unesp

UNIVERSIDADE ESTADUAL PAULISTA

Fractais e Sistemas-L

Giovanni Moraes Biban

INSTITUTO DE GEOCIÊNCIAS E CIÊNCIAS EXATAS

RIO CLARO

2025

Universidade Estadual Paulista
"Júlio de Mesquita Filho"
Instituto de Geociências e Ciências Exatas
Câmpus de Rio Claro

Giovanni Moraes Biban

Fractais e Sistemas-L

Dissertação de Mestrado apresentada ao Instituto de Geociências e Ciências Exatas do Câmpus de Rio Claro, da Universidade Estadual Paulista "Júlio de Mesquita Filho", como parte dos requisitos para obtenção do título de Mestre em Matemática.

Orientador
Prof. Dr. Thiago de Melo

Rio Claro/SP
2025

B581f

Biban, Giovanni
Fractais e sistemas-L / Giovanni Biban. -- Rio Claro, 2025
65 p. : il., tabs., fotos

Dissertação (mestrado profissional) - Universidade Estadual Paulista (UNESP), Instituto de Geociências e Ciências Exatas, Rio Claro
Orientador: Thiago Melo

1. Fractais. 2. Sistemas L. 3. Palavras fractais. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Dados fornecidos pelo autor(a).

Impacto potencial desta pesquisa

Ao longo deste trabalho, foram abordados os sistemas Lindenmeyer, curvas fractais e algumas possíveis interseções entre os temas, desenvolvendo métodos para construções de curvas fractais a partir dos sistemas-L. Além disso, foram desenvolvidos pelo autor, códigos em linguagem Python capazes de gerar algumas das curvas desenvolvidas, possibilitando a visualização mais precisa e agilizada. Em conjunto, o trabalho permite uma nova abordagem dos fractais, que pode ser utilizada tanto no Ensino Médio quanto no Ensino Superior como uma introdução ao tema. Além disso, devido à sua interdisciplinaridade, pode ser aplicada como uma ponte para estudos entre a Matemática, a Biologia e a Computação, entre outros.

Potential impact of this research

Throughout this work, Lindenmayer systems, fractal curves, and some possible intersections between these topics were discussed, developing methods for constructing fractal curves based on L-systems. In addition, the author developed Python code capable of generating some of the curves, enabling more precise and efficient visualization. Together, the work provides a new approach to fractals, which can be used in both secondary and higher education as an introduction to the topic. Furthermore, due to its interdisciplinary nature, it can serve as a bridge for studies between Mathematics, Biology, and Computer Science.

Universidade Estadual Paulista
"Júlio de Mesquita Filho"
Instituto de Geociências e Ciências Exatas
Câmpus de Rio Claro

Giovanni Moraes Biban

Fractais e Sistemas-L

Dissertação de Mestrado apresentada ao Instituto de Geociências e Ciências Exatas do Câmpus de Rio Claro, da Universidade Estadual Paulista "Júlio de Mesquita Filho", como parte dos requisitos para obtenção do título de Mestre em Matemática.

Comissão Examinadora

Prof. Dr. Thiago de Melo
Orientador

Prof(a). Dr(a). Suzete Maria Silva Afonso
IGCE - Unesp Rio Claro

Prof(a). Dr(a). Tatiana Miguel Rodrigues
FC - Unesp Bauru

Conceito: Aprovado

Rio Claro (SP), 31 de março de 2025

Agradecimentos

Agradeço ao meu orientador, Professor Doutor Thiago de Melo, por seus ensinamentos e sua dedicação ao longo deste trabalho. Seu domínio do tema e seus conhecimentos me permitiram alcançar este resultado.

Agradeço também aos professores do PGMAT e todos aqueles que ao longo de meu período de estudos compartilharam seus conhecimentos e experiências, me permitindo alçar novos horizontes.

Aos membros da banca examinadora, Professora Doutora Suzete Maria Afonso e Professora Doutora Tatiana Miguel Rodrigues, por todo conhecimento e apontamentos visando tornar deste trabalho o melhor possível, meus mais profundos agradecimentos.

Agradeço a meus amigos que estiveram sempre em meu apoio e incentivo, em especial Marcos Said e Isabela Ferreira, por trilharem esse caminho ao meu lado.

À minha família, que sempre me apoiou e mostrou o quanto o conhecimento é a chave para nosso crescimento, minha sincera gratidão. Em particular, à minha mãe, minha mais importante professora.

Nuvens não são esferas, montanhas não são cones,
costeiras não são círculos, nem raios viajam em linha reta.
Benoit Mandelbrot

Resumo

O principal objetivo deste trabalho é o estudo de construções fractais por meio de palavras geradas a partir de um alfabeto, baseando-se nos métodos de sistemas-L. Com esse foco, desenvolveremos métodos de construções para alguns fractais, como a curva do dragão, a curva e o floco de neve de Koch, em sua forma original e com algumas variações. Além disso, estudaremos propriedades algébricas destas palavras e propriedades geométricas dos fractais, utilizando linguagem Python para gerar e visualizar as figuras.

Palavras-chave: Fractais. Sistemas-L. Palavras Fractais.

Abstract

The main goal of this work is to study of fractal constructions using words created by an alphabet, based on L-systems methods. For this, we will develop construction methods for some fractals, such as the dragon curve, Koch's curve and snowflake, in their original form and with modifications. Besides that, we will also study algebric structures of the words and geometric properties of the fractals, using Python programming language to create and visualize the figures.

Keywords: Fractals. L-systems. Fractal Words.

Lista de Figuras

1.1 *Anabaena catenula* 14
1.3 Triângulo de Sierpiński 16
1.4 Samambaia de Barnsley 18
1.5 Comparação entre vasos sanguíneos e ramificação fractal 19

2.1 Curva de Fibonacci de iteração 29 23
2.2 Curva do Dragão com 0 iterações 23
2.3 Curva do Dragão com 1 e 2 iterações 24
2.4 Iteração da curva do dragão de 2 para 3 24
2.5 Curva do Dragão com 0, 2, 4 e 8 iterações 25

3.1 Curvas de Koch com diferentes iterações 27
3.2 Diferentes iterações com ângulos destacados 27
3.3 Curva de Koch de 1 iteração com ângulos destacados 28
3.4 Construção da curva de Koch através da palavra f_1 29
3.5 Curvas 6-poligonais de Koch com diferentes iterações 32
3.6 Curva 6-poligonal de Koch com ângulos destacados 33
3.7 Curvas 3, 4 e 5-poligonais de iteração 1 33
3.8 Floco de neve de Koch com iterações diferentes 35
3.9 Floco de neve de Koch com destaques 36
3.10 Flocos de neve l-poligonais de Koch de iteração 3 37
3.11 Curva aleatória de Koch 38
3.12 Curvas de Koch de iteração 2 em sentidos opostos 39
3.13 Curva de Koch alternada 39
3.14 Curva alternada de Koch de iteração $k = 2$ com palavra destacada 45

4.1 Curva do dragão de iteração 7 48
4.2 Curva de Koch de iteração 3 51
4.3 Curva 4-poligonal de iteração 3 55
4.4 Curva de Koch alternada de iteração 3 59

Lista de Tabelas

2.1 Comparação $|f_n|$ e os números da sequência de Fibonacci 22

3.1 Palavras de Koch 29
3.2 Palavras de Koch alternadas 40

Lista de Programas

4.1 Curva do Dragão 47
4.2 Animação em Manim para a curva do dragão 49
4.3 Curva de Koch 51
4.4 Animação em Manim para a curva de Koch 53
4.5 Curva l-poligonal de Koch 55
4.6 Animação em Manim para a Curva l-poligonal de Koch 57
4.7 Curva alternada de Koch 59
4.8 Animação em Manim para a curva alternada de Koch 62
4.9 Animação em Manim para a curva semi-alternada de Koch 63

Sumário

1 Introdução **13**
1.1 Fractais 16

2 Sistemas-L: definições e exemplos **20**

3 Curvas fractais clássicas e suas variações **26**
3.1 Curva de Koch 26
3.1.1 Comprimento da palavra de Koch 30
3.2 Curva l-poligonal de Koch 31
3.3 Floco de neve de Koch e suas variações 34
3.4 Curva de Koch alternada 38

4 Programação em Python **46**
4.1 Curva do dragão 46
4.1.1 Animação 47
4.2 Curva de Koch 49
4.2.1 Animação 51
4.3 Curva l-poligonal de Koch 54
4.3.1 Animação 56
4.4 Curva alternada de Koch 58
4.4.1 Animação 60
4.4.2 Animação 62

Referências **65**

1 Introdução

Em 1968, o biólogo Aristid Lindenmeyer (1925–1989) criou o sistema Lindenmayer, ou sistema-L, como forma de padronizar o crescimento de um filamento de bactérias da espécie *Anabaena catenula*. Inicialmente, os sistemas-L foram utilizados para descrever o crescimento e reprodução de plantas e bactérias através de um método axiomático e recursivo. Este método é baseado na construção e reconstrução de "strings", que neste trabalho serão chamadas de "palavras", as quais são conjuntos de dígitos em uma organização específica, predefinidos por um alfabeto, um conjunto de regras e um axioma. Definimos os elementos necessários a seguir:

- Alfabeto: Conjunto de dígitos com significados e regras definidas;
- Regra: Processo iterativo de substituição aplicado em cada dígito;
- Axioma: Dígito ou sequência de dígitos iniciais de um sistema;
- Palavra[1]: sequência obtida por iterações sucessivas a partir do axioma, conforme as regras do sistema, podendo ser finita ou infinita.

As situações modeladas com sistemas-L eram ilustradas listando as sucessivas palavras geradas ao longo do processo iterativo. Para se obter uma visualização adicional era necessária a interpretação humana dos dígitos como sendo módulos diferentes, através da imaginação ou de desenhos manuais [1, p. 29].

Na Figura 1.1 podemos ver filamentos da bactéria *Anabaena catenula*, objeto inicial de estudo dos sistemas-L. Esta espécie apresenta dois estágios reprodutivos: células mãe, que estão aptas para a reprodução, e células filhas, que são as células que não estão aptas para a reprodução. Além de seus estágios reprodutivos, as células também, apresentam uma lógica posicional, onde as células mãe geram células filhas para duas direções opostas apenas, que por praticidade consideraremos esquerda e direita a partir da célula inicial, gerando assim um único filamento.

A construção do filamento ocorre a partir de uma única célula que se separou de algum outro anterior. Como mencionado, esta célula possui uma lógica posicional e ,portanto, tomaremos a direita como direção inicial nos exemplos apresentados nesta dissertação. A partir disso, a reprodução seguirá os seguintes passos:

- Em toda etapa reprodutiva a célula mãe gerará uma célula filha na direção em que está orientada e depois irá se orientar para a direção oposta;

[1] Neste trabalho, as palavras de um sistema-L terão uma tipografia específica para facilitar a diferenciação, com exceção das palavras presentes em imagens não feitas pelo autor. Exemplo: `palavra`.

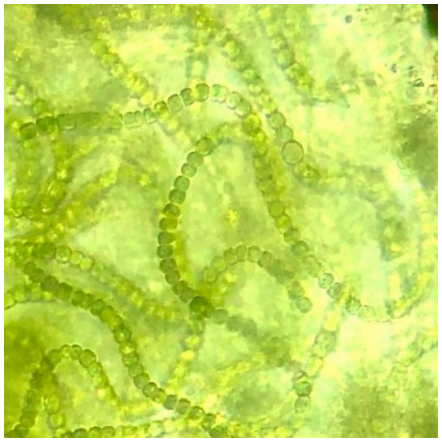


Figura 1.1: *Anabena catenula*

Fonte: Atlas of Living Australia

- Uma célula filha terá a mesma orientação de sua célula mãe quando a gerou;
- Após ser gerada, uma célula filha permanece uma etapa reprodutiva completa para se desenvolver e tornar-se uma célula mãe (mantendo sua orientação).

Portanto, uma célula mãe orientada para a direita gerará à sua direita uma célula filha orientada para a direita e se orientará para a esquerda. Na próxima etapa gerará à sua esquerda uma célula filha orientada para a esquerda e se orientará para a direita novamente, enquanto a célula filha gerada na etapa anterior se desenvolve para se tornar uma célula mãe e ficar apta para a reprodução.

Ao perceber isso, Lindenmayer constatou que poderia descrever todas essas etapas através de um sistema recursivo, desenvolvendo assim o primeiro sistema-L. Nesse sistema as células mãe são representadas pela sigla `a` e as células filhas por `b`. Além disso, para representar as orientações de cada célula, serão utilizados os índices `r` para direita (right) e `l` para esquerda (left), ou seja, uma célula mãe orientada para direita, será descrita por $\mathtt{a_r}$. Podemos observar um esquema visual dessa construção na Figura 1.2.

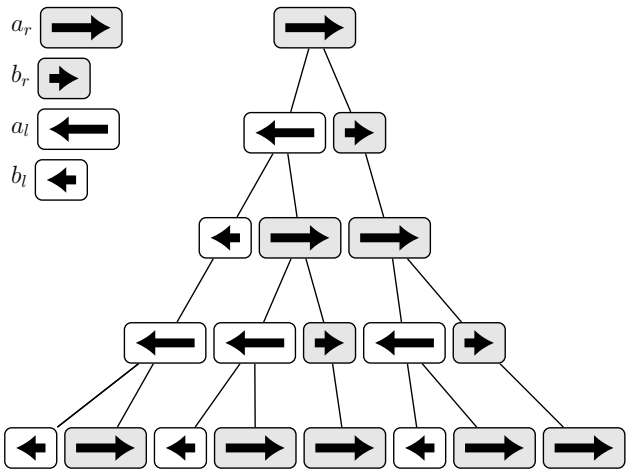


Figura 1.2: Esquema de crescimento de *Anabaena catenula*

Fonte: The Algorithmic Beauty of Plants

As estruturas geradas constituem cadeias unidimensionais de retângulos, reproduzindo assim a sequência de símbolos das palavras correspondentes. Contudo, o modelo apresentado não exige que os filamentos sejam retilíneos, sendo assim pode-se fazer uma representação curvilínea dos mesmos, aproximando-a mais das imagens reais destes organismos microscópicos.

Observando o esquema da Figura 1.2 percebemos os padrões de mudança nas células do filamento, podendo serem descritos da seguinte forma.

$\mathtt{b_r}$ e $\mathtt{b_l}$: As células filhas, durante uma etapa de reprodução, possuem apenas a função de se tornarem células-mães, mantendo sua orientação. Ou seja, $\mathtt{b_r} \to \mathtt{a_r}$ e $\mathtt{b_l} \to \mathtt{a_l}$.

$\mathtt{a_r}$: A célula-mãe $\mathtt{a_r}$, passa por dois eventos durante uma etapa reprodutiva. Os quais consistem em gerar uma célula filha, $\mathtt{b_r}$, à sua direita e em seguida, trocar sua própria orientação, tornando-se $\mathtt{a_l}$. Portanto $\mathtt{a_r} \to \mathtt{a_l b_r}$.

$\mathtt{a_l}$: A célula-mãe $\mathtt{a_l}$, assim como $\mathtt{a_r}$, passa por dois eventos durante uma etapa reprodutiva. Os quais consistem em gerar uma célula filha, $\mathtt{b_l}$, à sua esquerda e em seguida, trocar sua própria orientação, tornando-se $\mathtt{a_r}$. Portanto, $\mathtt{a_l} \to \mathtt{b_l a_r}$.

A partir disso Lindenmayer montou o seguinte sistema:

$$\begin{aligned} \text{Alfabeto:} &\quad \{\mathtt{a_r}, \mathtt{a_l}, \mathtt{b_r}, \mathtt{b_l}\} \\ \text{Axioma:} &\quad \mathtt{a_r} \\ \text{Regras:} &\quad \mathtt{a_r} \to \mathtt{a_l b_r} \\ &\quad \mathtt{a_l} \to \mathtt{b_l a_r} \\ &\quad \mathtt{b_r} \to \mathtt{a_r} \\ &\quad \mathtt{b_l} \to \mathtt{a_l} \end{aligned}$$

Após algumas iterações do sistema, teremos as seguintes palavras:

$$\begin{aligned} &\mathtt{a_r} \\ &\mathtt{a_l b_r} \\ &\mathtt{b_l a_r a_r} \\ &\mathtt{a_l a_l b_r a_l b_r} \\ &\mathtt{b_l a_r b_l a_r a_r b_l a_r a_r} \\ &\ldots \end{aligned}$$

que irá descrever o padrão do filamento de bactéria, visualizado na Figura 1.2

Em resumo, ao longo deste trabalho, além de sistemas Lindenmeyer, foram abordadas também, curvas fractais e algumas possíveis interseções entre os temas, desenvolvendo métodos para a construções de curvas fractais a partir dos sistemas-L. Além disso, foram desenvolvidos pelo autor códigos em linguagem Python capazes de gerar algumas das curvas desenvolvidas, possibilitando a visualização mais precisa e agilizada. Em conjunto, o trabalho permite uma nova abordagem dos fractais, podendo ser utilizado, tanto no Ensino Médio quanto no Ensino Superior, como uma introdução ao tema, bem como, devido à sua interdisciplinariedade, ser aplicado como uma relação para estudos entre a Matemática e Biologia, ou Computação.

A seguir, apresentamos os conceitos necessários para o entendimento da curvas criadas com a programação do Capítulo 4.

1.1 Fractais

Desde seu início, a matemática foi utilizada como uma ferramenta para superar dificuldades enfrentadas pela civilização humana a fim de resolver seus problemas. Mais especificamente, com o desenvolvimento da geometria foi possível regular as divisões de terras para plantio, além de habilitar avanços na arquitetura e na engenharia.

