Aproximação de Variedades Implícitas com Aplicação em Demonstrações Assistidas por Computador

10 de Julho de 2016

Prefácio

Use the template *preface.tex* together with the Springer document class SVMono (monograph-type books) or SVMult (edited books) to style your preface in the Springer layout.

A preface is a book's preliminary statement, usually written by the *author or editor* of a work, which states its origin, scope, purpose, plan, and intended audience, and which sometimes includes afterthoughts and acknowledgments of assistance.

When written by a person other than the author, it is called a foreword. The preface or foreword is distinct from the introduction, which deals with the subject of the work.

Customarily *acknowledgments* are included as last part of the preface.

Place(s), *Firstname Surname*
month year *Firstname Surname*

Conteúdo

Capítulo 1

Introdução

Adotou-se em todo esse trabalho as convenções do software Matlab/Octave para fazer referência a vetores e matrizes, ou às suas partes e serão apresentados algoritmos e programas com a notação do Matlab/Octave.

1.1 Contagem e Enumeração

Muitos problemas da área de ciências exatas, tais como a matemática, a física, a engenharia e a computação necessitam manipular objetos em dimensão elevada e também lidar com sua simulação em programas de computador. Para isto é necessário algoritmos capazes de trabalhar com um número de variáveis ou até mesmo número de laços de código que serão dados em tempo de execução. Para esta tarefa necessitamos fazer uma formulação matemática e algorítmos que atendam a esta necessidade. Muitas destas tarefas são traduzidas em problemas de contagem e enumeração, a contagem até mesmo para a alocação de memória e a enumeração para apresentar os vários dados do problema.

Esta área de pesquisa denomina-se Algoritmos Combinatórios e exemplos de livros clássicos são [6], [13] e [14]. Entre os tipos de problemas de contagem e enumeração mais importantes, podemos citar o produto cartesiano discreto, permutações e combinações. Com o primeiro podemos simular um ninho de laços em um programa de computador onde a quantidade de laços será dada em tempo de execução do código, representação de números inteiros em uma base, enumeração de malhas cartesianas em dimensão dada em tempo de execução, enumeração dos vértices de um hipercubo onde a dimensão da dada em tempo de execução, entre outros problemas.

Combinações são necessárias em problemas onde temos um conjunto de n dados e queremos obter todos os subconjuntos de $k \leq n$ dados deste. Também são

inúmeras as aplicações de enumeração das combinações como por exemplo listar todas as faces de dimensão k de um simplexo de dimensão n. Por último mas não menos importante a enumeração das permutações de um conjunto de n símbolos ordenados são necessários em problemas onde a ordem é requerida, e com a enumeração das combinações e permutações temos os arranjos. O Capítulo 3 apresenta alguns dos algoritmos combinatórios que serão utilizados nos Capítulos posteriores.

1.2 Aproximação de Variedades Implícitas

Variedades definidas implicitamente (curvas no caso de dimensão um e superfícies no caso de dimensão dois) aparecem em várias aplicações tais como, modelagem geométrica, modelagem molecular, computação gráfica, bifurcações em equações diferenciais, equações diferenciais com restrição, entre outras. Por este motivo desenvolver métodos, que sejam eficientes e produzam resultados com boa qualidade, para encontrar tais variedades é um problema importante. Dada uma função diferenciável $F : \mathbb{R}^m \to \mathbb{R}^n$, $m \geq n$ pretende-se encontrar o conjunto dos pontos onde F se anula:

$$\mathcal{M} = \{x \in \mathbb{R}^m \mid F(x) = 0\}$$

Nos pontos de $\mathcal{M}$ onde a derivada de F tem posto máximo, o Teorema da Função Implícita garante que $\mathcal{M}$ é uma variedade de dimensão $m - n$ (pelo menos localmente).

Para funções onde $m = 3$ e $n = 1$, um dos métodos mais conhecidos para se encontrar a superfície de nível $F(x_1, x_2, x_3) = 0$ é o algoritmo chamado Marching Cubes [11] [10]. Neste algoritmo, a região de interesse é dividida em uma malha retangular uniforme e cada célula da malha é testada para saber se esta contém uma parte da superfície. Este teste é feito por meio dos sinais de F nos vértices do cubo. Quando todas as células são testada obtém-se uma aproximação para a superfície. Este algoritmo é de difícil extensão para dimensões elevadas ($m > 4$). Neste caso os métodos mais utilizados são os baseados em decomposição simplicial (triangulação) do domínio e aproximação em cada simplexo por uma função mais simples (função afim, por exemplo) e encontrar os pontos onde estas aproximações se anulam. Existem vários trabalhos na literatura sobre este tipo de aproximação, porém poucos em dimensão elevada e entre eles podemos citar [15], [2], [3] e [4]. Quando se trata de aproximações adaptativas, isto é, com refinamento em regiões de interesse, o número de trabalhos se reduz ainda mais e entre eles podemos citar [4], [12] e [18].

Com o uso de triangulações de $\mathbb{R}^m$ para aproximar variedades impícitas de dimensões superiores a 1, o volume de informações sobre a triangulação e a aproximação linear por partes torna-se muito grande, portanto uma representação mais compacta tanto das triangulações como das aproximações é essencial. O objetivo deste do Capítulo 2 é apresentar uma breve introdução ao cálculo de uma uma aproximação poligonal para $\mathcal{M}$ utilizando técnicas de enumeração de produtos cartesianos discretos, permutações e combinações.

1.3 Aproximações de Funções por Mínimos Quadrados

Uma das formas possíveis de representar funções em um domínio discreto é armazenar os seus valores na maior quantidade de pontos possíveis (eventualmente em todos), assim quando necessário basta avaliar o seu valor em um desses pontos. Quando a função é essencialmente discreta, $f : D \to U$, onde por exemplo $D = \{x_1, \ldots, x_m\}$ e $U = \{0, 1\}$, ou seja, possui um conjunto finito de pontos no domínio e na imagem, essa representação é válida e define totalmente a função. Porém, é comum querermos representar funções onde D e U não precisam ser nem enumeráveis $f : \mathbb{R} \to \mathbb{R}$.

Com isso, como o armazenamento fica restrito a um conjunto finito de pontos, ao considerar a determinação do valor da função para um ponto que não esteja presente no conjunto de amostras disponíveis, representado pelas tuplas $(x_i, f(x_i)), i = 1, \ldots, m$, ou seja, o valor de f é conhecido apenas em x_i, faz-se necessário um processo que possibilite essa avaliação. Esse problema pode ser resolvido através de um processo de aproximação da função f por combinações de funções conhecidas.

Dois aspectos básicos acerca de f devem ser observados: A função f pode corresponder a uma amostragem de uma função conhecida, e.g. $f(x) = x^2$; As amostras $f(x_i)$ podem ser obtidas empiricamente, através de experimentos, medições, etc.

No primeiro caso, o erro da aproximação vai ficar determinado pela base de funções que serão utilizadas para aproximar f.

Já no segundo caso, como o comportamento da função não é conhecido, a escolha da base de funções para aproximação é dificultada e pode levar a resultados inadequados, como aproximar uma parábola por uma reta, por exemplo.

É fácil encontrar na literatura formas analíticas de resolver o problema da aproximação de funções por mínimos quadrados [16]. Apesar disso, no Capítulo **??** é feita uma breve revisão sobre o tema para ilustrar como produtos cartesianos discretos podem ajudar na solução desse tipo de problemas, sendo assim uma possível aplicação do conteúdo que será visto no Capítulo 3.

Capítulo 2

Aproximações de Funções Implícitas

Para maior aprofundamento no tema apresentado neste capítulo é sugerido a leitura da tese [3], do livro [1] e do artigo [4].

2.1 Topologia Linear por Partes

Dados os pontos $v_0, v_1, \ldots, v_n \in \mathbb{R}^m$, chama-se o conjunto $aff(v_0, \ldots, v_n) = \{v \in \mathbb{R}^m \mid \sum_{i=0}^{n} \lambda_i v_i = v \text{ e } \sum_{i=0}^{n} \lambda_i = 1\}$ de conjunto afim gerado pelos pontos $v_0, \ldots, v_n$.

Chama-se dimensão de $aff(v_0, \ldots, v_n)$ e denota-se por $dim(aff(v_0, \ldots, v_n))$ o maior número de vetores linearmente independentes entre os do conjunto $\{v_1 - v_0, \ldots, v_n - v_0\}$

Chama-se de célula convexa afim gerada pelos pontos $v_0, v_1, \ldots, v_n$ ao conjunto $\sigma = [v_0, \ldots, v_n] = \{v \in \mathbb{R}^m \mid \sum_{i=0}^{n} \lambda_i v_i = v, \sum_{i=0}^{n} \lambda_i = 1 \text{ e } \lambda_i \geq 0\}$. Definimos a dimensão de σ por $dim(\sigma) = dim(aff(v_0, \ldots, v_m))$.

Uma coleção C de células convexas afins é dita decomposição celular de um conjunto $K \subset \mathbb{R}^m$ se:

- $K = \cup_{\sigma \in C} \sigma$;
- se $\sigma_1, \sigma_2 \in C$, então $\sigma_1 \cap \sigma_2 = \emptyset$ ou $\sigma_1 \cap \sigma_2 \in C$;
- todo subconjunto compacto de K intercepta um número finito de células de C.

Uma célula convexa afim $\sigma = [v_0, \ldots, v_n]$ de $\mathbb{R}^m$ é chamada simplexo, se $dim(\sigma) = n$, isto é, um simplexo n-dimensional é uma célula convexa afim de dimensão n gerada por $n+1$ pontos.

Dado um simplexo $\sigma = [v_0, \ldots, v_n]$ de $\mathbb{R}^m$ e $\{w_0, \ldots, w_k\} \subset \{v_0, \ldots, v_n\}$, chama-se a célula convexa afim $\tau = [w_0, \ldots, w_k]$ de face de dimensão k de σ. Às faces de σ geradas por um único ponto denomina-se vértices de σ; por dois pontos, de arestas de σ; e por n pontos, de facetas de σ.

Uma decomposição celular C de $K \subset \mathbb{R}^m$ é chamada triangulação de K, se todas as células de C são simplexos.

Dado um simplexo $\sigma = [v_0, \ldots, v_n]$ de $\mathbb{R}^m$, define-se :

- a fronteira de σ, $\partial\sigma$ como a união de todas as faces de dimensão $n-1$ de σ;
- o baricentro de σ, $b(\sigma) = \frac{1}{n+1}\sum_{i=0}^{n} v_i$;
- o diâmetro de σ, $\rho(\sigma) = max\{\| v_i - v_j \|; i, j = 0, \ldots, n\}$;
- o raio de σ, $r(\sigma) = min\{\| v - b(\sigma) \| \mid v \in \partial(\sigma)\}$;
- a robustez de σ, $\theta(\sigma) = r(\sigma)/\rho(\sigma)$.

Se T é uma triangulação de $K \subset \mathbb{R}^m$, define-se:

- o diâmetro de T, $\rho(T) = sup\{\rho(\sigma) \mid \sigma \in T \text{ com } dim(\sigma) > 0\}$;
- a robustez de T, $\theta(T) = inf\{\theta(\sigma) \mid \sigma \in T \text{ com } dim(\sigma) > 0\}$.

2.2 Interpolação Linear Simplicial

Seja $F : D \subset \mathbb{R}^m \to \mathbb{R}^n$ uma aplicação e T uma triangulação de D.

