Universidade Federal de Uberlândia
Faculdade de Matemática

Coordenação do Curso de Matemática

DESENVOLVIMENTO DE UMA APLICAÇÃO WEB PARA CALCULAR O POLINÔMIO DE ALEXANDER

Victor Patrick Sena Barbosa Lima

Uberlândia-MG
2023

Victor Patrick Sena Barbosa Lima

DESENVOLVIMENTO DE UMA APLICAÇÃO WEB PARA CALCULAR O POLINÔMIO DE ALEXANDER

Monografia apresentada ao Curso de Graduação em Matemática da Universidade Federal de Uberlândia, como parte dos requisitos para a obtenção de título de **LICENCIATURA EM MATEMÁTICA**.

Área de concentração: Matemática
Linha de pesquisa: Geometria e Topologia

Orientadora: Taciana Oliveira Souza

Uberlândia-MG
2023

Agradecimentos

Agradeço, em primeiro lugar, a Deus por ter me auxiliado a concluir esta prolongada etapa, a qual frequentemente preenchia minha mente com sonhos e descobertas em um vasto mundo, no qual a matemática figura como um dos pilares. Esta etapa contou com a notável orientação da professora Taciana, que me prestou auxílio e o suporte necessário, acompanhando-me ao longo de quase dois anos.

Expresso minha gratidão aos meus pais, que mesmo à distância, mantiveram-se presentes através de ligações diárias, motivando-me a prosseguir, com minha mãe sempre orando por pedidos que fiz ao longo desses anos, e meu pai com seu jeito mais durão, mas que sempre me motivou a seguir em frente com mensagens de esperança e encorajamento. Agradeço também às minhas irmãs, Ana Caroline, por toda a mensagem de motivação e apoio oferecido durante esses anos, com noites que passávamos a conversar contando os segredos que a faculdade me reservava, Ana Paula, pois mesmo um pouco distante, me apoiou com sua força e luta, perseverando com o advento de dificuldades, e Ana Flávia, em especial, pois em 2019, passei a residir com ela, e ao longo do tempo, experimentamos altos e baixos, aprendendo a conviver com nossas diferenças e aceitá-las. Hoje, sinto uma imensa gratidão por toda a ajuda que ela me prestou e pelas experiências que adquiri. Mesmo agora, à distância, ela continua a me apoiar e a ajudar.

Agradeço a UFU, pois toda a rede de amigos e colegas, fiz por meio dela, amigos da matemática que marcaram minha vida e continuarão a fazer parte dela. Amigos que pude conversar quando a solidão se fazia presente, que me levaram a lugares e contaram segredos que lembrarei para o resto da minha estadia nessa terra. E mesmo passando a metade do curso em casa, por conta da SARS-COVID-19 pude fazer laços profundos com pessoas que vim conhecer na faculdade, em especial, Mateus Fernando, Marcos Rodrigues, Gabriel Mucci, Luis Felipe, Victor Cruz, Jackson Jonson, Jéssica Plifinar, uma pessoa que levarei para o resto da minha vida, e que veio fazer parte dela, no meio do ano de 2023.

Agradeço a todos que contribuíram para a minha formação e que de alguma forma fizeram parte dessa minha jornada. Jornada que está apenas no começo, e ainda muitas coisas viram, novos desafios, lutas e descobertas. Simplesmente, agradeço.

Resumo

Com o advento da era moderna, observamos um aumento significativo na velocidade com que resolvemos problemas que, no passado, demandavam muito tempo. Esse cenário de transição para a era digital trouxe consigo a adaptação de diversos métodos matemáticos, que foram transformados em algoritmos computacionais, com o propósito de tornar sua aplicação mais acessível e eficaz.

Neste trabalho, examinaremos dois invariantes da Teoria dos Nós Clássica, quais sejam, o *Determinante do nó* e o *Polinômio de Alexander do nó*. Estes, por serem obtidos de forma algorítmica, foram incorporados ao ambiente digital com o intuito de simplificar cálculos complexos e disponibilizar ao mundo um algoritmo de uso amigável.

O resultado principal deste trabalho consiste no desenvolvimento de uma aplicação web que calcula os invariantes *Determinante* e *Polinômio de Alexander*. Esta aplicação foi construída com PyScript, o qual consiste em uma estrutura web de código aberto que permite criar aplicações web front-end utilizando as linguagens de programação Python e JavaScript.

Palavras-chave: Teoria dos Nós, Determinante do Nó, Polinômio de Alexander do Nó, PyScript, Aplicação Web.

Abstract

With the advent of the modern age we have seen a significant increase in the speed with which we solve problems that used to take a long time. This transition to the digital age has brought with it the adaptation of various mathematical methods, which have been transformed into computer algorithms with the aim of making their application more accessible and effective.

In this work we are going examine two invariants of Classical Knot Theory namely the *Determinant of the knot* and the *Alexander Polynomial of the knot*. Since these are obtained algorithmically they have been incorporated into the digital environment in order to simplify complex calculations and make a user-friendly algorithm available to the world.

The main result of this work is the development of a web application that calculates the invariants *Determinant* and *Alexander Polynomial*. This application was made with PyScript which is an open-source web framework that allows you to create front-end web applications using the Python and JavaScript programming languages.

Keywords: Knot Theory, Determinant of a Knot, Alexander Polynomial of a Knot, PyScript, Web Application.

Sumário

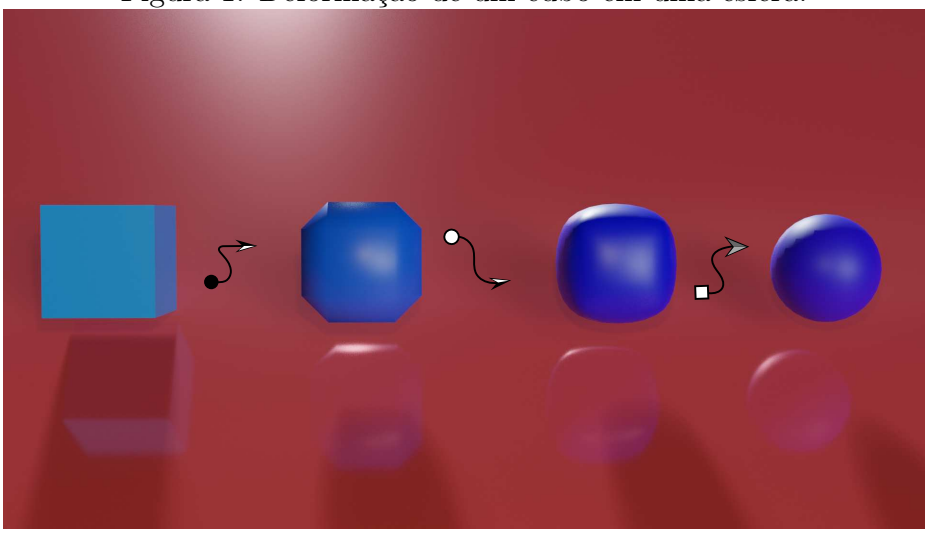
Lista de Figuras

Introdução

A Topologia é um dos mais importantes campos da Matemática. Essa área estuda propriedades de objetos geométricos que permanecem inalteradas quando são aplicados movimentos contínuos no próprio objeto ou no ambiente no qual ele está contido.

Como ilustração do que falamos, podemos observar um cubo e uma esfera, veja a Figura 1. Para a Topologia esses são objetos idênticos, pois um pode ser obtido a partir do outro por meio de uma deformação contínua.

Figura 1: Deformação de um cubo em uma esfera.

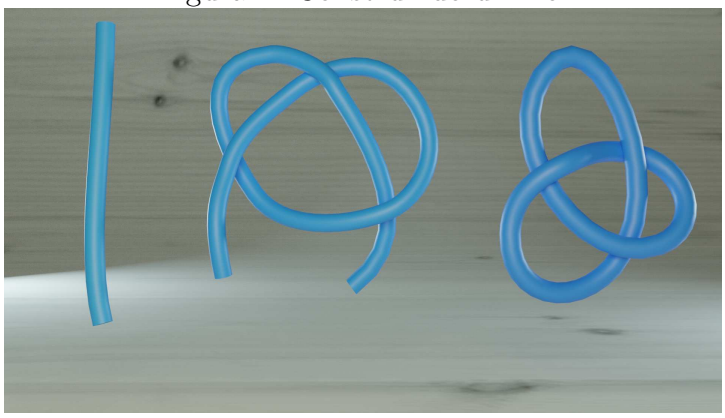


Fonte: Elaboração do autor.

Dentro da grande área da Topologia existe uma subárea chamada Teoria dos Nós. Essa teoria (a clássica) pode ser compreendida como o estudo das curvas fechadas, sem autointerseções, em três dimensões, as quais chamamos de *nós*. De modo intuitivo, um nó é construído torcendo e entrelaçando um fio e unindo suas extremidades, como mostra a Figura 2.

O início do conceito matemático de nó aparece nos anos de 1830 com pesquisas de Carl Friedrich Gauss (1777-1855), [3]. Seu interesse, na época, era aplicar esse conceito na área de eletrodinâmica. Nesse mesmo século, décadas depois, outro grande pesquisador, William Thomson (Lord Kelvin) (1824-1907), se interessou pelo assunto, pois acreditava que os nós seriam a chave para a compreensão das substâncias químicas que, de acordo com sua crença, seriam descritas pelas formas dos nós. Apesar de tal crença não condizer com a verdade, a Teoria dos Nós continuou a desenvolver-se e aplicações foram sendo descobertas em outras áreas como: na Física, no estudo do DNA na Biologia Molecular e no estudo de estruturas

Figura 2: Construindo um nó.

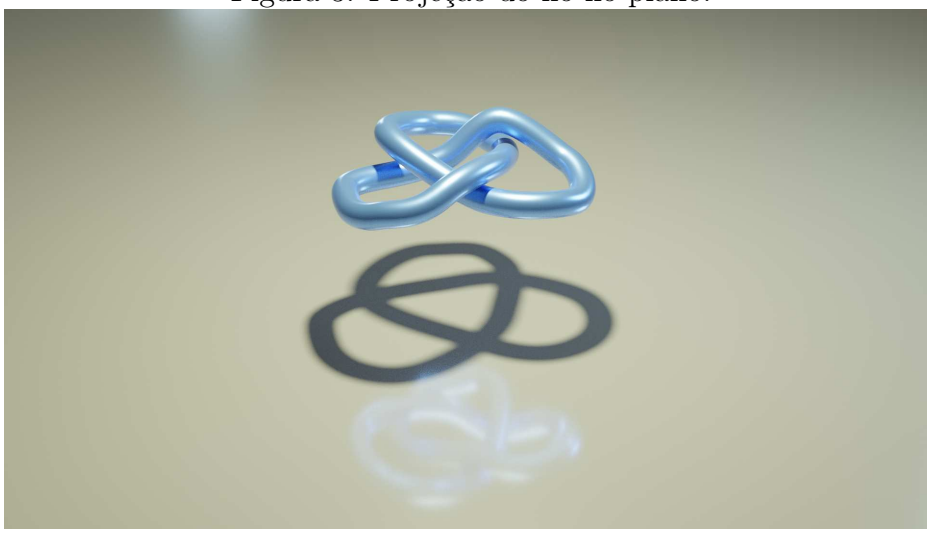


Fonte: Elaboração do autor.

tridimensionais de moléculas na Química. Fato que merece destaque ocorreu em 1984, quando um grande avanço do matemático Vaughan Jones (1952-2020) nessa área levou o matemático e físico-teórico Edward Witten a descobrir uma ligação entre a Teoria dos Nós e a Teoria do Campo Quântico. Por essa descoberta Jones e Witten receberam a Medalha Fields, veja [10].

Para estudarmos um nó, tendo em vista a dificuldade de desenharmos objetos tridimensionais, fixamos um plano do espaço e utilizamos sua projeção nesse plano. Sempre é possível considerar projeções que possuam apenas cruzamentos duplos, como ilustra a Figura 3.

Figura 3: Projeção do nó no plano.



Fonte: Elaboração do autor.

Nessa projeção, desenhamos pequenas interrupções próximas a cada cruzamento para indicarmos que um arco passa por cima do outro. Assim, obtemos o *diagrama do nó*, como ilustra a Figura 4.

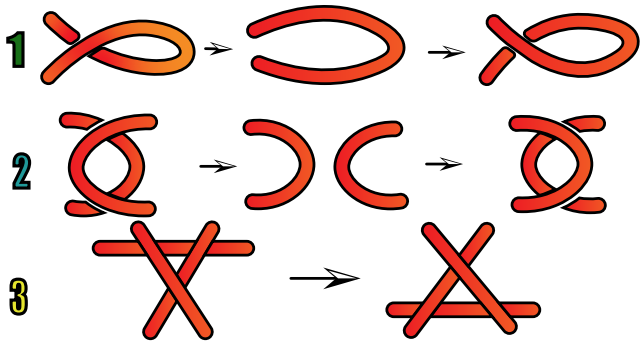
Para a Teoria dos Nós, quando um nó pode ser transformado continuamente em outro que, a princípio, aparenta ser diferente, diz-se que os dois nós são equivalentes ou têm mesmo tipo. Mas, se para registrarmos um nó utilizarmos um diagrama (uma projeção), é necessário investigar se dois diagramas que, a princípio, são aparentemente diferentes, podem representar o mesmo nó. Esse problema foi resolvido por Kurt Reidemeister (1883-1971) nos anos 1920.

Figura 4: Diagrama de um nó.

Fonte: Elaboração do autor.

Ele propôs que se considerasse três movimentos que poderiam ser realizados em diagramas de nós. Hoje esses movimentos são chamados *movimentos de Reidemeister*. Eles estão ilustrados na Figura 5.

Figura 5: Os movimentos de Reidemeister.



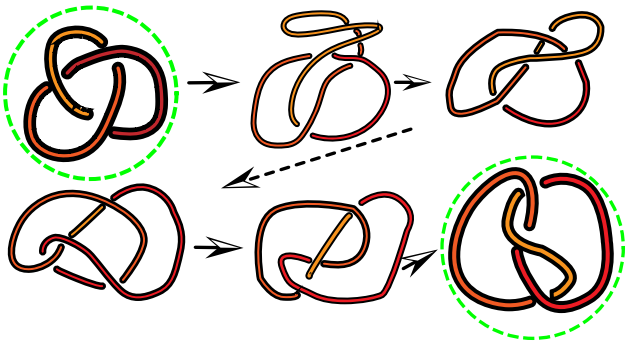
Fonte: Elaboração do autor.

Quando um diagrama de um nó pode ser transformado em outro diagrama mediante aplicação de uma sequência finita de movimentos de Reidemeister, diz-se que esses diagramas são equivalentes. Com essa consideração, Reidemeister provou que "dois nós são equivalentes se, e somente se, os respectivos diagramas são equivalentes". Deste modo, pode-se estudar equivalência dos nós estudando equivalência dos diagramas dos nós, como ilustrado na Figura 6.

