Aproximando Superfícies em Alta Dimensão

Arthur Garcia Tonus
arthur.tonus@usp.br

João Víctor de Melo
joaovictordemelo@ufmg.br

Paulo Andrés De Almeida
dealmeidapaulo@dc.uba.ar

January 2024

Resumo

Dada uma aplicação diferenciável $F\colon \mathbb{R}^n \to \mathbb{R}^k$ tal que $0 \in \mathbb{R}^k$ é um valor regular de F, temos que o conjunto de nível $\mathcal{M} = F^{-1}(0)$ é uma variedade diferenciável de dimensão $n-k$. Nosso objetivo é construir um algoritmo que aproxime a variedade $\mathcal{M}$ para elevados valores de n e k, buscando evitar dependência exponencial. Serão utilizados métodos de triangulação de hipercubos, em particular CFK, aliados a técnicas de combinatória e contagem para gerar uma estrutura chamada *esqueleto combinatório* da aproximação da variedade.

1 Introdução

Nas disciplinas de Cálculo e Geometria Analítica, nos familiarizamos com curvas e superfícies de nível. Por exemplo, uma circunferência dada por $\{(x,y) \in \mathbb{R}^2 \mid x^2+y^2-1=0\}$ pode ser pensada como uma curva de nível correspondente ao valor 0 da função

$$(x,y) \mapsto x^2+y^2-1.$$

Analogamente, um hiperboloide de uma folha é expresso por $\{(x,y,z) \in \mathbb{R}^3 \mid x^2+y^2-z^2-1=0\}$ é uma superfície de nível da função

$$(x,y,z) \mapsto x^2+y^2-z^2-1.$$

Podemos tomar conjuntos de nível de aplicações diferenciáveis, quaisquer que sejam as dimensões de seu domínio e contradomínio. Quando as dimensões são elevadas, a intuição sobre a geometria do objeto pode ser prejudicada. Dessa forma, desejamos elaborar um programa que permite a visualização de conjuntos dados por $F^{-1}(0)$, para $F\colon \mathbb{R}^n \to \mathbb{R}^k$, quaisquer que sejam $n \geq k$. Iniciamos nossos estudos nos algoritmos *Marching Simplex* e *Continuation Simplex* detalhadamente explicados em [3], e nos aprofundamos no algoritmo do tipo *Marching Simplex* apresentado em [2], do qual implementamos a nossa versão. Nosso código gera um arquivo de saída elaborado no modelo requerido pelo renderizador [1], elaborado por Gabriel Scalet Bicalho, de modo que podemos inseri-lo neste programa para visualizar projeções do conjunto de nível no espaço tridimensional.

A essência do *Marching Simplex* consiste em dividir o espaço em hipercubos, que são divididos em formas mais simples, chamadas *simplexos*. Por exemplo, no plano, dividiriamos uma região retangular em quadrados e cada quadrado em triângulos. No espaço tridimensional, dividiriamos uma região paralelepipédica em cubos e cada cubo em tetraedros. Posteriormente percorremos todos os simplexos em busca de pontos que aproximam a variedade, para enfim gerar uma estrutura de relações de adjacência entre os pontos encontrados, chamada *esqueleto combinatório*, que contém as informações topológicas do conjunto de nível.

O problema dessa abordagem é que, no geral, muitos simplexos desnecessários serão percorridos. Em dimensões elevadas, esse problema se traduz em um tempo de computação inviável ou até mesmo impossível. Uma solução para isso é o *Continuation Simplex*, que, a partir de um hipercubo inicial dado, percorre apenas os hipercubos adjacentes a ele, seguindo apenas pelos hipercubos adjacentes nos quais foram encontrados pontos de aproximação. Porém, ao elevarmos um pouco mais a dimensão do domínio, percorrer um hipercubo inteiro permanece sendo um problema, uma vez que, na divisão que fazemos, cada hipercubo de dimensão n é formado por $n!$ simplexos.

Buscando, entre outros objetivos, remover a dependência fatorial em n, os autores de [2] desenvolveram um algoritmo que possibilita o percorrimento apenas dos simplexos adjacentes, em contraste com os hipercubos adjacentes. Nosso trabalho se resume à implementação de uma nova versão do referido algoritmo.

2 Desenvolvimento teórico

2.1 Noções de variedades suaves

Nesta seção, foi seguida a referência [4] para definições e resultados básicos sobre *variedades suaves*. Esse objeto generaliza as noções de curvas e superfícies regulares, logo é o que precisamos para trabalharmos com "superfícies de dimensão arbitrária".

Durante todo o texto, trabalharemos com espaços euclidianos, ou seja, vamos sempre considerar a topologia usual em $\mathbb{R}^n$ e a topologia herdada da usual em subconjuntos de $\mathbb{R}^n$. Começando pelo caso conhecido do Cálculo, sejam $U \subset \mathbb{R}^n$, $V \subset \mathbb{R}^k$ abertos. Uma aplicação $f : U \to V$ é chamada *aplicação suave* se todas as suas derivadas parciais $\partial^n f / \partial x_{i_1} \cdots x_{i_n}$ de qualquer ordem existem e são contínuas.

Generalizando esse conceito, sejam $X \subset \mathbb{R}^n$, $Y \subset \mathbb{R}^k$ subconjuntos quaisquer. Uma aplicação $f : X \to Y$ é chamada *aplicação suave* se para todo $x \in X$ existe um aberto $U \subset \mathbb{R}^n$ com $x \in U$ e uma aplicação suave $F : U \to \mathbb{R}^k$ que coincide com f em $U \cap Y$. Desta definição, é imediato que, se $f : X \to Y$ e $g : Y \to Z$ são suaves, então $g \circ f : X \to Z$ é suave.

Diremos que uma aplicação $f : X \to Y$ é um *homeomorfismo* se é contínua e se possui inversa também contínua. Se, além de ser homeomorfismo, tivermos que f e f^{-1} são suaves, dizemos que f é um *difeomorfismo* e que X é difeomorfo a Y. Podemos agora definir o conceito de *variedade suave*.

Definição 2.1.1. Um subconjunto $M \subset \mathbb{R}^n$ é uma *variedade suave de dimensão* m se cada $x \in M$ admite uma vizinhança aberta $W \cap M$ que é difeomorfa a um aberto U de $\mathbb{R}^m$.

Se M é uma variedade suave de dimensão m e $g : U \to W \cap M$ é um difeomorfismo, chamaremos g de *parametrização* e g^{-1} de *sistema de coordenadas*. De agora em diante, quando escrevermos *variedade*, leia-se *variedade suave*.

Outros dois conceitos importantes são os de ponto e valor regulares de uma aplicação. Dada uma aplicação suave $F : X \subset \mathbb{R}^n \to \mathbb{R}^k$, dizemos que $x \in X$ é um *ponto crítico* de F se a matriz jacobiana $JF(x)$ no ponto x tem posto menor do que k, e nesse caso chamamos $F(x)$ de *valor crítico*. Se $x \in X$ não é ponto crítico, dizemos que é um *ponto regular*, enquanto que se $y \in \mathbb{R}^k$ não é um valor crítico, dizemos que é um *valor regular*.

No geral, não é usual encontrarmos parametrizações ou sistemas de coordenadas para exibir que um dado conjunto é uma variedade suave. O teorema abaixo facilita muito a obtenção de exemplos de variedades, sendo que seu uso para apresentar um exemplo de variedade é bastante comum. Efetivamente, as variedades com as quais trabalharemos são todas do tipo exibido abaixo.

Teorema 2.1.2. *Seja $F\colon X \subset \mathbb{R}^n \to \mathbb{R}^k$ suave. Se $y \in \mathbb{R}^k$ é um valor regular de F, então $F^{-1}(y) \subset \mathbb{R}^n$ é uma variedade suave de dimensão $n - k$.*

Nas hipóteses do teorema acima, dizemos que $F^{-1}(0)$ é uma variedade definida implicitamente. Exibamos dois exemplos de objetos desse tipo.