Se $\sigma = [v_0, \ldots, v_m] \in T$, tem-se para cada $v \in \sigma$ uma única $(m+1)$-upla $\lambda = (\lambda_0, \ldots, \lambda_m)$ tal que $\sum_{i=0}^{m} \lambda_i v_i = v$, $\sum_{i=0}^{m} \lambda_i = 1$ e $\lambda_i \geq 0$, $i = 0, \ldots, m$.

Define-se $F_\sigma : \sigma \to \mathbb{R}^n$ onde

$$F_\sigma(v) = F_\sigma(\sum_{i=0}^{m} \lambda_i v_i) = \sum_{i=0}^{m} \lambda_i F(v_i) \tag{2.1}$$

Observe que F_σ é uma aplicação afim e que $F_\sigma(v_i) = F(v_i)$, $i = 0, \ldots, m$, isto é, F_σ é uma interpolação linear de F nos vértices de σ.

Define-se uma aproximação linear por partes para F sendo : $F_T : D \subset \mathbb{R}^m \to \mathbb{R}^n$ onde $F_T(v) = F_\sigma(v)$ para $v \in \sigma \in T$.

Portanto, F_T é uma interpolação linear de F para os vértices de T (nos simplexos de dimensão zero de T).

Suponha que $F : D \subset \mathbb{R}^m \to \mathbb{R}^n$ é uma aplicação de classe C^2 com $\| D^2F(x) \| \leq \alpha$ para todo $x \in D$ e $\sigma = [v_0, \ldots, v_n]$ um simplexo em D, então valem os seguintes resultados que podem ser vistos em [1] : Se T é uma triangulação robusta de D, isto é, $\theta(T) > 0$, então $\| F(v) - F_T(v) \| \leq \alpha\rho^2(T)/2$ para todo $v \in D$ e $\| DF(v) - DF_T(v) \| \leq \alpha\rho(T)/\theta(T)$ para todo v no interior de simplexos de dimensão m de T.

Embora $DF_\sigma(v)$ esteja bem definida para todo simplexo σ de dimensão m, se τ é uma face comum de dois simplexos σ_1 e σ_2 de dimensão m e $v \in \tau$, $DF_{\sigma_1}(v)$ e $DF_{\sigma_2}(v)$ podem ser distintos e, portanto, $DF_T(v)$ não está bem definida.

2.3 Esquema de Diferenças Simplicial

Seja $F : D \subset \mathbb{R}^m \to \mathbb{R}^n$ uma aplicação de classe C^2 e T uma triangulação de D.

Seja $\sigma = [v_0, \ldots, v_m] \in T$. Como σ é um simplexo de dimensão m, os vetores $v_1 - v_0, v_2 - v_0, \ldots, v_m - v_0$ são linearmente independentes; logo, fazendo $\delta_i = \| v_i - v_0 \|$ e $\omega_i = (v_i - v_0)/\delta_i$, $i = 1, \ldots, m$, temos que a matriz cujas colunas são os vetores $w_1, \ldots, w_m$ é invertível.

Para $v \in \sigma$ seja λ, tal que $\sum_{i=0}^m \lambda_i v_i = v$, $\sum_{i=0}^m \lambda_i = 1$ e $\lambda_i \geq 0$, $i = 0, \ldots, m$. Então, pode-se escrever

$$\begin{gathered} v - v_0 = \\ = \sum_{i=0}^m \lambda_i v_i - v_0 \sum_{i=0}^m \lambda_i = \\ = \sum_{i=1}^m \lambda_i (v_i - v_0) \\ = \sum_{i=1}^m \lambda_i \delta_i \omega_i = \\ = (\omega_1 \cdots \omega_m) \begin{pmatrix} \lambda_1\delta_1 \\ \vdots \\ \lambda_m\delta_m \end{pmatrix} \end{gathered}$$

ou

$$\begin{pmatrix} \lambda_1\delta_1 \\ \vdots \\ \lambda_m\delta_m \end{pmatrix} = (\omega_1 \cdots \omega_m)^{-1} (v - v_0)$$

Agora

$$\begin{gathered} F_T(v) - F(v_0) = \\ = F_\sigma(v) - F(v_0) = \\ = \sum_{i=0}^{m} \lambda_i F(v_i) - F(v_0) \sum_{i=0}^{m} \lambda_i = \\ = \sum_{i=1}^{m} \lambda_i (F(v_i) - F(v_0)) = \\ = \sum_{i=1}^{m} \lambda_i \delta_i (F(v_i) - F(v_0))/\delta_i \end{gathered}$$

ou

$$\begin{gathered} F_T(v) - F(v_0) = \\ = \left(\frac{F(v_1)-F(v_0)}{\delta_1} \cdots \frac{F(v_m)-F(v_0)}{\delta_m} \right) \begin{pmatrix} \lambda_1 \delta_1 \\ \vdots \\ \lambda_m \delta_m \end{pmatrix} = \\ = \left(\frac{F(v_1)-F(v_0)}{\delta_1} \cdots \frac{F(v_m)-F(v_0)}{\delta_m} \right) (\omega_1 \cdots \omega_m)^{-1} (v - v_0) \end{gathered}$$

Daí, como $\omega_i = (v_i - v_0)/\delta_i$, tem-se $v_i = v_0 + \delta_i \omega_i$, $i = 1, \ldots, m$, e portanto

$$\begin{gathered} F_T(v) - F(v_0) = \\ = \left(\frac{F(v_0+\delta_1\omega_1)-F(v_0)}{\delta_1} \cdots \frac{F(v_0+\delta_m\omega_m)-F(v_0)}{\delta_m} \right) (\omega_1 \cdots \omega_m)^{-1} (v - v_0) \end{gathered}$$

Observe que as colunas da matriz

$$\left(\frac{F(v_0 + \delta_1\omega_1) - F(v_0)}{\delta_1} \cdots \frac{F(v_0 + \delta_m\omega_m) - F(v_0)}{\delta_m} \right)$$

são aproximações para as derivadas direcionais de F nas direções $\omega_1, \ldots, \omega_m$, em torno do ponto v_0; portanto este é um esquema de diferenças que chamaremos de esquema de diferenças simplicial.

Daí, se $\delta_i \to 0$ para $i = 1, \ldots, m$ com uma certa proporcionalidade, isto é, se o diâmetro de σ tende a zero mantendo a robustez limitada para que os vetores $w_1, \ldots, w_m$ continuem sendo linearmente independentes, então F_σ é uma aproximação para os dois primeiros termos da série de Taylor de F, em torno do ponto v_0. Outra observação é que a mesma análise pode ser feita para qualquer outro vértice de σ; portanto, o esquema descrito anteriormente não depende apenas do ponto v_0 e sim do simplexo σ.

Capítulo 3

Contagem e Enumeração

Alguns dos textos clássicos sobre algoritmos combinatórios para maior aprofundamento são [14], [6] e [13]. Os algoritmos em Matlab/Octave que serão desenvolvidos neste capítulo serão utilizados nos capítulos posteriores.

3.1 Produtos Cartesianos Discretos

3.1.1 Definição

Dado um conjunto de n conjuntos discretos, $I_1^{k_1}, I_2^{k_2}, \ldots, I_n^{k_n}$, onde k_i é o número de elementos do conjunto i. Define-se o produto cartesiano $I = I_1^{k_1} \times I_2^{k_2} \times \cdots \times I_n^{k_n}$ como o conjunto

$$I = \{(i_1, i_2, \ldots, i_n) \quad | \quad i_1 \in I_1^{k_1}, i_2 \in I_1^{k_1}, \ldots, i_n \in I_n^{k_n}\}$$

Como exemplo, considere os conjuntos $I_1^2 = \{0, 1\}$ e $I_2^3 = \{a, b, c\}$, então pela definição acima tem-se

$$I = I_1^2 \times I_2^3 = \{(0, a), (0, b), (0, c), (1, a), (1, b), (1, c)\}$$

3.1.2 Exemplos

A contagem (número de elementos neste caso) do conjunto $I = I_1^{k_1} \times I_2^{k_2} \times \cdots \times I_n^{k_n}$ é bastante simples e é dados por $k_1 k_2 \ldots k_n$. A enumeração (quais são os elementos apresentados em alguma ordem) já pode não ser uma tarefa tão simples.

A ordem da enumeração e o significado de cada ordem é de fundamental importância em vários problemas. Como exemplo, considere $I = I_1^{k_1} \times I_2^{k_2} \times \cdots \times I_n^{k_n}$, onde $I_i^{k_i} = \{0, 1\}$. Como os elementos de cada conjunto são 0 ou 1, este conjunto pode ser olhado como a representação na base 2 dos números 0 a $2^{n+1} - 1$, isto

depende da forma de enumeração deste conjunto. Para isto, podemos enumerar com a seguinte ordem:

$$\begin{aligned}
0 &= 0\cdot 2^0 + 0\cdot 2^1 + \cdots + 0\cdot 2^n && \text{correspondente a } (0,0,\ldots,0)\\
1 &= 1\cdot 2^0 + 0\cdot 2^1 + \cdots + 0\cdot 2^n && \text{correspondente a } (1,0,\ldots,0)\\
2 &= 0\cdot 2^0 + 1\cdot 2^1 + \cdots + 0\cdot 2^n && \text{correspondente a } (0,1,\ldots,0)\\
&\vdots\\
2^{n+1}-1 &= 1\cdot 2^0 + 1\cdot 2^1 + \cdots + 1\cdot 2^n && \text{correspondente a } (1,1,\ldots,1)
\end{aligned}$$

Raciocínio análogo pode ser feito para a representação de números em qualquer base como produto cartesiano com uma enumeração apropriada. Observem que é possível obter a ordem de uma enumeração de um produto cartesiano a partir de cada elemento, tal como a regra do exemplo acima, e também é possível obter o elemento a partir da ordem deste elemento na enumeração utilizando um algoritmo bem simples de divisão por 2 (que neste caso é a base).

Se y é um número entre 0 e $2^{n+1}-1$, para obter o dígito que multiplica 2^0 que vamos denominar $x(1)$, basta encontrar o resto da divisão de y por 2. Para encontrar o dígito que multiplica 2^1 que vamos denominar $x(2)$, basta encontrar o resto da divisão de $(y-x(1))/2$ por 2, e assim sucessivamente.

