UNIVERSIDADE
FEDERAL RURAL
DE PERNAMBUCO

Universidade Federal Rural de Pernambuco
Departamento de Matemática

Mestrado Profissional em Matemática

A ÁLGEBRA DE MATRIZES E O PROCESSAMENTO DE IMAGENS DIGITAIS

Áulus Santos Correia de Melo

DISSERTAÇÃO DE MESTRADO

Recife
2015

Universidade Federal Rural de Pernambuco
Departamento de Matemática

Áulus Santos Correia de Melo

A ÁLGEBRA DE MATRIZES E O PROCESSAMENTO DE IMAGENS DIGITAIS

Trabalho apresentado ao Programa de Mestrado Profissional em Matemática do Departamento de Matemática da Universidade Federal Rural de Pernambuco como requisito parcial para obtenção do grau de Mestre em Matemática.

Orientador: *Prof. Dr. Thiago Dias*

Recife
2015

Dedico este trabalho aos professores que tive desde o ensino básico ao mestrado, pois devo a seus entusiasmo e dedicação o fato de minha opção pela docência e aprimoramento profissional ter sido tão natural e repleta de significado.

AGRADECIMENTOS

Agradeço a minha esposa, Ellen, pelo companheirismo, compreensão e apoio sem os quais não seria possível o fim desta jornada. Agradeço a meus pais, Denisaldo e Helena, bem como a minha tia e segunda mãe, Creuza, pela capacidade de serem um exemplo constante em minha vida e agradeço ao meu filho, Pietro, por ser uma infindável fonte de inspiração e alegria para os meus dias.

Ao meu orientador, Thiago Dias, principalmente pela paciência e compreensão dos problemas enfrentados para a conclusão deste trabalho, bem como pelas sugestões e correções oportunas e enriquecedoras.

A Deus, por permitir a ocorrência dos eventos que vieram a me fazer grato neste momento.

RESUMO

Neste trabalho faremos um estudo sobre a relação entre a álgebra de matrizes e o processamento de imagens digitais com o intuito de fornecer um material de referência ao professor do ensino médio que se proponha a trabalhar tal tema em suas aulas. O trabalho apresenta-se em duas etapas: a primeira estabelece a relação entre matrizes e imagens digitais, introduz o software matemático *GNU Octave* e em seguida apresentamos uma série de aplicações deste na edição de imagens digitais. Na segunda etapa, discutiremos sobre a compressão de imagens e faremos uma exposição rigorosa da teoria por trás de tal tema.

Palavras-chave: Imagens Digitais, GNU Octave, Processamento de Imagens, Decomposição em Valores Singulares, norma de Frobenius, Compressão de Imagens Digitais.

ABSTRACT

In this work we do a study on the relationship between algebra and matrix processing digital images in order to provide references to high school teachers to work this theme in their classes. The work is presented in two stages: the first stablishes the relationship between arrays and digital images, introduces the mathematical *GNU Octave software* and then present a series of applications of this in editing digital images. In the second stage, we discuss about the image compression and do a thorough exposition of the theory behind such theme.

Keywords: Digital Images, GNU Octave, Image Processing, Singular Value Decomposition, Frobenius Norm, Digital Image Compression.

SUMÁRIO

CAPÍTULO 1

INTRODUÇÃO

No ensino básico, o aprendizado de tópicos da teoria de matrizes ainda encontra muitas limitações, sobretudo de caráter aplicacional e, portanto, motivacional.

Parece-nos um consenso entre todos os professores que o aprendizado de um determinado tema torna-se mais eficaz quando o aluno conhece sua aplicabilidade (pelo menos aquelas que lhe são possíveis acessar). Acreditamos que esta eficácia alcance seu ápice com o estudante que não apenas conhece tais aplicações, mas que é levado a adquirir consciência de como utilizá-la em sua própria vida pessoal ou profissional. A diferença entre o aluno que conhece as aplicações de um tema e o que tem consciência de seu uso é como a diferença entre alguém que, a partir de um ponto em uma estrada, consegue visualizar possíveis rotas e alguém que além de visualizá-las seja capaz de fazer projeções sobre como será sua ida em cada direção. Esta segunda pessoa certamente estará mais apta a decidir por qual caminho seguirá, bem como o estudante, consciente de como a matemática, ou qualquer outra disciplina, pode conduzi-lo a obtenção de um saber concreto sobre o que o rodeia, é capaz de definir, com relativa clareza, quais são as áreas que mais lhe afeiçoam e quais seriam viáveis a seu investimento pessoal.

As aplicações da matemática constituem um dos mais importantes meios de aliar o ensino desta disciplina aos projetos pessoais e profissionais do estudante, pois são capazes de apresentá-lo nuances da presença da matemática em diversas profissões e campos de conhecimento, ou simplesmente são capazes de fasciná-lo, mantendo-o motivado, em todo caso.

Há muito, as matrizes se desenlaçaram dos sistemas de equações lineares, sobre os quais se originaram, e se tornaram imensamente aplicadas dentro e fora da matemática. Para um rápido convencimento, basta apontarmos que elas são fundamentais para o desenvolvimento da álgebra linear e da estatística, campos de indiscutível importância em diversos seguimentos da ciência em geral. Em razão de sua relevância, o estudo das matrizes é um componente obrigatório no currículo do ensino médio e, felizmente, sua abordagem nos livros didáticos atuais é repleta de aplicações.

Entretanto, defendemos que a exposição de aplicações por si só são insuficientes para provocar uma aprendizagem satisfatória em uma parcela significativa dos alunos, visto que as aplicações, quando não efetivadas, podem parecer como coisas demasiadamente distantes, que só poderão ser melhor exploradas no futuro, e assim não careceriam de um grande investimento do estudante em sua compreensão, falhando, desta forma, na tarefa de mantê-lo estimulado. Em um cenário como este, pode-se dizer, no máximo, que o aluno conheceu as aplicações.

Consideramos o estudo de uma aplicação efetiva em sala de aula, quando o discente além de compreender seu funcionamento é capaz de produzir ideias a seu respeito e assim tornar-se capaz de propor e resolver problemas associados.

O ensino, pela condução efetiva de aplicações em sala de aula, das noções iniciais da teoria das matrizes, no entanto, encontra um difícil obstáculo, para o professor e para o aluno, na complexidade de tais aplicações.

A proposta do presente trabalho é reunir material suficiente para subsidiar uma condução efetiva do uso da álgebra de matrizes em computação gráfica, através do software matemático GNU octave. Faremos ainda a exposição das bases matemáticas para o uso da *Decomposição em Valores Singulares* na compressão de imagens digitais, a qual, embora de caráter técnico, muitas ideias relacionadas são acessíveis ao aluno do ensino médio, e a obtenção de resultados através do octave é muito simples.

Faremos, no primeiro capítulo, uma descrição dos conceitos de imagem digital, espaços de cor e comentaremos sobre os principais formatos das imagens digitais. No segundo apresentaremos os princípios básicos do GNU octave. No capítulo subsequente será feita a aplicação da álgebra elementar das matrizes junto as ideias desenvolvidas no capítulo 1 na edição de imagens digitais. No quarto capítulo teremos o desenvolvimento algébrico que leva a demonstração do teorema da decomposição em valores singulares e mostraremos porque tal decomposição fornece bons resultados no processo de redução de dados de uma matriz. No último capítulo, discutiremos a aplicação das ideias desenvolvidas no capítulo 3 na compressão de imagens digitais.

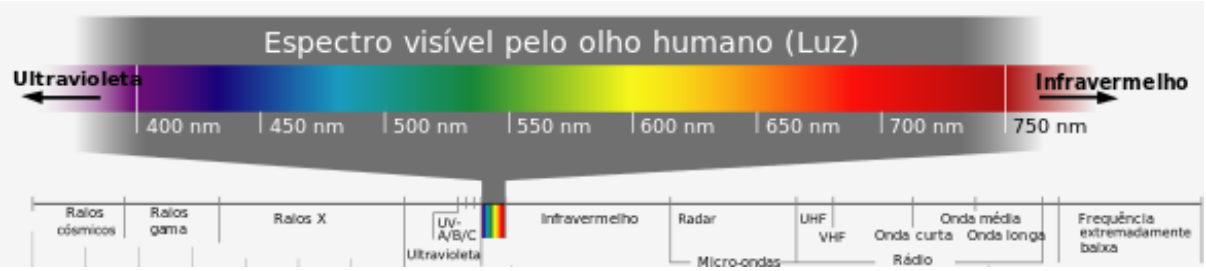
CAPÍTULO 2

IMAGENS DIGITAIS

As imagens digitais surgem como um produto dos esforços da ciência da computação em duas frentes: a codificação dos sinais relevantes para a constituição de uma imagem e a reconstrução desta a partir dos dados codificados. Suas aplicações vão muito além do uso em computação gráfica, alcançando a geologia, a medicina, a astronomia, a engenharia, entre outras.

Uma imagem é o resultado da interação da luz com um conjunto de receptores óticos, estes capazes de perceber a intensidade da luz e a variação de cores. A percepção de diferentes cores em uma imagem corresponde a maneira como cérebro distingue diferentes faixas do espectro eletromagnético (figura 2.1). Não há dúvida que a sensação de cor é fundamental na constituição da ideia de imagem, de tal modo que, para se descrever minunciosamente uma imagem, basta indicar a cor presente em cada ponto desta.

Figura 2.1 Espectro visível



Fonte: Wikipédia[1]

Sendo assim, é possível fornecer um modelo matemático para descrever uma imagem. Considerando que toda imagem possa ser representada nos limites de um retângulo, podemos identificá-la por uma função I que associa cada ponto (x, y) do retângulo $R \subset \mathbb{R}^2$ a uma cor $I(x, y)$ correspondente ao efeito da atuação da luz naquele ponto.

Em computação gráfica, ocorre a discretização do retângulo R do domínio de I, processo em que há uma quebra da imagem em um conjunto finito de elementos, cada um associado a uma cor em um sistema de cores C. Cada elemento de I é chamado de pixel (do inglês *picture element*) e uma imagem digital fica definida por I através da função

$$I : \mathbb{N}_m \times \mathbb{N}_n \to C; \tag{2.1}$$

onde $\mathbb{N}_i = \{1, 2, \ldots, i\}$.

As cores de C são obtidas de um sistema de cor padrão, o qual descreve matematicamente todas as cores visíveis e baseia-se no fato do olho humano possuir três tipos de

[1]Disponível em <https://pt.wikipedia.org/wiki/Espectro_visível> Acesso em out 2015

receptores de cores: receptores para a cor vermelha (*Red*), para a verde (*Green*) e para a azul (*Blue*). As demais cores são formadas através das combinações dessas três em diferentes intensidades. A intensidade de cada cor pode ser dada por uma escala numérica em que o 0 representa a ausência daquele componente e o 1 representa o componente em sua intensidade máxima. Podemos ainda, considerar imagens a partir da intensidade da luz, ou luminosidade, emitida ou refletida em cada ponto de R, atribuindo valores de 0, para a ausência de luz (preto), até 1, para a luminosidade máxima (branco). Tal imagem formada por um único componente é dita em *escala de cinza* enquanto as que possuem três componentes são as *coloridas*. Decorre destas considerações que uma imagem em escala de cinza pode ser representada por uma matriz, a qual, em computação gráfica, tem suas entradas discretizadas em elementos do conjunto $\mathbb{N}_{255}$, enquanto uma imagem colorida é dada por três matrizes como esta, cada uma correspondente a uma das componentes R, G e B.

O conjunto C, no entanto, não é composto por toda as cores do espectro visível. Além de ser uma discretização do sistema de cores padrão, Ele é construído conforme as limitações dos dispositivos de reprodução de imagem (monitores, impressoras, etc) e do o uso que se queira fazer da imagem. Um documento de texto, por exemplo, é uma imagem digital cujas cores utilizadas são o branco (fundo) e o preto (letras), já uma animação em .gif possui no máximo 256 cores em cada *frame* (imagem de cada quadro da animação).

A escolha de cores para compor o conjunto C determina algumas características de I. O número de bits nescessários para representar uma cor em C define a *dimensão* ou *profundidade do pixel*. O modo da imagem classifica-a segundo sua profundidade de cor e conjunto de cores utilizados. Os principais modos da imagem são:

- *bitmap*: Imagem de um bit que assume os valores preto (0) ou branco (1);
- *meio tom ou escala de cinza*: Imagem de oito bits em que varia do preto (0) ao branco (255) passando por diversos tons de cinza, totalizando 256 cores utilizadas.
- *RGB*: Imagem colorida que, Dependendo do formato, possui de 24 a 32 bits.
- *imagem indexada*:: 8 bits, colorida. Aqui podem ser estipuladas até 256 cores em um sistema de cor para representar uma imagem.

Exemplo 2.0.1.

(a) (b) (c) (d) (e) (f)

Figura 2.2

Na figura 2.2 temos uma mesma imagem exibida respectivamente em cada modo descrito acima. A figura 2.2(a) está no modo bitmap, a 2.2(b) está em escala de cinza,

a 2.2(c) colorida e na 2.2(d) indexada. Na figura 2.2(e) temos o chamado *mapa de cores* da imagem indexada 2.2(d) e contém todas as cores presentes nela.

Abaixo, exibimos as intensidades da região destacada na imagem 2.2(f):

$$R = \begin{bmatrix} 37 & 38 & 38 & 37 & 36 \\ 34 & 34 & 34 & 34 & 33 \\ 33 & 33 & 33 & 32 & 33 \\ 26 & 11 & 0 & 0 & 0 \\ 0 & 55 & 111 & 134 & 125 \end{bmatrix} \quad G = \begin{bmatrix} 124 & 125 & 125 & 125 & 124 \\ 118 & 118 & 118 & 118 & 117 \\ 113 & 113 & 113 & 112 & 113 \\ 108 & 105 & 101 & 97 & 93 \\ 96 & 99 & 106 & 120 & 128 \end{bmatrix}$$

$$B = \begin{bmatrix} 195 & 196 & 196 & 196 & 195 \\ 190 & 190 & 191 & 190 & 189 \\ 186 & 186 & 186 & 185 & 186 \\ 184 & 193 & 201 & 200 & 196 \\ 191 & 148 & 110 & 107 & 119 \end{bmatrix}$$

Serão trabalhados aqui, principalmente, imagens no modo RGB e em escala de cinza.

2.1 SISTEMAS DE CORES

Matematicamente, uma cor é dada por coordenadas do espaço de cores, definidas por um sistema padrão de cor. Diferentes sistemas de cores são obtidos tomando subconjuntos do sistema padrão e exibindo as cores destes por algum critério distinto, ou seja, utilizando bases diferentes para o subespaço. Discutiremos adiante a respeito de quatro sistemas de cor, escolhidos pela sua abrangência ou relevância para este texto.

2.1.1 Sistema RGB

O sistema de cor RGB baseia-se no sistema padrão e é muito usado em monitores e sistemas operacionais. Nele, como já foi dito, cada cor é dada por três coordenadas correspondentes a intensidade da luz vermelha, da verde e da azul, as quais combinadas geram a cor fisicamente. Na figura 2.3, temos uma foto acompanhada de imagens que correspondem a intensidade da luz nos canais RGB. As regiões mais escuras indicam as regiões em que a matriz associada possui valores próximos de 0. Na imagem vermelha vemos que o rosto e ombros da modelo são dados em um tom intenso e praticamente uniforme, o que revela uma grande participação desta cor nas regiões referidas. A componente verde é percebida com mais intensidade pelo olho humano, isto porque o verde

Figura 2.3 imagem original e decomposta em suas três componentes RGB

Figura 2.4 Diagramas de formação de cores

(a) Formação de cores no CMY (b) Formação de cores no RGB

Fonte: Wikipedia[2]

é responsável pela maior parte da *luminância* de uma imagem, o azul, por sua vez, exibe pouca luminância.

2.1.2 Sistemas CMY e CMYK

Em comparação com o RGB, o sistema de cores CMY abandona o paradigma da percepção da cor enquanto luz e adota o da cor enquanto pigmento. Enquanto no sistema RGB temos as cores formadas a partir da combinação luminosa de diferentes faixas do espectro eletromagnético, no CMY as cores são obtidas pela subtração destas em uma superfície. Se, por exemplo, percebemos a cor azul ao observar um objeto, isto significa que a superfície daquele objeto absorve as componentes vermelha e verde da luz e reflete apenas a azul. As coordenadas do CMY correspondem as cores ciano (*Cian*), magenta (*Magenta*) e amarelo (*Yellow*) cujos pigmentos absorvem, respectivamente, as cores vermelha, verde e azul. No sistema RGB, o aumento do valor das coordenadas corresponde a um aumento de luminância na cor, no CMY um tal aumento corresponde a perda de luminância, fazendo que, ao contrário do RGB, o CMY parta do branco $(0, 0, 0)$ ao preto $(255, 255, 255)$, conforme indicado na figura 2.4.

Assim, para obtermos o azul que no RGB é dado por $(0, 0, 100)$ basta tomarmos em coordenadas do CMY a cor $(255, 255, 155)$ que corresponde a uma cor que, submetida a uma iluminação, absorve as cores vermelha e verde, e também 155 níveis da cor azul, refletindo, portanto, 100 níveis.

Dada uma cor (r, g, b) em coordenadas RGB, sua representação (c, m, y) em CMY, é obtida através da transformação

$$(c, m, y) = (255, 255, 255) - (r, g, b)$$

e vemos um exemplo da formação de uma imagem mediante a decomposição CMY na figura 2.5(c)

[2]Disponível em <https://pt.wikipedia.org/wiki/RGB> Acesso em out 2015

(c) imagem original e decomposta em suas três componentes CMY

(d) decomposição em componentes CMYK

Figura 2.5

O sistema CMYK utilizado em impressoras foi instituido por razões econômicas (A tinta preta correspondente a componente K é mais barata), e uma cor (c', m', y', k) em CMYK é obtida de (c, m, k) em CMY mediante as operações

$$\begin{aligned} k &= \min\{c, m, y\} \\ c' &= c - k \\ m' &= m - k \\ y' &= y - k. \end{aligned}$$

A figura 2.5(d) fornece um exemplo de decomposição em CMKY.

2.1.3 Sistema HSV

O sistema HSV procura representar cores a partir do sistema RGB seguindo uma compreensão intuitiva do que elas são. Para tanto, estabelece coordenadas que descrevam o conceito de matiz ou tonalidade (*Hue*), que é a cor propriamente dita, saturação (*Saturation*), que indica o quanto a cor encontra-se diluida no branco ou próxima dele, e valor ou brilho (*Value*), que estabelece a relação claro/escuro entre as cores. Este sistema é muito utilizado por artistas e designers.