A função $F\colon \mathbb{R}^3 \to \mathbb{R}$ dada por $F(x,y,z) = x^2+y^2-z^2-1$ tem 0 como valor regular. De fato JF, que pode ser visto como o gradiente $\nabla F = \left(\frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}, \frac{\partial F}{\partial z}\right)$, é não sobrejetora se, e somente se, $2x = 2y = -2z = 0$, isto é, $x = y = z = 0$. Neste caso, teremos $F(0,0,0) = -1$, portanto o único valor crítico é -1. Isso mostra que o subconjunto $F^{-1}(0)$, um *hiperboloide de uma folha*, é uma variedade suave de dimensão 2.

A aplicação $F\colon \mathbb{R}^5 \to \mathbb{R}^3$ dada por

$$F(x,y,z,\theta,\omega) = \begin{pmatrix} x - \left(3 + \cos\frac{\theta}{2}\sin\omega - \sin\frac{\theta}{2}\sin\omega\right)\cos\theta \\ y - \left(3 + \cos\frac{\theta}{2}\sin\omega - \sin\frac{\theta}{2}\sin\omega\right)\sin\theta \\ z - \sin\frac{\theta}{2}\sin\omega - \cos\frac{\theta}{2}\sin\omega \end{pmatrix}$$

é uma submersão, ou seja, todo valor é regular. Assim, o conjunto $F^{-1}(0) \subset \mathbb{R}^5$ é uma variedade suave de dimensão 2 em $\mathbb{R}^5$. A projeção dessa variedade em $\mathbb{R}^3$ obtida ao descartarmos as duas últimas coordenadas θ e ω é uma figura que chamamos de *bagel de Klein*, uma das visualizações possíveis da garrafa de Klein.

2.2 Triangulação CFK e vértices de aproximação

Comecemos com a discretização do domínio. Para isso, precisamos do conceito de simplexo. Intuitivamente, um simplexo de dimensão k é a menor forma geométrica convexa de dimensão k que se pode formar. Formalmente, um *simplexo de dimensão k* é o conjunto $\left\{v \in \mathbb{R}^n \,\middle|\, v = \sum_{i=0}^k \lambda_i v_i,\ \sum_{i=0}^k \lambda_i = 1 \text{ e } \lambda_i \geq 0\right\}$, sendo $v_0, \ldots, v_k \in \mathbb{R}^n$ tais que o conjunto $\{v_1 - v_0, \ldots, v_k - v_0\}$ é linearmente independente. Denotamos esse simplexo por $[v_0, \ldots, v_k]$.

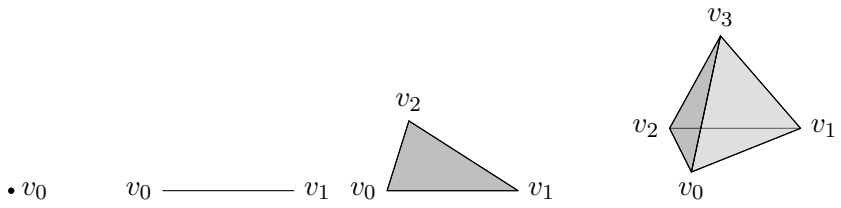

Figura 1: Simplexos de dimensão 0, 1, 2 e 3.

Com isso, podemos definir precisamente o conceito de triangulação. Dizemos que uma coleção $\mathcal{C}$ de simplexos é uma *triangulação* de um conjunto $D \subset \mathbb{R}^n$ se

- $D = \bigcup_{\sigma \in \mathcal{C}} \sigma$;
- $\sigma_1, \sigma_2 \in \mathcal{C}$ implica $\sigma_1 \cap \sigma_2 \in \mathcal{C}$ ou $\sigma_1 \cap \sigma_2 = \varnothing$;
- todo subconjunto compacto de H intercecta um número finito de simplexos de $\mathcal{C}$.

Nosso objetivo é dividir um domínio compacto retangular de $\mathbb{R}^n$ em hipercubos, depois dividir cada hipercubo em simplexos, criando uma triangulação desse domínio. Comecemos pensando no hipercubo unitário $[0,1]^n \subset \mathbb{R}^n$; tome $\pi\colon \{1,\ldots,n\} \to \{1,\ldots,n\}$ uma permutação dos primeiros n naturais. Defina $v_0 = (0,\ldots,0)$ e

$$v_i = v_{i-1} + e_{\pi(i)},\ 1 \leq i \leq n \tag{1}$$

sendo $e_1, \ldots, e_n$ os vetores da base canônica de $\mathbb{R}^n$. Temos que os pontos $\{v_0, \ldots, v_n\}$ geram um simplexo de dimensão n contido em $[0,1]^n$, que denotamos por S_π.

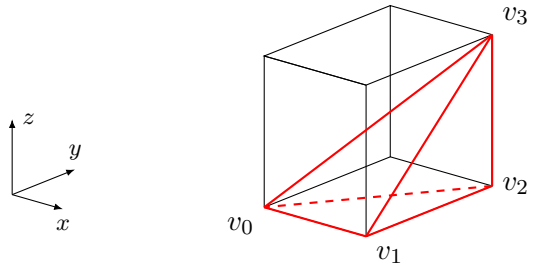

Figura 2: Simplexo de dimensão 3 gerado pela permutação identidade.

Afirmamos que a coleção de todos os S_π, fechada por interseções finitas, é uma triangulação do hipercubo $[0,1]^n$ (o produto cartesiano de intervalos fechados). Ainda, note que podemos triangular outros hipercubos ao escolher outro v_0 e escalar convenientemente, sendo essa triangulação resultando em $n!$ simplexos para cada hipercubo. Essa triangulação de um hipercubo individual é conhecida como *triangulação CFK* [3].

Faremos a triangulação de um domínio da forma $D = \prod_{i=1}^{n}[a_i, b_i]$ da seguinte maneira: dividimos D em hipercubos de mesmo tamanho e, em cada hipercubo, realizamos a triangulação CFK (sem reflexões, apenas transladando a triangulação de um hipercubo para o próximo). A triangulação de D resultante é chamada *triangulação K1*.

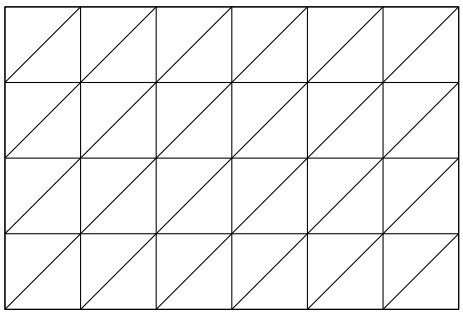

Figura 3: Exemplo de triangulação K1 em uma região de $\mathbb{R}^2$.

Feita a triangulação, precisamos encontrar os vértices de aproximação. Inicialmente, a título de exemplificação tome o intervalo aberto (a, b), de tal forma que possamos dizer que $P = a + (b - a) \cdot \lambda$ estará no interior do intervalo fechado $[a, b]$ se, e somente se λ é um valor entre 0 e 1. Agora quando ocorre o fato de que λ seja maior que 1, de fato, teremos tal ponto P fora do intervalo fechado $[a, b]$, e note que isso pode ser generalizado. Tome agora $\mathbb{R}$, note que no caso de um triângulo em particular, teremos, usando coordenadas baricêntricas o seguinte. Considere um triângulo definido por 3 vértices v_0, v_1 e v_3, e peguemos um ponto arbitrário P desse triângulo. Sabe-se que $P = \lambda_1 v_1 + \lambda_2 v_2 + \lambda_3 v_3$, em que a soma dos λ's vale 1, e vale que se algum λ for negativo, de fato, teremos o ponto fora do triângulo. De fato, caso tivéssemos um número negativo para algum dos λ's, de forma geral, o ponto em consideração não estará dentro da região considerada. Ainda, observemos que os vértices de aproximação devem ser ter as seguintes informações: sua coordenada e o rótulo dos vértices da aresta na qual tal vértice se encontra.