Um dos grandes marcos da geometria foi a criação de um tratado matemático chamado "Os elementos", escrito por volta de 300a.C. por Euclides em Alexandria. Essa obra se concretizou como uma das principais quando se trata do desenvolvimento científico graças a introdução de sua lógica argumentativa, trazendo ideias e demonstrações que as tornavam irrefutáveis, pavimentando o caminho para conhecimentos mais seguros e estruturados.

Com isso, tanto a matemática quanto a geometria tiveram a possibilidade de avançar mais rapidamente, já que novos estudos poderiam partir de provas anteriormente estabelecidas e assim, a matemática continua se desenvolvendo até o presente, passando por diversos séculos até a atualidade. Entretanto, a geometria euclidiana possui algumas limitações, já que tem como intuito descrever e estudar apenas objetos considerados ideais, os quais possuem margens bem definidas, facilitando seus estudos.

Graças à essa falta de abrangência, alguns objetos foram mantidos à parte de pesquisas por conta de suas formas irregulares e estruturas infinitamente detalhadas, como por exemplo o triângulo de Sierpiński, observado na Figura 1.3, que foi desenvolvido por Wacław Sierpiński em 1915.

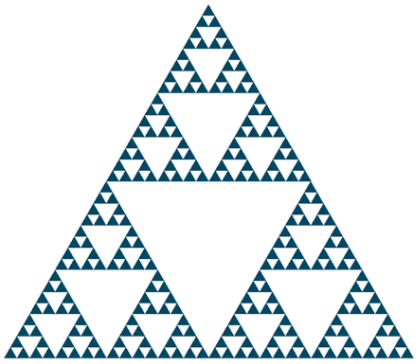


Figura 1.3: Triângulo de Sierpiński

Fonte: UFJF Fractalize

Podemos descrever esse triângulo a partir de um processo iterativo[2] infinito de duas maneiras distintas:

FORMA 1

Passo 1: Inicia-se o processo com um triângulo de qualquer medida (a construção original partia de um triângulo equilátero, mas pode ser generalizado) e segue para o Passo 2;

[2]Iteração: Ação de repetir um ou mais passos de um processo.

Passo 2: Em cada aresta determinam-se seus respectivos pontos médios e segue para o Passo 3;

Passo 3: Conecta-se os três pontos médios utilizando segmentos de reta, formando-se agora três triângulos congruentes (a menos do triângulo central, que será excluído), partindo para o Passo 4;

Passo 4: Retorne ao Passo 2 repetindo o processo para cada triângulo presente na figura.

FORMA 2

Passo 1: Inicia-se a construção a partir de um triângulo ABC de qualquer medida;

Passo 2: Reduz-se os lados do triângulo pela metade, obtendo um triângulo $A_1B_1C_1$;

Passo 3: Constrói-se duas cópias do triângulo $A_1B_1C_1$, sendo elas $A_2B_2C_2$ e $A_3B_3C_3$;

Passo 4: Translada-se os dois novos triângulos de maneira que o vértice A_2 coincida com B_1, o vértice A_3 coincida com C_1 e o vértice C_2 coincida com B_3;

Passo 5: Retorna ao passo 2 para cada um dos triângulos da figura.

Podemos observar que ao retornar do Passo 4 para o Passo 2 criamos um ciclo de construções, repetindo esses passos infinitamente. No geral, nomeamos como um processo iterativo, onde uma iteração é a conclusão de um ciclo, ou seja, podemos contar o triângulo inicial como a iteração de número 0, ao terminar o primeiro ciclo e retornar ao Passo 2 como iteração 1, ao terminar novamente como iteração 2 e assim sucessivamente, aumentando o número da iteração para cada vez em que o ciclo for concluído.

Benoit Mandelbrot, responsável pela criação e formalização dos fractais, foi um matemático polonês, nascido em Varsóvia no ano de 1924. Com 11 anos, emigrou, junto de sua família, para a França, onde de 1945 a 1947 estudou na École Polytechnique. Entre 1947 e 1949 estudou no Instituto de Tecnologia da Califórnia, nos Estados Unidos, onde aos 35 anos, iniciou sua carreira na IBM (International Business Machines Corporation).

Em sua autobiografia, Mandelbrot [2], afirma que após obter acesso às máquinas da empresa, Mandelbrot, destacou-se sendo um dos pioneiros na criação e exibição de imagens geométricas fractais o que permitiu suas descobertas em 1979.

Derivado do latim *fractus*, que significa fragmento, fractais são estruturas que possuem características até então caóticas que os faziam ser ignorados. Porém foi notado que essas figuras poderiam ser descritas em sua maioria por processos recursivos, como no caso do triângulo de Sierpiński apresentado anteriormente, permitindo seus estudos a partir disso.

Em suma, fractal não é um conceito perfeitamente definido para a matemática, em comparação, Falconer [3] relaciona o conceito de fractal para a matemática com o conceito de vida para a biologia, como uma ideia que não é possível definir concretamente. Para ele, fractais são objetos que apresentam algumas das seguintes características.

(i) Possuem uma estrutura fina, a qual é detalhada em escalas arbitrariamente pequenas.

(ii) É irregular demais para ser descrita por métodos geométricos tradicionais, sejam eles locais ou globais.

(iii) Na maioria dos casos, possuem alguma forma de autossimilaridade, seja aproximada ou estatística.

(iv) Na maior parte dos casos, sua dimensão fractal é maior do que sua dimensão topológica.

(v) Na maioria dos casos, é definido de forma ligeiramente simples, permitindo principalmente formas recursivas.

Assim como a Geometria Euclidiana possui aplicações em diversas áreas do conhecimento, como artes, arquitetura, engenharia, a geometria fractal possui grande campo para pesquisas em diversas outras ciências, com o potencial de estruturar novos conhecimentos em diversas áreas, como nas ciências da natureza, onde fractais podem ser encontrados na botânica, na química e até mesmo na medicina.

Quando observamos alguns exemplos de vegetações somos capazes de perceber certas características consideradas por muitos irregulares e complexas, entretanto, se estudarmos essa vegetação a partir de seu processo de desenvolvimento, notamos que algumas de suas formas se repetem ao longo do seu crescimento, apresentando um conceito de autossemelhança. Por exemplo, para algumas espécies de árvores, notamos que seus galhos se espalham de forma padronizada a cada divisão, mostrando que o processo de desenvolvimento dessa planta segue uma lógica oculta, mas se observada ao todo, inicialmente poderia ser considerada desorganizada.

Podemos a partir disso modelar o desenvolvimento de uma planta através de processos iterativos, como por exemplo na samambaia de Barnsley (Figura 1.4), onde cada uma de suas ramificações novas se assemelha à anterior.

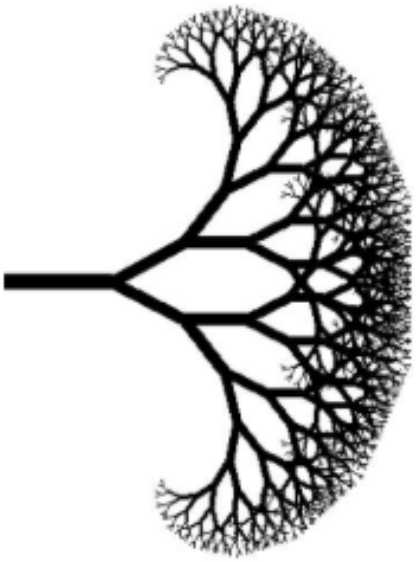
Figura 1.4: Samambaia de Barnsley

Fonte: Geometria Fractal no Ensino Médio

Processos de ramificações como esses podem ser notados em várias áreas do conhecimento, mais especificamente na medicina. No que se refere à anatomia humana, podemos ver em nossos vasos sanguíneos um processo de divisões sucessivas para que possam abranger uma maior área do nosso corpo com máxima eficiência. À primeira vista esse processo poderia ser pensado como algo aleatório e complexo, que segue alguma ordem de adaptação a cada etapa de seu desenvolvimento, porém, se observarmos mais a fundo, podemos notar alguns padrões no processo de bifurcação dos vasos. Por exemplo, na Figura 1.5, conseguimos ver claramente as divisões do vaso sanguíneo de um rim se repetindo de forma semelhante enquanto se afinam.

Por fim, o principal objetivo deste trabalho é o estudo de construções fractais por meio de palavras geradas a partir de um alfabeto, baseando-se nos métodos de sistemas-L. Com esse foco, desenvolveremos métodos de construções para alguns fractais, como a curva do dragão, a curva e o floco de neve de Koch, em sua forma original e com algumas variações. Além disso, estudaremos propriedades algébricas destas palavras e propriedades

(a) Vasos sanguíneos de um rim computadorizado.
Fonte: Journal of the American Society of Nephrology.



(b) Ramificação fractal.
Fonte: Scientia Plena Vol. 13, Num. 04.

Figura 1.5: Comparação entre vasos sanguíneos e ramificação fractal

geométricas dos fractais, utilizando linguagem Python para gerar e visualizar as figuras. Também foram gerados vídeos de animações do processo de construção das curvas do dragão, de Koch original, l-poligonal e alternada, disponíveis, na plataforma GitHub©, junto de seus códigos, através do link https://github.com/tmelorc/L-system. Além disso, o trabalho permite a relação entre geometria e sistemas numéricos, apresentando resultados de outros autores bem como contribuições originais para o estudo.

2 Sistemas-L: definições e exemplos

Por ser um método matemático axiomático, os sistemas-L, inicialmente pensados para a área da botânica, também podem ser aproveitados para outras diversas áreas. Por possuírem métodos recursivos assim como muitos objetos matemáticos, como sequências e fractais, os sistemas-L apresentam uma boa eficiência ao representá-los.

Para podermos avançar com a construção conjunta de fractais utilizando os sistemas-L, vamos precisar de operações e definições para podermos trabalhar com suas palavras, sendo assim, seguem algumas das operações, junto de seus exemplos, que precisaremos ao longo desta dissertação. O leitor poderá encontrar mais detalhes desse assunto em [4].

Definição 2.1. Dadas f e g duas palavras, definimos uma operação binária $*$ por meio da concatenação destas palavras, na ordem em que aparecem na operação.

Dadas duas palavras $f = \texttt{abc}$ e $g = \texttt{def}$, a operação $f * g$ se dá na forma

$$f * g = \texttt{abc} * \texttt{def} = \texttt{abcdef}.$$

Definição 2.2. Seja f uma palavra de um sistema-L e ϕ o morfismo[1] que define o conjunto de regras do sistema. Definimos $\phi(f)$ a partir da aplicação de ϕ em cada um dos símbolos de f.

Seja ϕ o morfismo de um sistema-L, onde $\phi(\texttt{a}) = \texttt{b}$, $\phi(\texttt{b}) = \texttt{c}$ e $\phi(\texttt{c}) = \texttt{a}$ e seja $f = \texttt{abc}$ uma palavra. Daí

$$\phi(f) = \phi(\texttt{a}) * \phi(\texttt{b}) * \phi(\texttt{c}) = \texttt{b} * \texttt{c} * \texttt{a} = \texttt{bca}.$$

Definição 2.3. Sejam A um alfabeto, f_0 o axioma, ϕ o morfismo que define o conjunto de regras do sistema e n um natural. Definimos a palavra f_n como a palavra de n-ésima iteração do sistema, do seguinte modo:

$$f_n = \begin{cases} f_0, & \text{para } n = 0; \\ \phi(f_{n-1}), & \text{para } n > 0. \end{cases} \tag{2.1}$$

Seja $\phi(\texttt{a}) = \texttt{b}$, $\phi(\texttt{b}) = \texttt{c}$ e $\phi(\texttt{c}) = \texttt{a}$, e o axioma $f_0 = \texttt{abc}$, a palavra f_3 será:

$$f_3 = \phi(f_2) = \phi[\phi(f_1)] = \phi\{\phi[\phi(f_0)]\} = \phi\{\phi[\phi(\texttt{abc})]\} = \phi\{\phi[\texttt{bca}]\} = \phi\{\texttt{cab}\} = \texttt{abc}.$$

Definição 2.4. Sejam f uma palavra e $l \geq 1$ um inteiro. Definimos, recursivamente, $l \cdot f$ por

$$l \cdot f = \begin{cases} f, & \text{se } l = 1, \\ ((l-1) \cdot f) * f, & \text{se } l > 1. \end{cases}$$

[1] Processo de transformação de um conjunto sobre outro.

Seja $f = \texttt{abc}$ uma palavra e $l = 3$ um inteiro a operação $l \cdot f$ será:

$$3 \cdot f = (2 \cdot f) * f = (f * f) * f = f * f * f = \texttt{abcabcabc}.$$

Definição 2.5. Seja f uma palavra. Definimos o comprimento de f como a quantidade de símbolos presentes na palavra, denotado na forma $|f|$.

Seja $f = \texttt{abcdefghij}$ uma palavra, então $|f| = 10$, pois possui 10 termos.

Definição 2.6. Seja f uma palavra. A palavra escrita de forma espelhada de f é definida como f reversa, denotada por f^{-1}.

Seja $f = \texttt{REVERSA}$ uma palavra, então $f^{-1} = \texttt{ASREVER}$.

Exemplo clássico: Fibonacci

Uma aplicação para os sistemas-L é na criação de uma palavra para descrever a sequência de Fibonacci. A sequência de Fibonacci é construída a partir de uma recorrência onde o próximo número é igual à soma dos dois anteriores, iniciando com o número 1 duas vezes seguidas. Podemos observar a construção dessa sequência a seguir.

$$\begin{aligned}
a_1 &= 1;\\
a_2 &= 1;\\
a_3 &= a_2 + a_1 = 2;\\
a_4 &= a_3 + a_2 = 3;\\
a_5 &= a_4 + a_3 = 5;\\
a_6 &= a_5 + a_4 = 8;\\
&\vdots\\
a_n &= a_{n-1} + a_{n-2}, \text{ para } n > 2.
\end{aligned}$$

Observando essa sequência podemos criar uma palavra de um sistema-L, semelhante à presente em [5], em que o comprimento da sua palavra de iteração n será igual ao n-ésimo termo da sequência de Fibonacci. Observe a seguir sua construção:

$$\begin{aligned}
\text{Alfabeto } (A): &\quad \{\texttt{0}, \texttt{1}\}\\
\text{Axioma } (f_0): &\quad \texttt{1}\\
\text{Regra } (\phi): &\quad \texttt{0} \to \texttt{01}\\
&\quad \texttt{1} \to \texttt{0}
\end{aligned}$$

Portanto, seguindo esse sistema, obteremos a seguinte sequência de palavras:

$$\begin{aligned}
f_0 &= \texttt{1}\\
f_1 &= \texttt{0}\\
f_2 &= \texttt{01}\\
f_3 &= \texttt{010}\\
f_4 &= \texttt{01001}\\
f_5 &= \texttt{01001010}\\
&\vdots
\end{aligned}$$

Chamaremos de palavra de Fibonacci de n-ésima iteração a palavra f_n gerada nesse sistema. Podemos realizar uma comparação entre os comprimentos das palavras e a sequência de Fibonacci observando a tabela 2.1 a seguir:

n	Palavra f_n	$\lvert f_n \rvert$	Fibonacci
0	1	1	1
1	0	1	1
2	01	2	2
3	010	3	3
4	01001	5	5
5	01001010	8	8
6	0100101001001	13	13
7	010010100100101001010	21	21
8	0100101001001010010100100101001001	34	34

Tabela 2.1: Comparação $|f_n|$ e os números da sequência de Fibonacci

Utilizando-nos desse sistema, atribuiremos novas funções para os símbolos de seu alfabeto para que possamos construir uma curva poligonal[2] no plano, conhecida como curva de Fibonacci.

Para a sua construção, iremos precisar de alguns simples itens além da palavra de Fibonacci e as novas atribuições de seus símbolos, que são:

- Ponto inicial (p_0): ponto inicial da curva, onde se iniciará toda sua construção;
- Direção inicial ($\vec{v_0}$): vetor que indica direção e sentido em que a curva iniciará sua construção.

Com isso, a partir de um ponto inicial e da direção inicial, realizaremos a construção da curva com as novas atribuições de cada símbolo, sendo elas:

`1`: Construir um segmento seguindo a direção anterior;

`0`: Construir um segmento seguindo a direção anterior e em seguida alterar a direção em:

$+90°$ se `0` estiver em posição par;

$-90°$ se `0` estiver em posição ímpar.

Com essas regras e atribuições, criamos então um sistema-L que construirá visualmente a curva de Fibonacci. Além disso, quando observada mais atentamente, a curva possui uma propriedade marcante, onde algumas partes da curva possuem características semelhantes entre si.

Conhecemos essa característica como autossemelhança, principal padrão presente em um fractal, de extrema importância para seu estudo. Portanto, a partir da curva de Fibonacci, podemos generalizar esse conceito para outras curvas e por isso iremos construir e explorar novas curvas desenvolvidas através de sistemas-L.

[2]Para facilitar a escrita, escreveremos simplesmente curva.