A figura 2.6, diferente do que foi feito nos sistemas anteriores, em que cada imagem-componente era constituida unicamente das informações contidas em um grupo de coordenadas, tem em cada uma das três últimas imagens atribuições para os valores das três coordenadas do HSV. A nescessidade de tal atribuição se deve ao fato de que no sistema HSV uma cor não pode ser obtida sem o estabelecimento das três coordenadas. Na figura

(a) (b) (c) (d)

Figura 2.6 Imagem original e representações de suas componentes HSV

2.6(b) foi obtida mantendo o matiz da imagem original e atribuindo o valor nulo a saturação e o valor máximo ao brilho, assim podemos visualizar como ocorre a distribuição de cores na imagem. Já em 2.6(c), observamos o que acontece com as cores em 2.6(b) quando diluidas no branco, tal imagem é feita com os valores da matiz e da saturação da imagem 2.6(a), no entanto, mantivemos o brilho no máximo. Por fim, a imagem 2.6(d) foi feita a partir dos valores originais da matiz e do brilho, com a saturação dada em seus valores mínimos. Esta última imagem contém as informações de luminância da foto, enquanto a 2.6(c) contém as informações de crominância e juntas ela formam a imagem original 2.6(a). Na figura 2.7 vemos a aplicação desta mesma técnica de representação em uma imagem diferente.

2.2 FORMATO DE ARQUIVOS DE IMAGENS

Na maioria das aplicações em computação gráfica, exibir uma imagem não é o bastante. É imprescindivel o desenvolvimento de métodos para arquivar os dados que descrevem a imagem, de modo que possa haver o tráfego de tais dados e a reconstrução da imagem a partir deles. Para facilitar o tráfego e armazenamento de imagens criou-se o os métodos de compressão, que consistem em técnicas de processamento de dados que procuram reduzir o uso da memória do computador em seu armazenamento. Um *formato de arquivo de imagem* reúne um conjunto de normas de compressão e codificação de dados de uma imagem, de maneira a permitir sua correta interpretação pelo computador. Dentre os inúmeros formatos existentes, destacamos quatro deles em função de seu amplo uso e sua importância na breve história do computador.

BMP: *Windows Bitmap.* O formato de imagem padrão dos antigos sistemas operacionais da microsoft foi um dos primeiros formatos criados e tem seu uso reduzido a cada dia. Embora o formato BMP seja capaz de salvar imagens de alta qualidade, o custo computacional para isto é muito alto, visto que este formato não faz compressão, nem suporta a existência de transparência na imagem.

GIF: *Graphics Interchange Format.* Desenvolvido principalmente para o uso na internet, suporta imagens com no máximo 256 cores, dentre as quais uma pode ser tomada como transparente, é capaz de fazer compressão sem perda da informação e suporta animações. Exceto pelo seu popular uso em animações na internet, também

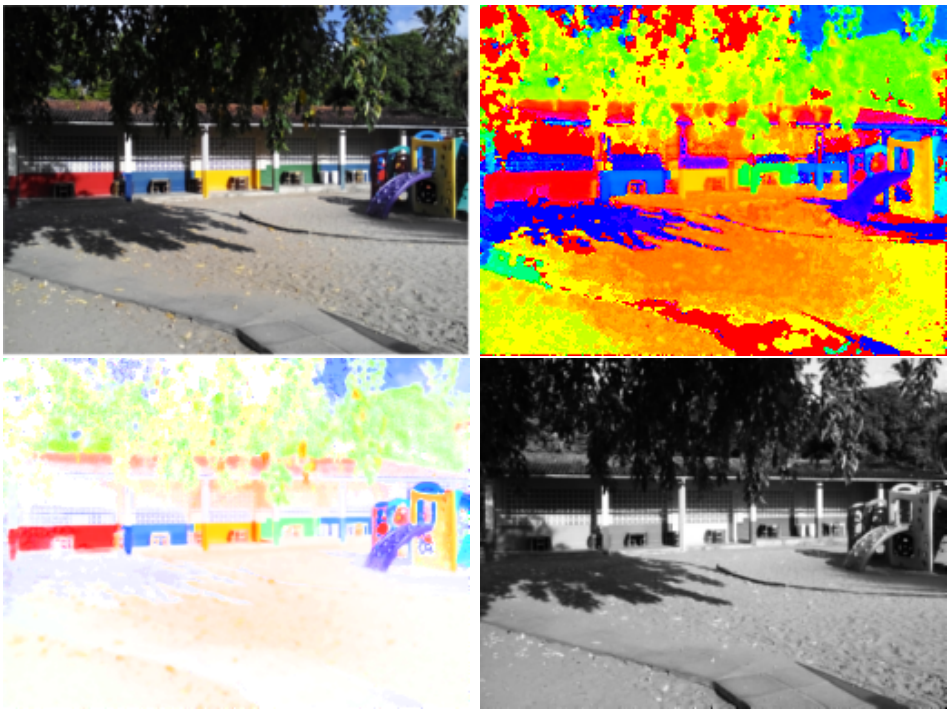


Figura 2.7

encontra-se caindo em desuso, principalmente pela sua limitação no uso de cores o que faz os usuários de imagens digitais optarem por um formato mais robusto como o JPEG ou o PNG.

JPEG: *Joint Photographics Expert Group*. Formato de imagem conhecido pela sua alta capacidade de compressão, valendo-se da redução na redundância interpixel e psicovisual da imagem. A compressão no formato JPEG é feita com perda da informação e baseia-se na leitura da matriz de dados pelo seu posto, o que, como veremos adiante, permite uma escrita simplificada de tal matriz, e daí, a compressão da imagem. O formato JPEG ainda leva em consideração a maior sensibilidade do olho humano na percepção do brilho de uma imagem, priorizando esta característica no momento da compressão. Suporta imagens de até 24 bits, sem transparência e sem animações. É suportado por quase todo dispositivo gráfico moderno, desde câmeras digitais a softwares de edição de imagens, é indicado para fotografias e desaconselhado para edição de imagens, visto que este formato comprime a imagem a cada salvamento fazendo com que ela perca bastante qualidade com essa prática.

PNG: *Portable Network Graphics*. Formato relativamente recente na computação gráfica, veio com o objetivo de suprir as limitações existentes no uso do GIF. Apresenta uma compressão sem perdas e suporta cores de até 32 bits, sendo 24 bits destinados a representação das cores RGB e 8 bits voltados para construção de transparências em 256 níveis, indo do opaco $(0, 0, 0)$ ao completamente transparente $(255, 255, 255)$.

CAPÍTULO 3

O GNU OCTAVE

O GNU octave, ou octave, é uma linguagem interativa de alto nível para computação numérica. Por ele é possível operar com matrizes, com funções, plotar gráficos em duas e três dimensões, desenvolver algoritmos para resolução dos mais variados problemas, processar imagens digitais, dentre outras aplicações. É seguro afirmar que a extensão do uso do octave é limitada apenas pela criatividade e domínio matemático-computacional do usuário.

O octave encontra-se disponível para os sistemas operacionais Windows, Linux e MacOS, é um software livre e de código aberto, isto significa que pode ser livremente distribuido e modificado segundo os termos da GNU, a *General Public License* (GPL). As versões para Windows e MacOS podem ser encontradas na página da sourceforge[1]; neste trabalho, utilizamos a versão 3.8.1 disponível para o Linux Ubuntu através da central de programas do Ubuntu.

Embora hajam algumas interfaces gráficas disponíveis para o octave, todos os processos do software são executados por linha de comando, em que o usuário digita uma instrução e tecla enter, e o octave fornece o resultado esperado, ou emite uma mensagem de erro caso o comando seja inválido. Tal maneira de conduzir a produção de resultados é pouco intuitiva a quem não possui intimidade com programação, pois exige do usuário o conhecimento prévio dos comandos na realização de alguma tarefa, o que torna seu uso pouco convidativo. No entanto, esta característica é imensamente compensada pela vasta possibilidade aplicacional do octave, fato que deve ser explorado em situações didáticas. Neste trabalho, faremos a exposição dos principais comandos e daqueles particularmente úteis para nossa abordagem através da seguinte notação:

Exemplo 3.0.1. : Comando (Título)

Breve descrição dos efeitos do comando.

```
>Entrada parâmetro #(quando houver)
Saída
```

Sugerimos ao leitor que a partir daqui leia o texto em curso com o octave aberto, para uma melhor experiência e compreensão de suas ferramentas. Nos próximos exemplos, enfatizaremos os *argumentos de entrada* para distingui-los do comando propriamente dito. O octave ignorará tudo que vier depois dos caracteres `#` e `%`, bem como o que estiver entre chaves. Este recurso é usado para inserir comentários no espaço de trabalho no octave. É possível inserir mais de uma instrução no prompt, bastando separá-las por vírgula.

[1]No site <http://octave.sourceforge.net/>

Exemplo 3.0.2. : *comando*, *comando* **(Vírgula)**

Atribuindo o valor 81 para a variável x e, em seguida, fazendo $\sqrt[4]{x}$.

```
>x=81, nthroot(x,4)
x = 81
ans = 3
```

Exemplo 3.0.3. : *comando*; *comando* **(Ponto e vírgula)**

Omitindo a visualização de um resultado pelo uso de ponto e vírgula.

```
>x=81; nthroot(x,4)
ans = 3
>x=81, nthroot(x,4);
x = 81
```

Nos últimos exemplos notamos a aparição do `ans` (do inglês *answer*), variável que armazena o último resultado obtido pelo octave. Outras variáveis especiais são o `pi` , número real $\pi \cong 3,1416$, o `i` e o `j`, ambos representando a unidade imaginária dos números complexos, o `inf` (*infinite*), resultado de divisões do tipo $1/0$ e o `NaN` (*Not a Number*) que é exibido como resultado de $0/0$.

Outro fato importante para atentar é a utilização do sinal de igualdade que aqui é um sinal de atribuição. Se, por exemplo, escrevemos `x=7`, `X=(17-3)/2` estaremos atribuindo valor 7 ao x, e o valor resultante de $(17-3)/2$ ao X (o octave diferencia maiúsculas de minúsculas). Caso se queira testar a igualdade entre as variáveis x e X deve-se digitar `x==X`.

3.1 GERENCIAMENTO DO AMBIENTE

Os comandos apresentados a seguir tem a função de gerenciar pastas e arquivos utilizados pelo octave, bem como gerenciar o próprio ambiente do octave, suas variáveis, tipos de variáveis etc.

Exemplo 3.1.1. : pwd

Indica o diretório de trabalho corrente do octave.

```
>pwd
ans = /home/pasta_pessoal
```

A pasta setada pela instrução `pwd` é onde ficam todos os arquivos salvos pelo octave, é nela, também, onde deve constar os arquivos a serem editados no software (imagens, funções, etc).

Exemplo 3.1.2. : mkdir e cd

Criando um novo diretório (`mkdir`) com o nome 'octave', e em seguida, mudando o diretório de trabalho corrente (`cd`).

```
>mkdir octave
ans = 1
>cd octave, pwd
ans = /home/pasta_pessoal/octave
```

Exemplo 3.1.3. : who e whos

Lista as variáveis utilizadas (`who`) e detalha-as (`whos`).

```
>who, whos
Variables in the current scope:
ans     x
Variables in the current scope:
 Attr  Name  Size         Bytes   Class
 ====  ====  ====         =====   =====
        ans  1x26         26       char
          x   1x1         1      double
```

Na coluna denominada `Class` o octave indica o tipo de variável naquela linha. Dentre os muitos tipos de variáveis, destacaremos o *double* e o *uint8*, que, respectivamente, representam números reais e números inteiros entre 0 e 255, associados à imagens digitais.

Exemplo 3.1.4. : clear *variável* **e clear all**

Limpa uma variável especificada (`clear`) ou todas (`clear all`).

```
>x=-3, y=pi, x+y
x = -3
y = 3.1416
ans = 0.1416
>clear x, who
Variables in the current scope:
ans    y>clear all, who
```

Os comando `clear` e `clear all` removem as atribuições das variáveis da memória do octave. Caso se queira limpar a tela, o comando a ser usado é o `clc`.

Exemplo 3.1.5. : help *comando*

Descreve uma função, em inglês, segundo a documentação constante no octave..

```
>help pwd
'pwd' is a built-in function from the file libinterp/corefcn/dirfns.cc
-- Built-in Function:  pwd ()
-- Built-in Function:  DIR = pwd ()
Return the current working directory.
See also:  cd, dir, ls, mkdir, rmdir.
Additional help for built-in functions and operators is available in the
online version of the manual.  Use the command 'doc <topic>' to search
the manual index.
Help and information about Octave is also available on the WWW at
http://www.octave.org and via the help@octave.org mailing list
```

Exemplo 3.1.6. : lookfor *vocábulo*

Lista todas as ocorrências no help de um conjunto de caracteres..

```
>lookfor root
matlabroot
roots
polynomial
nthroot
realsqrt
hypot
cbrt
sqrt
sqrtm
is_rooted_relative_filename
make_absolute_filename
rlocus
```

O `lookfor` exibe como resultado todos os comandos que possuem a palavra no parâmetro. Os comandos `help` e `lookfor` são os mais importantes para um usuário iniciante do octave. Usados em conjunto, permitem que o usuário saiba quais comandos contém a informação que ele deseja, qual a sintaxe de cada um e quais são os comandos relacionados.

3.2 MATRIZES

Para efetuar operações simples, o octave funciona como uma calculadora. Por exemplo, para encontrar o resultado de $7 \cdot \sqrt{3^2 - 8}$ basta inserir, como já foi visto, o comando `7*sqrt(3^2-8)`. Um grande diferencial aqui é a possibilidade de operar com matrizes, mesmo as de dimensões enormes, como é o caso das matrizes associadas a uma imagem digital. O octave trabalha ainda com as operações termo a termo entre matrizes, que são inseridas digitando um ponto antes da operação indicada. Por exemplo, se $A = [a_{ij}]$ e $B = [b_{ij}]$ são matrizes $m \times n$, a expressão `A.*B` retorna uma matriz $C = [c_{ij}]$ tal que

$c_{ij} = a_{ij} \cdot b_{ij}$. Se k é um escalar, é possível realizar uma operação termo a termo entre k e A fazendo `C=A.^k`, o retorno de tal expressão é $c_{ij} = a_{ij}^k$.

Exemplo 3.2.1. : [*var1*,*var2*] **e** [*var1*;*var2*] **(Vetor linha e vetor coluna)**

Definindo vetores.

```
>v=[3,4,5], u=[6;4;2]
 v  =
       3  4  5
 u  =
       6
       4
       2
```

Exemplo 3.2.2. : [*var1*, *var2*; *var3*, *var4*] **(Matrizes)**

Definindo matrizes.

```
>A=[1,2,3;4,5,6], B=[-1,3;-7,-2]
 A  =
        1   2  3
        4   5  6
 B  =
       -1   3
       -7  -2
```

Ao definir matrizes retangulares declaramos primeiramente as entradas da primeira linha, separando-as por espaço ou vírgula, e em seguida declaramos as entradas da segunda linha, separando estas daquelas na primeira linha com ponto e vírgula.

Exemplo 3.2.3. : *var*(*índice*), *var*(*linha*, *coluna*) **(Referência)**

Referenciando entradas dos vetores **u** e **v** e das matrizes A e B.

```
>v(2), u(3)
ans = 4
ans = 2
>A(2,3), B(4), A(5)
ans = 6
ans = -2
ans = 3
```

Dada $A = [a_{ij}]$, o elemento da i-ésima linha e j-ésima coluna é referido no octave como `A(i,j)`. O comando `A(n)` toma o n-ésimo elemento elemento da matriz, considerando-a ordenada de cima para baixo, da esquerda para a direita. Note a diferença nesta indexação dos elementos com a ordem em que se declaram as entradas na definição de uma matriz, que é da esquerda para a direita, de cima para baixo.

Exemplo 3.2.4. : *var1*:*var2*:*var3* **(Sequência aritmética)**

Através da instrução `início:passo:limite`, retorna uma matriz $1 \times n$ cujas entradas tem diferença constante dada por `passo`, começam do número `início` e não excedem o número em `limite`. Caso o `passo` não seja declarado, o octave toma por padrão `passo=1`.

```
>x=7:-2:-1, y=1:0.3:2.1, z=1:4
 x  =
      7   5   3   1   -1
 y  =
      1.0000  1.3000  1.6000  1.9000
 z  =
      1   2   3   4
```

Exemplo 3.2.5. : *var*(*linha*,:) **e** *var*(:,*coluna*) **(Referência a linhas e colunas)**

Definindo uma matriz C, e visualizando sua 2ªcoluna e 2ªlinha.

```
>C=[1:5;-8:-4;20:24;8:12;-2:2], C(:,2), C(2,:)
   C  =
          1   2   3   4    5
         -8  -7  -6  -5   -4
         20  21  22  23   24
          8   9  10  11   12
         -2  -1   0   1    2
 ans  =
          2
         -7
         21
          9
         -1
 ans  =
         -8  -7  -6  -5  -4
```

Dada uma matriz A os comandos `A(:,k)` e `A(k,:)` exibem, respectivamente, a k-ésima coluna e a k-ésima linha de A.

Exemplo 3.2.6. : *var*(*var1*:*var2*:*var3*; *var4*:*var5*:*var6*) **(Referência sequencial)**

Retorna matrizes cujos elementos são obtidos da matriz C definida no exemplo 3.2.5.

```
>C(3:5,2:4), C(1:2:5,1:3:5)
 ans  =
        21  22  23
         9  10  11
        -1   0   1
 ans  =
         1   4
        20  23
        -2   1
```

A instrução `A(início1:passo1:fim1,início2:passo2:fim2)`, toma nas linhas entre `início1` e `fim1`, cuja ordem entre as linhas dista `passo1`, as entradas das colunas entre `início2` e `fim2`, distantes `passo2` uma da outra.

Exemplo 3.2.7. : eye, ones, zeros e rand

Retorna matrizes pré-definidas com tamanho definido no parâmetro.

```
>eye(2,3), ones(2,3), zeros(2,3), rand(2,3)
 ans  =
 Diagonal Matrix
        1  0  0
        0  1  0
 ans  =
        1  1  1
        1  1  1
 ans  =
        0  0  0
        0  0  0
 ans  =
        0.092729  0.181822  0.510887
        0.861345  0.487002  0.282149
```

O comando `rand(m,n)` Retorna uma matriz $m \times n$ de entradas randômicas entre 0 e 1. Já as instruções `zeros(m,n)` e `ones(m,n)` retornam matrizes com entradas iguais a 0 e 1 respectivamente. A instrução `eye` retorna a matriz identidade generalizada, que é uma matriz diagonal generalizada com suas entradas não-nulas iguais a 1. Nestes quatro comandos, caso um único escalar seja declarado no parâmetro, o octave retornará uma matriz quadrada com o tamanho indicado, em outras palavras, `eye(n)` equivale a `eye(n,n)`.

