <- function(n = 1e4, proba = c(0.01, 0.85, 0.07, 0.07), title = NULL, subtitle = NULL){

  M1 <- matrix(c(0    , 0    , 0    , 0.16,
                 0.85 , 0.04 , -0.04, 0.85,
                 0.20 , -0.26, 0.23 , 0.22,
                 -0.15, 0.28 , 0.26 , 0.24), byrow = T, ncol = 4)

  M2 <- matrix(c(0, 0   ,
                 0, 1.60,
                 0, 1.60,
                 0, 0.44), byrow = T, ncol = 2)

  points <- data.frame(x = NA, y = NA)

  point <- c(0,0)


  for(i in 1:n){

    k           <- sample(1 : 4, size = 1, prob = proba)
    point       <- matrix(M1[k, ], byrow = T, ncol = 2) %*% point + M2[k, ]
    points[i, ] <- point

  }

  g <- ggplot(points, aes(x, y)) +
    geom_point(shape = ".") +
    labs(title = title, subtitle = subtitle) +
    theme_void() +
    theme(plot.title    = element_text(hjust = 0.5),
          plot.subtitle = element_text(hjust = 0.5))


  ggsave("Barnsley Fern.jpeg", width = 3, height = 3, path = "Plots")

  return(g)

}
```

Running the function with its parameters left to their default value generates Barnsley Fern.

```r
generate_bf()
```



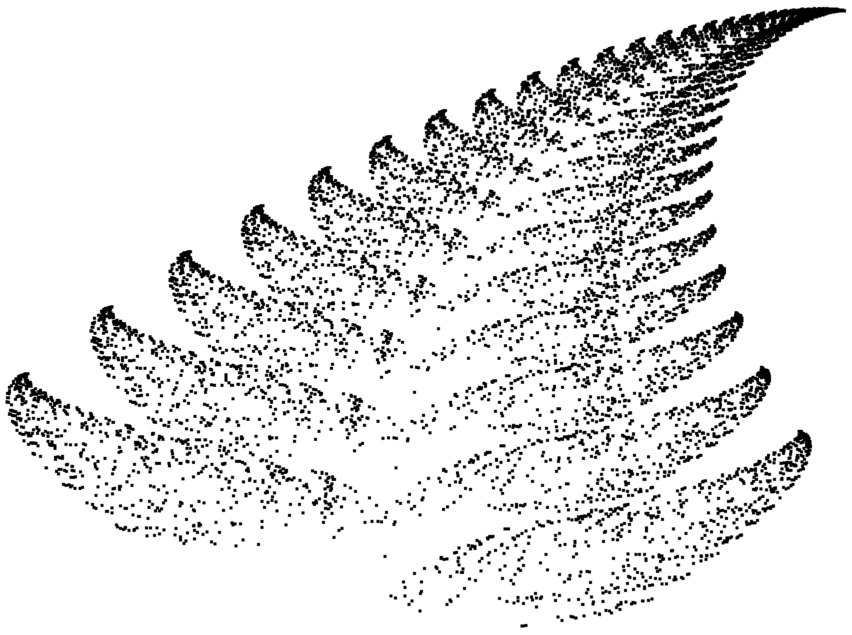
4.2 Experimenting

Changing the probabilities in `proba` changes the distribution of the points among the different elements of the fern (stem, leaves' end, left- and right-hand sides). If we set one of the elements of `proba` to `0`, it leaves the associated element of the fern blank.