Generalizando as exemplificações acimas, a interseção da variedade com um simplexo de dimensão k, gerado por $\{v_0, \ldots, v_k\}$ é aproximada ao linearizar a aplicação F. Isso significa que, dado um ponto $v = \sum_0^k \lambda_i v_i$ neste simplexo, consideramos a aplicação $F_T(v) = \sum_{i=0}^k \lambda_i F(v_i)$,

lembrando que $\sum_{i=0}^{k} \lambda_i = 1$. Dessa forma, precisamos encontrar os coeficientes λ_i de modo que

$$\begin{bmatrix} 1 & \cdots & 1 \\ F(v_0) & \cdots & F(v_k) \end{bmatrix} \begin{bmatrix} \lambda_0 \\ \vdots \\ \lambda_k \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}. \tag{2}$$

Como queremos um algoritmo eficiente para dimensões altas, não é viável armazenarmos os valores de F em todos os pontos da discretização em uma estrutura de dados, uma vez que teríamos um gasto excessivo de memória. O que fazemos é uma rotulação de cada vértice com um número inteiro, processo explicado com detalhes em 2.3.

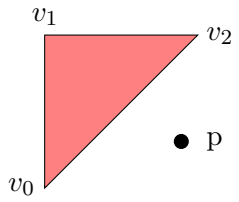

Figura 4: Se existe um i tal que λ_i não está no intervalo $[0, 1]$, a solução encontrada não está dentro do simplexo.

2.3 Enumeração dos vértices

Para visualizar os zeros de uma função em um conjunto compacto D contido em $\mathbb{R}^n$, utilizamos uma discretização do domínio. Após definir como dividir um hipercubo em $n!$ simplexos, devemos introduzir como vamos dividir o domínio em hipercubos.

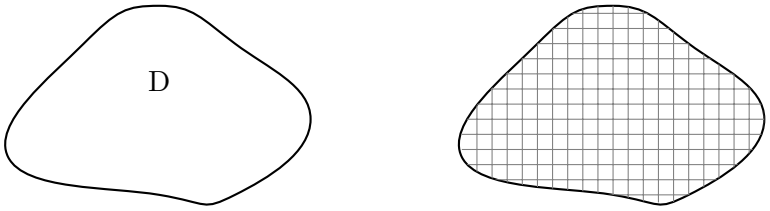

Figura 5: Divisão do domínio compacto em hipercubos, no caso $n = 2$.

Vamos trabalhar com funções F definidas sobre um domínio

$$D = [a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_n, b_n]$$

Limitar-nos ao caso em que F está definida em um hipercubo facilitará a divisão do domínio. Posteriormente, poderíamos estender para domínios compactos mais gerais utilizando noções de mudança de variáveis.

Dependendo da função, desejaremos que a quantidade de divisões em uma dimensão seja maior do que em outra. Por exemplo, a função $F(x, y) = x^2$ é constante em um de seus parâmetros. Portanto, será interessante maximizar a quantidade de divisões na primeira coordenada de seu domínio.

No entanto, aumentar a quantidade de divisões implica em aumento do tempo de computação. Na prática, a quantidade de divisões k_i em que cada intervalo $[a_i, b_i]$ será particionado é um dos parâmetros necessários em nosso algoritmo. Em seguida, para cada $i \leq n$, devemos considerar $k_i + 1$ pontos no intervalo $[a_i, b_i]$. Poderíamos escolher diferentes métodos para selecioná-los; no entanto, optamos por considerá-los equidistantes.

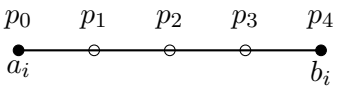

Figura 6: Segmento $[a_i, b_i]$ com $k_i = 4$ puntos equidistantes.

Dessa forma, podemos pensar em cada intervalo $[a_i,\ b_i]$ como a união de subintervalos

$$[a_i,\ b_i] = \bigcup_{j=1}^{k_i} [p_{(i,\ j-1)},\ p_{(i,\ j)}]$$

Onde os pontos $p_{(i,j)}$ são equidistantes no intervalo $[a_i,\ b_i]$ considerado. Havendo definido k_i subintervalos em cada dimensão i, podemos construir hipercubos utilizando produtos cartesianos. Cada hipercubo será o resultado do produto cartesiano de n subintervalos $[p_{(i,\ j_i)},\ p_{(i,\ j_i+1)}]$. Podemos descrever cada hipercubo pelas primeiras coordenadas $p_{(i,\ j_i)}$ em cada dimensão. Mais ainda, podemos descrever de forma única um hipercubo, conhecendo apenas $(j_1 \dots j_n)$, os inteiros que referenciam cada subintervalo do produto cartesiano. Chamaremos esses rótulos $(j_1, \dots, j_n)$ as coordenadas na malha do vértice $(p_{(1,j_1)}, \dots, p_{(n,j_n)})$.

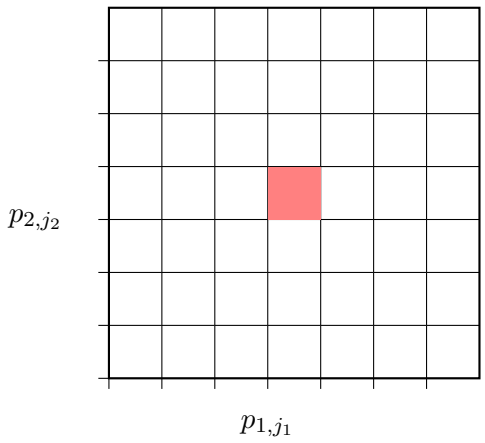

Figura 7: Hipercubo $[p_{(1,\ j_1)}, p_{(1,\ j_1+1)}] \times [p_{(2,\ j_2)}, p_{(2,\ j_2+1)}]$

Então, cada hipercubo é representado por n números naturais, cada um no intervalo de 0 a $k_i - 1$. Podemos reduzir isso a apenas um número natural, utilizando conceitos de aritmética modular. Dessa forma, não precisamos salvar cada uma das coordenadas do hipercubo. Um único número natural, menor do que a quantidade total de hipercubos, é suficiente para referenciar unicamente cada um deles.

Para isso, vamos definir as seguintes bases naturais

$$b_j = \prod_{i=1}^{n-j} k_i$$

onde j é um índice natural que varia de 1 a n e cada k_i é o número de divisões do domínio na dimensão i-ésima.. Essas bases serão coeficientes que relacionarão um número de hipercubo n_H com suas respectivas coordenadas na malha $(j_1, \dots, j_n)$.

Considere n_H, um número natural menor que a quantidade de hipercubos na triangulação. Em outras palavras, sabemos que n_H é menor que $\prod_{i=1}^{n} k_i$. Portanto, podemos pensar a n_H como a soma de um produto mais um resto

$$n_H = b_1 \cdot j_n + r_1,$$

onde j_n é um número entre zero e $k_n - 1$, e r_1 está entre zero e $b_1 - 1$.

Além disso, podemos escrever r_1 como a soma de um produto mais um resto

$$r_1 = b_2 \cdot j_{n-1} + r_2.$$

onde j_{n-1} é um número entre zero e $k_{n-1} - 1$, e r_2 está entre zero e $b_2 - 1$.

Em geral, podemos realizar o mesmo processo para cada r_i com i maior ou igual a zero e menor que n

$$r_i = b_{i+1} \cdot j_{n-i} + r_{i+1}.$$

Observamos que, dada a etiqueta de um hipercubo n_H, obtivemos suas coordenadas na malha discreta. Para cada valor de i, sabemos que j_i está entre zero e $k_i - 1$. Além disso, utilizando este método, a representação de n_H em coeficientes $(j_1, \dots j_n)$ é única.

De forma contrária, podemos encontrar a etiqueta natural do hipercubo com coordenadas $(j_1, \dots j_n)$. Devemos fazer um caminho inverso. Multiplicamos cada j_i por sua correspondente base b_{n-i} e somamos todos os valores.

$$n_H = \sum_{i=1}^{n} j_i \cdot b_{n-i}$$

Além disso, dado um hipercubo n_H com coordenadas $(j_1, \dots j_n)$ sabemos qual é seu vizinho n'_H na direção canônica e_k.

$$n'_H = b_{n-k} + \sum_{i=1}^{n} j_i \cdot b_{n-i} = n_H + b_{n-k}$$

Em conclusão, conseguimos simplificar as coordenadas de cada hipercubo para uma enumeração com números naturais. A forma de nomear cada um dos hipercubos é única e sabemos qual é a etiqueta de todos os vizinhos de um determinado hipercubo, sem ter que calcular suas coordenadas na malha.