Exemplo 3.2.8. : round, ceil, floor e fix

Retornam inteiros segundo algum critério.

```
> M=[1.3,-2.9;-3.2,1.9], round(M), ceil(M), floor(M), fix(M)
   M =
      1.3000  -2.9000
     -3.2000   1.9000
 ans =
     1  -3
    -3   2
 ans =
     2  -2
    -3   2
 ans =
     1  -3
    -4   1
 ans =
     1  -2
    -3   1
```

O comando `round(M)` retorna os inteiros mais próximos de cada entrada de `M`. Já o comando `ceil(M)` retorna os maiores inteiros menores ou iguais a cada entrada de `M` e `floor(M)` retorna os menores inteiros maiores ou iguais a cada entrada em `M`. `fix(M)` retorna a parte inteira de cada entrada da matriz `M`.

Exemplo 3.2.9. : length e size

Retorna o número de entradas de um vetor $\mathbf{v}$ e o número de linhas×colunas de uma matriz A.

```
>v=[2 1 3 4 5], A=[1:4;2:5;3:6], length(v), [m n]=size(A)
   v  =
         2  1  3  4  5
   A  =
         1  2  3  4
         2  3  4  5
         3  4  5  6

 ans  =  5
   m  =  3
   n  =  4
```

Caso o A seja uma matriz associada a uma imagem colorida, o octave a lerá como uma *matriz 3-d*, o que significa que quando fizermos `size(A)` encontraremos três valores de saída, ao invés de dois. O terceiro valor retorna uma matriz correspondente a alguma componente RGB, por exemplo, as matrizes `A(:,:,1)`, `A(:,:,2)` e `A(:,:,3)` correspondem as matrizes associadas as componentes vermelha, verde e azul respectivamente.

Exemplo 3.2.10. : diag

Retorna uma matriz diagonal a partir das entradas de **v**, caso o parâmetro seja uma matriz, retorna a diagonal principal desta na forma de um vetor.

```
>diag(v), diag(A)
 ans  =
 Diagonal Matrix
         2  0  0  0  0
         0  1  0  0  0
         0  0  3  0  0
         0  0  0  4  0
         0  0  0  0  5
 ans  =
         1
         3
         5
```

Exemplo 3.2.11. : *var1=var2\ va3* **(Resolução de sistemas)**

Resolve o sistema de equações lineares dado por $A\mathbf{x} = \mathbf{b}$.

```
>A=[2,4,-1,6;-5:-2;2:-3:-7;0,0,0,-1], b=[4;5;6;7], x=A\ b
 A  =
        2   4  -1   6
       -5  -4  -3  -2
        2  -1  -4  -7
        0   0   0  -1
 b  =
        4
        5
        6
        7
 x  =
       -12.450549
       17.747253
       0.087912
       -7.000000
```

Exemplo 3.2.12. : eig

Retorna o vetor de autovalores da matriz A ou a matriz diagonal L de autovalores de A, junto a uma matriz P que diagonaliza A.

```
>A=[1,0,-1;1,1,1;-1,0,0], eig(A), [P L]=eig(A)
   A  =
          1   0  -1
          1   1   1
         -1   0   0
 ans  =
           1.00000
           1.61803
          -0.61803
   P  =
           0.00000   0.75294    0.40045
           1.00000   0.46534   -0.64794
           0.00000  -0.46534    0.64794
   L  =
         Diagonal Matrix
           1.00000         0          0
                 0   1.61803          0
                 0         0  -0.61803
```

Por fim, algums comandos para matrizes são auto explicativos, não necessitando de exemplos portanto, eles são o `det(A)`, que calcula o determinante da matriz A, o `inv(A)`, calcula a matriz inversa da matriz quadrada A, `rank(A)`, calcula o posto de A, `trace(A)`, calcula o traço, `max`, `min`, `mean` e `median` retornam, respectivamente, o máximo, o mínimo, a média aritmética e a mediana de um vetor. Caso o parâmetro de entrada destes quatro últimos comandos seja uma matriz, o octave retornará um vetor cujas entradas serão o resultado da operação em questão aplicada a cada coluna da matriz. Se A é uma matriz, `A'` faz com que o octave retorne a transposta de A.

3.3 FUNÇÕES, SCRIPTS E ESTRUTURAS DE CONTROLE

O octave vem com uma série de funções pré-programadas, trigonométricas, hiperbólicas, logarítmicas, exponenciais, entre outras, no entanto, é na possibilidade de criar novas funções que enxergamos o quão abrangente este software pode ser.

Uma função é criada através da instrução `function` e recebe como parâmetros de entrada o nome da função, que torna-se o comando pelo qual o usuário pode recorrer a função, as variáveis de entrada e as variáveis de saída. Em seguida, deve-se digitar o comando, ou a cadeia de comandos, que definem a função. Esta instrução é encerrada com a inclusão de `endfunction`. No próximo exemplo veremos como funciona o uso desta instrução.

Exemplo 3.3.1. : Função valor_crítico

Define o valor crítico de uma matriz $A_{m\times n}$.

```
>function k=valor_critico(A)
>[m n]=size(A);
>s=m+n+1;
>k=s-sqrt(s^2-4*m*n);
>endfunction
>#utilização
>A=[2,3,4,5;1,2,3,4;6,5,4,3], y=valor_critico(A)
 A  =
       2  3  4  5
       1  2  3  4
       6  5  4  3

 y  =  4
```

Veremos no capítulo 5 que o valor crítico de uma matriz associada a uma imagem estabelece um parâmetro de sucesso na compressão desta.

É possível salvar uma função para um uso posterior, para isso, usamos o comando `save nome.m nome`, onde `nome` representa o nome da função. Feito isto, o octave criará um arquivo com a extensão .m em seu diretório corrente, ao qual, através de um editor de texto simples como o bloco de notas ou gedit, podemos inserir o código da função com aquele nome e posteriormente acessá-la como qualquer outra função pré-definida no octave. O comando para utilizar a função salva passa a ser o próprio nome da função.

Um *script* é uma cadeia de comandos salvos em um arquivo .m, difere de uma função por não possuir argumentos de entrada ou saída, apenas comandos. O exemplo a seguir contém um script que cria uma matriz retangular de tamanho aleatório entre 1 e 10, com entradas inteiras randômicas compreendidas entre 0 e 100. Para executar tal script, basta digitar "maleatoria" no prompt, nome do arquivo .m onde o script foi salvo.

Exemplo 3.3.2. : Script maleatória
Script que retorna uma matriz de tamanho entre 1 e 10 com entradas entre 0 e 100.

```
# Matriz Aleatória
m=round(9*rand)+1;
n=round(9*rand)+1;
A=round(100*rand(m,n))
```

Na instrução acima vemos o comando `rand` sendo usado sem parâmetros, isto faz com que ele retorne um número aleatório entre 0 e 1. O comando `round(A)` retorna, para cada entrada da matriz A, o número natural mais próximo.

As *estruturas de controle* ocupam um lugar fundamental na utilização do octave. Por meio destas estruturas é possível condicionar a execução de comandos ou o retorno de resultados, também é possível ordenar ao octave que continue a executar um comando enquanto uma determinada condição for satisfeita ou até uma determinada condição ser satisfeita. Alguns comandos que produzem os efeitos mencionados são o `if`, o `for` e o `while` aos quais exemplificaremos sua sintaxe.

Exemplo 3.3.3. : if

Estrutura.

```
>if condição1
>comando1
>elseif condição2
>comando2
>else comando3
>endif
```

Na estrutura acima, caso `condição1` seja satisfeita, o octave executará o `comando1`, caso contrário, o programa testará a `condição2`, satisfeita, ele executa `comando2`. Se as duas primeiras condições não se verificarem, o octave executa o `comando3`. Esta, porém, não é a única sintaxe válida para esta instrução. Podemos inserir mais de um `elseif` ou deixar apenas o `if` com a `condição1`. O argumento `else` também pode ser omitido, caso seja, e não forem satisfeitas nenhuma condição estabelecida pelo `if` ou pelos `elseif`'s, simplesmente não haverá retorno algum.

Exemplo 3.3.4. : Script teste_do_quatro

Testa se x é igual a 4 e exibe a mensagem "x é quatro!", em caso afirmativo. Caso o x seja maior que quatro, exibe "tente um inteiro menor"e caso seja menor do que quatro, exibe "tente um inteiro maior".

```
# No arquivo teste_do_quatro.m, escrevemos
if x==4
disp("x é quatro!")
elseif x>4
disp("tente um inteiro menor")
else disp("tente um inteiro maior")
endif
# No prompt, teremos:
>x=2; teste_do_quatro
tente um inteiro maior
```

O uso do comando `disp` no script acima faz com que o octave exiba a mensagem sem armazená-la no `ans`. O uso de aspas é necessário para que o programa enxergue o que está escrito apenas como uma sequência de caracteres, sem valor numérico algum.

Vemos também o uso das condições `x==4` e `x>4`. Tais relações fazem com que o octave retorne o valor 1, caso a relação seja verdadeira, e o valor 0 caso a relação seja falsa. Podemos combiná-las através dos operadores lógicos "e"e "ou", mediante o uso dos conectivos "`&`"e "—"; ou negá-las em uma proposição mais complexa, utilizando o comando "!". Nas tabelas abaixo, listamos os operadores lógicos e principais relações presentes no octave:

Relações	
Expressão	Retorna 1, quando:
`x==y`	$x = y$
`x<y`	$x < y$
`x>y`	$x > y$
`x>=y`	$x \geq y$
`x<=y`	$x \leq y$
`x!=y`	$x \neq y$

Operadores Lógicos	
Expressão	Retorna 1, quando
`x & y`	$x \neq 0$ e $y \neq 0$
`x | y`	$x \neq 0$ ou $y \neq 0$
`!x`	$x = 0$

Operadores de Incremento	
Expressão	Efeito semelhante a
`++x`	`x=x+1; x`
`x++`	`x, x=x+1;`
`--x`	`x=x-1; x`
`x--`	`x, x=x-1;`

Os operadores de incremento determinam a ordem em que o octave adiciona ou subtrai uma unidade a uma variável e como ele fará uso desta variável. Por exemplo, no comando `x=0, y=1, ++x<y`, o octave retornará para a desigualdade, `ans=0`, pois primeiro foi feito o incremento antes da comparação. Caso tivéssemos escrito `x++<y`, o retorno seria `ans=1`, e no entanto, o valor de `x` seria 1, revelando que houve a comparação antes da incrementação.

Os comandos que definem *laços*, fornecem instruções para que o programa execute um comando até que uma condição seja satisfeita. Construiremos laços através do uso do `for` e do `while`.

Exemplo 3.3.5. : for

Estrutura.

```
>for var=início:fim
>comandos
>endfor
```

A variável `var` será atribuído o valor em `início` e então serão executados `comandos`, em seguida, o octave vai adicionando uma unidade a `var` e executando `comandos` até que sejam executados os comandos para o número em `fim`.

Exemplo 3.3.6. : função eyes

Função que recebe como entrada um inteiro n e cria uma matriz quadrada de ordem n em que os elementos na diagonal secudária são iguais a 1 e os fora dela são nulos.

```
>function X=eyes(n)
>X=zeros(n)
>for i=1:n
>X(i,n+1-i)=1;
>endfor
>endfunction
```

Exemplo 3.3.7. : while

Estrutura.

```
>while condição
>comandos
>endwhile
```

Se a condição em `condição` for satisfeita, executa `comandos` e repete o processo até que `condição` retorne um resultado falso.

Exemplo 3.3.8. : Função teste_da_razao

Toma como argumentos uma matriz A em $\mathbb{R}^{m\times n}$ e um real positivo x e retorna o menor inteiro k para o qual o valor da expressão k*(m+n+1-k)/(m*n) é maior do que x.

```
>function y=teste_da_razao(A,x)
>[m n]=size(A)
>k=1;
>while k*(m+n+1-k)/(m*n)<=x
>k=k+1;
>endwhile
>y=k;
>endfunction
```

Na segunda linha do programa acima atribuímos às variáveis m e n, respectivamente, o número de linhas e de colunas de A. Na terceira, estabelecemos o valor inicial da variável k utilizada no laço `while`. Na sequência (linha 4), o laço é iniciado e será testado a desigualdade $k \cdot (m + n + 1 - k)/m \cdot n \geq x$, que caso seja verdadeira trocamos k por seu sucessor (linha 5) e repetiremos o teste até que a desigualdade se verifique falsa, aí o laço `while` é encerrado através de `endwhile` e fazemos a função retornar o valor de k ao qual a desigualdade se mostrou falsa. Por fim, a função é devidamente encerrada.

CAPÍTULO 4

A ÁLGEBRA DE MATRIZES E A EDIÇÃO DE IMAGENS

O GNU octave naturalmente é capaz de manipular imagens digitais. Dispõe de ferramentas que permitem abrir a matriz de dados de uma imagem a partir de vários formatos, bem como, a partir de uma matriz, salva uma imagem em outros tantos formatos. É possível visualizar uma imagem e fazer a conversão entre diversos sistemas, enfim, com a instalação do pacote *image*, o octave passa a dispor de ferramentas de edição avançadas.

Nosso objetivo aqui, no entanto, é o estudo dos efeitos obtidos através da aplicação da álgebra elementar de matrizes, acessíveis ao aluno e ao professor do ensino médio, por tal motivo, será evitado o uso de ferramentas de edição pré-definidas o tanto quanto possível.

4.1 FERRAMENTAS DE MANIPULAÇÃO

Para abrir uma imagem e armazená-la na variável A, utiliza-se o comando `A=imread ("nome_da_imagem.formato")` e para visualizá-la, `imshow(A)`. Uma imagem armazenada em `A` e salva através da instrução `imwrite(A,"nome_da_ imagem.formato")` e podemos acessar uma descrição técnica da imagem através de `imfinfo`.

Para o octave, imagens coloridas e em escala de cinza são objetos do tipo `uint8`, inteiros de 8 bits, e matrizes não associadas a imagens em escala de cinza ou coloridas são do tipo `double`, reais. Imagens em preto e branco são obtidas através de matrizes do tipo `double`. Uma variável x pode ser transformada em um tipo ou outro fazendo `uint8(x)` ou `double(x)`. A maior diferença entre estes dois tipos de imagem está no retorno de operações aritméticas, pois as do `uint8` retornam o inteiro entre 0 e 255 mais próximo do resultado esperado. Assim, as expressões `8/3`, `uint8(5/3)`, `uint8(8)/3`, `uint8(1000)*1`, `-uint8(10)`, fornecem as respostas `2.6667`, `1`, `3`, `255` e `0`, respectivamente.

É possível converter imagens RGB (padrão do octave) para o sistema HSV usando `rgb2hsv`. O processo inverso é feito com `hsv2rgb`. Nestes dois comandos, a saída é do tipo `double` normalizada. Isto significa que se `B=hsv2rgb(A)` deve-se efetuar `uint8(255*B)` antes de visualizar ou salvar a matriz `B`.

4.2 EDIÇÃO DE IMAGENS EM ESCALA DE CINZA

É possível, de uma imagem colorida, obter uma imagem em escala de cinza mediante inúmeras técnicas. Uma muito comum, é atribuir a cada pixel da imagem cinza uma intensidade dada pela média aritmética das intensidades de suas três componentes RGB. Outra maneira, mais utilizada, é a de atribuir a cada pixel a intensidade de sua luminância. Segundo Gomes e Velho em [1], a luminância L de uma cor $c = (r, g, b)$ é

dada por

$$L = 0,299 \cdot r + 0,587 \cdot g + 0,114 \cdot b$$

Para simplificar, utilizaremos os valores arredondados:

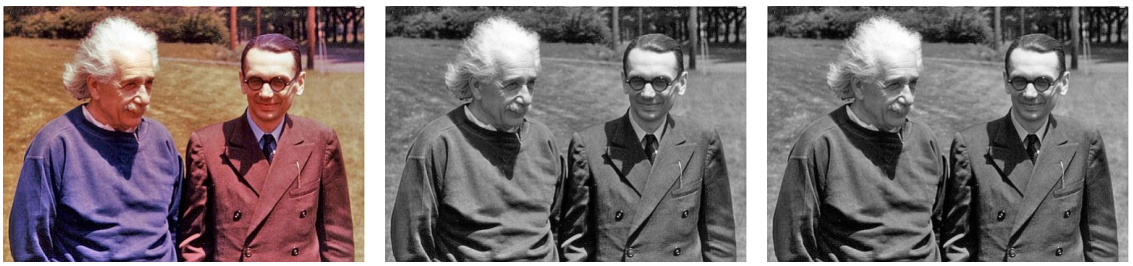
$$L = 0,3 \cdot r + 0,6 \cdot g + 0,1 \cdot b$$

e temos, a seguir, um exemplo de uso destas duas técnicas, junto ao código utilizado para produzi-las.

Exemplo 4.2.1.

Convertendo imagens coloridas em imagens em escala de cinza. **figura 4.1**

```
>I=imread("eg.tiff")
>I=double(I);
>G1=(I(:,:,1)+I(:,:,2)+I(:,:,3))./3;
>imwrite(uint8(G1),"egCinzaMedia.png")
>G2=0.3*I(:,:,1)+0.6*I(:,:,2)+0.1*I(:,:,3);
>imwrite(uint8(G2),"egCinzaLuminacia.png")
```



(a) Original (b) Média (c) Luminância

Figura 4.1

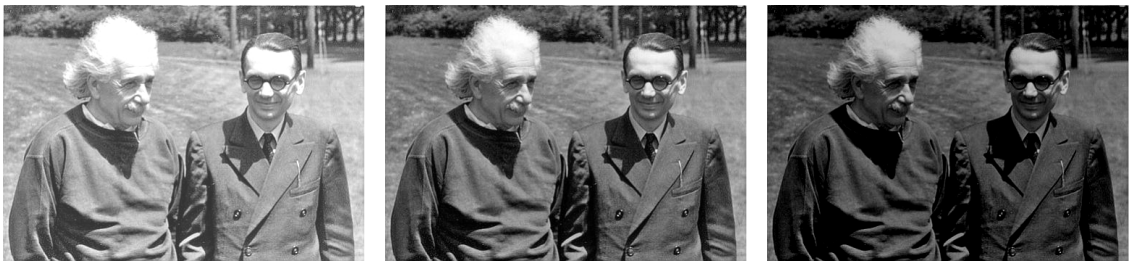
4.2.1 Ajuste do brilho

Intuitivamente, o conceito de *brilho* de uma imagem (ou de uma cor) corresponde ao quão claro ou escuro esta é percebida. Tal conceito subjetivo tem diferentes interpretações em diferentes sistemas de cores; no HSV, por exemplo, podemos aumentar ou diminuir o brilho de uma imagem reduzindo ou aumentando a intensidade dada pelo *Valor* (canal V). O controle do brilho de uma imagem em escala de cinza é feito adicionando ou subtraindo uma constante a cada entrada da matriz de intensidades.