Apesar do excepcional resultado de Reidemeister, é geralmente difícil determinar quando um nó é equivalente a outro. Uma forma de transpor essa dificuldade é associar ao diagrama de um nó um objeto algébrico ou aritmético que permaneça o mesmo quando se efetue uma sequência finita de movimentos de Reidemeister. Assim, esse objeto pode auxiliar na detecção de nós que não são equivalentes. Chamamos tais objetos matemáticos de invariantes do nó.

Dois importantes invariantes do nó são o *Determinante* e o *Polinômio de Alexander*, os quais são obtidos por processos algorítmicos. Neste trabalho, apresentamos uma aplicação Web que calcula o Determinante e o Polinômio de Alexander do nó, a qual foi construída com o PyScript por meio da adaptação da rotina Python desenvolvida em [7]. O PyScript é uma estrutura web de código aberto que permite criar aplicações web front-end utilizando as linguagens de

Figura 6: Aplicando os movimentos de Reidemeister.



Fonte: Elaboração do autor.

programação Python e JavaScript. Com sua estrutura, pode-se incorporar um código Python em HTML e este será executado no navegador - sem executar o Python no backend.

A estrutura desta obra pode ser assim entendida: no capítulo um, são apresentados conceitos básicos sobre espaços topológicos, homeomorfismos e homotopias. Estes são pré-requisitos necessários para a compreensão da Teoria dos Nós.

No capítulo dois, apresentamos uma breve introdução à Teoria dos Nós, com ênfase na construção dos invariantes Determinante e Polinômio de Alexander. Já no capítulo três, introduzimos a construção da aplicação Web falando das linguagens de programação usadas e o funcionamento do aplicativo.

1. PRÉ-REQUISITOS

O objetivo deste capítulo é apresentar resultados básicos de Topologia Geral e Teoria de Homotopia, necessários para a compreensão da Teoria dos Nós.

1.1 Topologia Geral

As definições e conceitos apresentados nesta seção estão baseados nas referências [4], [8] e [13].

1.1.1 Espaços Topológicos

Definição 1. *Um espaço topológico é um par (X, τ), onde X é um conjunto não vazio e τ é uma coleção de subconjuntos de X satisfazendo:*

1. $X, \emptyset \in \tau$.
2. *τ é fechado em relação à união arbitrária, isto é, dada uma família $\{A_\alpha\}_{\alpha \in L}$ de elementos de τ, onde L é o conjunto de índices, então*

$$\bigcup_{\alpha \in L} A_\alpha \in \tau.$$

3. *τ é fechado em relação à interseção finita, isto é, se A_1, A_2, ..., A_n são elementos de τ, então*

$$\bigcap_{i=1}^{n} A_i \in \tau.$$

Nesse caso, τ é chamada uma topologia em X.
Os elementos de τ são chamados τ-abertos de X ou, simplesmente, abertos de X.

Exemplo 1. Qualquer conjunto não-vazio X torna-se um espaço topológico de duas maneiras essencialmente triviais:

1. Os únicos conjuntos abertos de X são $\emptyset$ e o propio X, ou seja, $\tau = \{\emptyset, X\}$. Essa topologia é chamada topologia trivial e será denotada por $\boldsymbol{\tau_{trivial}}$.
2. Os conjuntos abertos de X são todos os subconjuntos de X, isto é, τ é o conjunto das partes de X. Essa topologia é chamada topologia discreta e será denotada por $\boldsymbol{\tau_{discreta}}$.

Exemplo 2. Seja $X = \{a, b\}$. Então as possíveis topologias em X estão listadas a seguir:

1. $\tau_1 = \{\emptyset, \{a, b\}\}$ é a topologia trivial.
2. $\tau_2 = \{\emptyset, \{a, b\}, \{a\}, \{b\}\}$ é a topologia discreta.
3. $\tau_3 = \{\emptyset, \{a, b\}, \{a\}\}$.
4. $\tau_4 = \{\emptyset, \{a, b\}, \{b\}\}$.

Exemplo 3. Sejam $X = \mathbb{R}$ e $\tau = \{\emptyset,\ A \subset \mathbb{R}\}$, onde $A \in \tau$ se, e somente se, para todo $x \in A$ existe um intervalo aberto (a, b) tal que $x \in (a, b) \subset A$.

1. Claramente $\emptyset, \mathbb{R} \in \tau$.
2. Seja $\{A_\alpha\}_{\alpha \in L}$ uma família de elementos de τ, então

$$\bigcup_{\alpha \in L} A_\alpha \in \tau.$$

   De fato, seja $x \in \bigcup_{\alpha \in L} A_\alpha$, então existe $\alpha_0 \in L$ tal que $x \in A_{\alpha_0} \in \tau$. Logo, exite um intervalo (a, b) tal que

$$x \in (a, b) \subset A_{\alpha_0} \subset \bigcup_{\alpha \in L} A_\alpha.$$

3. Sejam $B_1,\ B_2 \in \tau$. Dado $x \in B_1 \cap B_2$, existem intervalos (a_1, b_1) e (a_2, b_2) tais que $x \in (a_1, b_1) \subset B_1$ e $x \in (a_2, b_2) \subset B_2$. Denotando $a = max\{a_1, a_2\}$ (o máximo entre a_1 e a_2) e $b = min\{b_1, b_2\}$ (o mínimo entre b_1 e b_2), verifica-se $x \in (a, b) \subset B_1 \cap B_2$. Logo, $B_1 \cap B_2 \in \tau$.

   Assim, de modo indutivo, se $B_1,\ B_2, \ldots, B_n \in \tau$, então $\bigcap_{i=1}^{n} B_i \in \tau$.

Generaliza-se essa definição para $\mathbb{R}^n$, $n \geq 2$, de modo natural: $\tau = \{\emptyset, A \subset \mathbb{R}^n\}$, onde $A \in \tau$ se, e somente se, para todo $(x_1, x_2, x_3, \ldots, x_n) \in A$, existem intervalos abertos $(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)$, tais que:

$$(x_1, x_2, x_3, \ldots, x_n) \in (a_1, b_1) \times (a_2, b_2) \times (a_3, b_3) \times \cdots \times (a_n, b_n) \subset A.$$

Essa topologia sobre $\mathbb{R}^n$, $n \geq 1$, é chamada *topologia euclidiana ou usual* e será denotada por $\boldsymbol{\tau_{usual}}$.

1.2 Conjuntos fechados

Um conjunto F de um espaço topológico X diz-se *fechado* quando o seu complementar $X - F$ é aberto. Mais precisamente:

Definição 2. *Dados (X, τ) um espaço topológico e $F \subset X$. F é chamado fechado em X, ou τ- fechado, se o seu complementar $X - F$ é τ-aberto, isto é, se $X - F \in \tau$.*

Observação 1. *Verifica-se facilmente que os subconjuntos fechados de um espaço topológico (X, τ) possuem as seguintes propriedades:*

1. *$\emptyset$ e X são fechados em X.*
2. *Sejam $F_1, F_2 \ldots, F_n$ subconjuntos fechados em X, então*

$$\bigcup_{i=1}^{n} F_i$$

   é fechado em X.

3. *Sejam $\{F_\alpha\}_{\alpha \in L}$ uma família de subconjuntos fechados em X, então*

$$\bigcap_{\alpha \in L} F_\alpha$$

   é fechado em X.

Exemplo 4. Em $(\mathbb{R}, \tau_{usual})$ todo conjunto finito é fechado. De fato, dado $x \in \mathbb{R}$ então $\{x\}$ é fechado em $\mathbb{R}$ pois seu complementar

$$\mathbb{R} - \{x\} = (-\infty, x) \cup (x, \infty)$$

é aberto. Logo, se $F = \{x_1, x_2, \cdots, x_n\}$ é um subconjunto finito de $\mathbb{R}$, então pelo segundo item da Observação 1

$$F = \bigcup_{i=1}^{n} \{x_i\}$$

é um subconjunto fechado de $\mathbb{R}$.

1.3 Bases

Por vezes, para introduzir uma topologia em um conjunto não é necessário descrever todos os conjuntos abertos, mas apenas alguns conjuntos "mais simples", os quais são chamados *abertos básicos* da topologia.

Definição 3. *Seja (X, τ) um espaço topológico. Uma base para τ é uma coleção $\mathfrak{B} \subset \tau$ tal que, dado $A \in \tau$, A é uma união (arbitrária) de elementos de $\mathfrak{B}$.*

Exemplo 5.

1. Uma topologia é base de si mesma.
2. $\mathfrak{B} = \{X\}$ é uma base para $\tau_{trivial}$.
3. $\mathfrak{B} = \{\{x\} | x \in X\}$ é uma base para $\tau_{discreta}$.

1.4 Pré-bases

Definição 4. *Seja X um conjunto. Uma coleção $\mathfrak{A}$ de subconjuntos de X é uma pré-base sobre X se satisfaz:*

i) todo ponto $x \in X$ pertence a pelo menos um conjunto da coleção $\mathfrak{A}$.

ii) Se $U, V \in \mathfrak{A}$ e $x \in U \cap V$, então existe $W_x \in \mathfrak{A}$ tal que $x \in W_x \subset U \cap V$. Ou seja, $U \cap V = \bigcup_{x \in U \cap V} W_x$, com $W_x \in \mathfrak{A}$.

Exemplo 6.

1. Toda topologia é pré-base de si mesma.

2. Seja $X = \mathbb{R}$. Então:
$$\mathfrak{A} = \{(a, b) | a,\ b \in \mathbb{R}; a < b\}$$
é uma pré-base para a topologia usual de $\mathbb{R}$.

De fato:

i) Para todo $x \in \mathbb{R}$, $x \in (x - 1, x + 1) \in \mathfrak{B}$.

ii) Para todo $x \in \mathbb{R}$ tal que $x \in (a_1, b_1) \cap (a_2, b_2)$, tem-se:

$$x \in (a, b) \subset (a_1, b_1) \cap (a_2, b_2),$$

onde $a = max\{a_1, a_2\}$ e $b = min\{b_1, b_2\}$.

Teorema 1. *Sejam X um conjunto e $\mathfrak{A}$ uma pré-base sobre X. Seja Φ a coleção de todas as possíveis uniões de elementos de $\mathfrak{A}$. Então, Φ é uma topologia em X e $\mathfrak{A}$ é uma base para Φ.*

Demonstração. Considerando $\emptyset$ como uma união vazia, tem-se $\emptyset \in \Phi$.

Para cada $x \in X$, existe $U_x \in \mathfrak{A}$ tal que $x \in U_x$. Assim,

$$X = \bigcup_{x \in X} U_x \in \Phi.$$

Dada uma família $\{A_i\}_{i \in L}$ de elementos de Φ, onde L é o conjunto de índices, então cada A_i é uma união de elementos de $\mathfrak{A}$ e, portanto, $\bigcup_{i \in L} A_i \in \Phi$.

Por fim, basta mostrar que se $A, B \in \Phi$, então $A \cap B \in \Phi$. Sendo A e B elementos de Φ tem-se $A = \bigcup_{\alpha} U_\alpha$ e $B = \bigcup_{\beta} V_\beta$, com $U_\alpha, V_\beta \in \mathfrak{A}$, para todos os índices α e β. Assim,

$$A \cap B = \bigcup_{\alpha} \bigcup_{\beta} (U_\alpha \cap V_\beta).$$

Desde que $\mathfrak{A}$ é uma pré-base, tem-se $U_\alpha \cap V_\beta = \bigcup_{x \in U_\alpha \cap V_\beta} W_x$, com $W_x \in \mathfrak{A}$. Logo, $A \cap B$ é uma união de elementos de $\mathfrak{A}$ e, portanto, $A \cap B \in \Phi$. □

1.5 Topologia Relativa e Topologia Produto

Duas questões surgem naturalmente das definições apresentadas até então:

Fixada uma topologia num conjunto, um subconjunto não vazio herda de alguma forma esta estrutura?

Dados dois espaços topológicos, o produto cartesiano deles possui uma estrutura topológica herdada de suas topologias?

Estas questões têm respostas afirmativas, como veremos a seguir.

Definição 5. *Sejam (X, τ) um espaço topológico e $\emptyset \neq Y \subset X$. Então:*

1. *O conjunto*
$$\tau_Y = \{B \cap Y | B \in \tau\},$$
é uma topologia sobre Y chamada topologia relativa em Y.

2. *O par (Y, τ_Y) é dito subespaço topológico de (X, τ).*

3. *Os elementos de τ_Y são ditos abertos relativos.*

Observação 2. *Em geral, os abertos relativos não são abertos no espaço total, como ilustra o exemplo a seguir.*

Exemplo 7. Seja $\mathbb{R}$ com a topologia usual e consideremos $\mathbb{Q} \subset \mathbb{R}$ com a topologia relativa, então $A = \{x \in \mathbb{Q} | 0 < x < 1\}$ é aberto relativo, mas não é aberto em $\mathbb{R}$.

Proposição 1. *Sejam (X, τ) um espaço topológico e $\emptyset \neq Y \subset X$. Então F é τ_Y-fechado se, e somente se, $F = W \cap Y$, onde W é τ-fechado.*

Demonstração. Seja F τ_Y-fechado, isto é, $Y - F$ é τ_Y-aberto. Logo, $Y - F = A \cap Y$, sendo A um τ-aberto. Seja $W = X - A$, o qual é τ-fechado. Então, $W \cap Y = F$.

Reciprocamente, se $F = W \cap Y$, onde W é τ-fechado, então $X - W$ é τ-aberto e, portanto, $Y - F = Y - (W \cap Y) = (X - W) \cap Y$ é τ_Y-aberto. Logo, F é τ_Y-fechado. □

Sejam (X, τ) e (Y, τ') espaços topológicos. Considere a coleção $\mathfrak{P} = \{U \times V \,|\, U \in \tau, V \in \tau'\}$ de subconjuntos do produto cartesiano $X \times Y$. Note que $\mathfrak{P}$ satisfaz:

i) $X \times Y \in \mathfrak{P}$.

ii) Se $U_1 \times V_1, U_2 \times V_2 \in \mathfrak{P}$, então $(U_1 \times V_1) \cap (U_2 \times V_2) = (U_1 \cap U_2) \times (V_1 \cap V_2) \in \mathfrak{P}$.

Com isso, verifica-se que $\mathfrak{P}$ é uma pré-base sobre $X \times Y$. Segue, do Teorema 1, que $\mathfrak{P}$ gera uma topologia em $X \times Y$.

Definição 6. *A topologia em $X \times Y$ gerada por $\mathfrak{P}$ é chamada topologia produto.*

1.6 Funções contínuas

O objetivo desta seção é estabelecer o conceito de função contínua entre espaços topológicos.

