L-Systems in R

Alexander Ptok

August 31, 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

First we implmented DOL-Systems with natural recursions.

```
lsystem_rec  <- function(alphabet, axiom, productions) {

  derivation  <- function(axiom, new_word) {
    a <- substring(axiom,1,1)
    if (a  == "") {
      new_word
    } else derivation(substring(axiom,2),
                      paste(new_word,
                            productions[a],
                            sep=""))
  }

  derive_n  <- function(axiom, n) {
    if (n == 0) {
      axiom
    } else derive_n(derivation(axiom, ""),
                    n - 1)
  }

  function(n) {
    derive_n(axiom, n)
  }
}
```

Turtle

The function turtle gives back a closure. A closure is a poor mans object. You can give the function returned by turtle a character as argument, like sending a message to an object. The `<<-` operator assigns the variable on the right not in the namespace of the function but one level up, in the namespace of the closure.

First we set some variables that compose the context of the returned function.

```
turtle <- function(x, y, alpha, stepsize, delta) {

  x_orig <- x
  y_orig <- y
  alpha_orig <- alpha
  ret <- list(x1=c(),x2=c(),y1=c(),y2=c())
```

Than we define some methods as functions.

```
reset <- function() {
  ret <<- list(x1=c(),x2=c(),y1=c(),y2=c())
  x <<- x_orig
  y <<- y_orig
  alpha <<- alpha_orig
}

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

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

turn_right <- function() {
  alpha <<- alpha - delta
}

turn_left <- function() {
  alpha <<- alpha + delta
}
```

We put all the functions for methods into a function lookup table.

```
function_table <-
  list("F" = forward_draw,
       "f" = forward,
       "-" = turn_right,
       "+" = turn_left,
       "n" = reset)
```

Now we can recure over the given string of inputs nu. If we find a function in the function table we call it, else we ignore the symbol.

```
  rec_over_nu <- function(nu) {
    a <- substring(nu,1,1)
    if (nu == "") {
      ret
    } else {
      if (a %in% names(function_table)) {
        function_table[[a]]()
      }
      rec_over_nu(substring(nu,2))
    }
  }

  function(nu) {
    rec_over_nu(nu)
  }
}
```

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.

```
draw_turtle <- function(name, turtle, lsystem, n) {
  ls <- turtle(lsystem(n))
  turtle("n")
  draw_turtle_rec <- function(ls) {
    if (length(ls$x1) == 0) {
      return()
    } else {
      lines(x=c(ls$x1[1], ls$x2[1]), y=c(ls$y1[1], ls$y2[1]))
      draw_turtle_rec(list(x1=ls$x1[-1],
                           x2=ls$x2[-1],
                           y1=ls$y1[-1],
                           y2=ls$y2[-1]))
    }
  }
  png(name)
  plot(range(c(ls$x1, ls$x2)), range(c(ls$y1, ls$y2)), type="n", ann=FALSE, axes=FALSE)
  draw_turtle_rec(ls)
  dev.off()
}
```

After all the above was tangled into `lsystem-rec.r` we can evaluate the following lines to generate a koch curve and show it here.

```
source("lsystem-rec.r")
t <- turtle(0,0,pi,1,pi/2)
l <- lsystem_rec(alphabet="Ff+-",
                 axiom="F-F-F-F",
                 productions=c("F" = "F-F+F+FF-F-F+F",
                               "+" = "+",
                               "-" = "-",
                               "f" = "f"))
draw_turtle("koch-curve.png", t, l, 2)
```

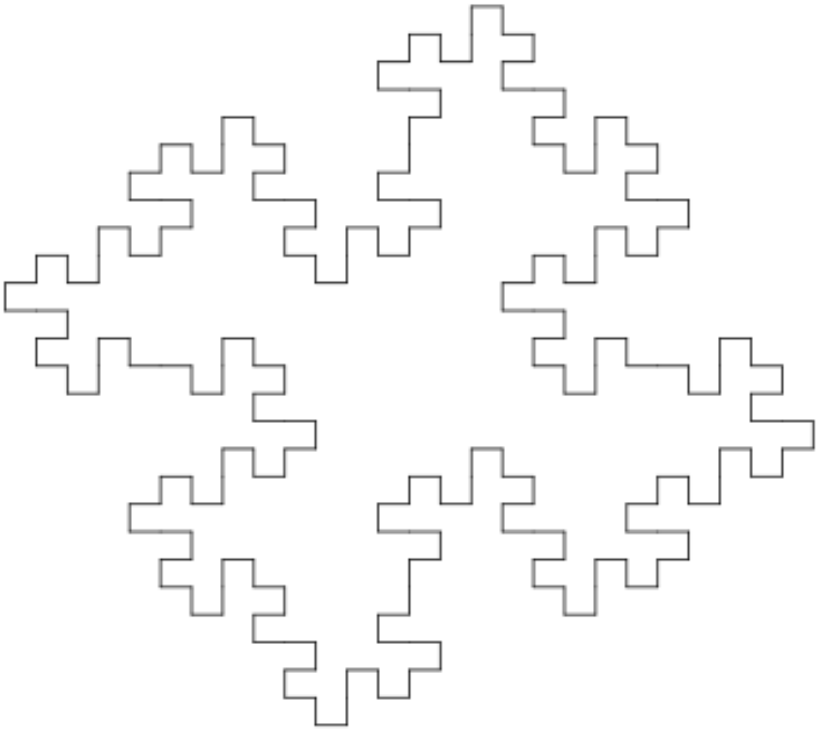

Figure 1: Koch Curve