Outra aplicação muito importante envolvendo a enumeração de produtos cartesianos é a possibilidade de enumerar as células de uma malha cartesiana e também obter uma célula a partir da ordem da enumeração desta. Por exemplo, uma malha cartesiana no plano pode ser rotulada da seguinte forma:

5	$(0,5)$	$(1,5)$	$(2,5)$	$(3,5)$	$(4,5)$	$(5,5)$	$(6,5)$
4	$(0,4)$	$(1,4)$	$(2,4)$	$(3,4)$	$(4,4)$	$(5,4)$	$(6,4)$
3	$(0,3)$	$(1,3)$	$(2,3)$	$(3,3)$	$(4,3)$	$(5,3)$	$(6,3)$
2	$(0,2)$	$(1,2)$	$(2,2)$	$(3,2)$	$(4,2)$	$(5,2)$	$(6,2)$
1	$(0,1)$	$(1,1)$	$(2,1)$	$(3,1)$	$(4,1)$	$(5,1)$	$(6,1)$
0	$(0,0)$	$(1,0)$	$(2,0)$	$(3,0)$	$(4,0)$	$(5,0)$	$(6,0)$
	0	1	2	3	4	5	6

Se uma enumeração for feita por linha, a ordem da enumeração fica:

5	$(0,5)$ 35	$(1,5)$ 36	$(2,5)$ 37	$(3,5)$ 38	$(4,5)$ 39	$(5,5)$ 40	$(6,5)$ 41
4	$(0,4)$ 28	$(1,4)$ 29	$(2,4)$ 30	$(3,4)$ 31	$(4,4)$ 32	$(5,4)$ 33	$(6,4)$ 34
3	$(0,3)$ 21	$(1,3)$ 22	$(2,3)$ 23	$(3,3)$ 24	$(4,3)$ 25	$(5,3)$ 26	$(6,3)$ 27
2	$(0,2)$ 14	$(1,2)$ 15	$(2,2)$ 16	$(3,2)$ 17	$(4,2)$ 18	$(5,2)$ 19	$(6,2)$ 20
1	$(0,1)$ 7	$(1,1)$ 8	$(2,1)$ 9	$(3,1)$ 10	$(4,1)$ 11	$(5,1)$ 12	$(6,1)$ 13
0	$(0,0)$ 0	$(1,0)$ 1	$(2,0)$ 2	$(3,0)$ 3	$(4,0)$ 4	$(5,0)$ 5	$(6,0)$ 6
	0	1	2	3	4	5	6

Assim se $(x(1), x(2))$ é um elemento da malha, a enumeração fica $y = x(1) + 7 \cdot x(2)$ e por exemplo $(4, 3)$ é o elemento número $4 + 7 \cdot 3 = 25$. Para obter $(x(1), x(2))$ a partir de y, $x(1)$ é o resto da divisão de y por 7 e $x(2) = (y - x(1))/7$. No exemplo anterior se $y = 25$, $x(1) = mod(y, 7) = mod(25, 7) = 4$ e $x(2) = (y - x(1))/7 = 3$.

3.1.3 Programas em Matlab/Octave

Se y é um inteiro entre 0 e $2^n - 1$, então a n-upla $(x_1, x_2, \ldots, x_n)$ pertencente a I que corresponde a representação de y na base 2 pode ser obtida com o seguinte algoritmo em Matlab/Octave:

```
copy = y;
for j = 1:n-1
     x(j)   = mod(copy,2);
     copy   = (copy-x(j))/2;
end
x(n) = copy;
```

Por exemplo, se $n = 4$ e $y = 13$, temos:

$$\begin{array}{lll} copy & = 13 & \\ x(1) & = mod(13, 2) & = 1 \\ copy & = (13 - 1)/2 & = 6 \\ x(2) & = mod(6, 2) & = 0 \\ copy & = (6 - 0)/2 & = 3 \\ x(3) & = mod(3, 2) & = 1 \\ copy & = (3 - 1)/2 & = 1 \\ x(4) & = 1 & \end{array}$$

e

$$y = x(1) \cdot 2^0 + x(2) \cdot 2^1 + x(3) \cdot 2^2 + x(4) \cdot 2^3 = 1 \cdot 1 + 0 \cdot 2 + 1 \cdot 4 + 1 \cdot 8 = 13$$

Observe que os vértices do hipercubo $I^n = [0, 1]^n$ podem ser obtidos por este mesmo algoritmo

```
for i = 0:2^n-1
      [Coords]  = HyperCubeCoords(n,i);
end

function [Coords] = HyperCubeCoords(n,i)
   copy = i;
   for j = 1:n-1
      Coords(j) = mod(copy,2);
      copy      = (copy-Coords(j))/2;
   end
   Coords(n) = copy;
   return
end
```

Se $I = I_1^{k_1} \times I_2^{k_2} \times \cdots \times I_n^{k_n}$ com $I_j^{k_j} = \{0, 1, \ldots, k_j\}$, definimos $Base(1) = 1$ e $Base(j+1) = (k_{j+1}+1) \cdot Base(j)$, $j = 1, \ldots, n$. Assim se $(x(1), x(2), \ldots, x(n))$ é um elemento de I, a enumeração equivalente ao exemplo anterior é dado por:

$$y = \sum_{j=1}^{n} x(j) \cdot Base(j)$$

O processo de recuperação do elemento de uma malha cartesiana com coordenadas inteiras a partir de seu rótulo y é obtida com o seguinte algoritmo em Matlab/Octave:

```
   copy = y;
   for j = n:-1:2
        aux    = mod(copy,Base(j));
        x(j)   = (copy-aux)/Base(j);
        copy   = aux;
   end
   x(1) = copy;
```

No exemplo anterior, $Base(1) = 1$, Base(2) = 7 e o elemento número 18 da malha é dado por:

$$\begin{array}{lll} copy & = 18 & \\ aux & = mod(18,7) & = 4 \\ x(2) & = (18-4)/7 & = 2 \\ copy & = 4 & \\ x(1) & = 4 & \end{array}$$

e

$$y = x(1) \cdot Base(1) + x(2) \cdot Base(2) = 4 \cdot 1 + 2 \cdot 7 = 18$$

Um trecho de código Matlab/Octave para gerar uma malha cartesiana com número de células em cada coordenada dada pelo vetor $Division = [k_1, k_2, \ldots, k_n]$ é dado a seguir:

```
    Base  = InitGrid(n, Division);
    for g = 0:Base(n+1)-1
       [Grid] = GridCoords(n,g,Base) ;
    end

function [Base] = InitGrid(n, Division)
   Base(1) = 1;
   for i = 2:n+1
      Base(i) = Base(i-1)*Division(i-1);
   end
   return
end

function [Grid] = GridCoords(n,i,Base)
   copy = i;
   for j = n:-1:2
      aux     = mod(copy,Base(j));
      Grid(j) = (copy-aux)/Base(j);
      copy    = aux;
```

```
   end
   Grid(1) = copy;
   return
end
```

Para gerar os vértices dos hipercubos de uma malha cartesiana definida pelos vetores $First$ correspondendo ao canto com as menores coordenadas da malha, $Last$ correspondendo ao canto com as maiores coodenadas da malha e $Division$ correspondendo ao número de divisões da malha em cada direção, pode-se agrupar os trechos de código anteriores como segue:

```
function GenVertGrid(n, First, Last, Division)
   Delta = (Last-First)./Division;
   Base  = InitGrid(n, Division);
   for g = 0:Base(n+1)-1
      [Grid] = GridCoords(n, g, Base)
      [Vert] = GenVert(n, Grid, First, Delta)
   end
   return
end

function [Base] = InitGrid(n, Division)
   Base(1) = 1;
   for i = 2:n+1
      Base(i) = Base(i-1)*Division(i-1);
   end
   return
end

function [Grid] = GridCoords(n,i,Base)
   copy = i;
   for j = n:-1:2
      aux     = mod(copy,Base(j));
      Grid(j) = (copy-aux)/Base(j);
      copy    = aux;
   end
   Grid(1) = copy;
   return
end

function [Vert] = GenVert(n, Grid, First, Delta)
```

```
   Vert  = [];
   for i = 0:2^n-1
      [Coords]  = HyperCubeCoords(n,i);
      [VHC]     = HyperCube(n,First,Delta,Grid,Coords,CoordPert);
      Vert      = [Vert; VHC];
   end
   return
end

function [Coords] = HyperCubeCoords(n,i)
   copy = i;
   for j = 1:n-1
      Coords(j) = mod(copy,2);
      copy      = (copy-Coords(j))/2;
   end
   Coords(n) = copy;
   return
end

function [VHC] = HyperCube(n, First, Delta, I, Coords)
   for i = 1:n
      VHC(i) = First(i) + (I(i)+Coords(i))*Delta(i);
   end
   return
end
```

3.2 Permutações

3.2.1 Definição

Seja $\pi : \{1, 2, \ldots, m\} \rightarrow \{1, 2, \ldots, m\}$ uma permutação de $\{1, 2, \ldots, m\}$ a qual pode-se denotar pela m-upla ordenada $\pi = (\pi_1, \pi_2, \ldots, \pi_m)$.

Pretende-se enumerar o conjunto das permutações de $\{1, 2, \ldots, m\}$ e para isto pode-se listar o conjunto das permutações de maneira indutiva, usando o fato que cada permutação de $\{1, 2, \ldots, m\}$ pode ser obtida das permutações de $\{1, 2, \ldots, m-1\}$, adicionando a coordenada m em todas as posições possíveis. Por exemplo se $(i_1, i_2, \ldots, i_{m-1})$ é uma permutação de $\{1, 2, \ldots, m-1\}$ pode-se gerar m permu-

tações de $\{1, 2, \ldots, m\}$ incluindo m como segue:

$$\begin{array}{c}(m, i_1, i_2, \ldots, i_{m-1})\\(i_1, m, i_2, \ldots, i_{m-1})\\(i_1, i_2, m, \ldots, i_{m-1})\\\vdots\\(i_1, i_2, \ldots, m, i_{m-1})\\(i_1, i_2, \ldots, i_{m-1}, m)\end{array}$$

3.2.2 Exemplos

A seguir mostra-se como são geradas as permutações de $\{1\}$, $\{1, 2\}$, $\{1, 2, 3\}$ e $\{1, 2, 3, 4\}$.

Figura 3.2.3

3.2.3 Programas em Matlab/Octave

Pode-se gerar as permutações de $\{1, 2, \ldots, m\}$ a partir do produto cartesiano. Primeiro observa-se que o conjunto $\{1\} \times \{1, 2\} \times \{1, 2, 3\} \times \cdots \times \{1, 2, \ldots, m\}$ tem exatamente $m!$ elementos e pode-se relacionar cada elemento deste conjunto $I_k = (i_1^k, i_2^k, \ldots, i_m^k)$ unicamente com uma permutação de $\{1, 2, \ldots, m\}$. Primeiro sabe-se que $i_1^k = 1$, $i_2^k = 1$ ou 2, $i_3^k = 1$ ou 2 ou 3, assim por diante. A permutação é construída da seguinte forma:

- O dígito 1 é colocado na posição i_1^k, ou seja (1).
- O dígito 2 é colocado na posição i_2^k, isto é, se $i_2^k = 1$ tem-se $(2, 1)$ e caso $i_2^k = 2$ tem-se $(1, 2)$.
- O dígito 3 é colocado na posição i_3^k, ou seja se a permutação de $\{1, 2\}$ escolhida obtida foi $(2, 1)$ por exemplo e $i_3^k = 1$ tem-se $(3, 2, 1)$ e caso $i_3^k = 2$ tem-se $(2, 3, 1)$ e se $i_3^k = 3$ tem-se $(2, 1, 3)$.