Definição 7. *Dados dois espaços topológicos (X, τ) e (Y, τ'), dizemos que uma função $f : (X, \tau) \to (Y, \tau')$ é contínua com relação às topologias τ e τ'(ou que f é $\tau - \tau'$ contínua) se, para cada conjunto τ'-aberto $V \subset Y$, o conjunto $f^{-1}(V) \subset X$ é τ- aberto.*

Exemplo 8.

1. Toda função constante é contínua. De fato, seja $f : (X, \tau) \to (Y, \tau')$ tal que $f(x) = y_0$ para todo $x \in X$ e seja $V \subset Y$ aberto. Tem-se

$$f^{-1}(V) = \begin{cases} X \text{ se } y_0 \in V, \\ \emptyset \text{ se } y_0 \notin V. \end{cases}$$

   Em ambos os casos $f^{-1}(V)$ é aberto, logo f é contínua.

2. Sejam (X, τ) um espaço topológico qualquer e $(Y, \tau_{trivial})$. Toda função:

$$f : (X, \tau) \to (Y, \tau_{trivial})$$

   é contínua.

3. Sejam $(X, \tau_{discreta})$ e (Y, τ) um espaço topológico qualquer. Toda função

$$f : (X, \tau_{discreta}) \to (Y, \tau)$$

   é contínua.

1.6.1 Alguns teoremas básicos sobre Continuidade

Apresentaremos alguns teoremas importantes acerca de funções contínuas.

Teorema 2. Sejam $f : (X, \tau) \to (Y, \tau')$ *e* $g : (Y, \tau') \to (Z, \tau'')$ *funções contínuas. Então,* $g \circ f : (X, \tau) \to (Z, \tau'')$ *é* $\tau - \tau''$ *contínua.*

Demonstração. Seja $W \subset Z$ um τ''-aberto. Segue, da continuidade da função $g : (Y, \tau') \to (Z, \tau'')$, que $g^{-1}(W) \in \tau'$ e como f é $\tau - \tau'$ contínua temos

$$(g \circ f)^{-1}(W) = (f^{-1} \circ g^{-1})(W) = f^{-1}(g^{-1}(W)) \in \tau.$$

Portanto, $g \circ f$ é $\tau - \tau''$ contínua. □

Teorema 3. *Uma função $f : (X, \tau) \to (Y, \tau')$ é $\tau - \tau'$ contínua se, e somente se, para cada $x \in X$ e para cada conjunto aberto $V_{f(x)} \in \tau'$ contendo $f(x)$, existe um conjunto aberto $U_x \in \tau$ contendo x, tal que $f(U_x) \subset V_{f(x)}$.*

Demonstração. Seja f $\tau - \tau'$ contínua. Então, dado $x \in X$ e um aberto $V_{f(x)} \in \tau'$ contendo $f(x)$, desde que f é $\tau - \tau'$ contínua, temos que $f^{-1}(V_{f(x)}) \in \tau$ é um aberto contendo x e, além disso, $f(f^{-1}(V_{f(x)})) \subset V_{f(x)}$. Assim, basta tomar $U_x = f^{-1}(V_{f(x)})$.

Reciprocamente, seja $V \in \tau'$. Se $f^{-1}(V) = \emptyset$, então é um conjunto $\tau-$ aberto.

Suponhamos $f^{-1}(V) \neq \emptyset$. Assim, por hipótese, para cada $x \in f^{-1}(V)$ existe $U_x \in \tau$, de modo que $x \in U_x$ e $f(U_x) \subset V$. Logo,

$$f^{-1}(V) \subset \bigcup_{x \in f^{-1}(V)} U_x.$$

Por outro lado, $U_x \subset f^{-1}(f(U_x)) \subset f^{-1}(V)$ para todo $x \in f^{-1}(V)$ e, portanto,

$$\bigcup_{x \in f^{-1}(V)} U_x \subset f^{-1}(V).$$

Assim, $f^{-1}(V) = \bigcup_{x \in f^{-1}(V)} U_x \in \tau.$ □

Definição 8. *Uma função $f : (X, \tau) \to (Y, \tau')$ é $\tau - \tau'$ contínua no ponto $x_0 \in X$ se dado um aberto $V \in \tau'$ contendo $f(x_0)$ existe um aberto $U \in \tau$ contendo x_0 tal que $f(U) \subset V$.*

Observação 3. *O Teorema 3 garante que $f : (X, \tau) \to (Y, \tau')$ é $\tau - \tau'$ contínua se, e somente se, f é contínua em cada ponto $x_0 \in X$.*

Teorema 4. *Seja $f : (X, \tau) \to (Y, \tau')$ uma função entre espaços topológicos. São equivalentes:*

1. f é $\tau - \tau'$ contínua.
2. Se $B \subset Y$ é $\tau'-$fechado, então $f^{-1}(B) \subset X$ é $\tau-$fechado.

Demonstração. $1 \Rightarrow 2$: Seja B $\tau'-$fechado. Então $Y - B$ é $\tau'-$aberto. Assim, por hipótese, $X - f^{-1}(B) = f^{-1}(Y - B)$ é $\tau-$aberto e, portanto, $f^{-1}(B)$ é $\tau-$fechado.

$2 \Rightarrow 1$: Seja A $\tau'-$aberto. Então $Y - A$ é $\tau'-$fechado. Assim, por hipótese, $X - f^{-1}(A) = f^{-1}(Y - A)$ é $\tau-$fechado e, portanto, $f^{-1}(A)$ é $\tau-$aberto. □

Lema 1 (Lema da Colagem)**.** *Seja (X, τ) um espaço topológico, onde $X = A \cup B$ e A, B são ambos $\tau-$fechados. Dadas funções contínuas $f : (A, \tau_A) \to (Y, \tau')$ e $g : (B, \tau_B) \to (Y, \tau')$ tais que $f|_{A \cap B} = g|_{A \cap B}$, então a função $h : (X, \tau) \to (Y, \tau')$ definida por:*

$$h(x) = \begin{cases} f(x), \text{ se } x \in A \\ g(x), \text{ se } x \in B \end{cases}$$

é $\tau - \tau'$ contínua.

Demonstração: Seja C fechado em (Y, τ'). Então

$$h^{-1}(C) = f^{-1}(C) \cup g^{-1}(C).$$

Pelo Teorema 4, $f^{-1}(C)$ é τ_A-fechado e, como A é fechado em (X, τ), segue da Proposição 1 que $f^{-1}(C)$ é $\tau-$fechado. De maneira análoga, concluímos que $g^{-1}(C)$ é $\tau-$fechado. Portanto, $h^{-1}(C)$ é a união de dois fechados de (X, τ), logo é $\tau-$fechado.

Assim, pelo Teorema 4, h é $\tau - \tau'$ contínua. □

O resultado a seguir mostra que a continuidade de uma função pode ser verificada por meio de abertos básicos.

Teorema 5. *Seja $f : (X, \tau) \to (Y, \tau')$ uma função entre espaços topológicos, e seja $\mathfrak{B}'$ uma base de τ'. Então f é $\tau - \tau'$ contínua se, e somente se, para cada $U \in \mathfrak{B}'$, $f^{-1}(U)$ é $\tau-$aberto.*

Demonstração. Seja f $\tau - \tau'$ contínua. Assim, se $U \in \mathfrak{B}' \subset \tau'$, então é imediato que $f^{-1}(U) \in \tau$.

Reciprocamente, se A é um $\tau'-$aberto, então $A = \bigcup_{\alpha} U_\alpha$ é uma união de elementos de $\mathfrak{B}'$. Assim, $f^{-1}(A) = \bigcup_{\alpha} f^{-1}(U_\alpha) \in \tau$ desde que, por hipótese, $f^{-1}(U_\alpha) \in \tau$. □

1.7 Homeomorfismos

Na introdução deste texto falamos em deformações contínuas, a definição dessa noção intuitiva é a seguinte:

Definição 9. *Sejam (X, τ) e (Y, τ') dois espaços topológicos, uma bijeção $f : (X, \tau) \to (Y, \tau')$ é um homeomorfismo se:*

1. *f é $\tau - \tau'$ contínua.*
2. *$f^{-1} : (Y, \tau') \to (X, \tau)$ é $\tau' - \tau$ contínua.*

Neste caso, diz-se que X e Y são homeomorfos, e isso é denotado por $X \simeq Y$.

Observação 4. *Um homeomorfismo $f : X \to Y$ é uma deformação que transforma X em Y continuamente e que pode ser desfeita, pois existe $f^{-1} : Y \to X$ que transforma Y em X. Assim, as estruturas topológicas são preservadas e, portanto, do ponto de vista da topologia, dois conjuntos homeomorfos são indistinguíveis.*

Observação 5. *Desde que, pelo Teorema 2 , a composição de funções contínuas é uma função contínua e $(g \circ f)^{-1} = f^{-1} \circ g^{-1}$, temos que se $f : (X, \tau) \to (Y, \tau')$ e $g : (Y, \tau') \to (Z, \tau'')$ são homeomorfismos, então a composição*

$$g \circ f : (X, \tau) \to (Z, \tau'')$$

é um homeomorfismo.

Teorema 6. *Seja $f : (X, \tau) \to (Y, \tau')$ uma bijeção. São equivalentes:*

1. f é um homeomorfismo.
2. f é contínua e aplica conjuntos abertos em conjuntos abertos (neste caso, f é chamada uma aplicação aberta).

Demonstração. Temos que $f : (X, \tau) \to (Y, \tau')$ é um homeomorfismo se, e somente se, f é uma bijeção contínua tal que $f^{-1} : (Y, \tau') \to (X, \tau)$ é contínua, o que é equivalente a dizer que: para cada $U \in \tau$, $(f^{-1})^{-1}(U) = f(U) \in \tau'$, e isso ocorre se, e somente se, f é uma bijeção aberta. □

Observação 6. *Pelo Teorema 6, um homeomorfismo $f : (X, \tau) \to (Y, \tau')$ possui a seguinte propriedade: $U \in \tau$ se, e somente se, $f(U) \in \tau'$.*

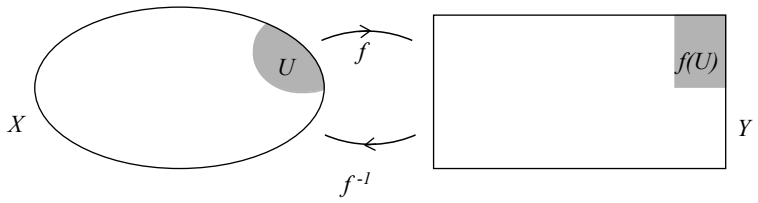


Figura 1.1: Um homeomorfismo leva abertos em abertos.

1.8 Homotopia

A referência desta seção é [5].

Doravante, I denotará o intervalo fechado $[0, 1]$. A topologia considerada em I será a induzida da topologia usual em $\mathbb{R}$.

Definição 10. *Sejam X e Y espaços topológicos, e sejam $f : X \to Y$ e $g : X \to Y$ funções contínuas. Dizemos que f é homotópica a g se existir uma função contínua $H : X \times I \to Y$ tal que, para todo $x \in X$ tem-se $H(x, 0) = f(x)$ e $H(x, 1) = g(x)$. A função H é chamada de homotopia entre f e g.*

A homotopia entre f e g será indicada por $f \overset{H}{\sim} g$.

Intuitivamente, podemos pensar na homotopia como um processo de deformação contínua da aplicação f na aplicação g. Em outras palavras, dada H uma homotopia entre f e g e

considerando, para cada $t \in I$, a aplicação contínua $H_t : X \to Y$, definida por $H_t(x) = H(x, t)$, temos que $(H_t)_{t\in I}$ define uma família de funções contínuas a um parâmetro. Em $t = 0$ temos f, para $t = 1$ temos g, e no intervalo $0 < t < 1$, as aplicações H_t fornecem as deformações intermediárias. A Figura 1.2 ilustra essa ideia intuitiva.

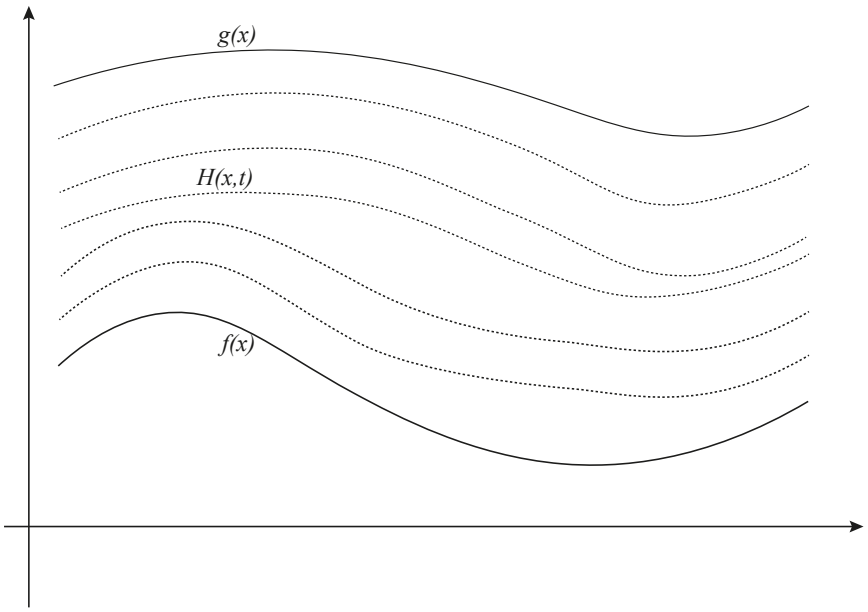


Figura 1.2: Homotopia entre f e g.

Exemplo 9. Seja X um espaço topológico qualquer. Então, quaisquer duas aplicações contínuas $f : X \to \mathbb{R}^n$ e $g : X \to \mathbb{R}^n$ são homotópicas. De fato, basta definir $H : X \times I \to \mathbb{R}^n$ por:

$$H(x, t) = (1 - t)f(x) + tg(x).$$

Exemplo 10. Considere $f : I \to \mathbb{R}^2$, definida por $f(x) = (cos(2\pi x), sen(2\pi x))$ e $g : I \to \mathbb{R}^2$ constante, $g(x) = (0, 0)$ para todo $x \in I$. De acordo com o Exemplo 9, $f \overset{H}{\sim} g$ onde $H(x, t) = (1 - t)(cos(2\pi x), sen(2\pi x))$, para $(x, t) \in I \times I$. Então, para cada t fixado temos $H_t(x) = (1 - t)(cos(2\pi x), sen(2\pi x))$, com x variando em I. Portanto, a família de funções contínuas $(H_t)_{t\in I}$, neste caso, corresponde a uma família de circunferências centradas na origem e de raio $1 - t$, para $0 \leq t \leq 1$.

Teorema 7. *A relação de homotopia é uma relação de equivalência*

Demonstração. Sejam $f, g, h : X \to Y$ funções contínuas.