2.4 Esqueleto Combinatório

Os algoritmos *Marching Simplex* e *Continuation Simplex* apresentados em [3] usam métodos combinatórios como contagem para dividir um domínio em uma malha de hipercubos, em que cada um deles é dividido em simplexos de mesma dimensão. *A posteriori*, geram-se faces de dimensão k para buscar interseção com a variedade em cada uma. Uma vez encontradas as interseções de $F^{-1}(0)$ com as faces dentro de um simplex de dimensão n, será necessário conectar esses vértices de tal maneira a obter um politopo que aproxime a variedade. Portanto, é necessário falarmos um pouco do *esqueleto combinatório*, uma estrutura que contém relações de adjacências entre elementos.

No que tange ao esqueleto combinatório, consideremos inicialmente uma aplicação de $\mathbb{R}^2$ para $\mathbb{R}$. Neste caso, consideremos um quadrado quaisquer. Enumerando seus vértices, teremos dois simplexos, um deles sendo o simplexo $[0, 1, 3]$. Agora, suponhamos que existem vértices em duas das arestas de nosso triângulo, mas não a diagonal. Elas estão contidas nas arestas $[0, 1]$ e $[1, 3]$ que são simplexos de dimensão 1. Então, para este triângulo, note que existem exatamente 2 simplexos de dimensão 1 que contém os vértices mencionados.

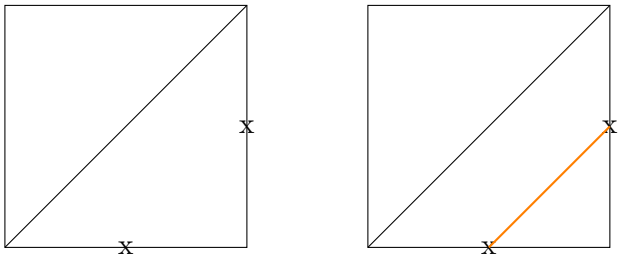

Figura 8: Exemplo de conexões em $\mathbb{R}^2$. Como as faces que contêm os vértices de aproximação são adjacentes, eles devem estar conectadas.

Para determinar se devemos conectar esses vértices, formando uma aresta, devemos analisar as adjacências. Dois politopos estão conectados se os simplexos de dimensão i em que estão pertencem a um mesmo simplex de dimensão $i+1$ na triangulação CFK. Sabe-se que $[0,1,3]$ contém tanto a face $[0,1]$ quanto a face $[1,3]$. Neste caso, as arestas às quais cada vértice de aproximação pertence compartilham o triângulo $[0,1,3]$. Portanto, devem ser conectadas, formando um politopo de dimensão 1.

Agora suponha uma aplicação de $\mathbb{R}^3$ em $\mathbb{R}$, de modo que a variedade terá dimensão 2. Buscamos vértices de aproximação nas arestas dos tetraedros da triangulação (primeiro "bloco" do esqueleto combinatório), e unimos os vértices desde que estejam em arestas adjacentes (segundo bloco), formando segmentos de reta que pertencem às faces triangulares do tetraedro. Como queremos aproximar uma superfície, devemos formar polígonos. Assim, a próxima etapa (terceiro bloco) é inserir quais são os polígonos maximais formados pelos segmentos de reta encontrados na etapa anterior, que estarão contidos nos tetraedros, respeitando a regra que unimos dois segmentos apenas quando eles pertencem a triângulos adjacentes.

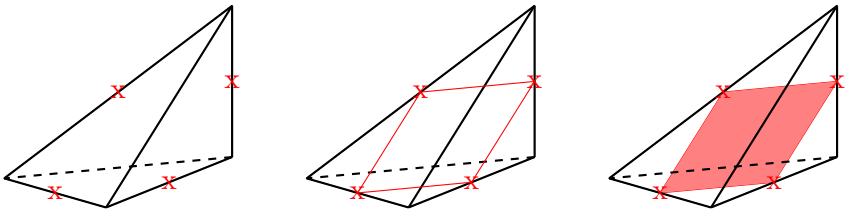

Figura 9: Exemplo de conexões em $\mathbb{R}^3$.

No caso geral de uma aplicação de $\mathbb{R}^n$ em $\mathbb{R}^k$, o procedimento é análogo. Começa-se buscando vértices de aproximação nas faces de dimensão k, depois forma-se arestas nas faces de dimensão $k+1$ e assim por diante, parando quando tivermos politopos de dimensão $n-k$ nos simplexos de dimensão n. Note que devemos sempre formar politopos maximais, isto é, o maior politopo que se pode formar seguinto as regras de adjacência estabelecidas. Em 3.4.2 encontra-se o algoritmo, em pseudocódigo, que usamos para gerar o esqueleto combinatório.

2.5 Cofaces

Até agora, introduzimos métodos para discretizar o espaço e enumerar cada um dos elementos da triangulação com um sistema de enumeração. No entanto, devemos estabelecer como percorreremos o domínio.

Em geral, a quantidade de hipercubos na triangulação cresce exponencialmente junto com a dimensão n. Além disso, a quantidade de simplexos dentro de cada hipercubo cresce de forma fatorial conforme a dimensão aumenta. É por isso que realizar uma aproximação de $F^{-1}(0)$ em todos os hipercubos e em todos os simplexos exigiria um tempo de computação elevado quando se trabalha em dimensões altas. Uma técnica para reduzir o tempo de computação é não calcular

vértices de aproximação para todos os elementos da triangulação. Para isso, será interessante introduzir o conceito de coface.

As cofaces de um simplexo τ de dimensão k são os elementos da triangulação de dimensão $k+1$ que contem a τ. Por exemplo, as cofaces de uma aresta em $\mathbb{R}^2$ são os triângulos na triangulação que contêm essa aresta.

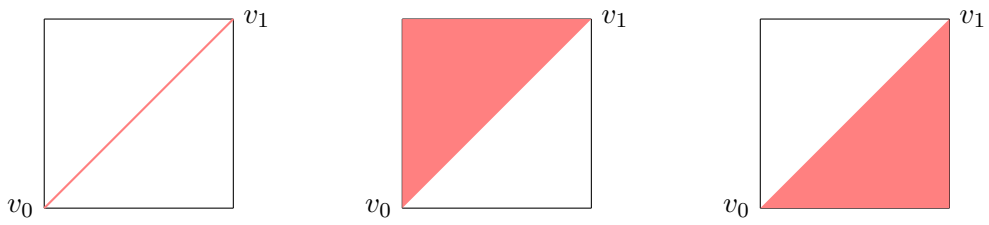

Figura 10: Cofaces de uma aresta em $\mathbb{R}^2$, usando a triangulacão K1.

Agora, suponhamos que queremos calcular as cofaces de um símplice $\tau = [v_0, \ldots, v_k]$. Sabemos que as cofaces pertencem à triangulação K1 e, portanto, devem ser construídas com os mesmos vértices, acrescidos de outro elemento linearmente independente no mesmo hipercubo. Em geral, teremos três maneiras de adicionar um novo vértice a τ.

Primeiramente, poderíamos completar o símplice adicionando um novo vértice no final. Para fazer isso, basta observar cuidadosamente as direções utilizadas na construção de τ. Vale a pena lembrar que os vértices de um símplice na triangulação K1 são gerados de maneira incremental. Se uma direção foi incorporada, não é possível retroceder nela. Dessa forma, o último vértice v_k abrange todas as direções previamente incluídas e, de forma complementar, podemos determinar todas as direções que não foram utilizadas. Portanto, podemos introduzir um novo vértice v_{k+1} que seja igual a v_k mais um conjunto de direções canônicas ausentes em v_k. Se tivermos as direções canônicas presentes em v_k como I:

$$v_k = v_0 + \sum_{i \in I} e_i$$

então definiremos v_{k+1} como um elemento:

$$v_{k+1} = v_k + \sum_{i \in I'} e_i$$

onde I' é um subconjunto não vazio do complemento de I.