Por exemplo, se $m = 4$ e $I_k = (1, 1, 3, 2)$ a permutação gerada é:

$$\begin{aligned} i_1^k = 1 &\rightarrow (1) \\ i_2^k = 1 &\rightarrow (2, 1) \\ i_3^k = 3 &\rightarrow (2, 1, 3) \\ i_3^k = 2 &\rightarrow (2, 4, 1, 3) \end{aligned}$$

Um algoritmo para obter uma permutação a partir de um elemento I_k é dado abaixo

```
function [F] = Map_Perm(m,f)
    for i = 1:m
        F(i) = 0;
    end
    for i = 1:m
    if F(f(i)) == 0
        F(f(i)) = i;
    else
        for j = m:-1:f(i)+1
            F(j) = F(j-1);
        end
        F(f(i)) = i;
    end
    end
    return
end
```

Daí, para gerar todas as permutações de $\{1, 2, \ldots, m\}$ temos:

```
function [P] = Enumerate_Perm(m)
    P = [];
    for i = 1:m
        Division(i) = i;
    end
    [Base] = InitGrid(m,Division);
    for k = 0:Base(m+1)-1
        [f] = GridCoords(m,k,Base);
        f = f+1;
        [F] = Map_Perm(m,f);
        P   = [P; F];
    end
    return
end
```

3.3 Combinações

3.3.1 Definição

Combinações podem ser definidas como todos os subconjuntos de k símbolos de um conjunto de m símbolos. Se $C = \{1, 2, 3, \ldots, m\}$, as combinações de k elementos deste conjunto são: $\{1, 2, \ldots, k-1, k\}$, $\{1, 2, \ldots, k-1, k+1\}$, ..., $\{1, 2, \ldots, k-1, m\}$, $\{1, 2, \ldots, k, k+1\}$, ...

3.3.2 Exemplos

Por exemplo se $C = \{1, 2, 3, 4, 5\}$, todas as combinações com 3 símbolos são: $\{1, 2, 3\}$, $\{1, 2, 4\}$, $\{1, 2, 5\}$, $\{1, 3, 4\}$, $\{1, 3, 5\}$, $\{1, 4, 5\}$, $\{2, 3, 4\}$, $\{2, 3, 5\}$, $\{2, 4, 5\}$ e $\{3, 4, 5\}$.

3.3.3 Programas em Matlab/Octave

As combinações também podem ser obtidas a partir do produto cartesiano. Se $C = \{1, 2, 3, \ldots, m\}$, para obter as combinações de k símbolos de C, considere $I = I_1 \times I_2 \times \cdots \times I_k$, $I_i = \{1, \ldots, m-k+i\}, i = 1, \ldots, k$ e observe que o número de elementos de I é $(m-k+1)(m-k+2)\cdots m = k! \left(\begin{array}{c} m \\ k \end{array} \right)$. Daí, basta descartar os elementos que não estão em ordem crescente, isto é, se $(i_1, i_2, \ldots, i_k)$

é um elemento do produto cartesiano, ele vai pertencer ao conjunto formado pelas combinações se $i_1 < i_2 < \cdots < i_k$.

Por exemplo, se $C = \{1, 2, 3, 4, 5\}$, para obter todas as combinações com 3 símbolos, enumera-se os elementos de $I = \{1, 2, 3\} \times \{1, 2, 3, 4\} \times \{1, 2, 3, 4, 5\}$ que são: $(1,1,1), (2,1,1), (3,1,1), (1,2,1), (2,2,1), (3,2,1), (1,3,1), (2,3,1), (3,3,1), (1,4,1), (2,4,1),$
$(3,4,1), (1,1,2), (2,1,2), (3,1,2), (1,2,2), (2,2,2), (3,2,2), (1,3,2), (2,3,2), (3,3,2), (1,4,2),$
$(2,4,2), (3,4,2), (1,1,3), (2,1,3), (3,1,3), (1,2,3), (2,2,3), (3,2,3), (1,3,3), (2,3,3), (3,3,3), (1,4,3), (2,4,3), (3,4,3), (1,1,4), (2,1,4), (3,1,4), (1,2,4), (2,2,4), (3,2,4), (1,3,4), (2,3,4),$
$(3,3,4), (1,4,4), (2,4,4), (3,4,4), (1,1,5), (2,1,5), (3,1,5), (1,2,5), (2,2,5), (3,2,5), (1,3,5), (2,3,5), (3,3,5), (1,4,5), (2,4,5), (3,4,5).$

Os elementos que estão em ordem crescente são: $(1,2,3), (1,2,4), (1,3,4), (2,3,4), (1,2,5), (1,3,5), (2,3,5), (1,4,5), (2,4,5), (3,4,5).$

Um algoritmo para gerar todas as combinações é apresentada a seguir:

```
function [P] = Enumerate_Comb(n,k)
    P = [];
    for i = 1:k
        Division(i) = n-k+i;
    end
    [Base] = InitGrid(k,Division);
    for j = 0:Base(k+1)-1
        [f] = GridCoords(k,j,Base);
        f = f+1;
        [lex] = Lexico(k,f);
        if lex == 1
            P   = [P; f];
        end
    end
    return
end


function [lex] = Lexico(k,f)
    for i = 1:k-1
        if f(i) >= f(i+1)
            lex = 0;
            return
```

```
        end
    end
    lex = 1;
    return
end
```

3.4 Representação de Simplexos e suas Faces

No capítulo 2 foi visto a definição de simplexos que será utilizada para gerar aproximações de variedades definidas implicitamente. Nesta sessão veremos como representar simplexos e suas faces. Observe que nosso objetivo aqui não é uma representação geométrica e sim algébrica utilizando símbolos.

3.4.1 Definição

Um simplexo de dimensão n em $\mathbb{R}^m$, $m \geq n$, gerado pelos vértices $v_1, v_2, \ldots, v_n, v_{n+1}$ e denotado por $\sigma = [v_1, v_2, \ldots, v_n, v_{n+1}]$ pode ser representado pelos índices dos vértices em ordem lexicográfica e denotado por $l(\sigma) = (1, 2, \ldots, n, n+1)$.

Como uma face de dimensão k, denotado por τ, de um simplexo de dimensão n, σ, também é um simplexo de dimensão k gerado pelo fecho convexo de $k+1$ vértices de σ, a representação de τ, $l(\tau) = (i_1, \ldots, i_k, i_{k+1})$ é uma combinação de $k+1$ símbolos de $l(\sigma) = (1, 2, \ldots, n, n+1)$.

3.4.2 Exemplos

Se $l(\sigma) = (1, 2, 3, 4, 5)$ representa o simplexo $\sigma = [v_1, v_2, v_3, v_4, v_5]$, suas faces de dimensão 2 são:

1. $l(\tau_1) = (1, 2, 3)$ representando a face $\tau_1 = [v_1, v_2, v_3]$;
2. $l(\tau_2) = (1, 2, 4)$ representando a face $\tau_2 = [v_1, v_2, v_4]$;
3. $l(\tau_3) = (1, 2, 5)$ representando a face $\tau_3 = [v_1, v_2, v_5]$;
4. $l(\tau_4) = (1, 3, 4)$ representando a face $\tau_4 = [v_1, v_3, v_4]$;
5. $l(\tau_5) = (1, 3, 5)$ representando a face $\tau_5 = [v_1, v_3, v_5]$;
6. $l(\tau_6) = (1, 4, 5)$ representando a face $\tau_6 = [v_1, v_4, v_5]$;
7. $l(\tau_7) = (2, 3, 4)$ representando a face $\tau_7 = [v_2, v_3, v_4]$;

8. $l(\tau_8) = (2, 3, 5)$ representando a face $\tau_8 = [v_2, v_3, v_5]$;
9. $l(\tau_9) = (2, 4, 5)$ representando a face $\tau_9 = [v_2, v_4, v_5]$;
10. $l(\tau_{10}) = (3, 4, 5)$ representando a face $\tau_{10} = [v_3, v_4, v_5]$;

3.4.3 Programas em Matlab/Octave

Um algoritmo para gerar todas as faces de dimensão k de um simplexo de dimensão n é um mero exercício da utilização de combinações, assim deixamos a cargo do leitor esta implementação.

3.5 Representação de Hipercubos e suas Faces

Nesta sessão veremos como representar hipercubos e suas faces.

3.5.1 Definição

Considere o hipercubo unitário de dimensão n, $I^n = [0, 1]^n$. Uma face de dimensão $n - 1$, ou faceta, de I^n é o conjunto dos pontos $x = (x_1, x_2, \ldots, x_n)$ de I^n tal que $x_i = 0$ ou $x_i = 1$ para algum $1 \leq i \leq n$. São num total de $2n$ facetas. Vamos representar a face I_i^{n-1} tal que $x_i = 0$ pelo inteiro i e a face I_{n+i}^{n-1} tal que $x_i = 1$ pelo inteiro $n + i$.

Assim, $I_1^{n-1} = \{x \in I^n \mid x_1 = 0\}$, $I_2^{n-1} = \{x \in I^n \mid x_2 = 0\}$, ..., $I_n^{n-1} = \{x \in I^n \mid x_n = 0\}$, e $I_{n+1}^{n-1} = \{x \in I^n \mid x_1 = 1\}$, $I_{n+2}^{n-1} = \{x \in I^n \mid x_2 = 1\}$, ..., $I_{2n}^{n-1} = \{x \in I^n \mid x_n = 1\}$.

3.5.2 Exemplos

Se $l(\sigma) = (1, 2, 3, 4, 5)$ representa o simplexo $\sigma = [v_1, v_2, v_3, v_4, v_5]$, suas faces de dimensão 2 são:

1. $l(\tau_1) = (1, 2, 3)$ representando a face $\tau_1 = [v_1, v_2, v_3]$;
2. $l(\tau_2) = (1, 2, 4)$ representando a face $\tau_2 = [v_1, v_2, v_4]$;
3. $l(\tau_3) = (1, 2, 5)$ representando a face $\tau_3 = [v_1, v_2, v_5]$;
4. $l(\tau_4) = (1, 3, 4)$ representando a face $\tau_4 = [v_1, v_3, v_4]$;
5. $l(\tau_5) = (1, 3, 5)$ representando a face $\tau_5 = [v_1, v_3, v_5]$;
6. $l(\tau_6) = (1, 4, 5)$ representando a face $\tau_6 = [v_1, v_4, v_5]$;

7. $l(\tau_7) = (2, 3, 4)$ representando a face $\tau_7 = [v_2, v_3, v_4]$;
8. $l(\tau_8) = (2, 3, 5)$ representando a face $\tau_8 = [v_2, v_3, v_5]$;
9. $l(\tau_9) = (2, 4, 5)$ representando a face $\tau_9 = [v_2, v_4, v_5]$;
10. $l(\tau_{10}) = (3, 4, 5)$ representando a face $\tau_{10} = [v_3, v_4, v_5]$;

3.5.3 Programas em Matlab/Octave

Um algoritmo para gerar todas as faces de dimensão k de um simplexo de dimensão n é um mero exercício da utilização de combinações, assim deixamos a cargo do leitor esta implementação.

Capítulo 4

Aproximações de Variedades Implícitas

Para maior aprofundamento no tema apresentado neste capítulo é sugerido a leitura da tese [3], do livro [1] e do artigo [4].

4.1 As Triangulações K_1 e J_1

Inicialmente será apresentada a triangulação $\mathcal{C}FK$ (devida a Coxeter [5], Freudenthal [7] e Kuhn [9]) do cubo unitário $I^m = [0,1]^m$, da qual derivam as triangulações K_1 e J_1 de $\mathbb{R}^m$.

Seja $\pi : \{1, 2, \ldots, m\} \to \{1, 2, \ldots, m\}$ uma permutação de $\{1, 2, \ldots, m\}$ a qual vamos denotar pela m-upla ordenada $\pi = (\pi_1, \pi_2, \ldots, \pi_m)$, e $\{e_1, e_2, \ldots, e_m\}$ base canônica de $\mathbb{R}^m$.