```
proba     <- c(0.01, 0.85, 0.07, 0.07)
subtitles <- c("1st element of `proba` is null: blank stem"           ,
               "2nd element of `proba` is null: no result",
               "3rd element of `proba` is null: blank left-hand side",
               "4th element of `proba` is null: blank right-hand side")

for(i in 1:4){

  proba_blank    <- proba
  proba_blank[i] <- 0
  subtitle       <- subtitles[i]

  print(generate_bf(proba    = proba_blank,
                    title    = "Barnsley Fern",
                    subtitle = subtitle))

}
```

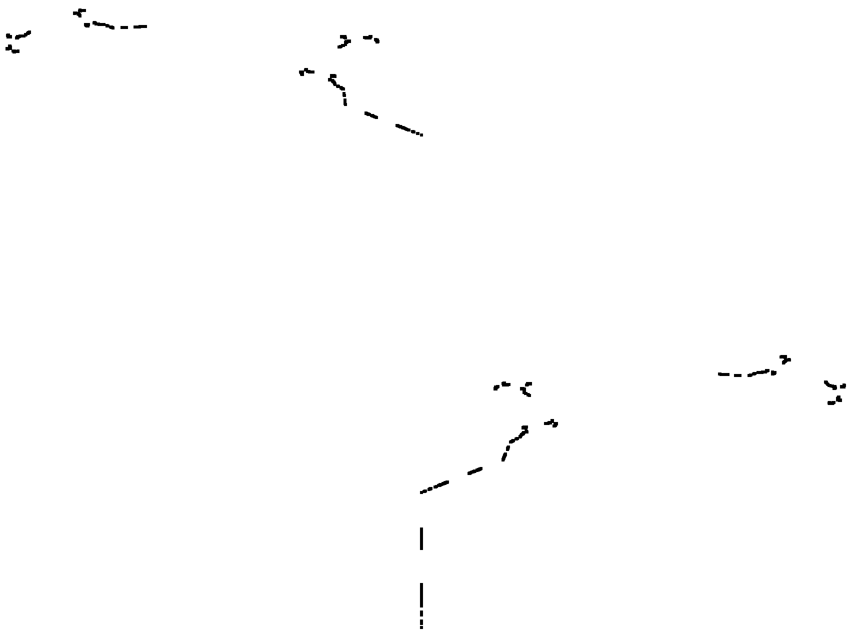
Barnsley Fern

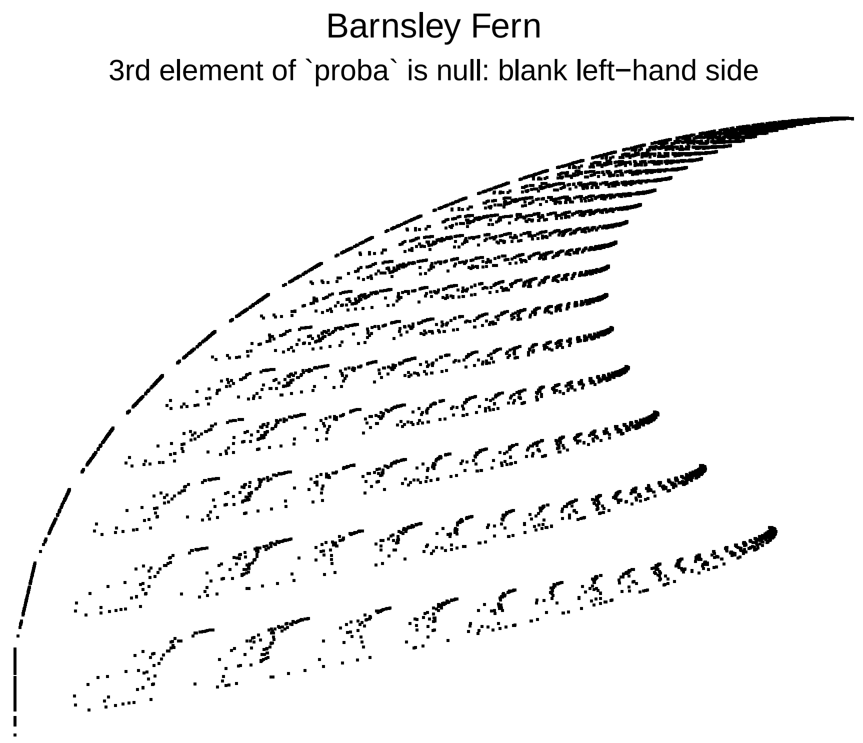
1st element of `proba` is null: blank stem