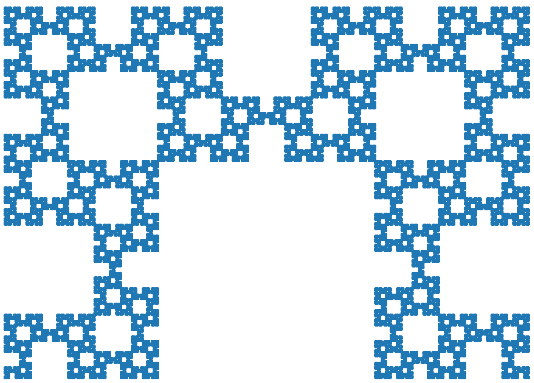


Figura 2.1: Curva de Fibonacci de iteração 29

Fonte: Elaborado pelo autor (2024)

Exemplo moderno: curva do dragão

Na década de 1960, os pesquisadores da NASA, John Heighway, Bruce Banks e William Harter desenvolveram e estudaram uma curva autossemelhante que se aproximava do formato de um dragão. Esta curva, assim como a maioria dos fractais, é construída através de um processo recursivo. Neste caso, é formada a partir de reconstruções a partir de um segmento de reta, seguindo o seguinte processo:

- Toma-se o primeiro segmento de reta ($\overline{AB}$);

Figura 2.2: Curva do Dragão com 0 iterações

Fonte: Elaborado pelo autor (2024)

- Tendo como hipotenusa o segmento $\overline{AB}$ (ponto inicial A), cria-se um triângulo retângulo isósceles, cujo vértice do ângulo reto é posicionado em um dos semiplanos formados por $\overline{AB}$, estando a uma angulação de:

  $+45°$, se o segmento anterior teve um posicionamento de $-45°$;

  $-45°$, se o segmento anterior teve um posicionamento $+45°$ ou for o primeiro segmento da figura[3].

[3] A contagem se iniciará pelo segmento que tiver como extremidade o ponto inicial, seguido por aquele que possuir interseção com esse, e assim por diante.

- Remove-se o segmento original, utilizado como hipotenusa.
- Repete-se o processo para todos novos segmentos criados.

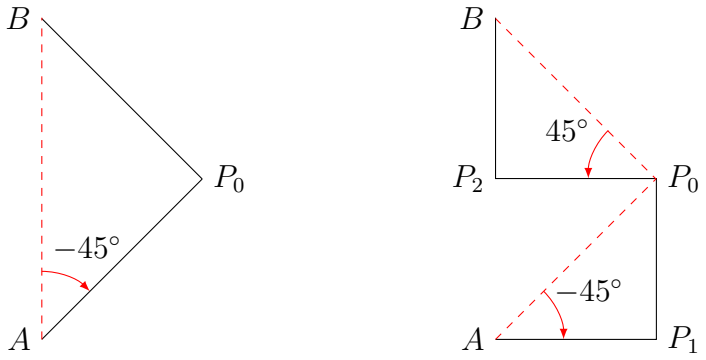


Figura 2.3: Curva do Dragão com 1 e 2 iterações

Fonte: Elaborado pelo autor (2024)

Observando sua construção, podemos notar um padrão interessante, que pode facilitar um método de construção gerado por um sistema-L: ignorando inicialmente a escala e a rotação, podemos notar que a figura de iteração $n+1$ é formada pela figura de iteração n duplicada, tendo a seguinte forma:

- A primeira figura n posicionada na posição anterior;
- A segunda figura n rotacionada em 90° em sentido horário e unida à primeira em seus pontos finais.

Ou seja, poderemos construir a curva $n+1$ utilizando a curva n, rotacionando 90° e em seguida utilizando a curva n novamente, mas de trás para frente.

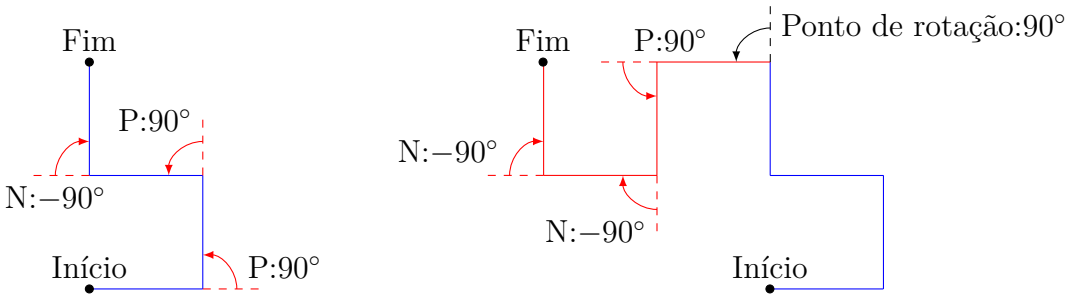


Figura 2.4: Iteração da curva do dragão de 2 para 3

Fonte: Elaborado pelo autor (2024)

Ao observar isso, podemos construir um sistema-L capaz de descrever as iterações da curva do dragão, seguido de um método que poderá construí-la.

$$
\begin{aligned}
\text{Alfabeto } (A): &\quad \{\mathtt{S}, \mathtt{E}, \mathtt{D}\} \\
\text{Axioma } (f_0): &\quad \mathtt{S} \\
\text{Regra } (\gamma): &\quad f_n \to f_n * \mathtt{E} * \phi(f_n^{-1}) \\
\text{onde } \phi: &\quad \mathtt{D} \to \mathtt{E} \\
&\quad \mathtt{E} \to \mathtt{D}
\end{aligned}
$$

Método de construção. Sendo $A = \{\mathtt{S}, \mathtt{E}, \mathtt{D}\}$ um alfabeto, $\vec{v}$ um vetor e p_0 um ponto inicial, os termos de A possuem as seguintes atribuições.

$\mathtt{S} \rightarrow$ Cria o ponto $p_i = p_{i-1} + \vec{v}$, traçando o segmento de p_{i-1} a p_i (Segmento).

$\mathtt{E} \rightarrow$ Rotaciona o vetor $\vec{v}$ em 90° no sentido anti-horário. (Esquerda)

$\mathtt{D} \rightarrow$ Rotaciona o vetor $\vec{v}$ em 90° no sentido horário (Direita).

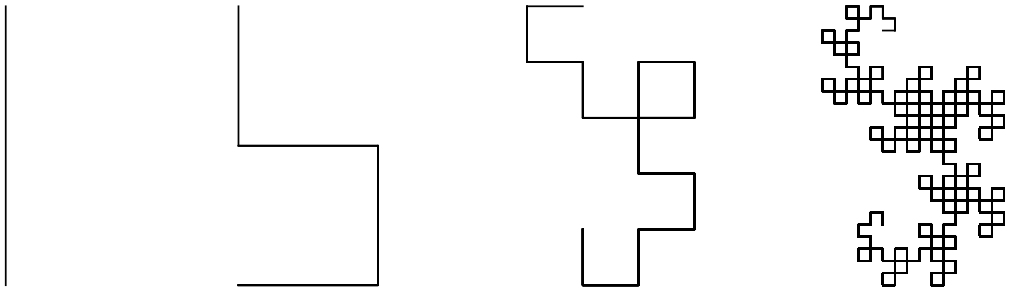


Figura 2.5: Curva do Dragão com 0, 2, 4 e 8 iterações

Fonte: Elaborado pelo autor (2024)

Um vídeo dessa construção, criado pelos autores em Python (ver Programa 4.1), pode ser visto no Youtube em Dragon Curve.

3 Curvas fractais clássicas e suas variações

3.1 Curva de Koch

No ano de 1904, o matemático sueco Niels Fabian Helge von Koch (1870–1924) publicou um trabalho contendo estudos sobre teoria de curvas no plano, dentre elas estava uma curva conhecida atualmente como curva de Koch que, de acordo com [6], graças a sua construção simples e forma aparentemente irregular, se tornou alvo de grande curiosidade, possibilitando diversos estudos dentro e fora da matemática.

Essa curva recebeu maior destaque a partir da definição do termo fractal, por Benoît Mandelbrot, na década de 60, que permitiu uma exploração mais profunda de seus conceitos.

A curva de Koch é construída através de um processo iterativo utilizando apenas um segmento de reta como ponto de partida, obtendo sua forma final a partir do seguinte processo:

1. Divide-se o segmento inicial em três de mesmo tamanho;
2. Sobre o segmento central cria-se um triângulo equilátero;
3. Retira-se o segmento central do triângulo anterior;
4. Retorna-se ao primeiro passo, para cada novo segmento.

Todo esse processo é repetido de forma iterativa, podendo ser realizado infinitas vezes. Podemos observar na Figura 3.1 este processo acontecendo até a iteração de número 3.

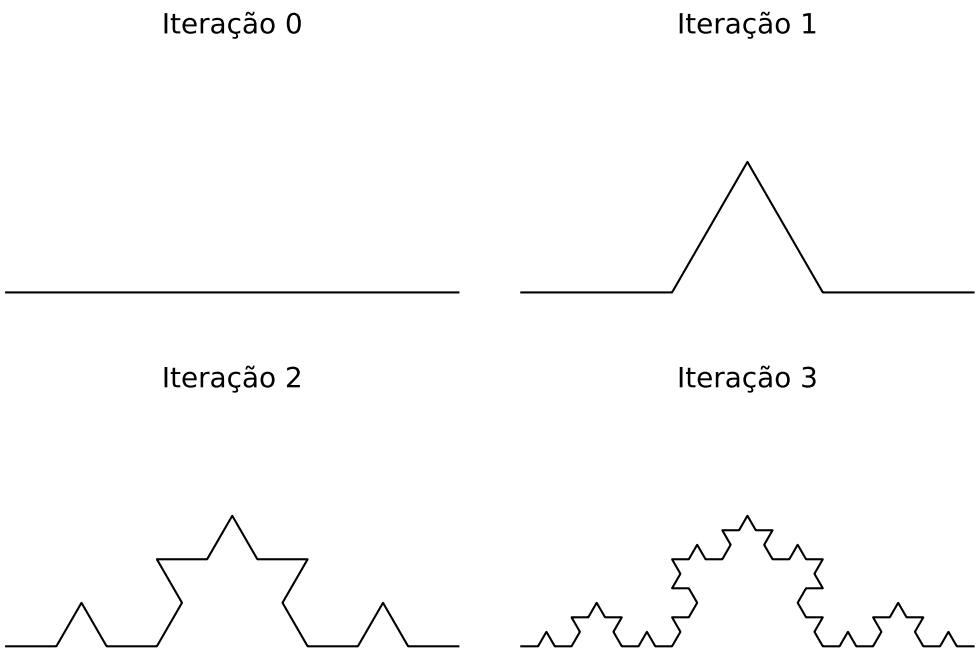


Figura 3.1: Curvas de Koch com diferentes iterações

Fonte: Elaborado pelo autor (2024)

Observando mais atentamente a Figura 3.2, notamos algumas características e elementos específicos, como a recorrência de ângulos e segmentos com a mesma medida. Vamos descrevê-los:

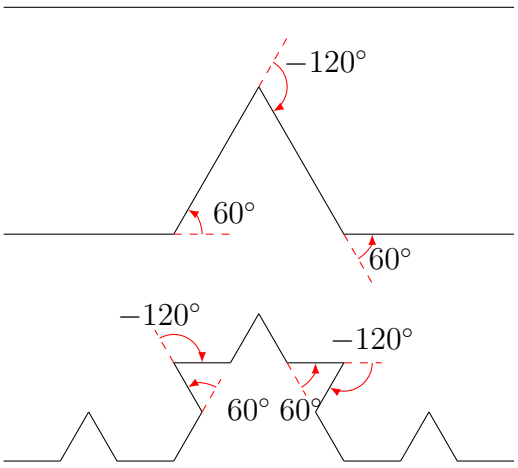


Figura 3.2: Diferentes iterações com ângulos destacados

Fonte: Elaborado pelo autor (2024)

Segmentos. Todos os segmentos da curva, de mesma iteração, possuem o mesmo comprimento.

Ângulos. Possuem as medidas de $60°$ e $-120°$. Nenhum angulo já existente sofre alteração pelas iterações feitas após sua primeira aparição.

A partir dessas características, podem-se dividir todos os elementos dessa curva em três: segmentos de mesmo comprimento, ângulos de 60° e ângulos de −120°. Numeram-se esses elementos como 0, 1 e 2, respectivamente.

Observando agora a Figura 3.3, é possível notar que a curva de Koch sem iterações é apenas um segmento, sendo correspondente a palavra $f_0 = 0$, enquanto a curva de primeira iteração corresponde a $f_1 = 0102010$, onde os símbolos 1 e 2 fornecem os ângulos de rotação entre dois segmentos adjacentes.

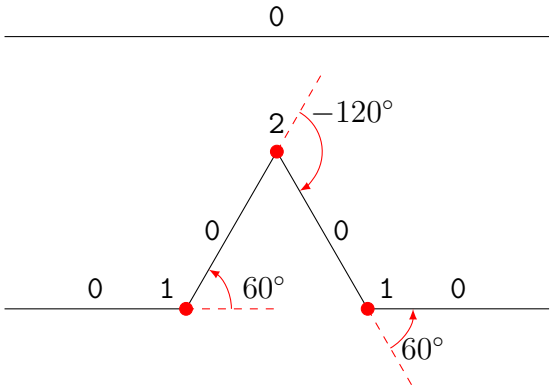


Figura 3.3: Curva de Koch de 1 iteração com ângulos destacados

Fonte: Elaborado pelo autor (2024)

Sabe-se que o processo de construção da curva de Koch segue uma lógica recursiva, realizando um novo processo para cada segmento presente. Com base nisso, constrói-se o morfismo ϕ que representará a iteração aplicada em um segmento, através das representações apresentadas anteriormente.

Sendo $A = \{0, 1, 2\}$, o morfismo $\phi\colon A^* \to A^*$ se dá por:

$$\phi(0) = 0102010, \quad \phi(1) = 1, \quad \phi(2) = 2. \tag{3.1}$$

Utilizando esse morfismo e inspirando-se nas construções de [7], é possível desenvolver um sistema-L capaz de formar uma palavra que generaliza a curva de Koch.

Palavra de Koch. Seja $A = \{0, 1, 2\}$ um alfabeto, o sistema-L que gera a palavra de Koch é da forma:

$$\begin{aligned} \text{Alfabeto } (A)&: \ \{0, 1, 2\} \\ \text{Axioma } (f_0)&: \ 0 \\ \text{Regra } (\phi)&: \ 0 \to 0102010 \\ &\quad\ \ 1 \to 1 \\ &\quad\ \ 2 \to 2 \end{aligned}$$

Utilizando esse sistema, considera-se f_n como a palavra de Koch de n-ésima iteração, de acordo com a Definição 2.3. A partir dessa palavra, definem-se regras para cada termo do alfabeto A, que gerarão a curva de Koch utilizando f_n.

Método de construção[1] Sendo $A = \{0, 1, 2\}$ um alfabeto, $\vec{v}$ um vetor e p_0 um ponto inicial, os termos de A possuem as seguintes atribuições.

[1] Vídeo: https://www.youtube.com/watch?v=JS4zxG4GvxE

$0 \rightarrow$ Cria o ponto $p_i = p_{i-1} + \vec{v}$, traçando o segmento de p_{i-1} a p_i.

$1 \rightarrow$ Rotaciona o vetor $\vec{v}$ em $+60°$.

$2 \rightarrow$ Rotaciona o vetor $\vec{v}$ em $-120°$.

$$f_1 = 0102010$$

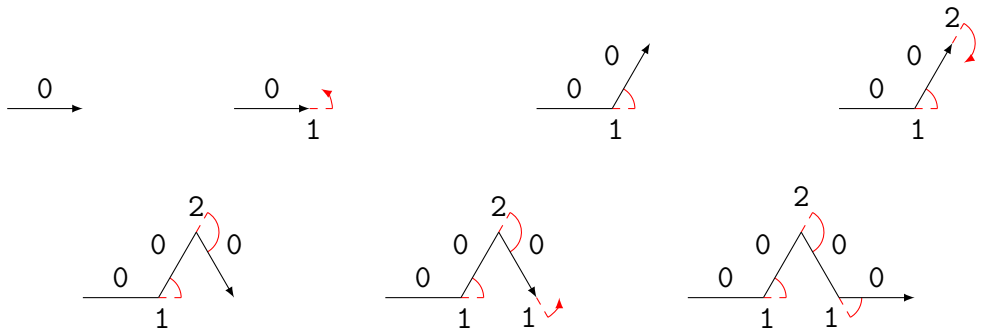


Figura 3.4: Construção da curva de Koch através da palavra f_1

Fonte: Elaborado pelo autor (2024)

Realizando algumas iterações, pode-se notar alguns padrões no morfismo ϕ, que iremos visualizar na tabela abaixo e formalizar.

f_0	0
$\phi(f_0) = f_1$	0 1 0 2 0 1 0
$\phi(f_1) = f_2$	0102010 1 0102010 2 0102010 1 0102010

Tabela 3.1: Palavras de Koch

Proposição 3.1. *Dada f_n a palavra de Koch de n-ésima iteração, é possível construir a palavra f_{n+1} seguindo a forma:*

$$f_{n+1} = f_n * 1 * f_n * 2 * f_n * 1 * f_n, \quad n \geq 0. \tag{3.2}$$

Demonstração. Para realizar esta demonstração, será utilizado o método de indução finita. Para isso, provemos que para $n = 0$ a equação (3.2) é verdadeira.

De fato, como $f_0 = 0$ tem-se:

$$f_1 = \phi(f_0) = \phi(0) = 0102010 = 0 * 1 * 0 * 2 * 0 * 1 * 0 = f_0 * 1 * f_0 * 2 * f_0 * 1 * f_0.$$

Suponhamos então que para algum $n \in \mathbb{N}$ a equação (3.2) seja verdadeira. A partir disso então, deve-se provar que para $n + 1$ também vale (3.2).