Considere o simplexo $\sigma_\pi = [v_0, v_1, \ldots, v_m]$, onde

$$\begin{aligned} v_0 &= (0, \ldots, 0) \in I^m; \\ v_i &= v_{i-1} + e_{\pi_i} = e_{\pi_1} + \cdots + e_{\pi_i}, \;\; i = 1, \ldots, m. \end{aligned}$$

É fácil verificar que σ_π é um simplexo de dimensão m contido em I^m.

A triangulação $\mathcal{C}FK$ é definida como o conjunto dos simplexos σ_π e suas faces, tal que π seja uma permutação de $\{1, 2, \ldots, m\}$.

Para exemplificar, a Figura 4.1 representam as triangulações $\mathcal{C}FK$ de I^2 e I^3, respectivamente.

Observe que existe uma relação biunívoca entre π e σ_π; portanto, a triangulação $\mathcal{C}FK$ de I^m contém $m!$ simplexos de dimensão m.

A triangulação K_1 de $\mathbb{R}^m$ é obtida pela translação da triangulação $\mathcal{C}FK$ de I^m.

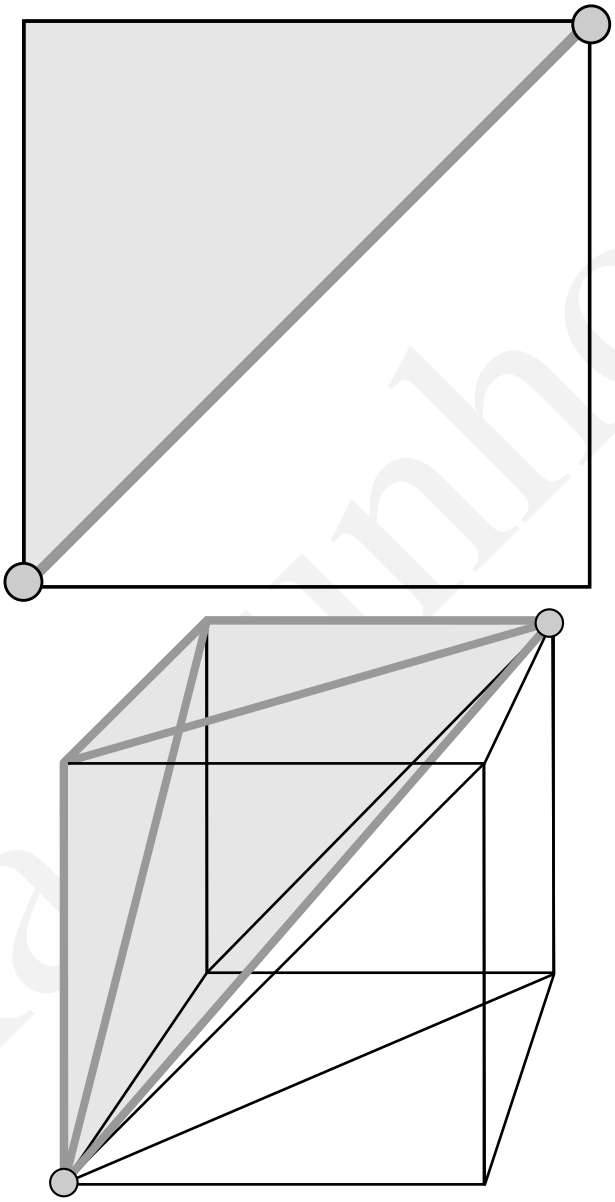


Figura 4.1: À esquerda a triangulação $\mathcal{CFK}$ de I^2 e à direita a triangulação $\mathcal{CFK}$ de I^3.

Para cada $v_0 \in \mathbb{Z}^m$ e π permutação de $\{1, 2, \ldots, m\}$, definimos $\sigma = K_1(v_0, \pi) = [v_0, v_1, \ldots, v_m]$, onde

$$v_i = v_{i-1} + e_{\pi_i} = v_0 + e_{\pi_1} + \cdots + e_{\pi_i}, \;\; i = 1, \ldots, m.$$

A triangulação K_1 de $\mathbb{R}^m$ é formada pelos simplexos $K_1(v_0, \pi)$ e suas faces.

A triangulação J_1 de $\mathbb{R}^m$ é obtida pela reflexão da triangulação $\mathcal{C}FK)$ de $(I^m$, com relação às faces de dimensão $m-1$ de I^m.

Para cada $v_0 \in \{v \in \mathbb{Z}^m \mid v_i \text{ é par }\}$, $s = (s_1, \ldots, s_m) \in \{-1, 1\}^m$ e π permutação de $\{1, 2, \ldots, m\}$, definimos $\sigma = J_1(v_0, \pi, s) = [v_0, v_1, \ldots, v_m]$, onde

$$v_i = v_{i-1} + s_{\pi_i} e_{\pi_i} = v_0 + s_{\pi_1} e_{\pi_1} + \cdots + s_{\pi_i} e_{\pi_i}, \;\; i = 1, \ldots, m.$$

A triangulação J_1 de $\mathbb{R}^m$ é formada pelos simplexos $J_1(v_0, \pi, s)$ e suas faces.

A figura 4.2 apresenta as triangulações K_1 e J_1 de $\mathbb{R}^2$, respectivamente.

Dados dois vetores $v, \delta \in \mathbb{R}^m$ com $\delta_i > 0$, $i = 1, \ldots, m$, pode-se obter as triangulações $v + \delta K_1$ e $v + \delta J_1$, trocando os vetores da base canônica $e_1, \ldots, e_m$ pelos vetores $\delta_1 e_1, \ldots, \delta_m e_m$ nas triangulações K_1 e J_1, e somando a seus vértices o vetor v. Estas triangulações são as triangulaçoões K_1 e J_1 escalonadas e transladadas.

Note-se que $\rho(K_1) = \rho(J_1) = \sqrt{m}$ e $\rho(v+\delta K_1) = \rho(v+\delta J_1) = \sqrt{\delta_1^2 + \cdots + \delta_m^2} = \| \delta \|_2$.

4.2 Os Vértices da Aproximação

Seja uma aplicação $F : U \to \mathbb{R}^n$ de classe C^2 no aberto $U \subset \mathbb{R}^m$ com $m > n$ e seja $\mathcal{M} = \{x \in U \mid F(x) = 0\}$. Suponha que $0 \in \mathbb{R}^n$ é valor regular de F, isto é, se $x \in \mathcal{M}$ então $DF(x)$ tem posto máximo. Neste caso $\mathcal{M}$ é uma variedade de dimensão $m - n$.

Dado um simplexo de dimensão n, $\sigma = [v_0, \ldots, v_n]$ a interseção de $\mathcal{M}$ com σ é aproximada por

$$v = \sum_{i=0}^{n} \lambda_i v_i$$

onde λ é dado por

$$\begin{pmatrix} 1 & \cdots & 1 \\ F(v_0) & \cdots & F(v_n) \end{pmatrix} \begin{pmatrix} \lambda_0 \\ \vdots \\ \lambda_n \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

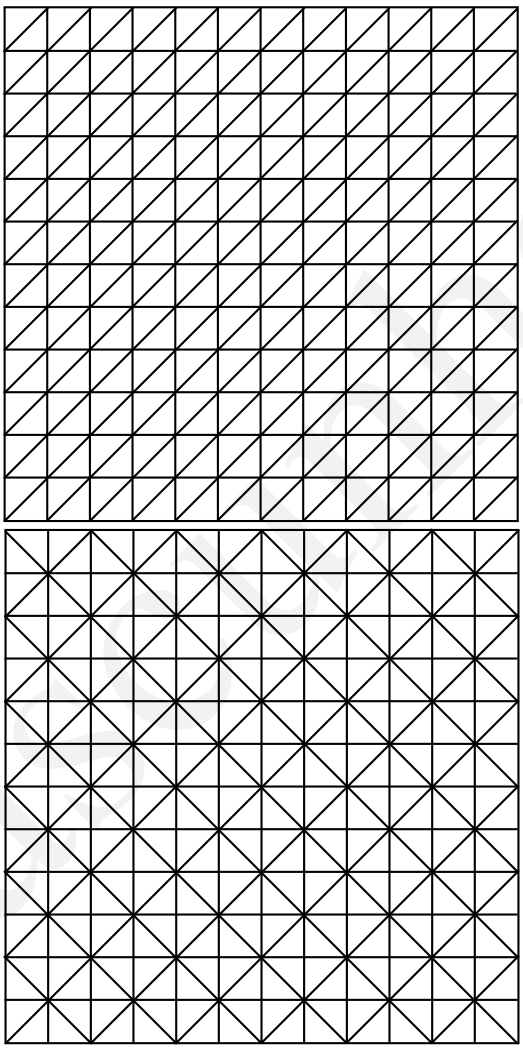


Figura 4.2: À esquerda a triangulação K_1 de $\mathbb{R}^2$ e à direita a triangulação J_1 de $\mathbb{R}^2$.

Caso $\lambda_i \geq 0$, v pertence ao simplexo σ e caso $\lambda_i > 0$, v pertence ao interior do simplexo σ.

Observe que se a matriz for invertível, existe uma única solução para v. Pode-se escolher os vértices do simplexo σ com uma pequena perturbação de modo que a matriz seja invertível e que $\lambda_i \neq 0$, isto é, ou v pertence ao interior de σ ou não pertence a σ.

Para calcular uma aproximação para $\mathcal{M}$ em um simplexo de dimensão m, basta obter o fecho convexo da aproximação para $\mathcal{M}$ em cada face de dimensão n, isto é, escolhendo todas as combinações de n vértices dos m vértices deste simplexo.

4.3 Programas em Matlab/Octave

Dado um paralelepípedo D de $\mathbb{R}^m$, considere a triangulação K_1 deste. Ela pode ser implementada utilizando um produto cartesiano para gerar uma malha cartesiana e depois pela geração das permutações de $\{1, 2, \ldots, m\}$ para obter os simplexos de dimensão m. Novamente podemos escolher as faces de dimensão n de cada simplexo de dimensão m para obter todos os vértices da aproximação para $\mathcal{M}$.