De maneira completamente análoga, poderíamos adicionar um elemento antes de v_0. Nesse contexto, estamos assumindo que o símplice de dimensão $k+1$ pertence a um hipercubo vizinho que se inicia em v_0'. Ao somar elementos que não estão presentes em v_k, alcançamos v_0. Nesse cenário, teríamos:

$$v_0 = v_0' + \sum_{i \in I'} e_i$$

onde I' é um subconjunto não vazio do complemento de I.

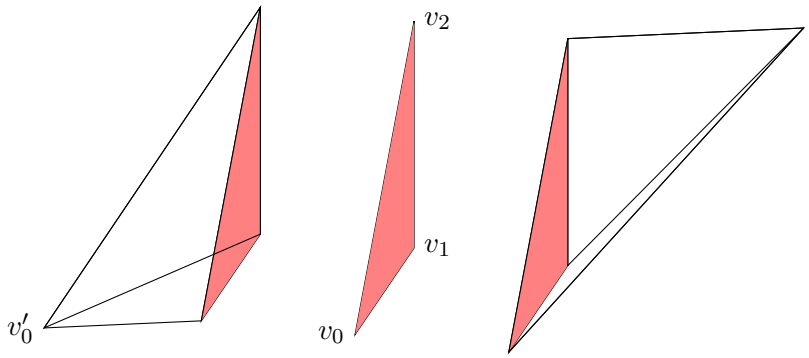

Figura 11: Cálculo de cofaces para $k = 2$ em $\mathbb{R}^3$, usando a triangulação K1. Neste caso, só podem ser adicionados pontos antes e depois do triângulo.

Por fim, teríamos a situação em que o ponto a adicionar está entre dois vértices v_j e v_{j+1}. Neste caso, teremos que incorporar um vértice de tal forma que respeite que os vértices de um simplex na triangulação K1 são gerados incrementalmente. Isso significa que se os canônicos que são adicionados na etapa $j + 1$ formam o conjunto I, teremos que adicionar um vértice v tal que

$$v = v_j + \sum_{i \in I'} e_i$$

onde I' é um subconjunto de I não vazio e diferente de I. Assim, estaremos adicionando ao simplex τ um ponto intermediário entre dois de seus vértices.

3 Especificação e Algoritmo

3.1 Descrição Geral

Retomando nosso problema principal, o objetivo é criar um algoritmo capaz de aproximar um conjunto $F^{-1}(0)$, uma variedade suave, dentro de um hipercubo D. Para isso, supomos que F é uma função que atende às condições do Teorema 2.1.2, com $y = 0$, no domínio D.

A abordagem geral do algoritmo consiste em retirar um elemento de uma fila SQ, verificar se possui algum vértice de aproximação e, em caso afirmativo, adicionar à fila SQ outros elementos adjacentes. Executando esse processo em um loop, percorremos o espaço em busca de vértices de aproximação, movendo-nos apenas nas adjacências dos simplexos que sabemos intersectam a variedade.

Em seguida, para cada hipercubo em que tenha sido encontrado algum vértice de aproximação, geramos politopos de dimensão $n - k$, seguindo o método apresentado na seção 2.4.

O resultado é um politopo de dimensão $n - k$ que aproxima $F^{-1}(0)$ dentro do conjunto D. Considerando os pré-requisitos fornecidos sobre F e D, podemos garantir que o politopo encontrado converge para a variedade conforme aumentamos a quantidade de divisões.

Consideremos, por exemplo, o caso de $\mathbb{R}^2$. Temos uma fila SQ com simplexos de dimensão $k = 1$, ou seja, arestas. De SQ, retiramos a próxima aresta τ, verificamos se já a analisamos antes e calculamos se ela possui um vértice de aproximação. Utilizando uma aproximação piece linear, conseguimos determinar que τ intersecta a variedade $F^{-1}(0)$ em um ponto. Portanto, salvamos o vértice de interseção. Em seguida, calculamos todas as arestas da triangulação que compartilham um triângulo com τ e as adicionamos a SQ. Repetimos o processo para a próxima aresta na fila.

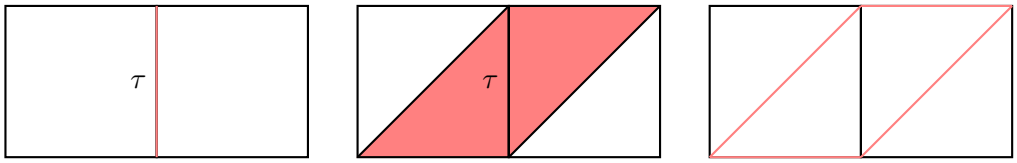

Figura 12: Começamos a procurar interseção em τ. Se encontrarmos, calculamos todas as arestas que compartilham um triângulo. Em seguida, repetimos para cada uma dessas arestas.

3.2 Pré-requisitos é Post-condições

Vamos desejar construir um processo que receba como parâmetro um hipercubo D em $\mathbb{R}^n$ e uma função F definida sobre D. Além disso, precisaremos de n valores naturais, uma tupla em $\mathbb{N}^n$, que represente a quantidade de divisões em cada uma das dimensões. Por fim, requeremos um simplex τ_0 na triangulação K1.

As condições para F são aquelas mencionadas no Teorema 2.1.2. Neste caso, é necessário que F seja de classe C^∞ para todos os elementos no domínio D. Além disso, zero deve ser um valor regular de F. Desta forma, podemos garantir que $F^{-1}(0)$ será uma variedade suave de dimensão $n - k$.

Para o simplexo inicial τ_0, será interessante que ele intersecte a variedade unicamente em um ponto em seu interior.

No caso da quantidade de divisões para cada dimensão $(K_1, \dots K_n)$, é suficiente que todos sejam valores positivos maiores que dois.

Dadas essas condições, podemos garantir que será retornado um politopo de dimensão $n - k$ que aproxima uma das componentes conexas de $F^{-1}(0)$.

3.3 Estrutura de dados

Para formular nosso algoritmo, será necessário definir como cada uma das entidades mencionadas será representada. O algoritmo geral será dividido em duas partes. Então, definiremos as estruturas necessárias para cada uma das partes separadamente. Inicialmente, apresentaremos cada estrutura de dados usada para calcular os vértices de aproximação. Em seguida, introduziremos aquelas usadas para calcular o Esqueleto Combinatório.

Dado um simplexo τ de dimensão k, o representaremos com base no número do hipercubo n_H ao qual pertence e em suas coordenadas dentro desse hipercubo, conforme detalhado na seção 2.2 e 2.5. Assim, podemos pensar em τ como uma tupla de inteiros com $k + 1$ elementos. Sua primeira coordenada representa o hipercubo n_H ao qual pertence, enquanto as coordenadas restantes são rótulos para os vértices do hipercubo n_H que compõem τ.

No entanto, para um simplexo de dimensão k, não existe um único hipercubo que o contenha. Para evitar lidar com múltiplas representações de um mesmo simplexo, definimos n_H como o hipercubo que inclui todos os vértices de τ, sendo que o menor deles tem rótulo 0. Dizemos que τ está em formato canônico se respeitar essa notação. Observamos que a segunda coordenada de um simplexo em formato canônico é sempre igual a zero.

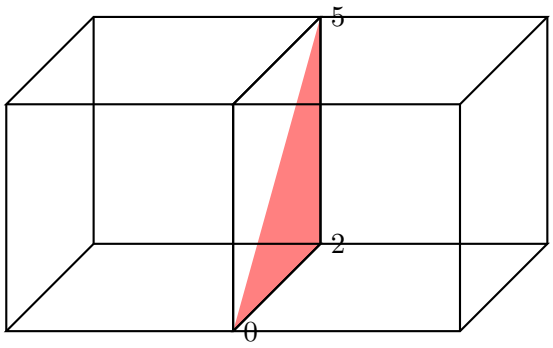

Figura 13: Neste exemplo, o triângulo poderia pertencer a qualquer um dos dois cubos. No entanto, para estar em forma canônica, seu primeiro vértice deve ser igual a zero. Portanto, o representaremos como um símplice do hipercubo da direita.

