L-Systems in R

Alexander Ptok

September 2, 2024

Introduction

The motivation was to try out the programming patterns teached in TLS to reproduce the image in TABOP. To do this R was the choosen programming language.

L-Systems

In 'The Algorithmic Beauty of Plants' (Prusinkiewicz, Przemyslaw and Lindenmayer, Aristid, 1990) we find the following definition:

> Let V denote an alphabet, V^* the set of all words over V , and V^+ the set of all nonempty words over V . A string OL-system is an ordered triplet $G = \langle V, \omega, P \rangle$ where V is the alphabet of the system, $\omega \in V^+$ is a nonempty word called the axiom and $P \subset V \times V^*$ is a finite set of productions. A production $(a, \chi) \in P$ is written as $a \to \chi$. The letter a and the word χ are called the predecessor and the successor of this production, respectively. It is assumed that for any letter $a \in V$, there is at least one word $\chi \in V^*$ such that $a \to \chi$. If no production is explicitly specified for a given predecessor $a \in V$, the identity production $a \to a$ is assumed to belong to the set of productions P. An OL-system is deterministic (noted DOL-system) if and only if for each $a \in V$ there is exactly one $\chi \in V^*$ such that $a \to \chi$.
>
> Let $= a_1 ... a_m$ be an arbitrary word over V. The word $\nu = \chi_1 ... \chi_m \in V^*$ is directly derived from (or generated by) μ, noted $\mu \Rightarrow \nu$, if and only if $a_i \to \chi_i$ for all $i = 1, ..., m$. A word ν is generated by G in a derivation of length n if there exists a developmental sequence of words $\mu_0, \mu_1, ..., \mu_n$ such that $\mu_0 = \omega, \mu_n = \nu$ and $\mu_0 \Rightarrow {}_1 \Rightarrow ... \Rightarrow \mu_n$.

It very soon becomes clear that we need to include letters with subscripts of only one character, like a_r and F_l into our alphabet. Therefore we transfrom the string ν into a list of letters that can have a subscripts. We do this wiht the funtion `letters_in_nu` that looks at the first three characters of ν and depending on the second character chooses between to cases. In one case there is just a letter and it is added to the the list `letters`. In the second case we have a letter with a subscript of one charcter and paste all three characters together and add it to the list `letters`.

The turtle interpreted characters, for now `F,f,n,+,-`, are not allowed as subscripts.

```
letters_in_nu <- function(nu) {
  letters <- c()
  while (nchar(nu) > 0) {
    a <- substring(nu,1,1)
    b <- substring(nu,2,2)
    c <- substring(nu,3,3)
    if (b == "_") {
      letters <- c(paste(a,b,c,sep=""), letters)
      nu <- substring(nu,4)
    } else {
      letters <- c(a, letters)
      nu <- substring(nu,2)
    }
  }
  rev(letters[letters != ""])
}

lsystem <- function(alphabet, axiom, productions) {
  function(n) {
    new_word <- ""
    while (n > 0) {
      for (symbol in letters_in_nu(axiom)) {
        new_word  <- paste(new_word,
                           productions[symbol],
                           sep="")
      }
      n <- n - 1
      axiom <- new_word
      new_word <- ""
    }
    axiom
  }
}
```

The above code is tangled into `lsystem.r`.

The Turtle Interpreter

In (Prusinkiewicz, Przemyslaw and Lindenmayer, Aristid, 1990) the turtle is defined:

