UNIVERSIDADE FEDERAL DE SANTA CATARINA

CAMPUS DE FLORIANÓPOLIS

PROGRAMA DE MESTRADO PROFISSIONAL DE MATEMÁTICA EM REDE NACIONAL - PROFMAT

Bruna da Silva Donadel

RESTAURAÇÃO DE IMAGENS:

Uma abordagem didática para ensino de subespaços vetoriais

Florianópolis

2021

Bruna da Silva Donadel

RESTAURAÇÃO DE IMAGENS:

Uma abordagem didática para ensino de subespaços vetoriais

Dissertação submetida ao Programa de Mestrado Profissional de Matemática em Rede Nacional - PROFMAT da Universidade Federal de Santa Catarina como requisito parcial para a obtenção do Grau de Mestre em Matemática. Com área de concentração no Ensino de Matemática.
Orientador: Prof. Dr. Leonardo Silveira Borges

Florianópolis

2021

Ficha de identificação da obra elaborada pelo autor.

Bruna da Silva Donadel
Restauração de Imagens: Uma abordagem didática para ensino de subespaços vetoriais / Bruna da Silva Donadel; Orientador, Leonardo Silveira Borges, 2021.
82 p.

Dissertação (Mestrado profissional) - Universidade Federal de Santa Catarina, Departamento de Matemática, Programa de Mestrado Profissional de Matemática em Rede Nacional - PROFMAT.

Inclui referências

1. Restauração de imagens. 2. LSQR. 3.TSVD. 4. Álgebra Linear. 5. Sequência didática. I. Leonardo Silveira Borges. II. Universidade Federal de Santa Catarina. Programa de Mestrado Profissional de Matemática em Rede Nacional - PROFMAT. III. RESTAURAÇÃO DE IMAGENS: Uma abordagem didática para ensino de subespaços vetoriais.

Bruna da Silva Donadel

RESTAURAÇÃO DE IMAGENS:

Uma abordagem didática para ensino de subespaços vetoriais

O presente trabalho em nível de mestrado foi avaliado e aprovado pela Banca Examinadora composta pelos seguintes membros:

Prof. Dr. Ednei Felix Reis
UTFPR

Prof. Dr. Leandro Batista Morgado
UFSC

Prof. Dr. Sérgio Tadao Martins
UFSC

Certificamos que esta é a **versão original e final** do trabalho de conclusão que foi julgado adequado para obtenção do título de Mestre em Matemática.

Documento assinado digitalmente
Maria Inez Cardoso Goncalves
Data: 17/01/2022 09:46:36-0300
CPF: 908.897.119-68
Verifique as assinaturas em https://v.ufsc.br

Prof[a]. Dr[a]. Maria Inez Cardoso Gonçalves
Coordenadora do Programa

Documento assinado digitalmente
Leonardo Silveira Borges
Data: 18/01/2022 14:13:13-0300
CPF: 041.171.889-41
Verifique as assinaturas em https://v.ufsc.br

Prof. Dr. Leonardo Silveira Borges
Orientador

Florianópolis, 29 de novembro de 2021.

Dedico este trabalho aos meus pais, minha base.

AGRADECIMENTOS

Agradeço primeiramente a Deus pela minha vida e dos que me rodeiam, como também por me fortalecer nos dias bons e ruins.

Ao meu orientador, Prof. Dr. Leonardo Silveira Borges, pela paciência, disponibilidade e inúmeras contribuições. Sua colaboração foi de suma importância.

Ao meu marido, Gustavo Machado, por literalmente estar ao meu lado me ajudando em diversos aspectos.

Aos meus pais, irmãos, avós, sogros e amigos pelo apoio e compreensão.

Aos colegas da Sinqia que tornam o trabalho mais descontraído.

RESUMO

Este trabalho apresenta um estudo sobre os problemas inversos de restauração de imagem, abordando os clássicos métodos de projeção TSVD e LSQR para solução de problemas de grande porte e sugere a abordagem dessa problemática na disciplina de Álgebra Linear do Ensino Superior, contribuindo para ampliar o entendimento de conceitos como Espaço Vetorial e Subespaço. Partindo do pressuposto que o processo de embaçamento de imagens é linear, algumas funções de propagação de ponto são apresentadas para posteriormente serem usadas na modelagem dos problemas de restauração. O método TSVD trunca a solução clássica de mínimos quadrados escrita em termos da decomposição em valores singulares da matriz A, descartando a informação relacionada aos menores valores singulares, a fim de amenizar a contribuição do ruído na solução. Embora este seja um método poderoso, o alto custo computacional necessário para calcular a SVD torna inviável sua utilização em problemas de grande porte. Já o método LSQR constrói uma sequência de soluções sobre os subespaços de Krylov $\mathcal{K}_j(A^TA, A^Tb)$ e obtém boa parte das informações relevantes do problema com relativamente poucas iterações. Para exemplificar esse fenômeno, o trabalho mostra o comportamento das soluções para alguns problemas numéricos com e sem ruído nos dados. Ao final, propõe uma sequência de aulas e alguns exercícios para uma primeira disciplina de Álgebra Linear que exploram os problemas de restauração de imagem com solução via método LSQR.