1. $f \sim f$ (reflexividade).

   Defina $H : X \times I \to Y$ por $H(x, t) = f(x)$, para todo $x \in X$ e para todo $t \in I$. É imediato que $f \overset{H}{\sim} f$.

2. $f \sim g \Rightarrow g \sim f$ (simetria).

   Por hipótese existe $H : X \times I \to Y$ função contínua tal que $H(x, 0) = f(x)$ e $H(x, 1) = g(x)$, para todo $x \in X$. Sendo assim, defina $K : X \times I \to Y$ por $K(x, t) = H(x, 1 - t)$.

Claramente K é contínua e $K(x,\,0) = H(x,\,1) = g(x)$ e $K(x,\,1) = H(x,\,0) = f(x)$, para todo $x \in X$. Logo, $g \overset{K}{\sim} f$.

3. $f \sim g$ e $g \sim h \Rightarrow f \sim h$ (transitividade).

   Por hipótese, existem funções contínuas $H : X \times I \to Y$, tal que $H(x,\,0) = f(x)$ e $H(x,\,1) = g(x)$, e $K : X \times I \to Y$, tal que $K(x,\,0) = g(x)$ e $K(x,\,1) = h(x)$. Definindo $H' : X \times I \to Y$ por

$$H'(x,t) = \begin{cases} H(x,\,2t), & 0 \leq t \leq \frac{1}{2} \\ K(x,\,2t-1), & \frac{1}{2} \leq 0 \leq 1. \end{cases}$$

   Pelo Lema da Colagem, H' é contínua. Além disso, $H'(x,\,0) = H(x,\,0) = f(x)$ e $H'(x,\,1) = K(x,\,2-1) = K(x,\,1) = h(x)$ para todo $x \in X$. Assim, $f \overset{H'}{\sim} h$. □

Proposição 2. *Sejam $f_1,\, f_2 : X \to Y$ e $g_1, g_2 : Y \to Z$ funções contínuas. Se $f_1 \sim f_2$ e $g_1 \sim g_2$, então $g_1 \circ f_1 \sim g_2 \circ f_2$.*

Demonstração. Por hipóteses, existem homotopias $H_1 : X \times I \to Y$, tal que $H_1(x,\,0) = f_1(x)$ e $H_1(x,\,1) = f_2(x)$, e $H_2 : Y \times I \to Z$, tal que $H_2(y,\,0) = g_1(y)$ e $H_2(y,\,1) = g_2(y)$. Seja $K : X \times I \to Z$ definida por $K(x,\,t) = H_2(H_1(x,\,t),\,t)$. Note que K é contínua e satisfaz $K(x,\,0) = H_2(H_1(x,\,0),\,0) = H_2(f_1(x),\,0) = g_1(f_1(x)) = (g_1 \circ f_1)(x)$ e $K(x,\,1) = H_2(H_1(x,\,1),\,1) = H_2(f_2(x),\,1) = g_2(f_2(x)) = (g_2 \circ f_2)(x)$. Portanto, $g_1 \circ f_1 \overset{K}{\sim} g_2 \circ f_2$. □

2. TEORIA DOS NÓS CLÁSSICA

Neste capítulo, apresentaremos uma breve introdução à Teoria dos Nós clássica. Para isso, devemos esclarecer o que chamamos de *nó clássico.*

Primeiramente, qualquer espaço topológico homeomorfo à circunferência unitária $\{(x, y) \in \mathbb{R}^2 \,|\, x^2 + y^2 = 1\}$ será chamado de esfera unidimensional, e será denotado por S^1.

O que chamamos de nó clássico em $\mathbb{R}^3$ é a imagem de uma aplicação contínua injetiva $f : S^1 \to \mathbb{R}^3$ (chamamos isso de mergulho). Por padrão, identificamos um nó com a imagem de uma aplicação contínua injetiva e o denotamos $K = f(S^1)$. Assim quando consideramos, por exemplo, dois nós, estamos considerando dois mergulhos $f(S^1) \subset \mathbb{R}^3$ e $g(S^1) \subset \mathbb{R}^3$. Então, a diferença entre os diversos nós não está no espaço topológico que é a imagem do mergulho, mas sim no espaço que circunda a imagem do mergulho, isto é, no complementar $\mathbb{R}^3 - f(S^1)$. Nossa noção intuitiva de deformação de um nó K_1 em outro nó K_2 é, na verdade, a noção de um movimento contínuo no espaço ambiente $\mathbb{R}^3$ que leva continuamente o subespaço topológico K_1 sobre o subespaço topológico K_2. Essa noção é associada à definição de isotopia ambiente.

Definição 11. *Seja X um espaço topológico. Uma isotopia ambiente H em X é uma homotopia $H : X \times I \to X$ tal que $H_0 : X \times \{0\} \to X$ é a aplicação identidade e $H_t : X \times \{t\} \to X$ seja homeomorfismo para todo $t \in I$.*

Definição 12. *Dizemos que dois nós K_1 e K_2 têm o mesmo tipo ou são equivalentes se existe uma isotopia ambiente $H : \mathbb{R}^3 \times I \to \mathbb{R}^3$ tal que $H(K_1, 1) = K_2$.*

Podemos, agora, estabelecer com precisão o que é a Teoria dos Nós em $\mathbb{R}^3$. Essa área de pesquisa consiste no estudo das várias maneiras possíveis de mergulhar uma esfera unidimensional S^1 no espaço euclidiano tridimensional. A principal questão nessa teoria é determinar quando um nó pode ser deformado continuamente em outro nó.

Para estudarmos um nó, tendo em vista a dificuldade de desenharmos objetos tridimensionais, fixamos um plano do espaço e utilizamos sua projeção nesse plano (sempre é possível considerar projeções que possuam apenas cruzamentos duplos). Nessa projeção, desenhamos pequenas interrupções próximas a cada cruzamento para indicarmos que um arco passa por cima do outro. Assim, obtemos o *diagrama do nó.*

Mas, se para registrarmos um nó utilizarmos um diagrama (uma projeção), é necessário investigar se dois diagramas que, a princípio, são aparentemente diferentes, podem representar o mesmo nó. Esse problema foi resolvido por Kurt Reidemeister em [11]. Ele propôs que se considerasse três movimentos que poderiam ser realizados em diagramas de nós. Hoje esses movimentos são chamados *movimentos de Reidemeister.*

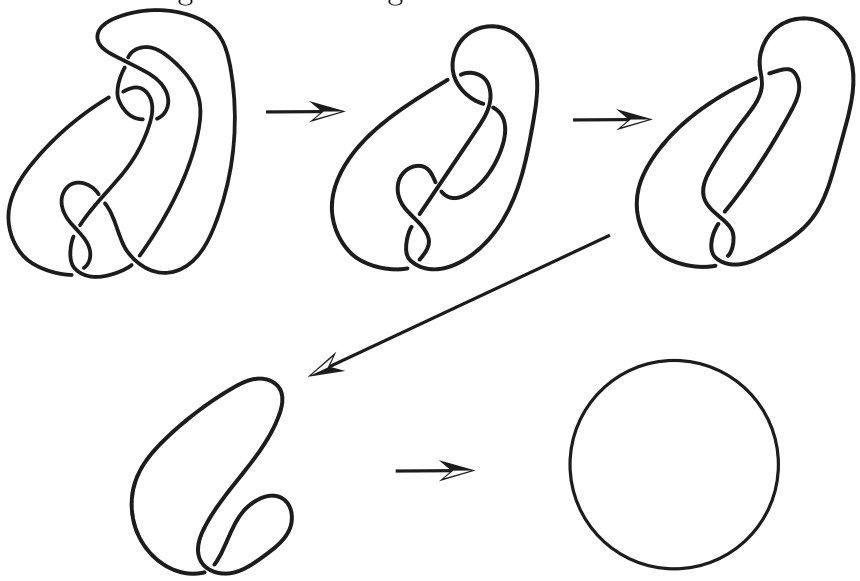
Cada um dos movimentos de Reidemeister é realizável devido à existência de uma isotopia ambiente que o possibilita. Na verdade, uma sequência finita de movimentos de Reidemeister está associada à existência de uma sequência finita de isotopias ambientes que permite posicionar um nó sobre o outro. Portanto, quando um diagrama de um nó pode ser transformado no diagrama de outro nó, mediante aplicação de uma sequência finita de movimentos de Reidemeister, então estes nós têm mesmo tipo.

Com os movimentos de Reidemeister estabelecemos o importante conceito de nó trivial:

Definição 13. *Um nó K é chamado trivial se seu diagrama pode ser transformado na circunferência unitária S^1, mediante aplicação de uma sequência finita de movimentos de Reidemeister.*

A Figura 2.1 ilustra o conceito de nó trivial.

Figura 2.1: O diagrama de um nó trivial.



Fonte: Elaboração do autor.

Apesar do excepcional resultado de Reidemeister, às vezes encontramos dificuldades para determinar visualmente que não existe uma sequência finita de movimentos que transforme o diagrama de um nó em outro, ou seja, que não existem isotopias ambientes que permitam posicionar um nó sobre o outro. Para isso, foram desenvolvidos objetos aritméticos e algébricos associados ao nó que nos fornecem respostas acerca da não existência de tais isotopias. Esses objetos são chamados "invariantes do nó". Mais precisamente,

Definição 14. *O invariante do nó é um objeto aritmético ou algébrico que não se altera quando no diagrama do nó se efetua uma sequência finita de movimentos de Reidemeister.*

Neste capítulo, abordaremos dois destes invariantes: o *Determinante* e o *Polinômio de Alexander*.

2.1 O DETERMINANTE

O Determinante é um número real não negativo associado ao nó. Este número é um invariante do nó, isto é, quaisquer dois nós de mesmo tipo possuem o mesmo determinante, veja [9].

O determinante de um nó K é estabelecido de forma algorítmica, como veremos a seguir.

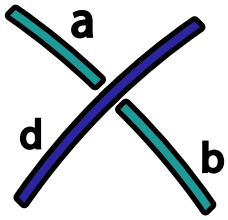
1. Escolha um diagrama para K.

2. Associe a cada arco uma variável e a cada cruzamento uma equação da forma

$$a + b - 2d = 0$$

seguindo o esquema da Figura 2.2.

Figura 2.2: Cruzamento.



Fonte: Elaboração do autor.

3. Coloque uma variável qualquer igual a zero.

4. Descarte uma equação qualquer.

5. Ficará então determinado um sistema de $n-1$ equações e $n-1$ variáveis, onde n é o número de cruzamentos do diagrama do nó. Calculamos o determinante da matriz formada pelos coeficientes das equações do sistema linear obtido. O valor absoluto deste determinante é o *Determinante* do nó K.

Exemplo 11. Vamos calcular o determinante do nó trevo, cujo diagrama aparece na Figura 2.3.

Aplicando os dois primeiros passos do algoritmo, obtemos as equações:

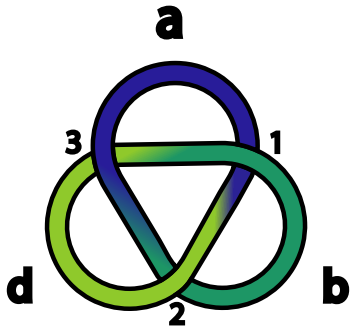
$$a + d - 2b = 0$$

$$b + a - 2d = 0$$

$$d + b - 2a = 0$$

Colocando $d = 0$ e descartando a equação $d + b - 2a = 0$, obtemos o sistema:

Figura 2.3: Nó trevo.



Fonte: Elaboração do autor.

$$\begin{cases} a - 2b = 0 \\ a + b = 0 \end{cases}$$

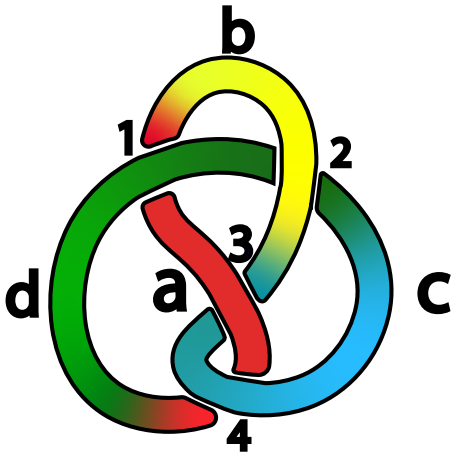
Agora, calculamos o determinante da matriz do sistema:

$$\begin{vmatrix} 1 & -2 \\ 1 & 1 \end{vmatrix} = 3$$

Portanto, o determinante do nó trevo é 3.

Exemplo 12. Vamos calcular o determinante do nó figura 8, cujo diagrama aparece na Figura 2.4.

Figura 2.4: Nó figura 8.



Fonte: Elaboração do autor.

No caso do nó figura 8, temos as seguintes equações:

$$a + b - 2d = 0$$

$$d + c - 2b = 0$$

$$b + c - 2a = 0$$

$$a + d - 2c = 0$$

Colocando a variável d = 0 e eliminando a quarta equação, obtemos o sistema:

$$\begin{cases} a + b = 0 \\ -2b + c = 0 \\ -2a + b + c = 0 \end{cases}$$

Calculamos o determinante da matriz associada ao sistema:

$$\begin{vmatrix} 1 & 1 & 0 \\ 0 & -2 & 1 \\ -2 & 1 & 1 \end{vmatrix} = -5$$

Assim, o determinante do nó figura 8 é o valor 5.

Observação 7. *Os nós trevo e figura oito não são equivalentes, pois seus determinantes são distintos.*

2.2 Polinômio de Alexander

O polinômio de Alexander é um invariante algébrico associado ao nó, veja [9].

Assim como o determinante, o polinômio de Alexander de um nó K é obtido de forma algorítmica, como veremos a seguir.

1. Escolha um diagrama para K e uma orientação para o diagrama.

2. Associe a cada arco uma variável, seguindo o esquema da Figura 2.5, e a cada cruzamento uma equação da forma
$$b - ta - (1 - t)d = 0.$$

   Para ilustração da Figura 2.5, d deve ser associada ao arco que passa superiormente na região do cruzamento. Para a escolha de a e b na equação, procedemos da seguinte forma: usando a orientação do trecho superior do nó, no cruzamento, a deve ser identificada com a variável associada à direita de d, e consequentemente, b deve ser identificada com a variável à esquerda de d.

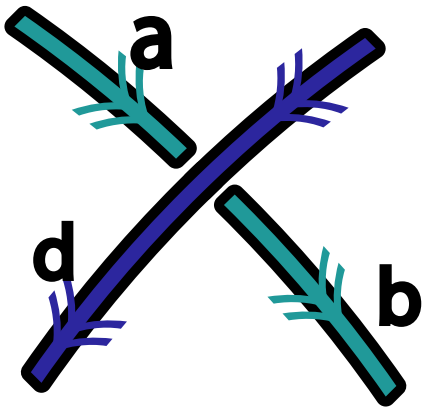
3. Coloque uma variável qualquer igual a zero.