O programa abaixo calcula todas as faces da aproximação $\mathcal{M}_T$ utilizando a triangulação K_1.

```
function MarchingTetrahedron(n, k, First, Last, Division, Func, filename)
   file  = fopen(filename,'w');
   fprintf(file,'%d  %d\n',n,k);

   Delta = (Last-First)./Division;
   Pert  = random('Normal',0,1,2^n,n)*0.000001;
   Base  = InitGrid(n, Division);

   for i = 1:n
        DivisionP(i) = i;
   end
   [BaseP] = InitGrid(n,DivisionP);

   for i = 1:k+1
        DivisionC(i) = n-k+i;
   end
   [BaseC] = InitGrid(k+1,DivisionC);

   for g = 0:Base(n+1)-1
```

```
[Grid] = GridCoords(n,g,Base);
[Vert FVert] = GenVert(n, Grid, Pert, First, Delta, Func);

for s = 0:BaseP(n+1)-1

  [f] = GridCoords(n,s,BaseP);
  f = f+1;
  [P] = Map_Perm(n,f);
  [Simp] = GenLabelSimplex(n,P);

  VertexManifold = [];
  NumberVertex = 0;

  for j = 0:BaseC(k+2)-1

    [C]   = GridCoords(k+1,j,BaseC);
    C     = C+1;
    [lex] = Lexico(k+1,C);

    if lex == 1
        for i = 1:k+1
            Face(i) = Simp(C(i));
        end
        [Vertex trans] = GetVertexManifold(n,k,Face,Vert,FVert);
        if trans == 1
            VertexManifold = [VertexManifold; Vertex];
            NumberVertex = NumberVertex + 1;
        end
    end

  end

  if NumberVertex > 0
    fprintf(file,'%d ',NumberVertex);
    fprintf(file,'\n');
    for j = 1:NumberVertex
      fprintf(file,'%f ',VertexManifold(j,:));
      fprintf(file,'\n');
    end
```

```
        end

      end

   end
   fprintf(file,'-1\n');

   fclose(file);

   return
end


function [Base] = InitGrid(n, Division)
   Base(1) = 1;
   for i = 2:n+1
      Base(i) = Base(i-1)*Division(i-1);
   end
   return
end


function [Grid] = GridCoords(n,i,Base)
   copy = i;
   for j = n:-1:2
      aux     = mod(copy,Base(j));
      Grid(j) = (copy-aux)/Base(j);
      copy    = aux;
   end
   Grid(1) = copy;
   return
end


function [Vert FVert] = GenVert(n, Grid, Pert, First, Delta, Func)
   Vert  = [];
   FVert = [];
   for i = 0:2^n-1
      [Coords]    = HyperCubeCoords(n,i);
      [CoordPert] = HyperCubePert(n,Grid,Coords,Pert);
```

```
      [VHC]       = HyperCube(n,First,Delta,Grid,Coords,CoordPert);
      Vert        = [Vert; VHC];
      [FVHC]      = Func(n,VHC);
      FVert       = [FVert; FVHC];
   end
   return
end


function [Coords] = HyperCubeCoords(n,i)
   copy = i;
   for j = 1:n-1
      Coords(j) = mod(copy,2);
      copy      = (copy-Coords(j))/2;
   end
   Coords(n) = copy;
   return
end


function [CoordPert] = HyperCubePert(n,Grid,Coords,Pert)
   pot = 1;
   label = 0;
   for i = 1:n
      p     = abs(Coords(i) - mod(Grid(i),2));
      label = label + pot*p;
      pot   = 2*pot;
   end
   CoordPert = Pert(label+1,:);
   return
end


function [VHC] = HyperCube(n,First,Delta,I,Coords,Pert)
   for i = 1:n
      VHC(i) = First(i) + (I(i)+Coords(i))*Delta(i) + Pert(i);
   end
   return
end
```

```
function [Vertex trans] = GetVertexManifold(n,k,Face,Vert,FVert)
   for i = 1:k+1
      A(1,i) = 1;
      A(2:k+1,i) = FVert(Face(i)+1,:);
   end
   b = [1; zeros(k,1)];
   lamb = A\b;
   trans = 0;
   Vertex = zeros(1,n);
   if lamb >= 0
       for i = 1:k+1
          Vertex = Vertex + lamb(i)*Vert(Face(i)+1,:);
       end
       trans = 1;
   end
   return
end


function [V FV] = GenVertHyperCube(n,Grid,Pert,First,Delta,Label,Func)
   [Coords]    = HyperCubeCoords(n,Label);
   [CoordPert] = HyperCubePert(n,Grid,Coords,Pert);
   [V]         = HyperCube(n,First,Delta,Grid,Coords,CoordPert);
   [FV]        = Func(n,V);
   return
end


function [F] = Map_Perm(n,f)
    for i = 1:n
        F(i) = 0;
    end
    for i = 1:n
        if F(f(i)) == 0
            F(f(i)) = i;
        else
            for j = n:-1:f(i)+1
                F(j) = F(j-1);
            end
```

```
            F(f(i)) = i;
        end
    end
    return
end

function [lex] = Lexico(k,f)
    for i = 1:k-1
        if f(i) >= f(i+1)
            lex = 0;
            return
        end
    end
    lex = 1;
    return
end

function [Simp] = GenLabelSimplex(n,P)
    Simp(1) = 0;
    for i = 1:n
        Simp(i+1) = Simp(i)+2^(P(i)-1);
    end
end
```

A seguir exibe-se um programa para gerar uma aproximação da esfera de dimensão 3 em $\mathbb{R}^4$ com centro na origem e raio 1 utilizando o programa descrito anteriormente:

```
function Esf

    First    = [-1.1 -1.1 -1.1 -1.1];
    Last     = [1.1 1.1 1.1 1.1];
    Division = [2 2 2 2];

    MarchingTetrahedron(4,1,First,Last,Division,@esfera,esfera.pol');

    return

    function [f] = esfera(n,v)
      f(1) = v(1)*v(1)+v(2)*v(2)+v(3)*v(3)+v(4)*v(4)- 1;
      return
```

```
    end

 end
```

A um trecho da saída que está escrita no arquivo *esfera.pol é dada a seguir:*

```
4  1
4
-0.303031 -0.303030 -0.303029 -0.303030
-0.303031 -0.303030 -0.303029 -0.000000
-0.454546 -0.454546 0.000001 -0.000000
-0.909091 0.000001 -0.000000 -0.000000
4
-0.303031 -0.303030 -0.303029 -0.303030
-0.303031 -0.303030 -0.303029 -0.000000
-0.454546 -0.454546 0.000001 -0.000000
0.000000 -0.909092 -0.000002 0.000000
4
-0.303031 -0.303030 -0.303029 -0.303030
-0.303031 -0.303030 -0.303029 -0.000000
-0.454546 0.000000 -0.454545 -0.000000
-0.909091 0.000001 -0.000000 -0.000000
4
-0.303031 -0.303030 -0.303029 -0.303030
-0.303031 -0.303030 -0.303029 -0.000000
-0.000001 -0.454545 -0.454545 -0.000000
0.000000 -0.909092 -0.000002 0.000000
4
-0.303031 -0.303030 -0.303029 -0.303030
-0.303031 -0.303030 -0.303029 -0.000000
-0.454546 0.000000 -0.454545 -0.000000
0.000000 -0.000000 -0.909090 -0.000000
4
-0.303031 -0.303030 -0.303029 -0.303030
-0.303031 -0.303030 -0.303029 -0.000000
-0.000001 -0.454545 -0.454545 -0.000000
0.000000 -0.000000 -0.909090 -0.000000
4
-0.303031 -0.303030 -0.303029 -0.303030
-0.454546 -0.454546 0.000001 -0.454546
```

```
-0.454546 -0.454546 0.000001 -0.000000
-0.909091 0.000001 -0.000000 -0.000000
```

Observe que todas as células convexas definidas pelos pontos (4 neste exemplo) são tetraedros, mas poderiam ser prismas, piramides, hexaedros, entre outros.

Um outro exemplo é o toro de dimensão 2 em $\mathbb{R}^4$*, definido por*

$$x^2 + y^2 = 1$$

$$z^2 + w^2 = \frac{1}{4}$$

```
function Tor

   First    = [-1.1 -1.1 -0.51 -0.51];
   Last     = [1.1 1.1 0.51 0.51];
   Division = [2 2 2 2];

   MarchingTetrahedron(4,2,First,Last,Division,@toro, toro.pol');

   return

   function [f] = toro(n,v)
     f(1) = v(1)*v(1) + v(2)*v(2) - 1;
     f(2) = v(3)*v(3) + v(4)*v(4) - 0.25;
     return
   end

end
```

A um trecho da saída que está escrita no arquivo toro.pol é dada a seguir:

```
4  2
3
-0.454546 -0.454546 -0.279453 -0.210744
-0.454546 -0.454545 -0.490197 0.000001
-0.909090 -0.000001 -0.490197 0.000001
3
-0.454546 -0.454546 -0.279453 -0.210744
-0.454546 -0.454545 -0.490197 0.000001
```

```
0.000001 -0.909090 -0.490196 0.000001
3
-0.528644 -0.380448 -0.245099 -0.245098
-0.909090 -0.000001 -0.421487 -0.068709
-0.909090 -0.000000 -0.245099 -0.245097
3
-0.380450 -0.528642 -0.245099 -0.245098
0.000001 -0.909090 -0.421488 -0.068707
0.000000 -0.909092 -0.245098 -0.245097
4
-0.528644 -0.380448 -0.245099 -0.245098
-0.909090 -0.000001 -0.421487 -0.068709
-0.454546 -0.454546 -0.279453 -0.210744
-0.909090 -0.000001 -0.490197 0.000001
4
-0.380450 -0.528642 -0.245099 -0.245098
0.000001 -0.909090 -0.421488 -0.068707
-0.454546 -0.454546 -0.279453 -0.210744
0.000001 -0.909090 -0.490196 0.000001
.
.
.
3
0.571360 -0.337731 -0.245099 -0.245097
0.475905 -0.433186 -0.289355 -0.200841
0.380452 -0.528640 -0.245098 -0.245097
4
0.571360 -0.337731 -0.245099 -0.245097
0.475905 -0.433186 -0.289355 -0.200841
0.909091 -0.000001 -0.401685 -0.088511
0.909091 -0.000001 -0.490197 0.000001
5
0.000002 -0.909091 -0.068705 -0.421490
0.000002 -0.909092 -0.000001 -0.490194
0.475906 -0.433186 -0.200842 -0.289353
0.042724 -0.866370 -0.000001 -0.490194
0.380452 -0.528640 -0.245098 -0.245097
3
0.571360 -0.337731 -0.245099 -0.245097
0.475906 -0.433186 -0.200842 -0.289353
```

```
0.380452 -0.528640 -0.245098 -0.245097
3
0.571360 -0.337731 -0.245099 -0.245097
0.909091 -0.000000 -0.245099 -0.245097
0.909091 -0.000001 -0.401685 -0.088511
3
0.475906 -0.433186 -0.200842 -0.289353
0.042724 -0.866370 -0.000001 -0.490194
0.909092 0.000001 -0.000000 -0.490196
4
0.571360 -0.337731 -0.245099 -0.245097
0.475906 -0.433186 -0.200842 -0.289353
0.909091 0.000001 -0.088513 -0.401683
0.909092 0.000001 -0.000000 -0.490196
```

Observe que todas as células convexas desta saída são triângulos, quadriláteros e pentágonos.

O programa a seguir gera uma aproximação da esfera com centro na origem e raio 1 em $\mathbb{R}^3$

```
function Esf

   First    = [-1.1 -1.1 -1.1];
   Last     = [1.1 1.1 1.1];
   Division = [10 10 10];

   MarchingTetrahedron(3,1,First,Last,Division,@esfera,'esfera.pol');

   return

   function [f] = esfera(n,v)
     f(1) = v(1)*v(1)+v(2)*v(2)+v(3)*v(3) - 1;
     return
   end

end
```

A figura 4.3 apresenta o resultado deste programa

Figura 4.3: Aproximação da esfera unitária em $\mathbb{R}^3$.

É possível gerar todas as faces destas células sem o uso de algoritmos tradicionais de fecho convexo, observando que as os vértices da célula estão nas faces de dimensão n do simplexo, as arestas estão nas faces de dimensão $n+1$ do simplexo, as faces de dimensão 2 da célula estão nos simplexos de dimensão $n+2$ e assim por diante. Somente os rótulos dos simplexos são suficientes para esta geração hierárquica denominada de esqueleto combinatório dos elementos da variedade [3].

Por fim, para gerar uma aproximação utilizando a triangulação J_1 basta gerar uma função sinal e modificar a função

```
function [VHC] = HyperCube(n,First,Delta,I,Coords,Pert)
```

como descrito na Seção 4.1.