Palavras-chave: Restauração de imagens; LSQR; TSVD; Álgebra Linear; Sequência didática.

ABSTRACT

This work presents a study on the inverse problems of image restoration, approaching the classical TSVD and LSQR projection methods to solve large-scale linear systems, and proposes to treat this problem in the Linear Algebra subject of higher education, which helps to broaden the understanding of concepts such as Vector Space and Subspace. Assuming that the image blurring process is linear, some Point Spread Function are introduced, which can be used later in modeling image restoration problems. The TSVD method truncates the naive least squares solution written in terms of the singular value decomposition of the matrix A, discarding the information related to the smallest singular values to mitigate the contribution of the noise in the solution. Although this is a powerful method, the high computational cost required to calculate the SVD makes its use unfeasible when dealing with large-scale problems. The LSQR method, on the other hand, generates a sequence of solutions on the Krylov subspaces $\mathcal{K}_j(A^TA, A^Tb)$ and obtains much of the relevant information of the problem with relatively few iterations. To illustrate this phenomenon, the work shows the behavior of solutions to some numerical problems with and without noise in the data. At the end, it proposes a set of classes and some exercises for a first course of Linear Algebra that explore the problems of image restoration problems with the solution by the LSQR method.

Keywords: Image restoration; LSQR; TSVD; Linear algebra; Teaching sequence.

Lista de Figuras

Lista de Tabelas

Abreviaturas e Siglas

LSQR Mínimos Quadrados via fatoração QR (Least Squares via QR factorization). 19

PSF Função de Propagação do Ponto (Point Spread Function). 11

TSVD Decomposição em Valores Singulares Truncada (Truncated Singular Value Decomposition). 19

Conteúdo

Capítulo 1

Introdução

A matemática é considerada por muitos uma disciplina difícil e complicada, tanto é que aqueles que têm maior facilidade nas disciplinas de exatas muitas das vezes são considerados "mais inteligentes" (COIMBRA, 2008). Além da dificuldade característica, não é difícil encontrar casos em que o aluno depara-se com novos conceitos que são apresentados de forma completamente abstrata em que não há preocupação em vincular com algo prático ou com os conhecimentos anteriores já consolidados na estrutura cognitiva dele.

Na graduação não é diferente, muita das vezes alunos dos cursos de exatas têm dificuldade em entender efetivamente alguns conceitos, principalmente quando exigem certo nível de abstração. Por conhecimento de causa e corroborado por Coimbra (2008) e Furtado (2010), dois desses assuntos que costumam gerar dúvidas são os Espaços Vetoriais e os Subespaços, conteúdos abordados em um primeiro curso de Álgebra Linear, sobre os quais muitos outros são desenvolvidos.

Nesta dissertação apresentamos uma proposta de abordagem de subespaços vetoriais aplicada ao problema inverso de restauração de imagens, pois nesse contexto conseguimos ilustrar espaços e subespaços vetoriais de grandes dimensões, tornando o assunto mais visual e prático.

Para embasar o problema de restauração de imagens, no Capítulo 2 introduzimos o conteúdo de processamento de imagens digitais, como também apresentamos alguns modelos de embaçamento, que são necessários para aplicação da nossa proposta de restauração.

No Capítulo 3 apresentamos duas metodologias de como resolver problemas $Ax = b$: (i) uma baseada na decomposição em valores singulares, que é ferramenta extremamente poderosa, mas que rapidamente se torna computacionalmente muito "cara" quando as

dimensões do problema aumentam; (ii) e uma outra baseada em projeção em subespaços de Krylov que é computacionalmente menos custosa e mais adequada para problemas de grande porte.

Já no Capítulo 4, propomos uma sequência de aulas para a disciplina de Álgebra Linear. Essa sequência foi pensada para que ao final os alunos tenham insumos suficientes para vislumbrar como espaços e subespaços vetoriais podem ser utilizados na prática. Por fim, temos as considerações finais no Capítulo 5.

Capítulo 2

Processamento de Imagem

Fenômenos físicos podem ser interpretados como uma sequência de causa, processamento e efeito, em que a "causa" é a entrada de dados (*input*) ou o estímulo que ao sofrer a ação das propriedades do sistema gera uma resposta que também pode ser chamada de saída, *output* ou efeito, conforme Figura 2.1.

Figura 2.1: Modelagem de um processo físico.

Os *problemas diretos* buscam descobrir a resposta do sistema a partir de determinado estímulo inicial e do processamento que ele sofre, enquanto os *problemas inversos* partem da resposta observada ou desejada para encontrar a causa.