> A state of the turtle is defined as a triplet (x, y, α), where the Cartesian coordinates (x, y) represent the turtle's position, and the angle α, called the heading, is interpreted as the direction in which the turtle is facing. Given the step size d and the angle increment δ, the turtle can respond to commands represented by the following symbols:
>
> **F** Move forward a step of length d. The state of the turtle changes to (x, y, α), where $x = x + dcos\alpha$ and $y = y + dsin\alpha$. A line segment between points (x, y) and (x', y') is drawn.
>
> **f** Move forward a step of length d without drawing a line.
>
> **+** Turn left by angle δ. The next state of the turtle is $(x, y, \alpha + \delta)$. The positive orientation of angles is counterclockwise.
>
> **-** Turn right by angle δ. The next state of the turtle is $(x, y, \alpha - \delta)$.
>
> Given a string ν, the initial state of the turtle (x_0, y_0, α_0) and fixed Interpretation parameters d and δ, the turtle interpretation of ν is the figure (set of lines) drawn by the turtle in response to the string ν.

```
turtle <- function(x, y, heading, stepsize, angle_increment) {

  x_orig <- x
  y_orig <- y
  heading_orig <- heading
  turtle_trace <- list(x1=c(x),x2=c(x),y1=c(y),y2=c(y))
  turtle_stack <- list(x=x,y=y,heading=heading)

  reset <- function() {
    x <<- x_orig
    y <<- y_orig
    turtle_trace <<- list(x1=c(x),x2=c(x),y1=c(y),y2=c(y))
    heading <<- heading_orig
  }

  forward <- function() {
    x <<- x + stepsize * cos(heading)
    y <<- y + stepsize * sin(heading)
  }

  forward_draw <- function() {
    turtle_trace$x1 <<- c(x, turtle_trace$x1)
    turtle_trace$y1 <<- c(y, turtle_trace$y1)
    forward()
    turtle_trace$x2 <<- c(x, turtle_trace$x2)
    turtle_trace$y2 <<- c(y, turtle_trace$y2)
  }

  turn_right <- function() {
    heading <<- heading - angle_increment
  }

  turn_left <- function() {
    heading <<- heading + angle_increment
```

```
    }

    draw_turtle <- function(ls) {
      print(c(range(c(turtle_trace$x1,turtle_trace$x2)),
              range(c(turtle_trace$y1,turtle_trace$y2))))

    plot(x=range(c(turtle_trace$x1,turtle_trace$x2)),
           y=range(c(turtle_trace$y1,turtle_trace$y2)),
           type="n", ann=FALSE, axes=FALSE)

      for (i in 1:length(turtle_trace$x1)) {
        lines(x=c(turtle_trace$x1[i], turtle_trace$x2[i]),
              y=c(turtle_trace$y1[i], turtle_trace$y2[i]))
      }
    }

    print_turtle_trace <- function() {
#      print(turtle_trace)
      print(heading*(180/pi))
    }

    push <- function() {
      turtle_stack[[length(turtle_stack) + 1]] <<-
        list(x = x, y = y, heading = heading)
    }

    pop <- function() {
      last_turtle <- turtle_stack[[length(turtle_stack)]]
      turtle_stack[[length(turtle_stack)]] <<- NULL
      x <<- last_turtle$x
      y <<- last_turtle$y
      heading <<- last_turtle$heading
    }

    function_table <-
      list("F" = forward_draw,
           "f" = forward,
           "-" = turn_right,
           "+" = turn_left,
           "n" = reset,
           "d" = draw_turtle,
           "p" = print_turtle_trace,
           "[" = push,
           "]" = pop)

    iter_over_nu <- function(nu) {
      for (i in 1:nchar(nu)) {
        a <- substring(nu,i,i)
        if (a %in% names(function_table)) {
          function_table[[a]]()
        }
      }
    }

    function(nu) {
      iter_over_nu(nu)
    }
  }
```

The above code is tangled into `turtle.r`

We can now source the two files `lsystem.r` and `turtle.r` and produce some turtle drawings.

Examples

Koch Island

```
source("lsystem.r")
source("turtle.r")
alphabet <- c("F", "f", "+", "-")
axiom <- c("F-F-F-F")
productions <- c("F" = "F-F+F+FF-F-F+F",
                 "+" = "+",
                 "-" = "-",
                 "f" = "f")
l <- lsystem(alphabet, axiom, productions)
t <- turtle(0, 0, pi, 1, pi/2)
t(l(3))
png("koch-island.png")
t("d")
dev.off()
```

