Quantização por Aproximações Sucessivas

Décio Angelo Fonini Junior

Tese submetida ao Corpo Docente da Coordenação dos Programas de Pós-Graduação de Engenharia da Universidade Federal do Rio de Janeiro como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências em Engenharia Elétrica.

Aprovada por:

Eduardo Antônio Barros da Silva, Ph. D.

Marcos Craizer, D. Sc.

Gelson Vieira Mendonça, Ph. D.

Roosevelt José Dias, Ph. D.

Sérgio Lima Netto, Ph. D.

Abraham Alcaim, Ph. D.

Hélio Cortes Vieira Lopes, Ph. D.

Rio de Janeiro, RJ – Brasil

Março de 2003

Fonini Jr., Décio Angelo

Quantização por Aproximações Sucessivas [Rio de Janeiro] 2003

XIV, 158 pp 29,7 (COPPE/UFRJ, D. Sc., Engenharia Elétrica, 2003)

Tese — Universidade Federal do Rio de Janeiro, COPPE

1. Processamento Digital de Sinais 2. Compressão de Imagens 3. Aproximações Sucessivas

I. COPPE/UFRJ II. Título (Série)

Agradecimentos

Aos meus pais, por tudo, e especialmente por indicar o caminho e ensinar a percorrê-lo. À Paula, por me acompanhar neste caminho. E ao Pedro & Gabriel, por fazerem tudo isto valer a pena.

Em que pese a opinião de Umberto Eco — não se deve agradecer ao orientador, que não fez mais do que sua obrigação — qualquer aluno de pós-graduação sabe muito bem que o orientador faz toda a diferença entre se chegar ao final com uma saúde mental ainda razoável ou não. E eu tive a sorte de contar não com um, mas dois orientadores cuja paciência e camaradagem estão além de qualquer agradecimento. Assim mesmo, obrigado! Grande parte dos bons resultados obtidos ao longo deste trabalho se deve a eles. Não, os erros não.

Finalmente, gostaria de expressar minha felicidade em ter encontrado entre os professores e alunos do LPS um grupo com o qual tanto me identifiquei, e que pelo convívio tornou esta uma época excelente. Obrigado a todos.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D. Sc.)

Quantização por Aproximações Sucessivas

Décio Angelo Fonini Jr.

Março / 2003

Orientador: Eduardo Antônio Barros da Silva

Co-Orientador: Marcos Craizer

Programa de Engenharia Elétrica

Muitos dos métodos mais modernos de compressão de imagem e vídeo utilizam quantização por Aproximações Sucessivas. Neste trabalho é desenvolvida a teoria das α-expansões — um método de aproximações sucessivas que pode ser visto como uma generalização da representação usual de números reais como uma expansão decimal ou binária. O método é aplicado na codificação de coeficientes de transformadas Wavelet, mas a teoria é genérica e pode aplicar-se a uma gama mais ampla de sinais. A teoria elucida os resultados obtidos com métodos de quantização escalar e vetorial por Aproximações Sucessivas publicados na literatura especializada, e indica possíveis melhorias de desempenho nestes métodos. Os experimentos realizados com base nestas indicações comprovam o potencial das α-expansões na obtenção de melhores desempenhos taxa-distorção.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Doctor of Science (D. Sc.)

Successive Approximation Quantization

Décio Angelo Fonini Jr.

March / 2003

Advisors: Eduardo Antônio Barros da Silva
Marcos Craizer

Department: Electrical Engineering

Many of the modern image and video compression methods use Successive Approximation quantization. This work presents the theory of α-expansions — a successive approximations method which can be seen as a generalization of the usual number representation as binary or decimal expansions. The method is applied to the coding of Wavelet Transform coefficients. However, the theory is general and can be applied to a larger spectrum of signals. The theory helps to explain the results obtained by Successive Approximation methods which use both scalar and vector quantization, as published in the literature. It also points out possible performance improvements for these methods. Experiments carried out based on this theory show that α-expansions can effectively generate rate-distortion gains relative to the usual methods.

Sumário

Lista de Figuras

Capítulo 1

Introdução

"Uma imagem vale por mil palavras."

— Frederick R. Barnard

Imagens e vídeo consomem uma enorme fatia da miríade de informações que são armazenadas e transmitidas no mundo informatizado. Assim, no vasto campo de pesquisas relativas ao processamento digital de sinais, é de se compreender que a busca de uma codificação eficiente de imagens e vídeo tenha um papel de destaque. Eficiência, neste contexto, significa ocupar menos espaço e ao mesmo tempo permitir a recuperação do sinal original — ou pelo menos uma boa aproximação dele. Assim, desejamos uma pequena distorção *e* uma pequena taxa de bits. Infelizmente, tais requisitos trabalham em direções opostas: em geral, quanto maior a qualidade desejada para uma imagem, mais bits ela vai requerer.

1.1 Processamento Digital de Sinais

Para melhor situar o presente trabalho dentro desse campo, é interessante realizarmos uma breve revisão de alguns dos assuntos envolvidos, particularmente os relativos ao processamento de imagens.

1.1.1 Sinais

Sinais, neste contexto, são usualmente manifestações de fenômenos físicos como sons ou imagens. Em geral, são funções contínuas do tempo ou do espaço. Freqüentemente temos a necessidade de armazenar ou transmitir tais sinais, para sua reprodução posterior. Obviamente, os meios usuais para isso são computadorizados, de forma que devemos *digitalizar* tais sinais. Normalmente isto é realizado através da amostragem (medição do valor instantâneo do sinal de tempos em tempos, ou de ponto a ponto) e posterior quantização fina do sinal (atribuição de um valor discreto de alta precisão à medição). Tal procedimento gera longas seqüências de valores, normalmente organizados em vetores ou matrizes. Para todos os propósitos práticos, estas seqüências serão consideradas como sendo os sinais "originais", ou seja, aqueles que desejamos recuperar após terem sido codificados. Afinal, a partir desta representação digital é possível reproduzir o fenômeno físico original; isto é o que acontece, por exemplo, quando escutamos um CD de áudio.

1.1.2 Domínios

A representação de um sinal digitalizado como descrito acima se diz no domínio do tempo (ou espaço). No entanto, há outras opções.

Algumas classes de funções podem ser decompostas em somas ponderadas de senos e cossenos. Podemos aplicar isto aos sinais, decompondo-os em somas de freqüências puras. As fases e pesos das diferentes freqüências — os coeficientes das senóides e/ou cossenóides — são tudo o que necessitamos para recompor o sinal original. Assim, esta série de coeficientes é uma outra representação possível para o sinal, agora dita no domínio da freqüência.

Outras decomposições também são possíveis; basicamente, utilizam-se outros conjuntos de funções-base. Dependendo do objetivo a que nos propomos — análise versus armazenamento do sinal, por exemplo — um ou outro domínio será mais vantajoso. Assim, cada

aplicação dará origem a um conjunto distinto de critérios para definir o que é uma "boa" representação.

1.1.3 Compressão

De um modo geral, os fenômenos físicos contêm uma quantidade praticamente infinita de informação. Mesmo após a digitalização, as representações iniciais geralmente consumirão muito espaço, seja numa memória digital ou no espectro de um sistema de transmissão. Os métodos de *compressão* nos permitem reduzir tais necessidades a valores mais praticáveis.

Ao se comprimir um conjunto de dados, basicamente estamos utilizando uma codificação diferente para estes dados. Para a utilização normal dos dados, precisamos recuperar a codificação original. Para muitas aplicações — compressão de arquivos genéricos, por exemplo — é absolutamente necessário que o processo de compressão seja totalmente reversível, isto é, deve ser possível recuperar *exatamente* o conjunto original. Isto é o que se chama *compressão sem perdas*, e é amplamente utilizada no nosso dia-a-dia. Diversos métodos de compressão sem perdas atuais são baseados nos algoritmos da família Lempel-Ziv. Tais métodos podem obter taxas de compressão da ordem de 3:1 para arquivos de texto ou programas executáveis, por exemplo. Obviamente, este tipo de informação só é passível de compressão sem perdas; não faria sentido recuperar "aproximadamente" um programa executável, por exemplo.

Sinais como sons ou imagens, no entanto, podem sofrer pequenas alterações sem comprometer sua utilidade. Isto abre as portas para todo uma nova gama de métodos de compressão *com* perdas, onde se permite a existência de distorção no sinal recuperado. Em compensação, tais métodos fornecem taxas de compressão muito maiores que os métodos sem perdas, o que é bastante conveniente.

A compressão com perdas normalmente é obtida através da eliminação de alguns "detalhes" do sinal. Claro que a qualidade do sinal recuperado depende da quantidade de

detalhes que foram abandonados, e ainda mais do tipo destes detalhes. Suponhamos, por exemplo, uma imagem em tons de cinza cujos pixels (cada um dos pontos resultantes da digitalização) são representados com 8 bits. Cada pixel, portanto, pode ter 256 níveis distintos de brilho. Uma primeira tentativa de compressão poderia ser eliminar 4 destes 8 bits, obtendo-se assim uma imagem com 16 níveis de cinza e uma taxa de compressão de 2:1. Infelizmente, o resultado deixa muito a desejar, em termos de qualidade de imagem, porque a visão humana é bastante sensível a este tipo de informação. Outra possibilidade seria eliminar completamente 1 de cada 2 pixels, por exemplo, e recuperá-los depois por interpolação. A taxa de compressão continua sendo 2:1, e embora o efeito final seja perceptível, a distorção assim obtida não é subjetivamente tão agressiva à visão humana. Estes artefatos[1] podem ser percebidos comparando uma imagem original (Fig. A.1) com as respectivas imagens processadas (Fig. A.2).

Assim, selecionando diferentes formas de representação para uma imagem — diferentes domínios — podemos escolher que tipo de informação abandonar, dando origem a diferentes tipos de artefatos. Desta maneira, podemos usar o domínio que ofereça os melhores resultados de acordo com o tipo de aplicação que se deseja, e os respectivos critérios de qualidade. No caso de compressão de imagens, o objetivo costuma ser obter a máxima redução possível na representação final, mas dentro de níveis de qualidade aceitáveis. Os métodos que têm se mostrado mais apropriados para estes fins são baseados em representações no domínio da freqüência.

1.1.4 Transformadas

Diferentes decomposições de um mesmo sinal podem ser obtidas umas das outras através de *transformadas* — basicamente, álgebra matricial.

[1] Defeitos na imagem causados por uma codificação não exata — uma compressão com perdas, por exemplo.

Em geral, a representação no domínio do tempo é vista como a "original", já que é a mais simples de se obter a partir do fenômeno físico correspondente. Além disso, normalmente é mais fácil reproduzir o sinal físico a partir desta mesma representação. A partir dela podemos obter as outras decomposições, e destas recuperar a original. Ou seja, estas transformações são inversíveis.

Transformadas inversíveis são comumente utilizadas em processamento de imagens. A transformada de uma imagem, neste contexto, pode ser vista como a decomposição desta imagem em uma soma ponderada de imagens elementares. Matematicamente, isto é obtido multiplicando o vetor original por uma matriz adequada [18]. Os elementos do vetor resultante — a transformada do original — têm agora um significado totalmente distinto, não diretamente correlacionado ao valor de qualquer um dos pixels originais; ao contrário, cada um dos elementos resultantes é função de vários (ou todos) os pixels originais. Conseqüentemente, qualquer alteração em um elemento da transformada resultará em uma distorção espalhada por vários pixels da imagem recuperada, quando da aplicação da transformada inversa.

Aí é onde normalmente se obtém a compressão na maioria dos métodos modernos: na quantização cuidadosa dos diversos coeficientes da imagem transformada. Os coeficientes cuja alteração produzem distorções visíveis na imagem recuperada são quantizados mais finamente, enquanto que os outros podem ser quantizados com menos bits, ou simplesmente abandonados. Isto tudo depende obviamente de qual transformada se está utilizando, bem como das características do sistema visual humano [20, 27].

Claro está que certas transformadas se mostrarão mais apropriadas do que outras para este fim. O padrão JPEG [30] para compressão de imagens estáticas utiliza a Transformada Cosseno Discreta (DCT), e assim o fazem diversas variantes do padrão MPEG [32, 33, 34] para compressão de imagens dinâmicas.

Métodos mais recentes utilizam a Transformada Wavelet. Ela é usada no padrão

JPEG2000 [31], baseado no algoritmo EBCOT [12]. O padrão MPEG-4 também permite a utilização de Wavelets, de forma opcional.

1.1.5 Codificadores

Até este ponto, na discussão sobre as diferentes representações de um sinal, temos nos deparado com *números* — os coeficientes das transformadas, ou mesmo os valores do sinal digitalizado. Números, neste contexto, são apenas quantidades abstratas. Para armazenar ou transmitir esta informação, é necessário representá-la com *símbolos*. Este processo é a *codificação*.

Em última instância, todos os símbolos serão compostos de bits ou algo equivalente. Assim, ao se codificar um conjunto de dados[2], produz-se uma seqüência de bits a partir da qual poderemos depois recuperar os dados originais. Poderá haver uma diferença entre os dados recuperados e os originais, se o processo não for perfeito. Uma medida desta diferença é a *distorção*, que será definida mais adiante.

A seqüência de bits também possuirá um determinado tamanho, que naturalmente será proporcional ao tamanho do sinal original. A razão entre estes tamanhos definirá a taxa de compressão obtida. Também podemos nos referir ao número de bits gerados por unidade digitalizada do sinal original — bits por pixel, no caso de imagens; esta é a taxa de dados (bitrate).

Em geral, devemos buscar uma solução de compromisso entre taxa de dados e distorção. Um codificador pode ser projetado para permitir a especificação da taxa de dados, da distorção ou de qualquer um dos dois (não simultaneamente, é claro). A qualidade de um codificador depende basicamente dos valores taxa $\times$ distorção que se podem obter com ele, bem como da facilidade com que se pode controlar um parâmetro ou outro.

[2]Isto é, executar um algoritmo específico para escolher os símbolos que o representarão.

1.1.6 Qualidade de imagem, ou a falta dela

A avaliação mais completa da qualidade de um método de compressão é dada pela impressão subjetiva causada por imagens tratadas por este método em pessoas devidamente treinadas. *Ça va sans dire*, este tipo de avaliação é impossível de ser mecanizada, pelo menos enquanto o nosso conhecimento sobre a visão humana não tiver avançado algumas ordens de grandeza. Assim, quaisquer métodos objetivos para avaliar a qualidade de uma imagem — e portanto, do método que a gerou — ficam muito aquém deste ideal [27]. No entanto, *algum* método objetivo de avaliação tem que ser usado. Normalmente, utiliza-se alguma função da diferença numérica (pixel a pixel) entre a imagem recuperada e a original. A medida de qualidade mais comum é o MSE — *Mean Squared Error*, definido [18] como

$$\mathrm{MSE} \equiv \frac{1}{N} \sum_{i=1}^{N} (x_i - \hat{x}_i)^2$$

onde x_i é o valor de um pixel original e $\hat{x}_i$ é o valor recuperado. Ou seja, o MSE é o quadrado da distância (a menos do fator $\frac{1}{N}$) entre a imagem original e a recuperada, vistas como vetores no $\mathbb{R}^N$. Quando o MSE é zero, as imagens são iguais, e o MSE cresce à medida que uma imagem se diferencia da outra. Na verdade, isto é uma medida de distorção. Uma definição bastante simplificada de qualidade seria o inverso da distorção. Para isto há uma medida também bastante usual, o PSNR — *Peak Signal to Noise Ratio*. Para imagens com 256 níveis de cinza, ele é definido como

$$\mathrm{PSNR} \equiv 10 \log_{10} \left(\frac{255^2}{\mathrm{MSE}} \right) \text{[dB]}$$

Dada a definição simplista de distorção utilizada aqui, devemos manter presente o fato de que estas medidas têm um âmbito de uso bastante restrito. A comparação de valores de MSE ou PSNR só tem sentido quando se referem a imagens recuperadas de um mesmo

original, após processadas por métodos semelhantes, e mesmo assim o resultado deve ser analisado cuidadosamente. Assim mesmo, devido exatamente à sua simplicidade, o MSE é o parâmetro de comparação mais largamente utilizado no processamento digital de imagens.

Além da qualidade das imagens geradas, o custo computacional de um determinado método também é um fator de decisão importante. A codificação MPEG em tempo real, por exemplo, no atual estado da arte, só pode ser obtida com *hardware* dedicado.

1.1.7 Taxas de dados

A taxa de dados é o outro fator na equação de compressão. Podemos definir taxa de dados como o número médio de bits por pixel necessários para representar uma determinada imagem, o que terá implicações no tamanho da área de armazenamento e da banda de transmissão correspondentes a esta imagem. Para um dado algoritmo, maior qualidade implicará em maior taxa de dados. O conjunto de pares taxa versus distorção de um determinado algoritmo é o que se chama seu desempenho taxa $\times$ distorção.

A Teoria da Informação nos fornece um limite superior[3] para o desempenho de qualquer algoritmo de codificação, dado o comportamento estatístico da fonte que gera os dados [22]. Uma determinada representação será dita ótima se atingir tal limite.

1.2 Aproximações Sucessivas

Vários dos métodos que representam o estado da arte em compressão de imagem e vídeo utilizam quantização por aproximações sucessivas (AS), de uma forma ou de outra [7, 11, 12, 15, 31]. A AS é encontrada na literatura sob vários nomes, como *successive refinements*, *embedded coding*, ou *progressive quantization* [12, 14, 16, 17]. No entanto, todos se referem a uma mesma idéia básica. Esta vem a ser a codificação de um conjunto de

[3]Mais especificamente, um limite inferior para a taxa, dada a distorção (ou vice-versa).

dados por uma série de segmentos, que acrescentam informações cada vez mais detalhadas sobre os dados. Isto é, do primeiro segmento podemos obter uma aproximação grosseira dos dados, e adicionando a informação contida em cada segmento subseqüente, podemos obter aproximações cada vez melhores (ou seja, com menor distorção). Assim, controlando quantos segmentos são gerados, podemos controlar a taxa *ou* a distorção.

De fato, esta é uma idéia bastante simples (embora muito poderosa) e é implementada, por exemplo, na maneira como usualmente representamos números reais, ou seja, na expansão decimal. Quando desejamos expressar o valor (aproximado) de π, por exemplo, escrevemos $3,14$; se desejarmos mais precisão (e pudermos arcar com o custo correspondente) teremos então $3,141$— $3,1415$ — $3,14159$ e assim por diante. Observe que cada representação contém a anterior, apenas acrescentando dígitos.

Infelizmente, esta representação é boa apenas para utilização por pessoas; além disso, ela não é ótima (no sentido taxa $\times$ distorção). Sua variante digital — a expansão binária — resolve o primeiro problema, mas é igualmente sub-ótima. O problema pode ser apreciado ao examinarmos a melhor representação de π com 4 casas decimais: $3,1416$. O próximo passo seria $3,14159$, que não contém o anterior.

Shapiro [7] define *embedded coding* como possuindo 2 características:

- 2 representações distintas (com taxas diferentes) dos mesmos dados devem ser exatamente iguais no trecho comum (por exemplo, $3,14$ e $3,14159$). Assim, uma dada representação deve conter todas as representações de taxa mais baixa, e pode, portanto, ser truncada em qualquer ponto (entre os segmentos) e ainda manter alguma informação.

- Cada representação (isto é, em cada taxa diferente) deve obter a menor distorção possível.

A primeira característica dá a idéia principal do *embedding*, enquanto que a segunda exclui

representações que artificialmente contêm as suas versões de menor taxa. Por exemplo, a seqüência de dígitos "3 / 3.1 / 3.14 / 3.141 / 3.1416 / 3.14159" contém todas as representações ótimas com até 5 decimais, mas obviamente o desempenho deixa muito a desejar em termos de taxa.

A expansão decimal (e analogamente, a binária) pode ser facilmente adaptada e tornar-se *embedded*; basta adicionar $\frac{1}{10^{(n+1)}}$ ao valor recuperado. Os quantizadores vetoriais, no entanto, em geral não produzem uma codificação *embedded*: seqüências geradas em taxas diferentes geralmente não guardam qualquer semelhança entre si.

O *embedded coding* é especialmente interessante para transmissão progressiva ou seletiva de dados. Uma aplicação pode, por exemplo, receber uma imagem codificada[4] de um servidor, e mostrar ao usuário as diversas versões, cada vez mais detalhadas, até atingir determinado critério de qualidade. Assim o usuário tem a opção de chegar a um compromisso qualidade $\times$ tempo (ou espaço em disco), sem haver a necessidade de se armazenar diversas representações (em taxas diferentes) no servidor.

Observe, entretanto, que produzir uma codificação com alta taxa de bits em um único passo é potencialmente mais eficiente, em termos de taxa $\times$ distorção, do que fazê-lo por aproximações sucessivas.

A possível perda de eficiência que se paga pela utilização de um codificador *embedded* pode ser compensada pelo ganho em flexibilidade, já que este permite a variação dinâmica da taxa e da distorção, além da maior simplicidade dos algoritmos envolvidos.

1.3 α-expansões

Como visto na seção anterior, a expansão binária (ou decimal) — que é um tipo de quantização escalar — é conveniente para a manipulação dos dados por computadores (ou pessoas),

[4]Ou seja, uma determinada seqüência de símbolos; não se trata de criptografia.

mas tem problemas de eficiência; quantizadores vetoriais, por outro lado, apesar de mais eficientes, têm outros problemas. Há aplicações para as quais nenhum destes métodos é ideal. Para estas aplicações, desejaríamos ter:

- Taxa variável;
- Menor distorção possível para uma dada taxa;
- Simplicidade computacional.

Tendo em vista o bom desempenho da quantização vetorial no que se refere a taxa $\times$ distorção, nos damos conta de que pode ser conveniente agrupar coeficientes em vetores ao invés de codificá-los separadamente. Assim, examinaremos um método direcionado para a codificação de vetores, denominado α-expansão.

1.3.1 Definição

Seja x um vetor no $\mathbb{R}^N$, e seja $V = \{v_k \in \mathbb{R}^N, k \in \{1, 2 \ldots, M\}\}$, onde x é o vetor a ser codificado e V é o dicionário ou *codebook*.

Uma α-expansão do vetor x sobre o dicionário V é definida por:

$$x = \sum_{i=0}^{\infty} \alpha^i v_{k_i} \tag{1.1}$$

onde $0 < \alpha < 1$ e $k_i \in \{1, 2, \ldots, M\}$.

Definida a α-expansão, podemos codificar o vetor x pela seqüência de inteiros $(k_0, k_1, \ldots)$ resultante dela. Geometricamente, a relação expressa pela Eq. 1.1 pode ser visualizada como sugerido na Fig. 1.1.

Como se pode ver na figura, dados apenas os primeiros L elementos $(k_0, k_1, \ldots, k_{L-1})$

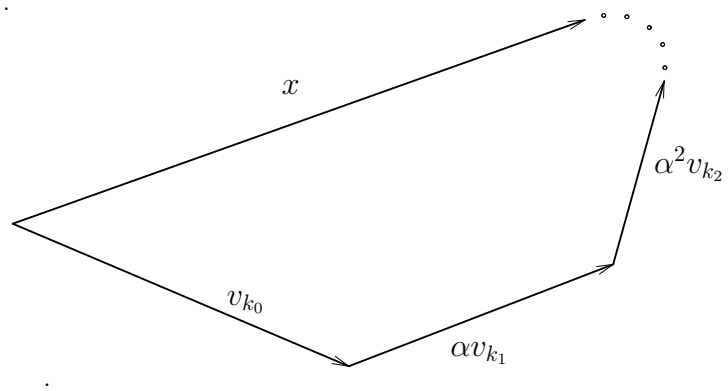


Figura 1.1: Primeiras aproximações

da seqüência, podemos calcular uma aproximação x_L para o vetor x:

$$x_L = \sum_{i=0}^{L-1} \alpha^i v_{k_i} \tag{1.2}$$

Para medir o erro resultante desta aproximação, definimos o resíduo

$$r_L = x - x_L$$

Para que esta representação truncada seja útil, é necessário que $|r_L|$ possua um limite superior que seja monotonicamente decrescente, do modo que r_L tenda a zero quando L cresce. Para ser comparável aos quantizadores vetoriais usuais, este limite superior deve ser da forma $C\beta^L$, onde C é uma constante positiva e $0 < \beta < 1$. Observe que isto apenas significa que conforme avançamos com a codificação, o *erro máximo* decresce exponencialmente, enquanto que $|r_L|$ propriamente dito pode variar de forma diferente (desde que dentro deste limite).

No caso em questão, $|r_L|$ é claramente limitado por $C\alpha^L$, para todos os x que *podem ser*

representados como uma α-expansão. Isto é facilmente verificado ao escrevermos

$$\begin{aligned} r_L &= \sum_{i=L}^{\infty} \alpha^i v_{k_i} \Rightarrow \\ |r_L| &\leq \sum_{i=L}^{\infty} \alpha^i |v_{k_i}| \\ &\leq \sum_{i=L}^{\infty} \alpha^i |v_k|_{\max} \\ &= |v_k|_{\max} \frac{1}{1-\alpha} \alpha^L \end{aligned}$$

A codificação de um vetor em uma α-expansão satisfaz ao primeiro requisito (Seção 1.2) que caracteriza um codificador *embedded*: qualquer seqüência $\{k_0, k_1, \ldots, k_{L-1}\}$ correspondente a uma determinada taxa contém todas as subseqüências correspondentes às taxas menores. O segundo requisito também pode ser atingido, dependendo do algoritmo escolhido para gerar a seqüência.

1.4 Escopo do trabalho

A Eq. 1.1 — a definição das α-expansões — é o tema central deste trabalho. Ela representa matematicamente um conceito bastante poderoso, numa forma elegantemente simples. Nós examinaremos as condições sob as quais tal expansão é possível para todos os vetores de um dado conjunto. Com este objetivo, desenvolveu-se a teoria que será apresentada no Cap. 4.

Além disso, pesquisaremos a adequação deste método para um problema real na área de processamento digital de imagens, que é a codificação de coeficientes de transformadas Wavelet de imagens naturais. Os resultados experimentais obtidos aplicando-se os algoritmos aqui descritos em vários conjuntos de dados são apresentados no final.

Resultados anteriores a este trabalho, tanto teóricos quanto experimentais, são revis-

tos brevemente no Cap. 3. Estes resultados foram desenvolvidos inicialmente na Tese de Doutorado de Eduardo A. B. da Silva [2] e outros trabalhos correlatos [6]. Tanto quanto sabemos, todos os outros resultados apresentados aqui sobre a teoria das α-expansões são originais.

Capítulo 2

Quantização por Aproximações Sucessivas

Como mencionado no capítulo anterior, as AS são utilizadas com sucesso em muitos métodos de compressão modernos. No entanto, de um ponto de vista teórico, em geral a quantização por AS não é ótima quanto ao desempenho taxa × distorção.

Neste capítulo, apresentaremos uma breve descrição do algoritmo EZW [7], que foi o primeiro a utilizar eficientemente a Quantização por Aproximações Sucessivas (QAS) de coeficientes de transformadas Wavelet em processamento de imagens. Mostraremos então porque a QAS é sub-ótima. Isto feito, apresentaremos as razões pelas quais a QAS é utilizada com sucesso em vários algoritmos, apesar da não-otimalidade.

Estes assuntos são tratados em maior detalhe no Apêndice B.

2.1 O algoritmo EZW

Quando foi apresentado [24], o algoritmo EZW — *Embedded Zerotree Wavelet* — representou uma grande inovação na área de compressão de imagens. Ele permite um controle

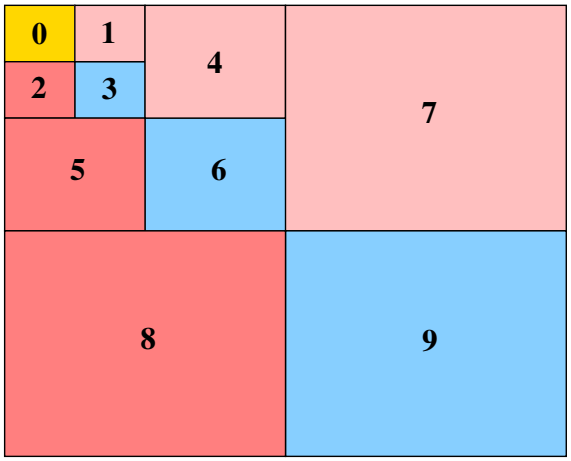


Figura 2.1: Sub-bandas da Wavelet

preciso da taxa de dados produzida, com excelente desempenho taxa × distorção em todas as taxas. Desde então diversos outros algoritmos foram introduzidos [11, 12, 15], todos baseados no EZW. O seu desempenho se deve a uma feliz combinação de transformadas Wavelet e QAS.

A transformada Wavelet[1] pode ser dividida em regiões como na Fig. 2.1. Quando aplicada a imagens naturais, as bandas correspondentes às menores freqüências (acima e à esquerda) tendem a concentrar os coeficientes de maior energia; além disso, os coeficientes de baixa energia tendem a aparecer em regiões correspondentes dentro das bandas de mesma orientação ($\{1,4,7\}$,$\{3,6,9\}$ e $\{2,5,8\}$). O principal passo do algoritmo consiste em reconhecer, dentro destes conjuntos de sub-bandas, os coeficientes com energia abaixo de um determinado limite que ocupem exatamente as mesmas regiões em relação à sub-banda em que se encontram. Todos estes coeficientes são então representados com um único símbolo. Estes grupos — as *Zerotrees* — ocorrem com relativa freqüência, dadas as características descritas acima, e proporcionam assim uma grande economia de bits na sua codificação, o que ultimamente se reflete na taxa final.

[1]Neste caso específico, uma transformada de 3 estágios.

A QAS faz com que este algoritmo e seus sucessores gerem uma codificação *embedded*. Tais algoritmos representam hoje o estado da arte em seu campo [12].

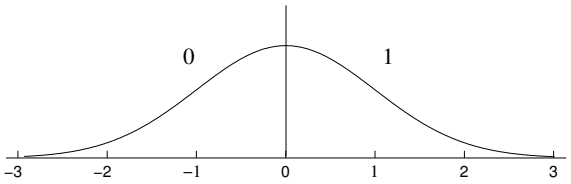
2.2 Não-otimalidade

Equitz e Cover (usando os termos "Refinamentos Sucessivos", em [14]), mostraram que "descrições ótimas nem sempre são refinamentos umas das outras"; eles estabelecem as condições sob as quais refinamentos sucessivos ótimos podem ocorrer, o que em última instância depende da distribuição estatística dos dados. A maneira mais simples de examinar o problema é ver um contra-exemplo, como a seguir.

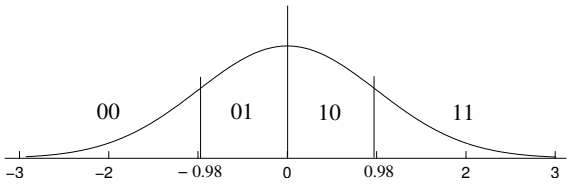
Seja uma seqüência de números aleatórios gerados por uma fonte normal $\mathsf{X} \sim \mathsf{N}(0, 1)$. A descrição ótima de 1 bit é, obviamente, a que utiliza este bit para definir o sinal do número (Fig. 2.2a). A descrição ótima de 2 bits, de acordo com as condições de Lloyd-Max [28] divide a reta em 4 regiões, sendo que cada uma está inteiramente contida em uma das 2 regiões do caso anterior (Fig. 2.2b). Podemos então utilizar o primeiro bit para selecionar uma das 2 regiões, e o segundo bit para selecionar uma sub-região dentro da primeira. Em outras palavras, temos refinamentos sucessivos. Já quando passamos para a descrição ótima de 3 bits (Fig. 2.2c), as novas sub-regiões não estão mais contidas nas anteriores, e portanto não temos mais um código *embedded*.