Para exemplificar, apresentamos a seguir alguns pares de problemas diretos e inversos: o problema de encontrar os valores $y_1, y_2, \cdots, y_n$ que um dado polinômio assume para $x_1, x_2, \cdots, x_n$ é um problema direto, enquanto encontrar um polinômio p de grau n que assume os valores $y_1, y_2, \cdots, y_n$ nos pontos $x_1, x_2, \cdots, x_n$ é um problema inverso. Ou então, o problema inverso de encontrar a imagem original, dados a imagem embaçada e o modelo de embaçamento, nesse caso o problema direto correspondente é obter a imagem embaçada sendo dados a imagem "exata" e o modelo de embaçamento. Ou ainda, encontrar a forma de um objeto, dada a intensidade do som ou das ondas eletromagnéticas espalhadas por esse objeto, é um problema inverso, enquanto o problema direto é calcular

a onda espalhada para um determinado objeto (KIRSCH, 2011).

Na matemática esses fenômenos podem ser modelados como uma equação do tipo

$$\mathcal{A}(f) = g \tag{2.1}$$

sendo $\mathcal{A} : \mathcal{F} \to \mathcal{G}$ um operador, linear ou não, $f \in \mathcal{F}$ e $g \in \mathcal{G}$ funções ou vetores de entrada e saída, respectivamente, para $\mathcal{F}$ e $\mathcal{G}$ espaços apropriados. Assim, dados $\mathcal{A}$ e f, os problemas diretos consistem em calcular $\mathcal{A}(f)$ e, dados $\mathcal{A}$ e g, os problemas inversos buscam f que satisfaz $\mathcal{A}(f) = g$ (KIRSCH, 2011).

Hadamard (apud. Kirsch, 2011) propôs que um modelo matemático para um problema físico é bem-posto se cumpre as três exigências seguintes:

1. Existe uma solução para o problema (existência);
2. Existe no máximo uma solução para o problema (unicidade);
3. A solução depende continuamente dos dados (estabilidade).

Desses fatores, o mais importante para conseguir obter uma solução satisfatória é a estabilidade, pois é quase impossível evitar perturbação nos dados. Porque, se estamos lidando com problemas práticos, os dados obtidos muitas vezes contém imprecisões ou erros de medições e, além disso, podem haver erros de arredondamentos e truncamentos quando estes dados são manipulados em sistemas computacionais que muitas vezes utilizam aritmética finita em vez de aritmética exata, por exemplo. Sem a garantia deste item a solução pode ficar muito distante da desejável. Já em casos de múltiplas soluções é preciso obter mais informações do problema e em caso de não haver solução uma possibilidade é ampliar o espaço da solução (KIRSCH, 2011). Caso algum item não seja satisfeito temos um problema mal-posto.

Em geral, senão sempre, os problemas inversos são mal-postos no sentido de Hadamard. Esses problemas aparecem em restauração de imagem (NAGY; PALMER; PERRONE, 2004), espalhamento acústico (COLTON; COYLE; MONK, 2000), (ZHANG; GUO, 2021), eletrocardiografia e magnetocardiografia (AHRENS, 2015), tomografia (MARTINS, 2016), convolução (BAUMEISTER; LEITÃO, 2005), entre outros.

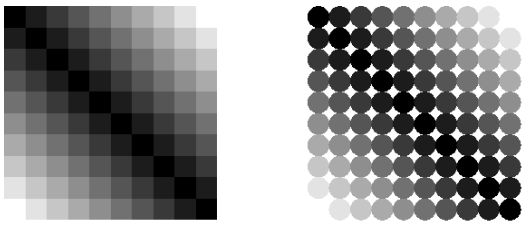
Neste trabalho, nosso foco está nos problemas inversos relacionados a restauração de imagens digitais e como podemos explorar este tema para facilitar a compreensão de subespaços. Neste capítulo introduziremos como é feita a representação de uma imagem em

formato digital, passando por diferentes formatos de cor e modelagem de alguns processos de embaçamento.

2.1 Captura e representação de imagens

O processo de reprodução de uma imagem real em formato digital basicamente passa, entre outras coisas, pela etapa de captura da luz pela câmera e a interpretação dela em forma de um conjunto de pequenos elementos que juntos formam a imagem. Esses "pequenos elementos" são chamados de pixels e carregam a informação sobre a área de uma região da figura. Cabe ressaltar que os pixels em uma imagem digital são, geralmente, representados por quadrados ou retângulos. Apesar de outros formatos como círculos ou linhas serem possíveis, o formato quadrado/retangular é mais comumente usado em monitores, câmeras digitais, celulares, entre outros, por não deixar "espaços" entre si. Ilustramos tal "falha" na Figura 2.2. Perceba os "espaços" não preenchidos quando os pixels são redondos. Tendo isto em mente, para nós o termo "pixel" será sempre retangular/quadrado.