Pela definição da palavra de Koch temos que:

$$f_{n+2} = \phi(f_{n+1}).$$

Utilizando a hipótese de indução $f_{n+1} = f_n * 1 * f_n * 2 * f_n * 1 * f_n$ e substituindo na expressão, podemos concluir:

$$f_{n+2} = \phi(f_n * 1 * f_n * 2 * f_n * 1 * f_n).$$

Por ϕ ser um morfismo (ver (3.1)), tem-se então que:

$$f_{n+2} = \phi(f_n) * \phi(1) * \phi(f_n) * \phi(2) * \phi(f_n) * \phi(1) * \phi(f_n),$$

que pela Definição 2.3 e morfismo (3.1) temos portanto que:

$$f_{n+2} = f_{n+1} * 1 * f_{n+1} * 2 * f_{n+1} * 1 * f_{n+1}.$$ □

3.1.1 Comprimento da palavra de Koch

Para realizar o cálculo do comprimento da palavra de Koch, em cada iteração (para cada n), deve-se separar seus dígitos em duas classes, Zeros e Rastros, onde o comprimento total da palavra é igual a soma da quantidade de termos em cada classe.

Zeros: $z_n = z(f_n) \rightarrow$ número de dígitos 0 em f_n

Rastros: $r_n = r(f_n) \rightarrow$ número de dígitos 1 e 2 em f_n

Pode-se observar que no caso de uma concatenação de duas palavras f e g, os zeros dessa composição seria igual à soma dos zeros de cada palavra, assim como os rastros dessa composição seria igual à soma dos rastros de cada palavra, ou seja:

$$z(f * g) = z(f) + z(g), \quad r(f * g) = r(f) + r(g)$$

Teorema 3.2. *A quantidade de zeros na palavra f_n é $z_n = 4^n$, para $n \geq 0$.*

Demonstração. Para realizar a prova deste teorema utilizaremos o princípio da indução finita. Pode-se verificar que a afirmação é verdadeira para $n = 0$, pois para $f_0 = 0$, temos

$$z_n = z_0 = 1 = 4^0 = 4^n$$

portanto, para $n = 0$ temos $z_n = 4^n$.

Agora, supondo que para um n arbitrário a afirmação $z_n = 4^n$ seja verdadeira, mostremos que para $n + 1$ também é válida.

$$\begin{aligned} z_{n+1} &= z(f_{n+1}) = z(f_n * 1 * f_n * 2 * f_n * 1 * f_n) \\ &= z(f_n) + z(1) + z(f_n) + z(2) + z(f_n) + z(1) + z(f_n) \\ &= 4^n + 0 + 4^n + 0 + 4^n + 0 + 4^n = 4 \times 4^n = 4^{n+1} \end{aligned}$$

Portanto, pelo que foi apresentado acima e pela hipótese de indução finita, temos que $z_n = 4^n$. □

Teorema 3.3. *A quantidade de rastros na palavra f_n é $r_n = 4^n - 1$, para $n \geq 0$.*

A demonstração se dá de forma similar à do teorema anterior.

Como consequência imediata dos Teoremas 3.2 e 3.3, temos o seguinte resultado.

Corolário 3.4. *O comprimento da palavra de Koch f_n é $|f_n| = 2 \cdot 4^n - 1$, para $n \geq 0$.*

A partir do Corolário 3.4 podemos constatar que o comprimento da palavra f_n é sempre ímpar, além disso, podemos considerar a possibilidade de a palavra ser um palíndromo[2], já que por não possuir termos iguais em sequência seria impossível ser um palíndromo caso possuísse comprimento par.

[2]Palavra que pode ser lida da esquerda para direita ou direita para esquerda mantendo sua ordem. Exemplos: osso, ama, ovo.

Teorema 3.5. *Sejam $f_0 = \mathtt{0}$ e $f_n = \phi(f_{n-1})$, para $n > 0$, então toda palavra f_k com $k \geq 0$ é um palíndromo.*

Demonstração. Para demonstração deste teorema utilizaremos o princípio de indução finita. Para isso, podemos observar que para $k = 0$ temos $f_0 = \mathtt{0}$ que é obviamente um palíndromo. Suponhamos que para um k arbitrário f_k também seja palíndromo, portanto, vamos observar a palavra f_{k+1}. Sabemos pela proposição 3.1 que a construção da palavra f_{k+1} ocorre na forma

$$f_{k+1} = f_k * \mathtt{1} * f_k * \mathtt{2} * f_k * \mathtt{1} * f_k.$$

Como f_k é palíndromo, podemos observar que a palavra $f_k * \mathtt{1} * f_k * \mathtt{2} * f_k * \mathtt{1} * f_k = f_{k+1}$ também o é. Portanto, pela hipótese da indução finita, todas as palavras f_k deste formato são palíndromos. □

3.2 Curva l-poligonal de Koch

Com as características da curva de Koch bem definidas e sua construção descrita através de um sistema-L, é possível extrapolar essa construção para investigar curvas semelhantes, que ao invés de utilizar o triângulo, como na Figura 3.1, utiliza outros polígonos como base para a iteração, construindo-a agora da seguinte forma:

1. Divide-se o segmento em três segmentos de mesmo tamanho;
2. Sobre o segmento central cria-se um polígono regular de l lados;
3. Retira-se o segmento utilizado de base para o polígono;
4. Retorna ao primeiro passo, aplicando em cada segmento da figura.

Denota-se esta curva de curva l-poligonal de Koch. Observa-se um de seus exemplares na Figura 3.5, sendo este a curva com base em um hexágono.

Pode-se notar que assim como a curva original de Koch, os segmentos e ângulos dessa nova curva podem ser destacados e agrupados de maneira semelhante à feita anteriormente. Entretanto, por não mais ser utilizado o triângulo equilátero como base, os ângulos presentes possuem valores diferentes para cada polígono que for utilizado como base, podendo ser descobertos baseando-se nos ângulos internos e externos do polígono.

Definição 3.6. Seja l a quantidade de lados de um polígono regular, definimos I_l e E_l como os valores dos ângulos internos e externos deste polígono, respectivamente.

Inspirados pela construção da curva original, da sessão anterior, é possível encontrar um novo sistema-L que forneça as curvas l-poligonais de Koch. A natureza desse sistema deve fornecer a autossemelhança na curva. Além disso, o método de construção será semelhante ao original, com alteração apenas nos valores dos ângulos de rotação.

Método de construção[3] Seja $A = \{\mathtt{0}, \mathtt{1}, \mathtt{2}\}$ um alfabeto e $\vec{v}$ um vetor, os termos de A tem as seguintes funções:

$\mathtt{0} \rightarrow$ Cria o ponto $p_i = p_{i-1} + \vec{v}$, traçando o segmento de p_{i-1} a p_i;

[3] Vídeo: https://www.youtube.com/watch?v=cwldDbZsCeQ

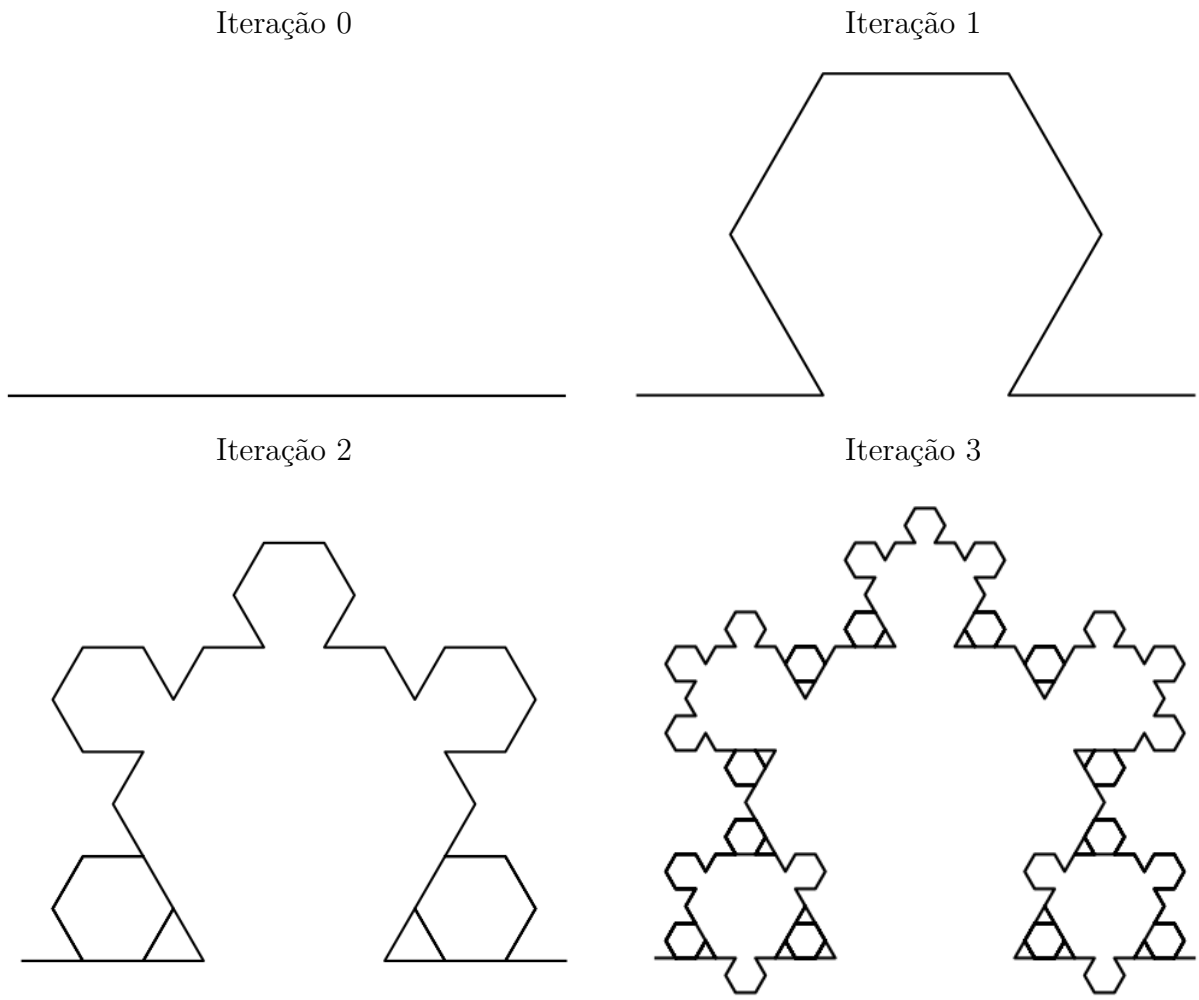


Figura 3.5: Curvas 6-poligonais de Koch com diferentes iterações

Fonte: Elaborado pelo autor (2024)

1 $\rightarrow$ Rotaciona $\vec{v}$ em um ângulo de valor I_l no sentido anti-horário;

2 $\rightarrow$ Rotaciona $\vec{v}$ em um ângulo de valor E_l no sentido horário.

De maneira semelhante à proposta realizada na curva original, pode-se, com as mesmas condições iniciais, definir palavras para construir a curva a partir do ponto e vetor inicial. Entretanto, a construção não é tão simples, pois varia de acordo com o polígono inicial escolhido.

Tem-se a seguir as palavras f_0 e f_1 das curvas baseadas em três polígonos: triângulo, quadrado e pentágono.

f_n	Triângulo	Quadrado	Pentágono
f_0	0	0	0
f_1	$010 * 20 * 10$	$010 * 2020 * 10$	$010 * 202020 * 10$

Pode-se notar que a única alteração entre as palavras f_1 de cada uma das três curvas é a quantidade de subpalavras 20 que aparece. Posto isso, cria-se um morfismo para gerar a palavra f_n que descreve a curva de iteração n.

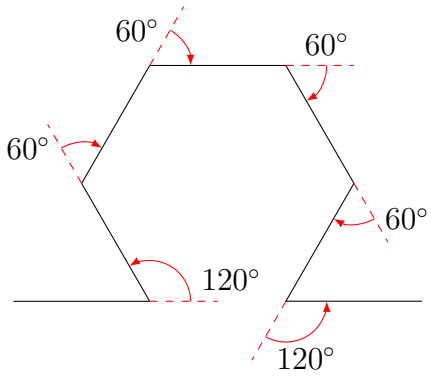


Figura 3.6: Curva 6-poligonal de Koch com ângulos destacados

Fonte: Elaborado pelo autor (2024)

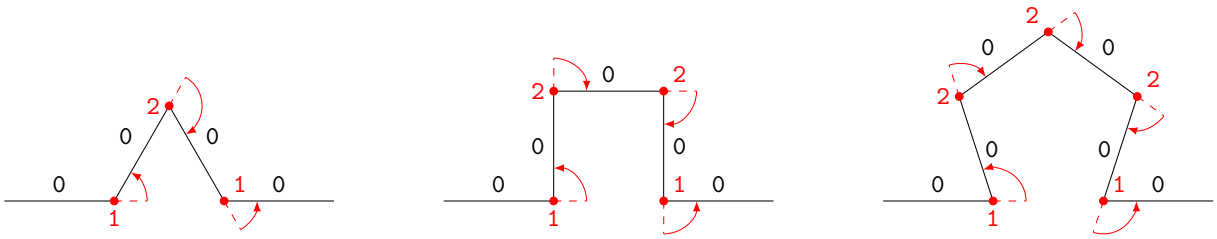


Figura 3.7: Curvas 3, 4 e 5-poligonais de iteração 1

Fonte: Elaborado pelo autor (2024)

Definição 3.7. Dado $l \geq 3$ inteiro, define-se o morfismo $\phi\colon A^* \to A^*$ por

$$\phi(0) = 010 * (l-2) \cdot 20 * 10, \quad \phi(1) = 1, \quad \phi(2) = 2.$$

Palavra l-poligonal de Koch Seja $A = \{0, 1, 2\}$ um alfabeto, o sistema-L que gera a palavra de Koch é da forma:

$$\begin{aligned} \text{Alfabeto } (A)\text{:} &\quad \{0, 1, 2\} \\ \text{Axioma } (f_0)\text{:} &\quad 0 \\ \text{Regra } (\phi)\text{:} &\quad 0 \to 010 * (l-2) \cdot 20 * 10 \\ &\quad 1 \to 1 \\ &\quad 2 \to 2 \end{aligned}$$

Proposição 3.8. *Seja f_n a palavra de Koch l-poligonal de n-ésima iteração, podemos construir a palavra f_{n+1} seguindo a forma:*

$$f_{n+1} = f_n * 1 * f_n * [(l-2) \cdot (2 * f_n)] * 1 * f_n \tag{3.3}$$

Demonstração. Tomemos inicialmente $n = 0$ e provemos que para este valor a aplicação é verdadeira:

Sabemos que

$$f_0 = 0$$

e aplicando o morfismo ϕ em f_0 temos

$$\phi(f_0) = f_1 = 010 * [(l-2) \cdot (2 * 0)] * 10 = f_0 * 1 * f_0 * [(l-2) \cdot (2 * f_0)] * 1 * f_0$$

logo

$$f_1 = f_0 * 1 * f_0 * [(l-2)\cdot(2 * f_0)] * 1 * f_0$$

Agora, assumindo que para um n qualquer a afirmação 3.3 é verdadeira, provemos que para um $n+1$ também será válida:

$$f_{n+2} = f_{n+1} * 1 * f_{n+1} * [(l-2)\cdot(2 * f_{n+1})] * 1 * f_{n+1}$$

como ϕ é um morfismo

$$\begin{aligned} f_{n+2} &= \phi(f_{n+1}) \\ &= \phi(f_n * 1 * f_n * [(l-2)\cdot(2 * f_n)] * 1 * f_n) \\ &= \phi(f_n) * \phi(1) * \phi(f_n) * \phi([(l-2)\cdot(2 * f_n)]) * \phi(1) * \phi(f_n) \\ &= f_{n+1} * 1 * f_{n+1} * [(l-2)\cdot(2 * f_{n+1})] * 1 * f_{n+1}. \end{aligned}$$

onde na segunda igualdade usa-se a hipótese de indução. □

Podemos notar que a palavra que gera a curva de Koch é um dos possíveis produtos da nova palavra que constrói a curva poligonal, basta apenas tomar o valor $l = 3$ e assim teremos a construção da palavra original pelo processo de iterações do novo morfismo.

3.3 Floco de neve de Koch e suas variações

A partir das construções e definições das curvas anteriores, podemos adaptar alguns fatores para gerar figuras, sejam novas ou conhecidas, como por exemplo o floco de neve de Koch, que é construído a partir de um triângulo equilátero, onde seus segmentos terão as mesmas iterações realizadas na curva de Koch.

Com isso em vista, podemos construir o floco de neve de Koch baseando-se nas construções da curva de Koch original. De forma simplificada, alteraremos o axioma do sistema-L para que a iteração 0, seu axioma, seja agora o triângulo equilátero, ao invés de apenas um segmento. Portanto, o novo sistema-L será dado na seguinte forma:

$$\begin{aligned} \text{Alfabeto } (A): &\quad \{0, 1, 2\} \\ \text{Axioma } (f_0): &\quad 02020 \\ \text{Regra } (\phi): &\quad 0 \to 0102010 \\ &\quad 1 \to 1 \\ &\quad 2 \to 2 \end{aligned}$$

Esse sistema-L nos fornecerá uma palavra f_n, que será utilizada para realizar a construção do floco de neve de Koch através do seguinte método de construção.

Método de construção. Sendo $A = \{0, 1, 2\}$ um alfabeto, $\vec{v}$ um vetor e p_0 um ponto inicial, os termos de A possuem as seguintes atribuições.

$0 \to$ Cria o ponto $p_i = p_{i-1} + \vec{v}$, traçando o segmento de p_{i-1} a p_i.

$1 \to$ Rotaciona o vetor $\vec{v}$ em $+60°$.

$2 \to$ Rotaciona o vetor $\vec{v}$ em $-120°$.