4. Descarte uma equação qualquer.

5. Escreva o sistema de equações obtidas anteriormente.

6. Calcule o determinante $\delta(t)$ deste sistema.

Figura 2.5: Cruzamento com orientação.

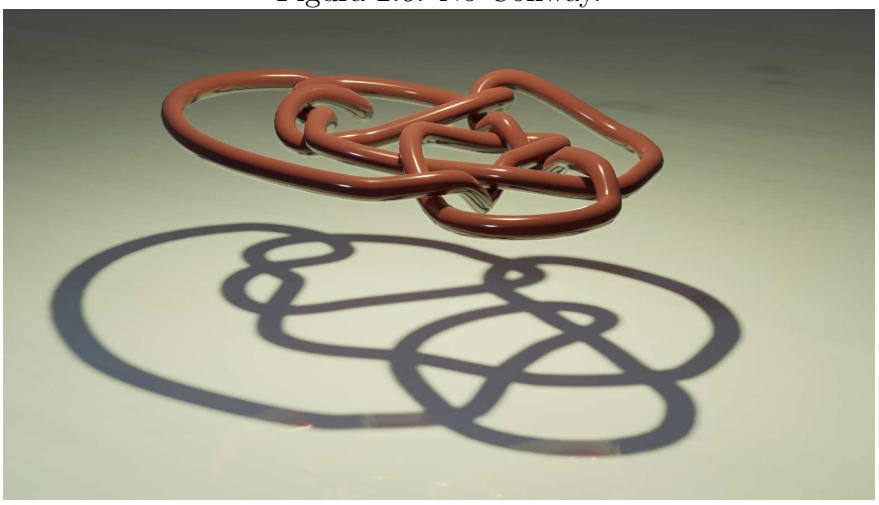


Fonte: Elaboração do autor.

7. Multiplique $\delta(t)$ por $\pm t^j$ apropriado para obter $p(t)$, tal que $p(t) = p(t^{-1})$ e $p(1) = 1$. Este $p(t)$ é o Polinômio de Alexander do nó K.

Exemplo 13. Vamos calcular o Polinômio de Alexander do Nó Conway, ilustrado na Figura 2.6. Este nó, com 11 cruzamentos, foi apresentado pela primeira vez pelo matemático John Conway nos anos 1970. Mais informações podem ser encontradas em [12, p. 173]. Olhamos agora para o seu diagrama na Figura 2.7.

Figura 2.6: Nó Conway.



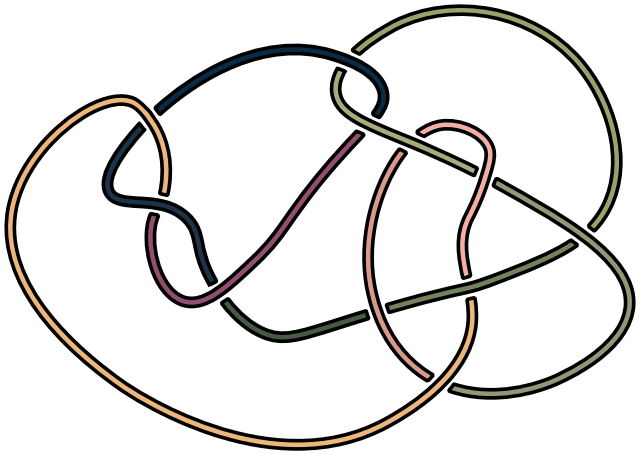
Fonte: Elaboração do autor.

A Figura 2.8 mostra uma escolha de orientação para o diagrama do nó e uma enumeração de seus 11 cruzamentos. A cada cruzamento vamos associar uma equação:

1. $i - tj - (1-t)a = 0$

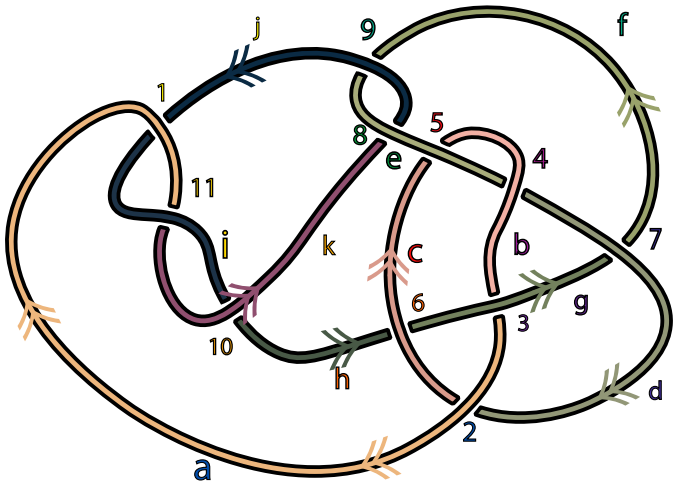
2. $c - td - (1-t)a = 0$

Figura 2.7: Diagrama do Nó Conway.



Fonte: Elaboração do autor.

Figura 2.8: Diagrama orientado do Nó Conway.



Fonte: Elaboração do autor.

3. $a - tb - (1-t)g = 0$

4. $e - td - (1-t)b = 0$

5. $c - tb - (1-t)e = 0$

6. $g - th - (1-t)c = 0$

7. $g - tf - (1-t)d = 0$

8. $k - tj - (1-t)e = 0$

9. $f - te - (1-t)j = 0$

10. $h - ti - (1-t)k = 0$

$$11.\ k - ta - (1-t)i = 0$$

No quadro abaixo, sintetizamos as informações obtidas com a aplicação do algoritmo ao diagrama do nó Conway. Na primeira linha encontram-se as variáveis utilizadas e na primeira coluna a enumeração dos cruzamentos. As demais células guardam os coeficientes das equações associadas aos cruzamentos.

	a	b	c	d	e	f	g	h	i	j	k
1	$t-1$	0	0	0	0	0	0	0	1	$-t$	0
2	$t-1$	0	1	$-t$	0	0	0	0	0	0	0
3	1	$-t$	0	0	0	0	$t-1$	0	0	0	0
4	0	$t-1$	0	$-t$	1	0	0	0	0	0	0
5	0	$-t$	1	0	$t-1$	0	0	0	0	0	0
6	0	0	$t-1$	0	0	0	1	$-t$	0	0	0
7	0	0	0	$t-1$	0	$-t$	1	0	0	0	0
8	0	0	0	0	$t-1$	0	0	0	0	$-t$	1
9	0	0	0	0	$-t$	1	0	0	0	$t-1$	0
10	0	0	0	0	0	0	0	1	$-t$	0	$t-1$
11	0	0	0	0	0	0	0	0	$t-1$	0	1

Colocando a variável $a = 0$ e eliminando a equação associada ao cruzamento 11, obtemos a matriz do sistema cujas entradas são as células do quadro acima, desconsiderando a primeira e última linhas, e desconsiderando as duas primeiras colunas. Esta matriz tem determinante

$$\delta(t) = -t^5.$$

Assim, para que $p(t) = p(t^{-1})$ e $p(1) = +1$, multiplicamos $\delta(t)$ por $-t^{-5}$:

$$p(t) = -t^{-5}(-t^5) = 1.$$

É conhecido que o nó Conway não é trivial. Este exemplo mostra que o Polinômio de Alexander não é um invariante completo, isto é, o Polinômio de Alexander ser trivial não garante que o nó seja trivial.

É relevante observar que os cálculos envolvidos podem ser bastante complexos, e a dedução do polinômio de Alexander frequentemente é conduzida com o auxílio de softwares especializados em teoria dos nós. Essas ferramentas computacionais têm a capacidade de automatizar os cálculos e as manipulações algébricas necessárias para obter o polinômio de Alexander, ajudando principalmente no calculo do determinante da matriz, que por vezes pode ser demasiadamente trabalhoso.

3. CONSTRUINDO UMA APLICAÇÃO WEB

No capítulo anterior, adquirimos conhecimento acerca dos conceitos de nó e também sobre dois invariantes. Conforme observado, a elaboração de ambos apresenta um caráter algorítmico. Considerando tal perspectiva, surge a indagação: *Será viável desenvolver um código computacional visando otimizar o procedimento?* A resposta a essa indagação é afirmativa. Neste capítulo, relataremos o desenvolvimento de uma aplicação Web que calcula os invariantes *Determinante* e *Polinômio de Alexander*.

Para ser melhor entendido pelo leitor, daremos algumas explicações de palavras que utilizaremos com certa frequência nas secções seguintes.

Uma **aplicação Web** é um software concebido para ser executado em um navegador da web, composta por duas partes principais: o frontend e o backend. Cada uma dessas partes desempenha uma função específica no funcionamento da aplicação.

O **frontend** constitui a porção da aplicação web que interage diretamente com os usuários. Compreende a interface do usuário, elementos visuais e todos os componentes com os quais os usuários interagem diretamente no navegador. As linguagens comuns para o desenvolvimento do frontend incluem HTML (linguagem de marcação para estruturação), CSS (linguagem de estilo para design) e JavaScript (linguagem de programação para interatividade).

O **backend**, por sua vez, é a porção "invisível"da aplicação responsável pela lógica de negócios, processamento de dados, armazenamento e interações com o banco de dados. Ele recebe solicitações provenientes do frontend, processa essas solicitações e retorna os resultados. Além disso, gerencia o estado da aplicação e executa operações que não são possíveis no frontend, como acesso a banco de dados, autenticação de usuários, entre outras. Linguagens comuns para o desenvolvimento do backend incluem Node.js (utilizando JavaScript), Python, Java, entre outras.

Agora, descreveremos brevemente as ferramentas computacionais utilizadas no desenvolvimento do nosso aplicativo.

3.1 PYTHON

A linguagem de programação Python é utilizada por sua simplicidade e clareza. Apesar de simples, é também uma linguagem poderosa, podendo ser utilizada para desenvolver grandes projetos.

Python é livre, isto é, pode ser utilizada gratuitamente, graças ao trabalho da *Python*

Foundation[1]. Destacamos ainda o fato da linguagem Python possuir interpretador disponível para diversos sistemas operacionais, como Linux, FreeBSD, Microsoft Windows e Mac OSX.

Em resumo, Python é uma linguagem de programação popular e versátil, que pode ser usada em uma ampla variedade de aplicações, desde programação Web e científica até automação e inteligência artificial. Para mais informações veja [6].

3.2 JAVASCRIPT

JavaScript é uma linguagem de programação que foi criada em 1995 por Brendan Eich, veja [2], enquanto trabalhava na *Netscape Communications Corporation*. Originalmente, ela foi projetada para ser executada no navegador Web, permitindo que os desenvolvedores criassem sites e aplicações Web dinâmicas.

O JavaScript é uma linguagem interpretada, ou seja, ela é executada diretamente pelo navegador sem a necessidade de compilação prévia. Ele é uma linguagem de programação orientada a objetos e possui recursos avançados de programação funcional.

Hoje em dia, o JavaScript é amplamente utilizado para criar aplicações web complexas, jogos, aplicações de desktop e móveis, e até mesmo para o desenvolvimento de servidores back-end. Ele tem uma sintaxe simples e elegante, com recursos de alta ordem, funções assíncronas e promise, o que o torna uma linguagem poderosa para o desenvolvimento de aplicações modernas.

Além disso, o JavaScript tem uma grande comunidade de desenvolvedores que criaram uma vasta biblioteca de recursos e frameworks para facilitar o desenvolvimento de aplicações Web. Entre os frameworks mais populares estão o Angular, React e Vue, que permitem a criação de interfaces de usuário ricas e dinâmicas.

Em resumo, o JavaScript é uma linguagem de programação dinâmica e poderosa que é amplamente utilizada para o desenvolvimento de aplicações Web modernas e avançadas.

3.3 HTML E CSS

HTML (*HyperText Markup Language*) é uma linguagem de marcação utilizada para estruturar e organizar o conteúdo de uma página na Web. Ela define os elementos e suas relações, como títulos, parágrafos, imagens, links e outros componentes, permitindo a criação de uma estrutura hierárquica que os navegadores interpretam para exibir o conteúdo de forma adequada.

CSS (*Cascading Style Sheets*) é uma linguagem de estilos usada para controlar a apresentação e o design visual de páginas Web. Com o CSS, é possível definir as cores, fontes, margens, espaçamentos e outros aspectos visuais de elementos HTML. Isso separa a estrutura do conteúdo (definido pelo HTML) da sua aparência (definida pelo CSS), possibilitando uma maior flexibilidade e consistência no design das páginas.

[1] https://www.python.org/

3.4 PyScript

O PyScript[2] é uma estrutura Web de código aberto que permite criar aplicações Web front-end utilizando Python. Com sua estrutura, pode-se incorporar um código Python em HTML e este será executado no navegador - sem executar o Python no backend. Assim, o usuário não necessita ter o Python instalado em seu computador.

3.5 Desenvolvendo a aplicação WEB

Desenvolvemos um site, disponível em [1], onde apresentamos um pouco da história da Teoria dos Nós, sua aplicação em outras áreas do conhecimento e os invariantes Determinante e Polinômio de Alexander, como ilustra a Figura 3.1. Além disso, o usuário encontrará um vídeo tutorial explicando o funcionamento do aplicativo e um "botão" para acessar o mesmo, veja a Figura 3.2. Este site foi todo construído utilizando HTML, CSS e JavaScript.

Figura 3.1: Página inicial

Fonte: Elaboração do autor.

[2] https://pyscript.net/

Figura 3.2: Botão encontrado no final do site para acessar o aplicativo.

Fonte: Elaboração do autor.

3.6 A interface gráfica e o funcionamento do aplicativo

Para facilitar o desenho de diagramas e os cálculos envolvidos, adotaremos que a esfera unidimensional S^1 pode ser considerada uma sucessão finita de segmentos de reta. Assim, um mergulho de S^1 em $\mathbb{R}^3$ pode ser considerado mediante o desenho de um diagrama constituído por segmentos de reta. Vamos, para efeito de simplificação, considerar que as interseções de dois possíveis segmentos se dá em um ponto interno a cada segmento, e nunca em vértices. Na área da Topologia, diz-se que iremos trabalhar com a categoria linear por partes (picewise-linear). Com essas considerações, pode-se obter um diagrama do nó, como ilustra a Figura 3.3.

Na Figura 3.3, temos um diagrama do nó Conway composto por 16 segmentos de reta. É importante destacar que a quantidade de segmentos que compõem um diagrama de um nó não influencia o cálculo do polinômio de Alexander de tal nó, pois existe uma isotopia ambiente que suaviza os vértices do diagrama.