Figura 2.2: Representação de pixels em formato quadrado (esquerda) e redondo (direita)



Por exemplo, na Figura 2.3 temos duas imagens com quantidades de pixels diferentes, a da esquerda possui 18×18 pixels e, por estar ampliada, é fácil identificá-los, pois são quadrados relativamente grandes, enquanto a imagem da direita possui 512×512 pixels e, neste caso, estes são tão menores que se tornam imperceptíveis.

Então, podemos concluir que quanto maior a quantidade de pixels, melhor é a qualidade da imagem? Não necessariamente, veja, por exemplo, a Figura 2.4 em que temos duas imagens, uma de 252×252 (esquerda) e uma de 512×512 (direita). Apesar de 252×252 ter menos de um quarto dos pixels de 512×512, esta é mais nítida do que a segunda que está fora de foco e não nos diz muita coisa. Portanto, em termos de qualidade, a capacidade do equipamento de conseguir registrar muitos pixels não é suficiente

Figura 2.3: Exemplo de figura 18×18 pixels (esquerda) e 512×512 pixels (direita)

se a imagem exata sofrer qualquer tipo de alteração ou perturbação durante o processo de digitalização.

Figura 2.4: Exemplo de imagem 252×252 pixels (esquerda) e 512×512 pixels (direita)

No que diz respeito às cores, as imagens podem ser representadas em preto e branco, tons de cinza (como na Figura 2.3) ou coloridas, além de existirem diferentes formas de quantificar os pixels. Por exemplo, para imagens em tons de cinza é comum usar uma escala de valores inteiros no intervalo [0, 255] ou [0, 65.535], em que o mínimo (0) representa o preto e o máximo (255 ou 65.535) representa o branco; enquanto imagens coloridas usam diferentes modelos de cores como RGB, HSV e CMY/CMYK em que as imagens são composta por múltiplas "camadas" ou canais de cor (*layers*).

No modelo RGB são usados três canais, um para cada cor primária da luz: vermelho (do inglês, *Red*), verde (*Green*) e azul (*Blue*) (HANSEN; NAGY; O'LEARY, 2006), como mostra a Figura 2.5. Cada uma dessas três cores pode ser interpretada como um vetor da base de um espaço de 3 dimensões, então cada pixel é a combinação linear dessa base. Esse modelo é considerado aditivo, é usado para representar cores emitidas (ou projetadas) por fontes de luz, em que a junção das três cores primárias gera o branco e a ausência delas gera o preto, ele é comumente usado em dispositivos digitais coloridos como os monitores,

câmeras, *scanners* e afins.

Figura 2.5: Exemplo de imagem decomposta em RGB e sua versão original

Fonte: <https://unsplash.com/photos/vXJ7uv2haSo>, 2021

No modelo CMY as camadas correspondem às cores ciano (do inglês, *Cyan*), magenta (*Magenta*) e amarelo (*Yellow*), veja a Figura 2.6.

Figura 2.6: Exemplo de imagem decomposta em CMY e sua versão original

Fonte: <https://unsplash.com/photos/a53e2Zwz7xY>, 2021

Ao contrário do RGB, esse modelo é subtrativo e representa a luz refletida (como um filtro que absorve/subtrai as demais cores), nesse caso, teoricamente, a presença das três cores gera o preto (todas as cores são absorvidas, subtrai-se a luz) e a ausência das cores produz o branco (nenhuma cor da luz é absorvida). A conversão de imagens em RGB (com valores entre 0 e 255) para CMY (com valores entre 0 e 1) é dada por

$$C = 1 - (R/255), \quad M = 1 - (G/255) \quad \text{e} \quad Y = 1 - (B/255).$$

E fazendo as operações inversas, temos a conversão contrária:

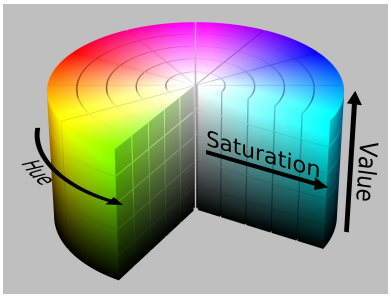
$$R = (1 - C)\,255, \quad G = (1 - M)\,255 \quad \text{e} \quad B = (1 - Y)\,255.$$

Semelhante a este, no formato CMYK, além das três citadas anteriormente, é acrescentada uma camada de cor preta (*blacK*), que para o uso em impressoras e fotocopiadoras ajuda a melhorar a qualidade de impressões, pois na prática o uso das três cores para gerar

o preto pode não produzir o efeito ideal e ainda deixar o papel muito úmido e sensível por conta da quantidade de tinta utilizada.