Exemplo 4.2.2.

Aumentando e reduzindo o brilho de uma imagem em 50 níveis. **figura 4.2**

```
>G2=double(G2)
>G2BrilhoAd=G2.+50;
>G2BrilhoSub=G2.-50;
```



(a) Brilho aumentado (b) Original (c) Brilho reduzido

Figura 4.2

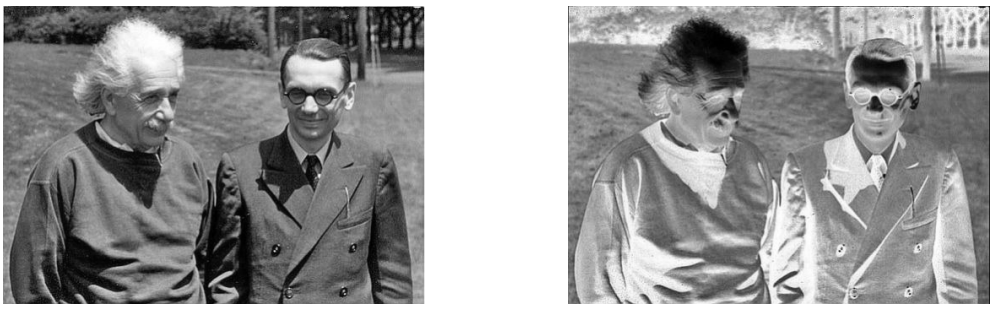
4.2.2 Negativo

O *negativo* de uma imagem corresponde a representação desta com seus valores de intensidades invertidos, isto significa que o percentual de proximidade de um tom de cinza ao branco torna-se, no negativo, o percentual de proximidade deste tom ao preto, e vice-versa. O negativo NI de uma imagem I é obtido fazendo para cada pixel $I(x, y)$ a transformação $NI(x, y) = 255 - I(x, y)$.

Exemplo 4.2.3.

Obtendo o negativo de uma imagem. **Figura 4.3**

```
>G1Negativa=255.-G1;
```



(a) Original (b) Negativo

Figura 4.3

4.2.3 Ajuste do contraste

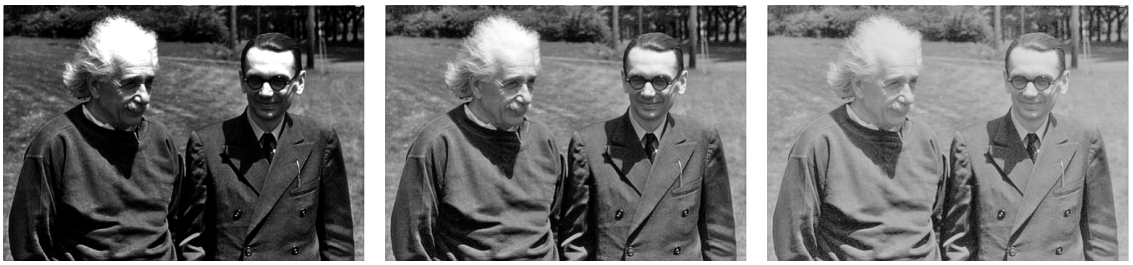
O *contraste* estabelece um distanciamento do conjunto tons de cinza da imagem a um valor pré definido, denominado limiar. O aumento do contraste torna os tons com intensidade acima do limiar, mais claros, e os com intensidade abaixo do limiar, mais escuros. O efeito de aumento de contraste obtém-se dividindo cada entrada da matriz

de intensidades pelo limiar e elevando cada termo da matriz resultante a um expoente maior do que 1. Para reduzir o contraste, elevamos a cada termo a um expoente entre 0 e 1. Vale mencionar que devemos fazer a conversão da imagem para o tipo `double` antes de realizarmos os cálculos, e reconverter os resultados para o tipo `uint8` antes salvá-los.

Exemplo 4.2.4.

Aumentando e reduzindo o contraste de uma imagem por um limiar de 200. **Figura 4.4**

```
>G2=double(G2);
>G2ContrasteAd=((G2./200).^2).*200
>G2ContrasteSub=sqrt(G2./200).*200
```



(a) Contraste aumentado (b) original (c) Contraste reduzido

Figura 4.4

4.2.4 Reflexões e rotações

A transposição de matrizes, junto ao produto por matrizes obtidas pelo comando `eyes` definido no exemplo 3.3.6 são responsáveis por efeitos de *reflexão* em torno de um eixo horizontal, vertical e diagonal na imagem. Combinando duas destas reflexões, obtemos *rotações* de mútiplos de 90°. Se $\tilde{I}_n$ é a matriz obtida mediante o comando `eyes(n)`, os produtos $\tilde{I}_n \cdot A$ e $A \cdot \tilde{I}_n$ produzem, respectivamente, reflexões em torno do eixo horizontal e vertical da imagem A. A transposta de A produz a reflexão da imagem em torno da diagonal principal de A.

Exemplo 4.2.5.

Fazendo reflexões em torno de um eixo horizontal, vertical e diagonal, e em seguida, produzindo rotações. **Figura 4.5**

```
>Im=imread("lena512.bmp");
>RI=eyes(512);
>RI*Im; Im*RI; Im';
>RI*(Im'); RI*Im*RI; (Im')*RI;
```

(a) Original (b) Reflexões (c) Rotações

Figura 4.5

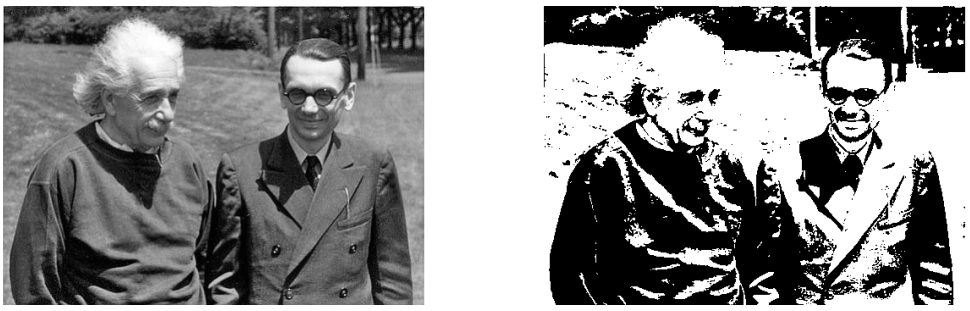
4.2.5 Binarização

A *binarização* é um processo que separa uma imagem em apenas dois tons, preto e branco, segundo algum critério. O mais comum é o thresholding, que atribui a cor branca para valores acima de um limiar, e a cor preta para valores abaixo deste limiar. Se I é uma imagem em escala de cinza obtemos facilmente tal resultado pelo octave através da atribuição $B = I \leq$ limiar.

Exemplo 4.2.6.

Tornando tons abaixo do limiar 150, pretos, e maiores ou iguais a tal limiar, brancos.
Figura 4.6

```
>B=G2>=150
```



(a) original (b) Binarizacao simples

Figura 4.6

Frequentemente, a aplicação da binarização pela definição de um limiar constante reduz muito a qualidade da imagem final. Existem alternativas de binarização que buscam resultados melhores, a um custo computacional maior. Valendo-se da impossibilidade do olho humano distinguir variações de tons em regiões muito pequenas, às quais o cérebro fornece uma informação da média dos tons ali presentes, tais técnicas buscam criar tons de cinza criando padrões branco-preto. Destacaremos o *dithering ordenado*, como um exemplo deste tipo de binarização.

No dithering ordenado dividimos a imagem em regiões quadradas com uma quantidade constante de pixeis, em seguida, comparamos cada uma destas regiões com um módulo

pré-definido. Pixeis com intensidade maior do que a respectiva entrada do módulo são tornados brancos e os com intensidade menor ou igual a do módulo são tornados pretos. A matriz $B1$ abaixo é um exemplo de módulo 2×2 para realização do dithering ordenado

$$B1 = \begin{bmatrix} 0 & 2 \\ 3 & 1 \end{bmatrix}$$

Outros módulos podem ser gerados através da recursão

$$\begin{cases} Y(1)=B1 \\ Y(n+1)=\begin{bmatrix} 4 \cdot Y(n) & 4 \cdot Y(n) + 2 \cdot U(n) \\ 4 \cdot Y(n) + 3 \cdot U(n) & 4 \cdot Y(n) + U(n) \end{bmatrix} \end{cases}$$

Onde, $Y(n)$ é o módulo de dimensões $2^n \times 2^n$ e $U(n)$ corresponde à matriz `ones(2^n)`. Com o fim de criar uma função para realizar o dithering ordenado, fizemos uma função para gerar automaticamente os módulos, dada a ordem da recursão, a qual descrevemos a seguir.

Exemplo 4.2.7. : Função modulo

Toma como argumento um número inteiro positivo n e retorna um módulo de dimensões $2^n \times 2^n$.

```
>function Y=modulo(n)
>if n==1
>Y=[0,2;3,1];
>else
>Y(1:2^(n-1),1:2^(n-1))=4*modulo(n-1);
>Y(1:2^(n-1),2^(n-1)+1:2^n)=4*modulo(n-1).+2;
>Y(2^(n-1)+1:2^n,1:2^(n-1))=4*modulo(n-1).+3;
>Y(2^(n-1)+1:2^n,2^(n-1)+1:2^n)=4*modulo(n-1).+1;
>endif
>endfunction
```

Para efetuar o dithering ordenado em uma imagem I construiremos uma matriz com as mesmas dimensões de I, no entanto, suas entradas serão réplicas do módulo obtidas mediante a instrução `repmap(A,m,n)`, onde A representa um módulo, que cria uma nova matriz replicando `A m` vezes na direção vertical e `n` vezes na horizontal. Tal matriz, será, por fim, multiplicada pelo número `255/2^(2*d)`, onde `2^d` representa a ordem da matriz do módulo.

Exemplo 4.2.8. : Função dito

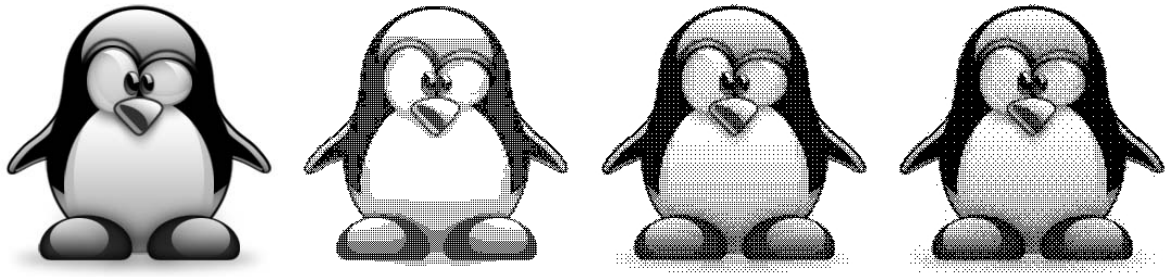
Toma como argumentos uma imagem I e um inteiro positivo d e Efetua o dithering ordenado na imagem usando módulos de dimensões $2^d \times 2^d$.

```
>function Y=dito(I,d)
>[m n]=size(I);
>map_size=ceil([m n]./2^d);
>mod_map=255*repmat(modulo(d),map_size)./(2^(2*d));
>mod_map=mod_map(1:m,1:n);
>Y=(I>mod_map);
>endfunction
```

Exemplo 4.2.9.

Aplicação da função `dito(I,d)` para `d`=1:3. **Figura 4.7**

```
>I=imread("egCinzaLuminancia.tiff");
>DO1=dito(I,1);
>DO2=dito(I,2);
>DO3=dito(I,3);
```



(a) Original (b) Módulo 2×2 (c) Módulo 4×4 (d) Módulo 8×8

Figura 4.7

Resultados melhores de dithering ordenado são obtidos conforme a imagem seja maior.

4.2.6 Transição de imagens

A operação de *cross dissolve* cria um efeito de transição de uma imagem em outra. Dada duas imagens, A e B, ambas com o mesmo tamanho, obtemos a imagem C, resultante do *cross dissolve*, através da expressão

$$C = t \cdot A + (1 - t) \cdot B$$

onde t é um número real no intervalo $[0, 1]$ e indica o percentual de participação da matriz A na transição.

Exemplo 4.2.10.

Aplicando sucessivamente o cross dissolve em duas matrizes. **Figura 4.8**

```
>A=imread("lin256.jpg");
>B=imread("link.jpg");
>for i=1:7
> C(:,:,i)=0.125*i*A+(1-0.125*i)*B;
>endfor
```

(b) transição de imagens

Figura 4.8

Após a execução do programa acima basta efetuar `imwrite(C(:,:,`k`), "lin`k`.jpg")` com k variando de 1 a 7, para o salvamento das imagens.

4.2.7 Marca d'água

Podemos produzir um efeito de marca d'água mediante a multiplicação de matrizes termo a termo. Seja A a matriz associada a imagem na qual será inserida a marca d'água e seja B a matriz associada a imagem da marca d'água propriamente dita. Primeiramente criamos uma matriz C através do comando `ones` que tenha o tamanho de A. Em seguida definimos M como se segue

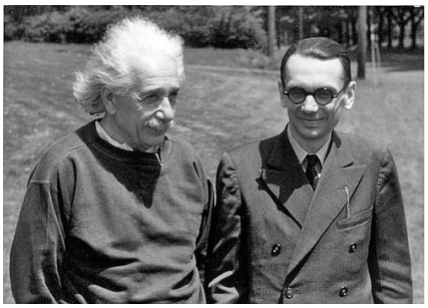
$$M = \frac{1}{max\{B\}} \cdot B$$

onde $max\{B\}$ é a maior entrada de B. Agora atribuímos em C uma região retangular igual a M e efetuamos o produto termo a termo entre C e A para obter o efeito em questão. Para aumentar o efeito de transparência da marca basta aumentar o brilho da matriz B antes de efetuar o processo descrito anteriormente.

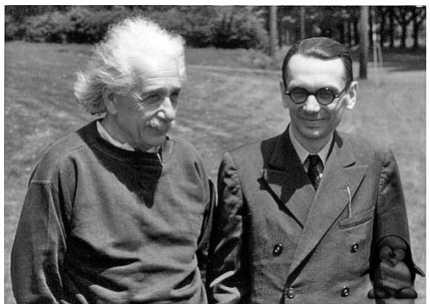
Exemplo 4.2.11.

Inserido como marca d'água a imagem em B no canto inferior direito da imagem em A.
Figura 4.9

```
>A=imread("egCinzaLuminancia.jpg");
>B=double(imread("lin.jpg"));
>C=ones(size(A));
>M=B./max(max(B));
>C(end-rows(B)+1:end,end-columns(B)+1:end)=M;
>A_com_marca=double(C).*double(Im);
```



(a) original



(b) com marca

Figura 4.9

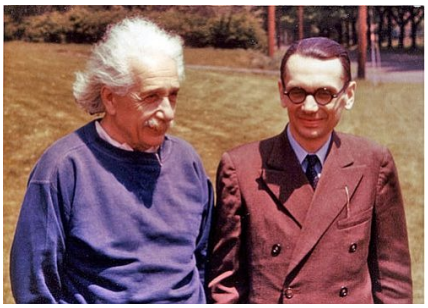
4.3 EDIÇÃO DE IMAGENS COLORIDAS

Os efeitos de manipulação do brilho e do contraste, A reflexão, a rotação de múltiplos de 90°, a binarização, a obtenção do negativo, a transição de imagens e a inclusão de marca d'água podem ser aplicadas em imagens coloridas, bastando efetuar as operações em cada canal RGB. Na figura a seguir, diminuimos o brilho em 50 níveis, aumentamos o contraste a partir de um limiar de 200, efetuamos a reflexão em torno do eixo vertical e incluímos uma marca d'água.

Exemplo 4.3.1.

Aplicando as alterações descritas no parágrafo anterior. **Figura 4.10**

```
>I=imread("eg.tiff");
>E=I.-50;
>E=((double(E)./200).^2).*200;
>B=double(imread("lin.jpg"));
>M=ones(size(E));
>B=B./max(max(B));
>for i=1:3
>E(:,:,i)=E(:,:,i)*eyes(columns(E));
>M(end-rows(B)+1:end,end-columns(B)+1:end,i)=B;
>endfor
>E=M.*E;
>imwrite(uint8(E),"egEditado.tiff")
```



(a) original

(b) Editada

Figura 4.10

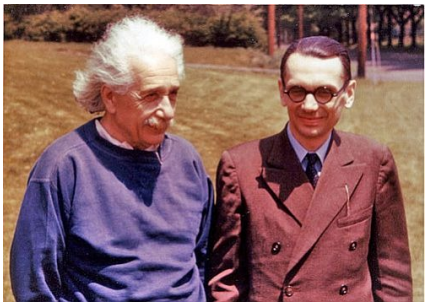
4.3.1 Colorização

Colorizar uma imagem corresponde à representá-la como se fôssemos representar toda a variação de luminância da imagem através de um único canal de cor, tal como se víssemos a imagem através de um vitral colorido. Para obter a colorização C de uma imagem I, escolhemos uma cor c e criamos uma imagem A com as dimensões de I contendo apenas a cor c, em seguida, se L é a matriz de luminância de I, efetuamos o produto termo a termo entre $L/255$ e A.

Exemplo 4.3.2.

Colorizando a imagem I na cor $c = (128, 255, 64)$. **Figura 4.11**

```
>I=double(imread("eg.jpg"));
>L=(0.3*I(:,:,1)+0.6*I(:,:,2)+0.1*I(:,:,3));
>A(:,:,1)=128*ones(size(L));
>A(:,:,2)=255*ones(size(L));
>A(:,:,3)=64*ones(size(L));
>C=A.*(L./255);
```



(a) original

(b) colorizada

Figura 4.11

4.3.2 Decomposição RGB e CMY

Para representar a decomposição de uma imagem em suas três componentes RGB, tal como foi feito na figura 2.3, mantemos a componente que se deseja visualizar e tornamos as demais iguais a matriz nula. Para uma representação de como cada componente CMY é impressa no papel, tal como foi feito na figura 2.5(c), observamos inicialmente que o octave salva e exibe imagens no sistema RGB com entradas do tipo `uint8`, logo, o objetivo aqui é visualizar em RGB o que seria impresso no papel em CMY. Imprimir uma cor $(c, 0, 0)$ no papel branco, por exemplo, é fazer com que este perca $255 - c$ níveis de luminância na componente vermelha, o que no RGB corresponde a cor $(255 - c, 255, 255) = (r, 255, 255)$. Assim, representamos a impressão de cada componente CMY no papel mantendo os valores do canal de sua cor oposta e fazendo as demais assumirem o valor máximo.