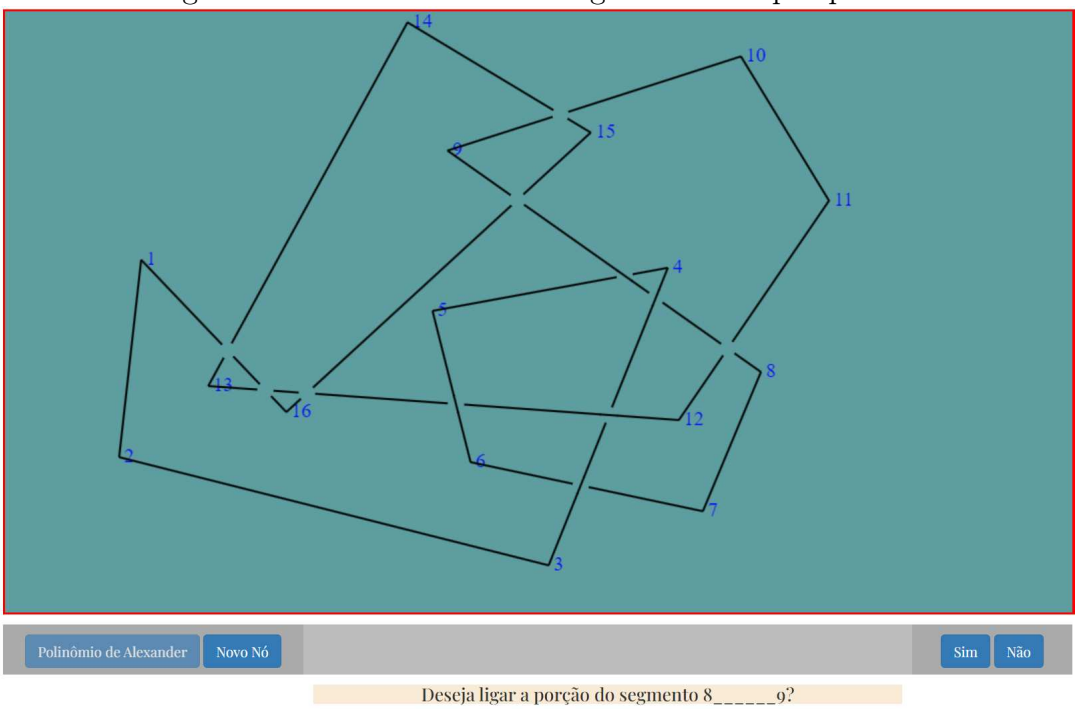
O usuário dispõe de uma área de desenho de 1070×600 pixels, na qual vai desenhar o diagrama do nó clicando no botão esquerdo do mouse para criar os vértices dos segmentos de reta que compõem o diagrama. À medida que desloca o mouse sobre a área de desenho, vai clicando no botão esquerdo e estocando as posições $A_i(x_i, y_i)$ e $A_{i+1}(x_{i+1}, y_{i+1})$.

Após o segundo ponto armazenado, determina-se a equação da reta que passa por A_i e A_{i+1} e, mediante a consideração da parametrização

$$A_i + t_i(A_{i+1} - A_i),\ t_i \in (0, 1)$$

desenha-se o segmento $\overline{A_i A_{i+1}}$.

Figura 3.3: Construindo um diagrama linear por partes.



Fonte: Elaboração do autor.

Para fechar o diagrama do nó o usuário deve mover o mouse sobre o primeiro ponto, neste momento o usuário é questionado se deseja fechar o diagrama. Se a escolha do usuário for a de fechar o diagrama, então o aplicativo traçará o segmento necessário.

Ao final deste processo tem-se uma coleção de segmentos indexados segundo a ordem executada pelo usuário, $I_1, I_2, \ldots, I_n$, onde I_1 indica o segmento ligando 1 a 2, I_2 o segmento ligando 2 a 3 e assim por diante.

As interseções entre os segmentos são obtidos resolvendo-se o seguinte sistema:

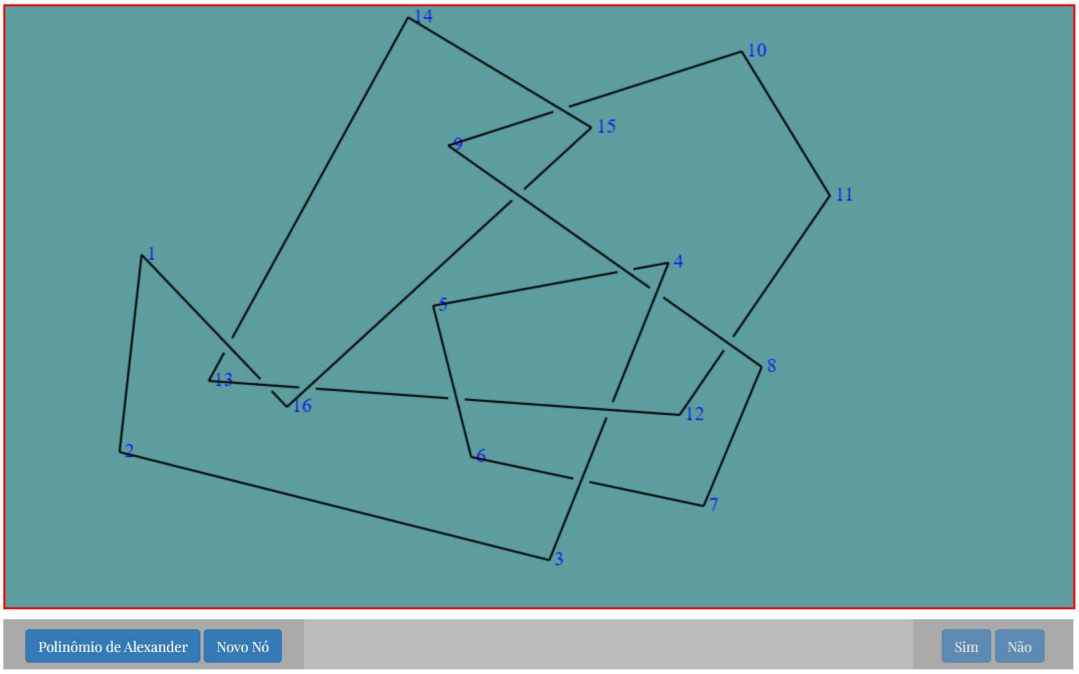
$$\begin{gathered} I_1 \cap I_3,\ I_1 \cap I_4,\ I_1 \cap I_5, \ldots, I_1 \cap I_n \\ I_2 \cap I_4,\ I_2 \cap I_5, \ldots, I_2 \cap I_n \\ \vdots \\ I_{n-3} \cap I_{n-1},\ I_{n-3} \cap I_n \\ I_{n-2} \cap I_n \end{gathered}$$

A título de explicação, fixa-se o primeiro segmento e procura-se as interseções deste com os demais segmentos; não é necessário procurar interseção com o subsequente, pois tal interseção será um dos vértices. Depois, fixa-se o segundo segmento e procura-se as interseções dele com os demais segmentos e assim sucessivamente.

Como consideramos segmentos de reta, ao determinarmos as interseções da reta r_i, que passa por A_i e A_{i+1}, com a reta r_j, que passa por A_j e A_{j+1}, restringimos os parâmetros t_i e t_j, respectivamente de r_i e r_j, ao intervalo $(0, 1)$, para evitar pontos de interseção que não façam parte do diagrama desenhado.

Finalizada essa etapa, os pontos de interseção são armazenados na memória. Na próxima etapa, o aplicativo apaga uma pequena área circular em torno de cada um dos pontos de interseção. Na barra de status do aplicativo, o usuário é indagado e deve escolher qual segmento deve passar por sobre o outro, veja a Figura 3.3. Isso será feito até que todos os cruzamentos tenham sido considerados. Ao final dessa etapa, o diagrama do nó fica semelhante ao ilustrado na Figura 3.4

Figura 3.4: Diagrama do nó Conway.



Fonte: Elaboração do autor.

Cada escolha fornecida pelo usuário sobre qual segmento passa por cima em cada cruzamento é armazenado em um vetor com a estrutura:

$$V_i = (i, j, i \text{ ou } j).$$

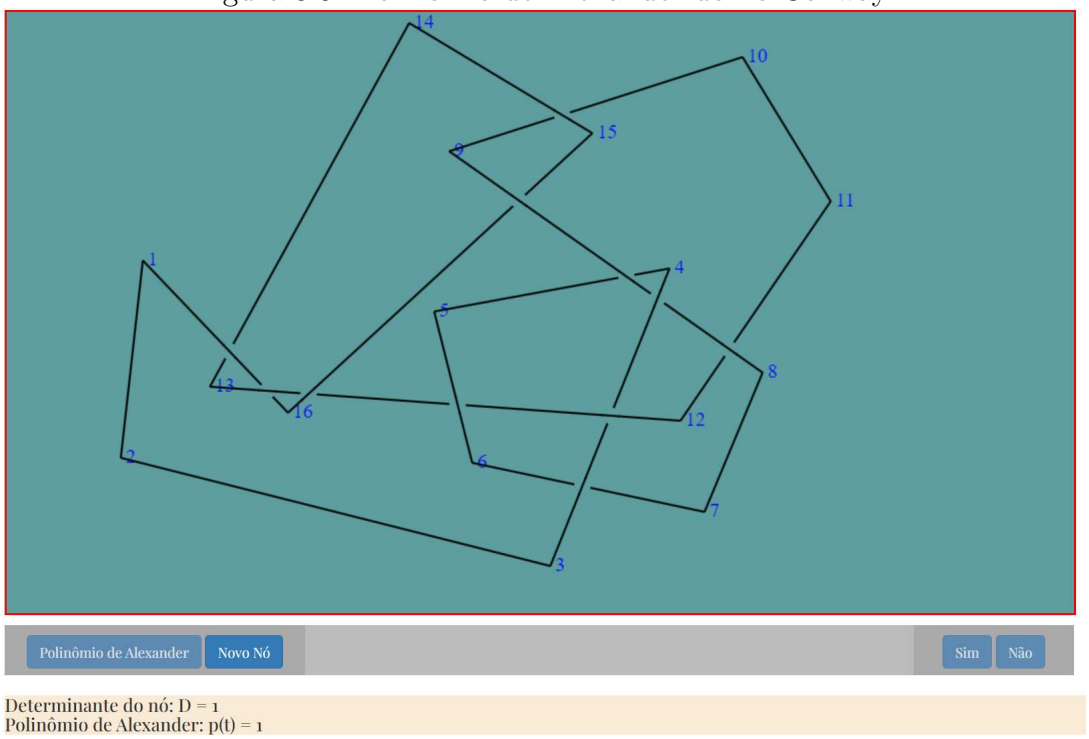
As duas primeiras coordenadas desse vetor indicam os índices dos segmentos que se interceptam; a terceira será igual a i se o segmento I_i passar por cima do segmento I_j, ou será igual a j se I_j passar por cima de I_i. Por exemplo, observando a Figura 3.4, temos um vetor $V_3 = (3, 6, 3)$, pois I_3 e I_6 se interceptam com I_3 passando por cima de I_6.

São as informações dos vetores V_i's que permitem determinar o sistema de equações e realizar os procedimentos descritos no Capítulo 2 para o cálculo do Polinômio de Alexander.

Para que o aplicativo calcule o polinômio de Alexander o usuário deve clicar no botão "Polinômio de Alexander", presente na barra de status. Logo abaixo do diagrama do nó, o aplicativo apresentará o Determinante do nó, **D**, e o Polinômio de Alexander do nó, $\mathbf{p(t)}$.

Nossa aplicação Web foi construída com o PyScript por meio da adaptação da rotina Python desenvolvida em [7]. O código fonte usando o PyScript encontra-se disponível no Apêndica A deste trabalho.

Figura 3.5: Polinômio de Alexander do nó Conway.



Fonte: Elaboração do autor.

Por fim, destacamos a importância da biblioteca SymPy[3] para o desenvolvimento da aplicação Web, pois esta biblioteca Python trabalha com a Matemática simbólica.

[3]https://www.sympy.org/en/index.html

4. Conclusões

Verificamos a eficiência do cálculo dos invariantes Determinante e Polinômio de Alexander pela aplicação web que desenvolvemos utilizando as linguagens de programação Python e JavaScript via PyScript. Ressaltamos a importância do conhecimento de linguagens de programação para que se possa implementar conceitos matemáticos que são descritos por meio de algoritmos, como foi o caso apresentado neste trabalho. Pretendemos, com isso, incentivar o uso de ferramentas computacionais por parte dos estudantes de Matemática.

O objetivo principal foi atingido, a aplicação web tornou possível compartilhar essa ferramenta valiosa com pessoas ao redor do mundo, eliminando barreiras geográficas e tornando-a acessível a qualquer pessoa com acesso à internet. Esse trabalho somente foi possível com o fomento fornecido pela FAPEMIG.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Barbosa, V. P. S.: *Teoria dos Nós*, 2021. https://victorpsena.github.io/ProjetoIC/index.html, acessado em 10/09/2023.

[2] Eich, B.: *Brendan Eich*, 2023. https://en.wikipedia.org/wiki/Brendan_Eich, acessado em 10/09/2023.

[3] Gauss, C. F.: *Zur mathematischen theorie der eletrodynamischen wirkungen*. Werke Konigl. Gessell. Wiss. Gottingen, 1833.

[4] Lima, E.: *Elementos De Topologia Geral*. Itc, 1976.

[5] Lima, E.: *Grupo Fundamental e Espaços de Recobrimento*. IMPA, 2012.

[6] Menezes, N. N. C.: *Introdução à programação com Python*. Berkeley, CA: Publish or Perish,, 3ª ed., 2019.

[7] Miranda, A. J., Souza, T. O. e Barros, R. M. O.: *Polinômio de Alexander via Linguagem Python*. REMAT: Revista Eletônica de Matemática, 6(1):1–16, 2020.

[8] Munkres, J.: *Topology*. Pearson, 2017.

[9] Neto, O. M.: *Teoria de Nós*. 2º Colóquio da Região Sudeste, 2013.

[10] Osserman, R.: *Knot theory*, 2023. https://www.britannica.com/science/knot-theory, acessado em 02/08/2023.

[11] Reidemeister, K.: *Elementare Begründung der Knotentheorie*. Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg, 5:24–32, 1927.

[12] Rolfsen, D.: *Knots and Links*. Berkeley, CA: Publish or Perish,, 1976.

[13] Vilches, M. A.: *Topologia Geral: notas*. Departamento de Análise-IME, UERJ.