Em uma abordagem diferente das anteriores, temos o HSV que significa matiz (do inglês, *Hue*), saturação (*Saturation*) e valor (*Value*). O modelo HSV utiliza um espaço de cor cilíndrico e as camadas não representam cores como RGB e CMY. Para melhor compreensão, a Figura 2.7 ilustra como cada parâmetro é interpretado neste espaço.

Figura 2.7: Cilindro sólido de cor HSV



Fonte: <https://commons.wikimedia.org>, 2021

Conforme a ilustração, matiz é a tonalidade e em geral varia de 0 a 360, a saturação representa a "pureza" e varia de 0 a 1 (raio do cilindro), sendo que quanto mais próximo de zero mais cinza é a cor e o valor diz respeito ao brilho e também varia de 0 a 1 (altura do cilindro). Para exemplificar a decomposição da imagem em HSV temos a Figura 2.8, que dessa vez apresentamos cada uma das camadas em tons de cinza, pois, como mencionamos anteriormente, neste formato cada componente da imagem não corresponde a uma cor propriamente, mas atua como parâmetro para definir a cor final.

Figura 2.8: Exemplo de imagem decomposta em HSV e sua versão original

Fonte: <https://unsplash.com/photos/wNypLh377_o>, 2021

Considerando uma imagem em RGB com entradas R, G e B num intervalo de 0 a 1 e denotando $\zeta_M = max\{R, G, B\}$ e $\zeta_m = min\{R, G, B\}$, podemos converter o formato de cor da seguinte forma

$$H = \begin{cases} 0, \text{ se } \zeta_M = \zeta_m \\ 60 \cdot \left(\dfrac{G-B}{\zeta_M - \zeta_m} \bmod 6 \right), \text{ se } \zeta_M = R \text{ e} \\ 60 \cdot \dfrac{B-R}{\zeta_M - \zeta_m} + 120, \text{ se } \zeta_M = G \\ 60 \cdot \dfrac{R-G}{\zeta_M - \zeta_m} + 240, \text{ se } \zeta_M = B \end{cases},$$

$$S = \begin{cases} \dfrac{\zeta_M - \zeta_m}{\zeta_M}, \text{ se } \zeta_M > 0 \\ 0, \text{ se } \zeta_M = 0 \end{cases},$$

$$V = \zeta_M.$$

Neste trabalho vamos usar imagens em tons de cinza, em que cada imagem é representada por apenas uma matriz de pixels. Para ilustrar, vamos considerar a flor de 18×18 pixels que está na Figura 2.3 (esquerda). Neste caso temos uma matriz 18×18 cujas entradas variam de 0 até 255, essa matriz é apresentada a seguir.

$$\begin{bmatrix}
255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 \\
255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 0 & 0 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 \\
255 & 255 & 255 & 0 & 255 & 255 & 255 & 0 & 255 & 255 & 0 & 255 & 255 & 255 & 0 & 255 & 255 & 255 \\
255 & 255 & 0 & 255 & 0 & 255 & 0 & 255 & 255 & 255 & 255 & 0 & 255 & 0 & 221 & 0 & 255 & 255 \\
255 & 255 & 0 & 255 & 255 & 0 & 255 & 255 & 255 & 255 & 255 & 255 & 0 & 255 & 255 & 0 & 255 & 255 \\
255 & 0 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 221 & 0 & 255 \\
255 & 0 & 255 & 255 & 255 & 0 & 0 & 255 & 255 & 255 & 0 & 0 & 255 & 255 & 255 & 221 & 0 & 255 \\
255 & 0 & 255 & 255 & 0 & 255 & 255 & 0 & 255 & 0 & 255 & 255 & 0 & 255 & 255 & 221 & 0 & 255 \\
255 & 0 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 221 & 221 & 0 & 255 \\
255 & 255 & 0 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 221 & 0 & 255 & 255 \\
255 & 255 & 255 & 0 & 221 & 255 & 255 & 255 & 255 & 255 & 255 & 221 & 221 & 221 & 0 & 255 & 255 & 255 \\
255 & 255 & 255 & 255 & 0 & 0 & 221 & 221 & 221 & 221 & 221 & 221 & 0 & 0 & 255 & 255 & 255 & 255 \\
255 & 255 & 255 & 255 & 255 & 255 & 0 & 0 & 0 & 0 & 0 & 0 & 255 & 255 & 255 & 255 & 255 & 255 \\
255 & 255 & 255 & 0 & 0 & 0 & 255 & 0 & 100 & 46 & 0 & 0 & 0 & 0 & 255 & 255 & 255 & 255 \\
255 & 255 & 0 & 100 & 100 & 46 & 0 & 0 & 100 & 46 & 0 & 100 & 100 & 46 & 0 & 255 & 255 & 255 \\
255 & 0 & 100 & 100 & 100 & 100 & 46 & 0 & 100 & 0 & 100 & 100 & 100 & 100 & 46 & 0 & 255 & 255 \\
255 & 0 & 46 & 46 & 46 & 46 & 46 & 0 & 0 & 0 & 46 & 46 & 46 & 46 & 46 & 0 & 255 & 255 \\
255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255
\end{bmatrix}.$$

Perceba que o que mais chama atenção nessa matriz são as bordas da flor que são pretas e aparecem como zeros, enquanto os demais pixels são claros e aparecem com números de dois ou três dígitos. Os casos de figuras coloridas são análogos, com uma matriz para cada camada de cor.