Pode-se perceber, então, de que forma a distribuição estatística dos dados permite ou não a otimalidade das descrições *embedded*. É importante observar que a otimalidade com respeito ao desempenho taxa $\times$ distorção depende da forma como definimos distorção; embora o MSE seja a medida mais utilizada, nada nos impede de utilizar outra métrica, como por exemplo o valor médio absoluto do erro.

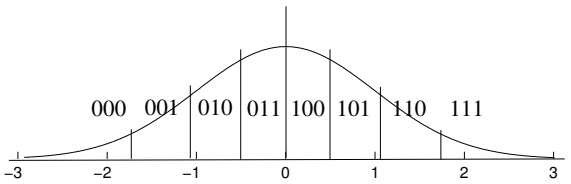
Em resumo, dada uma fonte aleatória com propriedades conhecidas, e uma função de distorção, há um limite inferior para a taxa média necessária para representar mensagens



(a) 1 bit



(b) 2 bits



(c) 3 bits

Figura 2.2: Codificação de uma variável aleatória gaussiana.

(seqüências de símbolos) geradas por esta fonte com uma determinada distorção [22, 23]. Representações ótimas são as que atingem este limite inferior, e em geral elas não são *embedded*.

2.3 A motivação para as Aproximações Sucessivas

Em que pese o resultado da seção anterior, que aparentemente contra-indica a utilização de AS, sabemos que elas são utilizadas com sucesso nos métodos mais modernos. Ocorre que não se conhece um bom modelo estocástico para "imagens naturais", de modo que devemos abdicar da otimalidade teórica, de qualquer forma. Mas as imagens naturais *têm* certas propriedades estocásticas, como uma certa dose de redundância, tanto no domínio do tempo quanto no de algumas transformadas. Os codificadores de melhor desempenho são aqueles que conseguem reconhecer esta redundância, e assim conseguem representar grandes quantidades de coeficientes com poucos símbolos.

2.3.1 Quantização em baixas taxas

Mallat e Falzon [13] mostram uma análise matemática das melhorias obtidas com codificação *embedded* de coeficientes de transformadas quando se utilizam baixas taxas de dados.

Um quantizador é dito de alta resolução quando a função densidade de probabilidade $p(x)$ da variável aleatória quantizada X é aproximadamente constante dentro de cada região do quantizador. Neste caso, sendo R a taxa de dados, o MSE varia com $C\left(2^{-2R}\right)$, com C dependendo da alocação de bits, e demonstra-se que o quantizador escalar é ótimo [28].

Os algoritmos de uso corrente, no entanto, operam na faixa de $R < 1$ bpp; assim, a hipótese da alta resolução não se aplica. Neste caso, o desempenho taxa $\times$ distorção tem um comportamento diferente.

As transformadas Wavelet ou transformadas cosseno da maior parte das imagens têm

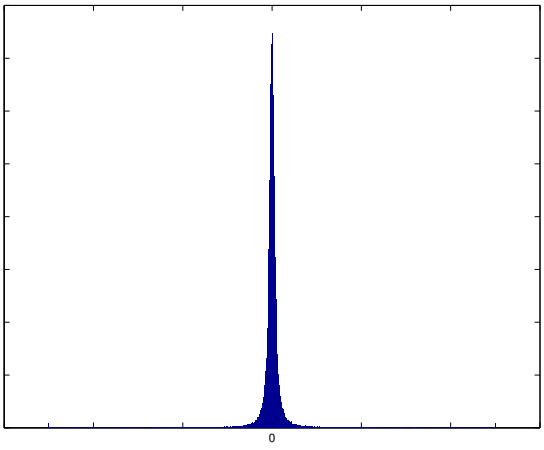


Figura 2.3: Histograma da DCT da imagem Lena

histogramas semelhantes ao da Fig. 2.3. A concentração de coeficientes próximo de zero, juntamente com a quantização em baixa resolução (a região de quantização é dividida em poucas grandes sub-regiões) faz com que uma grande parte dos coeficientes sejam quantizados como zero. Assim, é conveniente termos um tratamento especial para a representação do zero. De fato, a maior parte dos algoritmos trata o zero de forma diferente dos outros valores. Mallat & Falzon, em [13], demonstram que quando existe um comportamento conhecido da transformada quanto à localização dos coeficientes de baixa, média e alta energia, os codificadores *embedded* podem apresentar vantagens sobre os tradicionais. O resultado final é que o MSE varia aproximadamente como R^{-1}, e este é o caso quando se utiliza a transformada Wavelet, o que explica o bom desempenho de codificadores como o EZW e seus semelhantes [7, 11, 12, 15].

Capítulo 3

Resultados anteriores sobre α-expansões

Neste capítulo, examinaremos os resultados pré-existentes sobre a teoria das α-expansões, desenvolvidos em [1], [2] e [6].

3.1 O algoritmo

Em [1] apresentaram-se os resultados obtidos com a codificação de coeficientes de Wavelets via α-expansões. O método utilizado foi agrupar os coeficientes em vetores N-dimensionais e então determinar as k-seqüências para representá-los. Este método era restrito a dicionários de vetores unitários.

Um algoritmo *voraz* foi utilizado para encontrar a k-seqüência correspondente a um vetor x, como se segue:

1. Seja $r = x$.

2. Determine v_k, o vetor do dicionário mais próximo de x, ou seja, o que minimiza $|x - v_i|$. Como os vetores do dicionário são unitários, isto equivale a encontrar o

vetor que minimiza $\text{ang}(r, v_j)$, o ângulo entre r e v_j.

3. Calcule $(r - v_k)$ como o próximo valor de r.

4. Multiplique todos os vetores do dicionário por α.

5. Se $|r|$ for pequeno o suficiente, pare; senão, volte ao passo 2.

Dizemos que o algoritmo *converge* quando $|r|$ diminue exponencialmente, ou seja, é limitado por $C\alpha^n$ após o n-ésimo passo, onde C é uma constante positiva.

Observe que o algoritmo não garante erro mínimo *após* n passos, mas sim que o ele será mínimo *a cada passo*, fixados os passos anteriores; daí vem a classificação do algoritmo como *voraz*.

Seja x o vetor formado por uma determinada k-seqüência segundo a Eq. 1.1 na página 11. Observe que não há garantia de que o algoritmo ache esta mesma seqüência; na verdade, ele pode até mesmo achar uma seqüência para a qual $|r_L|$ não decresce exponencialmente, ou seja, ele pode *divergir*.

3.2 Condições para a convergência

Em [2] foram estudadas experimentalmente as condições para a convergência deste algoritmo, aplicado com diversos dicionários. Os resultados obtidos mostram que a convergência depende de uma determinada propriedade geométrica do dicionário, relacionada à distribuição dos vetores no espaço.

3.2.1 A distribuição espacial do dicionário e $\theta_{\max}$

Observou-se que a convergência depende, dentre outras coisas, de uma "boa" distribuição espacial dos vetores do dicionário, evitando a formação de quaisquer agrupamentos de

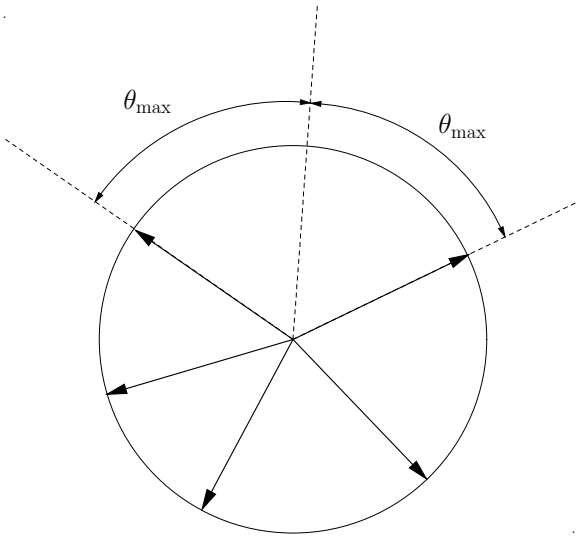


Figura 3.1: $\theta_{\max}$ para um dicionário 2-D

vetores. Mais objetivamente, a qualidade da distribuição pode ser medida pelo ângulo máximo entre um vetor qualquer do espaço e o vetor do dicionário mais próximo a ele. Chamamos este ângulo $\theta_{\max}$:

$$\theta_{\max} = \max_{x} \left\{ \min_{k} \{ \text{ang}\,(x, v_k) \} \right\}$$

Como exemplo, a Fig. 3.1 ilustra a situação com um dicionário bidimensional.

Nós desejamos que o dicionário seja capaz de gerar a origem (entre outros pontos) através de uma α-expansão, isto é, que o zero seja expressável como uma combinação convexa dos vetores do dicionário. Para isto, é necessário[1] termos $\theta_{\max} < \frac{\pi}{2}$. Neste capítulo, estamos assumindo tal tipo de dicionário.

3.2.2 α e $\theta_{\max}$

Os seguintes resultados foram obtidos experimentalmente em [2]:

[1] Mas não suficiente.

- Para dicionários com $\theta_{max} < 82°$, a convergência sempre é obtida, desde que um valor de α conveniente seja utilizado.
- Para qualquer dicionário, a convergência só é obtida com $\alpha \geq \frac{1}{2}$.
- Em geral, convergência com valores menores de α requer valores menores de θ_{max}.

O valor $82°$, como ficará claro mais adiante, deve-se a características dos algoritmos utilizados no experimento, não sendo uma limitação real das α-expansões.

3.2.3 Conseqüências

Os resultados acima apontam para uma característica dos "bons" dicionários, ou seja, um pequeno valor de θ_{max}, porque assim podemos ter também um pequeno valor de α e conseqüentemente pequenos resíduos. Portanto, dicionários com um determinado número de vetores podem ser otimizados para atingir um θ_{max} mínimo. Isto pode ser obtido através de um espalhamento mais "regular"[2] dos vetores no espaço. Além disso, em geral, dicionários com mais vetores terão θ_{max} menor.

Estes resultados são, de certa forma, esperados. Quanto mais vetores (no dicionário) tivermos à escolha, em cada passo, para aproximar um determinado vetor de coeficientes, menor será o resíduo naquele passo. Se os vetores do dicionário estiverem mais concentrados em determinada região do espaço, haverá outras regiões que não serão bem aproximadas por nenhum vetor.

A característica de espalhamento regular (embora não necessariamente na mesma acepção) aparece também na solução de outro problema, o de empacotamento de esferas [8]. Basicamente, este problema consiste em como arrumar o maior número possível de esferas em uma região do espaço, e tem soluções ótimas — ou pelo menos ótimas entre as

[2]Na verdade, a regularidade deve ser definida justamente como a propriedade que minimiza o θ_{max}, mas o resultado final fica bem próximo ao uso quotidiano da palavra.

conhecidas — em espaços de várias dimensões. As posições dos centros das esferas nestas soluções geram redes ("lattices") regulares. Em geral, estas redes geram dicionários bem distribuídos e com propriedades estruturais conhecidas.

Observe, no entanto, que de uma forma geral a convergência com valores menores de α exige dicionários com maior número de vetores, e conseqüentemente uma codificação com uma taxa de dados mais alta. Não há como fugir da solução de compromisso entre taxa e distorção.

3.3 $\alpha_{\min} \times \theta_{\max}$

Estes resultados nos impelem em busca de uma noção mais rigorosa da relação entre α e $\theta_{\max}$. Mais especificamente, observou-se que para cada dicionário existe um valor $\alpha_{\min}$ que garante a convergência para todo $\alpha \geq \alpha_{\min}$ (e, inversamente, garante a não-convergência para qualquer $\alpha < \alpha_{\min}$). Em [6], o seguinte teorema forneceu a relação que se buscava:

Teorema: O algoritmo descrito na Seção 3.1 sempre converge para pontos x tais que $|x| \leq \beta$, desde que

$$\begin{cases} \alpha \geq \frac{1}{2\cos\theta_{\max}} \\ \beta = 2\cos\theta_{\max} \end{cases} , \text{ se } \theta_{\max} \leq \frac{\pi}{4}$$

$$\begin{cases} \alpha \geq \sin\theta_{\max} \\ \beta = \frac{1}{\cos\theta_{\max}} \end{cases} , \text{ se } \theta_{\max} \geq \frac{\pi}{4}$$

Note que as condições especificadas aqui são suficientes, mas não necessárias. Este valor, portanto, é apenas um limite superior para $\alpha_{\min}$. O $\alpha_{\min}$ real de um determinado *codebook* pode ser menor. A demonstração do teorema encontra-se no Apêndice C.3.1.

Este teorema não resolve algumas questões:

1. Ele é baseado em um determinado algoritmo para a escolha de k. Este algoritmo não é necessariamente o melhor em termos de minimizar α.

2. O teorema assume o pior caso (vide demonstração) em cada iteração do algoritmo, ou seja, que todos os resíduos encontrados estarão a um ângulo $\theta_{\max}$ do vetor mais próximo do dicionário. Ele não implica que tal caso seja sequer possível. Na prática, obtém-se convergência para valores menores de α do que o limite superior indicado pelo teorema.

3. O limite superior obtido aqui não é necessariamente o menor limite superior possível.

4. O teorema garante a convergência para todos os pontos dentro da esfera de raio β. No entanto, ele não garante que a convergência seja obtida *apenas* para estes pontos. O teorema também não define a região em que *não há* convergência.

Apesar destes detalhes, o teorema nos dá uma valiosa visão do problema da convergência:

1. Ele mostra que a região de convergência pode conter um conjunto aberto. Isto é importante, já que garante que para qualquer conjunto de dados, haverá um α capaz de ser utilizado para codificá-los.

2. O teorema mostra o comportamento genérico da região de convergência: para valores maiores de α, a região de convergência cresce.

3. Finalmente, ele confirma um sentimento intuitivo: pode-se obter a convergência mais rapidamente, ou seja, pode-se usar um α menor, quando se usa um dicionário com mais vetores e/ou vetores mais bem distribuídos no espaço.

Capítulo 4

Teoria das α-expansões

Podemos agora avançar para uma teoria mais geral das α-expansões. O objetivo, neste ponto, é obter um método genérico, capaz de codificar vetores contidos em um determinado sub-conjunto do $\mathbb{R}^N$, e então estudar as propriedades deste método quanto a convergência e desempenho.

4.1 Elementos representáveis pela α-expansão

O método proposto para codificar um vetor consiste simplesmente em determinar a seqüência de coeficientes k_i correspondente à α-expansão daquele vetor sobre o dicionário V. A questão é determinar um dicionário e um valor de α que permitam representar todos os vetores contidos no conjunto de dados a ser codificado.

Qualquer método de codificação restringe os objetos a serem codificados, definindo um conjunto sobre o qual o método deve ser válido. Neste caso, vamos definir este conjunto como sendo $B = \left\{x \in \mathbb{R}^N | |x| \leq 1\right\}$. Não há perda de generalidade; qualquer outro sub-conjunto limitado de $\mathbb{R}^N$ pode ser normalizado de modo a estar contido em B.

Na Seção 3.1, o método era restrito a dicionários unitários; aqui, esta restrição não será

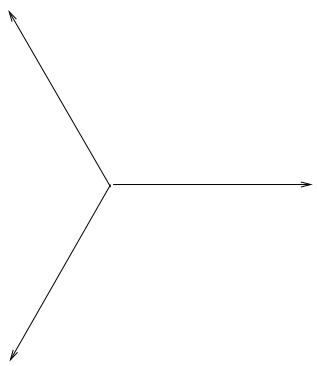


Figura 4.1: Dicionário 2-D

imposta.

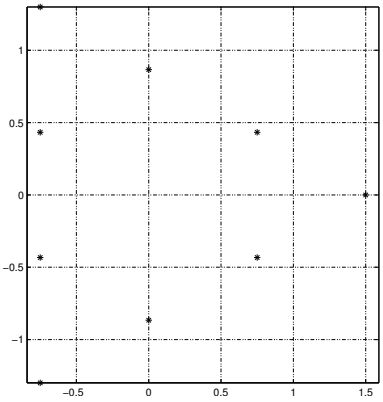
O problema, agora, é garantir que a α-expansão seja capaz de representar todos os elementos do conjunto B. Ou seja, devemos determinar quais dicionários V e quais valores de α permitem que se encontre uma α-expansão para todos os elementos de B.

No entanto, ao invés de buscar diretamente a condição de convergência, é interessante examinar o problema por um outro ponto de vista: vamos determinar o conjunto X de todos os pontos que *são* representáveis por uma α-expansão, e então verificaremos quando este conjunto contém B. Logicamente, X é dado pelo conjunto de todas as α-expansões possíveis.

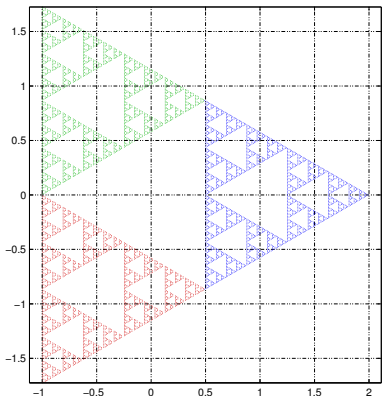
Consideremos um exemplo com um dicionário 2-D. Seja V o dicionário formado pelos vetores ilustrados na Fig. 4.1; os vetores são unitários e eqüidistantes. Seja $\alpha = 0.5$. Consideremos os conjuntos formados pelos possíveis valores de x_L, conforme definido na Eq. 1.2, para diversos valores de L.

Os valores possíveis para x_1 são os próprios elementos do dicionário:

$$x_1 = \begin{cases} v_1 \\ v_2 \\ v_3 \end{cases}$$



(a) x_2



(b) x_L

Figura 4.2: Possíveis valores de x_L

Na próxima iteração, temos 9 valores possíveis para x_2,

$$x_2 = \begin{Bmatrix} v_1 \\ v_2 \\ v_3 \end{Bmatrix} + \begin{Bmatrix} \alpha v_1 \\ \alpha v_2 \\ \alpha v_3 \end{Bmatrix}$$

os quais podem ser vistos na Fig. 4.2a. À medida que L cresce, o conjunto dos possíveis valores de x_L passa a tomar a forma ilustrada na Fig. 4.2b, que possui uma estrutura típica de um fractal[1]. Experimentando com diferentes dicionários e valores de α, obtemos resultados semelhantes, isto é, o conjunto x_L parece aproximar uma estrutura fractal quando $L \to \infty$. Se pudermos provar que isto realmente sempre acontece, poderemos utilizar os resultados da teoria dos fractais para verificar sob quais condições este conjunto conterá a bola unitária B.

[1]Na verdade, neste caso o conjunto dos possíveis x_L aproxima o *triângulo de Sierpinski*, um fractal bastante conhecido.

4.2 Fractais e IFS

A conjectura mencionada acima pode ser provada facilmente. Para isto, recorremos ao conceito de *Iterated Function Systems* — IFS.

Sejam x, k, α e V como definidos na Seção1.3.1. Seja o conjunto de funções

$$f_k(x) \;=\; \alpha x + v_k \tag{4.1}$$

e seja

$$F(A) = \bigcup_{1}^{M} f_k(A) \tag{4.2}$$

onde A é um sub-conjunto qualquer do $\mathbb{R}^N$ e $f_k(A) = \{f_k(a), a \in A\}$. Observe que F é definida no conjunto dos sub-conjuntos de $\mathbb{R}^N$, e f_k tem sua definição estendida de modo a aplicar-se também a elementos deste conjunto.

Da teoria dos fractais [3], segue que pelo fato de cada f_k ser um *mapeamento contrativo* (especificamente para $0 < |\alpha| < 1$), temos que o conjunto dos f_k forma um IFS. Uma das propriedades do IFS é que

$$\exists| \; \Lambda \mid F(\Lambda) = \Lambda \tag{4.3}$$

ou seja, a função F associada ao IFS sempre possuirá um único ponto fixo. Este "ponto" Λ — na verdade, um sub-conjunto do $\mathbb{R}^N$ — é chamado o *atrator* do IFS.

De posse destes conceitos, podemos provar o seguinte:

Teorema: O conjunto X de todos os pontos x que podem ser representados como na Eq. 1.1 é o atrator do IFS correspondente (isto é, aquele formado pelas funções definidas na Eq. 4.1).

A demonstração pode ser encontrada na pág. 102.

Os atratores de um IFS são, geralmente, fractais; eles também podem degenerar em

conjuntos "normais", isto é, não fractais. Resta-nos agora determinar em que condições o atrator conterá a bola unitária B.

4.3 Determinação de Λ

Nesta seção utilizaremos algumas definições de conceitos geométricos e resultados associados a elas, como resumido a seguir.

4.3.1 Resumo da geometria utilizada

Combinação: Sejam $a_1, \ldots, a_k \in \mathbb{R}$, e $p_1, \ldots, p_k \in \mathbb{R}^N$. A combinação linear $x = \sum_{i=1}^{k} a_i p_i$ é dita:

- Uma *combinação positiva*, se e somente se $a_i \geq 0, \forall i \in \{1, \ldots, k\}$;
- Uma *combinação afim*, se e somente se $\sum_{i=1}^{k} a_i = 1$;
- Uma *combinação convexa*, se e somente se ambas as condições anteriores valerem.

Conjunto convexo: Um conjunto S é dito convexo se e somente se para todo par de pontos $x, y \in S$, toda combinação convexa de x e y também pertencer a S (em termos mais intuitivos, se o segmento $\overline{xy}$ também estiver contido em S).

Fechamento convexo: O fechamento convexo de um conjunto S é o menor conjunto convexo que contém S.

Politopo: Um politopo é o fechamento convexo de um conjunto finito. Todo politopo é o fechamento convexo de um determinado sub-conjunto de seus pontos. Os pontos deste sub-conjunto são chamados *vértices* do politopo. Polígonos e poliedros são exemplos de politopos em 2 e 3 dimensões, respectivamente. Observe que um conjunto

infinito *pode* ter um fechamento convexo que *não seja* um politopo; por exemplo, uma esfera.

Obs: Uma definição mais genérica de poliedro inclui conjuntos ilimitados formados por interseções convexas de semi-espaços (por exemplo, um cone de seção poligonal), e politopos seriam poliedros limitados (em qualquer número de dimensões). No entanto, para nossos propósitos aqui, estamos considerando apenas conjuntos limitados (isto é, contidos em alguma esfera de raio finito).

Vértice: Os vértices de um politopo também podem ser definidos como aqueles pontos que não podem ser expressos como combinações convexas de quaisquer outros pontos do politopo. Um politopo também é o conjunto de todas as combinações convexas de seus vértices. Observe que nem todos os pontos de S serão necessariamente vértices do politopo associado (alguns podem resultar interiores ou na borda do politopo).

Homotetia: Uma homotetia de centro C e fator de escala α é a transformação

$$\begin{aligned} h(x) &= C + \alpha(x - C) \\ &= \alpha x + (1 - \alpha)C \end{aligned}$$

Também podemos ter $h(S) = \{h(s), s \in S\}$, isto é, estender S de modo a aplicá-la a um conjunto; assim, podemos falar que o conjunto $h(S)$ é uma homotetia de S.

Homotetia de um conjunto convexo: Seja S um conjunto convexo e sejam $\alpha \in [0, 1]$, $C \in S$ e $T = \alpha S + (1 - \alpha)C$. Então $T \subseteq S$. Da mesma forma, se $\alpha > 1$, temos que $T \supset S$. Em outras palavras, a homotetia $h(S)$ de um conjunto S, com centro $C \in S$ e fator de escala $\alpha < 1$, está contida em S, como ilustrado na Fig. 4.3. Observe que todas as condições dadas são necessárias. Por exemplo, a homotetia de um conjunto não convexo pode não estar contida neste conjunto.

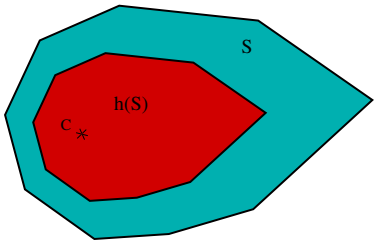


Figura 4.3: Homotetia

4.3.2 O politopo do dicionário V

Seja P o politopo definido pelo fechamento convexo do dicionário $V = \{v_k\}$. Da mesma forma, seja P_α o fechamento convexo do conjunto $W = \left\{w_k = \frac{1}{1-\alpha} v_k\right\}$. Observe que P_α e W podem ser obtidos de P e V, respectivamente, por uma homotetia de centro na origem e fator de escala $\frac{1}{1-\alpha}$.

Todo $x \in \Lambda$ é, por construção, uma combinação positiva de vetores do dicionário V, com coeficientes cuja soma é $\frac{1}{1-\alpha}$, e portanto uma combinação convexa dos vetores w_k. Assim, temos que

$$\Lambda \subseteq P_\alpha \tag{4.4}$$

Podemos reescrever $f_k(x)$ como

$$\begin{aligned} f_k(x) &= \alpha\left(x - \frac{1}{1-\alpha} v_k\right) + \frac{1}{1-\alpha} v_k \\ &= \alpha\,(x - w_k) + w_k \end{aligned}$$

Ou seja, cada f_k é uma homotetia de centro w_k e fator de escala α, o que implica em $f_k(P_\alpha) \subseteq P_\alpha$, já que (i) $0 < \alpha < 1$ e (ii) P_α é convexo. Assim, $F(P_\alpha)$ é a união de várias cópias reduzidas (homotetias) de P_α, cada uma delas estritamente contida em P_α. Portanto,

podemos ter duas situações mutuamente exclusivas:

$$F(P_\alpha) = P_\alpha \Leftrightarrow \Lambda = P_\alpha$$

ou

$$F(P_\alpha) \subset P_\alpha \Leftrightarrow \Lambda \subset P_\alpha$$

(o símbolo $\subset$ aqui é usado como "estritamente contido", isto é, "contido mas diferente de"). No primeiro caso, todos os pontos de P_α (*e apenas estes pontos*) podem ser representados como uma α-expansão; no segundo, apenas um sub-conjunto de P_α pode ser representado. Precisamos agora saber em que condições teremos $\Lambda = P_\alpha$.

4.3.3 Limites de α para obter $\Lambda = P_\alpha$

Como será mostrado a seguir, para cada dicionário V existe um valor crítico α_c tal que $\Lambda_\alpha = P_\alpha \Leftrightarrow \alpha \geq \alpha_c$. Também serão determinados os limites inferior e superior de α_c em função de N e M — a dimensão e a cardinalidade do dicionário, respectivamente.

Estes fatos serão ilustrados aqui com uma argumentação "geométrica". Demonstrações formais podem ser encontradas na Seção D.5.

A situação é ilustrada na Fig. 4.4, onde $Q = P_\alpha$, R é homotetia de P com centro v_k e razão α, e S é homotetia de Q com centro w_k e razão α. Segue que S também será homotetia de R com centro na origem e razão $\frac{1}{1-\alpha}$. Da figura, podemos ver que enquanto α cresce no intervalo $(0, 1)$, S cresce em relação a Q da mesma forma que R cresce em relação a P. As homotetias preservam as relações de continência entre pontos e conjuntos correspondentes nos vários polígonos. Assim, verificamos que quando α cresce, S ocupa uma proporção cada vez maior de Q, e portanto, se para um determinado valor α_1 a união dos vários $f_k(P_{\alpha_1})$ cobre o próprio P_{α_1} (isto é, $F(P_{\alpha_1}) = P_{\alpha_1}$), então isto também acontecerá para qualquer outro $\alpha_2 \geq \alpha_1$. Logo, se tomarmos α_c como o menor destes valores tais que

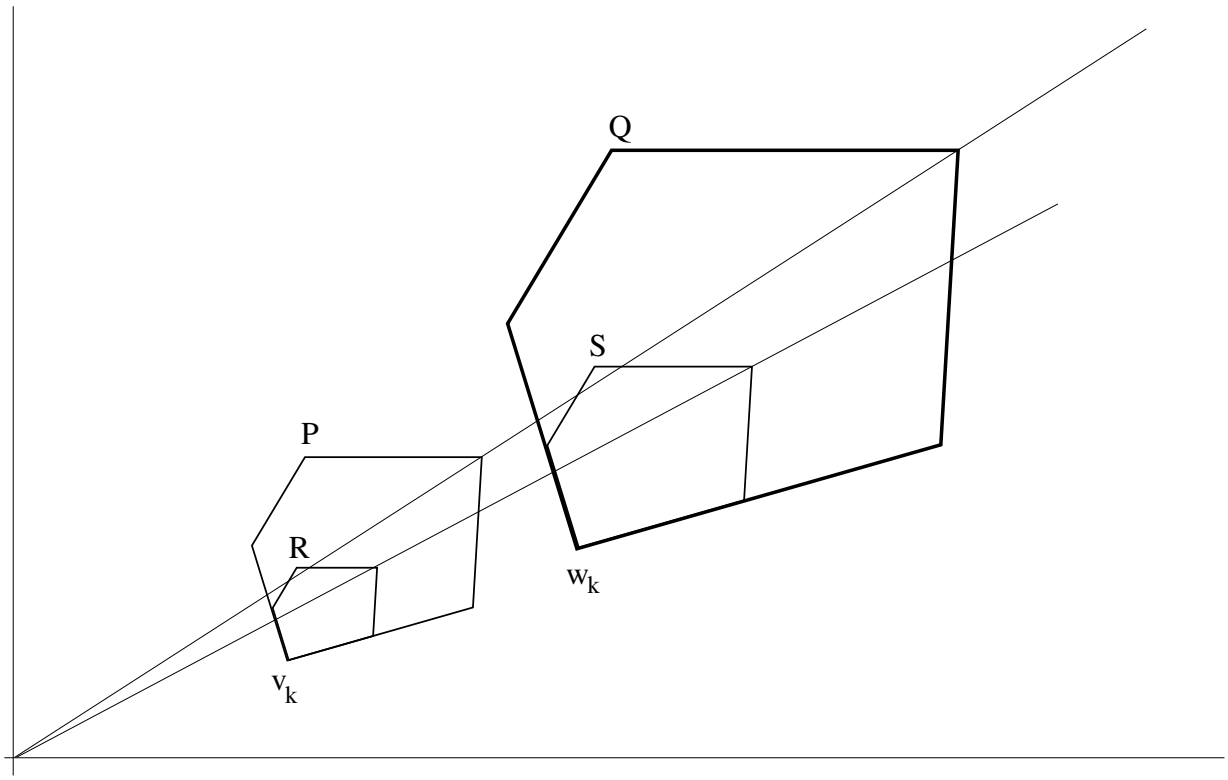


Figura 4.4: Homotetias de P

$F(P_\alpha) = P_\alpha$, temos que $\Lambda_\alpha = P_\alpha \Leftrightarrow \alpha \geq \alpha_c$.

Pode-se mostrar (Seção D.5) que $M^{(-\frac{1}{N})} \leq \alpha_c \leq \frac{N}{N+1}$, para qualquer dicionário composto de M vetores no $\mathbb{R}^N$. A única restrição imposta ao dicionário é que ele seja suficiente para gerar (via combinações positivas) qualquer ponto do $\mathbb{R}^N$. Para isto, é necessário que o politopo P contenha a origem em seu interior (isto é, não em sua fronteira ou "superfície"). O α_c de um determinado dicionário pode variar (dentro dos limites acima) de acordo com a distribuição geométrica de seus vetores no espaço, isto é, sua *forma*.

Assim, concluímos que é possível obter atratores (Λ) que contenham um conjunto aberto (P_α). Assim, para que o método de codificação possa ser usado para qualquer vetor de B, basta que o dicionário contenha a origem e seja escalado de forma que $P_\alpha \supset B$, para o valor de α escolhido. Obviamente, devemos ter $\alpha \geq \alpha_c$.