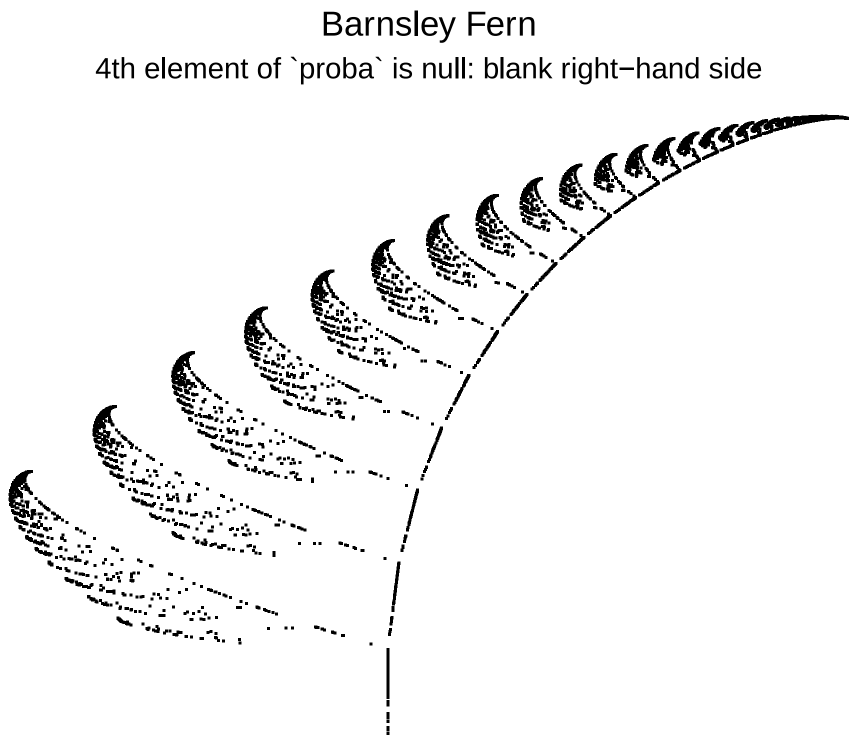


Barnsley Fern

2nd element of `proba` is null: no result



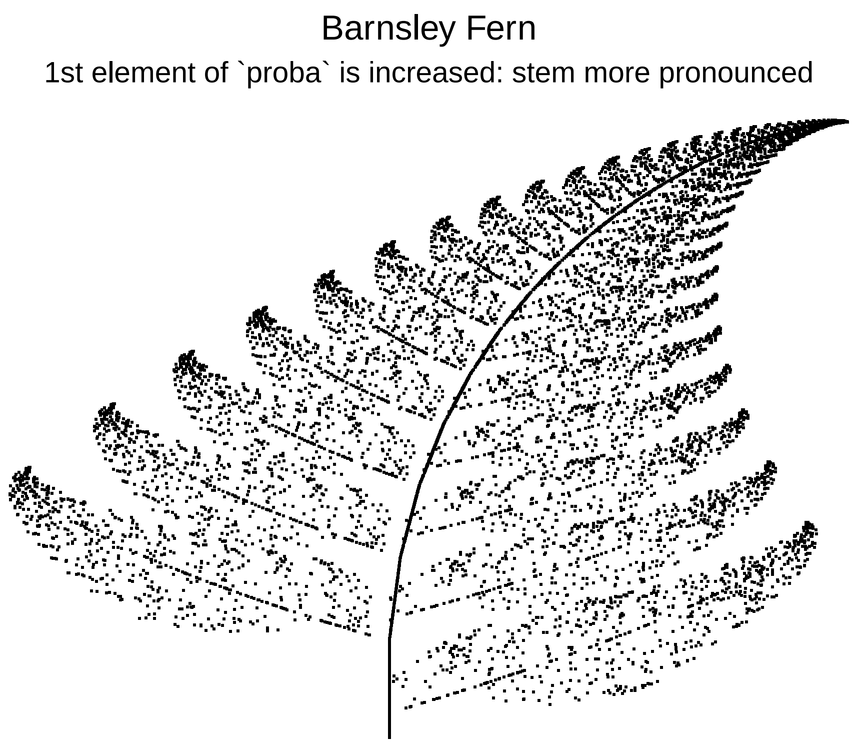




On the contrary, if we increase the value of an element of `proba`, we make the associated element of the fern more pronounced.

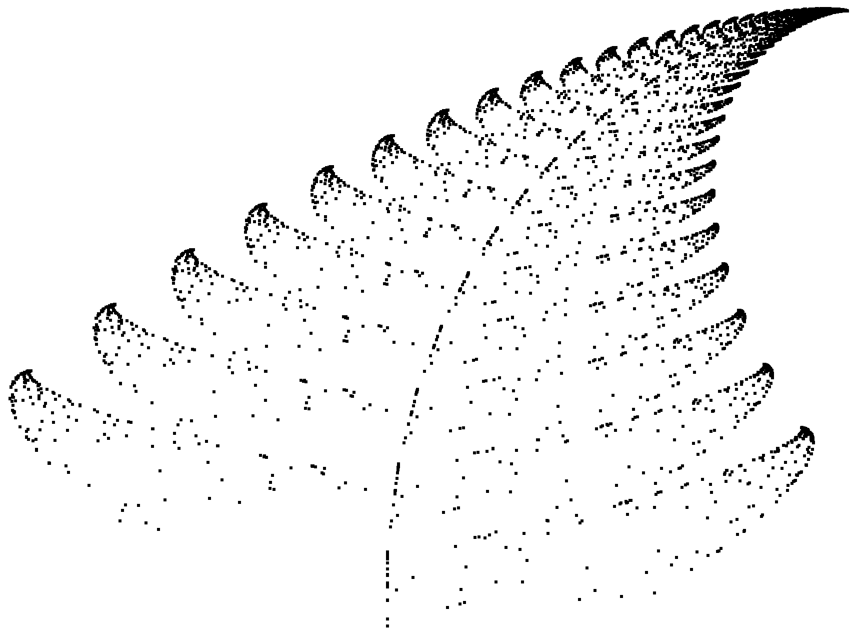
```
m <- 3
proba     <- c(0.01, 0.85, 0.07, 0.07)
```