A grande questão sobre as imagens digitais é que nem sempre elas representam a imagem real como desejado, conforme Figura 2.4, isso por inúmeros motivos, tais como a distorção provocada pelas lentes da câmera que fez a captura, por exemplo, fora de foco, movimento dos objetos da imagem, interferências atmosféricas, etc. Em outras palavras, a imagem capturada contém algum tipo de "embaçamento".

A fim de restaurar imagens via métodos matemáticos, é necessário ter um modelo que relacione a imagem embaçada com a imagem exata desejada. Nessa dissertação partimos do pressuposto que essa operação que ocorre de uma imagem à outra é linear, pois na física muitas vezes efetivamente é o que ocorre (HANSEN; NAGY; O'LEARY, 2006). Se denotarmos a imagem exata de $M \times N$ pixels como uma matriz X (também de dimensão $M \times N$) e a imagem embaçada de $M \times N$ pixels como uma matriz B, então existe uma matriz A de tamanho $MN \times MN$ que relaciona as imagens X e B por

$$Ax = b,$$

em que A modela o processo de embaçamento, $x = \text{vec}(X)$ e $b = \text{vec}(B)$. A operação "vec" nada mais é do que transformar uma matriz "$M \times N$" em um vetor coluna "empilhando" as colunas da matriz, por exemplo,

$$B = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{bmatrix} \quad \Rightarrow \quad b = \text{vec}(B) = \begin{bmatrix} b_{11} \\ b_{21} \\ b_{12} \\ b_{22} \\ b_{13} \\ b_{23} \end{bmatrix}.$$

Assim, se A é não singular, teoricamente $x = A^{-1}b$. Infelizmente, considerando nosso contexto, mesmo as menores figuras estão associadas a grandes matrizes, por exemplo, uma imagem de 64×64 pixels gera um sistema linear com A na ordem de 4.096×4.096.

Do ponto de vista computacional, calcular a inversa de matrizes é um processo "proibido", além, é claro, da possibilidade da inversa nem existir. Ademais, em aplicações práticas os dados (imagem capturada) raramente são exatos, isto é, dispomos apenas de uma imagem $b = b_{exato} + e$, em que e é um vetor que representa o erro ou ruído nos dados (erros numéricos de arredondamento/truncamento, erros de medição, etc) e b_{exato} é a imagem livre de ruídos. Logo,

$$x = A^{-1}b = A^{-1}(b_{exato} + e) = A^{-1}b_{exato} + A^{-1}e,$$

ou seja, a imagem restaurada contém, além da imagem desejada, algum tipo de artefato.

2.2 Modelos de embaçamento

Como mencionado anteriormente, vamos considerar embaçamentos provenientes de fenômenos físicos que podem ser aproximados por modelos lineares. Chamamos de PSF, do inglês *Point Spread Function*, que em tradução livre significa Função de Propagação do Ponto, a função que modela o embaçamento e determina a matriz A.

Figura 2.9: Um ponto de luz e sua PSF à direita

Por ser linear, a PSF pode ser representada por uma matriz P. Em alguns casos, a PSF pode ser descrita analiticamente e encontramos a matriz P associada através de uma função (ao invés de experimentação). Em outros casos, conhecemos o processo físico que provoca o embaçamento, então P é obtida por meio de expressões matemáticas. Por exemplo, em uma imagem fora de foco, os elementos p_{ij} são obtidos de

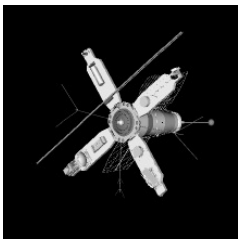
$$p_{ij} = \begin{cases} \dfrac{1}{\pi r^2}, \text{ se } (i-k)^2 + (j-l)^2 \leq r^2 \\ 0, \text{ caso contrário} \end{cases} \tag{2.2}$$

em que (k, l) é o centro de P e r é o raio do desfoque (HANSEN; NAGY; O'LEARY,

2006). Então obtemos A colocando os elementos de P nas posições apropriadas, conforme exploraremos mais adiante. Na prática, uma vez que temos a PSF, a matriz A é criada com auxílio do pacote *RestoreTools* (NAGY et al., 2007).