Exemplo 4.3.3.

Obtendo a componente verde e a componente magenta da imagem I.

```
>clear G, clear M
>G(:,:,1)=zeros(size(I(:,:,1)));
>G(:,:,2)=I(:,:,2);
>G(:,:,3)=zeros(size(I(:,:,1)));
>imwrite(uint8(G),"egG.tiff")
>M(:,:,1)=255*ones(size(I(:,:,1)));
>M(:,:,2)=I(:,:,2);
>M(:,:,3)=255*ones(size(I(:,:,3)));
>imwrite(uint8(M),"egM.tiff")
```

(a) (b)

Figura 4.12

4.3.3 Posterização

A *posterização* consiste em reduzir o número de cores de uma imagem. O processo de binarização de cada componente de uma imagem é um exemplo de posterização. A função `posteriza(X,k)` implementada aqui, separa cada componente RGB em `n` faixas de cores e enquadra cada pixel de `X` em uma destas faixas. Abaixo temos a descrição da função seguida de exemplos de uso.

Exemplo 4.3.4. : Função posteriza

Separa cada componente da imagem X em k níveis.

```
>function Y=posteriza(X,k)
>A=double(X); [m n c]=size(X);
>for i=1:c
>Y(:,:,i)=uint8(floor(k.*A(:,:,i)./256).*(255/(k-1)));
>endfor
>endfunction
```

Exemplo 4.3.5.

Posterizando uma imagem em 2, 4 e 8 níveis. **Figura 4.13**

```
>I=imread("eg.tiff");
>POST2=posteriza(I,2);
>POST4=posteriza(I,4);
>POST8=posteriza(I,8);
```

(a) 2 niveis (b) 4 niveis (c) 8 níveis

Figura 4.13

Este Capítulo encerra a primeira etapa deste trabalho ao reunir um conjunto de ferramentas básicas para exploração do octave e do processamento de imagens. Aqui foi evidente o intenso uso da multiplicação termo a termo de matrizes, a multiplicação por escalar e a adição de matrizes em expressões do tipo `X.+c` que são idênticas às `X+c*ones(size(X))`. O produto de matrizes, visto que, ao fornecer resultados dependentes das linhas e colunas das matrizes multiplicadas, foi pouco utilizado nas ferramentas expostas até o momento, uma vez que na edição ou pós processamento de imagens a maioria das operações são feitas mediante a transformação da cor de um pixel em uma outra cor dependendo apenas da intensidade da cor de entrada. O produto de matrizes, no entanto, conduzirá os resultados obtidos nos próximos capítulos, visto que o método de compressão de imagens digitais posteriormente apresentado utiliza-se da visualização de uma matriz como um produto de matrizes passíveis de simplificação.

CAPÍTULO 5

BASES MATEMÁTICAS PARA A COMPRESSÃO DE IMAGENS DIGITAIS

5.1 DECOMPOSIÇÃO EM VALORES SINGULARES

Os métodos de compressão de imagens digitais apresentados no próximo capítulo baseiam-se na possibilidade de decompor uma matriz em um produto de três matrizes, duas ortogonais e uma diagonal generalizada, o qual torna passível a obtenção de uma matriz com uma quantidade de dados reduzida e, no entanto, dentre todas as matrizes com a mesma quantidade de dados, esta nova matriz obtida é, em certo sentido, a mais próxima possível da matriz original. Demonstrar tal resultado é o objetivo deste capítulo e o conceito de *decomposição em valores singulares* governará esta meta.

Definição 5.1. *Uma matriz $A_{m\times n}$ é dita diagonal generalizada se possuir todos as suas entradas fora da diagonal principal, nulas.*

Definição 5.2. *Dada uma matriz $m \times n$ A, uma decomposição $A = U\mathcal{S}V^{\mathsf{T}}$ é dita uma decomposição em valores singulares se U e V são matrizes ortogonais, e $\mathcal{S}$ é uma matriz diagonal generalizada cujas entradas na diagonal principal são não-negativas.*

Iremos nos referir a decomposição em valores singulares de A como a *SVD* de A, sigla para *Singular Value Decomposition*. É notável a semelhança entre a decomposição em valores singulares e a decomposição espectral, garantida para toda matriz simétrica pelo teorema espectral, como enunciado na sequência.

Teorema 5.1 (Teorema espectral)**.** *Toda matriz simétrica é diagonalizável por matriz ortogonal, em outras palavras, dada uma matriz simétrica $A \in \mathbb{R}^{m\times m}$, existem uma matriz ortogonal P e uma matriz diagonal $\mathcal{E}$ tais que $A = P\mathcal{E}P^{\mathsf{T}}$.*

Decorre do teorema acima que as entradas na diagonal principal de $\mathcal{E}$ formam o conjunto de autovalores de A, e os vetores colunas de P constituem uma base ortonormal de autovetores de A associados aos autovalores em $\mathcal{E}$, correspondendo a i-ésima coluna de P ao i-ésimo autovalor em $\mathcal{E}$. O teorema espectral é um dos principais resultados da álgebra linear, sua demonstração foge ao escopo deste trabalho e será omitida aqui. No entanto, ela pode ser encontrada em [4] e [3]. Junto ao teorema espectral, o conceito de matrizes não negativas constituem os fundamentos para a demonstração da existência da *SVD* de uma matriz. A sequência de definições e proposições a seguir baseia-se na exposição feita em [2] e a próxima definição contém o conceito de matrizes não negativas, mas antes de fazê-la, por motivo de clareza, faremos uma breve exposição da notação adotada.

Tal como no octave, não faremos neste texto distinção entre espaços vetoriais isomorfos, ficando o $\mathbb{R}^{m\times n}$, espaço das matrizes $m \times n$ de dimensão finita sobre $\mathbb{R}$, como o representante de algum espaço vetorial de dimensão finita. Assim, consideraremos $\mathbb{R}^{1\times 1} = \mathbb{R}$ e $\mathbb{R}^{m\times n} = \mathcal{L}(E, F)$ ($\mathcal{L}(E, F)$ é o espaço das transformações lineares $E \to F$ tais que $\dim E = n$ e $\dim F = m$), por exemplo. Um vetor (coluna) é um elemento de $\mathbb{R}^{m\times 1} = \mathbb{R}^m$ e dado um vetor $\mathbf{u} \in \mathbb{R}^m$ chamaremos de vetor linha ao elemento $\mathbf{u}^\mathsf{T}$ de $\mathbb{R}^{1\times m}$. Neste contexto, uma transformação linear $\mathbf{v} \mapsto A(\mathbf{v})$, um produto interno $\langle \mathbf{u}, \mathbf{v}\rangle$ e uma forma quadrática $Q(\mathbf{x})$ será o dados pelos produtos matriciais $A\mathbf{v}$, $\mathbf{u}^\mathsf{T}\mathbf{v}$ e $\mathbf{x}^\mathsf{T}A\mathbf{x}$, respectivamente. Doravante, usaremos $\mathcal{G}B$ para indicar o espaço gerado pelo conjunto B de vetores, $\mathcal{P}(A)$ para denotar o posto de uma matriz $A_{m\times n}$ e a letra grega μ representa $\min\{m, n\}$, donde segue que $\mathcal{P}(A) \leq \mu$, $\forall A \in \mathbb{R}^{m\times n}$. O escalar zero, o vetor nulo e a matriz nula serão respectivamente escritos como 0, $\mathbf{0}$ e O.

Definição 5.3. *Uma matriz simétrica $A \in \mathbb{R}^{n\times n}$ é dita não negativa (respec. positiva) e denotada por $A \geq 0$ (respec. $A > 0$) se for simétrica e se $\mathbf{x}^\mathsf{T}A\mathbf{x} \geq 0$ (respec. $\mathbf{x}^\mathsf{T}A\mathbf{x} > 0$) para todo $\mathbf{x} \in \mathbb{R}^n$.*

O teorema seguinte relaciona matrizes não negativas e positivas ao sinal de seus autovalores.

Proposição 5.1. *Uma matriz simétrica $A \in \mathbb{R}^{m\times n}$ é não negativa se, e somente se, tem todos os seus autovalores não negativos. Ela será positiva se, e somente se, todos os seus autovalores forem positivos.*

Demonstração: Supondo $A \geq 0$, sejam $\lambda \in \mathbb{R}$ e $\mathbf{v} \in \mathbb{R}^n$ um vetor não nulo, tais que $A\mathbf{v} = \lambda\mathbf{v}$. Por definição

$$\mathbf{v}^\mathsf{T}A\mathbf{v} \geq 0 \Rightarrow \mathbf{v}^\mathsf{T}(\lambda\mathbf{v}) \geq 0 \Rightarrow \lambda(\mathbf{v}^\mathsf{T}\mathbf{v}) \geq 0$$

como $\mathbf{v}^\mathsf{T}\mathbf{v} = \|\mathbf{v}\|^2 \geq 0$, segue que $\lambda \geq 0$. Por outro lado, o teorema espectral garante a existência de uma base ortonormal de autovetores de A, $\mathbf{u}_1, \ldots, \mathbf{u}_n$, associados aos autovalores não negativos $\lambda_1, \ldots, \lambda_n$, respectivamente. Dado $\mathbf{v} \in \mathbb{R}^n$, existem escalares $\alpha_1, \ldots, \alpha_n$ tais que

$$\mathbf{v} = \sum_{i=1}^{n} \alpha_i \mathbf{u}_i \quad \text{e} \quad A\mathbf{v} = \sum_{j=1}^{n} \lambda_j \alpha_j \mathbf{u}_j$$

Assim

$$\begin{aligned}
\mathbf{v}^\mathsf{T}A\mathbf{v} &= \left(\sum_{i=1}^{n} \alpha_i \mathbf{u}_i\right)^\mathsf{T} \left(\sum_{j=1}^{n} \lambda_j \alpha_j \mathbf{u}_j\right) \\
&= \left(\sum_{i=1}^{n} \alpha_i \mathbf{u}_i^\mathsf{T}\right) \left(\sum_{j=1}^{n} \lambda_j \alpha_j \mathbf{u}_j\right) \\
&= \sum_{i=1}^{n} \alpha_i \mathbf{u}_i^\mathsf{T} \sum_{j=1}^{n} \lambda_j \alpha_j \mathbf{u}_j \\
&= \sum_{i,j=1}^{n} \lambda_j \alpha_i \alpha_j \mathbf{u}_i^\mathsf{T}\mathbf{u}_j
\end{aligned}$$

como $\mathbf{u}_i^\mathsf{T}\mathbf{u}_j = 0$ sempre que $i \neq j$, a última igualdade acima torna-se

$$\mathbf{v}^\mathsf{T} A\mathbf{v} = \sum_{i=1}^{n} \lambda_i \alpha_i^2 \mathbf{u}_i^\mathsf{T}\mathbf{u}_i = \sum_{i=1}^{n} \lambda_i \alpha_i^2 \|\mathbf{u}_i\|^2 = \sum_{i=1}^{n} \lambda_i \alpha_i^2$$

cujo termo geral de $\sum_{i=1}^{n} \lambda_i \alpha_i^2$ é não negativo, visto que $\lambda_i \geq 0$. Segue que $\mathbf{v}^\mathsf{T} A\mathbf{v} \geq 0$ e portanto $A \geq 0$. A demonstração para o caso positivo é idêntica, bastando substituir o $\geq$ por $>$. □

Sejam $\mathbf{a}_1, \ldots, \mathbf{a}_n$ os vetores coluna de uma matriz $A \in \mathbb{R}^{m\times n}$. A matriz $A^\mathsf{T}A = [c_{ij}]_{m\times n}$ tem a sua ij-ésima coordenada dada por $\mathbf{a}_i^\mathsf{T}\mathbf{a}_j$, o produto interno entre $\mathbf{a}_i$ e $\mathbf{a}_j$, por isso, pode-se afirmar que $A^\mathsf{T}A$ reúne toda informação da métrica entre os vetores $\mathbf{a}_i$. Analogamente, AA^T reúne informações a respeito dos vetores linhas de A. De fato, os próximos teoremas nos mostram que o estudo de $A^\mathsf{T}A$ e de AA^T revela propriedades da própria matriz A.

Proposição 5.2. *Dada $A \in \mathbb{R}^{m\times n}$, a matriz $A^\mathsf{T}A$ é não negativa.*

Demonstração: A matriz $A^\mathsf{T}A$ é simétrica, a saber

$$(A^\mathsf{T}A)^\mathsf{T} = A^\mathsf{T}(A^\mathsf{T})^\mathsf{T} = A^\mathsf{T}A$$

e para um dado vetor $\mathbf{v} \in \mathbb{R}^m$

$$\mathbf{v}^\mathsf{T} A^\mathsf{T} A\mathbf{v} = (A\mathbf{v})^\mathsf{T} A\mathbf{v} = \|A\mathbf{v}\|^2 \geq 0$$

por conseguinte, $A^\mathsf{T}A \geq 0$. □

Para a próxima proposição usaremos o conceito de *complemento ortogonal* de um subespaço vetorial W de V, usualmente representado por $W^\perp$, e correspondente ao conjunto de vetores de V ortogonais aos vetores em W. O conjunto $W^\perp$ goza das seguintes propriedades:

i) $W^\perp$ é um subespaço de V;

ii) $W \cup W^\perp = \{0\}$;

iii) $(W^\perp)^\perp = W$.

O uso do complemento ortogonal na próxima proposição se dará, tanto pela aplicação das propriedades acima, quanto pelo lema apresentado a seguir.

Lema 5.1.1. *A imagem de uma matriz qualquer é perpendicular ao núcleo de sua transposta.*

Demonstração: Em outras palavras, iremos mostrar que para toda matriz $A \in \mathbb{R}^{m\times n}$ tem-se $\mathcal{N}(A^\mathsf{T}) = \mathcal{I}m(A)^\perp$.

$$\begin{aligned}
\mathbf{v} \in \mathcal{N}(A^\mathsf{T}) &\Leftrightarrow A^\mathsf{T}\mathbf{v} = \mathbf{0} \\
&\Leftrightarrow \forall \mathbf{u} \in \mathbb{R}^m;\ (A^\mathsf{T}\mathbf{v})^\mathsf{T}\mathbf{u} = 0 \\
&\Leftrightarrow \forall \mathbf{u} \in \mathbb{R}^m;\ \mathbf{v}^\mathsf{T}A\mathbf{u} = 0 \\
&\Leftrightarrow \mathbf{v} \in \mathcal{I}m(A)^\perp.
\end{aligned}$$

como queríamos mostrar. □

Proposição 5.3. *Para qualquer matriz $A \in R^{m\times n}$ o posto de $A^\mathsf{T}A$ é igual ao posto de A.*

Demonstração: Primeiramente, nota-se que $\mathcal{N}(A^\mathsf{T}A) = \mathcal{N}(A)$, com efeito, dado $\mathbf{v} \in \mathcal{N}(A^\mathsf{T}A)$, tem-se

$$\begin{aligned}
A^\mathsf{T}A\mathbf{v} = 0 &\Rightarrow A\mathbf{v} \in \mathcal{N}(A^\mathsf{T}) = \mathcal{I}m(A)^\perp \\
&\Rightarrow A\mathbf{v} \in \mathcal{I}m(A) \cap \mathcal{I}m(A)^\perp = \{0\} \\
&\Rightarrow A\mathbf{v} = 0 \\
&\Rightarrow \mathbf{v} \in \mathcal{N}(A)
\end{aligned}$$

Por outro lado

$$\mathbf{v} \in \mathcal{N}(A) \Rightarrow A\mathbf{v} = 0 \Rightarrow A^\mathsf{T}A\mathbf{v} = 0 \Rightarrow \mathbf{v} \in \mathcal{N}(A^\mathsf{T}A)$$

e daí $\mathcal{N}(A^\mathsf{T}A) = \mathcal{N}(A)$. Prosseguindo com a demonstração, o teorema do núcleo e da imagem garante

$$\begin{aligned}
\dim\mathcal{N}(A^\mathsf{T}A) + \dim\mathcal{I}m(A^\mathsf{T}A) &= m &\Leftrightarrow \\
\dim\mathcal{N}(A) + \mathcal{P}(A^\mathsf{T}A) &= m &\Leftrightarrow \\
\mathcal{P}(A^\mathsf{T}A) &= m - \dim\mathcal{N}(A) &\Leftrightarrow \\
\mathcal{P}(A^\mathsf{T}A) &= \mathcal{P}(A)
\end{aligned}$$

□

É importante salientar que as duas últimas proposições continuam válidas caso tomemos AA^T no lugar de $A^\mathsf{T}A$. Com efeito, dada $A \in \mathbb{R}^{m\times n}$ sempre podemos obter sua transposta A^T. Em particular, as proposições acima garantem que $(A^\mathsf{T})^\mathsf{T}A = AA^\mathsf{T}$ é não negativa e vale $\mathcal{P}(AA^\mathsf{T}) = \mathcal{P}(A^\mathsf{T}) = \mathcal{P}(A)$. Logo, dada A tem-se AA^T não negativa e $\mathcal{P}(AA^\mathsf{T}) = \mathcal{P}(A)$.

Enfim, prossigamos com o Teorema da Decomposição em Valores Singulares.

Teorema 5.2 (Teorema da decomposição em valores singulares)**.** *Toda matriz possui uma decomposição em valores singulares.*

Demonstração: Seja $A \in \mathbb{R}^{m\times n}$ uma matriz qualquer. Devemos provar que existem matrizes U, V e $\mathcal{S}$ tais que U e V sejam ortogonais, $\mathcal{S} = [s_{ij}]_{m\times n}$, com $s_{ii} \geq 0$ e $s_{ij} = 0$ sempre que $i \neq j$, e $A = U\mathcal{S}V^{\mathsf{T}}$.

A matriz $A^{\mathsf{T}}A \in \mathbb{R}^{n\times n}$ é simétrica e não negativa, decorre do teorema espectral que existe uma base de autovetores ortonormais $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ tal que $A^{\mathsf{T}}A\mathbf{v}_i = \lambda_i\mathbf{v}_i$, e sendo $\mathcal{P}(A^{\mathsf{T}}A) = r$, podemos nomear os λ_i de maneira que $\lambda_i > 0$, para $1 \leq i \leq r$ e $\lambda_i = 0$ para $r \leq i \leq n$.

Os vetores do $\mathbb{R}^m$ $A\mathbf{v}_1, \ldots, A\mathbf{v}_n$ são ortogonais dois a dois, pois, para $i \neq j$

$$(A\mathbf{v}_i)^{\mathsf{T}}(A\mathbf{v}_j) = \mathbf{v}_i^{\mathsf{T}}A^{\mathsf{T}}A\mathbf{v}_j = \mathbf{v}_i^{\mathsf{T}}(\lambda_j\mathbf{v}_j) = \lambda_j\mathbf{v}_i^{\mathsf{T}}\mathbf{v}_j = 0.$$

Além disso, temos $\|A\mathbf{v}_i\|^2 = (A\mathbf{v}_i)^{\mathsf{T}}(A\mathbf{v}_i) = \lambda_i\|\mathbf{v}_i\|^2 = \lambda_i$ e portanto, $\|A\mathbf{v}_i\| = \sqrt{\lambda_i} = \sigma_i$.