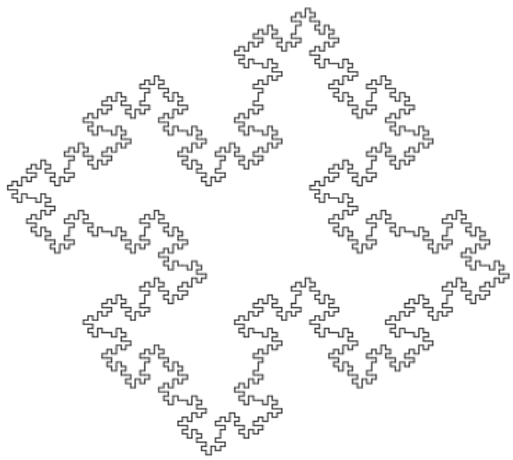
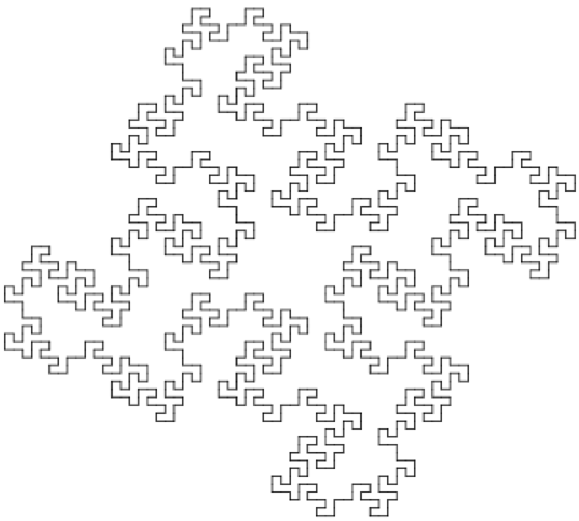


Figure 1: The Koch Island

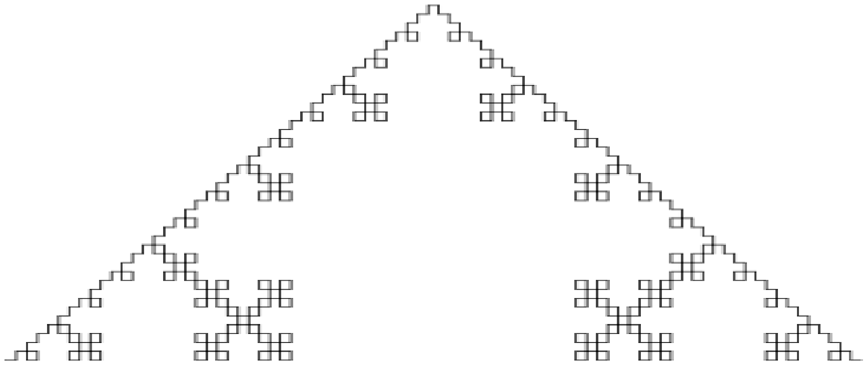
```
axiom <- c("F-F-F-F")
productions <- c("F" = "F-F+F+FF-F-F+F")
                 "+" = "+",
                 "-" = "-",
                 "f" = "f")
n <- 3
```

```
source("lsystem.r")
source("turtle.r")
alphabet <- c("F", "f", "+", "-")
axiom <- c("F-F-F-F")
```