Para ilustrar, criamos algumas PSFs e aplicamos sobre a Figura 2.10 de um satélite.

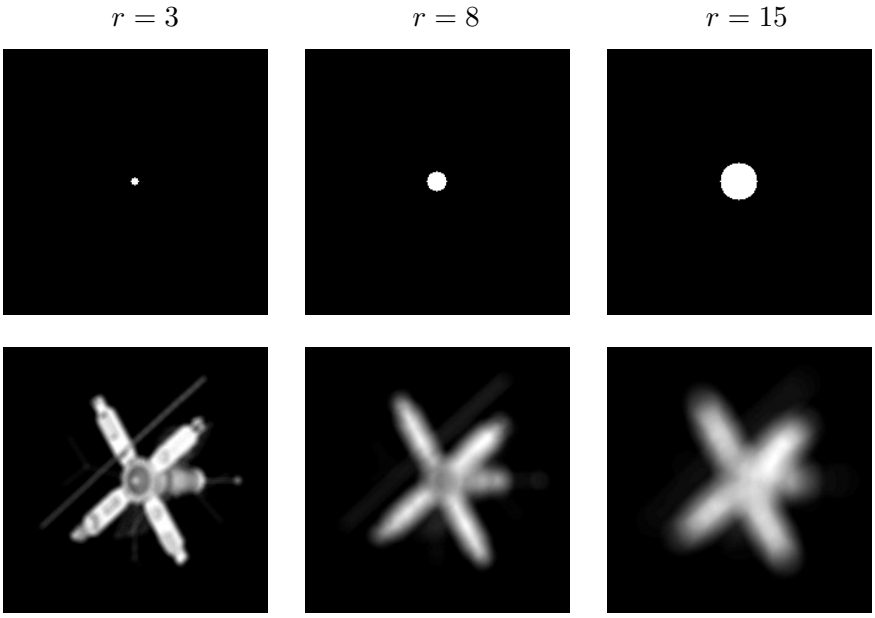
Figura 2.10: Satélite de 216×216 pixels



Na literatura geralmente aparecem apenas PSFs com centro da PSF coincidindo com o centro da imagem, caso mais comum de acontecer considerando os fenômenos físicos, então para os exemplos que apresentamos a seguir adotamos o centro coincidindo com o centro da imagem, nesse caso, $(k, l) = (108, 108)$.

Na Figura 2.11, ilustramos o efeito fora de foco para três raios r diferentes. Fica

Figura 2.11: Exemplos de PSF fora de foco, com diferentes raios, e respectivas imagens embaçadas



evidente que quanto maior o raio pior é a qualidade da imagem, pois cada pixel da imagem original interfere na região do entorno delimitada pelo raio, assim quanto maior

o raio de embaçamento, maior a influência do pixel sobre seus vizinhos e vice-versa, então a imagem final acaba perdendo a sensibilidade de captura dos detalhes e contornos.

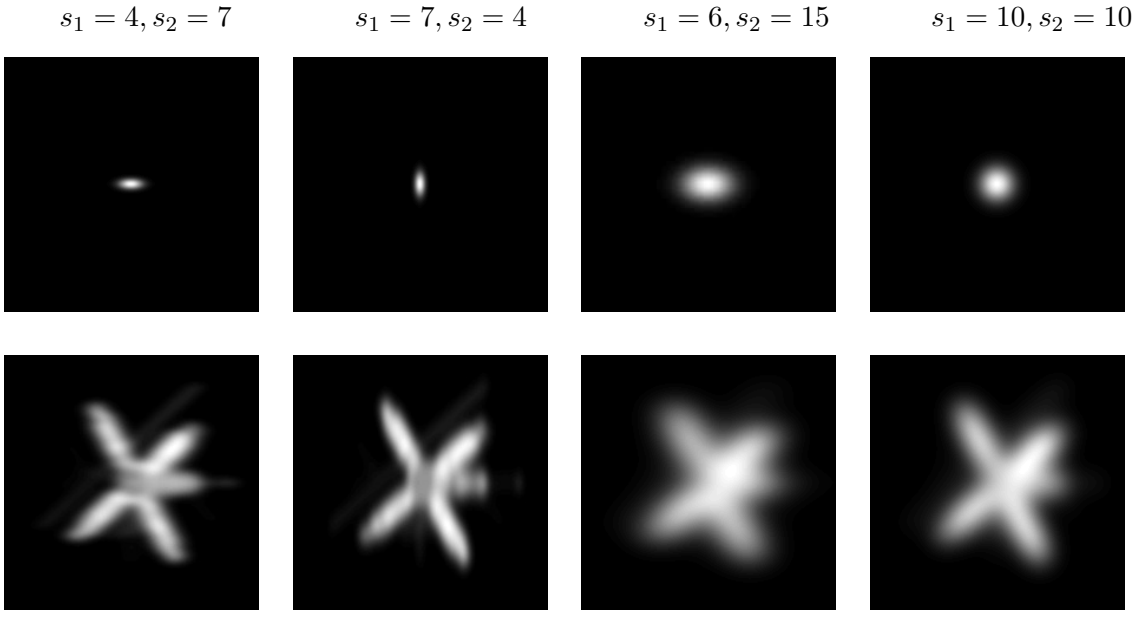
Cabe ressaltar que a PSF busca modelar o espalhamento de um ponto de luz mantendo seu brilho inalterado, isto é, a quantidade de energia capturada deve ser mantida, então a soma de todos os elementos da matriz P deve ser igual a 1.