Por fim, tomando $A\mathbf{v}_i = \sigma_i\mathbf{u}_i$, para $i \leq r$, obtemos um conjunto $\{\mathbf{u}_1, \ldots, \mathbf{u}_r\}$ de vetores ortonormais, ao qual, caso $r < m$, pode ser ampliado a uma base ortogonal $\{\mathbf{u}_1, \ldots, \mathbf{u}_m\}$ do $\mathbb{R}^m$, mediante o algoritmo de *Gram-Schmidt*.

Fazendo $U = \begin{bmatrix} \mathbf{u}_1 & \cdots & \mathbf{u}_m \end{bmatrix}$, $V = \begin{bmatrix} \mathbf{v}_1 & \cdots & \mathbf{v}_n \end{bmatrix}$ e $\mathcal{S} = [s_{ij}]_{m\times n}$ com $s_{ii} = \sigma_i$ e $s_{ij} = 0$, caso $i \neq j$, as considerações acima garantem que

$$\begin{bmatrix} A\mathbf{v}_1 & \cdots & A\mathbf{v}_r & A\mathbf{v}_{r+1} & \cdots & A\mathbf{v}_n \end{bmatrix} = \begin{bmatrix} \sigma_1\mathbf{u}_1 & \cdots & \sigma_r\mathbf{u}_r & \mathbf{0} & \cdots & \mathbf{0} \end{bmatrix} \Leftrightarrow$$

$$A\begin{bmatrix} \mathbf{v}_1 & \cdots & \mathbf{v}_n \end{bmatrix} = \begin{bmatrix} \mathbf{u}_1 & \cdots & \mathbf{u}_m \end{bmatrix} \begin{bmatrix} \sigma_1 & & & & \\ & \ddots & & & \\ & & \sigma_r & & \\ & & & \ddots & \\ & & & & 0 \end{bmatrix}_{m\times n}.$$

Na última matriz acima, caso $r < n$, as $n - r$ colunas não aludidas serão nulas e no caso de $r < m$, as $m - r$ últimas linhas serão nulas. Quando $r = m$ ($r = n$) as referidas linhas (colunas) não existirão. Em todo caso, teremos

$$AV = U\mathcal{S} \Leftrightarrow AVV^{\mathsf{T}} = U\mathcal{S}V^{\mathsf{T}} \Leftrightarrow A = U\mathcal{S}V^{\mathsf{T}}$$

o que conclui a demonstração. □

Os vetores $\mathbf{u}_1, \ldots, \mathbf{u}_m$ são os *vetores singulares esquerdos* de A, já $\mathbf{v}_1, \ldots, \mathbf{v}_n$ são os *vetores singulares direitos* de A.

Os números $\sigma_1, \ldots, \sigma_2$ são chamados de *valores singulares* da matriz A. Sem risco de perda para generalidade, daqui para frente convencionaremos que $\sigma_1 \geq \sigma_2 \geq \ldots \sigma_r$ e usaremos a notação $\sigma_i(A)$ para indicar o i-ésimo maior valor singular de A, tal escrita é particularmente útil para referirmo-nos a valores singulares de matrizes distintas. Consideraremos ainda $\sigma_i(A) = 0$ para todo $i > \mathcal{P}(A)$. Quando necessário incluir os valores singulares nulos em uma listagem escreveremos $\sigma_1, \ldots, \sigma_\mu$. Vejamos na prática como é feito para a obtenção da decomposição em valores singulares de uma matriz.

Exemplo 5.1.1. *Vamos calcular os valores singulares e os vetores singulares direitos e esquerdos da matriz A dada abaixo.*

$$A = \begin{bmatrix} 7 & 1 \\ 0 & 0 \\ 5 & 5 \end{bmatrix}$$

Calculando $A^{\mathsf{T}}A$ obtemos a matriz

$$A^{\mathsf{T}}A = \begin{bmatrix} 74 & 32 \\ 32 & 26 \end{bmatrix}$$

cujas raízes de seu polinômio característico $\lambda^2 - 100\lambda + 900$ são $\lambda_1 = 90$ e $\lambda_2 = 10$ e portanto, os valores singulares de A são $\sigma_1 = 3\sqrt{10}$ e $\sigma_2 = \sqrt{10}$. Como visto no teorema 5.1.2, os autovetores de $A^{\mathsf{T}}A$ constituem os vetores singulares direitos de A, tais autovetores são os vetores unitários obtidos através da solução dos sistemas de equações lineares

$$\begin{bmatrix} -16 & 32 \\ 32 & -64 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \mathbf{0} \qquad \text{e} \qquad \begin{bmatrix} 64 & 32 \\ 32 & 16 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \mathbf{0},$$

donde encontramos, respectivamente

$$\mathbf{v}_1 = \begin{bmatrix} 2/\sqrt{5} \\ 1/\sqrt{5} \end{bmatrix} \qquad \text{e} \qquad \mathbf{v}_2 = \begin{bmatrix} 1/\sqrt{5} \\ -2/\sqrt{5} \end{bmatrix}$$

como os dois vetores singulares direitos de A. Para calcular os vetores singulares esquerdos, observemos que $A\mathbf{v}_1 = \sigma_1\mathbf{u}_1$ e $A\mathbf{v}_2 = \sigma_2\mathbf{u}_2$ e por conseguinte

$$\mathbf{u}_1 = \frac{1}{3\sqrt{10}} \begin{bmatrix} 7 & 1 \\ 0 & 0 \\ 1 & 5 \end{bmatrix} \begin{bmatrix} 2/\sqrt{5} \\ 1/\sqrt{5} \end{bmatrix} = \begin{bmatrix} \sqrt{2}/2 \\ 0 \\ \sqrt{2}/2 \end{bmatrix}$$

e

$$\mathbf{u}_2 = \frac{1}{\sqrt{10}} \begin{bmatrix} 7 & 1 \\ 0 & 0 \\ 1 & 5 \end{bmatrix} \begin{bmatrix} 1/\sqrt{5} \\ -2/\sqrt{5} \end{bmatrix} = \begin{bmatrix} \sqrt{2}/2 \\ 0 \\ -\sqrt{2}/2 \end{bmatrix};$$

e fazendo

$$\mathbf{u}_3 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

teremos completado uma base ortonormal do $\mathbb{R}^3$ e encontrado todos os vetores singulares esquerdos de A, possibilitando-nos escrever

$$A = \begin{bmatrix} \sqrt{2}/2 & \sqrt{2}/2 & 0 \\ 0 & 0 & 1 \\ \sqrt{2}/2 & -\sqrt{2}/2 & 0 \end{bmatrix} \begin{bmatrix} 3\sqrt{10} & 0 \\ 0 & \sqrt{10} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 2/\sqrt{5} & 1/\sqrt{5} \\ 1/\sqrt{5} & -2/\sqrt{5} \end{bmatrix}.$$

No octave, o comando `svd` é responsável por encontrar os valores singulares de uma matriz, e ainda fornecer os vetores singulares direitos e esquerdos.

Exemplo 5.1.2. : svd
Encontrando os valores singulares da matriz `A` e armazenando nas variáveis `U`, `S` e `V` as matrizes da *SVD* de `A`.

```
>A=[7,1;0,0;5,5], svd(A), [U S V]=svd(A)
   A  =
       7  1
       0  0
       5  5
 ans  =
        9.4868
        3.1623
   U  =
       -0.70711   0.70711   0.00000
        0.00000   0.00000  -1.00000
       -0.70711  -0.70711   0.00000
   S  =
      Diagonal Matrix
          9.4868         0
               0    3.1623
               0         0
   V  =
       -0.89443   0.44721
       -0.44721  -0.89443
```

Os diferentes resultados obtidos manualmente e mediante o uso do octave indicam que a *SVD* de uma matriz não é única, entretanto, seus valores singulares são únicos, visto que são raízes quadradas dos autovalores de $A^{\mathsf{T}}A$ (que por sua vez são únicos). As matrizes $A^{\mathsf{T}}A$ e AA^{T} possuem os mesmos autovalores não nulos, com autovetores dados pelas colunas de V e de U respectivamente. Com efeito, dada $A = U\mathcal{S}V^{\mathsf{T}}$ tem-se $A^{\mathsf{T}}A = V\mathcal{S}^{\mathsf{T}}\mathcal{S}V^{\mathsf{T}}$ e $AA^{\mathsf{T}} = U\mathcal{S}\mathcal{S}^{\mathsf{T}}U^{\mathsf{T}}$, donde

$$\mathcal{S}^{\mathsf{T}}\mathcal{S} = \begin{bmatrix} \sigma_1^2 & & & & \\ & \ddots & & & \\ & & \sigma_r^2 & & \\ & & & \ddots & \\ & & & & 0 \end{bmatrix}_{n\times n} \quad \text{e} \quad \mathcal{S}\mathcal{S}^{\mathsf{T}} = \begin{bmatrix} \sigma_1^2 & & & & \\ & \ddots & & & \\ & & \sigma_r^2 & & \\ & & & \ddots & \\ & & & & 0 \end{bmatrix}_{m\times m}$$

e portanto, $A^{\mathsf{T}}A$ e AA^{T} possuem os mesmo conjunto de autovalores. Daí segue que A e A^{T} possuem os mesmos valores singulares.

Se $U\mathcal{S}V^{\mathsf{T}}$ é uma *SVD* de uma matriz A de posto r, então poderemos usar a notação

$$A = \sum_{i=1}^{\mu} \sigma_i \mathbf{u}_i \mathbf{v}_i^{\mathsf{T}} \qquad \text{ou} \qquad A = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^{\mathsf{T}}$$

para referenciar a *SVD* de A.

5.2 NORMAS E SVD

Assegurada a existência da *SVD* de uma matriz qualquer, voltemo-nos ao estudo das normas espectral e de Frobenius (ou euclidiana) profundamente relacionadas com a *SVD* e imprescindíveis para o estudo conduzido aqui.

Definição 5.4. *Dada uma matriz $A \in \mathbb{R}^{m\times n}$, denotaremos sua norma espectral por $\|A\|_2$, definida como*

$$\|A\|_2 = \max_{\mathbf{x}\in\mathbb{R}^n} \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|}, \text{ com } \quad \mathbf{x} \neq \mathbf{0}.$$

A norma espectral de A (também chamada de norma 2) sinaliza qual é o maior fator de dilatação de um vetor quando submetido a aplicação definida por A. Entendemos aqui como fator de dilatação um escalar ϵ tal que $\|A\mathbf{v}\| = \epsilon\|\mathbf{v}\|$.

O fator de dilatação de um vetor depende apenas de sua direção, possibilitando-nos escrever

$$\|A\|_2 = \max_{\|\mathbf{x}\|=1} \|A\mathbf{x}\| \quad (*)$$

com efeito, dado o vetor unitário $\mathbf{v}$, o fator de dilatação ϵ de um vetor $\mathbf{x}$ na direção de $\mathbf{v}$ é dado por

$$\epsilon = \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|} = \frac{\|A\delta\mathbf{v}\|}{\|\delta\mathbf{v}\|} = \frac{|\delta|\|A\mathbf{v}\|}{|\delta|\|\mathbf{v}\|} = \|A\mathbf{v}\|$$

Na próxima proposição, usaremos a expressão (*) no lugar da definição da norma espectral.

Teorema 5.3. *Para toda matriz $A \in \mathbb{R}^{m\times n}$ com valores singulares $\sigma_1, \cdots, \sigma_\mu$ tem-se*

$$\|A\|_2 = \sigma_1.$$

Demonstração: Seja $U\mathcal{S}V^\mathsf{T}$ a *SVD* de A. Por definição, dado um vetor unitário $\mathbf{x}$ temos

$$\|A\mathbf{x}\|^2 = (U\mathcal{S}V^\mathsf{T}\mathbf{x})^\mathsf{T}(U\mathcal{S}V^\mathsf{T}\mathbf{x}) = \mathbf{x}^\mathsf{T}V\mathcal{S}^\mathsf{T}U^\mathsf{T}U\mathcal{S}V^\mathsf{T}\mathbf{x} = \mathbf{x}^\mathsf{T}V\mathcal{S}^\mathsf{T}\mathcal{S}V^\mathsf{T}\mathbf{x}$$

Fazendo $\mathbf{y} = V^\mathsf{T}\mathbf{x}$ notamos que $\|\mathbf{y}\| = 1$ e que a última expressão nas igualdades acima assume a forma

$$\|A\mathbf{x}\|^2 = \mathbf{y}^\mathsf{T}\begin{bmatrix} \sigma_1^2 & & & & \\ & \ddots & & & \\ & & \sigma_\mu^2 & & \\ & & & \ddots & \\ & & & & 0 \end{bmatrix}\mathbf{y} = \sigma_1^2y_1^2 + \cdots + \sigma_\mu^2y_\mu^2$$

onde $\mathbf{y}^\mathsf{T} = [y_1, \ldots, y_n]$. Como $\sigma_i^2y_i^2 \leq \sigma_1^2y_i^2$, o somatório dos termos destas desigualdades, para $i = 1, \ldots, n$, fornece

$$\|A\mathbf{x}\|^2 = \sigma_1^2y_1^2 + \cdots + \sigma_r^2y_r^2 \leq \sigma_1^2y_1^2 + \cdots + \sigma_1^2y_n^2 = \sigma_1^2(y_1^2 + \cdots + y_n^2) = \sigma_1^2$$

logo, $\|A\mathbf{x}\| \leq \sigma_1$ e visto que para $\mathbf{x} = V[1\ 0\ \cdots\ 0]^\mathsf{T}$ a norma de $A\mathbf{x}$ alcança σ_1, concluímos que

$$\|A\|_2 = \max_{\|\mathbf{x}\|=1} \|A\mathbf{x}\| = \sigma_1$$

□

A norma de Frobenius é uma extensão da norma vetorial euclidiana, portanto, é possível falar de distância entre matrizes num sentido algébrico muito próximo da distância entre vetores, como veremos adiante.

Definição 5.5. *A norma de Frobenius de uma matriz é o número real denotado por* $\|A\|_F$ *e dado pela expressão*

$$\|A\|_F = \sqrt{tr(A^\mathsf{T}A)}.$$

Fazendo $A = [\mathbf{a}_i\ \cdots\ \mathbf{a}_n]$, $\mathbf{a}_i \in \mathbb{R}^m$ e $A^\mathsf{T}A = [a'_{ij}]_{m\times n}$, segue que $a'_{ij} = \mathbf{a}_i^\mathsf{T}\mathbf{a}_j$ e, pela definição acima,

$$\|A\|_F = \sqrt{\mathrm{tr}(A^\mathsf{T}A)} = \sqrt{\mathbf{a}_1^\mathsf{T}\mathbf{a}_1 + \cdots + \mathbf{a}_n^\mathsf{T}\mathbf{a}_n} = \sqrt{\|\mathbf{a}_1\|^2 + \cdots + \|\mathbf{a}_n\|^2}.$$

A última igualdade evidencia que a norma vetorial é um caso particular da norma de Frobenius. Relacionaremos a norma de Frobenius de uma matriz a seus valores singulares, para estabelecer esta relação, valeremo-nos do lema a seguir.

Lema 5.2.1. *Sejam* P *e* Q *matrizes ortogonais. Para uma dada matriz* $A \in \mathbb{R}^{m\times n}$*, tem-se*

$$\|A\|_F = \|PA\|_F = \|AQ\|_F.$$

Demonstração: Por definição,

$$\|PA\|_F^2 = \mathrm{tr}((PA)^\mathsf{T}(PA)) = \mathrm{tr}(A^\mathsf{T}P^\mathsf{T}PA) = \mathrm{tr}(A^\mathsf{T}A) = \|A\|_F.$$

Para a segunda igualdade, observemos que $\mathrm{tr}(A) = \mathrm{tr}(A^\mathsf{T})$, assim

$$\|AQ\|_F = \|(AQ)^\mathsf{T}\|_F = \|Q^\mathsf{T}A^\mathsf{T}\|_F = \|A^\mathsf{T}\|_F = \|A\|_F$$

. □

Teorema 5.4. *Uma matriz* A*, de valores singulares* $\sigma_1, \ldots, \sigma_\mu$*, tem sua norma de Frobenius dada por*

$$\|A\|_F = \sqrt{\sigma_1^2 + \cdots + \sigma_\mu^2}$$

Demonstração: Seja $A = U\mathcal{S}V^\mathsf{T}$ a *SVD* de A.

$$\begin{aligned}
\|A\|_F^2 &= \mathrm{tr}(A^\mathsf{T}A) \\
&= \mathrm{tr}((U\mathcal{S}V^\mathsf{T})^\mathsf{T}(U\mathcal{S}V^\mathsf{T})) \\
&= \mathrm{tr}(V\mathcal{S}^\mathsf{T}\mathcal{S}V^\mathsf{T}) && (5.1) \\
&= \mathrm{tr}(\mathcal{S}^\mathsf{T}\mathcal{S}) && (5.2) \\
&= \sigma_1^2 + \cdots + \sigma_\mu^2
\end{aligned}$$

A sequência entre as igualdades 5.1 e 5.2 deveu-se ao lema 5.2.1. Da última desigualdade acima, temos $\|A\|_F = \sqrt{\sigma_1^2 + \cdots + \sigma_\mu^2}$. □

Através da instrução `norm(A)`, nós podemos calcular a norma espectral da matriz `A`. Inserindo `"fro"` como segundo argumento, o comando `norm(A,"fro")` retorna a norma de Frobenius de `A`.

Exemplo 5.2.1. : norm

Obtendo, respectivamente, a norma espectral e de Frobenius da matriz `A`.

```
>A=[7,1;0,0;5,5], norm(A), norm(A,"fro")
   A  =
        7    1
        0    0
        5    5
 ans  = 9.4868
 ans  =10
```

5.3 DISTANCIA MÍNIMA ENTRE MATRIZES DE POSTOS DIFERENTES

Podemos enxergar a norma de Frobenius de uma matriz como a medida do quão distante esta se encontra da matriz nula. É natural, então, definir a distância $d(A, B)$ entre as matrizes A e B em $\mathbb{R}^{m\times n}$, como

Definição 5.6. $d(A, B) = \|A - B\|_F$

O objetivo desta seção será estabelecer, para uma dada matriz A, a existência de uma matriz B de posto $p < \mathcal{P}(A)$, tal que

$$d(A, B) = \min_{\mathcal{P}(X)\leq p} \|A - X\|_F.$$

A próxima definição descreve a matriz para a qual tal distância mínima é alcançada.