4.4 Determinação dos coeficientes

Vimos que cada f_k mapeia P_α em uma cópia reduzida. Assim, podemos escrever

$$\forall k, \forall x \in f_k(P_\alpha) \Rightarrow \exists y \in P_\alpha \,|\, x = f_k(y)$$

ou seja, para cada k, e para todo x pertencente a $f_k(P_\alpha)$, existirá um y correspondente (via homotetia) em P_α. Este y será

$$y = g_k(x) = \frac{x - v_k}{\alpha}$$

Desta forma, todo x de $f_k(P_\alpha)$ pode ser escrito como $x = v_{k_0} + \alpha y$. Além disso, todo ponto de P_α estará contido em ao menos uma cópia $f_k(P_\alpha)$ (supondo que $\Lambda = P_\alpha$). Assim, todo

y obtido acima também pode ser expandido, de modo que a expansão será infinita:

$$x = v_{k_0} + \alpha y_0 = v_{k_0} + \alpha(v_{k_1} + y_1) = \ldots$$

Chegamos assim às regras para determinar uma α-expansão (talvez uma dentre várias expansões possíveis):

1. Determine k tal que $x \in f_k(P_\alpha)$. Pode existir mais de um valor de k que satisfaça esta condição; basta selecionar um qualquer.

2. Determine $y = g_k(x)$. O fato de que $\Lambda = P_\alpha$ garante que $y \in P_\alpha$.

3. Tome este y como o próximo valor de x, e volte ao passo 1.

A aplicação repetida destes passos gerará a seqüência desejada $(k_0, k_1, \ldots)$. Geometricamente, cada k especifica que x está dentro de uma determinada sub-região dentro de P_α, e assim limita os possíveis valores de x. Cada sub-região tem a mesma *forma* da região inicial, e portanto pode ser subdividida de novo da mesma maneira, *ad infinitum*. A cada passo, os valores possíveis de x são mais restritos. A seqüência dos valores k nos diz, basicamente, qual é a sub-região onde estava o x original. Uma conclusão importante deste raciocínio é que

> Para qualquer dicionário, α_c depende apenas de sua *forma* geométrica, e não de seu tamanho, posição ou orientação com relação à origem.

Entendemos por *forma* a posição relativa dos elementos do dicionário, incluindo aqueles que porventura não sejam vértices de P. Mais formalmente, isto decorre do fato que $x = f_k(y)$ é invariante com respeito a qualquer combinação de translação, homotetia, reflexão e/ou rotação — e estas operações não modificam o que entendemos como *forma* do objeto.

Capítulo 5

Algoritmos

O passo principal no capítulo anterior consiste em determinar quais dos vários $\mathsf{f}_\mathsf{k}(\mathsf{P}_\alpha)$ contêm um determinado ponto x. Embora isto possa parecer um problema trivial em 2 dimensões, ele torna-se rapidamente impraticável em dimensões maiores, e com dicionários grandes. Assim, é necessário modificar aquele algoritmo básico, de forma a se obter um método que seja realmente implementável.

5.1 O algoritmo voraz[1]

Uma possível solução pode ser utilizar o algoritmo voraz já examinado na Seção 3.1, com as devidas modificações relacionadas ao fato de que não estamos mais restritos à utilização de dicionários unitários. Neste caso, o próximo elemento da seqüência é escolhido determinando o vetor do dicionário V mais próximo de r, que é o resíduo do passo anterior (e portanto, o que melhor o aproxima).

No entanto, agora sabemos que esta escolha será "correta" se e somente se o $\mathsf{f}_\mathsf{k}(\mathsf{P}_\alpha)$ correspondente contiver o x em questão. Para que isto sempre seja verdadeiro, é necessário que $\mathsf{f}_\mathsf{k}(\mathsf{P}_\alpha)$ contenha todos os pontos de P_α que possuem ν_k como vetor do dicionário V

[1] Em inglês, "greedy".

mais próximo de si. Este conjunto de pontos forma o que se chama a *célula de Voronoi* de ν_k, denotada por V_k (não confundir com o dicionário V). Este raciocínio está detalhado na Seção F.2.1.

Dado um conjunto de pontos q_k, a célula de Voronoi de cada um deles é definida como o conjunto de pontos do $\mathbb{R}^N$ que possuem aquele q_k como seu vetor mais próximo. No nosso caso, estamos interessados apenas nos pontos das células de Voronoi *que também estejam contidos em* P_α.

Assim, as condições de convergência para o novo algoritmo passam a ser:

$$F(P_\alpha) = P_\alpha$$

e

$$\forall k,\ f_k(P_\alpha) \supseteq \left\{ V_k \bigcap P_\alpha \right\}$$

Desta maneira, selecionando o ν_k mais próximo de x implica em $x \in f_k(P_\alpha)$; isto é suficiente para o algoritmo convergir.

Capítulo 6

Expansões Binárias como α-expansões

Podemos mostrar agora que a representação usual de números em expansões binárias é um caso particular das α-expansões.

6.1 Representação binária de escalares

Tomemos como ilustração as representações binárias (indicadas pelo subscrito 2 à esquerda das mesmas) das partes fracionárias de π e e (base dos logaritmos neperianos), truncadas após o 10^{O} bit: ${}_20.0010010000$ e ${}_20.1011011111$, respectivamente (veja Seção G.1).

Como se vê, a representação binária de um número qualquer no intervalo $[0, 1]$ pode ser especificada por uma seqüência $(b_1, b_2, \ldots)$, que por sua vez representa o valor

$$x = \sum_{i=1}^{\infty} b_i \left(\frac{1}{2}\right)^i, \quad b_i \in \left(0, \frac{1}{2}\right)$$

Nesta forma, a representação é imediatamente reconhecida como uma α-expansão, com $\alpha = \frac{1}{2}$, sobre um dicionário de "vetores" unidimensionais $V = \left\{v_0 = 0, v_1 = \frac{1}{2}\right\}$. Isto faz

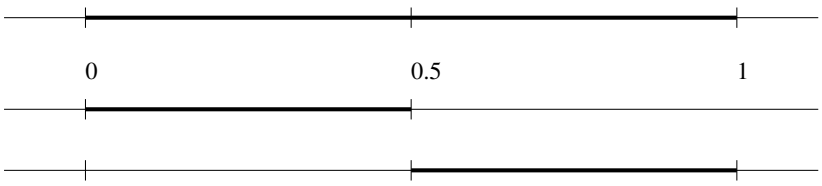


Figura 6.1: O intervalo $[0, 1]$

com que P_α seja o intervalo fechado $[0, 1]$. Como P_α é coberto por suas duas metades, temos que $\Lambda = P_\alpha$. Portanto, qualquer número neste intervalo pode ser representado por uma α-expansão sobre este dicionário.

As duas metades se interceptam em $x = 0.5$; isto significa que este ponto tem mais de uma representação possível. De fato, ${}_2 0.1000\ldots = {}_2 0.0111\ldots$ Observe também que o ponto $x = 1$, que pertence a P_α, pode ser representado como ${}_2 0.1111\ldots$

6.2 Representação binária de vetores

Analogamente ao caso escalar, qualquer vetor no hiper-cubo $[0, 1]^N$ pode ser representado como uma α-expansão sobre e o dicionário $V = \left\{[0, \frac{1}{2}]^N\right\}$ (isto é, o conjunto de vetores N-dimensionais com 2^N elementos, cujas coordenadas são 0 ou $\frac{1}{2}$). O caso bidimensional é ilustrado na Fig. 6.2.

Qualquer hiper-cubo pode ser coberto por 2^N cópias cujo tamanho (linear) é metade do original. Apesar de haver interseções não vazias entre as cópias, o hiper-volume total destas interseções é nulo. No exemplo bidimensional, a área das arestas comuns dos quatro quadrados é, obviamente, zero. Isto significa que a quantidade de pontos que admite mais de uma representação é uma proporção "pequena" do "total" — na verdade, 0%. Isto influencia positivamente o desempenho taxa-distorção deste tipo de dicionário.

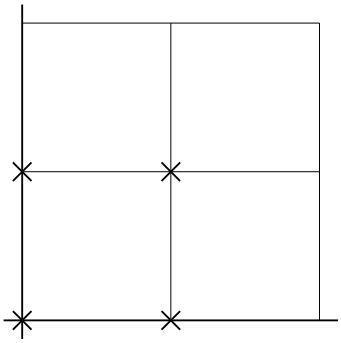


Figura 6.2: Expansão binária 2-D

6.3 Outras representações usuais

Os argumentos apresentados acima não se restringem às representações binárias; as expansões em qualquer base inteira podem ser vistas como uma α-expansão. A representação decimal, por exemplo, pode ser considerada uma α-expansão com $\alpha = \frac{1}{10}$.

6.4 Aproximações Sucessivas como generalização

Como visto acima, as representações usuais dos números não são mais do que casos particulares de α-expansões. Apesar de estes casos apresentarem vantagens visíveis (podem ser calculadas rapidamente, por exemplo), isto levanta uma questão sobre serem ótimos com respeito a outros critérios, e em todas as situações. Estamos interessados, especificamente, no desempenho taxa-distorção. Isto nos leva a estudar as características de outros dicionários, numa tentativa de encontrar um ou mais com desempenho superior ao do hiper-cubo.

Capítulo 7

Resultados experimentais

De acordo com as idéias expostas no capítulo anterior, diversos experimentos foram realizados com vistas a pesquisar a existência de dicionários com desempenho taxa-distorção superior ao do hiper-cubo, que é equivalente ao do quantizador escalar uniforme.

Será mostrado que para taxas altas, o quantizador escalar uniforme realmente possui o melhor desempenho. No entanto, na codificação de transformadas de imagens em taxas baixas, existe uma grande quantidade de coeficientes de pequena magnitude; nestes casos, a utilização de outros dicionários pode fornecer uma melhora no desempenho — desde que se utilize um método eficiente para a representação dos zeros (veja seção 2.3).

A comparação de diferentes dicionários é feita medindo-se as taxas de dados e distorções obtidas aplicando o método de codificação a diversos conjuntos de dados. Para isto, tanto os dicionários quanto os conjuntos de dados precisam de uma preparação prévia. Os conjuntos de dados, em geral, são simples conjuntos de escalares. Estes escalares são agrupados em vetores de mesma dimensão que o dicionário em questão, e então normalizados de forma que o maior vetor seja unitário. O dicionário, por sua vez, precisa ser escalado de forma que o P_α correspondente contenha a bola unitária. Desta forma, todos os vetores a serem codificados estarão contidos em P_α, o que é necessário para a convergência do algoritmo.

Os resultados de distorção são calculados em relação aos escalares, de forma a se poder comparar dicionários de diferentes dimensões.

Assim, a comparação envolve:

1. A escolha de um ou mais conjuntos de dados. Neste trabalho, selecionamos conjuntos aleatórios com diversas distribuições e também dados reais formados por coeficientes de transformadas Wavelet de imagens naturais.

2. A determinação dos parâmetros ideais para cada dicionário: α e o fator de escala.

3. A aplicação do método de codificação e posterior medição dos parâmetros de comparação.

7.1 O algoritmo

O algoritmo utilizado para a codificação é o mencionado na Seção 5.1. Este algoritmo não é restrito a dicionários unitários. Assim, devemos selecionar o próximo vetor do dicionário pela sua distância ao resíduo a ser codificado (e não mais pelo ângulo entre eles). Finalmente, para efeitos de medição de taxa de dados e distorção, os passos de determinação do k_i e escalonamento por $\frac{1}{\alpha}$ são realizados um número pré-determinado de vezes. Assim, o algoritmo pode ser descrito como se segue:

1. Tome $r = x$.

2. Repetir R vezes:

   (a) Determine v_k, o vetor do dicionário mais próximo de r, ou seja, aquele que minimiza $|r - v_k|$.

   (b) Tome $\left(\frac{r - v_k}{\alpha}\right)$ como o próximo valor de r.

O valor médio de $|r|$ nos dá a distorção total (vetorial) a cada passo; a partir deste valor e da dimensão do dicionário, podemos calcular a distorção média na representação dos escalares que formam o conjunto de dados original.

7.2 Adaptação do dicionário

De modo a se poder codificar qualquer vetor formado a partir de um conjunto de dados, utilizando α-expansões sobre um dicionário V, é necessário que:

1. $\alpha \geq \alpha_c$, onde α_c é o valor crítico para o dicionário e algoritmo em questão. O α_c para o algoritmo voraz utilizado aqui pode ser maior que aquele para o algoritmo descrito no Cap. 4. Isto porque além das restrições originais, a convergência do algoritmo voraz depende de que cada $f_k(P_\alpha)$ contenha a região de Voronoi relativa a w_k.

2. O conjunto de dados esteja contido em P_α. Isto é garantido normalizando o conjunto de dados, e escalando o dicionário de modo que P_α contenha a bola unitária. Além de garantir a convergência do algoritmo, isto também torna comparáveis os dados de distorção obtidos a partir de diferentes conjuntos de dados.

O valor α_c, como visto na Seção 4.4, depende apenas da forma do dicionário (e não da sua escala), e pode ser obtido experimentalmente.

Para garantir o item 2, no entanto, temos dois graus de liberdade. Poderíamos utilizar um α maior, de modo a se obter um P_α maior — já que ele é proporcional a $\frac{1}{1-\alpha}$ — a ponto de conter a bola unitária. No entanto, valores maiores de α produzem uma convergência mais lenta — já que a distorção média é limitada por $C\alpha^n$. Assim, no interesse de se obter o melhor desempenho possível, utilizamos $\alpha = \alpha_c$. A opção que nos resta então é escalar o próprio dicionário V, de modo que o P_α correspondente seja circunscrito à esfera unitária. Observe que desejamos o menor P_α possível; caso contrário, a distorção nos primeiros passos

será maior. Seja γ_c o menor fator de escala tal que P_{α_c} contenha a esfera unitária. Por razões práticas, este γ_c será sempre especificado em relação a um dicionário normalizado, isto é, onde os vetores com o maior módulo sejam unitários.

Com valores apropriados de α e γ, podemos garantir que $r \in P_\alpha \Rightarrow \left(\frac{r-vk}{\alpha}\right) \in P_\alpha$. Este fato pode ser utilizado para a determinação experimental de α_c e γ_c. Basicamente, isto é feito executando muitas iterações do algoritmo para um grande número de pontos, e controlando o módulo do resíduo. Se em qualquer passo se obtiver um resíduo com módulo maior do que γ, um ou ambos os parâmetros são menores do que os valores críticos. Para alguns dicionários simples, os valores críticos podem ser calculados teoricamente e comparados aos valores obtidos experimentalmente. Neste trabalho, isto foi feito para validar o método experimental.

7.3 Medições de taxa e distorção

O procedimento utilizado para se comparar os desempenhos de diversos dicionários foi calcular o MSE para as α-expansões de grandes conjuntos de dados, implementando-se o algoritmo descrito acima.

Os resultados obtidos dependem fortemente da distribuição espacial dos vetores codificados. Estes vetores são obtidos pelo simples agrupamento dos coeficientes escalares a serem codificados, numa dimensão apropriada, isto é, a dimensão do dicionário em questão. Dois tipos de conjuntos de dados foram utilizados: um contendo pontos uniformemente distribuídos no espaço dentro da N-esfera de raio 1, e outro obtido a partir de coeficientes de transformadas Wavelet de imagens naturais.

Note que cada vetor codificado representa N coeficientes do conjunto de dados inicial. De forma a se poder comparar dicionários com dimensões diferentes, devemos obter os valores de taxa de dados e distorção *por coeficiente*.

Obviamente, tanto taxa como distorção dependem do número n de iterações do algoritmo. A cada passo, a distorção média total diminui, já que $|r_n|$ é limitado por $C\alpha^n$.

A taxa de dados depende da distribuição estatística dos coeficientes k_i. No entanto, como os dicionários são em geral isotrópicos e não correlacionados aos vetores a serem codificados, podemos esperar que todos os valores possíveis de k apareçam com a mesma probabilidade. Assim, a taxa total será $R_T = n.\log_2(M)$ bits por símbolo, onde M é a cardinalidade do dicionário.

Portanto, a taxa por coeficiente, em bits, será $R = \frac{n.\log_2(M)}{N} = n.\log_2\left(M^{\frac{1}{N}}\right)$.

Naturalmente, a distorção total obtida depende da distribuição espacial dos vetores a serem codificados. No entanto, o comportamento assintótico de $|r_n|$ depende unicamente da forma do dicionário, já que os resíduos tendem a se distribuir por todo o P_α. Ou seja, a distribuição espacial dos r_n após muitos passos não depende mais da distribuição inicial dos r_0. Assim, assintoticamente, teremos $|\bar{r_n}| = C_0\alpha^n$, onde C_0 é alguma constante positiva. A distorção total então será $D_T = C_0^2\alpha^{2n}$; a distorção por coeficiente será $D = \frac{C_0^2}{N}\alpha^{2n}$. Finalmente, a distorção média por coeficiente em função da taxa será:

$$D(R) = \frac{C_0^2}{N}\left(\alpha^{\frac{1}{\log_2\left(M^{\frac{1}{N}}\right)}}\right)^{2R} \tag{7.1}$$

Assim, a fim de minimizar a distorção para taxas altas (isto é, para um grande número de passos, n), devemos minimizar $\alpha^{\frac{1}{\log_2\left(M^{\frac{1}{N}}\right)}}$. Como sabemos que $\alpha_c \geq M^{-\frac{1}{N}}$ (Seção 4.3.3), podemos escrever que $M^{\frac{1}{N}} = \frac{k}{\alpha} \geq \frac{1}{\alpha}$, com $k \geq 1$. Analisando a função $\alpha^{\frac{1}{\log_2\left(\frac{k}{\alpha}\right)}} = 2^{\frac{\log_2\alpha}{\log_2 k - \log_2\alpha}}$, dada a restrição $k \geq 1$, verificamos que seu mínimo ocorre para $k = 1$. Neste caso ótimo), de acordo com a Eq. 7.1, temos $D(R) = \frac{C_0^2}{N}2^{-2R}$.

Para os dicionários correspondentes à expansão binária tradicional, temos $M = 2^N$ e $\alpha = \frac{1}{2}$, e portanto $k = 1$. Concluímos, pois, que para *altas taxas* de dados, tais dicionários serão ótimos para α-expansões. Como para estes dicionários temos $M^{\frac{1}{N}} = 2$, o seu de-

sempenho taxa $\times$ distorção não se altera com a dimensão N, e portanto neste caso as quantizações escalar e vetorial são equivalentes.

7.4 Preparação dos experimentos

Para os experimentos descritos a seguir, determinou-se a distorção média por coeficiente D (MSE) em decibéis, isto é, $10\log_{10}(\mathsf{D})$, e a taxa de dados acumulada R em bits, para 20 iterações do algoritmo de α-expansão. Detalhes sobre os dicionários e conjuntos de dados utilizados são apresentados a seguir.

7.4.1 Geração dos dicionários

Como vimos, a convergência é mais rápida para valores menores de α. Além disso, dados M e N, dicionários com vetores mais bem distribuídos no $\mathbb{R}^{\mathsf{N}}$ tendem a possuir um α_{c} menor. Observe que não definimos exatamente o que se entende por "bem distribuído"; apenas utilizamos o termo no sentido intuitivo.

Este problema de distribuição assemelha-se ao problema do preenchimento do espaço com esferas, estudado exaustivamente por Conway & Sloane em [8]. Outros resultados relevantes também aparecem em [35, 36]. As soluções deste tipo de problemas (*sphere packings*) envolvem certas estruturas compostas por distribuições regulares de pontos no espaço, conhecidas como *lattices* ou reticulados regulares. Os sub-conjuntos compostos por alguns estratos destes reticulados (estratos definidos pela distância de seus pontos à origem) são conhecidos como *shells*, ou camadas. Os bons resultados obtidos com estas estruturas nos levaram a experimentar dicionários formados por algumas das camadas correspondentes.

Assim, nos experimentos descritos a seguir, utilizaram-se 3 classes de dicionários; normalmente nos referimos a estes dicionários pelo nome do politopo cujos vértices utilizamos

como vetores do dicionário. As classes são:

- Os dicionários equivalentes à expansão binária, em diversas dimensões: quadrados, cubos, tesseracts, etc. — conhecidos conjuntamente como hiper-cubos. Inclui-se aí o "quadrado" unidimensional equivalente à quantização binária escalar. Nos referimos à esta classe como os dicionários binários.

- Alguns polígonos (isto é, politopos bidimensionais) regulares.

- Camadas de algumas soluções de *sphere packings*. Por exemplo, D4 é um dicionário quadri-dimensional de 24 vetores, obtido a partir da primeira camada do conjunto de esferas mais denso que se conhece em 4 dimensões.

Muitos dos métodos modernos de codificação utilizam quantização escalar uniforme, exceto nas regiões próximas à origem, onde uma região (*quantizer bin*) maior que as outras é utilizada. O raciocínio que leva a esta escolha é que para conjuntos de dados típicos obtidos de transformadas de imagens, muitos dos vetores resultantes têm pequena magnitude. A região de quantização maior próximo à origem faz com um maior número de pontos seja codificado inicialmente como zeros. Vale lembrar que a maior parte da informação está nos coeficientes de maior magnitude. Desta forma, nos experimentos a seguir também utilizamos variações dos dicionários anteriores, nas quais se agrega o vetor nulo como elemento do dicionário. Isto cria uma região de Voronoi envolvendo a origem, e vetores que antes eram codificados inicialmente como um dos vértices do dicionário são então codificados como zeros.

7.4.2 Geração dos conjuntos de dados

Mais uma vez, utilizamos 3 classes de conjuntos de dados:

- Vetores uniformemente distribuídos no espaço, dentro da esfera N-dimensional de raio 1. Estes conjuntos de dados servem basicamente para a determinação experimental dos valores de α e γ a serem utilizados, mas também para comparação de dicionários.

- Coeficientes aleatórios com distribuição laplaciana. Estes conjuntos simulam com mais realismo o que se obteria a partir de coeficientes de transformadas de imagens no domínio da freqüência, que são as efetivamente usadas em compressão.

- Conjuntos de vetores formados a partir de coeficientes de transformadas Wavelet de imagens naturais. Os valores obtidos aqui são os mais importantes, pois esta é a aplicação principal das α-expansões, no contexto deste trabalho.

Todos os conjuntos de vetores assim obtidos foram normalizados. As variações nas distribuições dos conjuntos de dados aplicam-se unicamente aos raios; as distribuições de direção espacial são sempre uniformes.

7.4.3 Codificação dos zeros

Usualmente, na codificação de imagens e vídeo, os zeros são codificados de forma diferente dos outros valores. Isto porque a maioria absoluta dos coeficientes a serem codificados são de pequena magnitude, e portanto são aproximados como zero; uma codificação que utilize um mínimo de bits para codificar os zeros torna o método mais eficiente. Esta é uma das razões pela quais o método das *zerotrees* e seus similares são tão eficientes (Seção 2.1).

Desta forma, a seqüência resultante da codificação é composta basicamente de símbolos que representam os vetores não nulos. A fim de simular este efeito, na maioria dos experimentos descritos a seguir as taxas de dados foram calculadas sem levar em conta os elementos codificados como zeros. Isto justifica-se assumindo que algum método eficiente de codificação para os zeros será adotado, resultando em uma taxa adicional desprezível.

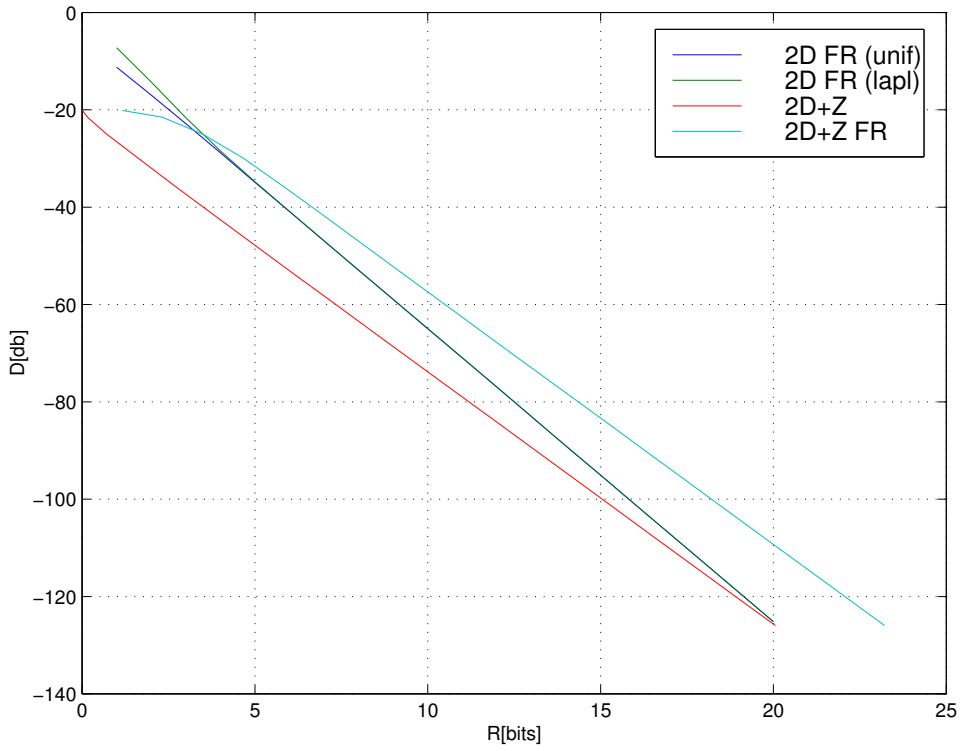


Figura 7.1: Gráfico I

Nos resultados que se seguem, quando os zeros são codificados normalmente como os outros valores, as taxas de dados resultantes são indicadas como “FR” (full rate) nos gráficos correspondentes.

7.5 Comparação de desempenho dos dicionários

O desempenho dos dicionários binários é tomado como a referência contra a qual os outros dicionários são comparados. Nos gráficos a seguir, é representado pela linha azul-escura. Vale lembrar que desejamos a menor a distorção possível (eixo vertical) para uma dada taxa de dados (eixo horizontal); assim, as curvas mais próximas do eixo horizontal são as que apresentam o melhor desempenho.

No Fig. 7.1, são mostradas as curvas $\mathsf{D} \times \mathsf{R}$ para os seguintes casos:

2D FR (unif): Conjunto de dados uniforme no espaço, codificado com dicionário binário 2D, full rate.

2D FR (lapl): Idem, conjunto de dados com distribuição laplaciana.

2D+Z: Distribuição laplaciana, dicionário binário 2D com a origem incluída.

2D+Z FR: Idem, full rate.

Algumas características importantes das α-expansões podem ser percebidas a partir dos gráficos:

- O dicionário binário "puro" tem o melhor desempenho assintótico (isto é, sua curva $\mathsf{R} \times \mathsf{D}$ apresenta a maior inclinação na região direita do gráfico, correspondente às taxas de dados mais altas), em comparação com o dicionário no qual se incluiu a origem. Este comportamento era esperado, visto os resultados da Seção. 7.3.

- A distribuição estatística do conjunto de dados influencia fortemente os resultados nos primeiros passos (região esquerda do gráfico), mas a forma do dicionário dita o desempenho assintótico, quando a distribuição dos resíduos atinge sua fase estável. Isto explica o fato de as curvas para as distribuições uniforme e laplaciana se sobreporem para altas taxas de dados.

- A melhora de desempenho com a inclusão do vetor nulo no dicionário é evidente (*2D+Z*), mas tal ganho só é efetivo quando não se considera a taxa de dados correspondente à codificação dos zeros; caso contrário, este dicionário rapidamente tem seu desempenho comprometido quando a distribuição de resíduos não mais se concentra próxima à origem (*2D+Z, FR*).

Embora a curva padrão de referência para as comparações tenha sido calculada com base no dicionário binário 2D, a dimensão neste caso não tem qualquer influência no resultado.

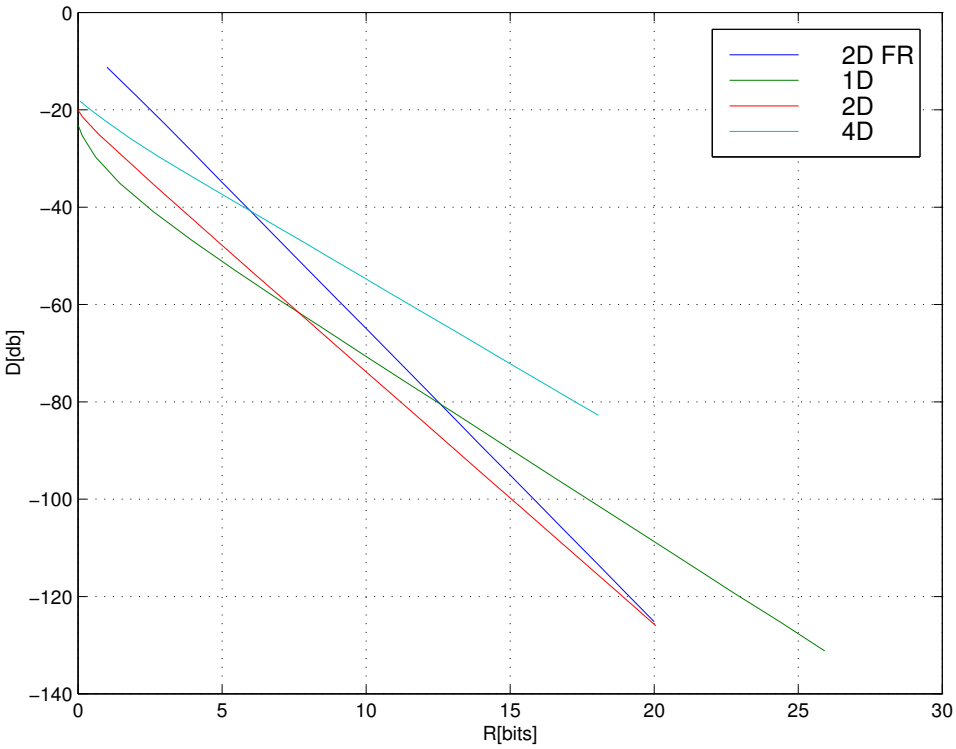


Figura 7.2: Gráfico II

Todos os dicionários correspondentes às expansões usuais, em qualquer dimensão, têm o mesmo desempenho que o quantizador escalar uniforme. Isto ocorre pelo fato de os vetores do dicionário serem ortogonais, o que nos permite separar os coeficientes de todos os vetores envolvidos em conjuntos independentes de escalares. Da mesma forma, expansões em qualquer base são equivalentes: a expansão binária não apresenta qualquer vantagem sobre a decimal, ou vice-versa.

O gráfico II (Fig. 7.2) mostra as curvas $\mathsf{R} \times \mathsf{D}$ para os dicionários binários (mais a origem) de $\mathsf{1}$, $\mathsf{2}$ e $\mathsf{4}$ dimensões, aplicados a dados Laplacianos. Ao contrário do caso anterior, verificamos que agora a dimensão do dicionário influencia sensivelmente o resultado, e para taxas baixas o dicionário $\mathsf{1}$-D apresenta grande vantagem.