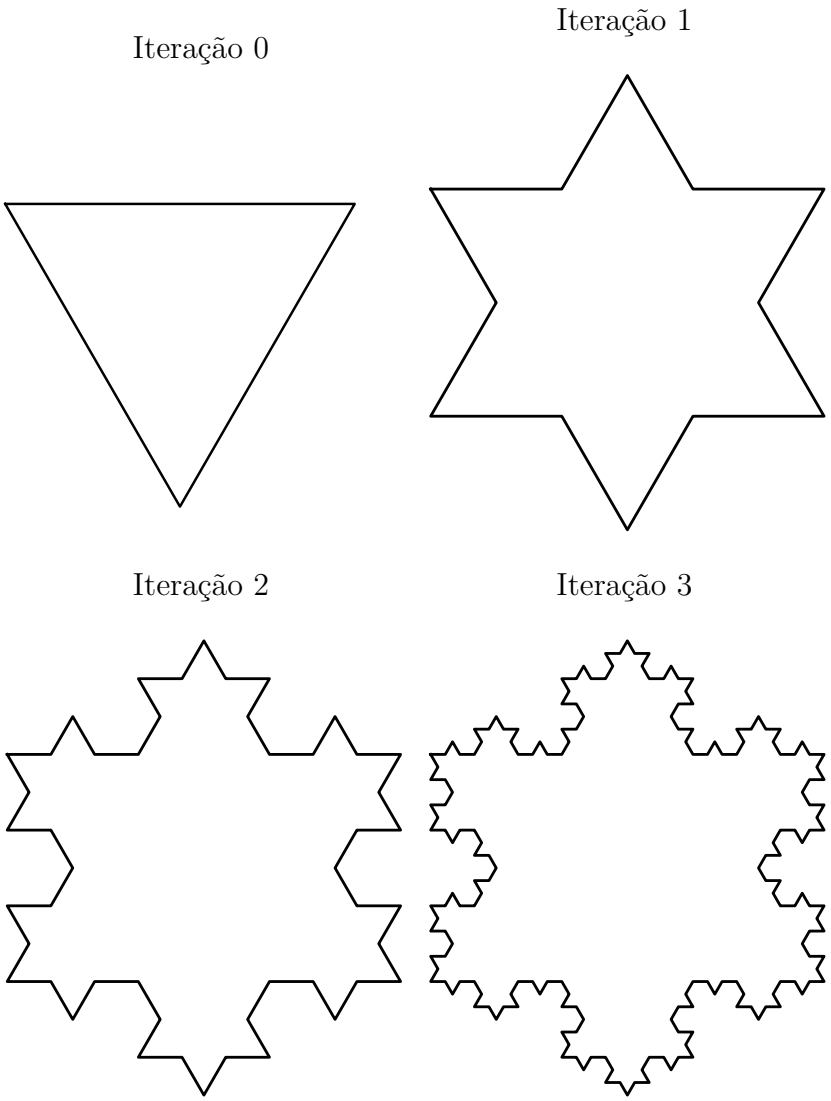


Figura 3.8: Floco de neve de Koch com iterações diferentes

Fonte: Elaborado pelo autor (2024)

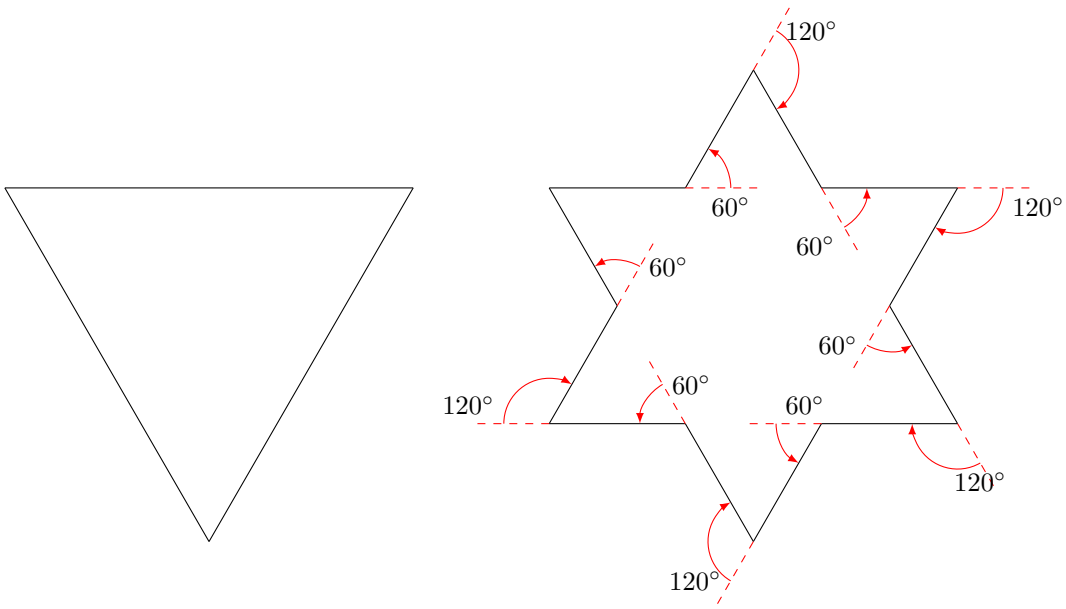
Podemos observar seu desenvolvimento a partir da Figura 3.9.

Obtemos portanto, a partir dessa construção, o floco de neve de Koch, já conhecido e explorado em estudos de Geometria Fractal. Entretanto, aprofundaremos ainda mais essa construção com a utilização dos sistemas-L, utilizando não apenas triângulos, mas diversos polígonos como base de construção do floco.

Para desenvolver essa técnica, tomaremos um polígono regular de l lados, que servirá de base para a construção de nosso fractal, além disso, utilizaremos os mesmos conceitos de ângulos internos e externos apresentados na Definição 3.6.

Ao observar construções já realizadas, podemos notar que para as construções de palavras l-poligonais de Koch, necessitamos de uma regra específica para cada polígono diferente. Este problema é solucionado utilizando o morfismo da Definição 3.7, mas diferente das curvas l-poligonais, que iniciavam sua construção a partir de um segmento, os flocos de neve l-poligonais de Koch necessitam de polígonos diferentes para iniciar suas iterações, portanto, também utilizam de axiomas diferentes em seus respectivos sistemas.

Neste caso, criaremos um axioma condicional, que irá variar de acordo com a quanti-



$f_0 = \texttt{02020}$ $f_1 = \texttt{0102010201020102010201020}$

Figura 3.9: Floco de neve de Koch com destaques

Fonte: Elaborado pelo autor (2024)

dade de lados do polígono (l), possuindo portanto o seguinte formato:

$$f_0 = \texttt{02} * (l-2) \cdot \texttt{02} * \texttt{0}$$

Com esse resultado, podemos enfim construir um sistema-L capaz de criar palavras para gerar os flocos de neve l-poligonais de Koch, bem como um método de construção adequado para cada polígono, sendo estes:

$$\begin{aligned} \text{Alfabeto } (A): &\quad \{\texttt{0}, \texttt{1}, \texttt{2}\} \\ \text{Axioma } (f_0): &\quad \texttt{02} * (l-2) \cdot \texttt{02} * \texttt{0} \\ \text{Regra } (\phi): &\quad \texttt{0} \to \texttt{010} * (l-2) \cdot \texttt{20} * \texttt{10} \\ &\quad \texttt{1} \to \texttt{1} \\ &\quad \texttt{2} \to \texttt{2} \end{aligned}$$

Método de construção. Sendo $A = \{\texttt{0}, \texttt{1}, \texttt{2}\}$ um alfabeto, $\vec{v}$ um vetor e p_0 um ponto inicial, os termos de A possuem as seguintes atribuições.

$\texttt{0} \to$ Cria o ponto $p_i = p_{i-1} + \vec{v}$, traçando o segmento de p_{i-1} a p_i.

$\texttt{1} \to$ Rotaciona o vetor $\vec{v}$ em I_l no sentido anti-horário.

$\texttt{2} \to$ Rotaciona o vetor $\vec{v}$ em E_l no sentido horário.

Com isso, podemos realizar as construções de fractais utilizando os mais diversos polígonos como base. Vejamos na Figura 3.10 alguns exemplos desses fractais com 3 iterações.

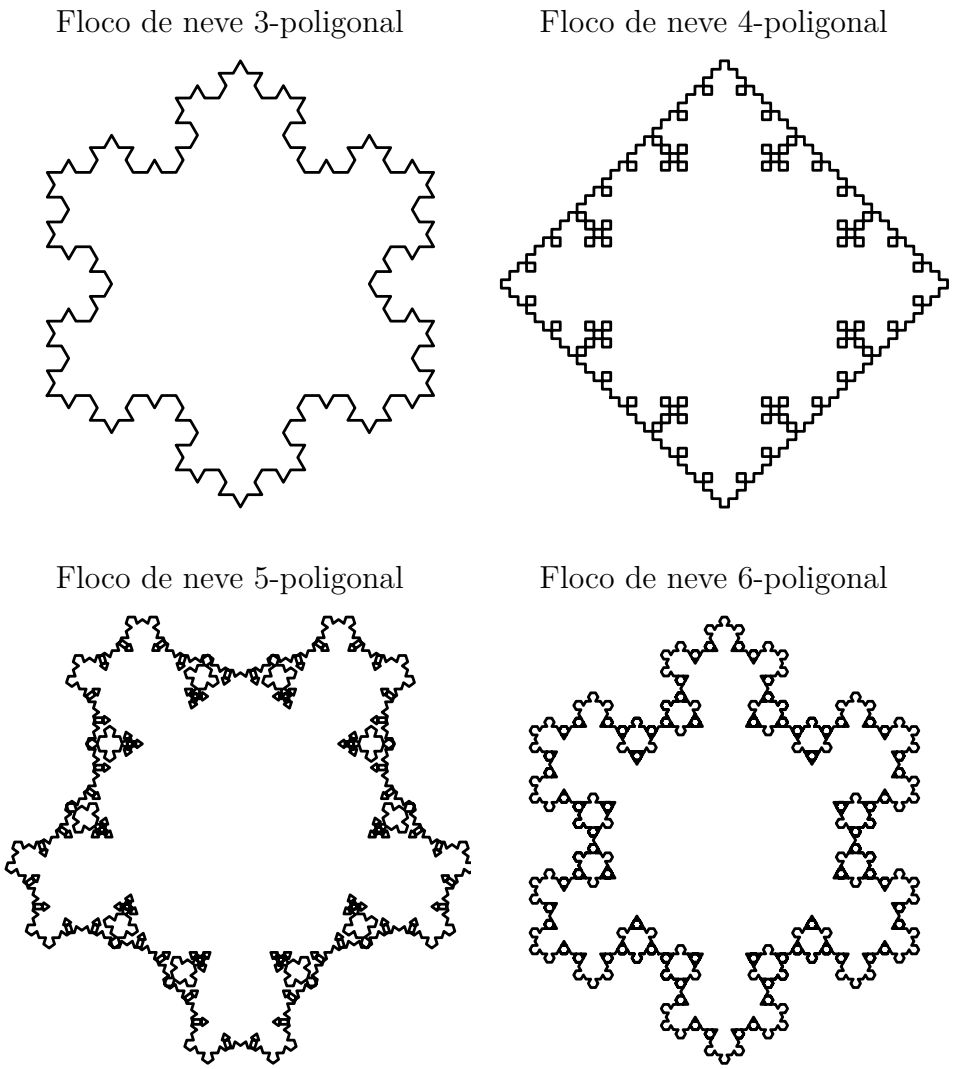


Figura 3.10: Flocos de neve l-poligonais de Koch de iteração 3

Fonte: Elaborado pelo autor (2024)

3.4 Curva de Koch alternada

Como citado anteriormente, um dos objetivos da Geometria Fractal é a construção de modelos que possam representar formas naturais com mais precisão, permitindo assim compreender e elaborar soluções mais avançadas para situações em que a Geometria Euclidiana apresentaria falhas.

Um exemplo claro destes problemas é na representação de regiões costeiras, pois graças às irregularidades apresentadas em sua forma, uma modelagem diretamente por Geometria Euclidiana traria diversas complicações, exigindo diversas etapas e processos.

Por outro lado, utilizando a Geometria Fractal aliada aos sistemas-L, poderíamos construir de forma iterativa e simples uma forma muito aproximada da costa de uma ilha. Em seu livro, Falconer [3] apresenta um modelo no qual, adaptando alguns processos iterativos da curva de Koch, ele encontra o seguinte fractal.

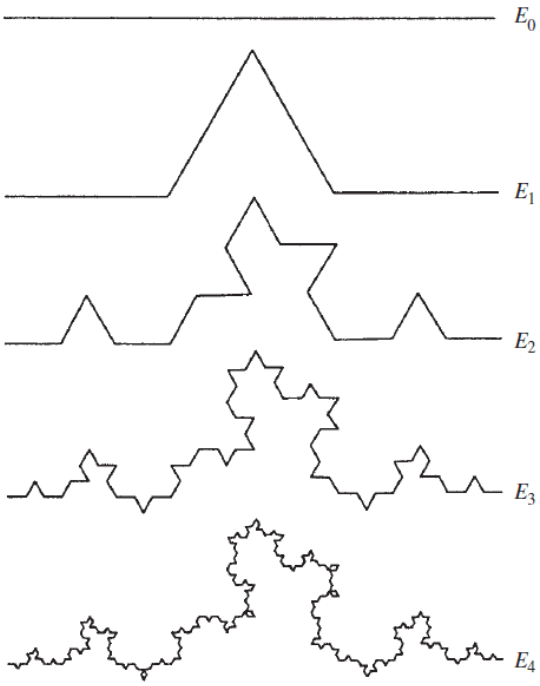


Figura 3.11: Curva aleatória de Koch

Fonte: Mathematical Foundations and Applications [3]

Falconer realiza essa curva através de uma aleatoriedade de seus elementos, podemos notar que as duas primeiras iterações são iguais a curva de Koch original, entretanto, na segunda iteração (E_2) existe em uma de suas pontas uma curva em sentido diferente das demais. No caso, o segundo segmento da primeira iteração (E_1) sofreu iteração em sentido contrário, orientando para "dentro" da figura.

Observando essa construção, podemos explorar novas formas de construção da curva de Koch, alterando os sentidos de construção por um exemplo. Consideremos sentido A sendo o sentido definido inicialmente de construção, os quais determinamos a utilização dos ângulos de $60°$ e $-120°$. Porém, caso apenas invertêssemos o sentido da construção, criando um sentido B que possuiria ângulos de $-60°$ e $120°$, não teríamos nenhuma alteração prática além da reflexão horizontal, como podemos observar na Figura 3.12.

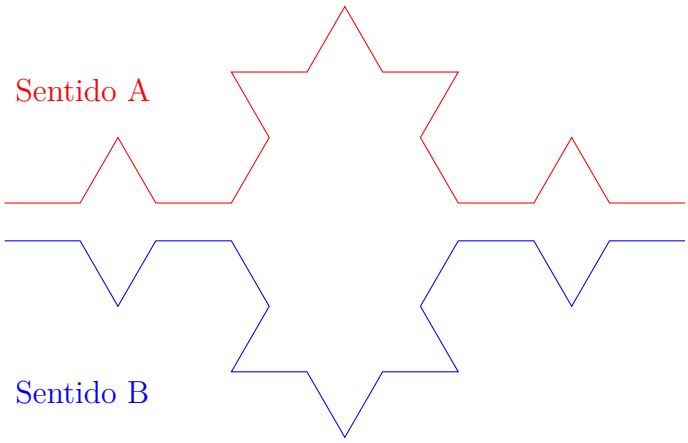


Figura 3.12: Curvas de Koch de iteração 2 em sentidos opostos

Fonte: Elaborado pelo autor (2024)

Portanto, simplesmente realizar a alteração de sentido não apresentaria nenhum resultado diferente dos quais já obtivemos anteriormente. Entretanto, podemos alterná-los entre as iterações. Por exemplo, podemos realizar uma curva de Koch alternada ao trocar o sentido de construção em cada passo, ou seja, se na primeira iteração o triângulo foi posicionado utilizando o sentido A, na segunda será utilizado o sentido B e assim sucessivamente, o que irá nos gerar um novo formato de curva, como podemos ver na Figura 3.13.

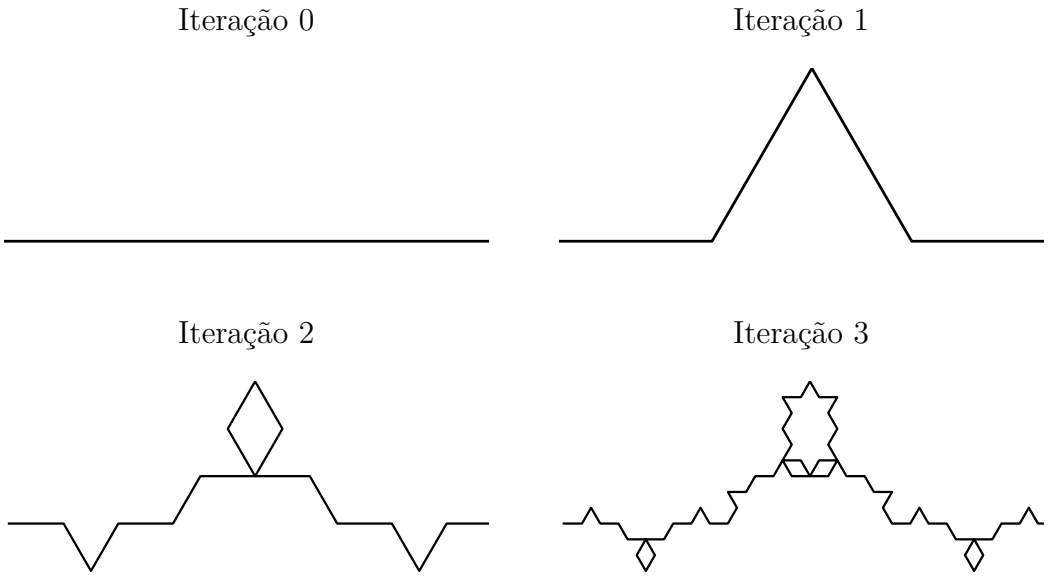


Figura 3.13: Curva de Koch alternada

Fonte: Elaborado pelo autor (2024)

O processo de construção da curva alternada é bem simples, basta que em iterações alternadas a construção de novos segmentos possua sentidos diferentes, ou seja, caso a iteração anterior tenha gerado segmentos no sentido acima da curva, os segmentos da próxima iteração serão posicionados abaixo dela. Representar essa curva através de um novo sistema-L adicionando termos ao alfabeto e colocando novas funções a eles seria

extremamente simples, porém não ideal. Mesmo assim, para deixar claro a ideia, o faremos por hora para depois encontrarmos um meio de representar a curva utilizando a mesma palavra da curva original de Koch.