Em seguida, criaremos duas estruturas para acompanhar quais simplexos já foram percorridos e quais devem ser visitados posteriormente. No primeiro caso, definimos um conjunto S que armazena dados do tipo simplex. Para saber quais simplexos ainda precisam ser revisados, definimos uma fila SQ.

Por fim, criamos dois dicionários para armazenar os vértices já calculados. No caso de encontrar um vértice de aproximação, suas coordenadas serão armazenadas em formato de tupla em um dicionário V, que tem chaves do tipo simplexo. Por outro lado, para armazenar a qual hipercubo pertence cada um dos simplexos, utilizamos o dicionário H. Este último tem como chave um número de hipercubo, inteiro, e armazena uma lista com dados do tipo simplex.

Para o Esqueleto Combinatório, precisaremos definir três estruturas. Inicialmente, criaremos um array CS com $n-k$ elementos. Cada posição i de CS será uma lista de conjuntos, onde serão armazenados os elementos de dimensão $i-1$ que estão conectados, identificados pelos seus índices naturais. Em outras palavras, a informação armazenada em CS será o próprio Esqueleto Combinatório. Um conjunto de vértices se conecta se pertencerem ao mesmo conjunto em $CS[0]$. Da mesma forma, arestas formarão um polígono se pertencerem ao mesmo conjunto em $CS[1]$.

Além disso, definiremos dois dicionários, $Faces$ e $Cofaces$. Ambos associam um simplexo a um índice único. Eles serão usados para determinar quais simplexos são adjacentes na triangulação CFK.

3.4 Algoritmo

O algoritmo descreve o processo principal para a aproximação do conjunto $F^{-1}(0)$ dentro de um dominio D usando a triangulação K1. A abordagem geral é explorar os simplexos e suas facetas na triangulação por meio de uma fila SQ. Inicialmente, retiramos um simplexo τ da fila e verificamos suas cofaces. Para cada cofaceta ϕ, calculamos o número do hipercubo n_H ao qual pertence e inserimos τ nesse hipercubo dentro de H. Em seguida, examinamos as facetas σ de ϕ. Se uma faceta não está na lista S e intersecta $F^{-1}(0)$, adicionamos essa faceta à fila SQ, à lista S e salvamos seus vértices de aproximação em V. Esse processo é repetido até que a fila SQ esteja vazia. Após essa etapa, calculamos o Esqueleto Combinatório CS utilizando a função EsqueletoCombinatorio com base nos simplexos armazenados nos hipercubos em H. Esse esqueleto é fundamental para compreender a conectividade entre os simplexos na triangulação CFK, formando a base para a construção dos politopos que aproximam a variedade suave $F^{-1}(0)$.

Algoritmo 3.4.1 Algoritmo principal

```
1: while SQ not empty do
2:     Pop a simplex τ from SQ
3:     for each cofacet φ of τ do
4:         m ← Hypercube number of φ
5:         Insert τ into H[m]
6:         for facet σ of φ do
7:             Insert σ into S
8:             if σ does not lie in S and intersects M then
9:                 Insert σ into SQ
10:                Save AproxVertices(σ) in V
11: CS ← combinatrialSkeleton(H)
```

A maneira de calcular cofaces é conforme descrito na 2.5, dividida em três casos. No primeiro caso, poderíamos adicionar um novo vértice ao final do simplexo. Isto é, dado um certo simplexo $(n_H, 0, v_1, \dots v_k)$, considerando a ordem crescente dos vértices na triangulação CFK, onde existem conjuntos $\emptyset = I_0 \subset I_1 \subset I_2 \dots \subset I_k \subset [n]$ tal que

$$v_j = \sum_{i \in I_j} 2^i$$

podemos ver quais canônicos não estão presentes no último vértice v_k, e adicionar um subconjunto destes. Descartando o conjunto vazio, cada subconjunto distinto será um vértice distinto. Portanto, existem $2^m - 1$ cofaces possíveis nesse primeiro caso, onde m é a quantidade de elementos que não estão presentes no último vértice v_k.

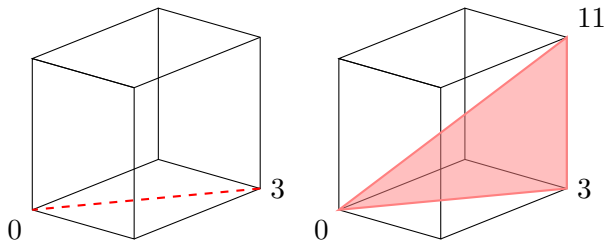

Figura 14: Por exemplo, em $\mathbb{R}^3$, sabemos que o vértice 3 é composto pelos elementos e_1 e e_2 e, portanto, só poderíamos adicionar o elemento e_3. A coface é completada com o elemento com rotulo 11.

A maneira de calcular explicitamente cada uma dessas cofaces é conhecer o conjunto I de direções canônicas não presentes em v_k. Para isso, sabendo apenas o rótulo inteiro do vértice v_k dentro do hipercubo, podemos calcular

$$w = \sum_{i \in [n] \setminus I_k} 2^i$$

Este valor contém, em sistema binário, todas as direções canônicas que não estão presentes na construção do simplex. Em seguida, para conhecer cada uma dessas direções, podemos utilizar a parte inteira do logaritmo na base dois

$$i_1 = \lfloor \log_2(w) \rfloor$$

Dessa forma, obtivemos uma primeira direção. Para encontrar as seguintes $n - m$ direções, devemos subtrair o número obtido:

$$i_j = \lfloor \log_2(w - \sum_{l=1}^{j-1} 2^{i_l}) \rfloor$$

Observamos que esta última expressão contém o caso $j = 1$.

Por tanto, o conjunto I de todas as direções canônicas não incluídas na construção do simplex será

$$I = \{i_j \mid 0 \leq j \leq n - m\}$$

Finalmente, os possíveis vértices v que poderiam completar o simplex de dimensão k, para obter uma coface, serão da forma

$$v = v_k + \sum_{i \in I'} 2^i$$

onde I' é um conjunto não vazio do conjunto das partes de I. Para obter cada elemento I' do conjunto das partes de I, podemos utilizar um número α entre 1 e 2^{n-m}, onde i_j pertence a I' quando 2^{i_j} faz parte da expansão binária de α. A quantidade de cofaces calculadas dessa maneira, adicionando um vértice ao final, será então $2^{n-m} - 1$. Podemos limitar esse número por 2^{n-k}, considerando que a quantidade de elementos que é somada em cada I_j é pelo menos um.

De forma análoga, podemos continuar com o caso em que, para completar a *coface*, um vértice é adicionado antes do simplex $(n_H, 0, v_1, \ldots v_k)$. Nesse caso, também precisamos encontrar quais direções não estão presentes em v_k, procurar todos os seus subconjuntos e calcular os rótulos no hipercubo. No entanto, tendo calculado I no caso anterior, não é necessário calcular novamente. Em seguida, para cada rótulo, em vez de adicionar um certo v ao final do simplexo, vamos inserir um $-v$ no início. Desta forma, respeitamos a ordem aditiva da triangulação CFK. No entanto, para respeitar nossa representação de simplexos, devemos colocá-lo no formato canônico. Isso significa que devemos encontrar um n'_H tal que inclua todos os vértices de $(n_H, -v, 0, v_1, \ldots v_k)$, sendo que o menor deles tem rótulo zero. Devido a como enumeramos os hipercubos, podemos fazer isso seguindo o que foi mencionado na seção 2.3. A quantidade de cofaces calculadas dessa maneira, adicionando um vértice ao início do simplexo, também será $2^{n-m} - 1$.