Os gráficos III e IV (Figs. 7.3 e 7.4) mostram os resultados obtidos codificando-se os coeficientes de uma transformada Wavelet da imagem Lena $\mathsf{512} \times \mathsf{512}$ (uma imagem natu-

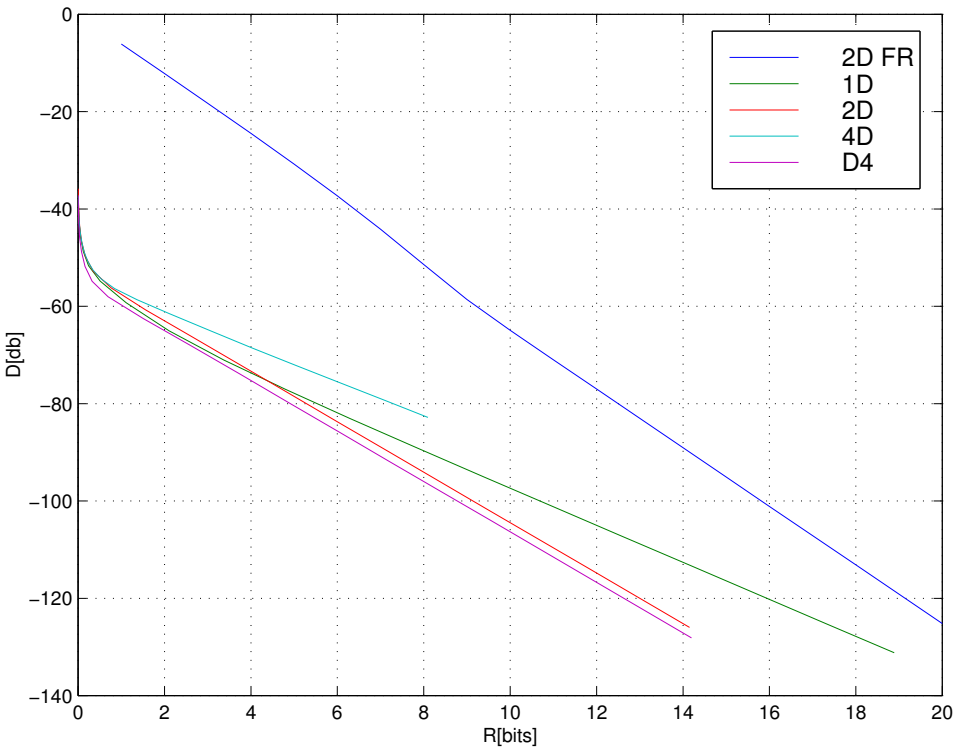


Figura 7.3: Gráfico III

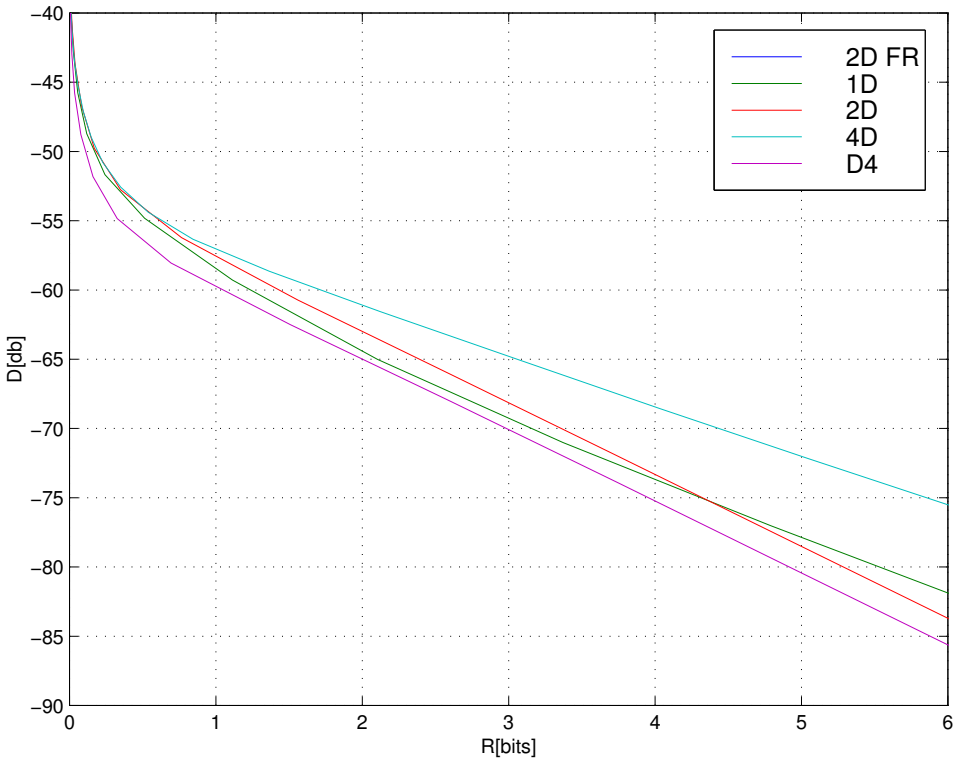


Figura 7.4: Gráfico IV

ral bastante conhecido neste campo de pesquisa). Além dos mesmos dicionários binários utilizados anteriormente, experimentaram-se os dicionários binário 4-D (tesseract) e D4 (derivado do lattice D4 descrito anteriormente), com a origem. Os resultados são semelhantes aos obtidos com dados Laplacianos, mas a distribuição aqui é muito mais concentrada em torno da origem. Deste modo, as conseqüências de se incluir a origem no dicionário são dramaticamente acentuadas. Como se pode ver nos gráficos, o dicionário D4 mostra um desempenho excelente em uma grande faixa de taxas de dados.

O gráfico IV mostra os mesmas curvas que o anterior, mas ampliado para mostrar em mais detalhe a região onde as diversas curvas se aproximam. Nele é possível perceber a grande vantagem apresentada pelo dicionário D4.

Também foram conduzidos experimentos (independentes, mas baseados nesta teoria)

com o algoritmo MGE [15], numa taxa de dados de 0.5 bpp. O PSNR obtido para o caso escalar (algoritmo original) foi de 36.68 dB, enquanto que o algoritmo modificado, utilizando o dicionário Λ_{16} (mais uma vez, baseado no lattice regular Λ_{16}, que representa a distribuição mais densa que se conhece de esferas em 16 dimensões [8]), apresentou um PSNR de 37.11 dB, com $\alpha = 0.6$. Isto demonstra o potencial das α-expansões de melhorar o desempenho de métodos existentes de compressão, através da alteração do algoritmo de codificação e da escolha de um dicionário adequado.

Capítulo 8

α-expansões e decomposições em *frames*

Neste capítulo faz-se uma breve comparação das α-expansões com as decomposições em *frames*. Esta aplicação das α-expansões foi objeto de publicação no *Applied and Computational Harmonic Analysis Journal* [4].

8.1 Decomposições em frames

Seja $\mathcal{F} = \{e_1, e_2, \ldots, e_p\}$ uma coleção de vetores que gera o $\mathbb{R}^N$. Isto significa que todo $x \in \mathbb{R}^N$ pode ser expresso como

$$x = \sum_{i=1}^{p} a_i e_i$$

para determinados coeficientes $a_1, a_2, \ldots, a_p$. Os vetores $\{e_1, e_2, \ldots, e_p\}$ podem ser ou não linearmente independentes. No caso em que eles são linearmente dependentes, o conjunto $\mathcal{F}$ é dito um "frame" ou uma base supercompleta. Para os propósitos desta discussão, no entanto, consideraremos $\mathcal{F}$ um frame mesmo no caso em que os vetores sejam linearmente independentes. Uma discussão mais completa sobre frames pode ser encontrada em [28, 38].

Dado um frame $\mathcal{F}$, considere o dicionário $\mathcal{C} = \{e_1, e_2, \ldots, e_p, -e_1, -e_2, \ldots, -e_p\} = \{\nu_1, \nu_2, \ldots, \nu_q\}$, onde $q = 2p$. Então a α-expansão sobre $\mathcal{C}$ de um vetor x, dada por

$$x = \sum_{i=0}^{\infty} \alpha^i \nu_{k_i}$$

pode ser vista como a decomposição de x em um frame $\mathcal{F}$. A vantagem de se utilizar α-expansões como um método de expansão em frames é que podemos armazená-las simplesmente codificando $(k_1, k_2, \ldots)$, enquanto que o procedimento usual envolve a decomposição e posterior quantização dos coeficientes.

8.2 Comparações entre as características de taxa-distorção

Podemos estimar a taxa em função da distorção como se segue. Os detalhes dos cálculos encontram-se no Apêndice I.

Para decomposições em frames, temos

$$R_1(D) \geq \frac{p}{2} \log_2 \left(\frac{C_1}{D} \right) \tag{8.1}$$

enquanto que para α-expansões temos

$$R_2(D) \leq \frac{\log_2(q)}{2 \log_2 \left(\frac{1}{\alpha} \right)} \log_2 \left(\frac{C_2}{D} \right) \tag{8.2}$$

onde C_1 e C_2 são constantes apropriadas. Assim, para uma mesma distorção D teremos

$$\frac{R_2(D)}{R_1(D)} \leq \left[\frac{\log_2(2p)}{p \log_2 \left(\frac{1}{\alpha} \right)} \right] \left[\frac{\log_2 \left(\frac{C_2}{D} \right)}{\log_2 \left(\frac{C_1}{D} \right)} \right]$$

Para altas taxas de dados, temos $D \ll C_1$ e $D \ll C_2$, de modo que a última fração aproxima-se de 1. Assim, para que tenhamos $R_2 \leq R_1$ é suficiente que tenhamos

$$\frac{\log_2(2p)}{\log_2(\frac{1}{\alpha})} \leq p$$

Um exemplo simples de um caso onde esta desigualdade é válida pode ser construído a partir do frame $\mathcal{F} = \left\{(1,0), (\frac{\sqrt{(3}}{2}, \frac{1}{2}), (-\frac{\sqrt{3}}{2}, \frac{1}{2})\right\}$ e do dicionário correspondente, que neste caso será um hexágono. Se decidirmos utilizar $\alpha = \frac{1}{2}$ (que é suficiente para a convergência), verificamos a desigualdade acima ($p = 3$) e concluímos que para este caso as α-expansões podem ser mais eficientes.

8.3 Observações

Não obstante o possível ganho em eficiência que se pode obter utilizando α-expansões ao invés de decomposições em frames, vale lembrar que as α-expansões podem requerer alto poder computacional, dependendo do dicionário utilizado.

Também é interessante observar a semelhança entre as α-expansões e o método *Matching Pursuits* (MP). O MP funciona projetando ortogonalmente um vetor de entrada no vetor mais próximo do dicionário, e então quantizando o módulo da projeção. Isto é repetido recursivamente para os resíduos assim gerados. O valor quantizado de cada projeção é codificado junto com o índice do vetor correspondente do dicionário. Diferentemente das expansões em frames usuais, no entanto, não se utiliza um quantizador uniforme. Isto torna a análise teórica do comportamento taxa-distorção extremamente difícil, impossibilitando uma comparação com as α-expansões. Lembramos, no entanto, que ao se utilizar α-expansões, não é necessário codificar o valor quantizado, apenas o índice do vetor correspondente do dicionário.

Capítulo 9

Conclusões

Os resultados apresentados no Cap. 7 são importantes por comprovar, com exemplos reais, a conjectura da existência de dicionários mais eficientes que o binário, ao menos para certos conjuntos de dados com características especiais. Lembramos que tais características não são artificialmente introduzidas para gerar tais resultados (o que seria possível), mas são sim encontradas usualmente nas transformadas no domínio da freqüência.

Assim, embora se tenha mostrado que os dicionários binários apresentem melhor desempenho assintótico, para qualquer distribuição inicial dos dados, também se verifica que para certas faixas de taxas de dados outros dicionários podem apresentar melhor desempenho. E estas faixas de dados são exatamente as que nos interessam para as aplicações de compressão de imagens.

Conjecturou-se a existência de dicionários mais eficientes que o binário ao se reconhecer que as α-expansões podem ser vistas como uma generalização das expansões binárias usuais. Assim, justifica-se a relevância da teoria aqui desenvolvida, não só por nos alertar para esta possibilidade, mas também por ajudar a explicar os bons resultados obtidos em [1], [2] e [6], onde uma generalização do algoritmo EZW [7] para vetores mostrou um ganho de desempenho em relação a resultados anteriores.

Finalmente, a teoria nos permite uma busca eficiente de dicionários de melhor desempenho, ao nos explicar as relações entre os valores de α, γ (o coeficiente de escalonamento) e a convergência do algoritmo.

Deve-se ter em mente que os dicionários aqui apresentados não são necessariamente ótimos; apenas desejou-se mostrar que eles são mais eficientes que o binário — e portanto que o quantizador escalar uniforme. Este fato justifica a busca de outros dicionários, possivelmente ainda mais eficientes. Esta busca limita-se, atualmente, a dicionários cujas dimensão e cardinalidade permitam sua utilização prática; afinal, o custo computacional do método de codificação aqui sugerido cresce exponencialmente com estes valores. No entanto, o dicionário D4, por exemplo, é prático o suficiente para justificar sua utilização. E enquanto o desenvolvimento computacional do *hardware* computacional continuar seguindo as leis de Moore, tendo sua capacidade de processamento também crescendo exponencialmente com o tempo, é de se esperar que outros dicionários possam vir a ser utilizados num futuro próximo.

Capítulo 10

Publicações

Ao longo do desenvolvimento deste trabalho, partes da teoria aqui desenvolvida e outros trabalhos baseados nela foram apresentados em congressos e/ou em publicações especializadas. Segue-se a lista destas publicações.

- Poster "Successive Approximations Vector Quantization" no 4º congresso *Curves and Surfaces* da Association Française d'Approximation, Saint Malo, França, Jul 1999, com Marcos Craizer e Eduardo A. B. da Silva.

- "Quantized Frame Decompositions", em *Saint-Malo 1999' Curve and Surface Fitting*, págs. 153–160, Vanderbilt University Press, 2000, com M. Craizer e Eduardo A. B. da Silva.

- "Alpha-expansions: a class of frame decompositions", no *Applied and Computational Harmonic Analysis Journal*, Vol. 13, issue 2, págs. 103–115, Set 2002, Academic Press, com M. Craizer e Eduardo A. B. da Silva.

- "Successive Approximation Quantization for Image Compression", no *IEEE Circuits & Systems Magazine*, 3rd Quarter 2002, págs. 20–45, com Eduardo A. B. da Silva e M. Craizer.

Apêndice A

Introduction

"An image is worth ten thousand words."

— Frederick R. Barnard

Images may be worth ten thousand words[1], but they generally occupy much more space in a hard disk, or bandwidth in a transmission system, than their proverbial counterpart. In more general terms, the digital representation of a physical signal is often a resource hog. So, in the broad field of digital signal processing, a very high-activity area is the research for space-efficient signal representations. Efficiency, in this context, generally means to have a representation from which we can recover some approximation of the original signal, but which does not occupy a lot of space. Unfortunately, these are contradictory requirements; in order to have better pictures, we usually need more bits.

This work presents new ideas regarding the space-efficient representation of images. More specifically, it develops the theory of successive approximations as applied to the coefficients of wavelet transforms of images. This theory, however, applies equally well to a more general class of signals.

[1] Or just one thousand, as the more popular version goes.

A.1 Digital Signal Processing

In order to better situate the specific subject of this work, it is worthwhile to do a brief review of some of the several issues involved in digital signal processing, particularly in image processing.

A.1.1 Signals

The signals which we want to store or transmit usually come from physical phenomena like sounds or images, which are really continuous functions of time or space. Of course, in order to use digital computers to work on them, we must *digitize* those signals. This is normally accomplished by sampling (measuring its instantaneous value from time to time, or from point to point) and finely quantizing the signal (assigning a high-precision, discrete value to the measurement) [10].

This procedure will produce long, ordered series of numbers, normally in the form of vectors or matrices. For all purposes of this work, from now on we will proceed as if these sequences were the original signals which need to be stored or transmitted, and the ones which we will eventually want to recover. After all, we can consider that from this digitized representation we can recover the true (physical) signal. This is true for audio or imaging applications, for example, as long as human eyes or ears are concerned. This is what happens, for example, when we play an audio CD.

A.1.2 Domains

This first form of representing a signal is said to be in the time (or space) domain. There are, however, other forms of representing a signal.

Certain classes of functions can be decomposed as infinite sums of weighted sines and cosines. This can be applied to signals, which can be likewise decomposed as a sum of pure-

frequency signals. The phases and weights of the different frequencies — the coefficients of the component functions — are all we need to specify the original signal. Thus, this set of coefficients forms another possible representation for the same signal, which is referred to as being in the frequency domain.

Other forms of decomposition are also possible, basically by selecting different sets of component functions. Different objectives — signal analysis versus compression, for example — will be better served by different representations. Thus, each objective will require a different set of criteria to define what is a "good" representation.

A.1.3 Compression

Both time and frequency domain representations of a signal are usually very space hungry, making the storage or transmission of these digital representations very costly. We must use *compression* methods to modify these sets of numbers so as to require less space or bandwidth.

While truly invertible methods do exist which provide lossless compression (e.g., the Ziv-Lempel method of ZIP fame), they do not provide — for images — the efficiency levels we would like to have. For that we must use the so-called lossy compression methods.

Lossy image compression is usually obtained by leaving out "unnecessary" detail. For example, suppose our image is composed of 8-bit pixels, where a value from 0 to 255 represents the brightness of each pixel. We could retain only the most significant 4 bits of each pixel, which would result in an image with only 16 gray levels. We would have thus obtained 2:1 compression, although at the cost of a very degraded image. But what we are really looking for is an acceptable compromise between compression levels and image quality. In general, *acceptable* means a quality loss not readily apparent to a casual viewer. In particular, coarser quantization of brightness levels is usually unacceptable, because our visual system is very aware of the resulting difference [18]. Moreover, this difference is

Figura A.1: Lena 256x256 original

usually perceived as somewhat aggressive to the eye. For comparison, take an image where every other pixel has been abandoned and then recovered by interpolation: although the difference to the original is still quite visible, the subjective result is much better. These artifacts can be perceived[2] in Fig. A.2 (a) and (b); compare them to the original picture in Fig. A.1.

By selecting different domains to specify a signal, we are able to choose diverse qualities of "details" to leave out, giving rise to distinct kinds of artifacts in the recovered images. This way we can select the domain which provides the best results, i.e., that whose artifacts — caused by quantization — are less noticeable to normal human vision. In general, the use of frequency-like domains is the best choice for this purpose.

[2]The differences can be hard to see in print; they are more easily seen when the images are displayed in a CRT screen.

(a) 4 bpp

(b) Interpolated

Figura A.2: Lena (compressed)

A.1.4 Transforms

Different decompositions of the same signal can be obtained from one another by means of mathematical transforms — basically, matrix algebra. Those are useful when we need different representations for the same signal.

In general, the time domain representation is seen as the "original" one, since it is the easiest to obtain from the respective physical phenomenon. Also, it is usually needed to reproduce the same phenomenon. From the time domain we can obtain other decompositions, and from them, recover the original. That is, those transforms are invertible.

Invertible transforms are widely used in image processing. Transforming an image, in this context, can be seen as decomposing it as a weighted sum of elementary images. Mathematically, this is obtained by multiplying the number sequences which represent the signal (taken as a vector) by a certain matrix [18]. The elements of the resulting vector — the transformed image — have now a totally different meaning, not directly correlated with the brightness value of any single pixel. On the contrary, each coefficient of the transformed signal is a function of several pixels of the original image. Conversely, any alteration in a transform coefficient will translate to a modification in most of the pixels of the recovered image, when the inverse transform is applied.

This is where most of the compression is obtained in modern methods: by carefully quantizing the different coefficients of the transformed image [19]. Depending on the transform used, the loss of precision in certain coefficients is less visible than in others, when considering the effects of the resulting recovered images in the human visual system [20, 27].

Of course some transforms are better suited than others for this purpose. The JPEG standard [30] for compression of static images uses the Discrete Cosine Transform, and so do the several variants of the MPEG standard [32, 33, 34] for compression of moving images.

More recent algorithms make use of the Wavelet Transform. It is used in the modern JPEG2000 standard [31], based on the EBCOT algorithm [12]. The MPEG-4 standard also allows (optionally) the use of wavelets.

A.1.5 Coders

Up to this point, in our discussion about different representations for a signal, we have come up with large sets of *numbers* — the transforms coefficients. So far, these are just abstract quantities. The next step in order to actually store or transmit those numbers is to have them written down as *symbols*. This process is called coding.

When coding a data set[3], a bit-stream is produced from which the data set may be recovered. The whole process may be not perfect, causing a difference between the original and the recovered data. One measure of this difference is the *distortion*, which will be defined in a later section.

That bit-stream also has a certain size that can be measured simply by counting the bits involved. This size will naturally be proportional to the size of the original data set. Thus, their ratio measures the compression obtained by the coding process and is referred to as its *data rate*. In general, there is a compromise between rate and distortion: we would like both to be very low, but these are conflicting requirements. A coder may be designed to produce a given rate or a given distortion, based on certain external controls. The quality of a coder can be measured by its rate $\times$ distortion characteristics, as well as by the extent to which one or the other parameter can be controlled.

A.1.6 Image quality, or lack thereof

The ultimate assessment of the quality of a compression method is the subjective impression caused by the recovered images on suitably trained human individuals. There's hardly any

[3]That is, executing a specific algorithm to choose symbols to represent the data set.

need to say that this is but impossible to determine a priori algorithmically, so any objective methods for quality assessment fall very short of this ideal case [27]. Nonetheless, some objective methods must be used, at least as a first-order approximation for the quality of a method. Usually, what is used is some function of the brightness differences of the pixels in the original and the recovered images. The most common objective measure of the quality of an image (in relation to an original) is the MSE – Mean Squared Error, defined [18] as

$$\text{MSE} \equiv \frac{1}{N}\sum (x_i - \hat{x}_i)^2$$

where x_i is the value of an original pixel, and $\hat{x}_i$ is the corresponding approximation. The square function is used to make the definition more easily tractable in mathematical terms.

The careful reader will have noticed that this is just the square of the Euclidean distance (within a multiplication factor) from the original image to its approximation : if the approximation is identical to the original, the MSE is zero, and it grows as the approximation departs from the original. This is a measure of *distortion*. For those who think that more is better, there is an inverse measure derived from the MSE, which is the PSNR – Peak Signal to Noise Ratio. For 256-gray level images, this is defined as

$$\text{PSNR} \equiv 10 \log_{10}\left(\frac{255^2}{\text{MSE}}\right) \text{[dB]}$$

Given the simplistic definition of distortion used here, we should keep in mind that those measures have a somewhat restricted use. The comparison of MSE or PSNR values is meaningful only when comparing similar algorithms applied to a common original image, and even then these results must be taken with a grain of salt. That said, the MSE is by far the most used form of comparing the image quality obtained from competing algorithms. Computational cost is often also taken into account.

A.1.7 Data rates

The data rate is the other factor in the compression equation. We can define the data rate as the average number of bits per pixel needed to represent an image, and it has direct consequences on file sizes and bandwidth needs. For any given algorithm, better quality will imply larger data rates. The set of distortion versus rate measurements for an algorithm is referred to as its rate-distortion performance.

Information theory provides us with a theoretical upper bound on the performance of any coding algorithm, given the statistical behavior of the source of the coefficients [22]. A certain representation will be said to be optimal when this upper bound is achieved.

A.2 Successive Approximations

Several state-of-the-art image and video compression methods make use of successive approximation quantization, in one form or another [7, 11, 12, 15, 31]. It is found in the literature under several names, as successive refinements, embedded coding, or progressive quantization [12, 14, 16, 17]. However, all these refer to the same basic idea, namely that of achieving a representation for a signal made up of several segments; from the first, we can recover a coarse approximation of the original, and then the next segments give us finer and finer detail. Thus, by carefully controlling how many segments we store or transmit, we may control either the amount of data involved or the quality of the recovered approximation.

This is in fact a very simple idea and is implemented, for example, in the most common form of number representation, which is the decimal expansion. For example, when we want to refer to the value of π, we may use the string "3.14"; if we need more precision (and we can afford the cost), we may then use "3.14159". This second string contains the first at its beginning, and then some more data which does not mean anything by itself,

but does give us more information about the value of π when used together with the first part.

Sadly, the decimal expansion is good only for human consumption; besides, it's not optimal for representing signals. Its digital cousin, the binary expansion, is better suited for computer use, but is equally non-optimal. Suffice it to show that the optimal (least error) 5-digit representation for π is 3.1416; to reach the optimal 6-digit representation, we must get rid of the final 6, substituting 59 for it. So, this representation does not make optimal use of the allowed space.

An *embedded code*, as defined by Shapiro [7] has two defining characteristics:

- When coding the same data with two different rates (and, implicitly, with different distortions), the two resulting representations must match exactly for the extent of the smaller one. In other words, the smaller one must be exactly reproduced in the beginning of the larger one. This way, the coded representation for a given data rate contains all the representations for smaller data rates. In short, the representations get more precise as we add more symbols to it. Thinking in circles again, "3.14" is a (distorted) representation for π, and "3.1415" is a more precise one.

- It should obtain the best possible representation (lowest distortion) for a given data rate. Again, "3.1416" is a better approximation for π than "3.1415", but the next step is "3.14159"; thus, the usual decimal expansion is not strictly an embedded coding.

The former characteristic conveys the main idea of embedding, while the latter guards against excessive generalization.[4] For example, the usual binary or decimal representations can be considered embedded, if we take some precautions on reconstructing values, such as adding $\frac{1}{2^{(n+1)}}$ to a truncated (binary) number. On the other hand, representations

[4]Just appending a whole high-rate description to a low-rate one, for example, should not qualify as embedding.

generated by vector quantization [28] are not ordinarily embedded; different resolutions may produce totally different coded representations (see Sec. E.3).

Embedded coding are specially interesting for progressive or selective transmission of data. For example, a client application may display increasingly better versions of an image while it is downloaded from a server. The client may even limit the downloads to an acceptable quality, size or time limit. The ability of embedded coders to finely control rate or distortion when coding is also highly appreciated.

Observe, however, that producing a higher-rate coding in a single step can potentially generate a better (lower distortion) representation than doing it in stages; this comes from the fact that higher-dimensional vector quantizers are more efficient [9]. On the other hand, those quantizers have fixed rate; the codes they produce become meaningless if truncated.

The possible losses in efficiency incurred by the use of embedded coding may well be compensated by gains in flexibility (to dynamically choose a data rate) and algorithm simplicity.

A.3 α-expansions

The coding of real numbers is a well established field. The usual decimal or binary expansions — a kind of scalar quantization — are adequate for human or computer manipulation, while vector quantization methods provide efficient representations, in rate $\times$ distortion terms. There are applications, though, for which these methods are not well suited.

For most applications, we would like to have:

- Variable rate;
- Lowest possible distortion for a given rate;
- Reasonable computational costs.

Prompted by the success of vector quantization, we are well aware that it may be convenient to group numbers into vectors before coding them. So, we'll use a method directed at the encoding of vectors, namely α-*expansions*. We will verify to what extent this method fits the requirements stated above.

A.3.1 Definition

Let x be a vector in $\mathbb{R}^N$, and let $V = \{v_k \in \mathbb{R}^N, k \in \{1, 2, \ldots M\}\}$, where x is the vector to be coded and V is the *dictionary* or *codebook*.

The α-*expansion* coding of the vector x is:

$$x = \sum_{i=0}^{\infty} \alpha^i v_{k_i} \tag{A.1}$$

where $0 < \alpha < 1$ and $k_i \in \{1, 2, \ldots M\}$. This is what we will call an α-expansion over the codebook V.

This representation can be coded as the sequence $(k_0, k_1, \ldots)$ of integers. Geometrically, the relation expressed by equation Eq. A.1 could be shown as in Fig. A.3.

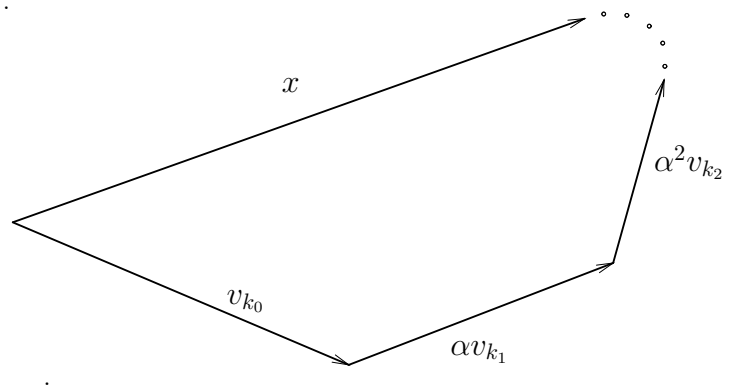


Figura A.3: First approximations

As the figure suggests, given only the first L elements $(k_0, k_1, \ldots k_{L-1})$ of the k-sequence

we can find an approximation x_L to the vector x:

$$x_L = \sum_{i=0}^{L-1} \alpha^i \nu_{k_i} \tag{A.2}$$

To measure the error incurred by this approximation we define the residue

$$r_L = x - x_L \tag{A.3}$$

For this truncated representation to be useful, we want $|r_L|$ to have a smoothly decreasing upper bound, so that r_L tends to zero as L grows. To be comparable to other coders, we need $|r_L|$ to be bounded (at least) by $C\beta^L$, where C is some positive real constant and $0 < \beta < 1$. Observe that this only means that as we proceed with the coding, the *maximum error* in the reconstructed vector will decrease exponentially; $|r_L|$ itself may vary otherwise, as long as it stays below this upper bound.

In the present case, $|r_L|$ is clearly bounded by $C\alpha^L$, for all x that *can* be represented as an α-expansion. This can be easily seen when we write

$$\begin{aligned} r_L &= \sum_{i=L}^{\infty} \alpha^i \nu_{k_i} \Rightarrow \\ |r_L| &\leq \sum_{i=L}^{\infty} \alpha^i |\nu_{k_i}| \\ &\leq \sum_{i=L}^{\infty} \alpha^i |\nu_k|_{\max} \\ &= |\nu_k|_{\max} \frac{1}{1-\alpha} \alpha^L \end{aligned}$$

The α-expansion coding of a vector satisfies the first requirement (Sec. A.2) for an embedded coder: higher rates are obtained adding elements to the sequence $(k_0, k_1, ...)$. So, it contains all lower-rate representations. The second requirement may also be met,

depending on the algorithm chosen to produce the sequence.

A.4 Scope

Eq. A.1 — the definition of an α-expansion — is the central point of this work. It embodies a very powerful concept in an elegantly simple form. We will examine the conditions under which such a representation is possible for all vectors of a given data set. With this aim, an α-expansions based theory for Successive Approximations has been developed, which will be presented in chapter D.

In addition, we will examine the adequacy of this method to a real-world problem in the area of digital image processing, namely the coding of coefficients of Wavelet Transforms. The experimental results obtained by applying the algorithms described here to several data sets are presented at the end.

Previous results also exist, both theoretical and experimental, and are reviewed in chapter C. These initial facts related to α-expansions first appeared in da Silva's PhD Thesis [2] and related work [6]; all other results stated here about the theory of α-expansions are believed to be original.

Apêndice B

Successive Approximation Quantization

Although being part of many state-of-the-art image compression methods, successive approximation quantization is not generally optimal from a rate-distortion perspective. In this section, we will first describe the EZW algorithm [7], a wavelet-based image encoder that was the first one to efficiently use the concept of successive approximation quantization in image processing. We will then present a more formal definition of successive approximation, and then proceed to show why it's not always optimal. That said, we will present the reasons why it's used in so many algorithms nonetheless.

B.1 The EZW algorithm

When the EZW — Embedded Zerotree Wavelet — algorithm for image compression was introduced [24], it represented a breakthrough in image compression methods. It is capable of compressing an image with excellent rate $\times$ distortion performance; besides, the generated bit-stream is such that the encoding rate can be precisely controlled, with optimal

Figura B.1: 3-stage wavelet image transform

performance for all rates. Several other algorithms have since been introduced, all based on the same principle [11, 12, 15]. The key to EZW efficiency lies in a clever combination of wavelet transforms and successive approximation quantization.

The 3-stage wavelet transform of a natural image [21], as can be seen in Figs. B.1 and B.2, has the following features:

- Its coefficients can be grouped in several frequency bands, obtained from an octave band decomposition. Band 0 is a low-pass band; bands 1, 4 and 7 have mainly vertical high frequency detail; bands 2, 5 and 8 have mainly horizontal detail, and bands 3, 6 and 9, diagonal detail. In each group, frequency increases as the band indices increase.