Capítulo 5

Demonstrações Assistidas por Computador

Como foi visto no Capítulo 1, pode ser necessário saber como avaliar uma função contínua f em um ponto arbitrário do seu domínio a partir apenas de um conjunto finito de amostras, ou mesmo ajustar uma função aos dados para inferir um comportamento mais geral. Podem ser encontradas duas formas de contornar esses problemas: métodos de interpolação e métodos de aproximação de funções. Em ambos, parte-se de um conjunto finito de tuplas $\mathrm{X} = \{(x_i, y_i), i = 1, \ldots, m\}$ com o objetivo de determinar uma F, como combinação de funções mais simples, que se ajuste aos pontos amostrados, obtendo comportamento similar à f. Assumindo que $y_i = f(x_i)$.

No primeiro caso, tem-se a restrição que, fixado um x_i, $F(x_i) = f(x_i)$, ou seja, define-se que a função a ser determinada F *interpole* o conjunto X de pontos dados. Além disso, nos casos de ajuste, o fato de interpolar os pontos restringe bastante a classe de funções possíveis para ajuste, desconsiderando eventuais erros nos dados, como erros de medição.

No segundo caso, essa condição de interpolação é relaxada para que, fixado um x_i, $F(x_i) = f(x_i) + \epsilon_i$, ou seja, a função se *aproxime* dos dados, não necessariamente passando diretamente pelas tuplas definidas em X.

Cada um dos casos possui aplicações específicas, sendo fora do escopo deste trabalho efetuar essa análise. Além disso, apesar de algumas construções que serão explicitadas nas próximas seções serem úteis para ambos os casos, o foco será dado ao segundo caso apenas.

5.1 Método dos mínimos quadrados

Como já bem conhecido na literatura [16, 17], o ajuste por mínimos quadrados pode, por exemplo, determinar uma função $F : \mathbb{R} \to \mathbb{R}$ que melhor aproxime o conjunto de dados (x_i, y_i), com

$$y_i = f(x_i) \; e \; F(x_i) = f(x_i) + \epsilon_i,$$

de forma que o erro de aproximação (ou resíduo)

$$E = \sum_i \|y_i - F(x_i)\|^2 = \sum_i \|\epsilon_i\|^2$$

seja o menor possível.

A construção da função F pode ser feita a partir de combinações de uma base de outras funções, de forma que

$$F(x) = \alpha_0\phi_0(x) + \alpha_1\phi_1(x) + \ldots + \alpha_n\phi_n(x)$$

Uma escolha bastante comum consiste em definir a base de funções $\{\phi_i\}$ com uma base de polinômios. Essa escolha, em geral, deve-se ao fato da facilidade de derivação, integração e boa capacidade de ajuste a uma série de conjuntos de dados. Porém, deve-se salientar que, a depender do problema, a escolha dessa base não é trivial e pode levar aproximações ruins.

As construções que seguem utilizam-se da aproximação polinomial de funções por mínimos quadrados.

5.1.1 Polinômios univariados: O caso real ($\mathbb{R}$)

Utilizando como exemplo o caso ilustrado na Figura 5.1, onde os pontos do conjunto de amostras X distribuem-se próximo a uma reta, é razoável utilizar como base uma função polinomial de grau um para definir F, na forma $F(x) = \alpha_0 + \alpha_1 x$. Assim, para cada tupla (α_0^k, α_1^k), há uma reta $F_k(x) = \alpha_0^k + \alpha_1^k x$ associada e então a tupla para a qual E^k seja o menor possível deve ser determinada.

A ideia de melhor ajuste corresponde a determinar F com o menor resíduo possível, com

$$F(x_i) = y_i + \epsilon_i$$

Reorganizando o problema tem-se o seguinte sistema sobredeterminado:

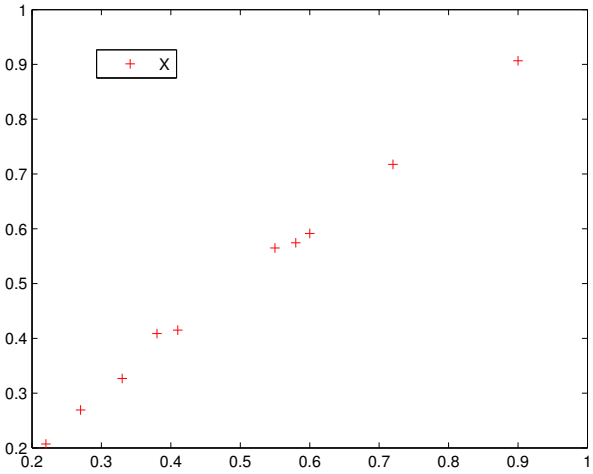


Figura 5.1: Pontos distribuidos de forma linear.

$$\begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_m \end{pmatrix} \begin{pmatrix} \alpha_0^k \\ \alpha_1^k \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix} \tag{5.1}$$

$$\mathbf{X}\boldsymbol{\alpha}^k = \mathbf{Y} \tag{5.2}$$

A partir daí, como pode ser visto em [16], a tupla $(\alpha_0^k, \alpha_1^k) = \boldsymbol{\alpha}^k$ que melhor aproxima os m pontos dados é obtida utilizando as equações normais,

$$\mathbf{X}^T\mathbf{X}\boldsymbol{\alpha}^k = \mathbf{X}^T\mathbf{Y} \Leftrightarrow \boldsymbol{\alpha}^k = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$$

Assim, é obtida uma função $F \in \mathcal{P}_1(\mathbb{R})$, ou seja, uma função polinomial de grau $n \leq 1$ e o problema está resolvido.

Observe que

$$F(x) = \alpha_0 + \alpha_1 x = \begin{pmatrix} 1 \\ x \end{pmatrix}^T \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix} = \mathbf{b}^T\boldsymbol{\alpha},$$

ou seja, o vetor $\boldsymbol{\alpha}$ corresponde às coordenadas da função F na base B do espaço vetorial de polinômios $\mathcal{P}_1(\mathbb{R})$.

De uma forma mais geral, quando é considerada uma aproximação de f por um polinômio F com grau máximo n, tem-se que

$$f(x) \approx \alpha_0 + \alpha_1 x + \ldots + \alpha_n x^n = \begin{pmatrix} 1 \\ x \\ \vdots \\ x^n \end{pmatrix}^T \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix} = \mathbf{b}^T \boldsymbol{\alpha} = F(x)$$

Assim, dado o conjunto de pontos X formado pelas tuplas $(x_1, y_1), \ldots, (x_m, y_n)$, para determinar $\boldsymbol{\alpha} = (\alpha_0, \ldots, \alpha_n)$, primeiro constrói-se a base de $\mathcal{P}_n(\mathbb{R})$, a saber $\mathrm{B} = \{1, x, x^2, \ldots, x^n\}$, contendo $n+1$ elementos. Em seguida, deve ser definida a matriz **B** como sendo a avaliação de cada um dos m pontos de X na base B.

$$\mathbf{B} = \begin{pmatrix} 1 & x_1 & \ldots & x_1^n \\ 1 & x_2 & \ldots & x_2^n \\ \vdots & & & \\ 1 & x_m & \ldots & x_m^n \end{pmatrix},$$

onde, analogamente à equação 5.1, tem-se

$$\begin{pmatrix} 1 & x_1 & \ldots & x_1^n \\ 1 & x_2 & \ldots & x_2^n \\ \vdots & \vdots & & \\ 1 & x_m & \ldots & x_m^n \end{pmatrix} \begin{pmatrix} \alpha_0^k \\ \alpha_1^k \\ \vdots \\ \alpha_n^k \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix} \tag{5.3}$$

(5.4)

e portanto

$$\begin{pmatrix} \alpha_0^k \\ \alpha_1^k \\ \vdots \\ \alpha_n^k \end{pmatrix} = \left[\begin{pmatrix} 1 & x_1 & \ldots & x_1^n \\ 1 & x_2 & \ldots & x_2^n \\ \vdots & \vdots & & \\ 1 & x_m & \ldots & x_m^n \end{pmatrix}^T \begin{pmatrix} 1 & x_1 & \ldots & x_1^n \\ 1 & x_2 & \ldots & x_2^n \\ \vdots & \vdots & & \\ 1 & x_m & \ldots & x_m^n \end{pmatrix} \right]^{-1} \begin{pmatrix} 1 & x_1 & \ldots & x_1^n \\ 1 & x_2 & \ldots & x_2^n \\ \vdots & \vdots & & \\ 1 & x_m & \ldots & x_m^n \end{pmatrix}^T \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix} \tag{5.5}$$

Com essa solução apresentada na equação 5.5, é possível aproximar um conjunto de pontos quaisquer na forma $(x_i, f(x_i)) \in \mathbb{R}^2, i = 1, \ldots, m$ por um polinômio de grau máximo n.

Em seguida é descrita uma possível implementação da solução discutida:

```
function alpha = ls_polinomial(points, degree)
        B = base_polinomios(points,degree);
        B_t = B';
        alpha = inv(B_t*B)*(B_t*points(:,end));

        % Opcional para visualizacao
        alpha = fliplr(alpha');
        f_ls = polyval(alpha, points(:,1));
        plot(points(:,1), points(:,2), 'r+', points(:,1), f_ls,'-');
        return
end
```

Na Figura 5.2(a), é ilustrado o conjunto X e um polinômio F de grau um, que melhor se ajusta a esses pontos no sentido de mínimos quadrados, com a função f não conhecida. Além disso, na Figura 5.2(b) é exibido a avaliação de F em muito mais pontos que do que o seu conjunto X, amostrado em uma função conhecida, a saber $f(x) = x^2$.

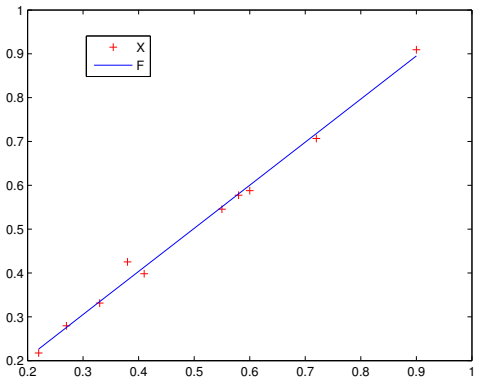
Base de $\mathcal{P}_n(\mathbb{R})$

Ao definir o grau máximo n do polinômio que será utilizado para o ajuste às amostras, faz-se necessário determinar a base B de $\mathcal{P}_n(\mathbb{R})$, que como já foi visto na seção anterior corresponde à $\mathrm{B} = \{1, x, x^2, \ldots, x^n\}$.

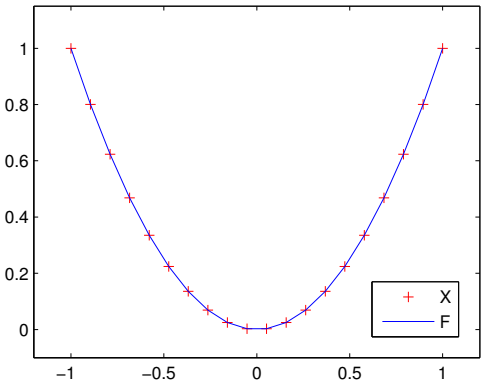
Uma possível implementação e avaliação nas amostras é ilustrada no código que segue:

```
function B = base_polinomios(points, degree)
        m = length(points);
        B = zeros(m,degree+1);

        for i = 1:m
                for j = 1:degree+1
                        B(i,j) = points(i,1)^(j-1);
                end
        end
        return
end
```



(a)



(b)

Figura 5.2: Exemplos de aproximações polinomiais.