Considere um novo alfabeto $\{\texttt{x}, \texttt{a}, \texttt{b}, \texttt{m}, \texttt{n}\}$ com as seguintes regras:

$$\begin{aligned} \texttt{x} &\rightarrow \text{ constrói o segmento} \\ \texttt{a} &\rightarrow +60^\circ \\ \texttt{b} &\rightarrow -120^\circ \\ \texttt{m} &\rightarrow -60^\circ \\ \texttt{n} &\rightarrow +120^\circ \end{aligned}$$

Note que `x`, `a` e `b` possuem as mesmas funções das palavras `0`, `1` e `2`, respectivamente. Apenas trocamos os dígitos numéricos por alfabéticos para evitar confusões, já que usaremos números para as posições dos dígitos na palavra.

Portanto, o sistema-L que utilizaremos para representar a curva alternada de Koch, na iteração k, é descrito por:

$$\begin{aligned} \text{Alfabeto } (A): &\quad \{\texttt{x}, \texttt{a}, \texttt{b}, \texttt{m}, \texttt{n}\} \\ \text{Axioma } (f_0): &\quad \texttt{x} \\ \text{Regra } (\phi): &\quad \texttt{x} \rightarrow \texttt{xmxnxmx}, \text{ se } k \text{ é par} \\ &\quad \texttt{x} \rightarrow \texttt{xaxbxax}, \text{ se } k \text{ é ímpar} \\ &\quad \texttt{a} \rightarrow \texttt{a} \\ &\quad \texttt{b} \rightarrow \texttt{b} \\ &\quad \texttt{m} \rightarrow \texttt{m} \\ &\quad \texttt{n} \rightarrow \texttt{n} \end{aligned}$$

A Tabela 3.2 mostra a construção a partir de f_0 das três primeiras iterações. Note que, na segunda e terceira iterações (f_2 e f_3), os dígitos destacados em vermelho foram gerados na primeira iteração (f_1), mas suas posições em cada palavra avançaram graças às iterações dos demais dígitos. Observe também que a primeira aparição do dígito `a` na palavra f_3 ocorre na posição 2, e se repete nas posições 6, 10, 14, 18,..., 126. É notável um padrão claro de repetição do dígito a cada 4 posições a partir da posição 2 ($\text{pos}(\texttt{a}) = 2 + 4i$, $i \in \mathbb{N}$). Entretanto, isso não é válido para todos os dígitos `a` de f_3, por exemplo, os dígitos `a` destacados em vermelho (que foram gerados em f_1) estão nas posições 32 e 92 ($\text{pos}(\texttt{a}) = 32 + 64i$), que não são da forma $2 + 4d$.

f_0	x
$\phi(f_0) = f_1$	x a x b x a x
$\phi(f_1) = f_2$	xmxnxmx a xmxnxmx b xmxnxmx a xmxnxmx
$\phi(f_2) = f_3$	xaxbxax m xaxbxax n xaxbxax m xaxbxax a xaxbxax m xaxbxax n xaxbxax m xaxbxax b xaxbxax m xaxbxax n xaxbxax m xaxbxax a xaxbxax m xaxbxax n xaxbxax m xaxbxax

Tabela 3.2: Palavras de Koch alternadas

Podemos observar que os dígitos `a`, `b`, `m` e `n` não sofrem alteração pelo morfismo, mantendo-se na palavra a partir do momento em que foram gerados. Além disso, iremos verificar mais a frente que dígitos semelhantes de uma palavra f_n que foram gerados em iterações diferentes vão apresentar padrões diferentes. Por exemplo, os dígitos `a`

destacados em vermelho na Tabela 3.2 são gerados na iteração 1, enquanto os dígitos a na cor preta são gerados na iteração 3. Por conta disso, consideraremos não apenas as posições do dígito nas palavra, mas também a iteração que foi gerado através da seguinte definição.

Definição 3.9. Seja d um dígito de f_n, a geração de d, determinado por ger(d), indica em qual iteração esse dígito apareceu pela primeira vez.

Observando a palavra f_3, podemos notar um padrão de repetição para cada um de seus dígitos, destacando também, que no caso de a existirão dois padrões distintos ($\text{pos}(\mathtt{a}) = 2 + 4i$ e $\text{pos}(\mathtt{a}) = 32 + 64i$), dependendo de sua geração ($\text{ger}(\mathtt{a}) = 3$ ou $\text{ger}(\mathtt{a}) = 1$, respectivamente). Em resumo, não apenas o dígito mas também sua geração deverão ser considerados na análise.

Com isso, vamos descrever o comportamento de repetição para cada dígito de f_k dependendo de seus inícios e passos:

- Início $I_{k,j}$: Sejam d dígitos de f_k, com alfabeto Σ, para cada $\text{ger}(\mathtt{d}) = j$ $(1 \leq j \leq k)$ em f_k existirá um Início $(I_{k,j})$, tal que o Início será a menor entre as posições dos dígitos d com geração j, em f_k. Assim,

$$I_{k,j}(\mathtt{d}) = \min\{\text{pos}(\mathtt{d});\ \mathtt{d} \in f_k,\ \text{ger}(\mathtt{d}) = j\},\ 1 \leq j \leq k,\ \mathtt{d} \in \Sigma.$$

  O conjunto com todos os inícios de d em f_k é definido por:

$$\mathcal{I}_k(d) = \{I_{k,k}(d), I_{k,k-1}(d), \ldots, I_{k,1}(d)\},\ k \in \mathbb{N}^*.$$

- Passo $\Delta_{k,j}$: Diferença entre as posições de dois dígitos d de mesma geração consecutivos, ou seja, $\Delta_{k,j}(\mathtt{d}) = I^+_{k,j}(\mathtt{d}) - I_{k,j}(\mathtt{d})$, onde $I^+_{k,j}(\mathtt{d})$ é a segunda menor posição de d, com $\text{ger}(\mathtt{d}) = j$, em f_k.

Note que os dígitos são gerados de forma alternada entre as iterações. Por exemplo, os dígitos a são gerados apenas em iterações ímpares, enquanto m são gerados em iterações pares. Por conta disso, quando formos buscar os inícios de cada dígito, desconsideraremos os casos em que não existe dígito d de geração j.

Portanto, é possível dizer que as aparições de a na terceira iteração ($k = 3$) podem ser descritas por:

Dígito	Geração	Início	Passo
a	$\text{ger}(\mathbf{a}) = 3$	$I_{3,3} = 2$	$\Delta_{3,3}(\mathtt{a}) = 4$
a	$\text{ger}(\mathtt{a}) = 1$	$I_{3,1} = 32$	$\Delta_{3,3}(\mathtt{a}) = 64$

Note que os demais dígitos de f_3 possuem um comportamento semelhante, o qual observa-se a seguir:

Dígito	Geração	Início	Passo
b	$\text{ger}(\mathtt{b}) = 3$	$I_{3,3} = 4$	$\Delta_{3,3}(\mathtt{b}) = 8$
b	$\text{ger}(\mathtt{b}) = 1$	$I_{3,1} = 64$	$\Delta_{3,3}(\mathtt{b}) = 128$
m	$\text{ger}(\mathtt{m}) = 2$	$I_{3,2} = 8$	$\Delta_{3,3}(\mathtt{m}) = 16$
n	$\text{ger}(\mathtt{n}) = 2$	$I_{3,2} = 16$	$\Delta_{3,3}(\mathtt{n}) = 32$

Note que o valor 128 é utilizado como segundo passo do dígito `b`, porém isso se dá apenas para reforçar o padrão de formação da palavra, já que a palavra f_3 possui somente 127 dígitos (ver Corolário 3.4).

A partir dessa visualização podemos notar alguns outros padrões interessantes. O primeiro deles é a relação entre o início e o passo, onde o segundo é sempre o dobro do primeiro. Ou seja, para um dígito com início $I_{k,j}(\mathtt{d}) = \alpha$, o passo desse mesmo dígito será $\Delta_{k,j}(\mathtt{d}) = 2\alpha$. Além disso, os inícios de cada dígito seguem a ordem das potências de 2, onde o início de $\mathtt{a}, \mathtt{b}, \mathtt{m}, \mathtt{n}$ será 2^1, 2^2, 2^3, 2^4, continuando a sequência 2^5 e 2^6 para `a` e `b` respectivamente, apresentando um ciclo.

$$I_{3,3}(\mathtt{a}) = 2^1 \qquad I_{3,2}(\mathtt{m}) = 2^3 \qquad I_{3,1}(\mathtt{a}) = 2^5$$
$$I_{3,3}(\mathtt{b}) = 2^2 \qquad I_{3,2}(\mathtt{n}) = 2^4 \qquad I_{3,1}(\mathtt{b}) = 2^6$$

Para podermos nos aprofundar nesse contexto e simplificar a construção, vamos observar os inícios e passos de cada termo nas iterações $k = 1, 2, 3, 4$:

$k = 1$		
Dígito	**Início**	**Passo**
`a`	$I_{1,1} = 2$	$\Delta_{1,1} = 4$
`b`	$I_{1,1} = 4$	$\Delta_{1,1} = 8$
`m`	$\varnothing$	$\varnothing$
`n`	$\varnothing$	$\varnothing$

$k = 2$		
Dígito	**Início**	**Passo**
`m`	$I_{2,2} = 2$	$\Delta_{2,2} = 4$
`n`	$I_{2,2} = 4$	$\Delta_{2,2} = 8$
`a`	$I_{2,1} = 8$	$\Delta_{2,1} = 16$
`b`	$I_{2,1} = 16$	$\Delta_{2,1} = 32$

$k = 3$		
Dígito	**Início**	**Passo**
`a`	$I_{3,3} = 2$	$\Delta_{3,3} = 4$
`b`	$I_{3,3} = 4$	$\Delta_{3,3} = 8$
`m`	$I_{3,2} = 8$	$\Delta_{3,3} = 16$
`n`	$I_{3,2} = 16$	$\Delta_{3,3} = 32$
`a`	$I_{3,1} = 32$	$\Delta_{3,3} = 64$
`b`	$I_{3,1} = 64$	$\Delta_{3,3} = 128$
`m`	$\varnothing$	$\varnothing$
`n`	$\varnothing$	$\varnothing$

$k = 4$		
Dígito	**Início**	**Passo**
`m`	$I_{4,4} = 2$	$\Delta_{4,4} = 4$
`n`	$I_{4,4} = 4$	$\Delta_{4,4} = 8$
`a`	$I_{4,3} = 8$	$\Delta_{4,3} = 16$
`b`	$I_{4,3} = 16$	$\Delta_{4,3} = 32$
`m`	$I_{4,2} = 32$	$\Delta_{4,2} = 64$
`n`	$I_{4,2} = 64$	$\Delta_{4,2} = 128$
`a`	$I_{4,1} = 128$	$\Delta_{4,1} = 256$
`b`	$I_{4,1} = 256$	$\Delta_{4,1} = 512$

Com isso, podemos notar claramente que existe uma troca nas posições iniciais de $\mathtt{a} \leftrightarrow \mathtt{m}$ e $\mathtt{b} \leftrightarrow \mathtt{n}$ a cada iteração, podendo ser divididas em par e ímpar. A partir disso, vamos estudar os inícios de cada termo para iterações ímpar, tendo em mente que para iterações par o raciocínio será semelhante, apenas havendo a troca mencionada acima.

Portanto, para uma iteração k ímpar, podemos definir os inícios de cada termo facilmente utilizando um $p \in \mathbb{N}$:

$\mathcal{I}(\mathtt{a})$	$\mathcal{I}(\mathtt{b})$	$\mathcal{I}(\mathtt{m})$	$\mathcal{I}(\mathtt{n})$
2^1	2^2	2^3	2^4
2^5	2^6	2^7	2^8
2^9	2^{10}	2^{11}	2^{12}
$\vdots$	$\vdots$	$\vdots$	$\vdots$
2^{4p+1}	2^{4p+2}	2^{4p+3}	2^{4p}

Temos então que para qualquer iteração ímpar, os inícios de `a` serão descritos por 2^{4p+1} e como vimos anteriormente, o passo será seu dobro, portanto $2 \cdot 2^{4p+1}$. Sabemos que a posição de um dígito pode ser descrita a partir do seu início acrescentando uma quantidade finita de passos. Mais precisamente, as posições de `a` (pos(`a`)) após n passos são dadas por:

$$\begin{aligned}\text{pos}(\texttt{a}) &= I_{k,j}(\texttt{a}) + n \cdot \Delta_{k,j}(\texttt{a}) \\ &= I_{k,j}(\texttt{a}) + n \cdot (2 \cdot I_{k,j}(\texttt{a})) \\ &= 2^{4p+1} + 2n \cdot 2^{4p+1} \\ &= 2^{4p+1}(2n+1).\end{aligned}$$

De forma semelhante, podemos descrever as posições de `b`, `m` e `n`, sendo elas respectivamente

$$\text{pos}(\texttt{b}) = 2^{4p+2}(2n+1), \quad \text{pos}(\texttt{m}) = 2^{4p+3}(2n+1), \quad \text{pos}(\texttt{n}) = 2^{4p}(2n+1).$$

Note que todas as descrições de posição são baseadas em uma potência de 2, multiplicando um valor ímpar. Usaremos dessa lógica para criar um método de construção para a curva alternada de Koch, que possa utilizar da palavra original de Koch. Observe também que é essa potência que determinará qual termo estará na posição correspondente, possuindo uma relação cíclica a cada 4 passos, que obedece uma regra módulo 4.

Com isso, podemos descobrir qual dígito estará em uma posição dada, através do seguinte processo.

Seja pos(`d`) $= \alpha$ a posição de um dígito `d` desconhecido de f_k. Para descobrir qual é seu correspondente (`a`, `b`, `m` ou `n`), seguimos os seguintes passos:

- Dividir α por 2 até obter um valor ímpar, encontrando $\alpha = 2^{\lambda}(2i+1)$, com $i \in \mathbb{N}$;
- Tome β satisfazendo $\beta \equiv \lambda \pmod 4$.

Por fim, `d` será:

$$\texttt{d} = \begin{cases} \texttt{a}, & \text{se } \beta = 1, \\ \texttt{b}, & \text{se } \beta = 2, \\ \texttt{m}, & \text{se } \beta = 3, \\ \texttt{n}, & \text{se } \beta = 0. \end{cases}$$

Com isso podemos realizar todo o processo de construção da curva alternada de Koch através da palavra original. Portanto, a construção se dará da seguinte forma:

Palavra (original) de Koch. Seja $A = \{\texttt{0}, \texttt{1}, \texttt{2}\}$ um alfabeto, o sistema-L que gera a palavra de Koch é da forma:

$$\begin{aligned} \text{Alfabeto } (A): &\quad \{\texttt{0}, \texttt{1}, \texttt{2}\} \\ \text{Axioma } (f_0): &\quad \texttt{0} \\ \text{Regra } (\phi): &\quad \texttt{0} \to \texttt{0102010} \\ &\quad \texttt{1} \to \texttt{1} \\ &\quad \texttt{2} \to \texttt{2} \end{aligned}$$

Método de construção (Curva alternada de Koch)[4] Sendo $A = \{0, 1, 2\}$ um alfabeto, k o número de iterações da palavra f_k, $\vec{v}$ um vetor, p_0 um ponto inicial, α a posição de um dígito `d` de f_k e $\lambda \in \mathbb{N}$ tal que $\alpha = 2^{\lambda}(2j + 1)$, com $j \in \mathbb{N}$, os termos de A possuem as seguintes atribuições, dependendo da paridade de k:

Para k ímpar:

`0` → Cria o ponto $p_i = p_{i-1} + \vec{v}$, traçando o segmento de p_{i-1} a p_i.

`1` → Rotaciona o vetor $\vec{v}$ em:

- $+60°$, se $\lambda \equiv 1 \pmod 4$; (comportamento semelhante ao dígito `a`)
- $-60°$, se $\lambda \equiv 3 \pmod 4$. (comportamento semelhante ao dígito `m`)

`2` → Rotaciona o vetor $\vec{v}$ em:

- $-120°$, se $\lambda \equiv 2 \pmod 4$; (comportamento semelhante ao dígito `b`)
- $+120°$, se $\lambda \equiv 0 \pmod 4$. (comportamento semelhante ao dígito `n`)

Para k par:

`0` → Cria o ponto $p_i = p_{i-1} + \vec{v}$, traçando o segmento de p_{i-1} a p_i.

`1` → Rotaciona o vetor $\vec{v}$ em:

- $+60°$, se $\lambda \equiv 3 \pmod 4$; (comportamento semelhante ao dígito `a`)
- $-60°$, se $\lambda \equiv 1 \pmod 4$. (comportamento semelhante ao dígito `m`)

`2` → Rotaciona o vetor $\vec{v}$ em:

- $-120°$, se $\lambda \equiv 0 \pmod 4$; (comportamento semelhante ao dígito `b`)
- $+120°$, se $\lambda \equiv 2 \pmod 4$. (comportamento semelhante ao dígito `n`)

Para melhor visualização, podemos observar na Figura 3.14 a palavra original f_2 destacada sobre a curva alternada de Koch de segunda iteração, com a coloração semelhante à Tabela 3.2.