- It has a few high-energy coefficients and a large number of low-energy coefficients.

- The low-energy coefficients tend to appear in clusters.

- The coefficients in lower frequency bands tend to have higher energies than those in higher frequency bands.

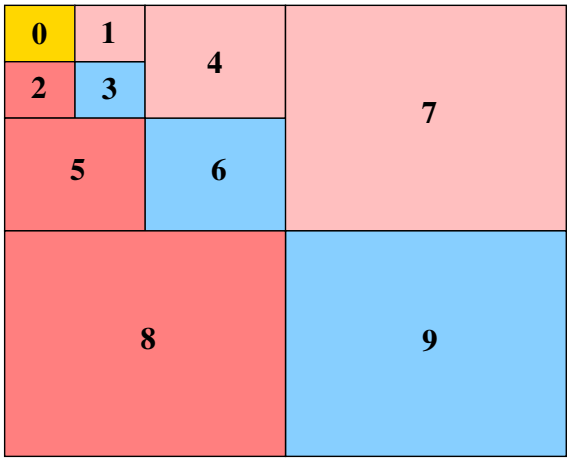


Figura B.2: Wavelet sub-bands

- The bands of same orientation (1, 4 and 7; 2, 5 and 8; 3, 6 and 9) tend to have their low-energy coefficients in the same corresponding spatial locations.

The EZW algorithm works by quantizing all coefficients using successive approximation quantization. It does so by first computing the magnitude of the highest energy coefficient, and then setting a threshold T equal to half this value. It then transmits the sign of all the coefficients with magnitude higher than T. It is important to note that it uses a very efficient method to encode the location of these coefficients. It considers coefficients with magnitude lower than T as non-significant; if a coefficient in a band is non-significant, and all spatially corresponding coefficients in the higher bands of the same group are also non-significant, this whole set is then encoded as a zerotree. Otherwise, that non-significant coefficient is simply encoded as a zero. Note that with a zerotree a single encoded symbol is used to represent a large number of coefficients; therefore, the zerotree concept is very efficient. Since the bands are self-similar, it is likely that a large number of zerotrees will occur, and thus just a small number of bits will be spent to encode all non-significant coefficients.

After the sign of the non-significant coefficients is transmitted, the threshold is halved.

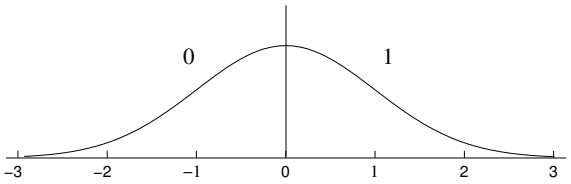
Then, the coefficients that are already significant are refined (by adding or subtracting $\frac{T}{2}$ from its reconstruction value), and the sign of the coefficients that just became significant are then transmitted. The whole process is repeated until a target distortion or bit-rate is achieved. All symbols are encoded using an arithmetic encoder [25]; details can be found in [7]. The rate $\times$ distortion characteristics of this encoder are very good; several variations of it have been proposed [11, 12, 15], based on the same basic principles. Such algorithms represented a breakthrough in image compression, and today they represent the state-of-the art in the field [12].

The successive approximation quantization lends an important characteristic to those algorithms: they are all embedded, that is, the bit-stream they generate for one rate contain the the bit-streams for all achievable lower rates (see Sec. A.2).

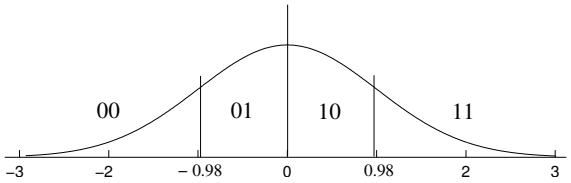
B.2 Not always optimal

Equitz and Cover (who use the terms Successive Refinements, in [14]) have shown that "optimal descriptions are not always refinements of one another"; they establish the conditions under which optimal successive refinements can occur. A simple example where this does *not* happen is also given, which goes as follows.

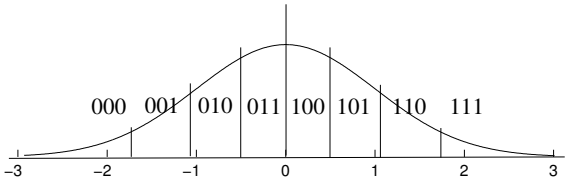
Take a sequence of numbers from a standard normal distribution $X \sim N(0, 1)$. The optimal 1-bit description for these numbers is obviously that which uses this bit to specify the sign of the number (see Fig. B.3a). If we now decide to use 2 bits, we can have 4 quantizer bins, as in Fig. B.3b. Notice that the new bins are subsets of the old bins. The placement of the bins' frontiers are defined by the Lloyd-Max conditions [28], in order to minimize the MSE of the reconstructed values. So far, we still have an embedded code; all numbers which were coded as 0 are now coded with a 0 as the first symbol. But if we proceed to 3-bit codes, the new bins (Fig. B.3c) are not subsets of the previous bins



(a) 1-bit coding



(b) 2-bit coding



(c) 3-bit coding

Figura B.3: Coding a Gaussian random variable

anymore; so, we do not have embedded coding if we decide to keep the codes optimal.

It should be noted that optimality with respect to rate $\times$ distortion depends on how we decide to measure distortion. Although the MSE is the most common form, nothing prevents us from choosing another metric as, for example, the average absolute value of the error.

In short, given a random source with known properties, and a distortion function, there is a lower bound on the average rate necessary to represent messages (symbol sequences) from that source with a given distortion [22, 23]. Optimal representations would be those which reach this lower bound, and they usually would not be embedded.

B.3 The case for Successive Approximations

The previous results seem to imply that embedded coding is not such a bright idea. However, there seems to be no good stochastic model for "natural" images, so we have to give up theoretical optimality anyway. But natural images do share some stochastic properties, such as a great deal of redundancy, both in space and in some transform domains. The best-performing image compression algorithms are those which are able to recognize this redundancy and so can represent large portions of a transform with few symbols.

State-of-the-art algorithms like EZW, previously described, can presently represent most images with less than 1 bit per pixel (bpp), with no discernible loss of quality [7, 11, 12, 15]. Most of those algorithms do use embedded coding of transform coefficients.

This is in apparent contradiction with the discussion shown in Sec. B.2. However, it turns out that no practical image coder is optimal, and embedded coders can be best, in certain cases, amongst the practical, non-optimal coders.

B.3.1 Low bit-rate quantization

Mallat & Falzon [13] have shown a mathematical analysis of the improvements obtained by embedded coding of transform coefficients when working with low bit rates, as follows.

A quantizer is considered to have high-resolution when the probability density function $p(x)$ of the quantized random variable X is approximately constant inside each quantizer bin [13]. The rate $\times$ distortion performance of high-resolution quantizers for Gaussian processes is well known; the MSE will vary as $C\left(2^{-2R}\right)$, with C depending on the bit allocation. In this case, the uniform quantizer has been shown to be optimal [28].

Current algorithms, however, operate on the $R < 1$ bpp region; therefore, the high-resolution hypothesis does not apply. In this case, the rate $\times$ distortion performance behaves differently.

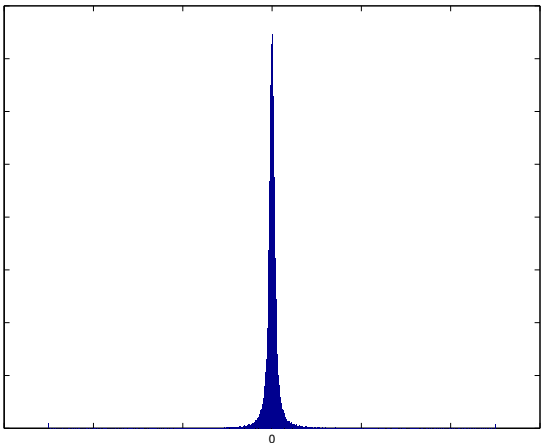


Figura B.4: Histogram for DCT of the Lena image

The wavelet or block cosine transform of most images have histograms resembling that of Fig. B.4, which depicts the histogram of the DCT of the Lena image. The clustering of coefficients near the zero region, together with the coarse quantization, result in a large number of coefficients being quantized as zero. Thus, the zero symbol deserves special treatment, and most algorithms treat it differently from the other values. For example,

the EZW algorithm, as seen before, uses zerotrees in order to efficiently encode the zero elements. Mallat & Falzon, in [13], show that when there is a known transform behavior as to the usual location of large, average and small coefficients — as is the case with wavelet transforms in which small or zero–valued coefficients tend to be clustered — embedded coders can present an advantage over classical coders. The net result is to have the MSE varying approximately as R^{-1}. This explains the good performance of embedded encoders such as EZW and the like [7, 11, 12, 15].

B.3.2 Applications of embedded codings

Besides presenting improved efficiency, embedded coders are better suited for several applications where the possibility of dynamically choosing rate or distortion is a plus.

Consider, for example, the multi-casting of a movie, i.e., the transmission, through a network, from one source to several destinations. The network may form a tree, with the source as root and the destinations as leaves. Each branch may have a different bandwidth. Now, if the source limits the transmission to the rate allowed by the narrower branch, the other branches will not get all the quality they could. On the other hand, if the source transmits at full rate, some branches may not be able to cope with the large data rate. However, if the transmission uses an embedded code, special hardware at each branching point may dynamically select from the received bit-stream just the amount of data which can be sent toward each of its outgoing branches. This way, all destinations get the largest possible rate — and thus image quality — allowed by the channel extending from them to the source.

As another example, imagine a high-resolution image database set up for remote browsing and downloading. We want to be able to make a fast browsing to select the images before actually downloading them. For this we must have a low-quality version of each image available, but the best dimensioning of these previews depend on the bandwidth of

our connection to the server as well as the time available to complete the job, and may vary from user to user.

If the image files are produced with a non-embedded coding, the only choice is to have separate files for the previews, which is inefficient; besides, there will be a fixed set of preview resolutions. Using embedded coding, the viewer can select[1] the data rate which best suits him or her as a compromise between image quality and download time.

B.4 A real-world example

Another nice example of a real-world method which uses successive approximations is Taubman's EBCOT [12] — Embedded Block Coding with Optimized Truncation, which is the basis for the JPEG2000 standard [31].

The successive approximations in EBCOT appear at the bit-plane coding of the quantized values of the wavelet transform coefficients. It uses uniform, scalar quantization, except for the near-zero region. In bit-plane coding, the most significant bits of a set of values are sent first, then the next most significant, and so on, until all bit-planes have been sent. Of course, like in other modern algorithms, the bit-planes are also further entropy coded to take advantage of the redundancy present in the transform coefficients.

[1] We suppose the server can generate the previews at the selected data rate by selectively transmitting the proper subsets of the embedded coding.

Apêndice C

The Background on α-expansions

In this chapter, we will review the existing results on the theory of α-expansions, as developed in [1], [2] and [6].

C.1 The Algorithm

In [1], α-expansions were used to quantize wavelet coefficients. The method used was to group the coefficients into N-dimensional vectors and then find k-sequences to represent them. The method was restricted to codebooks of unitary vectors.

A *greedy* algorithm was used to find the k-sequence corresponding to a vector x, as follows:

1. Take $r = x$.

2. Find v_k, the codevector nearest to r, i.e., the one which minimizes $|r - v_j|$. As all codevectors are of unit length, this is equivalent to find the codevector which minimizes $\text{ang}\,(r, v_j)$, the angle between r and v_j.

3. Take $(r - v_k)$ as the next value of r.

4. Multiply all codevectors by α.

5. If $|r|$ is small enough, stop; else, go back to step 2.

The algorithm is said to converge when $|r|$ decreases exponentially, i.e., it is bound by $C\alpha^n$ after the n-th step, where C is some real constant. In this case, the α-expansion for this particular x is defined by the sequence of numbers k found at each iteration of step 2. The fact that we minimize the approximation error at each step justifies the classification of the algorithm as a greedy one.

Now, take an x formed by a given, random k-sequence. Observe that there is no guarantee that the algorithm will find the same k-sequence. In fact, it can even find a k-sequence for which $|r_L|$ does *not* decrease exponentially, i.e., it may diverge.

C.2 Conditions for Convergence

Experiments were conducted [2] to study the convergence of this algorithm for several codebooks, and the results have shown that the α necessary for convergence depends on a certain geometric characteristic of the codebook.

C.2.1 The codebook spatial distribution and θ_{max}

For every point to be coded there will be one codevector which will be nearest to it, both in distance and in angle separation. This angle is $\arccos < x, v_{k_n} >$. Define θ_{max} as the largest such angle, taken from all points on the sphere surface. For other points, except the origin, there will be a point on the surface with the same angle; the origin may be considered to be at angle zero to all codevectors. Thus,

$$\theta_{max} = \max_{x} \left\{ \min_{j} \{ang(x, v_j)\} \right\}$$

As an example, Fig C.1 shows $\theta_{\max}$ for a 2-D codebook.

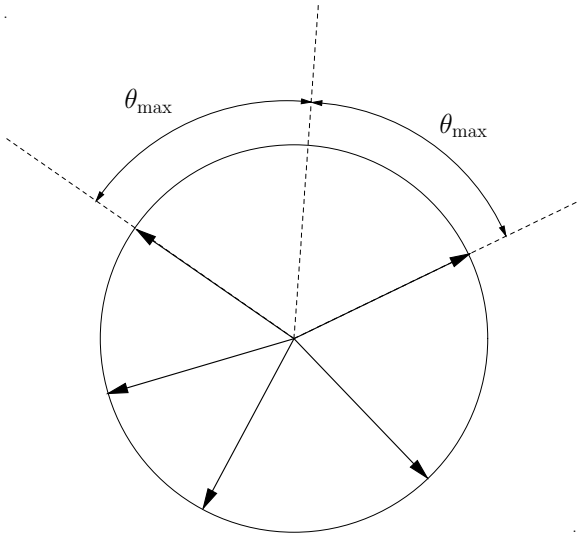


Figura C.1: $\theta_{\max}$ for a 2-D codebook

We want the codebook to be able to generate the origin (amongst other points as well), i.e., the zero vector must be obtainable as a convex combination of the codevectors[1]. This implies that we must have $\theta_{\max} < \frac{\pi}{2}$. In this chapter, we are assuming this kind of codebooks.

To see why this last condition is needed, observe Fig. C.2. Although it depicts a 2-D case, the following discussion is valid for any number of dimensions. If $\theta_{\max} \geq \frac{\pi}{2}$, then there will exist at least one point P such that all codevectors are at or farther than $\frac{\pi}{2}$ from it. We can then have an hyper-plane AA' containing the origin and orthogonal to $\overline{OP}$ such that all codevectors will be contained in the same half-space opposite to P. Thus, any convex combination of the codevectors will always have a component on this half-space, and so points like P or the origin cannot be generated by an α-expansion over this codebook.

[1]This is necessary, but not sufficient, to generate the origin as an α -expansion.

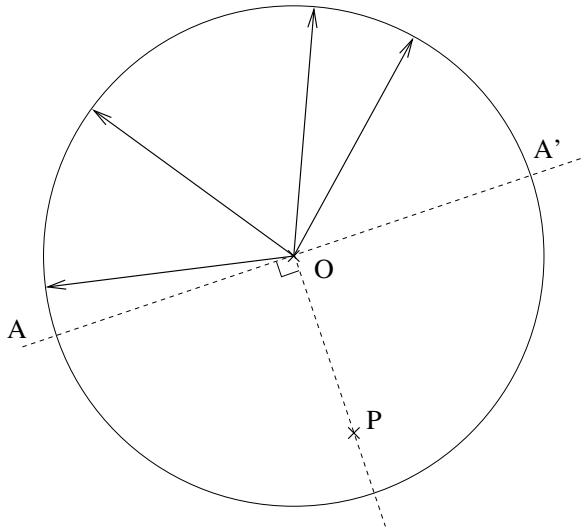


Figura C.2: A half-space

C.2.2 α and θ_{max}

The following interesting results have been developed from the experiments [2]:

- For codebooks with $\theta_{max} \leq 82°$, the algorithm always converged, provided a suitable α was used.
- For any codebook, convergence requires that $\alpha \geq \frac{1}{2}$.
- In general, smaller values of α required smaller values of θ_{max} for convergence.

C.2.3 Consequences

The above results provide a valuable insight on a characteristic of "good" codebooks, namely a small θ_{max} (because a smaller α provides smaller residues).

Thus, codebooks with a given number of vectors can be optimized to attain a minimum possible θ_{max}. This can be achieved by spreading the vectors more "regularly"[2] over the

[2]In fact, regularity here should be defined exactly so as to minimize θ_{max}, but the result would be close enough to the everyday meaning of the word.

sphere. Also, codebooks with more vectors can have smaller values of $\theta_{\max}$.

These results are somewhat expected. The more codevectors we have to choose at each step, the better the approximation. If the codevectors are clustered around certain regions on the sphere, other regions will be depleted, and data points on the latter regions will not be well represented by any codevector.

Taken as a whole, this knowledge points to a special class of codebooks: those originated by the *regular lattices* related to sphere packings [8]. In general, these lattices produce well distributed codebooks and have other interesting properties: they have well known structural characteristics and some have simple and fast encoding algorithms, leading to efficient implementations.

It should be noted, however, that in general the requirement to have a small α leads to codebooks with more vectors. This, in turn, enlarges the bit-rate of the representation. So, a compromise must be chosen to optimize rate $\times$ distortion.

C.3 $\overline{\alpha} \times \theta_{\max}$

The previous results prompted the search for a more precise evaluation of the relationship between α and $\theta_{\max}$. More specifically, it was noted that for each codebook there was a special value $\alpha_{\min}$ that guaranteed convergence for any $\alpha \geq \alpha_{\min}$ (and, conversely, guaranteed non-convergence for any $\alpha < \alpha_{\min}$). In [6], the following theorem provided the desired result.

C.3.1 First convergence theorem

Theorem: The algorithm described in Sec. C.1 always converges for data points x such that $|x| \leq \beta$, provided that

$$\begin{cases} \alpha \geq \frac{1}{2\cos\theta_{max}} \\ \beta = 2\cos\theta_{max} \end{cases} , \quad \text{if } \theta_{max} \leq \frac{\pi}{4}$$

$$\begin{cases} \alpha \geq \sin\theta_{max} \\ \beta = \frac{1}{\cos\theta_{max}} \end{cases} , \quad \text{if } \theta_{max} \geq \frac{\pi}{4}$$

Note that the conditions given here are sufficient, but not necessary, providing just an upper bound for α_{min}; the real α_{min} for a given codebook may be smaller. We will call this upper bound $\overline{\alpha}$.

The original proof was entirely algebraic. Here, we will present a geometric proof for this theorem; its consequences are more easily grasped this way.

Proof: For this, we will use a slightly modified version of the algorithm presented in C.1. The new version is as follows:

1. Take $r = x$.
2. Find v_k, the codevector nearest to r, i.e., the one which minimizes $|r - v_j|$. As all codevectors are of unit length, this is equivalent to find the codevector which minimizes $\text{ang}(r, v_j)$, the angle between r and v_j.
3. Take $\left(\frac{r - v_k}{\alpha}\right)$ as the next value of r.
4. If $|r|$ is small enough, stop; else, go back to step 2.

Observe that we modified rule #3 and took out the old rule #4. This amounts to scale up the residue at each step instead of working with successively smaller versions of the codebook. In other words, it is as if we were zooming in into the region containing the residue. Here, we say that the algorithm converges if $|r|$ remains bounded.

Now, suppose the data points to be coded are contained in a sphere of radius β. If after each step the residue (*after* scaling it up by $\frac{1}{\alpha}$) is also contained in this sphere, then we can iterate the loop endlessly, thus achieving whatever precision we want in the α-expansion.

Next, note that we must have $\beta > 1$. Else, for $x = 0$ we would have the first residue to have size $\left|\frac{x-v_k}{\alpha}\right| = \frac{1}{\alpha} > 1 > \beta$, which would contradict the above condition (i.e., the scaled-up residue would be outside the sphere).

Thus, given a codebook with a certain $\theta_{\max}$, we want to find the minimum $\alpha < 1$ such that

$$|x| < \beta \Rightarrow \left|\frac{x - v}{\alpha}\right| < \beta \tag{C.1}$$

for some value $\beta > 1$, where v is the codevector nearest to x, and so $\text{ang}\,(x, v) \leq \theta_{\max}$.

So, for any given x under such conditions, we can find the corresponding v. We can now work in the 2-D plane defined by $\overrightarrow{x}$ and $\overrightarrow{v}$; we can also think of it as the plane containing the points x, v and the origin. This is the plane depicted in Fig. C.4, where vector v is represented by the segment $\overline{OF}$, which has length 1.

For this development, we will use the following lemma.

Lemma: In the situation depicted in Fig. C.3, if points O and F are fixed and K varies so that angle $\widehat{KOF}$ is constant, $t = \overline{KF}$ and $s = \overline{KO}$, then the value $\frac{t}{s}$ will be minimum when $KF \perp OF$.

Proof: From sine law, $\frac{t}{\sin \hat{O}} = \frac{s}{\sin \hat{F}}$, which implies $\frac{t}{s} = \frac{\sin \hat{O}}{\sin \hat{F}}$. Since $\hat{O}$ is constant, this will be a minimum for maximum $\sin \hat{F}$, i.e., $\hat{F} = \frac{\pi}{2}$.

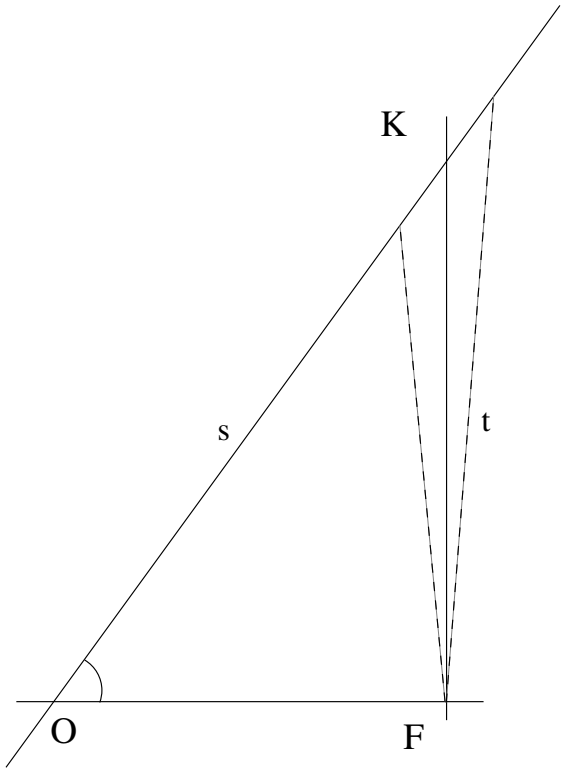


Figura C.3: Extreme $\frac{y}{x}$

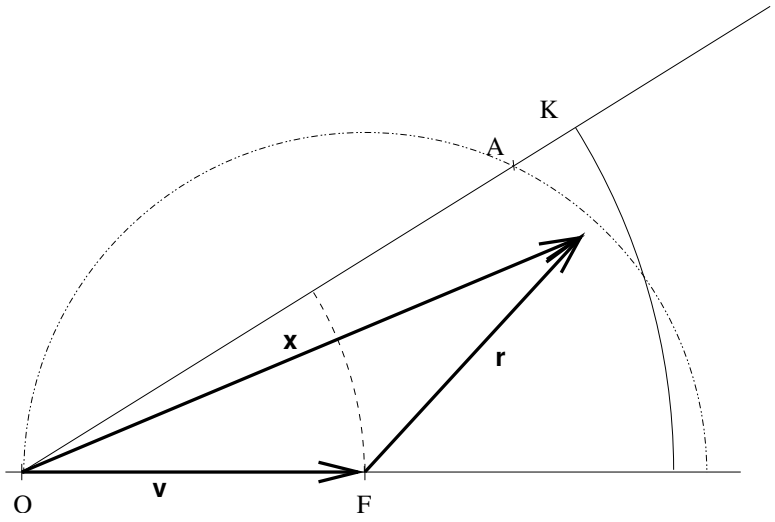


Figura C.4: Case 1

We can now proceed to the actual problem, which is to guarantee that Eq. C.1 holds. In the following figures, $\overline{OF}$ represents the unitary vector v_k, $\overline{OK}$ has length β, the angle $\widehat{KOF}$ is θ_{max} and $\overline{OA}$ has length $2\cos\theta_{max}$ (i.e., $\overline{AF}$ has length 1). We will consider several cases:

Case 1: $\theta_{max} \leq \frac{\pi}{4}$ and $\beta \geq 2\cos\theta_{max}$, as in Fig. C.4. Point x can be anywhere inside the sector depicted in the figure, as long as $\|x\| < \beta$. The residue r will be maximum when $r = \overline{KF}$. Since we must have $\frac{r}{\alpha} \geq \beta$, then $\alpha_{min} = \left(\frac{r}{\beta}\right)_{min}$. From the reasoning developed in the above lemma, $\frac{r}{\beta}$ grows monotonically for $\beta \geq 2\cos\theta_{max}$, so it will be minimum when $\beta = 2\cos\theta_{max}$, and so $\overline{\alpha} = \frac{1}{2\cos\theta_{max}}$.

Case 2: $\theta_{max} \leq \frac{\pi}{4}$ and $\beta \leq 2\cos\theta_{max}$, as in Fig. C.5. Here, r will be maximum when $x = 0$ and $r = 1$. Then, $\overline{\alpha} = \frac{1}{\beta_{max}} = \frac{1}{2\cos\theta_{max}}$.

So, for $\theta_{max} \leq \frac{\pi}{4}$ (cases **1** and **2**), $\overline{\alpha} = \frac{1}{2\cos\theta_{max}}$ and $\beta = 2\cos\theta_{max}$.

Case 3: $\theta_{max} \geq \frac{\pi}{4}$ and $\beta \geq 2\cos\theta_{max}$, as in Fig. C.6. Again, r will be a maximum when $r = \overline{KF}$. In this case, by the lemma above, $\frac{r}{\beta}$ will be minimum when $KF \perp OF$, so that $\beta = \frac{1}{\cos\theta_{max}}$ and $\overline{\alpha} = \frac{r}{\beta} = \sin\theta_{max}$.

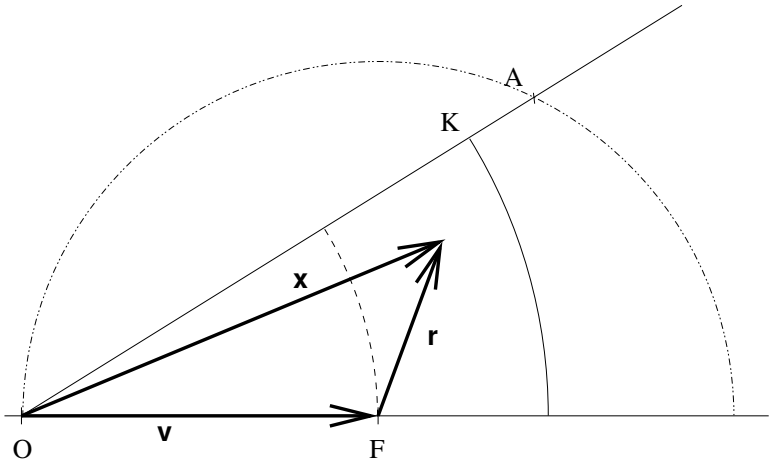


Figura C.5: Case 2

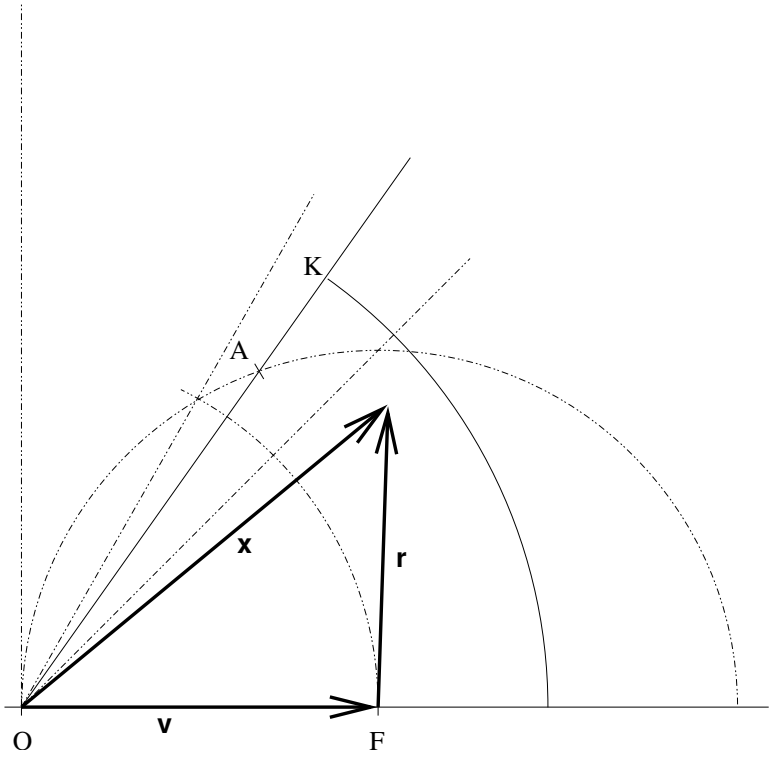


Figura C.6: Case 3

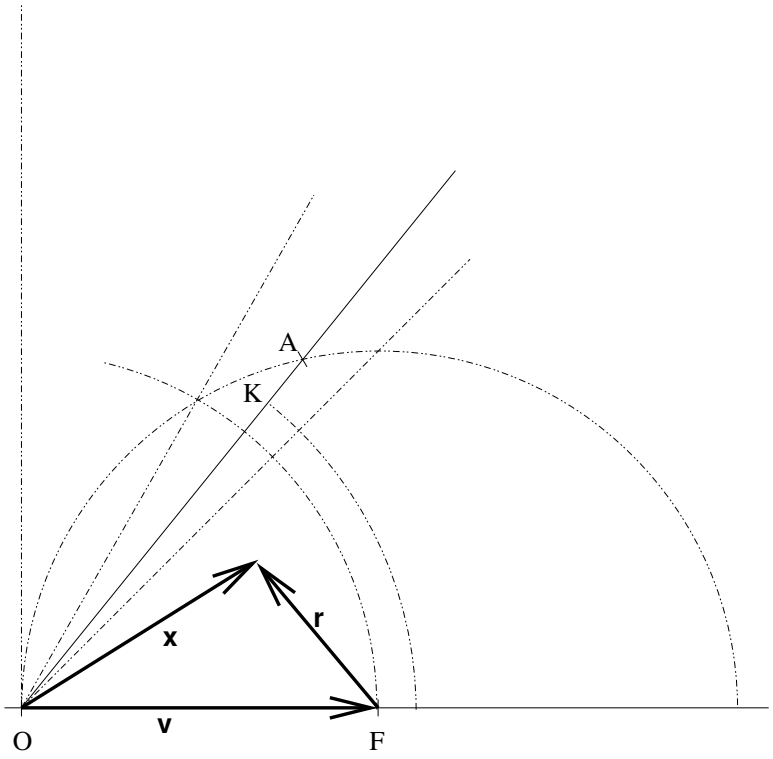


Figura C.7: Case 4

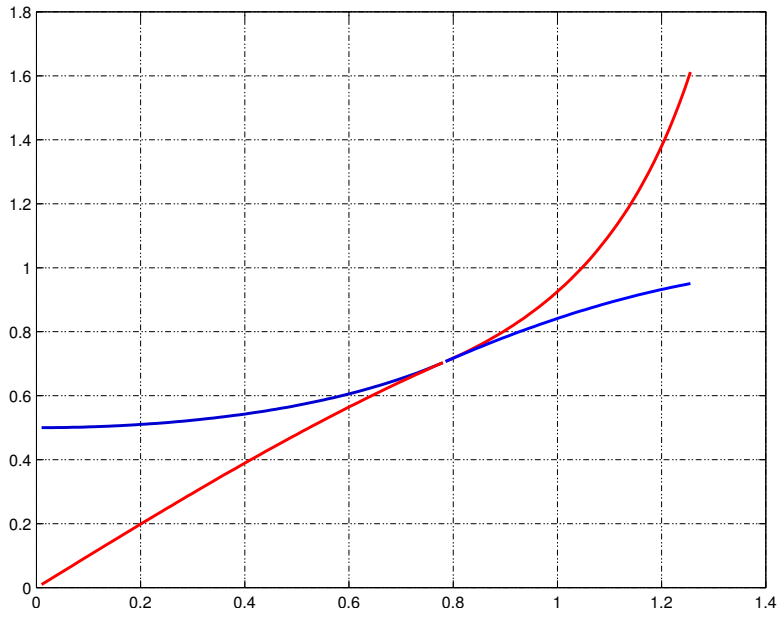


Figura C.8: $\overline{\alpha} \times \theta_{max}$

Case 4: $\theta_{max} \geq \frac{\pi}{4}$ and $\beta \leq 2\cos\theta_{max}$, as in Fig. C.7. This is only possible for $\theta_{max} \leq \frac{\pi}{3}$, else we would have $\beta \leq 1$. Now, the maximum possible r is again $\overline{KF}$. The maximum possible $\frac{r}{\beta}$ is when K is nearest the position where we would have $KF \perp OF$, i.e., $\beta = 2\cos\theta_{max}$, and so $\overline{\alpha} = \frac{1}{2\cos\theta_{max}}$.