A. O código fonte usando PyScript

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,initial-scale=1" />
    <title>Alexander Polynomial via HTML and Pyscript for WEB</title>
    <link rel="stylesheet"
    href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/
    bootstrap.min.css">
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap
    /3.4.1/js/bootstrap.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.2/
    dist/js/bootstrap.bundle.min.js">
    </script>
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com
    /ajax/libs/jquery-confirm/3.3.2/jquery-confirm.min.css">
    <script src="https://cdnjs.cloudflare.com/ajax/libs/
    jquery-confirm/3.3.2/jquery-confirm.min.js"></script>
    <!--<script defer onload="console.log(`${pyscript.version}`)"
    src="https://pyscript.net/latest/pyscript.js"></script>-->
    <script src="https://cdn.jsdelivr.net/pyodide/v0.21.3/full/pyodide.js">
    </script>
    <link rel="stylesheet" href="https://pyscript.net/latest/pyscript.css" />
    <script defer src="https://pyscript.net/latest/pyscript.js"></script>
    <meta name="description" content="math.js | basic usage">
    <script src="https://unpkg.com/mathjs/lib/browser/math.js"></script>
<!-- ================================================================== -->
    <style>
        canvas {
            border: 2px solid red;
            display: block;
            background-color: cadetblue;
```

```
        margin-right: auto;
     }
</style>

<style>
    * {
        box-sizing: border-box;
    }
    /* Create two equal columns that floats next to each other */
    .column {
        float: left;
        width: 300px;
        padding: 10px;
        height: 50px; /* Should be removed. Only for demonstration */
        text-align: center;
    }
    .column2 {
        float: left;
        width: 608px;
        padding: 10px;
        height: 50px; /* Should be removed. Only for demonstration */
        text-align: center;
    }
    .column3 {
        float: left;
        width: 160px;
        padding: 10px;
        height: 50px; /* Should be removed. Only for demonstration */
        text-align: center;
    }
    #div_container {
        width: 1070px;
        height:600px;
        border: 0px solid red;
        margin: 0px auto;
        text-align: center;
    }
    #Dialog_closeKnot{
        margin: 0;
        position: absolute;
```

```
        }
</style>
</head>
<!-- ============================================================================ -->
<body>
<py-config>
    packages = ["sympy"]
</py-config>
<!-- ============================================================================-->
<div class="div_container">
    <h3> <strong>Instruction:</strong>  Desenhe a representação do diagrama
    do nó no
    quadro abaixo. </h3>
    <canvas id="canvas_drawing" width="1066" height="600"></canvas>
    <div class="column" style="background-color:#aaa;">
        <button id="btn_PolyAlexander" py-click="PolyAlexander_OnClk()"
        type="button"
        class="btn btn-primary">
            <span class="spinner-border spinner-border-sm"></span>
            Polinômio de Alexander
        </button>
        <button id="btn_NewKnot"    onclick="NewKnot()"      type="button"
        class="btn btn-primary">Novo Nó</button>
    </div>
    <div class="column2" style="background-color:#bbb;">
        <h4 id="DesejaLigar"> </h4>
    </div>
    <div class="column3" style="background-color:#aaa;">
        <button id="btn_Yes" onclick="YesClick()" type="button" class="btn btn-
        primary">Sim</button>
        <button id="btn_No"  onclick="NoClick()"  type="button" class="btn btn-
        primary">Não</button>
    </div>
    <dialog id="Dialog_closeKnot" style="opacity: 0.3">
        <p>Fechar?</p>
        <form method="dialog">
            <button class="button-close" onclick="DiagramKnot()" >Yes</button>
            <button class="button-close" onclick="OK=0">No</button>
        </form>
    </dialog>
```

```
</div>
<br>
<br>
<h4 id="demo"> </h4>
<!-- ========================================================================== -->

<script>// define canvasEl etc....and the construction of diagram of knot
    i = 0; m = 0; let  newKnot= 1; let z1, w1, xc, yc;
    Sx = [];  // screen x coordinate of click
    Sy = [];  // screen y coordinate of click
    Cx = [];  // screen x converted in cartesian coordinate
    Cy = [];
    KnotMatrixAux=[]
    let canvasEl = document.querySelector('#canvas_drawing') //"canvas_drawing"
    is the "id" of
    "canvas" defined in html body (<canvas> is a html object)
    let ctx = canvasEl.getContext('2d')  //getContext('2d')  allows us drawing
    2d geometry
    ctx.fillStyle='blue'  //text color (color of number vértices)
    let OK=0

    canvasEl.addEventListener('mousemove', (MouseM) => {//get (x,y)-mouse
    move position
        if (OK==0){
            xM = MouseM.offsetX;
            yM = MouseM.offsetY;
            //document.getElementById("demo3").innerHTML = [xM,yM]
            if (m>2){
                //document.getElementById("demo2").innerHTML = "Para fechar
                o nó mova e clica o mouse no primeiro ponto."
                if (((Sx[0]-7 < xM) && (xM < Sx[0]+7)) && ((Sy[0]-7 < yM)
                && (yM < Sy[0]+7))){
                    Dialog_closeKnot.style.left = Sx[0] + "px";
                    Dialog_closeKnot.style.top = Sy[0] + "px"
                    window.Dialog_closeKnot.showModal()
                    OK = 1
                }
            }
        }
    })
```

```
    canvasEl.addEventListener('click', (MouseC) => {//get (x,y)-mouse
    click position
        if (newKnot == 1) {//if newKnot = 1, start the construction of
        knot diagram
            m = m+1
            x1 = MouseC.offsetX;  // x-coordinate of mouseclick
            y1 = MouseC.offsetY;
            xc = canvasEl.clientWidth;  //width of canvas
            yc = canvasEl.clientHeight;
            z1 = x1 - xc / 2  // transform screen coordinates to cartesian
            coordinates (with origin (0,0) in the center of canvas)
            w1 = -y1 + yc / 2
            Sx.push(x1) //push() add new item to the final of array
            Sy.push(y1)
            Cx.push(z1)
            Cy.push(w1)
            ctx.font = "20px serif";  //size of font
            ctx.fillText(m, x1 + 5, y1 + 5);
            ctx.lineWidth = 2
            i = i + 1

            if (i > 1) {//if number points > 1, then start the construction
            of diagram
                ctx.beginPath()
                ctx.moveTo(x11, y11)
                ctx.lineTo(x1, y1)
                ctx.stroke()
            }
            x11 = x1;
            y11 = y1;
        }
    })
</script>
<!-- ============================================================================ -->

<script>
    function IntersectCircleLine(xC,yC,r,p1,p2,q1,q2){
        //(xC,yC) =  center of circle;  r = radius of circle:
        (x-xC)^2+(y-yC)^2 = r^2
```

```
        //P(p1,p2) and Q(q1,q2) points that define segmente
        line rPQ by P and Q:  rPQ = [(1-
        s)p1+sq1, (1-s)p2+sq2] (parametrized with parameter s)
        G  = p1 * p1 - 2 * p1 * q1 + p2 * p2 - 2 * p2 * q2 +
        q1 * q1 + q2 * q2

        K  =  -p1 * p1 * q2 * q2 + 2 * p1 * p1 * q2 * yC + p1
        * p1 * r * r - p1 * p1 * yC * yC
        + 2 * p1 * p2 * q1 * q2 - 2 * p1 * p2 * q1 * yC - 2 *
        p1 * p2 * q2 * xC + 2 * p1 * p2
        * xC * yC - 2 * p1 * q1 * q2 * yC - 2 * p1 * q1 * r *
        r + 2 * p1 * q1 * yC * yC + 2 *
        p1 * q2 * q2 * xC - 2 * p1 * q2 * xC * yC - p2 * p2 *
        q1 * q1 + 2 * p2 * p2 * q1 * xC
        + p2 * p2 * r * r - p2 * p2 * xC * xC + 2 * p2 * q1 *
        q1 * yC - 2 * p2 * q1 * q2 * xC
        - 2 * p2 * q1 * xC * yC - 2 * p2 * q2 * r * r + 2 *
        p2 * q2 * xC * xC + q1 * q1 * r *
        r - q1 * q1 * yC * yC + 2 * q1 * q2 * xC * yC + q2 *
        q2 * r * r - q2 * q2 * xC * xC
        L  = p1 * p1 - p1 * q1 - p1 * xC + p2 * p2 - p2 *
        q2 - p2 * yC + q1 * xC + q2 * yC
        s  = [(Math.sqrt(K)+L)/G, -(Math.sqrt(K)-L)/G]
        return s
    }
</script>
<!-- ========================================================================== -->

<script>
    function disabled_button(id_button, true_or_false){//
    funtion to disabled or enabled the button by id="id_button"
        document.getElementById(id_button).disabled = true_or_false;
    }
</script>
<!-- ========================================================================== -->

<script>//start script function DiagramKnot
function DiagramKnot() {//this function close the knot
and start the questions about intersections
    document.getElementById("btn_Yes").disabled = false;
```

```
document.getElementById("btn_No").disabled = false;
newKnot = -1
ctx.beginPath()
ctx.moveTo(x1,y1)
ctx.lineTo(Sx[0],Sy[0]) //connect the first with de
last point (to close the knot diagram)
ctx.stroke()
//-------------------- starting to find the intersections between
the segments
let Var, Paramt1, num_I, done,  r
//Var: used to labeled the name of arcs of knot. num_I:
indicate the number of intersections
done = false
Paramt1 = []  //this array store all information for
each intersection point
num_I = 0
numpoints=Cy.length  //get the number of vertices of diagram
Sx.push(Sx[0])  //to close the diagram, the last point
coincide with the firs point
Sy.push(Sy[0])
Cx.push(Cx[0])
Cy.push(Cy[0])

for (let i = 0; i < numpoints; i++) {
    a1 = Cx[i]
    b1 = Cy[i]    //cartesian coordinate of A_i-click
    a2 = Cx[i+1]
    b2 = Cy[i+1]  //cartesian coordinate of A_{i+1}-click
    num_I = 0
    Paramt1 = []
    done = false
    for (let j = 0; j < numpoints; j++) {
          if (i != j) {
              // i!=j: intersection of adjacent segments do not matter
              c1 = Cx[j]  //cartesian coordinate of A_j-click
              d1 = Cy[j]
              c2 = Cx[j+1] //cartesian coordinate of A_{j+1}-click
              d2 = Cy[j+1]
              t1 = (a1*d1-a1*d2-b1*c1+b1*c2+c1*d2-c2*d1)/
              (a1*d1-a1*d2-a2*d1+a2*d2-b1*c1+b1*c2+b2*c1-b2*c2)
```

```
            t2 = -(a1 * b2 - a1 * d1 - a2 * b1 + a2
            * d1 + b1 * c1 - b2 * c1) / (a1 * d1
            - a1 * d2 - a2 * d1 + a2 * d2 - b1 * c1
            + b1 * c2 + b2 * c1 - b2 * c2)

            if ((0 < t1 && t1 < 1) && (0 < t2 && t2 < 1))
            {//then there was
            intersection between the segments
                done = true
                num_I = num_I+1
                z1 =  a1+t1*(a2-a1) //local variable of
                cartes. x-coord. of intersection
                w1 =  b1+t1*(b2-b1)
                xa =  z1+xc/2      //local variabel of
                screen x-coord. of intersection
                ya = -w1+yc/2

                Paramt1.push([t1, i+1, j+1, z1, w1, xa,
                ya, a1, b1, a2, b2, c1, d1, c2, d2])
                //store all information of each intersection point
                r = 8
                ctx.fillStyle='cadetblue'
                ctx.beginPath()
                ctx.ellipse(xa,ya, r, r,  Math.PI/4, 0, 2*Math.PI)
                //clears a region around the intersection
                ctx.fill()
            }
        }
}
for (let i =0; i < (num_I-1); i++){//ordena a lista
Paramt1 de acordo com a ordem crescente dos parâmetros t1.
    for (j = i+1; j < num_I; j++){
        if (Paramt1[j][0] < Paramt1[i][0]){
            aux = Paramt1[i]
            Paramt1[i] = Paramt1[j]
            Paramt1[j] = aux
        }
    }
}
if (done == true){
```

```
            for (let n =0; n < Paramt1.length; n++)
            {//significa que n varia de 0 até (len(Paramt1)-1)
            VD.push(Paramt1[n]) //adiciona Paramt1[n]
            para o novo vetor "VD"
            }
        }
    }

//---------------------- termino para encontrar os
pontos de interseção dos segmentos
    NC = VD.length/2 //NC = número de cruzamentos
    do diagrama do nó;  // = divisão inteira
    if (NC == 0){
        disabled_button("btn_Yes", "true")
        disabled_button("btn_No", "true")
        document.getElementById("demo").innerHTML
        = "Knot determinant: D = 1 <br> Alexander polynomial: p(t) = 1"
    }
    dNC = 2*NC

    //============================================================================
    q=-1
    for (let i = 0; i < dNC; i++) {//cria um array somente com
    os indices I que houve interseção
        for (let k = i; k < dNC; k++) {
            if ((VD[i][2] == VD[k][1]) && (VD[i][1] == VD[k][2]))
            {//busca pelos vetores v_i com [i,j] permutados,
            isto é, busca pelo vetor oposto
                q = q + 1
                VecIndex_I[q] = VD[i][1]
            }
        }
    }
    //============================================================================
    for(let i=0; i < dNC; i++){ //Cria uma matrix (6 x dNC) preenchida com zeros
        VecUpDown[i]=[]
        for(let j=0; j < 6; j++){
            VecUpDown[i][j]=0
        }
    }
```

```
    //=========================================================================
    //VecUpDown = [[0 for j in range(6)] for i in range(dNC)]
    #preenche o vetor com zeros
    //VecUpDown[0] guarda o índice i do segmento I_i,
    formado pelos pontos A_i e A_{i+1}
    //VecUpDown[1] guarda o índice j do segmento I_j,
    formado pelos pontos A_j e A_{j+1}
    //VecUpDown[2] guarda o índice i se I_i p
    assa por cima de I_j e guarda índice j se I_j passa por cima de I_i
    //VecUpDown[3] guarda números que correspondem
    as variáveis (nomes dos arcos), 1--> a, 2-->b, ...
    //VecUpDown[4] guarda +1 se o determinante
    referente à terceira coord. do prod. vetorial,
    ui= vec(A_iA_{i+1}) com wj=vec(A_iA_j) for positivo e
    //guarda -1 se o determinante referente à terceira coord. do prod.
    vetorial, ui= vec(A_iA_{i+1}) com wj=vec(A_iA_j) for negativo
    //                +1 e -1 diz qual arco está a
    direita ou a esquerda do segmento que passa
    por cima em cada cruzamento.
    // VecUpDown[5] guarda a posição do vetor oposto,
    útil para montar a matriz do sistema.
    //*************************************************************************
    k=-1
    document.getElementById("DesejaLigar").textContent =
    "Deseja ligar a porção do segmento " +
    VD[0][1] + '______' + (VD[0][1]+1)+"?";
    return VD, VecUpDown, dNC, k, VecIndex_I
    //retorna essas variáveis definidas antes como globais
}//End function DiagramKnot
</script>

<script>
    disabled_button("btn_Yes", "true") //disabled = true:  button started disabled
    disabled_button("btn_No", "true")
    disabled_button("btn_PolyAlexander", "true")