Definição 5.7. *Seja A uma matriz $m \times n$ de posto r e $A = \sigma_1\mathbf{u}_1\mathbf{v}_1^\mathsf{T} + \cdots + \sigma_r\mathbf{u}_r\mathbf{v}_r^\mathsf{T}$ sua SVD . $A_p = \sigma_1\mathbf{u}_1\mathbf{v}_1^\mathsf{T} + \cdots + \sigma_p\mathbf{u}_p\mathbf{v}_p^\mathsf{T}$, com $p < r$,* é a aproximação de posto p pela *SVD* de A.

Visto que $A - A_p = \sigma_{p+1}\mathbf{u}_{p+1}\mathbf{v}_{p+1}^\mathsf{T} + \cdots + \sigma_r\mathbf{u}_r\mathbf{v}_r^\mathsf{T}$, a mudança de índices $\sigma_i' = \sigma_{p+i}$, $\mathbf{u}_i' = \mathbf{u}_{p+i}$ e $\mathbf{v}_i' = \mathbf{v}_{p+i}$ permite a visualização deste somatório como a *SVD* de $A - A_p$. Decorre dos dois últimos teoremas que $\|A - A_p\|_2 = \sigma_{p+1}$ e $\|A - A_p\|_F^2 = \sigma_{p+1}^2 + \cdots + \sigma_r^2$.

O próximo teorema afirma que A_r é a matriz de posto r mais próxima de A, segundo a norma de Frobenius. Para demonstrar tal resultado, estabeleceremos através dos próximos três lemas que dada uma matriz B de posto não maior do que p temos $\sigma_i(A - B) \geq \sigma_{p+i}(A)$.

Lema 5.3.1. *Sejam A e B matrizes em $\mathbb{R}^{m\times n}$. Considerando o posto de B menor ou igual a p segue que*

$$\sigma_1(A-B) \geq \sigma_{p+1}(A).$$

Demonstração: Fazendo $p = n$, m não poderá ser menor do que n, portanto $\mathcal{P}(A) < n+1$ o que implica em $\sigma_{p+1}(A-B) = \sigma_{n+1}(A) = 0$ e daí $\sigma_1(A-B) \geq \sigma_{p+1}(A)$.

Seja agora $p < n$. Mostramos a existência de um vetor $\mathbf{w}$ ao qual $\sigma_1(A-B) \geq \|A\mathbf{w}\| \geq \sigma_{p+1}(A)$. Dado que $\mathcal{P}(B) \leq p$ o teorema do núcleo e da imagem nos garante que $\dim \mathcal{N}(B) \geq n-p > 0$ e portanto $\mathcal{N}(B) \neq \{\mathbf{0}\}$. Sejam $\mathbf{v}_1, \ldots, \mathbf{v}_n$ os vetores singulares direitos de A. Como $\dim \mathcal{G}\{\mathbf{v}_1, \ldots, \mathbf{v}_{p+1}\} + \dim \mathcal{N}(B) > n$ segue que $\mathcal{G}\{\mathbf{v}_1, \ldots, \mathbf{v}_{p+1}\}$ e $\mathcal{N}(B)$ possuem em comum um subespaço W no qual podemos escolher um vetor unitário $\mathbf{w}$ que satisfaz as condições propostas. Com efeito, visto que $\mathbf{w} \in \mathcal{N}(B)$, temos

$$\begin{aligned} \|(A-B)\mathbf{w}\|^2 &= \mathbf{w}^{\mathsf{T}}(A-B)^{\mathsf{T}}(A-B)\mathbf{w} \\ &= \big((A\mathbf{w})^{\mathsf{T}} - (B\mathbf{w})^{\mathsf{T}}\big)\big(A\mathbf{w} - B\mathbf{w}\big) \\ &= \mathbf{w}^{\mathsf{T}}A^{\mathsf{T}}A\mathbf{w} \\ &= \|A\mathbf{w}\|^2 \end{aligned} \tag{5.3}$$

Pelo teorema 5.3, $(\sigma_1(A-B))^2 = \|A-B\|_2^2$. A definição de norma espectral e o fato de $\mathbf{w}$ ser unitário garantem que

$$(\sigma_1(A-B))^2 = \|A-B\|_2^2 \geq \|(A-B)\mathbf{w}\|^2$$

e pela igualdade 5.3

$$(\sigma_1(A-B))^2 \geq \|A\mathbf{w}\|^2.$$

Como elemento de $\mathcal{G}\{\mathbf{v}_1, \ldots, \mathbf{v}_{p+1}\}$, $\mathbf{w}$ pode ser escrito como $\mathbf{w} = w_1\mathbf{v}_1 + \cdots + w_{p+1}\mathbf{v}_{p+1}$. Considerando $\sigma_1\mathbf{u}_1\mathbf{v}_1^{\mathsf{T}} + \cdots + \sigma_\mu\mathbf{u}_\mu\mathbf{v}_\mu^{\mathsf{T}}$ a *SVD* de A, temos que

$$A\mathbf{w} = \sum_{i=1}^{\mu} \sigma_i\mathbf{u}_i\mathbf{v}_i^{\mathsf{T}}\mathbf{w}$$

E como $\mathbf{w}$ é ortogonal a todo $\mathbf{v}_j$, $p+1 < j \leq \mu$, obtemos as igualdades

$$A\mathbf{w} = \sum_{i=1}^{p+1} \sigma_i\mathbf{u}_i\mathbf{v}_i^{\mathsf{T}}\mathbf{w} = \sum_{i=1}^{p+1} \sigma_i w_i\mathbf{u}_i$$

donde

$$\begin{aligned} \|A\mathbf{w}\|^2 &= (A\mathbf{w})^{\mathsf{T}}(A\mathbf{w}) \\ &= \left(\sum_{i=1}^{p+1} \sigma_i w_i\mathbf{u}_i^{\mathsf{T}}\right)\left(\sum_{j=1}^{p+1} \sigma_j w_j\mathbf{u}_j\right) \\ &= \sum_{i,j=1}^{p+1} \sigma_i\sigma_j\ w_i w_j\ \mathbf{u}_i^{\mathsf{T}}\mathbf{u}_j \end{aligned}$$

como $\mathbf{u}_i^{\mathsf{T}}\mathbf{u}_j = 0$ sempre que $i \neq j$, segue que

$$\|A\mathbf{w}\|^2 = \sum_{i=1}^{p+1} \sigma_i^2 w_i^2 \geq \sum_{i=1}^{p+1} \sigma_{p+1}^2 w_i^2$$

como $w_1^2 + \cdots + w_{p+1}^2 = 1$, obtemos

$$\|A\mathbf{w}\|^2 \geq \sigma_{p+1}^2$$

e portanto

$$(\sigma_1(A-B))^2 \geq \|A\mathbf{w}\|^2 \geq (\sigma_{p+1}(A))^2$$

donde segue que

$$\sigma_1(A-B) \geq \sigma_{p+1}(A);$$

como queríamos provar. □

Lema 5.3.2. *Se $A = A' + A''$, então, quaisquer que sejam i, j pertencentes ao conjunto de números naturais, $\sigma_{i+j-1}(A) \leq \sigma_i(A') + \sigma_j(A'')$.*

Demonstração: Inicialmente, nota-se que dados vetores unitários $\mathbf{u}$ e $\mathbf{v}$ tem-se

$$\mathbf{u}^{\mathsf{T}} A\mathbf{v} = \|A\mathbf{v}\| \cos\theta \leq \|A\mathbf{v}\| \leq \sigma_1(A),$$

onde θ é o ângulo entre $\mathbf{u}$ e $\mathbf{v}$.

Assim, para o caso em que $i = j = 1$ teremos

$$\begin{aligned} \sigma_1(A) &= \mathbf{u}_1 A \mathbf{v}_1 \\ &= \mathbf{u}_1^{\mathsf{T}}(A' + A'')\mathbf{v}_1 \\ &= \mathbf{u}_1^{\mathsf{T}} A'\mathbf{v}_1 + \mathbf{u}_1^{\mathsf{T}} A''\mathbf{v}_1 \\ &\leq \sigma_1(A') + \sigma_1(A''), \end{aligned} \tag{5.4}$$

e conforme visto nos comentários que sucederam a definição 5.7, $\sigma_{p+1}(A) = \sigma_1(A - A_p)$, tem-se

$$\sigma_i(A') + \sigma_j(A'') = \sigma_1(A' - A'_{i-1}) + \sigma_1(A'' - A''_{j-1})$$

e a desigualdade em 5.4 permite-nos fazer

$$\sigma_i(A') + \sigma_j(A'') \geq \sigma_1\big(A' + A'' - (A'_{i-1} + A''_{j-1})\big).$$

De fato, $\mathcal{P}(A'_{i-1} + A''_{j-1}) \leq i + j - 2$, por isso, o lema 5.3.1 pode ser aplicado no último termo da desigualdade acima, deixando-o como

$$\sigma_1\big(A' + A'' - (A'_{i-1} + A''_{j-1})\big) \geq \sigma_{(i+j-2)+1}(A' + A'') = \sigma_{i+j-1}(A)$$

da?, $\sigma_{+j-1}(A) \leq \sigma_i(A') + \sigma_j(A'')$. □

Lema 5.3.3. *Teremos $\sigma_i(A-B) \geq \sigma_{p+i}(A)$ para quaisquer matrizes A e B em $\mathbb{R}^{m\times n}$, com $\mathcal{P}(B) \leq p$.*

Demonstração: De $\mathcal{P}(B) \leq p$ observamos que $\sigma_{p+1}(B) = 0$. Aplicando o lema 5.3.2 com $A' = A - B$ e $A" = B$, obtemos

$$\sigma_i(A-B) = \sigma_i(A-B) + \sigma_{p+1}(B) \geq \sigma_{i+p+1-1}\big((A-B)+B\big) = \sigma_{p+i}(A).$$

□

Por fim, o próximo teorema estabelece que a menor distância de uma matriz $A_{m\times n}$, de posto r, com $\mu = \min\{m,n\}$, a uma matriz B, de posto p não maior que r, é alcançada quando $B = A_p$.

Teorema 5.5. *Sejam A e B matrizes tal como descritas acima.*

$$\|A-B\|_F \geq \|A-A_p\|_F.$$

Demonstração: Pelo teorema 5.4,

$$\|A-B\|_F^2 = \sum_{i=1}^{\mu} \sigma_i(A-B)^2.$$

Aplicando o lema 5.3.3 no lado direito desta igualdade,ficamos com

$$\begin{aligned} \|A-B\|_F^2 &= \sigma_1(A-B)^2 + \cdots + \sigma_\mu(A-B)^2 \\ &\geq \sigma_{p+1}(A)^2 + \cdots + \sigma_\mu(A)^2 \\ &= \|A-A_p\|_F^2. \end{aligned}$$

□

CAPÍTULO 6

A SVD E A COMPRESSÃO DE IMAGENS DIGITAIS

Como já foi dito, uma imagem digital é gerada pelo computador a partir de uma matriz, vista agora como um cojunto de dados. Desta forma, a compressão de imagens digitais corresponde, estatísticamente falando, a redução das informações contidas na matriz de dados, de modo que os dados resultantes ainda sejam capazes de transmitir a maior parte da informação contida na matriz original. Em computação gráfica, esta redução é chamada de compressão com perda (da informação). Veremos neste capítulo um método de redução com perda que baseia-se no *truncamento da matriz de dados*, processo em que A, matriz correspondente a uma imagem, é substituida por alguma aproximação por sua SVD .

O posto de uma matriz determina a *quantidade de informação* nescessária para em sua representação. Para uma matriz $A_{m\times n}$, por exemplo, teoricamente necessitamos de mn dados, correspondentes as entradas de A, para descrevê-la. O posto de A, sendo menor do que μ, pemite que voltemo-nos a obtenção de p vetores L.I. em A, aos quais serão registradas suas pm entradas, deixando os demais $n-p$ vetores serem dados pelos coeficiente das combinações lineares entre os L.I.. O esquema abaixo exibe o número de dados em cada conjunto de linhas e colunas.

$$m\Big\{\Big[\overbrace{\underbrace{\mathbf{a}_1 \quad \cdots \quad \mathbf{a}_p}_{p} \;\Big|\; \underbrace{\mathbf{a}_{p+1} \quad \cdots \quad \mathbf{a}_n}_{n-p}}^{n}\Big]$$

A quantidade de informação contida na matriz A é dada pela expressão $mp+p(n-p) = p(m+n-p)$ que, por sua vez, assume μ como valor máximo.

Pela SVD entretanto, o vetor A é dado por p vetores ortogonais do $\mathbb{R}^m$, $\mathbf{u}_1, \ldots, \mathbf{u}_p$, e p vetores ortogonais do $\mathbb{R}^m$, $\sigma_1\mathbf{v}_1, \ldots, \sigma_p\mathbf{v}_p$. Com isso, para salvar toda a informação contida em A é suficiente que o computador guarde todas as entradas de $\mathbf{u}_1$ e $\sigma_1\mathbf{v}_1$, todas as entradas de $\mathbf{u}_2$ e $\sigma_2\mathbf{v}_2$, exceto uma de cada, que podem ser determinadas a partir das equações

$$\mathbf{u}_1^{\mathsf{T}}\mathbf{u}_2 = 0 \qquad \text{e} \qquad (\sigma_1\mathbf{v}_1)^{\mathsf{T}}(\sigma_2\mathbf{v}_2) = 0,$$

todas as entradas de $\mathbf{u}_3$ e $\sigma_3\mathbf{v}_3$, exceto duas em cada vetor, as quais são obtidas mediante os sistemas

$$\begin{cases} \mathbf{u}_1^{\mathsf{T}}\mathbf{u}_3 &= 0 \\ \mathbf{u}_2^{\mathsf{T}}\mathbf{u}_3 &= 0 \end{cases} \qquad \text{e} \qquad \begin{cases} (\sigma_1\mathbf{v}_1)^{\mathsf{T}}(\sigma_3\mathbf{v}_3) &= 0 \\ (\sigma_2\mathbf{v}_2)^{\mathsf{T}}(\sigma_3\mathbf{v}_3) &= 0 \end{cases}$$

Enfim, para os vetores $\mathbf{u}_i$ e $\sigma_i\mathbf{v}_i$ salvam-se suas entradas, exceto $i-1$ em cada vetor, que são facilmente calculadas pelo computador através dos sistemas

$$\begin{cases} \mathbf{u}_1^{\mathsf{T}}\mathbf{u}_i &= 0 \\ \vdots & \\ \mathbf{u}_{i-1}^{\mathsf{T}}\mathbf{u}_i &= 0 \end{cases} \qquad \text{e} \qquad \begin{cases} (\sigma_1\mathbf{v}_1)^{\mathsf{T}}(\sigma_i\mathbf{v}_i) &= 0 \\ \vdots & \\ (\sigma_{i-1}\mathbf{v}_{i-1})^{\mathsf{T}}(\sigma_i\mathbf{v}_i) &= 0 \end{cases}$$

Constituídos através dos vetores já obtidos $\mathbb{R}^m$, $\mathbf{u}_1, \ldots, \mathbf{u}_{i-1}$ e $\sigma_1\mathbf{v}_1, \ldots, \sigma_p\mathbf{v}_{i-1}$. A quantidade de informação suficiente para descrever cada conjunto de vetores é dada por

$$m + (m-1) + \cdots + (m-(p-1)) = pm - \frac{p(p-1)}{2}$$

$$n + (n-1) + \cdots + (n-(p-1)) = pn - \frac{p(p-1)}{2}$$

e assim, a quantidade de informação contida em A, dada pela sua *SVD* , totaliza $p(m + n - p + 1)$ dados.

Embora o número $p(m + n - p + 1)$ seja maior do que $p(m + n - p)$, obtido pelo isolamento de p vetores L.I., a superioridade da *SVD* na compressão de imagens reside em sua capacidade de redução de dados com o mínimo de comprometimento da qualidade destas.

Ao substituirmos A, uma matriz associada a uma imagem em escala de cinza, por A_k, sua aproximação de posto k pela *SVD* , com $k < \mathcal{P}(A)$, estaremos obtendo uma imagem com uma quantidade menor de informação, e no entanto, interpretando segundo o resultado do teorema 5.5, tal imagem, dentre todas que possuem a mesma quantidade de informação, é a que mais se assemelha a original. No octave, obteremos A_k mediante a função `apsvd` descrita no exemplo 6.0.1.

Exemplo 6.0.1. : Função apsvd

Retorna A_k, a aproximação de posto k pela *SVD* de A.

```
>function Y=apsvd(A,k)
>[m n c]=size(A);
>for i=1:c
>[U(:,:,i) S(:,:,i) V(:,:,i)]=svd(double(A(:,:,i)));
>Y(:,:,i)=U(:,1:k,i)*S(1:k,1:k,i)*V(:,1:k,i)';
>endfor
>endfunction
```

A aproximação obtida pela função `apsvd`, Vale para matrizes associadas a imagens em escala de cinza e associadas a imagens coloridas, para estas, calcula-se a aproximação de posto k pela *SVD* de cada componente RGB.

Exemplo 6.0.2.

Obtendo o truncamento da imagem lena com k valores singulares. **Figura 6.1**

```
>A=imread(lenaGray.png);
>A_50=apsvd(A,50);
>imwrite(uint8(A_50),"lenaGraySVD50.jpg")
```

Na figura 6.1, temos um exemplo de compressão por truncamento da matriz utilizando 50 valores singulares em uma imagem com tamanho 512×512. Uma questão que se apresenta naturalmente é se a *SVD* é igualmente eficiente para comprimir qualquer

(a) Original: lena (b) Imagem reconstruída com 50 valores singulares

Figura 6.1

imagem. Para respondê-la, valeremo-nos de ferramentas para decidir o quão boa é uma compressão por truncamento da matriz de dados.

A *razão de compressão* é o quociente ente a quantidade de informação na imagem comprimida e a quantidade de informação na imagem original. Dada $A \in \mathbb{R}^{m\times n}$ a razão de compressão de A_k é dada por $k(m+n-k+1)/mn$.