5.1.2 Polinômios multivariados: O caso geral ($\mathbb{R}^d$)

Como pôde-se observar, polinômios univariados podem representar curvas no plano.
Quando deseja-se estender o processo para dimensões maiores, para superfícies por exemplo, F passa a ser escrita como

$$F(\mathbf{x}) = \alpha_0\phi_0(\mathbf{x}) + \alpha_1\phi_1(\mathbf{x}) + \ldots + \alpha_n\phi_n(\mathbf{x}),$$

com $\mathbf{x} \in \mathbb{R}^{d-1}$, ou seja, $\mathbf{x} = (x_1, x_2, \ldots, x_{d-1})$. Assim, as funções ϕ_i passam a depender de várias variáveis. Uma escolha possível é ainda utilizar polinômios, mas multivariados.

Para efetuar a aproximação efetua-se uma construção análoga ao processo descrito na seção 5.1.1, porém com uma decomposição $(\mathbf{x}_i, f(\mathbf{x}_i)), \mathbf{x}_i \in \mathbb{R}^{d-1}$ e $f(\mathbf{x}_i) \in \mathbb{R}$, ou seja, a aproximação é feita na função altura de cada ponto. Para simplificar as exposições, será considerado aqui que f consiste na coordenada de um vetor da base paralelo a algum vetor à base canônica de $\mathbb{R}^d$, não sendo, dessa forma, necessário em falar de vetores normais a hiperplanos. Quando $d = 2, f$ corresponde a coordenada de $\mathbf{e}_2$ (eixo y); $d = 3$, coordenada de $\mathbf{e}_3$ (eixo z) e por aí vai.

Partindo de um exemplo mais simples, seja $F \in \mathcal{P}_2(\mathbb{R}^2)$, ou seja, um polinômio com $d = 2$, $g = 2$ e $\mathbf{p} = (x, y) \in \mathbb{R}^2$, então

$$F(\mathbf{p}) = \alpha_0 + \alpha_1 x + \alpha_2 y + \alpha_3 x^2 + \alpha_4 xy + \alpha_5 y^2 = \begin{pmatrix} 1 \\ x \\ y \\ x^2 \\ xy \\ y^2 \end{pmatrix}^T \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \end{pmatrix} = \mathbf{b}^T\boldsymbol{\alpha},$$

e a matriz $\mathbf{B}$ passa a ser escrita como

$$\mathbf{B} = \begin{pmatrix} 1 & x_1 & y_1 & x_1^2 & x_1y_1 & y_1^2 \\ 1 & x_2 & y_2 & x_2^2 & x_2y_2 & y_2^2 \\ \vdots & & & & & \\ 1 & x_m & y_m & x_m^2 & x_my_m & y_m^2 \end{pmatrix},$$

Ao considerar o caso mais geral, uma vez que estamos em $\mathbb{R}^d$, faz-se necessário determinar uma base do espaço $\mathcal{P}_g(\mathbb{R}^d)$ de polinômios multivariados de d variáveis e grau máximo g.

Base de $\mathcal{P}_n(\mathbb{R}^d)$

Ainda utilizando o exemplo descrito na seção 5.1.2, para determinar B, pode ser efetuado o produto da combinação dos elementos $\{1, x, y\}$, tomados dois à dois, sem repetição. Assim tem-se

$$\begin{aligned}\{1, x, y\} \times \{1, x, y\} &= \{1.1, 1.x, 1.y, x.1, x.x, x.y, y.1, y.x, y.y\} \\ &= \{1, x, y, x, x^2, xy, y, yx, y^2\}\end{aligned}$$

e eliminando as repetições, a base é determinada por $\mathrm{B} = \{1, x, y, x^2, xy, y^2\}$.

De forma análoga, no caso geral, B seria determinada pelo produto da combinação dos elementos $\{x_0,\ x_1,\ x_2, \ldots, x_d\}$, tomados g à g, sem repetição

$$\{x_0,\ x_1,\ x_2, \ldots, x_d\} \overbrace{\times \ldots \times}^{\text{(g-1) vezes}} \{x_0,\ x_1,\ x_2, \ldots, x_d\}$$

onde g representa o grau máximo do polinômio.

Com isso, a construção de um algoritmo para geração da base, como feito na seção 5.1.1, não é imediata, pois teria de ser construído um ninho de laços com $g - 1$ laços:

```
function B = base_polinomios(points, degree)
   m = length(points);
   dim = size(points,2);
   dim_base_pol = nchoosek(dim+degree,degree);
   B = zeros(m,dim_base_pol);
   B(:,1) = 1;

   for i = 1:m
       for j2 = 2:dim_base_pol
           ...
           for jg = jk:dim_base_pol
                B(i,j2) = points(i,j2)*points(i,j3)*...*points(i,jg)
           end
        end
   end
   return
end
```

Uma forma de resolver isso é através da associação de cada elemento de $W =$

$\{1, x, y\}$ com sua respectiva posição, gerando o conjunto de índices $J = \{1, 2, 3\}$. A partir disso, é efetuado um processo de enumeração como descrito na seção 3.3.3, que remove os elementos dos $g - 1$ produtos cartesianos $I_1 \times I_2 \times \ldots \times I_g$ que não estão em ordem não decrescente, gerando um novo conjunto de índices I. Assim, para $\mathcal{P}_2(\mathbb{R}^2)$ teremos o conjunto $W = \{1, x, y\}$. Logo

$$I = \{(1,1), (1,2), (2,2), (1,3), (2,3), (3,3)\}$$

Com isso, a base fica determinada por

$$\begin{aligned} \mathrm{B} &= \{W(1) * W(1), W(1) * W(2), W(2) * W(2), W(1) * W(3), W(2) * W(3), W(3) * W(3)\} \\ &= \{1, x, x^2, y, xy, y^2\} \end{aligned}$$

Note que essa construção é geral, podendo ser utilizada para geração da base de $\mathcal{P}_g(\mathbb{R}^d)$. Um fato importante é saber que a dimensão do espaço $\mathcal{P}_g(\mathbb{R}^d) = \binom{d+g}{g}$ [19].

Em seguida está descrita uma possível implementação do código para geração dos elementos da base:

```
function [P] = Enumerate_Base(n,k)
    P = [];
    for i = 1:k
        Division(i) = n;
    end
    [Base] = InitGrid(k,Division);
    for j = 0:Base(k+1)-1
        [f] = GridCoords(k,j,Base);
        f = f+1;
        [lex] = Lexico(k,f);
        if lex == 1
           P   = [P; f];
        end
    end
    return
end
```

Vale salientar aqui que a função *Lexico* possui diferenças com a definida na seção 3.3.3:

```
function [lex] = Lexico(k,f)
    for i = 1:k-1
        if f(i) > f(i+1)
            lex = 0;
            return
        end
    end
    lex = 1;
    return
end
```

Uma vez gerado o conjunto I, a geração da matriz **B**, de avaliação dos pontos na base, pode ter a seguinte implementação:

```
function B = base_polinomios_rd(points, degree)
   m = length(points);
   dim = size(points,2);
   dim_base_pol = nchoosek(dim+degree,degree)
   B = zeros(m,dim_base_pol);
   P = [ones(m,1) points];
   indices_base = Enumerate_Base(dim+1, degree);

   for i = 1:m
       for j = 1:dim_base_pol
           value = 1;
           for k = 1:degree
               value = value * P(i,indices_base(j,k));
           end
           B(i,j) = value;
       end
   end
   return
end
```

Deve ser tomado cuidado com a ordem dos coeficientes quando for efetuada a avaliação do polinômio em um ponto qualquer do domínio, após obter a base B e em seguida o vetor $\boldsymbol{\alpha}$, usando as construções descritas.

Lista de Figuras

Lista de Tabelas

Bibliografia

[1] ALLGOWER, E; GEORG, K - Numerical Continuation Methods - Sringer Verlag (1990). 11, 13, 29

[2] BLOOMENTHAL - Polygonization of Implicit Surfaces - CAGD 5 (1988) 341-355. 8

[3] CASTELO, A. - Aproximações Adaptativas de Variedades Variedades Implicitas e Aplcações em Modelagem Implícita e Equações Algébrico-Diferenciais - Tese de Doutorado - PUC - Rio (1992). 8, 11, 29, 44

[4] CASTELO, A. ; NONATO, L. G. ; SIQUEIRA, M. ; MINGHIM, R. ; TAVARES, G. . The J1a Triangulation: an adaptive triangulation in any dimension. Computers & Graphics, 30 (5) (2006), 737-753. 8, 11, 29

[5] COXETER, H. S. M. - Discrete Groups Generated by Reflections - Annals of Mathematics, 35 (1934) 588-621. 29

[6] EVEN, S. - Algorithmic Combinatorics - The Macmillan Company (1973). 7, 15

[7] FREUDENTHAL, H. - Simplizalzerlegungen von Beschränkter Flachheit - Annals of Mathematics 43 (1942) 580-582. 29

[8] HALL, M.; WARREN, J. - Adaptative Tesselation of Implicitly Defined Surfaces - Technical Report, Rice Comp TR 88-84 (1990).

[9] KUHN, H. W. - Simplicial Approximation of Fixed Points - Proceedings of National Academy of Sciences of United States of America, 61 (1968) 1238-1242. 29

[10] LEWINER, T.; LOPES, H; VIEIRA, A. W; TAVARES, G. - Efficient implementation of Marching Cubes? cases with topological guarantees, Journal of Graphics Tools, (2003) 8 (2) 1-15. 8

[11] LORENSEN, W. E.; CLINE, H. E. - Marching cubes: A high resolution 3d surface construction algorithm, SIGGRAPH, (1987) 163-169. 8

[12] MOORE, D.; WARREN, J. - Multidimensional Adaptative Mesh Generation - Technical Report, Rice Comp TR 90-106 (1990). 8

[13] NIJENHUIS, A.; WILF, H. S. - Combinatorial Algorithms - Academic Press (1978). 7, 15

[14] PAGE, E. S.; WILSON, L. B. - An Introduction to Computational Combinatorics - Cambridge Computer Science Text - 9 (1979). 7, 15

[15] SCARF, H. - On Piecewise Linear Aproximation to Smooth Mappings - SIAM J. App. Math. 15 (5) (1967) 1328-1343. 8

[16] TREFETHEN, L.N.; BAU, D. - Numerical Linear Algebra - SIAM (1997), ISBN: 9780898714876. 9, 46, 47

[17] PAIVA, A., SOUZA, F.S., Do método de mínimos quadrados até funções de base radial. Disponível em: <http://www2.icmc.usp.br/∼apneto/download/mmq2rbf.pdf>. Acessado em: 12 de dezembro de 2012. 46

[18] WEISS, K.; FLORIANI, L. - Simplex and Diamond Hierarchies: Models and Applications, Computer Graphics Forum, 30 (8) (2011) 2127-2155. 8

[19] COX, D.A.; LITTLE, J.; O'SHEA, D. - Using Algebraic Geometry, Graduate Texts in Mathematics - Springer (2005), ISBN: 9780387207063. 53