No último caso, devemos calcular as cofaces do simplexo $(n_H, 0, v_1, \ldots v_k)$, adicionando um vértice entre v_j e v_{j+1}, para algum j. Para isso, devemos mencionar novamente a ordem crescente dos vértices na triangulação $K1$. Nesse caso, dado um j, as direções que podem ser incluídas devem estar presentes em v_{j+1}, mas não em v_j. Em outras palavras, devemos adicionar um vértice v, de tal forma que

$$v = v_j + \sum_{i \in I'} 2^i$$

onde agora I' é um elemento do conjunto de partes de $I_{j+1} \setminus I_j$.

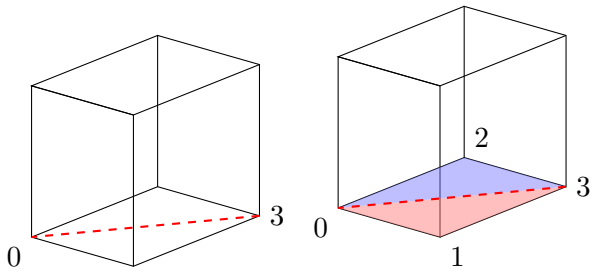

Figura 15: Novamente, sabemos que o vértice 3 é composto pelos elementos e_1 e e_2 e, portanto, só poderíamos considerar um elemento entre 0 e 3. A coface é completada com o elemento 1 ou com o elemento 2.

A quantidade de cofaces encontradas desta maneira, adicionando um elemento no meio de dois vértices do simplex, será igual a $\sum_{j=0}^{k-1}(2^{m_j} - 1)$, onde m_j é igual à quantidade de elementos do conjunto $I_{j+1} \setminus I_j$.

Sabendo que todos os m_j são inteiros positivos, maiores ou iguais a 1 e menores que $n-k$, podemos encontrar uma cota superior para esse último valor. Neste caso, a cota superior será considerando $m_j \leq n-k$ para todos os j, portanto

$$\sum_{j=0}^{k-1}(2^{m_j}-1) < k\ 2^{n-k}$$

Por sua vez, a forma de calcular todas as faces de um simplexo de dimensão $k+1$ apenas requer remover um vértice da tupla. Dado $(n_H, 0, v_1, \ldots v_{k+1})$, vamos gerar todas as suas faces, eliminando um de seus elementos. No entanto, assim como com as cofaces, é necessário considerar que se o primeiro elemento for removido, a face resultante deve ser convertida para a forma canônica, com seu primeiro vértice igual a zero. A quantidade de faces obtidas será $k+1$.

No que diz respeito à obtenção dos vértices de aproximação e à maneira de saber se um simplex intersecta a variedade, utilizamos os procedimentos mencionados na seção 2.2.

Algoritmo 3.4.2 Esqueleto combinatório

Input: Dictionary kFaces of the labels of the faces inside a hypercube n_H in which were found approximation vertices, indexed by integers

Output: Array of lists of sets containing the information of the combinatorial skeleton of the hypercube n_H

Initialize the empty array of lists of sets CS
Initialize the set of faces $Faces$ with $kFaces$
Initialize the empty dictionary $Cofaces$ associating sets with integers
for $1 \leq \text{step} \leq n-k$ **do**
    Index $\leftarrow 0$
    for S_1, S_2 in Faces **do**
        if S_1, S_2 are adjacent **then**
            $S \leftarrow \text{convexHull}(S_1, S_2)$
            if S is not in Cofaces **then**
                $\text{Cofaces}(S) \leftarrow$ index
                index $\leftarrow$ index $+1$
                $CS[step][index] \leftarrow \{\}$
            $CS[step][Cofaces(S)] \leftarrow CS[step][Cofaces(S)] \cup \{Faces(S_1), Faces(S_2)\}$
    Faces $\leftarrow$ copy of Cofaces

4 Resultados

4.1 Análise de Complexidade

Inicialmente, abordaremos a complexidade do algoritmo principal, antes de realizar as uniões segundo as adjacências, relacionadas ao Esqueleto Combinatório.

A análise da complexidade do algoritmo 3.4.1 pode ser dividida em partes. Estaremos particularmente interessados em encontrar uma cota para a complexidade das operações $faces$ e $cofaces$.

O processo de calcular $cofaces$ de um certo simplexo $(n_H, 0, v_1, \ldots v_k)$, em uma primeira instância, constrói o conjunto I de direções canônicas não presentes no último vértice v_k. Conforme visto, isso pode ser feito em $n-m$ operações aritméticas. Como a ordem em que os valores são adicionados ao conjunto I é decrescente, sabemos que adicionar cada um dos elementos a I não adiciona complexidade.

Em seguida, para obter o conjunto de possíveis vértices v no hipercubo n_H, precisamos iterar entre todos os α entre 1 e 2^{n-m}. Para calcular cada um dos vértices, será necessário iterar sobre os, no máximo, n bits de α utilizando n operações. Para calcular todos os vértices, serão necessárias $n \cdot 2^{n-m}$ operações. Por fim, são geradas duas $cofaces$ para cada um desses vértices, que precisam ser normalizadas na forma canônica. Esse processo, conforme visto, requer n operações.

De maneira semelhante, calcular as $cofaces$ adicionando um vértice no interior da tupla exigirá calcular as direções canônicas presentes em $I_{j+1} \setminus I_j$ e, em seguida, encontrar todos os possíveis vértices v. Seguindo o raciocínio anterior, para um dado j, saber quais elementos compõem $I_{j+1} \setminus I_j$ envolveria $n - m_j$ operações. Por sua vez, encontrar os vértices v para tal j exigiria formar um conjunto de partes de $I_{j+1} \setminus I_j$, o que requer $n \cdot 2^{n-m_j}$ operações.

Somando a quantidade de operações que cada um dos casos realiza, temos que nos dois primeiros, em que se adiciona um vértice antes ou depois do simplex, a complexidade total de calcular todas as $cofaces$ é $(n-m) + (2n) \cdot 2^{n-m}$. No caso das $cofaces$ obtidas adicionando um vértice no interior da tupla, exigirá $\sum_{j=0}^{k}(n - m_j + n\ 2^{n-m_j})$.

Considerando a quantidade assintótica de operações de todos os casos e acotando conforme mencionado em 3.4.1, concluímos que encontrar todas as $cofaces$ de um simplexo $(n_H, 0, v_1, \dots v_k)$, normalizando nos casos que seja necessário, tem complexidade $O(n\ k\ 2^{n-k})$.

Seguindo com a tarefa de encontrar as $faces$ de dimensão $(k-1)$ que contêm o simplexo $(n_H, 0, v_1, \dots v_k)$, isso exigirá gerar $k+1$ tuplas com $k-1$ elementos. Portanto, requererá $(k+1)(k-1)$ operações. Além disso, para a $face$ gerada descartando o vértice 0, são adicionadas n operações destinadas à normalização na forma canônica. A quantidade assintótica de operações será $O(k^2 + n)$.

Agora, podemos estimar a quantidade total de operações utilizadas no algoritmo. Dado um simplexo $(n_H, 0, v_1, \dots, v_k)$, o algoritmo começa calculando todas as suas $cofaces$ e armazenando as tuplas calculadas de acordo com seus valores de n_H em um dicionário H. Como vimos, encontrar $cofaces$ tem uma complexidade de $O(n \cdot k \cdot 2^{n-k})$, enquanto adicionar um elemento a um conjunto dentro de um dicionário tem uma complexidade de $O(\log(|H|) + \log(|H[n_H]|))$, que pode ser limitada por $O(\log(|S|) + n\ \log(n))$. Neste caso, $|S|$ é a quantidade de simplexos percorridos.

Em seguida, para cada uma dessas $cofaces$ calculadas, vamos encontrar suas $faces$ e verificar se estas intersectam a variedade. Neste caso, teremos que a complexidade de calcular todas as $faces$ para cada $coface$ é $O(n + k^2)$. Chamaremos a quantidade de operações necessárias para determinar se a variedade intersecta um simplexo como ρ. Se a variedade de fato intersecta o simplexo, será necessário adicionar $(n_H, 0, v_1, \dots, v_k)$ a um diccionário V, com complexidade limitada por $O(\log(|S|))$.

Podemos limitar a complexidade total de calcular os vértices de aproximação a $O(|S| \log(|S|)\ n^2 \rho\ 2^{n-k})$.