`xmxnxmx a xmxnxmx b xmxnxmx a xmxnxmx`

[4] Vídeo: https://www.youtube.com/watch?v=rb5V-x0vKZ4

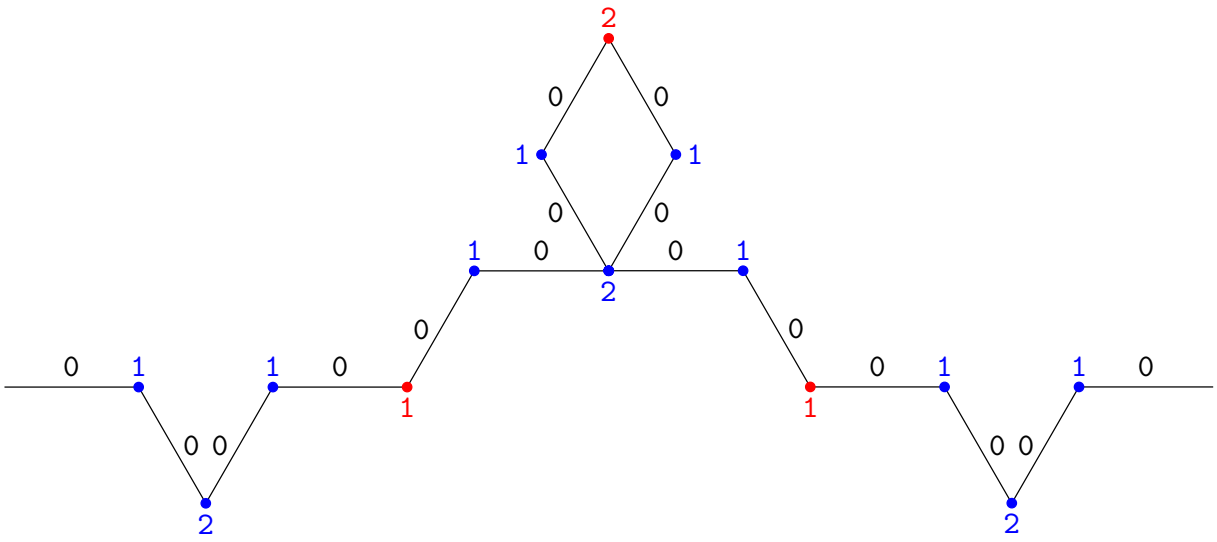


Figura 3.14: Curva alternada de Koch de iteração $k = 2$ com palavra destacada

Fonte: Elaborado pelo autor (2024)

4 Programação em Python

Ao longo desse trabalho, algumas imagens referentes às curvas do dragão, de Koch, l-poligonal de Koch e de Koch alternada foram geradas através de programas Python desenvolvidos pelo autor. Os programas trazem a lógica dos sistemas-L para uma linguagem computacional, permitindo que usemos os métodos discutidos nesse trabalho em um código, possibilitando assim uma visualização mais avançada e agilizada das curvas idealizadas.

Os programas permitem algumas variações nas curvas, além disso, a quantidade de iterações que a curva sofrerá é definida pelo usuário, permitindo que crie imagens de iterações diversas de acordo com sua vontade.

Abaixo estão alguns dos códigos desenvolvidos ao longo dos estudos nesse trabalho. Os códigos também estão disponíveis na plataforma GitHub© e podem ser acessados no link https://github.com/tmelorc/L-system (ver [8]).

4.1 Curva do dragão

```
from operator import add
import matplotlib.pyplot as plt
import numpy as np


def morphism(L):
    return L + [1] + [-x for x in L[::-1]]


def get_direction(x):
    if x == 0:
        return directions[-1]
    if x == 1:
        return [-directions[-1][1], directions[-1][0]]
    if x == -1:
        return [directions[-1][1], -directions[-1][0]]
    else:
        print('Erro no argumento da função.')


def plot(x):
    if x == 0:
        new_vertice = list(map(add, vertices[-1], directions[-1]))
```

```
        vertices.append(new_vertice)
        directions.append(directions[-1])

    else:
        new_direction = get_direction(x)
        directions.append(new_direction)

    X = [v[0] for v in vertices]
    Y = [v[1] for v in vertices]
    plt.plot(X, Y, c='b', linewidth=4)


colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd',
          '#8c564b', '#e377c2', '#7f7f7f', '#bcbd22', '#17becf']

i = int(input('Digite o número de iterações:'))
word = [0]

for a in range(0, i):
    word = morphism(word)

directions = [[(np.cos((np.pi/2) - (i*(np.pi/4)))) * ((np.sqrt(2)/2)**i),
               (np.sin((np.pi/2) - (i*(np.pi/4)))) * ((np.sqrt(2)/2)**i)]]

vertices = [[0, 0]]


for x in word:
    plot(x)

# Opções de vizualização:
plt.title(f'Curva do dragão de iteração {i}')       # Inserir título
plt.axis('off')    # Retirar eixos e caixas
plt.axis('equal')  # Mesma escala nos eixos
plt.savefig(f'Curva do dragão de iteração {i}.pdf') # Salvar figura em PDF
plt.show()         # Mostra a figura
```

Programa 4.1: Curva do Dragão

4.1.1 Animação

Utilizando a biblioteca Manim do Python, podemos criar arquivos de vídeo MP4, com animações das curvas estudadas nesse trabalho. Para mais detalhes dessa biblioteca, ver www.manim.community.

```
from manim import *

OPAQUE = DARK_GRAY

```

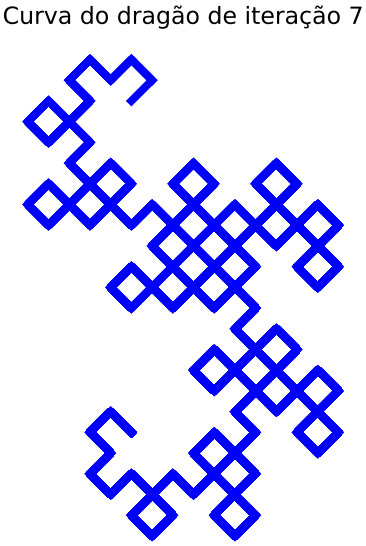


Figura 4.1: Saída do programa: Curva do dragão de iteração 7

```
5
6  class Path(VMobject):
7      def __init__(self, points, *args, **kwargs):
8          super().__init__(*args, **kwargs)
9          self.set_points_as_corners(points)
10
11     def get_important_points(self):
12         return list(self.get_start_anchors()) +
           ↪  [self.get_end_anchors()[-1]]
13
14
15 class Dragon(Scene):
16     def construct(self):
17
18         author = Text('T. de Melo and G. Biban',
19                       font_size=12,
20                       fill_opacity=.7,
21                       color=WHITE
22                       ).to_corner(DR, buff=0.2)
23         self.add(author)
24
25         new_scale = 1 / np.sqrt(2)
26         num_iteractions = 10
27
28         txt = Tex('$i=0$').to_corner(UP).set_color(BLUE).scale(1.3)
29         points = [2 * LEFT, 2 * RIGHT]
30         curve = Path(points).set_color(ORANGE).scale(1.5).shift(0.5 * UP)
31         self.play(FadeIn(curve), FadeIn(txt))
32         self.wait(.5)
```

```
33
34          for i in range(1, num_iteractions + 1):
35              self.remove(txt)
36              txt = Tex(f'$i={i}$').to_corner(UP).set_color(BLUE).scale(1.3)
37
38              left_curve = curve.copy()
39              A = left_curve.get_start()
40              self.play(left_curve.animate
41                        .set_color(OPAQUE)
42                        .rotate(-PI / 4, about_point=A)
43                        .scale(new_scale, about_point=A),
44                        FadeIn(txt)
45                        )
46
47              right_curve = left_curve.copy()
48              B = right_curve.get_end()
49              self.play(right_curve.animate
50                        .set_color(OPAQUE)
51                        .rotate(-PI / 2, about_point=B)
52                        )
53
54              self.remove(curve)
55
56              new_curve = Path(
57                  list(left_curve.get_important_points()) +
58                  list(right_curve.get_important_points()[::-1])
59              ).set_color(ORANGE)
60
61              self.play(Create(new_curve, run_time=1),
62                        rate_func=linear,
63                        )
64
65              self.remove(left_curve, right_curve)
66              curve = new_curve
67              self.wait(.5)
68
69          self.play(FadeOut(curve))
70
```

Programa 4.2: Animação em Manim para a curva do dragão

4.2 Curva de Koch

```
1 from operator import add
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd',
6           '#8c564b', '#e377c2', '#7f7f7f', '#bcbd22', '#17becf']
7
```

```
f = '0'  # Para construir o floco de neve, basta substituir por '02020'


def morphism(n, f):
    for i in range(n):
        f = f.replace('0', '0102010')
    return f


def get_direction(d):
    if d == '0':
        # Se d = 0, mantém direção
        return directions[-1]
    if d == '1':
        # Se d = 1, muda direção em +60°
        return [1/2*directions[-1][0]-np.sqrt(3)/2 * directions[-1][1],
        ↪  +1/2 * directions[-1][1]+np.sqrt(
            3)/2 * directions[-1][0]]
    if d == '2':
        # Se d = 2, muda direção em -120°
        return [-1/2*directions[-1][0]+np.sqrt(3)/2 * directions[-1][1],
        ↪  -1/2 * directions[-1][1]-np.sqrt(
            3)/2 * directions[-1][0]]
    else:
        print('Erro no argumento da função.')


n = int(input('Digite o número de iterações:'))

word = morphism(n, f)
directions = [[1, 0]]
vertices = [[0, 0]]

for d in word:
    if d == '0':
        new_vertice = list(map(add, vertices[-1], directions[-1]))
        vertices.append(new_vertice)
        directions.append(directions[-1])
    else:
        new_direction = get_direction(d)
        directions.append(new_direction)

vertices0 = [[0, 0]]
directions0 = [[1, 0]]

for d in f:
    if d == '0':
        new_vertice = list(map(add, vertices0[-1], directions0[-1]))
        vertices0.append(new_vertice)
        directions0.append(directions0[-1])
    else:
```

```
57          new_direction = get_direction(d)
58          directions0.append(new_direction)
59
60  # /(3**n) e /(np.sqrt(12)*(3**(n-1))) mantém a escala de redução original
    ↪  da curva, mas visualmente são opcionais
61  X = [v[0]/(3**n) for v in vertices]
62  Y = [v[1]/(np.sqrt(12)*(3**(n-1))) for v in vertices]
63
64  plt.plot(X, Y, c='b', linewidth=4)
65  plt.title(f'Curva de Koch de iteração {n}')        # Inserir título
66  plt.axis('off')    # Retirar eixos e caixas
67  plt.axis('equal')  # Mesma escala nos eixos
68  plt.savefig(f'Curva de Koch de iteração {n}.pdf')  # Salvar figura em PDF
69  plt.show()         # Mostra a figura
```

Programa 4.3: Curva de Koch

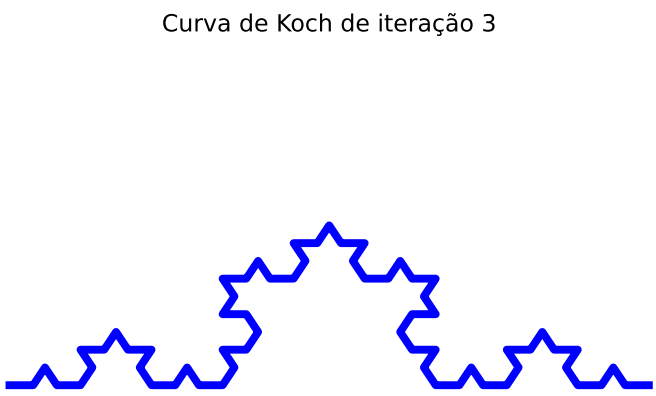


Figura 4.2: Saída do programa: Curva de Koch de iteração 3

4.2.1 Animação

```
1  from manim import *
2
3  OPAQUE = DARK_GRAY
4
5
6  class Path(VMobject):
7      def __init__(self, points, *args, **kwargs):
8          super().__init__(*args, **kwargs)
9          self.set_points_as_corners(points)
10
11     def get_important_points(self):
12         return list(self.get_start_anchors()) +
           ↪  [self.get_end_anchors()[-1]]
13
14
```

```
class Koch(Scene):
    def construct(self):

        thiago = Text('T. de Melo and G. Biban',
                      font_size=12,
                      fill_opacity=.7,
                      color=WHITE
                      ).to_corner(DR, buff=0.2)
        self.add(thiago)

        new_scale = 1 / 3
        num_iteractions = 5
        line_stroke = DEFAULT_STROKE_WIDTH

        txt = Tex('$i=0$').to_corner(UP).set_color(BLUE).scale(1.3)
        points = [1.5 * LEFT, 1.5 * RIGHT]
        curve = Path(points).set_color(ORANGE).scale(3).shift(1 * DOWN)
        self.play(FadeIn(curve), FadeIn(txt))
        self.wait(.5)

        for i in range(1, num_iteractions + 1):
            self.remove(txt)
            txt = Tex(f'$i={i}$').to_corner(UP).set_color(BLUE).scale(1.3)

            left_curve = curve.copy()
            right_curve = curve.copy()

            ''' lado esquerdo '''
            A = left_curve.get_start()
            self.play(left_curve.animate
                      .set_color(OPAQUE)
                      #   .rotate(-PI / 4, about_point=A)
                      .scale(new_scale, about_point=A),
                      FadeIn(txt)
                      )

            left_stair = left_curve.copy()
            # B = left_stair.get_start()
            self.play(left_stair.animate
                      .set_color(OPAQUE)
                      .rotate(PI / 3, about_point=left_curve.get_start())
                      .move_to(left_curve.get_end(),
                      ↪  aligned_edge=(LEFT+DOWN))
                      )

            ''' lado direito '''
            A = right_curve.get_end()
            self.play(right_curve.animate
                      .set_color(OPAQUE)
                      #   .rotate(-PI / 4, about_point=A)
                      .scale(new_scale, about_point=A),
```

```
                    #   FadeIn(txt)
                    )

            right_stair = right_curve.copy()
            # B = right_stair.get_start()
            self.play(right_stair.animate
                      .set_color(OPAQUE)
                      .rotate(- PI / 3, about_point=A)
                      .move_to(right_curve.get_start(),
                      ↪  aligned_edge=(RIGHT+DOWN))
                      )

            self.remove(curve)

            if i > 3:
                line_stroke *= .8

            new_curve = Path(
                list(left_curve.get_important_points()) +
                list(left_stair.get_important_points()) +
                list(right_stair.get_important_points()) +
                list(right_curve.get_important_points()),
                ↪  stroke_width=line_stroke
            )

            self.play(Create(new_curve, run_time=1),
                      rate_func=linear,
                      )

            self.remove(left_curve, left_stair, right_curve, right_stair)
            curve = new_curve
            self.wait(.5)

        self.play(FadeOut(curve))
        self.wait(1)
```

Programa 4.4: Animação em Manim para a curva de Koch

4.3 Curva l-poligonal de Koch

```python
from operator import add
import matplotlib.pyplot as plt
import numpy as np


def morfismo(l, n, f):
    for i in range(n):
        f = f.replace('0', '010' + (l-2) * '20' + '10')
    return f


def rotation(vetor, ang):
    x, y = vetor[0], vetor[1]
    return [x*np.cos(ang) - y*np.sin(ang), x*np.sin(ang) + y*np.cos(ang)]


def get_direction(d):
    if d == '0':
        return directions[-1]

    if d == '1':
        return rotation(directions[-1], angle)

    if d == '2':
        return rotation(directions[-1], negative_angle)


def plot():
    for d in word:
        if d == '0':
            new_vertice = list(map(add, vertices[-1], directions[-1]))
            vertices.append(new_vertice)
            directions.append(directions[-1])
        else:
            new_direction = get_direction(d)
            directions.append(new_direction)

    X = [v[0] for v in vertices]
    Y = [v[1] for v in vertices]
    plt.plot(X, Y, c='b', linewidth=3)

    # Inserir título
    plt.title(f'Curva {l}-poligonal de iteração {n}')
    plt.axis('off')    # Retirar eixos e caixas
    plt.axis('equal')  # Mesma escala nos eixos
    # Salvar figura em PDF
    plt.savefig(f'Curva {l}-poligonal de iteração {n}.pdf')
    plt.show()         # Mostra a figura

```

```
50
51 colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd',
52           '#8c564b', '#e377c2', '#7f7f7f', '#bcbd22', '#17becf']
53
54 l = int(input('Digite o número de lados do polígono:'))
55 n = int(input('Digite o número de iterações:'))
56 tipo = input(
57     f'Escolha o tipo de curva:\n s -> curva simples\n p -> polígonal
     ↪  fechada\n ')
58
59 if tipo == 's':
60     # curva simples
61     f = '0'
62 if tipo == 'p':
63     # curva fechada
64     f = l * '02'
65 if tipo not in 'sp':
66     print(f'O tipo da curva deve ser s ou p.')
67     quit()
68
69
70 word = morfismo(l, n, f)
71 vertices = [[0, 0]]
72 directions = [[1, 0]]
73 angle = ((l-2) * np.pi) / l
74 negative_angle = (angle - np.pi)
75
76 plot()
```