For cases **3** and **4**, the smallest $\overline{\alpha}$ is $\overline{\alpha} = \sin\theta_{max}$, which happens when $\beta = \frac{1}{\cos\theta_{max}}$ (because $\sin\theta_{max} \leq \frac{1}{2\cos\theta_{max}}$ when $\frac{\pi}{4} \leq \theta_{max} \leq \frac{\pi}{3}$).

From the above, the theorem is proved. The blue line in Fig. C.8 depicts the variation of $\overline{\alpha} \times \theta_{max}$ for angles up to $\frac{\pi}{2.5}$. Observe, from the equations, that $\overline{\alpha}$ varies continuously for $0 < \theta_{max} < \frac{\pi}{2}$.

C.3.2 Observations on the theorem

The theorem stated above is weak with relation to several issues:

1. The theorem is based on a particular algorithm for choosing k. That algorithm is

not necessarily the best one in terms of minimizing α.

2. The theorem assumes the worst case on every iteration of the algorithm, i.e., that all residues will be approximated by a codevector at an angle θ_{max}. It does not imply that such a case is at all possible. Indeed, in practice convergence is generally achieved for values of α smaller than the upper bound suggested by the theorem.

3. The upper bound found here is not necessarily the smallest possible upper bound.

4. The theorem asserts convergence for all points inside the sphere of radius β. However, it does not assert that convergence is *only* for those points. Also, it does not point the region for which the algorithm does *not* converge. So, we still do not know what is the complete region of convergence, but only a subset of it.

Despite these minor shortcomings, that theorem does provide valuable insight into the convergence problem:

1. It shows that the convergence region can contain an open set (as opposed to a set of isolated points), and so shows that Successive Approximations is a feasible coding method.

2. Also, the reasoning behind it shows the general behavior of the convergence region with regard to α (it grows larger as α approaches 1).

3. Finally, it confirms an intuitive feeling: convergence can be faster, i.e., can use a smaller α, for more populated and more evenly spaced codebooks (those with smaller θ_{max}).

Apêndice D

The theory of α-expansions

> **"When I am working on a problem I never think about beauty. I only think about how to solve the problem. But when I have finished, if the solution is not beautiful, I know it is wrong."**
> **— Buckminster Fuller (1895-1983)**

We can now proceed to a more general theory of α-expansions. At this point, the objective is to define a generic coding method, able to encode vectors in $\mathbb{R}^N$, and then study the properties of this method with respect to convergence and performance.

This theory allows us to grasp what happens when we are doing an α-expansion, and we can arguably assert that it satisfies Mr. Fuller's requirement for correctness.

D.1 Establishing the goals

Most codings pose some assumptions on the form of the objects to be coded. In this case, we will assume that the objects we want to code are the vectors $x \in \mathbb{R}^N$ such that $|x| \leq 1$. In cases where this is not true, we just have to normalize the data involved. Given this restriction on x, we must find conditions on V and α to guarantee that any such x can be

represented as in Eq. A.1.

In Sec. C.1 we restricted the codebook to contain only unitary vectors. In the following development, no such restriction will be imposed.

Instead of looking directly for convergence conditions, let us try to look at the problem from another point of view: we may consider the set of all vectors that can be generated by the right side of Eq. A.1; then we can look for the conditions for this set to contain the unit ball on $\mathbb{R}^N$.

Let us consider a 2-D example. Let V be the set defined in Fig. D.1, where the 3 vectors

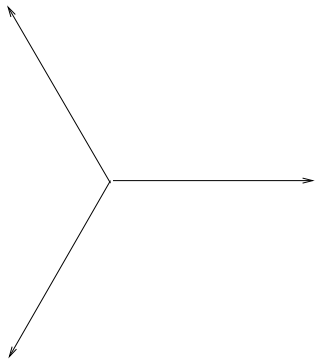


Figura D.1: A codebook V

are of unit length and evenly spaced on the circle. Let $\alpha = 0.5$. The possible values of x_1 are the codebook elements themselves:

$$x_1 = \begin{cases} v_1 \\ v_2 \\ v_3 \end{cases}$$

On the next iteration, we have 9 possible values for x_2:

$$x_2 = \begin{cases} v_1 \\ v_2 \\ v_3 \end{cases} + \begin{cases} \alpha v_1 \\ \alpha v_2 \\ \alpha v_3 \end{cases}$$

which can be seen in Fig. D.2 (a). As L grows, we can see that the set of possible points x_L is as depicted in Fig. D.2 (b).

(a) x_2 (b) x_L

Figura D.2: Possible values for x_i.

Now, this last figure looks a lot like a fully fledged fractal[1]. If we can prove that it is indeed a fractal (when $L \to \infty$), and that this happens for any codebook and scale factor α, we can then use fractal theory to verify under what conditions this fractal will contain the unit ball.

[1]Indeed, this set approximates the Sierpinski triangle [3], of fractal theory fame.

D.2 Iterated Function Systems (IFS)

Let us define x, k, V and α as in section A.3.1. Now define the functions

$$f_k(x) = \alpha x + v_k \tag{D.1}$$

and

$$F(A) = \bigcup_{1}^{M} f_k(A) \tag{D.2}$$

where A is any subset of $\mathbb{R}^N$ and $f_k(A) = \{f_k(a),\, a \in A\}$. Note that F is defined on the set[2] of the subsets of $\mathbb{R}^N$, and that f_k is extended to apply to elements of this set also.

From fractal theory [3], since each f_k is a *contraction mapping* (specifically for $0 < |\alpha| < 1$), it follows that the set $\{f_k\}$ forms an *iterated function system* (IFS). One of its properties is that

$$\exists |\, \Lambda \,|\, F(\Lambda) = \Lambda \tag{D.3}$$

where this set Λ (which is the unique fixed point of F) is called the *attractor* of the IFS.

Theorem: The set X of all points that can be represented by the right side of Eq. A.1, is the attractor of the corresponding IFS.

Proof: X contains *all* the points of the form

$$\begin{aligned} x &= v_{k_0} + \alpha v_{k_1} + \alpha^2 v_{k_2} + \ldots \\ &= v_{k_0} + \alpha(v_{k_1} + \alpha v_{k_2} + \ldots) \\ &= v_{k_0} + \alpha y \end{aligned}$$

where $y \in X$ and $\{y\} = X$. We can now partition this set according to the index of

[2]This set may become a *space*, if we define a proper *metric* for it.

the first term, k_0:

$$\begin{aligned} X &= \{v_1 + \alpha y\} \cup \{v_2 + \alpha y\} \cup \ldots \cup \{v_M + \alpha y\} \\ &= (v_1 + \alpha X) \cup (v_2 + \alpha X) \cup \ldots \cup (v_M + \alpha X) \\ &= f_1(X) \cup f_2(X) \cup \ldots \cup f_M(X) \\ &= F(X) \end{aligned}$$

Since $X = F(X)$, then $X = \Lambda$, the attractor of the IFS.

D.3 Fractals

We have already established that the points which can be represented as an α-expansion are those pertaining to the attractor Λ of the corresponding IFS.

IFS' attractors are (generally) fractals; they can also degenerate into "normal" (i.e., non-fractal) sets.

We must find the conditions under which the attractor will contain the unit ball.

D.4 Determining Λ

D.4.1 Geometric background

In the following discussion, we will use some readily available [26] geometric definitions and results. These are summarised here.

Combination: Let $a_1, \ldots a_k \in \mathbb{R}$, and $p_1, \ldots p_k \in \mathbb{R}^N$. The linear combination $x = \sum_{i=1}^{k} a_i p_i$ is said to be:

- a *positive combination*, iff $a_i \geq 0$, $\forall i \in (1, \ldots k)$;

- an *affine combination*, iff $\sum_{i=1}^{k} a_i = 1$;
- a *convex combination*, iff both previous conditions hold.

Convex sets: A set S is said to be convex iff for every $x, y \in S$, every convex combination of x and y is also in S.

Convex hull: The convex hull of a set S is the smallest convex set containing S.

Polytope: The convex hull of any finite set S is a polytope; conversely, every polytope is the convex hull of a certain subset of its points. The elements of this subset are called the vertices of the polytope. Polygons and (3-D) polyhedra are examples of polytopes. Note that infinite *sets* may have a convex hull which is not a polytope, e.g. a sphere.

Obs: The more generic definition of polyhedron includes unbounded sets defined as convex intersections of half-spaces, and polytopes as bounded polyhedra; for our purposes, however, we are considering only bounded sets.

Vertex: The vertices of a polytope can also be defined as those points which cannot be expressed as a convex combination of any other points of the polytope. A polytope is also the set of all convex combinations of its vertices. Observe that not all points in S (the finite set over which we build the convex hull) will necessarily be vertices; some may be in the interior of the polytope.

Homothety: An homothety of center C and scaling factor α is the transformation

$$\begin{aligned} h(x) &= C + \alpha(x - C) \\ &= \alpha x + (1 - \alpha)C \end{aligned}$$

We can also have $h(S) = \{h(s), s \in S\}$, i.e., extend h to apply to a set.

Homothety of a convex set: Let S be a convex set, $\alpha \in [0,1]$, $P \in S$, and $T = \alpha S + (1-\alpha)P$. Then $T \subset S$. Conversely, if $\alpha > 1$, then $T \supset S$.

D.4.2 The codebook's polytopes

Let P be the polytope defined by the convex hull of the vectors $\{v_k\}$. Likewise, let P_α be the convex hull of the set $W = \left\{w_k = \frac{1}{1-\alpha} v_k\right\}$. Observe that P_α and W can be obtained from P and V, respectively, by the same homothety with center at the origin and scaling factor $\frac{1}{1-\alpha}$.

By construction, any $x \in \Lambda$ is a positive combination of vectors from the codebook V, with coefficients whose sum is $\frac{1}{1-\alpha}$, thus a convex combination of the vectors w_k. Therefore,

$$\Lambda \subseteq P_\alpha \tag{D.4}$$

We can rewrite $f_k(x)$ as

$$\begin{aligned} f_k(x) &= \alpha\left(x - \frac{1}{1-\alpha} v_k\right) + \frac{1}{1-\alpha} v_k \\ &= \alpha(x - w_k) + w_k \end{aligned}$$

Therefore, each f_k is an homothety of center w_k and scaling factor α, which implies that $f_k(P_\alpha) \subset P_\alpha$, since (i) $0 < \alpha < 1$, (ii) $w_k \in P_\alpha$ and (iii) P_α is convex. This way, $F(P_\alpha)$ is the union of several scaled-down copies of P_α, each of them contained in P_α.

That said, two situations may arise: either

$$F(P_\alpha) = P_\alpha \Rightarrow \Lambda = P_\alpha$$

or

$$F(P_\alpha) \subset P_\alpha \Rightarrow \Lambda \subset P_\alpha$$

where $\subset$ is used here meaning "strictly contained". In the first case, all points inside P_α (and only these points) can be represented by the α-expansion; in the other, only a subset of P_α can be so represented. We must know the conditions under which we will have $\Lambda = P_\alpha$.

D.5 Bounds on α for $\Lambda = P_\alpha$

We will show that there is a critical value α_c such that $\Lambda_\alpha = P_\alpha \Leftrightarrow \alpha \geq \alpha_c$. In addition, we will show lower and upper bounds for α_c as functions of N and M — the codebook's dimension and cardinality.

D.5.1 Lemma: $(1-\alpha)f_k(P_\alpha)$ is monotonic

We can show that $(1-\alpha)f_k(P_\alpha)$ grows with α. In the following lines we will be referring simultaneously to two different IFS's (corresponding to the values α_1 and α_2). Thus, the respective sets and functions must be qualified by a subscript.

Consider the transformation

$$T_{k,\alpha}(x) = (1-\alpha)f_{k,\alpha}(\frac{x}{1-\alpha}) = (1-\alpha)(\alpha\frac{x}{1-\alpha} + v_k) =$$

$$\alpha x + (1-\alpha)v_k = \alpha(x - v_k) + v_k$$

Thus, this transformation is an homothety of center v_k and scaling α. Observe that it operates on P in the same way that f_k operates on P_α. Therefore, since P is convex and contains v_k,

$$\alpha_1 < \alpha_2 \Leftrightarrow T_{k,\alpha_1}(P) \subset T_{k,\alpha_2}(P) \Leftrightarrow$$

$$(1-\alpha_1) f_{k,\alpha_1}\left(\frac{P}{1-\alpha_1}\right) \subset (1-\alpha_2) f_{k,\alpha_2}\left(\frac{P}{1-\alpha_2}\right) \Leftrightarrow$$

$$(1-\alpha_1) f_{k,\alpha_1}(P_{\alpha_1}) \subset (1-\alpha_2) f_{k,\alpha_2}(P_{\alpha_2}) \tag{D.5}$$

which proves the lemma.

Now, suppose $F_{\alpha_1}(P_{\alpha_1}) = P_{\alpha_1}$ and $\alpha_2 > \alpha_1$. Using Eq. D.5 and the fact that $F(X) = \bigcup f_k(X)$, we can write

$$(1-\alpha_1) F_{\alpha_1}(P_{\alpha_1}) \subset (1-\alpha_2) F_{\alpha_2}(P_{\alpha_2}) \Leftrightarrow$$

$$(1-\alpha_1) P_{\alpha_1} \subset (1-\alpha_2) F_{\alpha_2}(P_{\alpha_2})$$

But $F_\alpha(P_\alpha) \subset P_\alpha$, so we can write

$$P = (1-\alpha_1) P_{\alpha_1} \subset (1-\alpha_2) F_{\alpha_2}(P_{\alpha_2}) \subset (1-\alpha_2) P_{\alpha_2} = P$$

Thus, $F_{\alpha_2}(P_{\alpha_2}) = P_{\alpha_2}$. To wrap it up, we have shown that

$$(\alpha_2 > \alpha_1) \text{ and } (\Lambda_{\alpha_1} = P_{\alpha_1}) \Rightarrow \Lambda_{\alpha_2} = P_{\alpha_2} \tag{D.6}$$

Let α_c be the smallest α for which $\Lambda = P_\alpha$; then, $\Lambda = P_\alpha \Leftrightarrow \alpha \geq \alpha_c$.

D.5.2 Lower limit for α_c

Let us consider the case $\alpha = 0$:

$$f_k(x) = \alpha x + v_k = v_k \Rightarrow$$

$$f_k(V) = v_k \Rightarrow$$

$$F(V) = V \Rightarrow$$

$$\Lambda = V$$

i.e., the attractor is just the codebook (i.e., the set $\{v_k\}$) — which is *not* P_α. So, we must have $\alpha_c > 0$. However, a stronger result can also be shown; a larger lower bound may be obtained as follows.

Theorem: $\Lambda = P_\alpha \Rightarrow \alpha \geq M^{(-\frac{1}{N})}$

Proof: Suppose $F(P_\alpha) = P_\alpha$. The set F is composed by the union of M smaller copies, which may or may not overlap each other. Let $m(P_\alpha)$ be the hyper-volume of P_α. Then, each component copy of F will have volume $m(f_k(P_\alpha)) = \alpha^N m(P_\alpha)$. Thus, we have that

$$M\alpha^N m(P_\alpha) \geq m(P_\alpha) \Rightarrow$$

$$\alpha \geq M^{(-\frac{1}{N})}$$

In fact, given only M and N, this is the largest possible lower bound for a generic codebook, since this value of α holds for the usual binary (or decimal, etc.) expansion. Indeed, for the binary expansion of vectors in $\mathbb{R}^N$, we have $M = 2^N$ vectors, yielding $\alpha \geq \frac{1}{2}$; convergence is obtained with any $\alpha \geq \alpha_c = \frac{1}{2}$.

D.5.3 The Simplex

Let us now consider the special case of codebooks with $(N + 1)$ elements and positive volume (i.e., there is no $(N - 1)$–dimensional hyper-plane containing them all) — i.e., those forming the (simplest) hyper-pyramids, rightly called *simplices*. Their 2- and 3-D representatives are the familiar triangle and triangular pyramid (tetrahedron). These polytopes have $(N + 1)$ vertices and the same number of hyper-faces; note also that the hyper-faces of a simplex are also simplices — but have one dimension less than the original (e.g., the faces of the tetrahedron are triangles).

Let us consider P_α and all its scaled-down copies $f_k(P_\alpha)$, as depicted in Fig. D.3 (The thin lines form P_α; the thick lines form the various $f_k(P_\alpha)$). Each copy shares 1 vertex and N faces with the original polytope. There remains one face, opposite to each shared vertex. These remaining faces, when extended (dashed lines in the figure), form another polytope which is similar to the original: each of its faces is parallel to a face of the main polytope. We could even appreciate the fact that its volume tends to be 2^N times the original volume, when α tends to zero. Let us call it the conjugate polytope P^c_α.

Observe that $\alpha \geq 0.5 \Rightarrow P^c_\alpha \subseteq P_\alpha$.

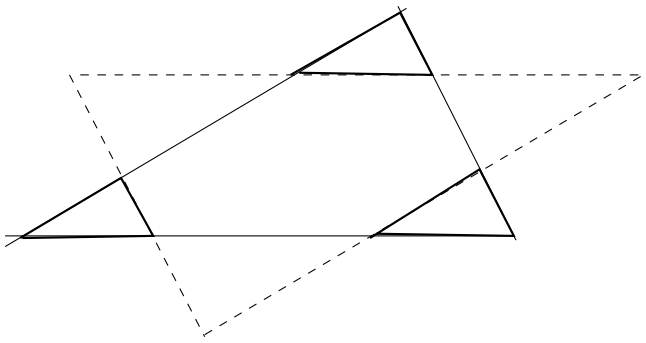


Figura D.3: A $(N + 1)$ polytope

Consider now the barycenter of P_α,

$$B = \frac{\sum v_k}{N+1}$$

and the line from one v_k to the barycenter, intersecting the opposite face at c_k, as in Fig. D.4. Let us create a coordinate system with one axis perpendicular to the face opposite

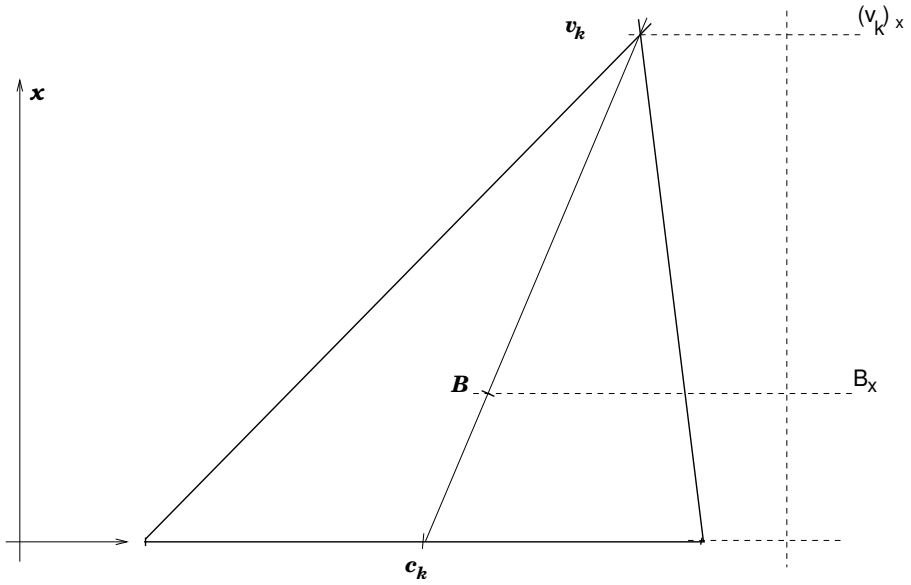


Figura D.4: The barycenter

v_k and with the origin on this face, so that all other vertices have $v_{i_x} = 0$. We then have $B_x = \frac{\sum v_{i_x}}{N+1} = \frac{v_{k_x}}{N+1}$, so that $\overline{v_k\, B} = \frac{N}{N+1}\overline{v_k\, c_k}$.

This implies that for $\alpha = \frac{N}{N+1}$, all the faces of the conjugate polytope meet at the barycenter, so that its volume reduces to zero. Note that

$$\alpha \leq \frac{N}{N+1} \Rightarrow F(P_\alpha) = P_\alpha - \mathrm{Interior}(P_\alpha^c)$$

where the minus sign denotes the set-theoretical subtraction. When $\alpha = \frac{N}{N+1}$, we have $P_\alpha^c = \{B\}$, and thus $F(P_\alpha) = P_\alpha = \Lambda$.

For any *smaller* α, the barycenter will be *outside* $\mathsf{F}(\mathsf{P}_\alpha)$, since it will not be contained in any $\mathsf{f}_\mathsf{k}(\mathsf{P}_\alpha)$. From Eq. D.6, we have that for hyper-pyramids

$$\alpha \geq \frac{\mathsf{N}}{\mathsf{N}+1} \Leftrightarrow \Lambda = \mathsf{P}_\alpha$$

A more algebraically inclined demonstration may be given using *barycentric coordinates*, as shown in Appendix L.

D.5.4 Upper limit for α_c

Let us consider now the generic codebook V. From Carathéodory's Fundamental Theorem [37], its convex hull can always be decomposed into several hyper-pyramids. For $\alpha \geq \frac{\mathsf{N}}{\mathsf{N}+1}$, each of these pyramids will be totally covered by its scaled-down copies, so that the complete P_α will be covered. Thus, also in this case we will have $\Lambda = \mathsf{P}_\alpha$.

Note that now we cannot guarantee that there will be a point lying outside all the $\mathsf{f}_\mathsf{k}(\mathsf{P}_\alpha)$ when $\alpha < \frac{\mathsf{N}}{\mathsf{N}+1}$; thus, for generic codebooks, we may possibly have $\Lambda = \mathsf{P}_\alpha$ for smaller values of α. However, this upper bound is effectively reached for all simplices. Thus, this is the smallest possible upper bound on α_c.

D.5.5 Critical value for α

From D.6 and from sections D.5.2 and D.5.4, we have that for any codebook V there will be a critical value α_c, $\mathsf{M}^{(-\frac{1}{\mathsf{N}})} < \alpha_\mathsf{c} \leq \frac{\mathsf{N}}{\mathsf{N}+1}$, such that $\alpha \geq \alpha_\mathsf{c} \Leftrightarrow \Lambda = \mathsf{P}_\alpha$.

Apêndice E

A geometric view

The theory developed throughout chapter D allows us to build a geometric interpretation for α-expansions and Successive Approximations.

E.1 The convex hull and $F(P_\alpha)$

Consider the polytope P_α associated with an α such that $\Lambda = P_\alpha$. In this case, any point inside the polytope can be represented as an α-expansion; this means that there exists (at least one) infinite sequence $(k_0, k_1, \ldots)$ such that

$$x = \sum_{i=0}^{\infty} \alpha^i \nu_{k_i}$$

Geometrically, the condition $F(P_\alpha) = P_\alpha$ means that the various $f_k(P_\alpha)$ must completely cover P_α. To better appreciate this condition Let us examine a situation where it *fails*, as in Fig. E.1. It represents a codebook with $N = 2$ and $M = 5$. Observe that P_α has only 4 vertices, for the 5th codevector is inside the convex hull of $\{\nu_k\}$.

Some characteristics of this figure are common to *every* codebook, for any value of α:

- The vertices of P_α are some of the points $\left\{w_k = \frac{1}{1-\alpha}\nu_k\right\}$, but possibly not all of them.

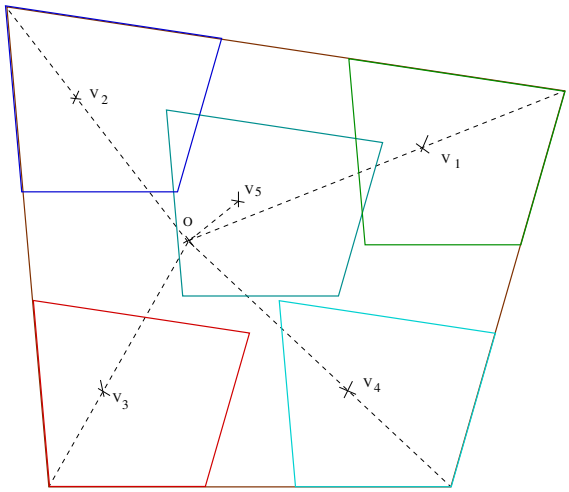


Figura E.1: 5-point codebook

- Each scaled-down copy $f_k(P_\alpha)$ can be obtained as an homothety centered at the origin and then translated to the codebook element v_k, or, alternatively, as an homothety centered at the respective w_k.
- All copies are entirely contained in P_α.
- The copy corresponding to each vertex of P_α shares this vertex and the corresponding faces with P_α.

In this case, $F(P_\alpha)$ (the union of the copies) *does not cover* P_α (there are points inside the larger polygon which is not contained in any of the smaller polygons).

In the next figure (E.2), $\alpha = \alpha_c = 0.55$, so that the covering is complete — but just so. In this figure, the codebook's elements are the vertices of the inner polygon. (In this case, $M = 4$).

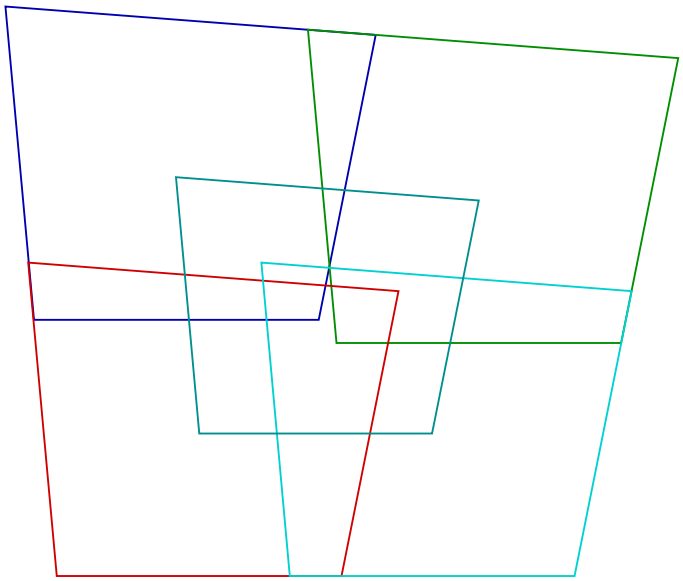


Figura E.2: A covered P_α

E.2 Determining the α-expansion coefficients

As defined in Eq. D.1, each f_k maps the entire P_α into a smaller copy of itself. Thus, we can write that

$$\forall k, \forall x \in f_k(P_\alpha) \Rightarrow \exists\, y \in P_\alpha \mid x = f_k(y)$$

Namely,

$$y = g_k(x) = \frac{x - v_k}{\alpha} \tag{E.1}$$

This way, every x in $f_k(P_\alpha)$ can be written in the form $x = v_k + \alpha y$. When $\Lambda = P_\alpha$, every point in P_α will be contained in at least one $f_k(P_\alpha)$; moreover, every y obtained as in Eq. D.1 will also be in P_α, so the expansion can always be continued:

$$x = v_{k_0} + \alpha y_0 = v_{k_0} + \alpha(v_{k_1} + \alpha y_1) = \ldots$$

Thus, we arrive at the rules which determine (one) α-expansion for x:

1. Determine k such that $f_k(P_\alpha) \subset x$ (there may exist more than one such k); $\Lambda = P_\alpha$ guarantees the existence of at least one such k.

2. Determine $y = g_k(x)$; $\Lambda = P_\alpha$ guarantees that $y \in P_\alpha$.

3. Take this y as the next x.

The repeated application of these steps will generate the desired sequence $(k_0, k_1, \ldots)$.

Each sequence element localizes x as being into one of the copies of the outer region; since this copy has the same form as the outer region, it can also be so divided, thus permitting the process to continue indefinitely. The regions so obtained are smaller and smaller, thus reducing the possible locations of x. In short, the k sequence defines a region in space where x is contained. Also, this same sequence can be used to find the approximation x_L (see Eq. A.2).

Whenever a point pertains to more than one $f_k(P_\alpha)$, any of the corresponding k can be selected, leading to different but equivalent representations (k sequences) for the same point (like "1" and "0.999…").

A very important conclusion may be drawn from the above discussion:

> *For any codebook,* α_c *depends only on its geometrical* form*, not on its size, orientation or position with relation to the origin.*

By *form* we mean the relative position of all elements, including those which may be inside the convex hull.

More formally, this comes from the fact that $x = f_k(y)$ is invariant under any combination of translation, homothety, reflection and/or rotation.

E.3 α-expansions and vector quantization

Vector quantization can be described as follows.

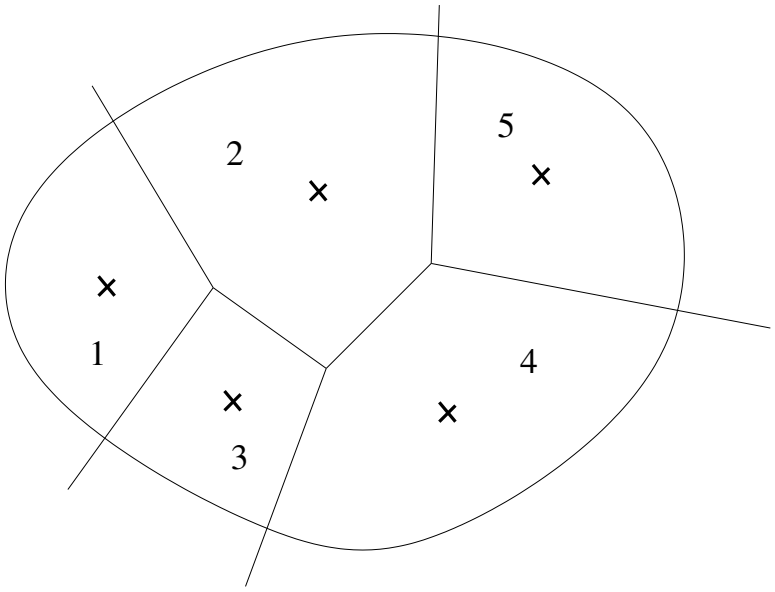


Figura E.3: Vector quantization

1. Divide the region containing the vectors to be quantized in sub-regions; index those sub-regions.

2. Each input vector is coded as the index of the sub-region where that vector is contained.

This process is illustrated in Fig. E.3. Once coded, the original input vectors can be recovered, to a certain extent, as one certain vector chosen from a dictionary; this choice is governed by the index. In the example illustrated here, those vectors are indicated as points inside each sub-region. Of course, given the desired number of subdivisions, there will be an optimum way to chose the regions and the vectors of the recovery dictionary. The optimality criterion is usually to minimize the mean squared error of the recovered points, but other applications could have other criteria.