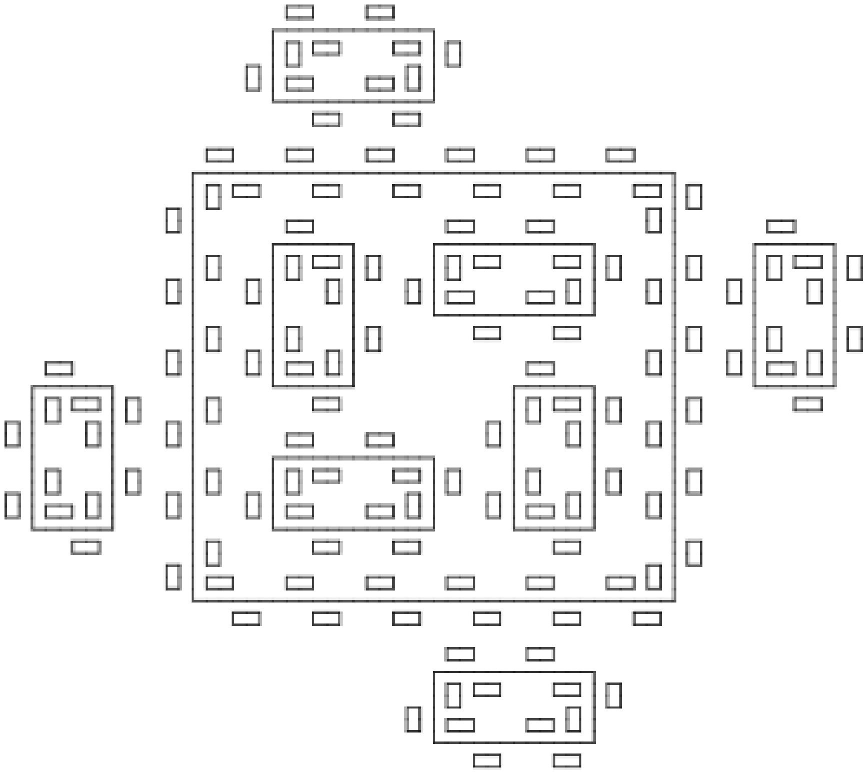
```
productions <- c("F" = "F+FF-FF-F-F+F+FF-F-F+F+FF+FF-F",
                 "+" = "+",
                 "-" = "-",
                 "f" = "f")
l <- lsystem(alphabet, axiom, productions)
t <- turtle(0, 0, pi, 1, pi/2)
t(l(2))
png("koch-island-variation.png")
t("d")
dev.off()
```



```
source("lsystem.r")
source("turtle.r")
alphabet <- c("F", "f", "+", "-")
axiom <- c("-F")
productions <- c("F" = "F+F-F-F+F",
                 "+" = "+",
                 "-" = "-",
                 "f" = "f")
l <- lsystem(alphabet, axiom, productions)
t <- turtle(0, 0, pi/2, 1, pi/2)
t(l(4))
png("snowflake-variation.png")
t("d")
dev.off()
```



```
source("lsystem.r")
source("turtle.r")
alphabet <- c("F", "f", "+", "-")
axiom <- c("F+F+F+F")
productions <- c("F" = "F+f-FF+F+FF+Ff+FF-f+FF-F-FF-Ff-FFF",
                 "+" = "+",
                 "-" = "-",
                 "f" = "ffffff")
l <- lsystem(alphabet, axiom, productions)
t <- turtle(0, 0, pi/2, .1, pi/2)
t(l(2))
png("islands-and-lakes.png")
t("d")
dev.off()
```



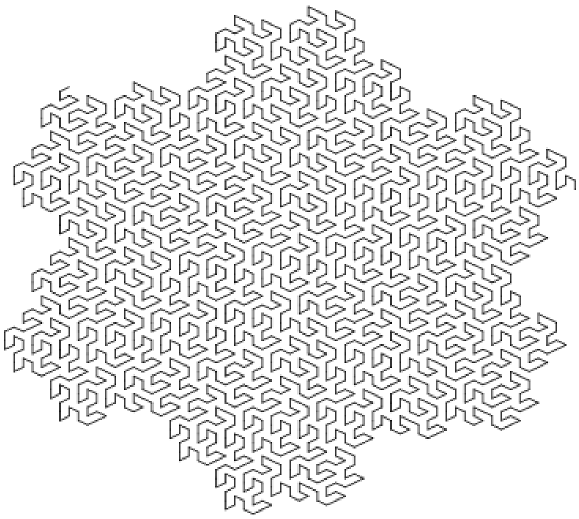
Drawing the Turtle Trace

We want a function that given the parameters name, l-system, turtle and number of recursions n draws the turtle traces into a png file.