4.2 Exemplos

A seguir, listamos uma série de exemplos gerados pelo nosso algoritmo, implementado na linguagem Python e executado no ambiente Jupyter Notebook em um computador com as seguintes especificações: processador Intel Core i3-1115G4 @ 3.00GHz × 4 com gráfico integrado, 4 GB de memória RAM, sistema operacional Ubuntu 22.04.

Bagel de Klein, dado pelo conjunto de zeros da aplicação real

$$F(x,y,z,\theta,\omega) = \begin{pmatrix} x - \left(3 + \cos\frac{\theta}{2}\sin\omega - \sin\frac{\theta}{2}\sin\omega\right)\cos\theta \\ y - \left(3 + \cos\frac{\theta}{2}\sin\omega - \sin\frac{\theta}{2}\sin\omega\right)\sin\theta \\ z - \sin\frac{\theta}{2}\sin\omega - \cos\frac{\theta}{2}\sin\omega \end{pmatrix}$$

O domínio escolhido foi $[-5.5, 5.5] \times [-5.5, 5.5] \times [-2.5, 2.5] \times [-0.5, 2\pi + 0.5] \times [-0.5, 2\pi + 0.5]$, com 20 divisões por intervalo. O tempo de processamento foi de aproximadamente 122 segundos.

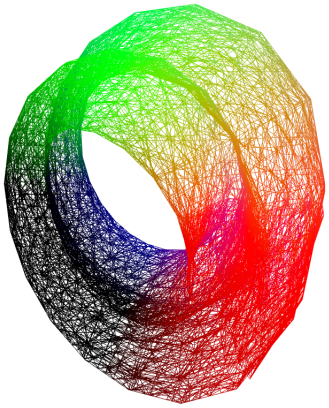

Figura 16: Visualização do Bagel de Klein.

Curva de nível complexa dada pelo polinômio $f\colon \mathbb{C}^2 \to \mathbb{C}$, $f(z,w) = z + w^3$. Colocando em coordenadas reais, obtemos

$$F(x,y,u,v) = \begin{pmatrix} x - 3uv^2 + u^3 \\ y - v^3 + 3u^2v \end{pmatrix}$$

O domínio escolhido foi $[-1,1]^4$, com 20 divisões por intervalo. O tempo de processamento foi de aproximadamente 31 segundos.

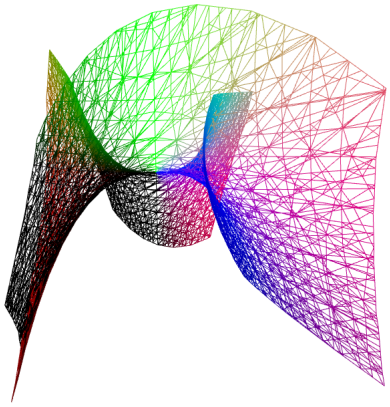

Figura 17: Visualização da curva complexa.

Um toro $S_1 \times S_1$ imerso em $\mathbb{R}^4$, dado pelo conjunto de zeros da aplicação

$$F(x,y,u,v) = \begin{pmatrix} x^2 + y^2 - \frac{1}{4} \\ u^2 - v^2 - 1 \end{pmatrix}$$

O domínio escolhido foi $[-1,1]^2 \times [-1.5,1.5]^2$, com 20 divisões por intervalo. O tempo de processamento foi de aproximadamente 29 segundos.

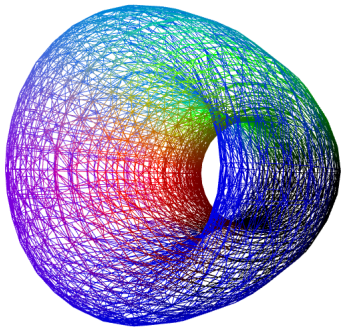

Figura 18: Visualização do toro.

Um toro triplo $S_1 \times S_1 \times S_1$ imerso em $\mathbb{R}^6$, dado pelo conjunto de zeros da aplicação

$$F(x, y, z, w, u, v) = \begin{pmatrix} x^2 + y^2 - \frac{1}{4} \\ z^2 - w^2 - 1 \\ u^2 - v^2 - 4 \end{pmatrix}$$

O domínio escolhido foi $[-1, 1]^2 \times [-1.5, 1.5]^2 \times [-4.5, 4.5]$, com 5 divisões por intervalo. O tempo de processamento foi de 509 segundos.

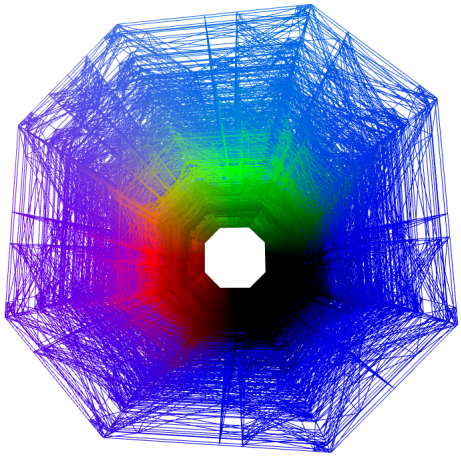

Figura 19: Visualização do toro triplo. Note que sua projeção é um toro sólido.

4.3 Conclusão

Conseguimos um algoritmo que aproxima politopos para uma variedade suave $F^{-1}(0)$ com dimensão $n - k$. Usando técnicas de discretização espacial e, posteriormente, de divisão de hipercubos em simplexos, pudemos, por meio um processo de linearização de uma certa função, obter pontos, tais que, por meio de um esqueleto combinatório, obter uma boa aproximação de uma função definida de modo implícito. Ainda, utilizando ferramentas desenvolvidas anteriormente, é possível visualizar esses politopos através de projeções em $\mathbb{R}^3$ por meio de um visualizador específico, sendo que ele possui 3 eixos possíveis para a projeção.

Algumas decisões importantes foram tomadas durante o desenvolvimento, como o uso de dicionários para armazenarmos os vértices de aproximação (dado que eles possuem um tamanho menor em memória, por exemplo, se compararmo-los a listas).

Outros aspectos relevantes que foram considerados são pensarmos na representação de um simplexo dentre as muitas possíveis. Para evitar ambiguidade, foi escolher, por exemplo, o menor

dos vértices de um dado simplexo como sendo 0 além de definirmos o número do hipercubo como sendo aquele que inclui todos os seus vértices do simplexo considerado.

Devido à estrutura de dados escolhida para representar o problema, conseguimos uma complexidade total de $O(|S|\log(|S|) \cdot n^2\rho \cdot 2^{n-k})$ para calcular os vértices de aproximação e armazenar a entrada necessária para calcular o esqueleto combinatório. Considerando $n-k$ como uma variável limitada, observamos que a dependência na dimensão do problema cresce de forma polinomial.

Como tópicos de discussão futura, pode-se mencionar métodos que tentam dividir cada dimensão da malha de forma ótima, ou mesmo, usar métodos adaptativos como *octatree* e *quadtree* para melhorar a aproximação da variedade.

Referências

[1] G. S. Bicalho. *TrueNgine - A N-Dimensional Renderer*. `https://github.com/GSBicalho/TrueNgine.git`. 2018.

[2] Boissonnat, Kachanovich e Wintraecken. "Tracing Isomanifolds in $\mathbb{R}^d$ in Time Polynomial in d Using Coxeter–Freudenthal–Kuhn Triangulations*". Em: *SIAM J. Comput.* 52.2 (2023). © 2023 Society for Industrial and Applied Mathematics, pp. 452–486. DOI: `10.1137/21M1412918`.

[3] Antonio Castelo Filho e Lucas Moutinho Bueno. *Aproximações de Variedades Definidas Implicitamente Utilizando Técnicas de Contagem e Enumeração*. Vol. 88. Notas em Matemática Aplicada. CDD - 51, ISBN 978-85-8215-091-7 e-ISBN 978-85-8215-090-0. São Carlos, SP: SBMAC, 2019, p. 108.

[4] J. Milnor. *Topology from the differentiable viewpoint*. Charlottesville University Press of Virginia, 1965.