    //Below globals variables
    VD=[]  //double vector
    var VecUpDown = new Array(6)
    VecIndex_I=[]  // store the indexes i of intersections
```

```
    var dNC        // double of intersections
    let k=-1
    let cont=0
    KnotMatrixAux = []
    KnotMatrix = []
</script>

<script>
    function NewKnot(){
        window.location.reload() //restart o programa
    }
</script>

<script>
    function YesClick(){
        start_position: while (true) {
            k=k+1
            if (VecUpDown[k][2] != 0) continue start_position
            break;
        }
        cont = cont+1 //conta a quantidade de cliques
        (ou seja, a quantidade de interseção = dNC/2

        document.getElementById("DesejaLigar").textContent =
        "Deseja ligar a porção do segmento " +
        VecIndex_I[cont] + '______' + (VecIndex_I[cont]+1)+"?";

        if (cont==(dNC/2)){//se alcançar a quantidade de
        interseções, desativa os botoeSim e BotaoNao
            document.getElementById("btn_Yes").disabled = true;
            document.getElementById("btn_No").disabled = true;
            document.getElementById("DesejaLigar").textContent =""
        }
        xa    = VD[k][5]    //xa e ya são coordenadas
        (tela) das interseções dos segmentos, usado para criar o círculo
        ya    = VD[k][6]
        ui_x  = VD[k][9]-VD[k][7]      //i = vetor vec(A_iA_{i+1})
        ui_y  = VD[k][10]-VD[k][8]
        wj_x  = VD[k][11]-VD[k][7]     // = vetor vec(A_iA_{j})
        wj_y  = VD[k][12]-VD[k][8]
```

```
sinal = ui_x*wj_y-ui_y*wj_x    //coord. k do prod.
vetorial, para fornecer a orientação
if (sinal > 0) {//orientacao +
    VecUpDown[k][4] = 1
} else {
    VecUpDown[k][4] = -1 //orientação -
}
r = 9        //raio do disco
VecUpDown[k][0] = VD[k][1]  //VD[k][1] = i
VecUpDown[k][1] = VD[k][2]  //VD[k][2] = j
VecUpDown[k][2] = VD[k][1]  //então, q
uer dizer que o segmento I_i passou por cima do segmento I_j
//temos que: p_i retorna os dois param.
da interseção do círculo (x-z1)^2+(y-z1)^2=r^2
com segmento A_iA_i+1
p_i = IntersectCircleLine(VD[k][3], VD[k][4],
r, VD[k][7], VD[k][8], VD[k][9], VD[k][10])
//agora converter para coord. da tela e ligar
esses dois pontos para fechar a porção desejada do segmento.
xi = VD[k][7] + p_i[0] * (VD[k][9] - VD[k][7])
yi = VD[k][8] + p_i[0] * (VD[k][10] - VD[k][8])
Xi = xi + xc / 2
Yi = -yi + yc / 2

xii = VD[k][7] + p_i[1] * (VD[k][9] - VD[k][7])
yii = VD[k][8] + p_i[1] * (VD[k][10] - VD[k][8])
Xii = xii + xc / 2
Yii = -yii + yc / 2

ctx.beginPath()
ctx.moveTo(Xi, Yi)
ctx.lineTo(Xii, Yii) //segmento que une o segmento interrompido
ctx.stroke()
//Inicio_____________ for (let i = k; k < dNC; i++)
tex=""
for (let i = k; k < dNC; i++) {
    if ((VD[i][2] == VD[k][1]) && (VD[i][1] == VD[k][2]))
    {//busca pelos vetores v_i com [i,j] permutados, isto é,
    busca pelo vetor oposto
        VecUpDown[i][0] = VD[k][2] //uma vez encontrado o
```

```
                oposto em VD, então apenas atualizamos em VecUpDown
                VecUpDown[i][1] = VD[k][1]
                VecUpDown[i][2] = VD[k][1]
                VecUpDown[k][5] = i        // guarda a posição que
                está o vetor oposto
            }
            if (i == dNC-1 && cont==(dNC/2)) {
                document.getElementById("btn_PolyAlexander").disabled = false;
            } // Este último if funcionou aqui (não sei por que).
            Coloquei esse mesmo if abaixo deste for e não funcionou
            (não sei por que);
        }
    }
</script>

<script>//NoClick
    function NoClick(){
        start_position: while (true) {//só vai executar o bloco
        abaixo desse while, se VecUpDown[k][2] = 0.
            k=k+1
            if (VecUpDown[k][2] != 0) continue start_position
            break;
        }
        cont = cont+1 //conta a quantidade de cliques
        (ou seja, a quantidade de interseção = dNC/2
        document.getElementById("DesejaLigar").textContent =
        "Deseja ligar a porção do segmento " + VecIndex_I[cont]
        + '______' + (VecIndex_I[cont]+1)+"?";

        if (cont==(dNC/2)){
            document.getElementById("DesejaLigar").textContent =""
            document.getElementById("btn_Yes").disabled = true;
            document.getElementById("btn_No").disabled = true;
        }
        xa    = VD[k][5]    //xa , ya are screen coordinates
        of intersections of segments
        ya    = VD[k][6]
        ui_x  = VD[k][9]-VD[k][7]      //i = vetor vec(A_iA_{i+1})
        ui_y  = VD[k][10]-VD[k][8]
        wj_x  = VD[k][11]-VD[k][7]     // = vetor vec(A_iA_{j})
```

```
wj_y  = VD[k][12]-VD[k][8]
sinal = ui_x*wj_y-ui_y*wj_x    //coord. k do prod.
vetorial, para fornecer a orientação
if (sinal > 0) {//orientacao +
    VecUpDown[k][4] = 1
    } else {
    VecUpDown[k][4] = -1 //orientação -
    }
VecUpDown[k][0] = VD[k][1]//VD[k][1] = i
VecUpDown[k][1] = VD[k][2]//VD[k][2] = j
VecUpDown[k][2] = VD[k][2]// então, quer dizer que I_j
passou por cima de I_i
r=9
p_i = IntersectCircleLine(VD[k][3], VD[k][4], r,
VD[k][11], VD[k][12], VD[k][13], VD[k][14])
//p_i retorna os dois param. da interseção do
círculo (x-z1)^2+(y-z1)^2=r^2 com segmento A_jA_{j+1}
xi = VD[k][11] + p_i[0] * (VD[k][13] - VD[k][11])
yi = VD[k][12] + p_i[0] * (VD[k][14] - VD[k][12])
Xi = xi + xc / 2
Yi = -yi + yc / 2

xii = VD[k][11] + p_i[1] * (VD[k][13] - VD[k][11])
yii = VD[k][12] + p_i[1] * (VD[k][14] - VD[k][12])
Xii = xii + xc / 2
Yii = -yii + yc / 2

ctx.beginPath()
ctx.moveTo(Xi, Yi)
ctx.lineTo(Xii, Yii) //segment joining the
interrupted segment
ctx.stroke()

for (let i = k; k < dNC; i++) {
    if ((VD[i][2] == VD[k][1]) && (VD[i][1] == VD[k][2])){
        VecUpDown[i][0] = VD[k][2]
        VecUpDown[i][1] = VD[k][1]
        VecUpDown[i][2] = VD[k][2]
        VecUpDown[k][5] = i // store position of oposto vector
        //VecIndex_I[m][0]
```

```
            }
            if (i == dNC-1 && cont==(dNC/2)) {
                document.getElementById("btn_PolyAlexander").disabled = false;
            } // Este último if funcionou aqui (não sei por que).
            Coloquei esse mesmo if abaixo deste for
            e não funcionou (não sei por que);
        }
    }
</script>

<script>
    function createObject(object, variableName){
        //Bind a variable whose name is the string
        variableName to the object called 'object'
        let execString = variableName + " = object"
        console.log("Running '" + execString + "'");
        eval(execString)
    }
</script>
<!-- ================================================================== -->
<py-script>
    import js
    from js import document
    from js import createObject
    #from pyodide import create_proxy
    from pyodide.ffi import create_proxy
    createObject(create_proxy(globals()), "pyodideGlobals")
    from js import disabled_button

    def PolyAlexander_OnClk():
        import sympy
        from js import VecUpDown
        from js import VD
        from js import DiagramKnot

        from sympy import Symbol
        #from sympy.solvers import solve
        from sympy.matrices import Matrix

        disabled_button("btn_PolyAlexander","false")
```

```
NC = (len(VD)//2)   #NC = number of intersection
of knot diagram; // = integer division
dNC = 2*NC
Var = 1

for k in range(dNC):  #this for create the variables arc of knot
    if VecUpDown[k][0] == VecUpDown[k][2]:
        VecUpDown[k][3] = Var
        #Var start with value = 1. Var is used to
        name the arcs. Ex. var = 'a', then var = 1
        (column 1), etc...
    if VecUpDown[k][1] == VecUpDown[k][2]:
    # if the seconc and third entries of
    VecUpDown are equals, means there was
    an intersection in the segment, and
        Var = Var+1
        # we must name a new variable. (then var = var+1).
        VecUpDown[k][3] = Var
        # VecUpDown[3] store numbers that
        corresponding the variables (names of arcs),
        1 <--> a, 2 <--> b, 3 <..> c, ...
    if Var == NC+1:
    # as we have NC variables, then var = NC+1,
    means we're back to the first variable, so
    VecUpDown[k][3] = 1
        VecUpDown[k][3] = 1
# Next get the node matrix and calculate its determinant
KnotMatrixAux = []
for i in range(NC):   #s quare matrix NC x NC
filled with zeros
    KnotMatrixAux.append([0 for j in range(NC)])

t = sympy.Symbol('t')   # symbolic variable t
J = sympy.Symbol('J')   # symbolic variable J,
Li = -1                 # used to representate
the i-line of matrix KnotMatrixAux

for i in range(dNC):            # MatrixNo[i][j],
line i, column j
```

```
        if VecUpDown[i][4] != 0:    # VecUpDown[i][4]
        store the orientation +1 or -1. VecUpDown[i][4]
        is used to search or verify the not used vector
            Li = Li+1               # Note that there
            are NC lines, but the for "for" go from 0 to dNC.
            # Quando o vetor oposto é usado VecUpDown[i][4]
            do oposto continua zero. e no futuro, pulamos
            este vetor, devido ao fato de ter sido usado.
            op = VecUpDown[i][5]
            # retorna posicao do vetor oposto
            if VecUpDown[i][0] == VecUpDown[i][2]:
            #v[1]=v[3]
                if (VecUpDown[i][4] < 0):
                    KnotMatrixAux[Li][VecUpDown[i][3] -1] =
                    t-1   + (KnotMatrixAux[Li][VecUpDown[i][3] -1])
                    KnotMatrixAux[Li][VecUpDown[op][3]-1] = 1
                    KnotMatrixAux[Li][VecUpDown[op][3]-2]
                    = -t    + (KnotMatrixAux[Li][VecUpDown[op][3]-2])
                else:  #quer dizer que VecUpDown[i][4] > 0
                    KnotMatrixAux[Li][VecUpDown[i][3] - 1]
                    = t - 1 + (KnotMatrixAux[Li][VecUpDown[i][3] - 1])
                    KnotMatrixAux[Li][VecUpDown[op][3] - 1]
                    = -t + (KnotMatrixAux[Li][VecUpDown[op][3] - 1])
                    KnotMatrixAux[Li][VecUpDown[op][3] - 2] = 1
            if VecUpDown[i][1] == VecUpDown[i][2]:  #v[2]=v[3]
                if VecUpDown[i][4] < 0:
                    KnotMatrixAux[Li][VecUpDown[i][3] - 1]
                    = -t + (KnotMatrixAux[Li][VecUpDown[i][3] - 1])
                    KnotMatrixAux[Li][VecUpDown[i][3] - 2]= 1
                    KnotMatrixAux[Li][VecUpDown[op][3] - 1]
                    = t-1 + (KnotMatrixAux[Li][VecUpDown[op][3] - 1])
                else:   # quer dizer que VecUpDown[i][4] > 0
                    KnotMatrixAux[Li][VecUpDown[i][3] - 1] = 1
                    KnotMatrixAux[Li][VecUpDown[i][3] - 2]
                    = -t + (KnotMatrixAux[Li][VecUpDown[i][3] - 2])
                    KnotMatrixAux[Li][VecUpDown[op][3] - 1]
                    = t - 1 + (KnotMatrixAux[Li][VecUpDown[op][3] - 1])
    #print(KnotMatrixAux)
    KnotMatrix = []
    for i in range(NC-1):   #matrix quadrada de ordem
```

```
= (num. cruzamentos -1), preenchida com zeros,
pois tem-se que deletar uma linha e uma coluna
    KnotMatrix.append([0 for j in range(NC-1)])

for i in range(NC-1):
    for j in range(NC-1):
        KnotMatrix[i][j] = KnotMatrixAux[i][j]
#print('Matriz do Nó Final = ', KnotMatrix) #ver matriz do nó
ma = sympy.Matrix(KnotMatrix)
p_t = ma.det()
st = str(p_t)   #transforma p_t em uma string
if  st == 'nan':   #se p_t = nan, é porque ocorreu
erro no cálculo do determinante, e assim aplicar o
método LU para obter o determinante
    p_t = ma.det(method='lu').simplify()  #usa o
    méthodo LU para o calc. do determinante
    p_t = p_t.expand()   #para expandir o polinômio,
    'lu' retorna polinômio sem simplificar

# Aqui proc para obter o polinômio de alexander a
partir de P(t) = determinante da matrix quadrada de ordem (n-1)
if p_t.subs(t, 1) == -1:  #para garantir que P(1)=1.
    p_t = -p_t
p1 = (t**J)*p_t
p2 = p1.subs(t, 1/t)
p3 = p1-p2
i = 1
q = 1
if p_t == 1:
    display("Determinante do nó: D = " + str(1),
    target="demo", append=True)
    display("Polinômio de Alexander: p(t) = " + str(1),
    target="demo", append=True)
if p_t != 1:
    while q != 0:   #intercalando j = -i e j = i,
    até encontrar polinomio de alexander
        p4=(p3.subs(J, -i)).simplify()
        if p4 == 0:
            P = p1.subs(J,-i)
            display("Determinante do nó: D = " +
```

```
                    str(abs(P.subs(t, -1))), target="demo", append=True)
                    display("Polinômio de Alexander: p(t) =
                    " + str(P.expand()), target="demo", append=True)
                    break
                p4 = (p3.subs(J, i)).simplify()
                if p4 == 0:
                    P = p1.subs(J, i)
                    display("Determinante do nó: D =
                    " + str(abs(P.subs(t, -1))), target="demo", append=True)
                    display("Polinômio de Alexander:
                    p(t) = " + str(P.expand()), target="demo", append=True)
                    break
                i = i+1
        return ma
</py-script>
</body>
</html>
```