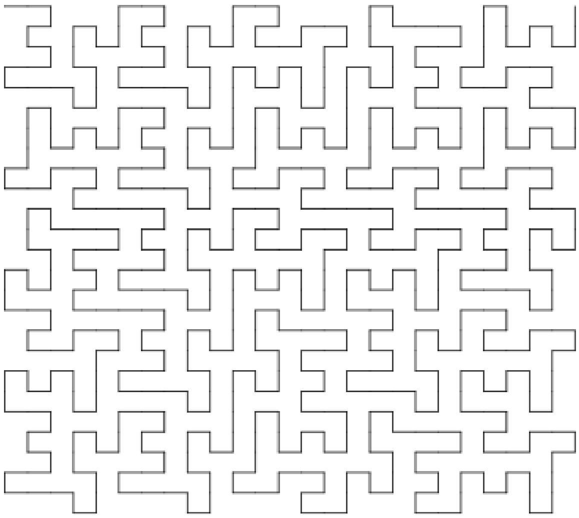
FASS

```
source("lsystem.r")
source("turtle.r")
alphabet <- c("F_l", "F_r", "f", "+", "-", "[", "]")
axiom <- c("F_l")
productions <- c("F_l" = "F_l+F_r++F_r-F_l--F_lF_l-F_r+",
                 "F_r" = "-F_l+F_rF_r++F_r+F_l--F_l-F_r",
                 "+" = "+",
                 "-" = "-",
                 "f" = "f",
```

```
                 "[" = "[",
                 "]" = "]")
l <- lsystem(alphabet, axiom, productions)
t <- turtle(0, 0, pi/2, .1, (60*pi)/180)
t(l(4))
png("hexagonal-gosper-curve.png")
t("d")
dev.off()
```

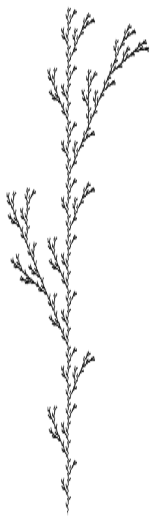


```
source("lsystem.r")
source("turtle.r")
alphabet <- c("F_l", "F_r", "f", "+", "-", "[", "]")
axiom <- c("-F_l")
productions <- c("F_l" = paste(
                   "F_lF_l-F_r-F_r+F_l+F_l-F_r-F_r",
                    "F_l+F_r+F_lF_lF_r-F_l+F_r+F_lF_l",
                    "+F_r-F_lF_r-F_r-F_l+F_l+F_rF_r-",
                   sep=""),
                 "F_r" = paste(
                   "+F_lF_l-F_r-F_r+F_l+F_lF_r+F_l-",
                   "F_rF_r-F_l-F_r+F_lF_rF_r-F_l-F_r",
                   "F_l+F_l+F_r-F_r-F_l+F_l+F_rF_r",
                   sep=""),
                 "+" = "+",
                 "-" = "-",
                 "f" = "f",
                 "[" = "[",
                 "]" = "]")
l <- lsystem(alphabet, axiom, productions)
t <- turtle(0, 0, pi/2, .1, (90*pi)/180)
t(l(2))
png("quadratic-gosper-curve.png")
t("d")
dev.off()
```

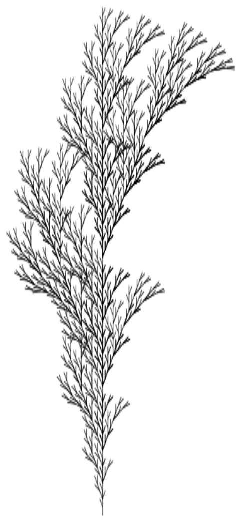


Branching

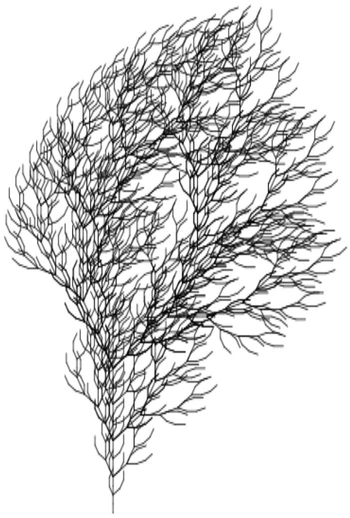
```
source("lsystem.r")
source("turtle.r")
alphabet <- c("F", "f", "+", "-", "[", "]")
axiom <- c("F")
productions <- c("F" = "F[+F]F[-F]F",
                 "+" = "+",
                 "-" = "-",
                 "f" = "f",
                 "[" = "[",
                 "]" = "]")
l <- lsystem(alphabet, axiom, productions)
t <- turtle(0, 0, pi/2, .1, (25.7*pi)/180)
t(l(5))
png("branching1.png")
t("d")
dev.off()
```