Exemplo 6.0.3. : Função razao_de_compressao
Retorna a razão de compressão de uma matriz com as dimensões de A quando truncada para a matriz A_k.

```
>function y=razao_de_compressao(A,k)
>[m,n]=size(A);
>y=k*(m+n+1-k)/(m*n);
>endfunction
```

O uso de tal razão é justificado apenas quando $k(m+n-k+1) < mn$ (**). Com o fim de estabelecer os valores de k que validam tal desigualdade, nós a reescrevemos da seguinte maneira:

$$k^2 - (m+n+1)k + mn > 0.$$

E daí obtemos as desigualdades

$$k < \frac{(m+n+1) - \sqrt{(m+n+1)^2 - 4mn}}{2} \quad \text{ou} \quad k > \frac{(m+n+1) + \sqrt{(m+n+1)^2 - 4mn}}{2}.$$

Visto que

$$k > \frac{(m+n+1) + \sqrt{(m+n+1)^2 - 4mn}}{2} > \frac{m+n+1}{2}$$

e quando substituímos k por $(m+n+1)/2$ não ocorre compressão, pois $k(m+n-k+1)$ torna-se $(m+n+1)^2/4 > mn$, por conseguinte, a expressão (**) é verdadeira apenas quando

$$k < \frac{(m+n+1) - \sqrt{(m+n+1)^2 - 4mn}}{2}.$$

A expressão no lado direito da desigualdade é o valor crítico da imagem, cujo cálculo no octave é feito através da função definida no exemplo 3.3.1. A razão de compressão expressa, em percentual, a quantidade de informação em A_k comparada a quantidade de informação em A. Este percentual não deve ser confundido com um percentual da quantidade de informação de A utilizada, visto que estamos usando codificações diferentes para A e A_k. Na figura 6.1 a razão de compressão obtida com o uso de 50 valores singulares foi de, aproximadamente, $18,5\%$, conforme exibido no exemplo a seguir.

Exemplo 6.0.4.

Calculando o valor crítico da imagem 'lena', bem como sua razão de compressão para o truncamento com 50 valores singulares. **Figura 6.1**

```
>I=imread("lena512.bmp"); >valor_critico(I)
ans = 489.87
>razao_de_compressao(I,50)
ans = 0.18597
```

Para uma medida do quanto é perdido em uma imagem após o truncamento de sua matriz de dados, usaremos o conceito de *erro relativo* referente a norma de Frobenius. O erro cometido ao substituir A por A_k é, segundo a norma de Frobenius, dado por $\|A - A_k\|_F$. O erro relativo corresponde a razão $\|A - A_k\|_F / \|A\|_F$ e torna-se maior conforme k se aproxime de 1.

Exemplo 6.0.5. : Função erro_relativo

Retorna o erro relativo cometido ao substituir A por A_k.

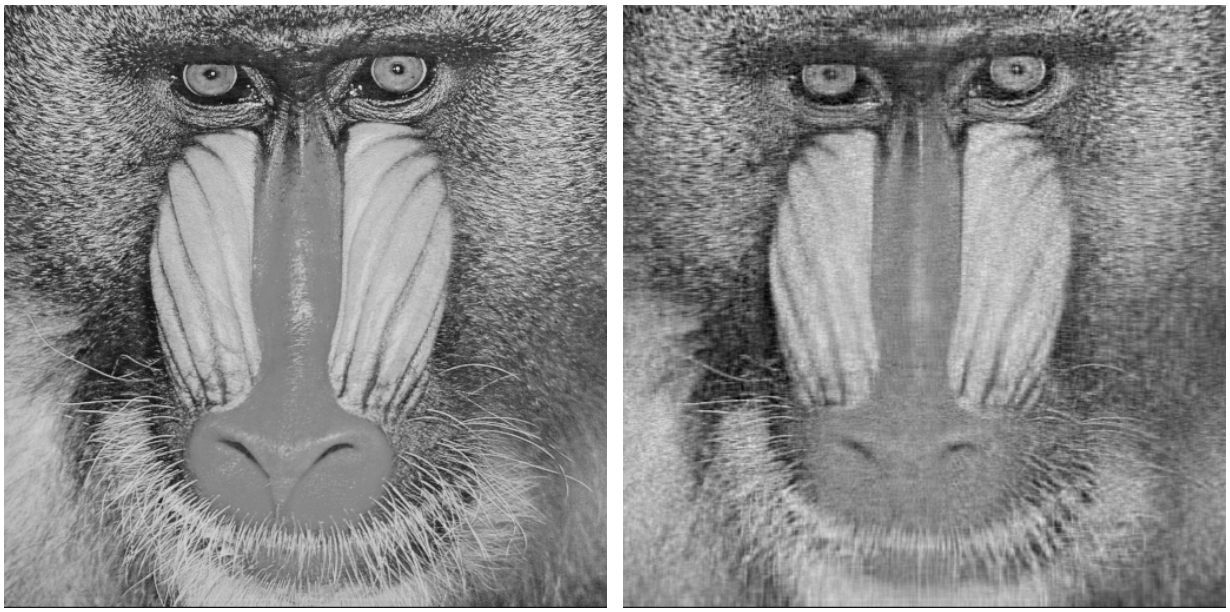
```
>function y=erro_relativo(A,k)
>y=norm(double(A)-apsvd(A,k),"fro")/norm(double(A),"fro");
>endfunction
```

Exemplo 6.0.6.

Calculando o erro relativo da imagem I da figura 6.1, cometido quando substituimos I por I_{50}. **Figura 6.1**

```
> erro_relativo(I,50)
ans = 0.059439
```

Já as imagens da figura 6.2 possuem as mesmas dimensões daquelas na figura 6.1 e também foi feita uma reconstrução com 50 valores singulares, ou seja, obtemos a mesma razão de compressão próxima de $18,5\%$, e no entanto, conforme calculado no exemplo 5.0.7, o erro relativo cometido foi de $13,3\%$.



(a) Original: mandril (b) Reconstruida com 50 valores singulares

Figura 6.2

Exemplo 6.0.7.

Calculando o erro relativo da imagem na figura 6.2 quando tomamos sua aproximação de posto 50 pela *SVD* e obtendo tal aproximação.. **Figura 6.2**

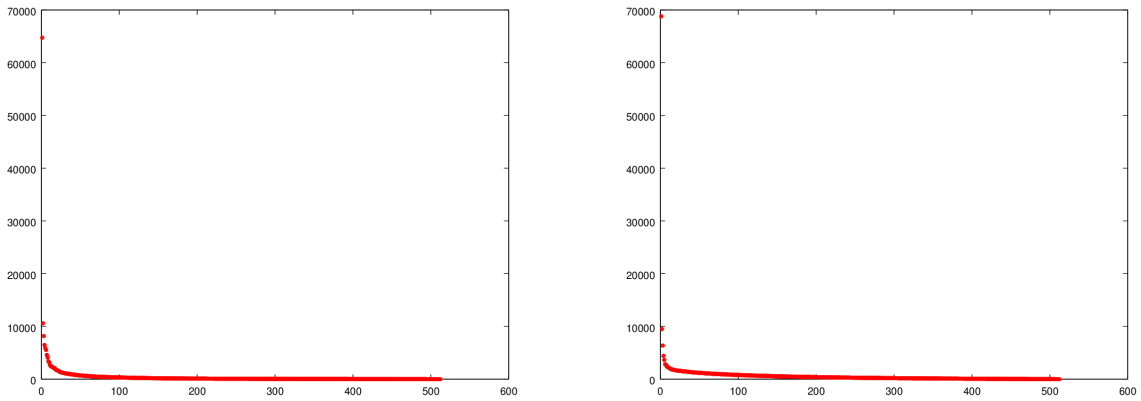
```
>M=imread("mandril.bmp");
>erro_relativo(M,50)
ans = 0.13334
>imwrite(uint8(apsvd(M,50)),"mandrilSVD50.jpg")
```

Não é nosso objetivo estabelecer rigorosamente as razões pelas quais se dão diferentes resultados na compressão de imagens através do truncamento, no entanto, como o intuito de firmar bases intuitivas para tal, exibiremos de que maneira se dá a perda de informação na compressão usando a aproximação pela *SVD* . Faremos isso analisando como ocorreria o ganho de informação na imagem partindo da matriz A_1, a qual representa uma compressão máxima, e reconstruindo-a até a matriz A_k.

Primeiramente observemos que dada uma matriz $m \times n$ A de posto p, em sua *SVD* dada por $A = \sigma_1 \mathbf{u}_1 \mathbf{v}_1 + \cdots + \sigma_p \mathbf{u}_p \mathbf{v}_p^\mathsf{T}$ cada uma das matrizes $\mathbf{u}_i \mathbf{v}_i^\mathsf{T}$ possuem posto e norma de Frobenius iguais a 1. Com efeito, sendo $\mathbf{v}_i^\mathsf{T} = [v_{1i}, \ldots, v_{ni}]$, o produto $\mathbf{u}_i \mathbf{v}_i^\mathsf{T}$ fica expresso como $\mathbf{u}_i \mathbf{v}_i^\mathsf{T} = [v_{1i}\mathbf{u}_i, \ldots, v_{ni}\mathbf{u}_i]$, o que evidencia o posto 1 de $\mathbf{u}_i \mathbf{v}_i^\mathsf{T}$. Vimos na seção 5.2 que a norma de Frobenius pode ser calculada a partir das colunas de $\mathbf{u}_i \mathbf{v}_i^\mathsf{T}$ como

$$\|\mathbf{u}_i \mathbf{v}_i^\mathsf{T}\|_F^2 = \|v_{1i}\mathbf{u}_i\|^2 + \cdots + \|v_{ni}\mathbf{u}_i\|^2 = (v_{1i}^2 + \cdots + v_{ni}^2)\|\mathbf{u}_i\|^2 = \|\mathbf{v}_i^\mathsf{T}\|^2 \|\mathbf{u}_i\|^2$$

e portanto, por $\mathbf{v}_i$ e $\mathbf{u}_i$ serem unitários, $\|\mathbf{u}_i \mathbf{v}_i^\mathsf{T}\|^2 = 1$. Assim, cada *componente* $\sigma_i \mathbf{u}_i \mathbf{v}_i$ da *SVD* de A é uma matriz de posto 1 e norma de Frobenius igual a σ_i.



(a) valores singulares 'lena' (b) valores singulares 'mandril'

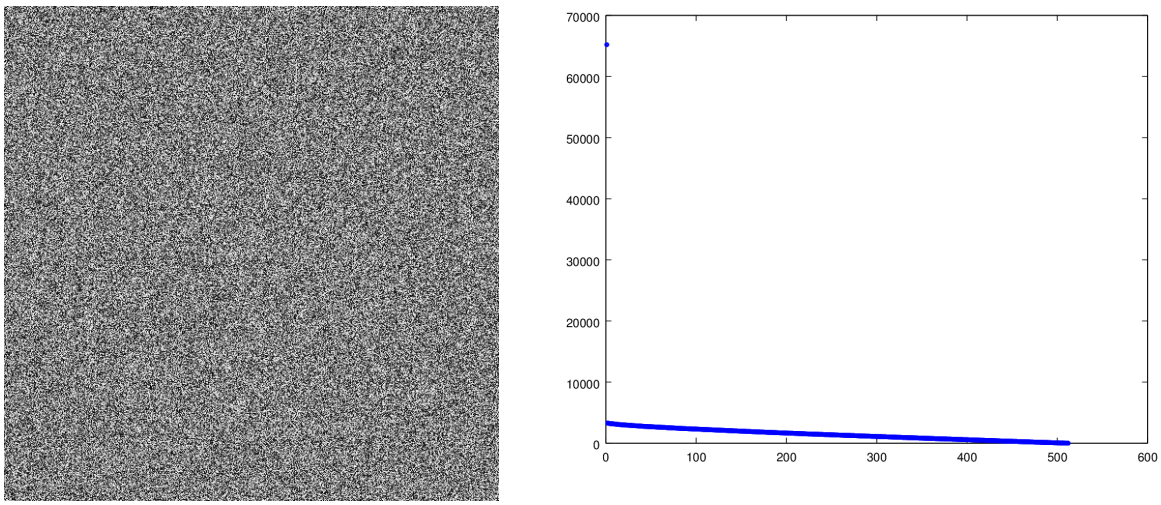
Figura 6.3

A distribuição dos valores singulares nas imagens 'lena' e 'mandril' são exibidas na figura 6.3. Na figura 6.3(a) temos o gráfico dos valores singulares da imagem 'lena' e na figura 6.3(b) o gráfico dos valores singulares da imagem "mandril". é possível notar nas duas figuras a importância dos primeiros valores singulares na composição da imagem, sobretudo do primeiro. O gráfico na figura 6.4(b) corresponde a uma plotagem dos valores singulares da imagem na figura 6.4(a) de mesmas dimensões das imagens citadas anteriormente e gerada mediante o comando `rand`. Fizemos sua reconstrução com 50 valores singulares e obtivemos um erro relativo de 42%.

Baseando-nos no que foi exibido nos dois parágrafos precedentes, podemos enxergar a imagem gerada por $A_1 = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^\mathsf{T}$ como a base sobre a qual serão feitas melhores aproximações de A, visto que A_1 reúne uma parcela significativa da informação contida em A. Tais aproximações tornam-se melhores conforme incluímos detalhes a A_1, mediante a adição das matrizes de posto 1, $\sigma_i \mathbf{u}_i \mathbf{v}_i^\mathsf{T}$. Pela análise dos gráficos de valores singulares, notamos que a relevncia de tais detalhes para a aproximação da imagem A por A_i dependem do quão grande seja o valor da razão $\sigma_i / \|A\|_F$, visto que $\|\sigma_i \mathbf{u}_i \mathbf{v}_i^\mathsf{T}\|_F = \sigma_i$. A eficiência na compressão de imagens fotográficas pela substituição de A por A_k ocorre devido a existência de grandes regiões com tons de intensidade muito próximas. Na imagem 'lena', por exemplo, muitos pixeis correspondentes ao tom da pele da modelo são iguais ou parecidos, decorre que a perda de detalhes nessa região não compromete a visualização final da imagem.

Na figura 6.5 vemos as vinte e uma primeiras etapas de como é "montada"a imagem 'lena' pelos seus valores singulares. O erro relativo para 1 valor singular é de 31% e o erro relativo cometido pelo truncamento com 21 valores singulares é de um pouco mais de 10%.

A compressão por truncamento de uma imagem colorida será feita aplicando o truncamento a cada componente RGB da imagem. A razão de compressão de uma imagem colorida $m \times n$ comprimida como k valores singulares igual a razão de compressão de uma imagem em tons de cinza $m \times n$ comprimida com a mesma quantidade de valo-



(a) Imagem randômica

(b) valores singulares da imagem randômica

Figura 6.4

res singulares. A norma de Frobenius será estendida para uma matriz 3d I da seguinte maneira:

$$\|I\|_F = \sqrt{\|R\|_F^2 + \|G\|_F^2 + \|B\|_F^2}$$

onde R, G e B são, respectivamente, as matrizes das componentes vermelha, verde e azul da matriz I. Por meio desta norma, fica definido o erro relativo cometido pelo truncamento de uma imagem colorida mediante a instrução exposta no exemplo 6.0.8.

Exemplo 6.0.8. : Função erro_relativo2

Retorna o erro relativo cometido ao substituir A por A_k quando A é uma imagem colorida.

```
>function y=erro_relativo2(X,k)
>A=double(X);
>a=norm(A(:,:,1)-apsvd(A(:,:,1),k),"fro");
>b=norm(A(:,:,2)-apsvd(A(:,:,2),k),"fro");
>c=norm(A(:,:,3)-apsvd(A(:,:,3),k),"fro");
>d=norm(A(:,:,1),"fro")^2+norm(A(:,:,2),"fro")^2+norm(A(:,:,3),"fro")^2;
>y=sqrt(a^2+b^2+c^2/d)
>endfunction
```

Por fim, segue na figura 6.6 o resultado de uma compressão a razão de 50%, na qual obtivemos o erro relativo de $2,2\%$. Vale mencionar que esta imagem reconstruída com 150 valores singulares é praticamente indistinguível da original.

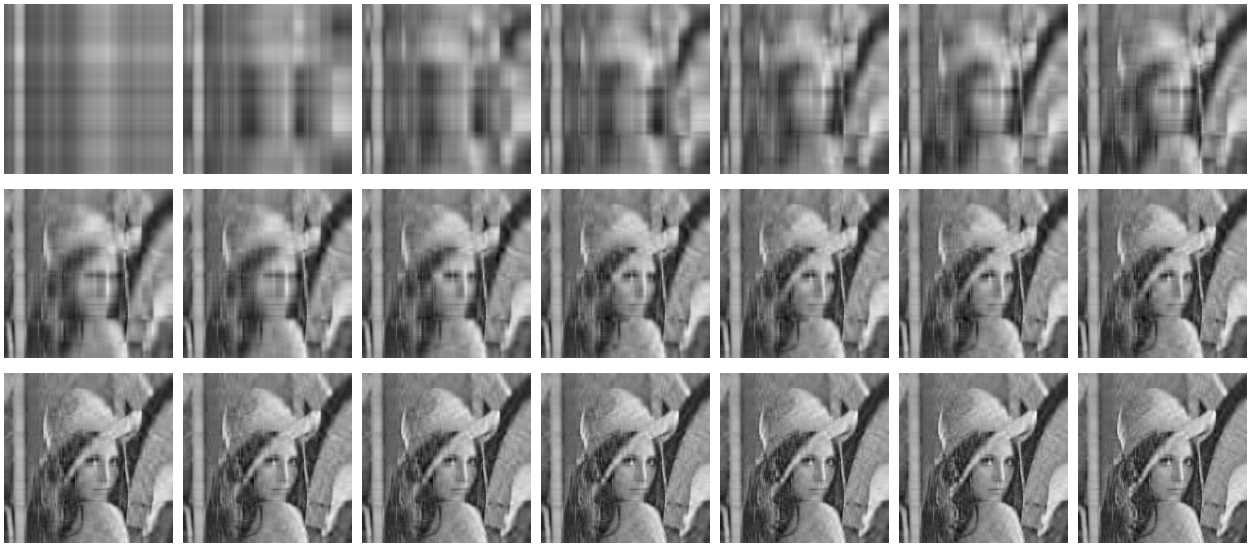


Figura 6.5

(a) Original (b) Reconstruida com 150 valores singulares

Figura 6.6

REFERÊNCIAS BIBLIOGRÁFICAS

[1] GOMES, Jorge; VELHO, Luiz. *Computação Gráfica:* Imagem. 2 ed. Rio de Janeiro: IMPA, 2002.

[2] LIMA, Elon Lages. *Álgebra Linear*. 8 ed. Rio de janeiro: IMPA, 2014.

[3] LAY, David C; tradução de Valéria de Magalhães Iorio. *Álgebra linear e suas aplicações*. 4 ed. Rio de Janeiro, LTC, 2013.

[4] HEFEZ, Abramo; FERNANDEZ, Cecília S. *Introdução à Álgebra Linear*. Rio de Janeiro: SBM, 2012.

[5] ARAÚJO, Thelmo de. *Álgebra Linear:* Teoria e Aplicações. Rio de Janeiro: SBM, 2014.

[6] TEIXEIRA, Sergio Roberto. *Octave:* Uma Introdução. Disponível em http://www.rodrigofernandez.com.br/ecomp/ref/octave-final.pdf. Acesso em 29 de Julho de 2015.