Once the quantization scheme is defined, the data rate and the mean distortion are fixed. The data rate depends mainly on the number of sub-regions. The mean distortion depends basically on the size of the sub-regions and the location of the recovery points.

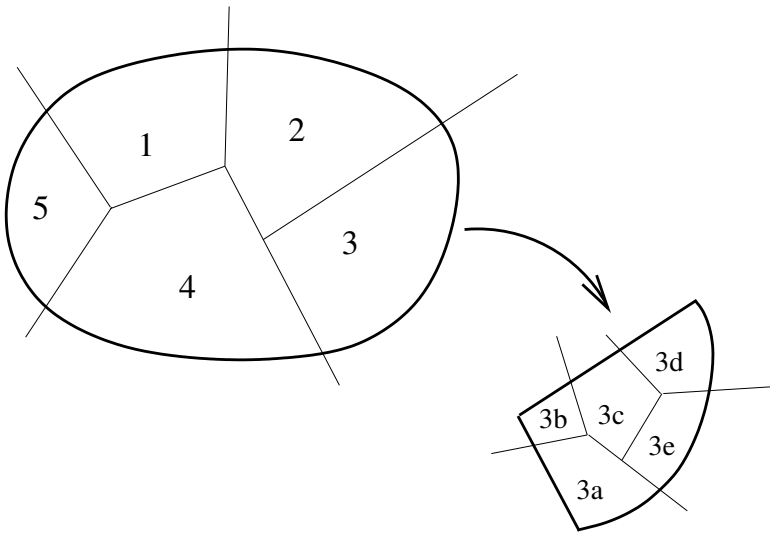


Figura E.4: Multi-stage quantizer

Both also depend on the probability density function for the input point distribution inside the overall region. If we want to change one of the $\mathsf{R} \times \mathsf{D}$ parameters, we need to change the vector quantizer.

Of course, once we locate an input point as being inside a certain sub-region, we could then repeat the whole process again — and again, and again, until we reach a desired distortion level. This procedure is illustrated in Fig. E.4, where an input point is first identified as being, say, in sub-region 3; then, this sub-region is further divided into smaller sub-regions, and a new quantization process is executed. Clearly, that may involve using several different quantizers, which would not be very practical.

Notice that we do not need the sub-regions to be a partition of the original region. That is, the union of the sub-regions must completely cover the original set, but they can overlap. Points inside an overlap may be coded as being in any of the overlapping sub-regions. The quantization scheme can still be used as is.

However, if we use overlapping regions, they are bigger then they needed to be, and the mean squared error (or almost any other distortion measure) will be larger. Assigning

parts of the overlapping regions to one or other sub-region would give us a smaller overall mean squared error.

On the other hand, the use of overlapping regions opens up the possibility of covering the *original region with smaller copies of itself*. If this is done, the same quantization process can be done again and again, on ever smaller regions, because each sub-region always has the same *form* of the original region. At each iteration, we localize the input vector with more precision. This way, we can reach any distortion requirements we want — with growing data rates, of course.

It should be obvious by now that α-expansions are exactly that: a recursive vector quantization, where the quantizer bins (the sub-regions) are possibly overlapping. Thus, α-expansions can also be seen as a special case of the (multistage) vector quantization process, where both the sub-regions and the recovery points are determined by the dictionary and the value α. The above reasoning is an alternative explanation for why we need the union of the $\mathsf{f}_k(\mathsf{P}_\alpha)$ to cover P_α, which is the "original region".

This point of view also brings into focus the relationship between α-expansions and fractals. The covering relates directly to the Collage Theorem [3], and can be seen as a novel approach to explain that theorem. Moreover, the "successive approximations" nature of α-expansions is clearly shown by the iterative choosing of ever smaller quantization bins.

The usual binary expansion, which can be seen as a special case of α-expansion, is particularly well suited because the square — and all its other-dimensional cousins — can be subdivided into smaller squares without overlapping. That gives these dictionaries some advantage regarding rate $\times$ distortion performance. The statistical distribution of the dataset, however, can tip the balance in favor of other dictionaries. These issues are treated in more detail in Sec. H.3 and H.5.

E.4 Self-similarity

The α-expansion process works on a region — a subset of $\mathbb{R}^N$ — that can be divided in smaller copies of itself. We have concluded that polytopes are such regions, but that does not mean they are the only kind. The attractors of the IFS's related to codebooks, as defined in Sect. D.2, also have this property, even when they do not contain an open set.

Other such regions are also possible, as the one suggested by Fig. E.5. In this case, the quasi-hexagon is divided in 7 smaller copies (one at the center); each copy is rotated 90° in relation to its "original". However, the very complexity of the set precludes its use as a basis for an α-expansion (see Sec. F.1).

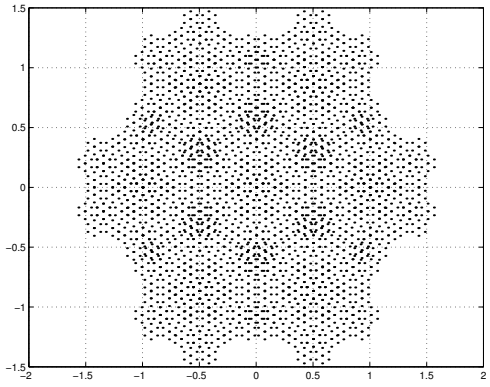


Figura E.5: Not a polygon

In all these cases, the original figure is similar to its constituent parts; each part, when "zoomed in", resembles the original one, thus creating an infinite self-similarity. This is a striking feature of many fractals.

Apêndice F

Algorithms

"Make everything as simple as possible, but not simpler."

— Albert Einstein (1879–1955)

The main step in the algorithm described in Sec. E.2 involves knowing whether a point is inside a polytope or not. This is so we can find which values of k satisfy $x \in f_k(P_\alpha)$.

Do not be deluded by the simplicity of the 2-D case; in general, this is a rather elusive problem which deserves some observations.

F.1 The containment problem

The problem discussed here can be stated very simply:

> Given a point x and a *convex* polytope P, defined as being the convex hull of the set $\{v_k\}$, does $x \in P$?

Unfortunately, there seems to be no feasible algorithm to solve it (i.e., one that runs in polynomial time). In the very definition of the problem lies one of its difficulties: we do not know the *faces* of the polytope, only a super-set of its *vertices* (for some of the codevectors

may even not be vertices of the convex hull — they can be *inside* the convex hull). Of course, the faces *can* be inferred from the vertices, but this is not trivial.

One way to solve this problem is to first find the convex hull of the vertices of the polytope. That means knowing which vertices form each face. This could be done taking every combination of N vertices and testing whether all other vertices are on or in the same side of the hyper-plane thus formed. Those hyper-planes which pass the test are the faces of the polytope. This would have to be done only once for a given codebook, since the polytopes are similar to P.

Once this is done, a point would be inside the polytope if it is on the same side as the other vertices for every face. Considering that

1. we may be working in large dimensions (16, 24, 32?),
2. the codebooks may have several thousand vectors, and
3. this procedure must be done once for each element of a k sequence, for each point in a data set,

this kind of solution is only of theoretical value. It would be impossibly time expensive to do it for real, even if in practical cases we only use codebooks whose convex hulls are known a priori.

We may put this problem in a different form, as follows.

Let us take a coordinate system whose origin is the point x, and transform all v_k accordingly. Our problem is now:

> Does the (new) convex polytope P contain the origin?

Algebraically, this can be stated as:

> Is there a set of non-negative coefficients $\{b_i\}$ such that $\sum_i b_i v_i = 0$?

We can discard the trivial case where x is one of the v_k. Thus, knowing that all $|v_i| > 0$, we can write

$$\begin{aligned}
\sum_i b_i v_i &= 0 \Leftrightarrow \\
\sum_i b_i \frac{|v_i|}{|v_i|} v_i &= 0 \Leftrightarrow \\
\sum_i b_i |v_i| \frac{v_i}{|v_i|} &= 0 \Leftrightarrow \\
\sum_i c_i u_i &= 0,\ c_i \geq 0
\end{aligned}$$

That is, we can normalize the vectors v_k and ask whether the resulting convex polytope contains the origin. These transformations are depicted in Fig. F.1. Observe that the

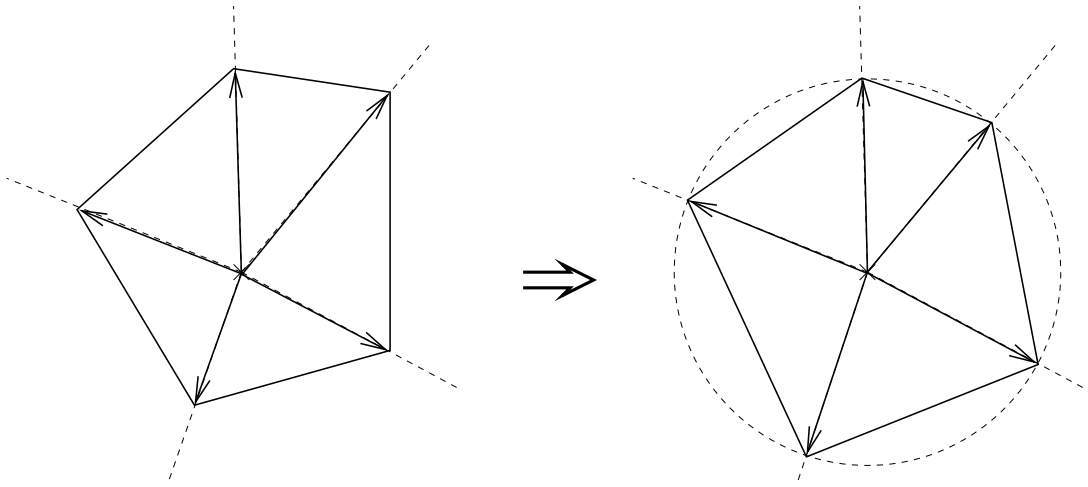


Figura F.1: Normalizing the vertices of the polytope

angles between the vectors do not change.

If we further suppose that the point in question (which is now the origin) is inside the polytope — not on its frontier — then there is a point on the frontier which is closest to the origin. Let us call this smallest distance δ. Then every point inside the sphere of radius $\frac{\delta}{2}$ is inside the polytope, meaning that for these points we can write $x = \sum_i d_i u_i$, where

all $d_i \geq 0$. Than, we can scale those d_i to obtain any point in space. Thus, the final form of the problem can be stated as

> Does a given set of (normalized) vectors generate all space with only non-negative coefficients?

This form of the problem is probably more adequate to be solved by matrix theory. However, in any of the above stated forms, the problem seems to grow as C_M^N, which makes it computationally unfeasible for most cases.

F.1.1 A simple case

However intractable the problem may be in the general case, it has a remarkably simple solution in the case of the triangle. For 3 co-planar, unitary vectors, it can be easily shown that they generate the space if and only if $|v_1 + v_2 + v_3| < 1$.

F.2 The greedy algorithm

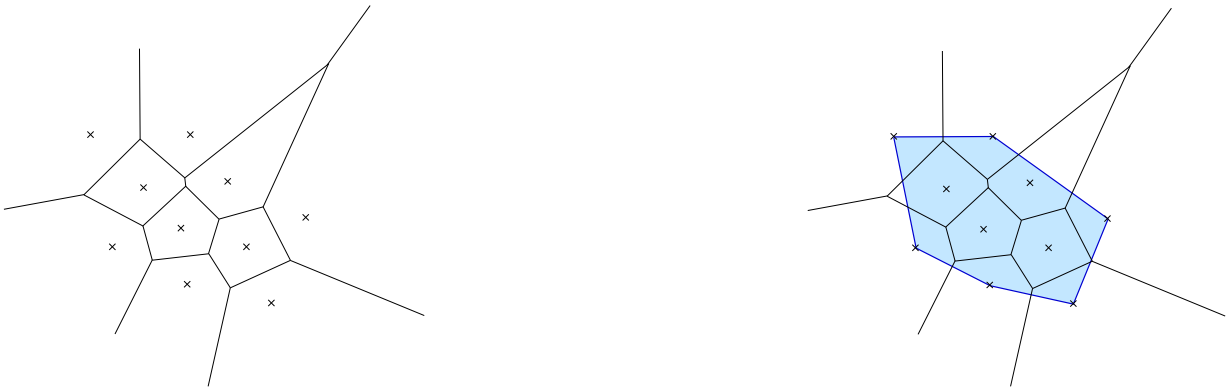
The difficulties associated with the basic algorithm prompt the search for simpler and faster algorithms. We can now review the greedy algorithm of Sec. C.1 or its modified version of Sec. C.3.1.

In both cases, the next element in the coding sequence is chosen by searching for the codevector nearest the residue of the previous iteration. This will be the "right" decision if and only if the corresponding $f_k(P_\alpha)$ contains the point in question. If we want this to be always true, we must have $f_k(P_\alpha)$ to contain all points which have v_k as its nearest codevector. We now need the concept of the *Voronoi cell*.

F.2.1 The Voronoi cell

Given a set $S = \{s_i \in \mathbb{R}^N\}$, we define the Voronoi cell of point s_i as being the subset $V_i = \{x \in \mathbb{R}^N\}$ of points which are at least as close to s_i as to any other s_j [8, p. 33]. In other words,

$$x \in V_i \Leftrightarrow d(x, s_i) \leq d(x, s_j), \forall j$$



(a) Points and cells

(b) Convex hull

Figura F.2: Voronoi cells

As an example, Fig. F.2 (a) depicts the Voronoi cells of a 2-D set of points. For those points which are inside the convex hull of S, the cells are convex polytopes. For those which are on the surface of the convex hull, the cells are the union of an infinite cone or cylinder with a convex polytope, always forming an infinite convex region. When referring to Voronoi cells related to codebooks, we mean the intersection of the "real" Voronoi cells of the set $\{w_k = \frac{v_k}{1-\alpha}\}$ with its convex hull, as depicted in Fig. F.2 (b).

F.2.2 Convergence of the greedy algorithm

We can now state the conditions under which we will have convergence of the greedy algorithm, i.e., we must have both

$$\mathsf{F}(\mathsf{P}_\alpha) = \mathsf{P}_\alpha \tag{F.1}$$

and

$$\mathsf{f}_\mathsf{k}(\mathsf{P}_\alpha) \supseteq \left\{\mathsf{V}_\mathsf{k} \bigcap \mathsf{P}_\alpha\right\} \tag{F.2}$$

where V_k is the Voronoi cell of w_k. In fact, the second condition implies the first, since

$$\bigcup_\mathsf{k} \left\{\mathsf{V}_\mathsf{k} \bigcap \mathsf{P}_\alpha\right\} = \mathsf{P}_\alpha$$

This way, selecting the w_k nearest to x implies that $\mathsf{x} \in \{\mathsf{V}_\mathsf{k} \bigcap \mathsf{P}_\alpha\}$, thus $\mathsf{x} \in \mathsf{f}_\mathsf{k}(\mathsf{P}_\alpha)$; this is sufficient for the algorithm to converge.

Observe that the minimum α needed to obtain Eq. F.2 is at least as large as the critical value α_c defined in D.5.5, needed for convergence of the basic algorithm. So, in order to use the greedy algorithm *and* have the minimum possible α, we must look for codebooks where Eq. F.1 implies the relationship defined by Eq. F.2.

Apêndice G

Binary expansions revisited

We can now show that the familiar representation of numbers as binary expansions is a particular case of Successive Approximations.

G.1 Binary representation of scalars

The binary representation of a number in the interval $[0, 1]$ is a sequence $(b_1, b_2, \ldots)$ whose value is

$$x = \sum_{i=1}^{\infty} b_i \left(\frac{1}{2}\right)^i, b_i \in \{0, 1\}$$

which can also be written as

$$x = \sum_{i=0}^{\infty} b_i \left(\frac{1}{2}\right)^i, b_i \in \left\{0, \frac{1}{2}\right\}$$

In this form, it can immediately be recognized as an α-expansion.

Consider the codebook $V = \left\{v_0 = 0, v_1 = \frac{1}{2}\right\}$, and $\alpha = \frac{1}{2}$. This leads to P_α being the closed interval $[0, 1]$, as depicted in Fig. G.1. Since P_α is totally covered by its two halves, we have $\Lambda = P_\alpha$. Thus, any number in this interval can be represented as an α-expansion

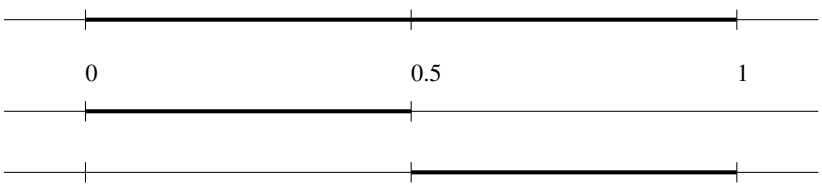


Figura G.1: The [0,1] interval

of this codebook. Take, say, $x = 0.1011\ldots$:

$$x = v_1 + \frac{1}{2}(v_0 + \frac{1}{2}(v_1 + \frac{1}{2}(v_1 + \ldots))) = \left(\frac{1}{2}\right)^0 v_1 + \left(\frac{1}{2}\right)^2 v_1 + \left(\frac{1}{2}\right)^3 v_1 + \ldots$$

The two half-segments intersect at $x = 0.5$; that means that this point can have more than one representation. Indeed, $0.1000\ldots = 0.0111\ldots$. Note also that the point $x = 1$, contained in P_α, *can* be represented as $0.1111\ldots$.

G.2 Binary representation of vectors

Similarly to the scalar case, any vector in the unit hyper-cube $[0,1]^N$ can be coded as an α-expansion over the codebook $V = \left\{[0, \frac{1}{2}]^N\right\}$ (i.e., a 2^N–element set of N–dimensional vectors whose coordinates are either 0 or $\frac{1}{2}$). The 2-D case is illustrated in Fig. G.2.

Any hyper-cube can be covered by 2^Ncopies of itself, where the side of each copy is $\frac{1}{2}$ of the original. Although there are non-empty intersections, the total hyper-volume of these intersections is null.

G.3 Other usual representations

The arguments presented above are not limited to base-2 representations; they can also be made for the usual base-10 numbering system, or for any other integer base. Take, for

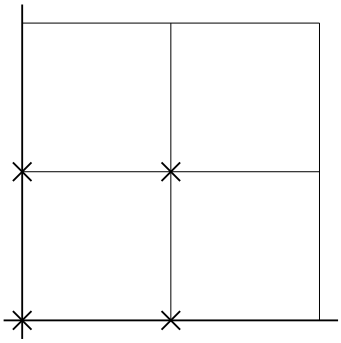


Figura G.2: 2-D binary expansion

example, 2-D vectors represented in base 3; the corresponding codebook is illustrated in Fig. G.3.

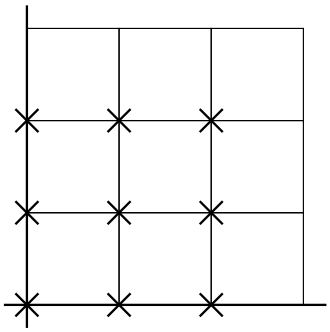


Figura G.3: Base-3 expansion

It is interesting to notice that in this case we have $\alpha = \frac{1}{3}$. To have $\Lambda = \mathsf{P}_\alpha$ with $\alpha < \frac{1}{2}$, additional codebook elements must exist at the sides of the convex hull.

G.4 SA as a general case

As seen in the preceding sections, the usual representations of numbers — either scalar or vector — are but particular cases of α-expansions.

Although these particular cases present some definite advantages (fast computation, for instance), we may ask whether they are optimal from the perspective of other criteria. We are specifically interested in the rate x distortion characteristics of codings. This prompts us to study these characteristics for the general case of α-expansions, in an attempt to find a codebook with better performance than the hyper-cube. This is done in the next chapter.

Apêndice H

Experimental results

Since α-expansions are a generalization of the uniform scalar quantizer, we wonder whether we can find a codebook with a better rate $\times$ distortion performance than that of the particular case. After all, it would result in a moot theory indeed if that were not the case.

Throughout this chapter we will deal with this problem. The objective is to compare the rate $\times$ distortion of the α-expansion over diverse codebooks.

We will show that, for high rates, the best codebook for α-expansion is the one equivalent to uniform scalar quantization (see Sec. G.1). However, in low bit-rate image transform coding, we have a majority of small magnitude coefficients; in this case, α-expansions using other multidimensional codebooks can be a better choice, provided that an efficient coding method for the zero values is devised (see Sec B.1 and B.3.1).

In order to compare different codebooks we must:

1. Define one or more test datasets – the sets of values to be coded. These datasets will be normalized: their largest vector will have norm $\mathbf{1}$. We do not loose any generality with that; normalization is necessary for any practical coder.

2. Adapt (scale) the codebooks to the datasets. Also, the codebooks' parameters must be found.

3. Run the coding algorithm and measure rates and distortions.

H.1 The Algorithm

The algorithm used in the experiments is similar to that of Sec. C.3.1, but we are not restricted to unit codevectors anymore. Thus, we must select the nearest codevector using the distance to the data point, and not the angle between them. Also, for rate & distortion measurements, we stop the loop after a predetermined number Q of iterations:

1. Take $r = x$.
2. Repeat for R times:
   (a) Find v_k, the codevector nearest to r, i.e., the one which minimizes $|r - v_j|$.
   (b) Take $\left(\frac{r - v_k}{\alpha}\right)$ as the next value of r.

From $|r|$ we can calculate the distortion at each step.

H.2 Codebook adaptation

In order to be able to code any given point from a dataset using α-expansion over a codebook V, we must have:

1. $\alpha \geq \alpha_c$
2. The dataset must be contained in P_α.

The value α_c, as we have seen in Sec. E.2, depends only on the form of the codebook (not its scale), and can be found experimentally.

To ascertain item 2, however, we have 2 degrees of freedom. We could use a larger α, what also increases the size of P_α — which is proportional to $\frac{1}{1-\alpha}$ (see Sec. D.4.2). But a larger α would imply slower convergence: the average total distortion decreases as $C\alpha^n$. So, in the interests of a good performance, we will want to use $\alpha = \alpha_c$. The other option to get a larger P_α is to scale the codebook itself, since P_α is an homothety of the codebook's convex hull.

Let γ be the scale factor needed to make P_α large enough to contain a given dataset. Of course, this value will depend on that particular dataset. However, a practical coder would need to use a unique γ, valid for any dataset. Since the datasets are normalized, it is sufficient for P_α to contain the unit sphere; that guarantees that it will contain any normalized dataset. Let γ_c be the smallest value such that P_{α_c} contains the unit sphere.

For practical reasons, γ_c will always be specified relative to a normalized codebook — one whose largest codevector has length 1.

With proper values of α and γ, we can guarantee that $r \in P_\alpha \Rightarrow \left(\frac{r-v_k}{\alpha}\right) \in P_\alpha$. This fact can be used to experimentally find both α_c and γ_c. Basically, we run the algorithm described in Sec. H.1 for a large number of data points, while controlling the error behavior; if at any moment the error magnitude grows larger than γ, either one or both parameters are smaller than the needed critical values. For some simple codebooks, these parameters can be determined theoretically. When the smallest possible values for α and γ are found, we can proceed to performance measurements.

It is worthwhile to notice that using the smallest possible γ is also important from a performance point of view; larger values of γ would give rise to larger distortion.

H.3 Rate and distortion measurements

The procedure used to compare codebooks' performances was to calculate the average MSE for the α-expansions of large numbers of data points. This was done implementing the greedy algorithm of Sec. H.1.

The results depend strongly on the spatial distribution of the datasets used. The datasets result from the clustering of coefficients into vectors of the appropriate dimension, i.e., that of the codebook. Two kinds of datasets were used: uniformly distributed (in space) points inside the N-dimensional sphere of radius 1, and vectors generated by the clustering of actual wavelet transforms of natural images.

Remember that each coded vector represents N coefficients from the dataset. In order to properly compare codebooks with possibly different dimensions, we must determine the per-coefficient distortion and rate.

Clearly, both distortion and rate are dependent on the number of iterations, n, through the algorithm. At each step, the average total distortion decreases (since $|r_n|$ is bounded by $C\alpha^n$).

The rate depends on the statistical distribution of the symbols found — the k-sequence. However, practical codebooks will be isotropic, and we can expect that all possible values of k will appear with the same probability. In this case, the total rate increases simply as $R_T = n.\log_2(M)$ bits per symbol, where M is the cardinality of the codebook.

Thus, we have that the per-coefficient rate, in bits, is $R = \frac{n.\log_2(M)}{N} = n.\log_2(M^{\frac{1}{N}})$.

Naturally, the actual total distortion depends on the spatial distribution of the data set; however, the asymptotic behavior of $|r_n|$ depends only on the form of the codebook: for large n, we will have that $|\bar{r}_n| = C_0\alpha^n$, where C_0 is some positive constant. Thus, for the total distortion, we will have $D_T = C_0^2\alpha^{2n}$; the per-coefficient average distortion will

be $D = \frac{C_0^2}{N}\alpha^{2n}$. Average per-coefficient distortion as a function of rate will thus be

$$D(R) = \frac{C_0^2}{N}\left(\alpha^{\frac{1}{\log_2\left(M^{\frac{1}{N}}\right)}}\right)^{2R} \tag{H.1}$$

So, to minimize the distortion for large rates, we must have the minimum possible $\alpha^{\frac{1}{\log_2\left(M^{\frac{1}{N}}\right)}}$. Since we know that $\alpha \geq M^{-\frac{1}{N}}$ (see Sec. D.5), we may write $M^{\frac{1}{N}} = \frac{k}{\alpha} \geq \frac{1}{\alpha}$, with $k \geq 1$. Analyzing the function $\alpha^{\frac{1}{\log_2\left(\frac{k}{\alpha}\right)}} = 2^{\frac{\log_2 \alpha}{\log_2 k - \log_2 \alpha}}$, given the restriction $k \geq 1$, we find its minimum at $k = 1$. In this case, from Eq. H.1, $D(R) = \frac{C_0^2}{N}2^{-2R}$. For the codebooks corresponding to the binary expansion, $M = 2^N$ and $\alpha = \frac{1}{2}$, hence $k = 1$. Thus, we conclude that for *large rates*, these codebooks are optimal for α-expansions. Since $M^{\frac{1}{N}} = 2$, the rate $\times$ distortion performances of these codebooks do not change with their dimension, so in this case scalar and vector quantization are equivalent.

H.4 Experiment setup

For the experiments described below, we determined the resulting average distortion D (MSE) in decibels, that is, $10\log_{10}(D)$, and the accumulated data rate R in bits, up to 20 iterations of the α-expansion algorithm. Details about the codebooks and datasets are given below.

H.4.1 Codebook generation

As we have seen, convergence will be faster for smaller α. Also, for given M and N, codebooks whose vectors are more uniformly spread in $\mathbb{R}^N$ tend to have smaller α_c. Note that we have not defined "spread" here; we are using the term in a somewhat intuitive sense. The problem reminisces that of sphere packings, several variations of which are tackled by Conway & Sloane in [8]; some related results can also be found in [35, 36].

For these experiments, we have used a few classes of codebooks; we sometimes refer to them by the name of the polytopes whose vertices we are using as codevectors:

- The binary expansion–equivalent codebook, on several different dimensions: square, cube, tesseract, and so on (including the 1-dimensional "square", equivalent to uniform scalar quantization); we refer to them as the "binary codebooks".
- Some 2-D regular polygons;
- Shells of some best known sphere packings [8]. For example, D4 is the 4–D codebook with the 24 vectors of the first shell of the best known sphere packing in 4–D.

Most modern methods use uniform scalar quantizers, except for the region near the origin, where a double-sized quantizer bin is used. The reasoning behind this behavior is that most of the points to be coded have small magnitude; the oversized bin near the origin helps to net still more points to be coded as zeros. Remember that most of the significant information about the image is concentrated on the large magnitude coefficients. Accordingly, we tested the codebooks listed above with an additional zero-vector as a codevector.

H.4.2 Dataset generation

The datasets used for the experiments were of three kinds:

- Uniformly distributed (in volume) inside the N–dimensional sphere of radius 1; those were mostly used for the convergence tests to find suitable values for α and γ, but also for performance comparison of codebooks.
- Laplace-distributed coefficients. Those resemble more accurately the datasets obtained by clustering the coefficients of wavelet transforms of natural images.
- Clusterings of the coefficients of real wavelet transforms.

All datasets were normalized so that $|\mathsf{x}|_{\max} = 1$.

H.4.3 The coding of the zeros

Usually, in image and video compression, zeros and other values are coded with different methods, in order to take advantage of the overwhelming majority of zeros generated by the quantization stage; that is why zerotrees and the like lend those methods such good performances (see Sec. B.1 and B.3.1). So, the resulting bitstream is composed basically of symbols which represent the non-zero values. In order to simulate this behavior, in most experiments we have considered only the rate due to the coding of non-zero values. This is justified because we are assuming that an efficient method will be used for encoding the zeros, resulting in negligible added rate. In the results that follow, the case where the zeros are coded in the same way as other values is shown for comparison; this is indicated as "FR" (full rate) in the graphs.

H.5 Comparison of codebook performances

The performance of the uniform scalar quantizer on the uniformly distributed dataset is taken as baseline for comparison; it is seen as the dark–blue, straight line on the graphs.

The graph shown in Fig. H.1 depicts the $\mathsf{D} \times \mathsf{R}$ curves for the following cases:

2D FR (unif): A uniformly distributed dataset, coded with the 2D binary codebook; full rate is shown.

2D FR (lapl): Ditto, for a Laplace-distributed dataset.

2D+Z: Laplace-distributed dataset, 2D binary codebook plus the origin as a codevector.

2D+Z FR: Ditto, full rate shown.

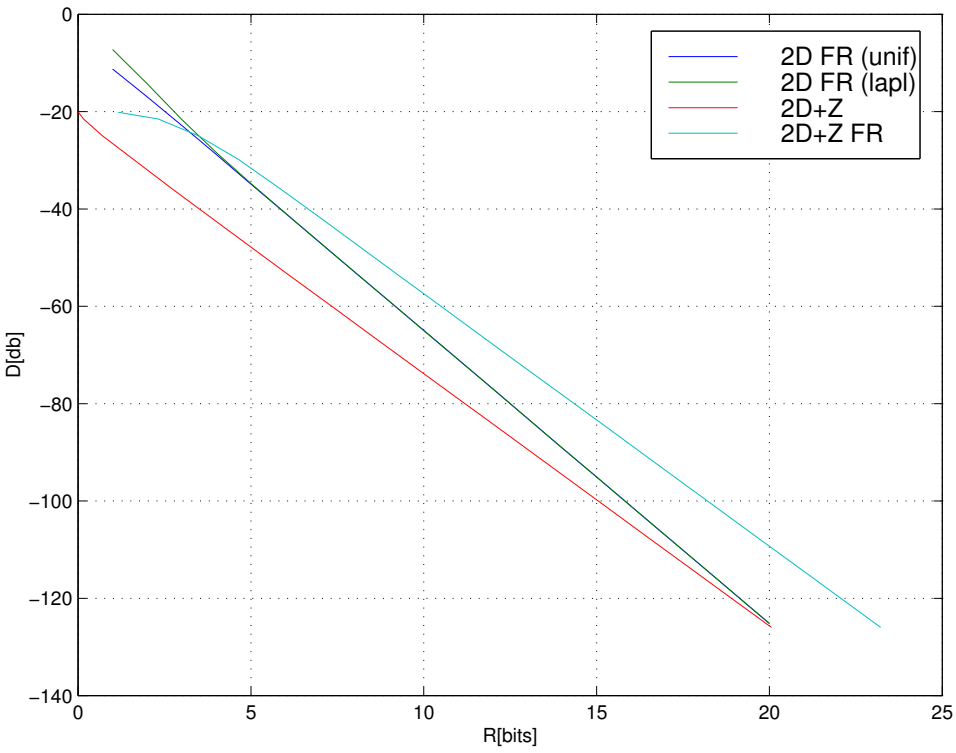


Figura H.1: Graph I: Results from 2D binary codebook

Some interesting features can be perceived from the graph:

- The binary codebook, that is equivalent to scalar uniform quantization, has the best asymptotic performance; this is in accord with the results from Sec. H.3.
- The statistical distribution of the dataset influences the results for the first steps, but the codebook dictates the performance as the residues tend to a distribution which depends on the codebook; that is why the curves for the uniform and laplacian distributions merge for high rates.
- The performance improvement obtained by the efficient encoding of zeros is clearly shown by the 3rd and 4th curves ("2D+Z" and "2D+Z FR").