```r
subtitles <- c("1st element of `proba` is increased: stem more pronounced"           ,
               "2nd element of `proba` is increased: leaves' ends more pronounced"  ,
               "3rd element of `proba` is increased: left-hand side more pronounced",
               "4th element of `proba` is increased: right-hand side more pronounced")

for(i in 1:4){

  proba_blank    <- proba
  proba_blank[i] <- proba_blank[i] * m
  subtitle       <- subtitles[i]

  print(generate_bf(proba    = proba_blank,
                    title    = "Barnsley Fern",
                    subtitle = subtitle))

}
```



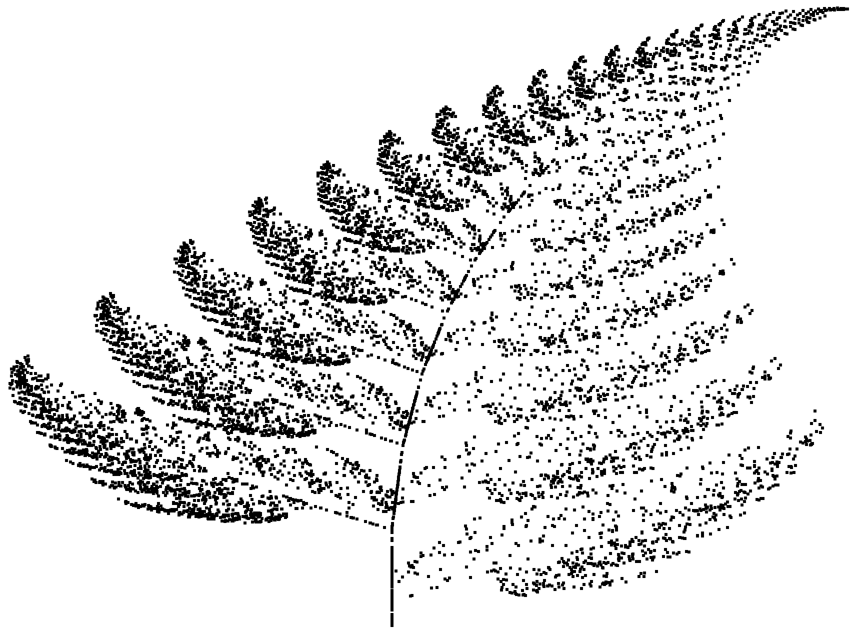
Barnsley Fern

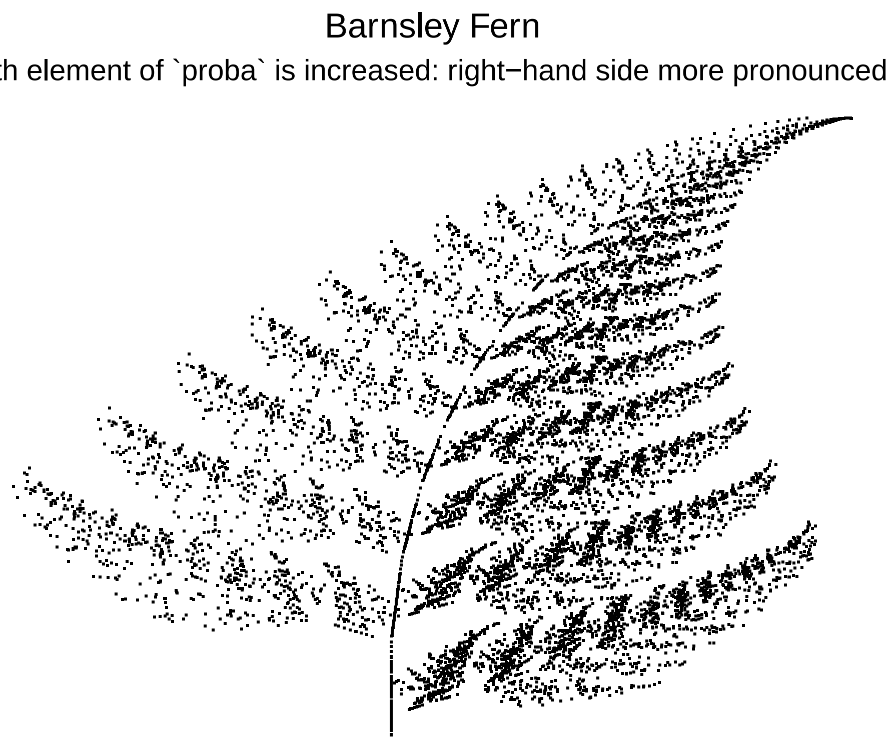
2nd element of \`proba\` is increased: leaves' ends more pronounced



Barnsley Fern

3rd element of \`proba\` is increased: left−hand side more pronounced





5 Summary

I designed three functions `generate_sg`, `generate_fractal` and `generate_bf` that respectively generate the Sierpinski Gasket, fractals with any number of vertices and the Barnsley Fern. Each function has parameters that allow us to tweak the shape of and pattern in the obtained fractal. Most important is the argument `p` of the functions `generate_sg` and `generate_fractal` which determines whether the figure is a fractal with a clear pattern or a bunch of points located in a chaotic manner.