Programa 4.5: Curva l-poligonal de Koch

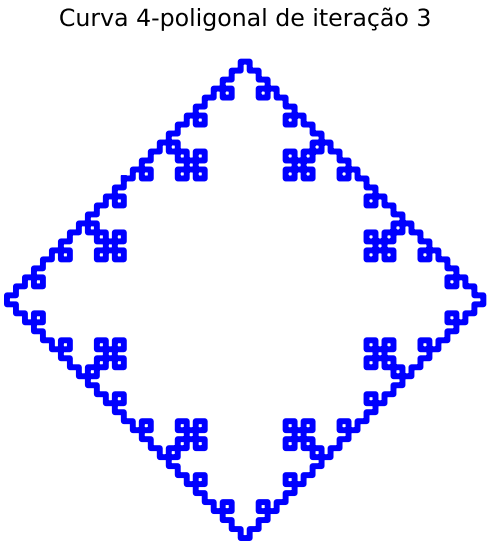


Figura 4.3: Saída do programa: Curva 4-poligonal de iteração 3

4.3.1 Animação

```
class KochLPol(Scene):
    def construct(self):
        author = Text('T. de Melo and G. Biban',
                      font_size=12,
                      fill_opacity=.7,
                      color=WHITE
                      ).to_corner(DR, buff=0.2)
        self.add(author)

        new_scale = 1 / 3
        num_iteractions = 5
        line_stroke = DEFAULT_STROKE_WIDTH

        txt = Tex('$i=0$').to_corner(DOWN,
        ↪  buff=1).set_color(BLUE).scale(1.3)
        points = [1.5 * LEFT, 1.5 * RIGHT]
        curve = Path(points).set_color(WHITE).scale(3).shift(DOWN)
        self.play(FadeIn(curve), FadeIn(txt))
        self.wait(.5)

        for i in range(1, num_iteractions + 1):
            self.remove(txt)
            txt = Tex(f'$i={i}$').to_corner(
                DOWN, buff=1).set_color(BLUE).scale(1.3)

            left_curve = curve.copy()

            A = left_curve.get_start()
            self.play(left_curve.animate
                      .set_color(OPAQUE)
                      .scale(new_scale, about_point=A),
                      FadeIn(txt)
                      )

            B = left_curve.get_end()
            left_stair = curve.copy()
            self.play(left_stair.animate
                      .set_color(OPAQUE)
                      .scale(new_scale, about_point=B)
                      .rotate(PI / 2, about_point=left_curve.get_start())
                      .move_to(left_curve.get_end(),
                      ↪  aligned_edge=(RIGHT+DOWN))
                      )

            C = left_stair.get_end()
            top = curve.copy()
            self.play(top.animate
                      .set_color(OPAQUE)
                      .scale(new_scale, about_point=C)
```

```
                    .move_to(left_stair.get_end(),
                    ↪  aligned_edge=(LEFT+DOWN))
                    )

            D = top.get_end()
            right_stair = curve.copy()
            self.play(right_stair.animate
                    .set_color(OPAQUE)
                    .scale(new_scale, about_point=D)
                    .rotate(-PI/2, about_point=C)
                    .move_to(top.get_end(), aligned_edge=(LEFT+UP))
                    )

            E = right_stair.get_end()
            right_curve = curve.copy()
            self.play(right_curve.animate
                    .set_color(OPAQUE)
                    .scale(new_scale, about_point=E)
                    .move_to(right_stair.get_end(),
                    ↪  aligned_edge=(LEFT+DOWN))
                    )

            self.remove(curve)

            if i > 3:
                line_stroke *= .8

            new_curve = Path(
                list(left_curve.get_important_points()) +
                list(left_stair.get_important_points()) +
                list(top.get_important_points()) +
                list(right_stair.get_important_points()) +
                list(right_curve.get_important_points()),
                ↪  stroke_width=line_stroke
            )

            self.play(Create(new_curve, run_time=1),
                    rate_func=linear,
                    )

            self.remove(left_curve, left_stair, top, right_curve,
            ↪  right_stair)
            curve = new_curve
            self.wait(.5)

        self.play(FadeOut(curve))
        self.wait(1)
```

Programa 4.6: Animação em Manim para a Curva l-poligonal de Koch

4.4 Curva alternada de Koch

```
from operator import add
import matplotlib.pyplot as plt
import numpy as np


def morf(n, f):
    for i in range(n):
        f = f.replace('0', '0102010')
    return f


def get_direction(i, d):
    k = 0
    while i % 2 == 0:
        k += 1
        i = i / 2

    if n % 2 == 1:
        if k % 4 == 1:  # +60
            return [1/2*direcoes[-1][0]-np.sqrt(3)/2 * direcoes[-1][1],
                    +1/2 * direcoes[-1][1]+np.sqrt(3)/2 * direcoes[-1][0]]

        if k % 4 == 2:  # -120
            return [-1/2*direcoes[-1][0]+np.sqrt(3)/2 * direcoes[-1][1],
                    -1/2 * direcoes[-1][1]-np.sqrt(3)/2 * direcoes[-1][0]]

        if k % 4 == 3:  # -60
            return [1/2*direcoes[-1][0]+np.sqrt(3)/2 * direcoes[-1][1],
                    +1/2 * direcoes[-1][1]-np.sqrt(3)/2 * direcoes[-1][0]]

        if k % 4 == 0:  # +120
            return [-1/2*direcoes[-1][0]-np.sqrt(3)/2 * direcoes[-1][1],
                    -1/2 * direcoes[-1][1]+np.sqrt(3)/2 * direcoes[-1][0]]
    else:
        if k % 4 == 3:  # +60
            return [1/2*direcoes[-1][0]-np.sqrt(3)/2 * direcoes[-1][1],
                    +1/2 * direcoes[-1][1]+np.sqrt(3)/2 * direcoes[-1][0]]

        if k % 4 == 0:  # -120
            return [-1/2*direcoes[-1][0]+np.sqrt(3)/2 * direcoes[-1][1],
                    -1/2 * direcoes[-1][1]-np.sqrt(3)/2 * direcoes[-1][0]]

        if k % 4 == 1:  # -60
            tmp = [1/2*direcoes[-1][0]+np.sqrt(3)/2 * direcoes[-1][1],
                   +1/2 * direcoes[-1][1]-np.sqrt(3)/2 * direcoes[-1][0]]

        if k % 4 == 2:  # +120
            tmp = [-1/2*direcoes[-1][0]-np.sqrt(3)/2 * direcoes[-1][1],
                   -1/2 * direcoes[-1][1]+np.sqrt(3)/2 * direcoes[-1][0]]
```

```
50
51
52 n = int(input('Digite o número de iterações:'))
53
54 f = '0'
55 word = morf(n, f)
56
57 direcoes = [[3, 0]]
58 vertices = [[0, 0]]
59
60 for i, d in enumerate(word, start=1):
61     if d == '0':
62         new_vertice = list(map(add, vertices[-1], direcoes[-1]))
63         vertices.append(new_vertice)
64         direcoes.append(direcoes[-1])
65     else:
66         new_direction = get_direction(i, d)
67         direcoes.append(new_direction)
68
69 X = [v[0] for v in vertices]
70 Y = [v[1] for v in vertices]
71
72 plt.plot(X, Y, c='b', linewidth=3)
73 # Inserir título
74 plt.title(f'Curva de Koch alternada de iteração {n}')
75 plt.axis('off')    # Retirar eixos e caixas
76 plt.axis('equal')  # Mesma escala nos eixos
77 # Salvar figura em PDF
78 plt.savefig(f'Curva de Koch alternada de iteração {n}.pdf')
79 plt.show()         # Mostra a figura
```

Programa 4.7: Curva alternada de Koch

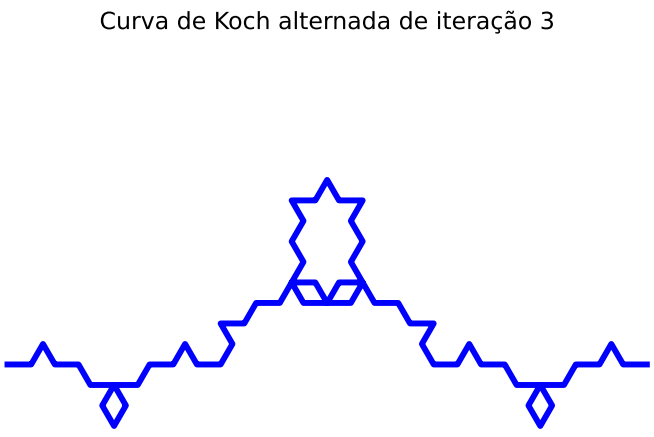


Figura 4.4: Saída do programa: Curva de Koch alternada de iteração 3

4.4.1 Animação

```
from manim import *

OPAQUE = DARK_GRAY


class Path(VMobject):
    def __init__(self, points, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.set_points_as_corners(points)

    def get_important_points(self):
        return list(self.get_start_anchors()) +
        ↪  [self.get_end_anchors()[-1]]


class KochAlt(Scene):
    def construct(self):
        author = Text('T. de Melo and G. Biban',
                      font_size=12,
                      fill_opacity=.7,
                      color=WHITE
                      ).to_corner(DR, buff=0.2)
        self.add(author)

        new_scale = 1 / 3
        num_iteractions = 5
        line_stroke = DEFAULT_STROKE_WIDTH

        txt = Tex('$i=0$').to_corner(UP).set_color(BLUE).scale(1.3)
        points = [1.5 * LEFT, 1.5 * RIGHT]
        curve = Path(points).set_color(ORANGE).scale(3).shift(DOWN)
        self.play(FadeIn(curve), FadeIn(txt))
        self.wait(.5)

        for i in range(1, num_iteractions + 1):
            self.remove(txt)
            txt = Tex(f'$i={i}$').to_corner(UP).set_color(BLUE).scale(1.3)

            left_curve = curve.copy()
            right_curve = curve.copy()

            ''' lado esquerdo '''
            A = left_curve.get_start()

            self.play(left_curve.animate
                      .set_color(OPAQUE)
                      .scale(new_scale, about_point=A),
                      FadeIn(txt)
                      )
```

```
49
50              if i == 2:
51                  self.play(left_curve.animate
52                            .flip(LEFT)
53                            .shift(np.sqrt(3)/2 * DOWN)
54                           )
55              if i > 2:
56                  self.play(left_curve.animate
57                            .flip(LEFT)
58                            .shift((np.sqrt(3)/3 * DOWN))
59                           )
60
61              left_stair = left_curve.copy()
62              self.play(left_stair.animate
63                        .set_color(OPAQUE)
64                        .rotate(PI / 3, about_point=left_curve.get_start())
65                        .move_to(left_curve.get_end(),
                          ↪  aligned_edge=(LEFT+DOWN))
66                        )
67
68              ''' lado direito '''
69              A = right_curve.get_end()
70
71              self.play(right_curve.animate
72                        .set_color(OPAQUE)
73                        .scale(new_scale, about_point=A)
74                        )
75
76              if i == 2:
77                  self.play(right_curve.animate
78                            .flip(LEFT)
79                            .shift(np.sqrt(3)/2 * DOWN)
80                           )
81              if i > 2:
82                  self.play(right_curve.animate
83                            .flip(LEFT)
84                            .shift((np.sqrt(3)/3 * DOWN))
85                           )
86
87              right_stair = right_curve.copy()
88              self.play(right_stair.animate
89                        .set_color(OPAQUE)
90                        .rotate(- PI / 3, about_point=A)
91                        .shift(3*LEFT)
92                        )
93
94              self.remove(curve)
95
96              if i > 3:
97                  line_stroke *= .8
98
```

```
            new_curve = Path(
                list(left_curve.get_important_points())
                + list(left_stair.get_important_points())
                + list(right_stair.get_important_points())
                + list(right_curve.get_important_points()),
                ↪  stroke_width=line_stroke
            )

            self.play(Create(new_curve, run_time=1), rate_func=linear)

            self.remove(left_curve, left_stair, right_curve, right_stair)
            curve = new_curve
            self.wait(.5)

        self.play(FadeOut(curve))
        self.wait(1)
```

Programa 4.8: Animação em Manim para a curva alternada de Koch

4.4.2 Animação

```
class KochSemiAlt(Scene):
    def construct(self):
        author = Text('T. de Melo and G. Biban',
                      font_size=12,
                      fill_opacity=.7,
                      color=WHITE
                      ).to_corner(DR, buff=0.2)
        self.add(author)

        new_scale = 1 / 3
        num_iteractions = 5
        line_stroke = DEFAULT_STROKE_WIDTH

        txt = Tex('$i=0$').to_corner(UP).set_color(BLUE).scale(1.3)
        points = [1.5 * LEFT, 1.5 * RIGHT]
        curve = Path(points).set_color(ORANGE).scale(3).shift(DOWN)
        self.play(FadeIn(curve), FadeIn(txt))
        self.wait(.5)

        for i in range(1, num_iteractions + 1):
            self.remove(txt)
            txt = Tex(f'$i={i}$').to_corner(UP).set_color(BLUE).scale(1.3)

            left_curve = curve.copy()
            right_curve = curve.copy()

            ''' lado esquerdo '''
            A = left_curve.get_start()
            self.play(left_curve.animate
```

```
                    .set_color(OPAQUE)
                    .scale(new_scale, about_point=A),
                    FadeIn(txt)
                    )

            left_stair = left_curve.copy()
            B = left_stair.get_end()
            self.play(left_stair.animate
                    .set_color(OPAQUE)
                    .rotate(- 2 * PI / 3, about_point=B)
                    )

            ''' lado direito '''
            A = right_curve.get_end()
            self.play(right_curve.animate
                    .set_color(OPAQUE)
                    .scale(new_scale, about_point=A),
                    )

            right_stair = right_curve.copy()
            B = right_stair.get_start()
            self.play(right_stair.animate
                    .set_color(OPAQUE)
                    .rotate(2 * PI / 3, about_point=B)
                    )

            self.remove(curve)

            if i > 3:
                line_stroke *= .8

            new_curve = Path(
                list(left_curve.get_important_points()) +
                list(left_stair.get_important_points()[::-1]) +
                list(right_stair.get_important_points()[::-1]) +
                list(right_curve.get_important_points())
            )

            self.play(Create(new_curve, run_time=1), rate_func=linear)

            self.remove(left_curve, left_stair, right_curve, right_stair)
            curve = new_curve
            self.wait(.5)

        self.play(FadeOut(curve))
        self.wait(1)
```

Programa 4.9: Animação em Manim para a curva semi-alternada de Koch

Códigos. Todos os códigos acima, bem como os vídeos das animações, estão disponíveis para o leitor no endereço https://github.com/tmelorc/L-system ([8]).

Considerações finais

Com o conteúdo desse trabalho é possível uma abordagem visual avançada dos fractais. Enquanto nos livros e artigos os fractais são vistos apenas em poucas iterações, com a programação aqui apresentada, é possível escolher qual iteração utilizar e assim, observar o processo de desenvolvimento dos fractais. Utilizando essas tecnologias em sala de aula, o professor possui um facilitador no ensino de processos iterativos, Geometria Fractal, ou até mesmo em áreas computacionais.

Além disso, a representação da construções de fractais através de vetores pode ser utilizado como uma forma de aprofundamento quando no ensino de Geometria Analítica. Também, as variações dos fractais podem ser utilizados como objetos de estudo em disciplinas como Análise Real, no cálculo de áreas e perímetro das figuras e como as alterações afetam esses fatores.

O trabalho apresenta também, interdisciplinariedade com a Biologia, favorecendo uma ponte entre a Matemática e as demais áreas das Ciências da Natureza. Entre elas, o estudo da relação entre fractais e o desenvolvimento de determinadas vegetações é uma possibilidade interessante para desenvolvimentos futuros, pois além possibilitar a previsão de crescimento de algumas espécies, pode auxiliar no diagnóstico de pragas e doenças agrícolas.

Na Medicina, também são possíveis estudos utilizando fractais, desde o diagnóstico de tumores até a compreensão do desenvolvimento de tecidos e órgãos. Sendo assim, a área ainda apresenta diversas possibilidades de abordagens e desenvolvimento.

Por fim, por ser um campo relativamente novo da matemática, a Geometria Fractal ainda apresenta diversas áreas de estudo inexploradas e áreas carentes de pesquisadores. Portanto, é esperada uma continuação nessa linha de estudos e desenvolvimento de novas pesquisas em áreas correlatas.

Referências

[1] MARTINS, J. M. dos S. *Sistemas de Lindenmayer*: Modelação de árvores com recurso ao Maple. Dissertação de mestrado — Faculdade de Ciências da Universidade do Porto, Março 2008.

[2] MANDELBROT, B. *The fractalist: Memoir of a scientific maverick*. [S.l.]: Vintage, 2012.

[3] FALCONER, K. *Fractal Geometry: Mathematical Foundations and Applications*. [S.l.]: Wiley, 2007.

[4] PRUSINKIEWICZ, P.; LINDENMAYER, A. *The algorithmic beauty of plants*. [S.l.]: Springer Science & Business Media, 2012.

[5] MONNEROT-DUMAINE, A. The Fibonacci Word fractal. (hal-00367972). 2009. Disponível em: <https://hal.archives-ouvertes.fr/hal-00367972>.

[6] SMITH, G. *Fractal Geometry: History and Theory*. 2011.

[7] BROGNA, V. D. A. *Palavras de Fibonacci e seus Fractais*. Dissertação de mestrado — UNESP-Universidade Estadual Paulista "Júlio de Mesquita Filho", Setembro 2022. Disponível em: <http://hdl.handle.net/11449/236826>.

[8] BIBAN, G. M.; MELO, T. de. *L-system*. 2025. Acesso em: 18 fev. 2025. Disponível em: <https://github.com/tmelorc/L-system>.