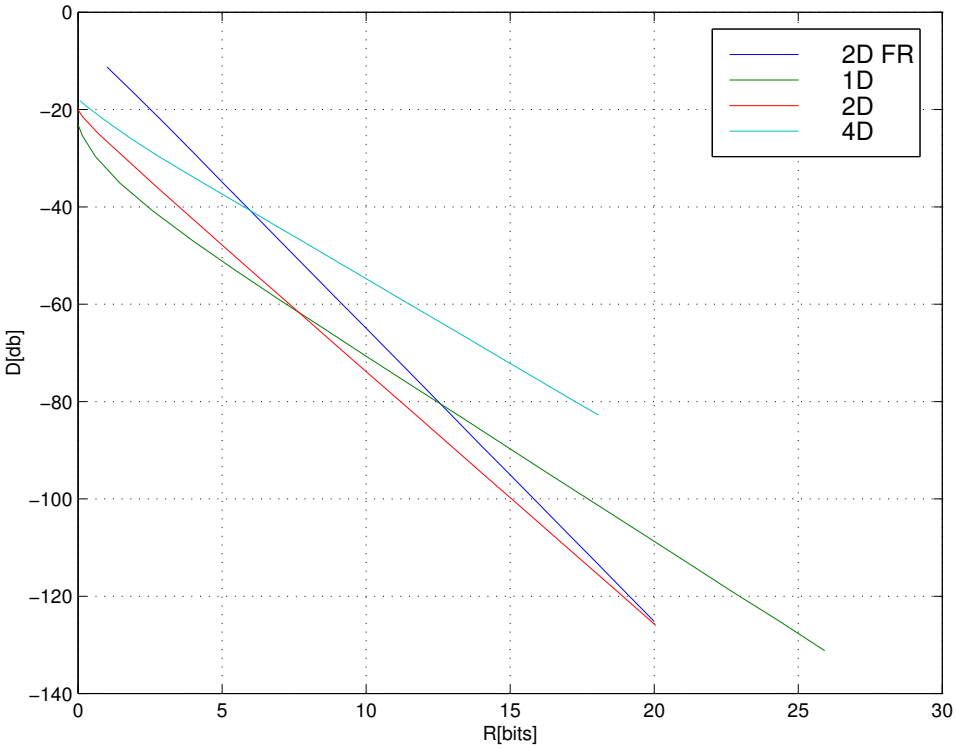


Figura H.2: Graph II: Binary codebooks on laplacian datasets

Graph II (Fig. H.2) depicts the $\mathsf{D} \times \mathsf{R}$ curves for the 1–, 2– and 4–D binary codebooks (plus the origin), applied to a Laplace–distributed dataset. We note that for laplacian

distributions, even in the binary case, the codebook's dimension may have a strong influence on the $\mathsf{D} \times \mathsf{R}$ performance.

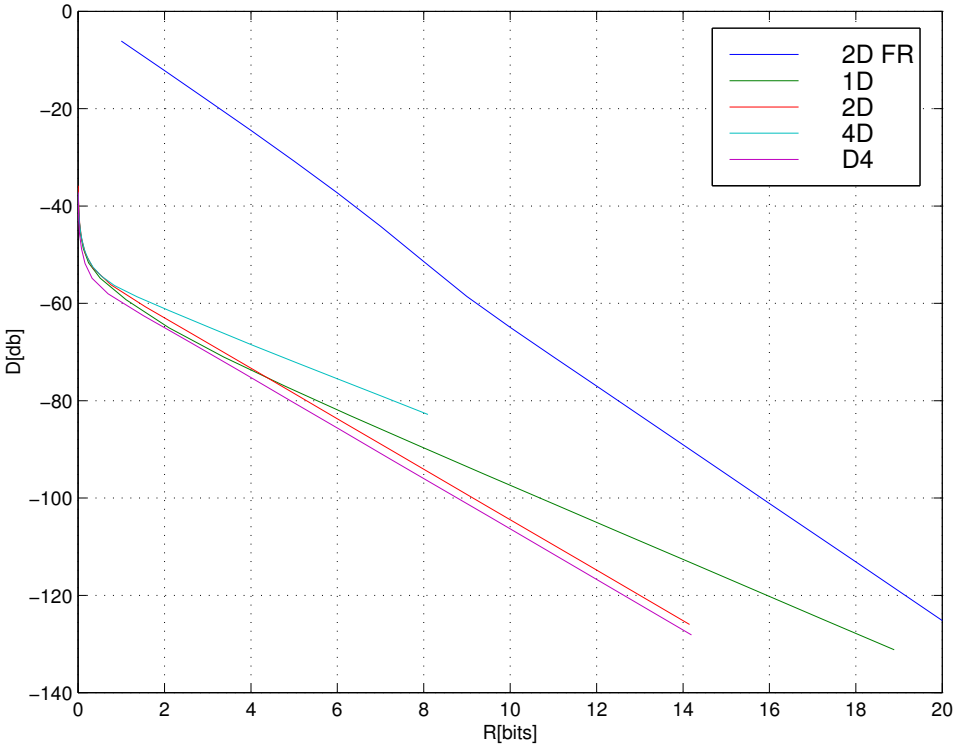


Figura H.3: Graph III: D4 x binary codebooks

Graph III (Fig. H.3) shows the results obtained on a dataset composed by the wavelet coefficients of the transformed Lena (512×512) image (a classical image in this field of research). Graph IV (H.4) shows the same data in more detail, zooming around the low rate region. Besides the same binary codebooks as before, the 4-D binary and the D4 codebooks (plus the origin) are also used. The results are similar to those obtained on the Laplace–distributed dataset, but the coefficient distribution here is much more concentrated around the origin, so the advantages of including the zero-vector are much more pronounced. As can be seen from the graphs, the D4 codebook has a very good performance on a large range of data rates. Thus, although the binary codebooks present the best asymptotic behavior, this shows that for practical data rates other kinds of codebooks can outperform

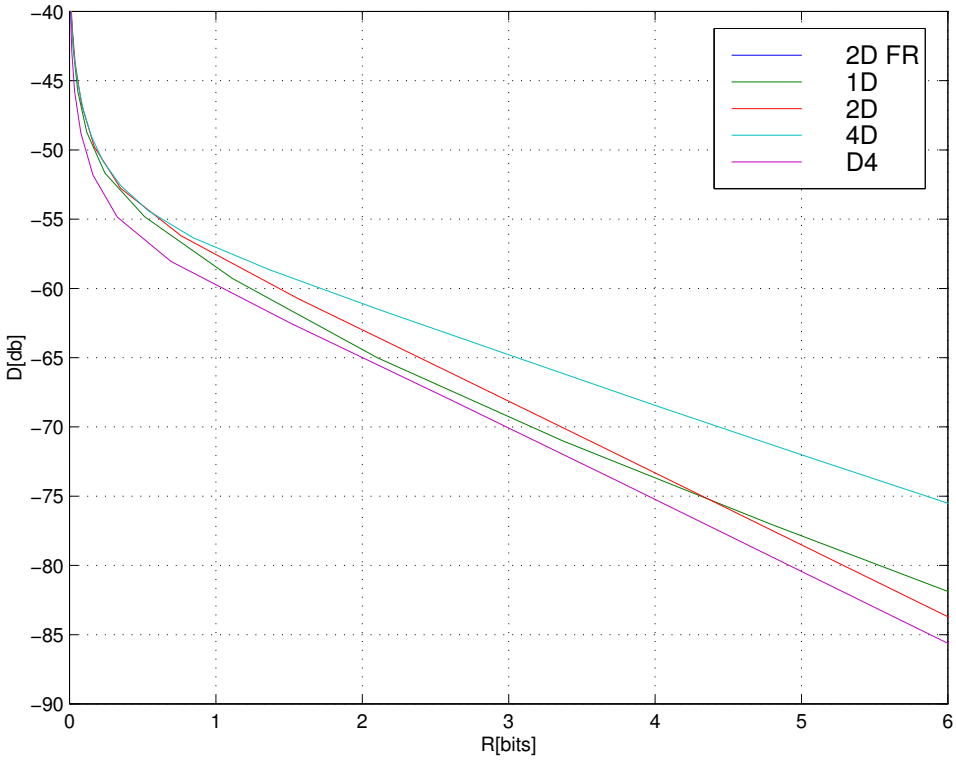


Figura H.4: Graph IV: D4 x binary codebooks (detail)

them.

Thus, α-expansions as a generalization of the usual binary expansion do lead to a rate $\times$ distortion performance improvement. This explains the results obtained in [1], [2] and [6] where a generalization of the EZW [7] algorithm for vectors is shown to yield superior performance. Similar experiments have also been conducted with the MGE [15] algorithm, for a bit-rate of 0.5 bpp. The PSNR for the scalar case was 36.68 dB, while the modified MGE with the Λ_{16} codebook (the first shell of the best known sphere packing in 16 dimensions [8]) resulted in a PSNR of 37.11 dB, with $\alpha = 0.6$.

Apêndice I

α-expansions and frame decompositions

This chapter examines the use of α-expansions as a scheme for multistage frame decompositions. Since α-expansions directly produce a quantized value, we will compare its rate $\times$ distortion characteristics with those of the decompose-quantize procedure, which is another scheme for frame decompositions. This application of α-expansions has been published in [4].

I.1 Introduction to frame expansions

Let $\mathcal{F} = \{e_1, e_2, \ldots, e_p\}$ be a collection of vectors generating $\mathbb{R}^N$. This means that every $x \in \mathbb{R}^N$ can be expressed as

$$x = \sum_{i=1}^{p} a_i e_i$$

for some coefficients $a_1, a_2, \ldots, a_p$. The vectors $\{e_1, e_2, \ldots, e_p\}$ may or may not be linearly independent. In the case where they are linearly dependent, the set $\mathcal{F}$ is called a frame or an overcomplete basis. For the purposes of this development, we shall call $\mathcal{F}$ a frame even

when the vectors are linearly independent. A more complete discussion of frames can be found in [28, 38].

Given a frame $\mathcal{F}$, consider the codebook $\mathcal{C} = \{e_1, e_2, \ldots, e_p, -e_1, -e_2, \ldots, -e_p\} = \{v_1, v_2, \ldots, v_q\}$, where $q = 2p$. Then the α-expansion over $\mathcal{C}$ of a vector x, given by

$$x = \sum_{i=0}^{\infty} \alpha^i v_{k_i}$$

can be seen as a decomposition of x in the frame $\mathcal{F}$. The advantage of using α-expansions as a method for frame expansions is that we can store them by simply coding the indexes $(k_1, k_2, \ldots)$, while in the usual decompose-quantize procedure we have to quantize the coefficients before coding them.

I.2 Rate-distortion characteristics of the decompose-quantize procedure

A general decompose-quantize procedure works as follows:

1. In the first step, the frame coefficients $(a_1, a_2, \ldots, a_p)$ are obtained from the vector $x \in P \subset \mathbb{R}^N$. The vector of coefficients is contained in a compact subset S of an N-dimensional subspace $L = L(\mathcal{F})$ of $\mathbb{R}^p$.

2. In the second step, the vector of coefficients is uniformly quantized to $\left(a_1^n, a_2^n, \ldots, a_p^n\right)$, where each a_i^n is represented by n bits. The quantization cells (or bins) are the hyper-cubes

$$\left[\frac{i_1}{2^n}, \frac{i_1+1}{2^n}\right] \times \ldots \times \left[\frac{i_p}{2^n}, \frac{i_p+1}{2^n}\right], \quad i_j \in \mathbb{Z}$$

   We shall denote by $\mathcal{H}$ the partition of $\mathbb{R}^p$ in the above hyper-cubes.

3. In the third and last step, a frame reconstruction scheme recovers a vector y near to x from the quantized coefficients $\left(a_1^n, a_2^n, \ldots, a_p^n\right)$.

We shall denote by f the initial probability distribution function of the points $x \in P$ and *assume that f is absolutely continuous*. The mean squared error D_n of the scheme is the mean of $\|y - x\|^2$ with respect to f. It was proved in [9] that there exists a coefficient $c(N, f)$ such that, for a large enough n,

$$D_n \geq \frac{c(N, f)}{F_n^{\frac{2}{N}}} \tag{I.1}$$

where F_n denotes the number of quantization cells in the polytope P.

Lemma: There exists a constant $b = b(S, L)$ such that

$$\#(\mathcal{H}_n \cap S) \leq b.2^{nN}$$

Proof: Without loss of generality, we can assume that the orthogonal projection π of the subspace L in

$$L_N = \{(x_1, x_2, \ldots, x_p) \,|x_{N+1} = \ldots = x_p = 0\}$$

is injective. In other words, the subspace L can be defined by the $(p - N)$ equations

$$x_{N+1} = \sum_{j=1}^{N} l_{ij} x_j, \;\; 1 \leq i \leq p$$

The partition $\mathcal{H}_n$ of $\mathbb{R}^p$ determines a partition $\pi(\mathcal{H}_n)$ of L_N. It is easy to see that, for any cell $B \in \pi(\mathcal{H}_n)$, the number of cells of $\mathcal{H}_n$ whose projection is B and intersecting L is at most $l^* = \max\{|l_{ij}|, 1 \leq i \leq p, 1 \leq j \leq N\} + 1$.

Also, if n is large, the number of cells of $\pi(\mathcal{H}_n) \cap \pi(S)$ is less than $(m(\pi(S)) + 1).2^{nN}$,

where m denotes the N-dimensional Lebesgue measure. Hence, the number of cells of $\mathcal{H}_n$ intersecting L is at most $l^*(m(\pi(S)) + 1).2^{nN}$. So, we conclude that

$$\#(\mathcal{H}_n \cap S) \leq b.2^{nN}$$

where $b(S, L) = l^*(m(\pi(S)) + 1)$.

Corollary: There is a constant $C = C(\mathcal{F}, \alpha, f)$ such that

$$D_n \geq C \left(\frac{1}{2}\right)^{2n} \tag{I.2}$$

Proof: Observe first that $F_n \leq \#(\mathcal{H}_n \cap S)$. Then substitute the result of the above lemma in Eq. I.1. Taking

$$C = \frac{c(N, f)}{b^{\frac{2}{N}}}$$

we obtain the desired result.

Now we can make an estimate of the rate-distortion characteristics of the general decompose-quantize procedure. We will not consider entropy-coding for the vector $\left(a_1^n, a_2^n, \ldots, a_p^n\right)$, so the total number of bits to represent it is $R = np$. Therefore, using Eq. I.2, we can write

$$R(D) \geq \frac{p}{2} \log_2 \left(\frac{C}{D}\right) \tag{I.3}$$

which is the estimate we were looking for.

I.3 Rate-distortion characteristics of the α-expansions

Assume that $\Lambda(\alpha, \mathcal{C}) = P(\alpha, \mathcal{C})$. If we approximate $x \in P(\alpha, \mathcal{C})$ by its n-term expansion $(\nu_{i_0}, \nu_{i_1}, \ldots, \nu_{i_{n-1}})$, the error of the approximation is given by $r_n(x) = x - \sum_{i=0}^{n-1} \alpha^i \nu_{k_i}$.

Hence the mean squared distortion is given by

$$D = \int_P \|r_n(x)\|^2 dm(x) \tag{I.4}$$

where m denotes the usual N-dimensional volume. We can make an estimate of this value as follows.

Since $r_n(x) \in P(\alpha, C)$, we have that $\|r_n(x)\| \leq \frac{B}{1-\alpha}\alpha^n$, where B is the maximum norm of a vector in $\mathcal{C}$. So we conclude from Eq. I.4 that

$$D \leq \frac{B^2 m(P(\alpha, \mathcal{C}))}{(1-\alpha)^2}\alpha^{2n}$$

The total number of bits used to represent $(k_0, k_1, \ldots, k_{n-1})$ is at most $n \log_2 q$. Therefore the rate distortion function $R(D)$ must satisfy the inequality

$$R(D) \leq \frac{\log_2(q)}{2\log_2\left(\frac{1}{\alpha}\right)} \log_2\left(\frac{C}{D}\right) \tag{I.5}$$

where $C = \frac{B^2 m(P(\alpha,\mathcal{C}))}{(1-\alpha)^2}$ is a constant.

I.4 Rate-distortion comparison between α-expansions and decompose-quantize procedures

From Eqs. I.3 and I.5, for a fixed distortion D we can write

$$\frac{R_2(D)}{R_1(D)} \leq \left[\frac{\log_2(2p)}{p\log_2\left(\frac{1}{\alpha}\right)}\right]\left[\frac{\log_2\left(\frac{C_2}{D}\right)}{\log_2\left(\frac{C_1}{D}\right)}\right]$$

where the indexes 1 and 2 in R and C are associated to the decompose-quantize and α-expansion procedures, respectively. For high bit-rates, $D \ll C_1$ and $D \ll C_2$ and hence

the factor in the second bracket approaches 1. So, to have $R_2 \leq R_1$ it is sufficient to have

$$\frac{\log_2(2p)}{\log_2(\frac{1}{\alpha})} \leq p$$

As an example of where this inequality holds, take the frame

$$\mathcal{F} = \left\{ (1,0), (\frac{\sqrt{(3}}{2}, \frac{1}{2}), (-\frac{\sqrt{3}}{2}, \frac{1}{2}) \right\}$$

and the corresponding codebook, whose convex hull is an hexagon. In this case, $p = 3$ and $\alpha = \frac{1}{2}$ is sufficient for convergence. For these values, high-bit rate coding with α-expansions should be more efficient than with the decompose-quantize procedure.

I.5 Remarks

Although the above results show that α-expansions can be more efficient (rate-distortion wise) than the general decompose-quantize procedure, it should be noted that α-expansions can be computationally harder.

It is also worthwhile to mention the strong links between the α-expansions with the matching pursuits algorithm, which is a popular frame decomposition algorithm. In the M. P. algorithm the input vector is orthogonally projected on the nearest frame vector, and the same process is repeated recursively for the residues thus generated. The value of each projection is quantized, being encoded along with the index of the corresponding vector in the codebook. The M. P. algorithm differs from the general decompose-quantize procedure in that the frame coefficients are not uniformly quantized. This makes the M. P. theoretical rate-distortion calculations extremely hard, and there are no readily available results regarding those calculations, so we cannot directly compare them to α-expansions. It should be noted, however, that α-expansions do not need to encode the quantized value

of the projection on the codebook vector.

Apêndice J

Conclusions

The results presented in Chapter H are important because they crown our search for the existence of "better" codebooks than the usual binary one — at least they are better for datasets bearing some special characteristics. These characteristics, however, have not been artificially introduced to produce such results (which would have been possible), but are normally found when we use frequency domain transforms of natural images.

So, although binary codebooks have been shown to present better asymptotic behavior for any initial dataset distribution, we also verified that for certain data rates and certain distributions other dictionaries can outperform the binary ones. These data rates are exactly those which interest us for image coding applications.

The conjecture about the existence of those alternative dictionaries was possible when we recognized that α-expansions could be seen as a generalization of the binary expansion. That justifies the relevance of the theory developed here. It not only presents us with this possibility, but it also helps explain the good results obtained in [1, 17, 6], where a generalization of the EZW algorithm [7] for vectors has shown a performance improvement relative to previous results.

Finally, the theory allows us to efficiently search for better dictionaries, by explaining

the relationships between the parameters α and γ (the scaling factor) and algorithm convergence. This is arguably the best contribution of this work; this knowledge makes it possible to improve existing representation schemes.

We must keep in mind that the dictionaries presented here are not necessarily the best; we just wanted to show that they are better than the binary one, and so better than the uniform scalar quantizer. This fact prompts the search for still better dictionaries. This search is limited, at a given time, to those dictionaries whose dimension and cardinality allow for their practical usage; after all, the computational cost associated with the method suggested here grows exponentially with those parameters. However, dictionary D4 is small enough to warrant its practical use now.

As long as computing hardware keeps following Moore's law [39] — i.e., while computational power also keeps growing exponentially with time for a fixed cost — we may expect for other dictionaries to be found and to enter the usability realm in the near future.

J.1 Future directions

This work can be extended in some directions. The search for efficient codebooks was done here with the sole purpose of showing that better codebooks than the binary do exist; a more comprehensive search will probably unveil still better dictionaries. Also, different algorithms for applying the concept of α-expansions may provide still better results; the theory may need to be extended to cover those. Indeed, an application to video encoding applying α-expansions is already being researched, with promising results [40].

Apêndice K

Publications

During development of this work, parts of the theory presented here and other related works have been presented in congresses and/or specialized literature. Here is the list of those publications.

- Poster "Successive Approximations Vector Quantization", *4 th Curves and Surfaces*, Association Française d'Approximation, Saint Malo, France, July 1999, with Marcos Craizer and Eduardo A. B. da Silva.

- "Quantized Frame Decompositions", *Saint-Malo 1999' Curve and Surface Fitting*, pp. 153–160, Vanderbilt University Press, 2000, with Marcos Craizer and Eduardo A. B. da Silva.

- "Alpha-expansions: a class of frame decompositions", *Applied and Computational Harmonic Analysis Journal*, Vol. 13, issue 2, pp. 103–115, Sep 2002, Academic Press, with Marcos Craizer and Eduardo A. B. da Silva.

- "Successive Approximation Quantization for Image Compression", *IEEE Circuits & Systems Magazine*, Q3 2002, pp. 20–45, with Eduardo A. B. da Silva and Marcos Craizer.

Apêndice L

Barycentric coordinates and the $(N+1)$ polytope

Suppose a polytope $P = \{v_k\}$ has $N+1$ vertices. Then every point in it can be written in a unique way as

$$x = \sum_{k=1}^{N+1} t_k v_k$$

where $0 \leq t_k \leq 1$ and $\sum_{k=1}^{N+1} t_k = 1$. The t_k are called the BARYCENTRIC COORDINATES of x. The barycenter b of P is the point whose coordinates are $\left(\frac{1}{N+1}, \frac{1}{N+1}, ..., \frac{1}{N+1}\right)$.

Lemma: In barycentric coordinates, the function f_k is defined by

$$f_k(t_1, t_2, ..., t_{N+1}) = (\alpha t_1, ..., \alpha_{t_k} + 1 - \alpha, ..., \alpha t_{N+1})$$

Proof: If $x = (t_1, ..., t_{N+1})$, then

$$f_k(x) = \alpha \sum_{j=1}^{N+1} t_j w_j + (1-\alpha) w_k$$

$$= \left(\sum_{j \neq k} \alpha t_j w_j \right) + (\alpha t_k + 1 + -\alpha)\, w_k$$

Theorem: For $(N+1)$ polytopes, $\Lambda = P_\alpha \Leftrightarrow \alpha \geq \frac{N}{N+1}$

Proof: First, observe that if $\alpha \leq \frac{N}{N+1}$, then

$$\alpha t_k + 1 - \alpha \geq 1 - \alpha > \frac{1}{N+1}$$

which implies that $b \notin F_\alpha(P_\alpha)$. Thus, $F_\alpha(P_\alpha) \neq P_\alpha$ and so $\Lambda \neq P_\alpha$.

On the other hand, suppose that $\alpha \geq \frac{N}{N+1}$. Any $s = (s_1, ...s_{N+1}) \in P_\alpha$ must have a coordinate, say s_k, larger than or equal to $\frac{1}{N+1}$. All others must then be smaller than $\frac{N}{N+1}$. Take

$$x = \left(\frac{s_1}{\alpha}, ..., \frac{s_k - 1 + \alpha}{\alpha}, ..., \frac{s_{N+1}}{\alpha} \right)$$

Observe that $\alpha \geq \frac{N}{N+1}$implies that all coordinates of x are in the interval $[0, 1]$, so $x \in P_\alpha$. Since $f_k(x) = y$, it means that $y \in f_k(P_\alpha)$. As this holds for any $y \in p_\alpha$, we conclude that

$$\bigcup_{j=1}^{N+1} f_j(P_\alpha) = P_\alpha$$

and hence $\Lambda_\alpha = P_\alpha$.

Referências Bibliográficas

[1] da Silva, E.A.B., Sampson, D.G., Ghanbari, M., "A Successive Approximation Vector Quantizer for Wavelet Transform Image Coding*", IEEE Transactions on Image Processing, Special Issue on Vector Quantization*, Vol. 5, No. 2, pp. 299–310, February 1996

[2] da Silva, E.A.B, *Wavelet Transforms for Image Coding*, Ph.D. Dissertation, University of Essex, U.K., June 1995

[3] Barnsley, M.F., *Fractals Everywhere*, Academic Press, Inc., 1988

[4] Craizer, M., Fonini Jr., D.A., da Silva, E.A.B, "Alpha-expansions: a class of frame decompositions", *Applied and Computational Harmonic Analysis Journal*, Vol 13, issue 2, pp 103-115, Sep 2002, Academic Press

[5] Craizer, M., Fonini Jr., D.A., da Silva, E.A.B, "Quantized Frame Decompositions", in *Saint-Malo 1999' Curve and Surface Fitting*, pp. 153–160, Vanderbilt University Press, 2000

[6] Craizer, M., da Silva, E.A.B, Ramos, E.G., "Convergent algorithms for successive approximation vector quantization with applications to wavelet image compression", *IEE Proceedings - Vision, Image and Signal Processing*, 146(3):159–164, July 1999

[7] Shapiro, J.M., "Embedded image coding using zero-trees of wavelet coefficients", *IEEE Trans. Signal Processing*, Vol. 41, pp. 3445-3462, Dec 1993

[8] Conway, J.H., Sloane, N.J.A., *Sphere Packings, Lattices and Groups*, Springer-Verlag, 1988

[9] Zador, P.L., *Development and evaluation of procedures for quantizing multivariate distributions*, Ph.D. Dissertation, Stanford University, 1963

[10] Diniz, P.S.R., da Silva, E.A.B, Netto, S.L., *Digital Signal Processing: System Analysis and Design*, Cambridge University Press, 2002

[11] Said, A., Pearlman, W.A., "A new, fast and efficient image codec based on set partitioning in hierarchical trees", *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 6, No. 3, pp. 243–250, June 1996

[12] Taubman, D., "High Performance Scalable Image Compression with EBCOT", *IEEE Transactions on Image Processing*, Vol. 9, No. 7, July 2000

[13] Mallat, S., Falzon, F., "Analysis of Low Bit Rate Image Transform Coding", *IEEE Transactions on Signal Processing*, Vol. 46, No. 4, April 1998

[14] Equitz, W.H.R., Cover, T.M., "Successive Refinement of Information", *IEEE Transactions on Information Theory*, Vol. 37, No. 2, March 1991

[15] Tse-Hua Lan, Tewfik, A.H., "Multigrid Embedding (MGE) Image Coding", in *Proceedings of the 1999 International Conference on Image Processing*, Kobe, Japan

[16] Taubman, D., Zakhor, A., "Multirate 3-D Subband Coding of Video", *IEEE Transactions on Image Processing*, Vol. 3, No. 5, September 1994

[17] Taubman, D., Zakhor, A., "A Common Framework for Rate and Distortion Based Scaling of Highly Scalable Compressed Video", *IEEE Transactions on Circuits Systems for Video Technology*, Vol. 46, No. 4, August 1996

[18] Jain, A.K., *Fundamentals of Digital Image Processing*, Englewood Cliffs, NJ, Prentice Hall, 1989

[19] Sayood, K., *Introduction to Data Compression*, 2nd Edition, San Francisco, CA, Morgan Kaufmann Publishers, 2000

[20] Jayant, N., *Image Coding Based on Human Visual Models in Image Processing*, Academic Press, 1994

[21] Mallat, S.G., *A Wavelet Tour of Signal Processing*, San Diego, CA, Academic Press, 1998

[22] Berger, T., *Rate Distortion Theory: A Mathematical Basis for Data Compression*, Englewood Cliffs, NJ, Prentice Hall, 1971

[23] Cover, T.M., Thomas, J.A., *Elements of Information Theory*, New York, NY, John Wiley and Sons, 1991

[24] Shapiro, J.M., "An Embedded Wavelet Hierarchical Image Coder", in *Proceedings of the 1992 ICASSP Conference*, pp IV-657–IV-660. San Francisco, March 1992

[25] Bell, T.C., Cleary, J.G., Witten, I.H., *Text Compression*, Englewood Cliffs, NJ, Prentice Hall, 1990

[26] http://carbon.cudenver.edu/~hgreenbe/glossary/tours/linearalgebra.html

[27] Watson, A.B. (Editor), *Digital Images and Human Vision*, MIT Press, 1993

[28] Gersho, A., Gray, R.M., *Vector Quantization and Signal Compression*, New York, NY, Kluwer Academic Publishers, 1991

[29] Fonini Jr., D.A., *Quantization with Alpha-expansions* (Internal Report), http://lps.ufrj.br/IR/Fonini/alpha-exp.pdf

[30] Pennebaker, W.B., Mitchell, J.L., *JPEG: Still Image Compression Standard,* New York, NY, Kluwer Academic Publishers, 1992

[31] *JPEG2000 Verification Model 5.3*, ISO/IEC JTC1/SC29/WG1 (ITU/T SG28), 1999

[32] *Coding of Moving Pictures and Associated Audio for Digital Storage Media up to 1.5Mbit/s*, ISO/IEC JTC1/CD 11172, 1992

[33] *Generic Coding of Moving Pictures and Associated Audio*, ISO/IEC JTC1/CD 13818, 1994

[34] *Coding of Moving Pictures and Audio*, ISO/IEC JTC1/SC29/WG11/CD 14496, 1997

[35] Lovisolo, L., da Silva, E.A.B, "Uniform distribution of points on a hyper-sphere with applications to vector bit-plane encoding", *IEE Proceedings, Vision, Image and Signal Processing*, Vol. 148, No. 3, pp. 187–193, June 2001

[36] Saff, E.B., Kuijlaars, A.B.J., "Distributing many points on a sphere", *Math. Intelligencer*, Vol. 19, No. 1, pp. 5–11, 1997

[37] Eckhoff, J., "Helly, Radon, and Carathéodory Type Theorems", In: P.M. Gruber, J.M. Wills, *Handbook of Convex Geometry*, Ch. 2.1, pp. 389–448, Amsterdam, the Netherlands, North-Holland, 1993

[38] Hutchinson, J.E., "Fractals and Self-Similarity", *Indiana Journal of Mathematics*, Vol. 30, No. 5, pp 713–747, 1981

[39] Moore, G.E., "Cramming more components onto integrated circuits", *Electronics*, Vol. 38, No 8, April 19, 1965

[40] Caetano, R., da Silva, E.A.B, "Using Greedy Decompositions on Generalized Bit Planes", Electronic Letters, 35(11), pp 507–508, May 2002