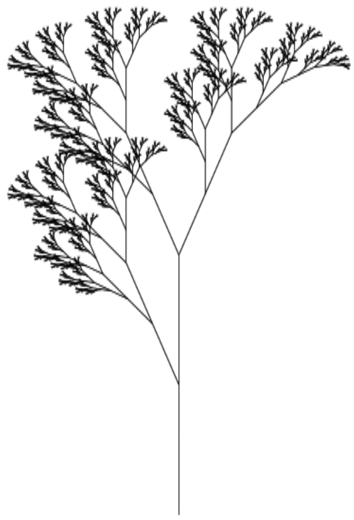
```
source("lsystem.r")
source("turtle.r")
alphabet <- c("F", "f", "+", "-", "[", "]")
axiom <- c("F")
productions <- c("F" = "F[+F]F[-F][F]",
                 "+" = "+",
                 "-" = "-",
                 "f" = "f",
                 "[" = "[",
                 "]" = "]")
l <- lsystem(alphabet, axiom, productions)
t <- turtle(0, 0, pi/2, 1, (20*pi)/180)
t(l(5))
png("branching2.png")
t("d")
dev.off()
```



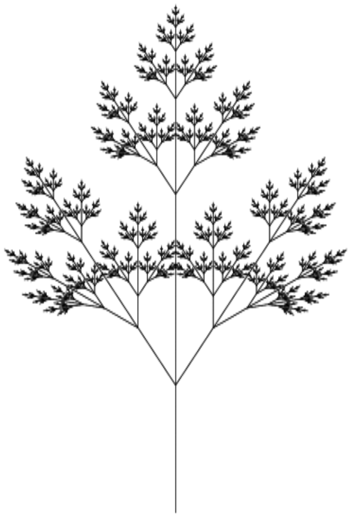
```
source("lsystem.r")
source("turtle.r")
alphabet <- c("F", "f", "+", "-", "[", "]")
axiom <- c("F")
productions <- c("F" = "FF-[-F+F+F]+[+F-F-F]",
                 "+" = "+",
                 "-" = "-",
                 "f" = "f",
                 "[" = "[",
                 "]" = "]")
l <- lsystem(alphabet, axiom, productions)
t <- turtle(0, 0, pi/2, 1, (22.5*pi)/180)
t(l(4))
png("branching3.png")
t("d")
dev.off()
```



```
source("lsystem.r")
source("turtle.r")
alphabet <- c("X", "F", "f", "+", "-", "[", "]")
axiom <- c("X")
productions <- c("X" = "F[+X]F[-X]+X",
                 "F" = "FF",
                 "+" = "+",
                 "-" = "-",
                 "f" = "f",
                 "[" = "[",
                 "]" = "]")
l <- lsystem(alphabet, axiom, productions)
t <- turtle(0, 0, pi/2, 1, (20*pi)/180)
t(l(8))
png("branching4.png")
t("d")
dev.off()
```



```
source("lsystem.r")
source("turtle.r")
alphabet <- c("X", "F", "f", "+", "-", "[", "]")
axiom <- c("X")
productions <- c("X" = "F[+X][-X]FX",
                 "F" = "FF",
                 "+" = "+",
                 "-" = "-",
                 "f" = "f",
                 "[" = "[",
                 "]" = "]")
l <- lsystem(alphabet, axiom, productions)
t <- turtle(0, 0, pi/2, 1, (25.7*pi)/180)
t(l(8))
png("branching5.png")
t("d")
dev.off()
```



```
source("lsystem.r")
source("turtle.r")
alphabet <- c("X", "F", "f", "+", "-", "[", "]")
axiom <- c("X")
productions <- c("X" = "F-[[X]+X]+F[+FX]-X",
                 "F" = "FF",
                 "+" = "+",
                 "-" = "-",
                 "f" = "f",
                 "[" = "[",
                 "]" = "]")
l <- lsystem(alphabet, axiom, productions)
t <- turtle(0, 0, pi/2, 1, (22.5*pi)/180)
t(l(6))
png("branching6.png")
t("d")
dev.off()
```

Literature

Prusinkiewicz, Przemyslaw and Lindenmayer, Aristid (1990). The algorithmic beauty of plants, Springer.