Outro fenômeno conhecido são as turbulências atmosféricas, que podem ser modeladas como uma função gaussiana bidimensional, cujos elementos da matriz P são dados por

$$p_{ij} = \exp\left(-\frac{1}{2}\begin{bmatrix} i-k \\ j-l \end{bmatrix}^T \begin{bmatrix} s_1^2 & \rho^2 \\ \rho^2 & s_2^2 \end{bmatrix}^{-1} \begin{bmatrix} i-k \\ j-l \end{bmatrix}\right) \tag{2.3}$$

em que s_1, s_2 e ρ determinam a largura e orientação da PSF, centrada no elemento (k, l) (HANSEN; NAGY; O'LEARY, 2006). Analogamente nas Figuras 2.12 e 2.13 ilustramos

Figura 2.12: Exemplos de PSF gaussiana para turbulência atmosférica, com $\rho = 0$ e diferentes s_1 e s_2, e respectivas imagens embaçadas

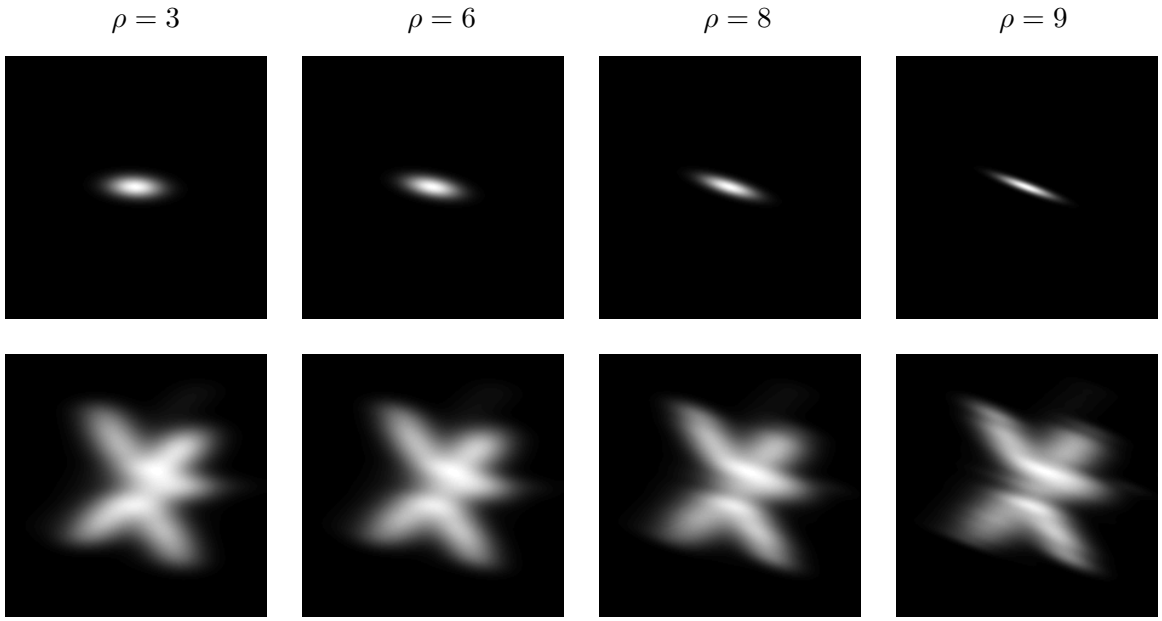


algumas PSFs desse fenômeno, variando s_1 e s_2 na primeira e ρ na segunda, a fim de perceber melhor a influência dos parâmetros no resultado final.

Podemos perceber que para $\rho = 0$ a PSF é simétrica em relação ao eixos horizontal e vertical. Nessas direções a amplitude é definida pelos parâmetros s_1 e s_2, respectivamente, e ao modificar o parâmetro ρ, a orientação do embaçamento fica mais diagonalizada.

Em particular, temos a PSF gaussiana em que o espalhamento é simétrico em todas

Figura 2.13: Exemplos de PSF gaussiana para turbulência atmosférica, com $s_1 = 6$, $s_2 = 15$ e diferentes ρ, e respectivas imagens embaçadas



as direções em relação ao centro (como acontece quando $s_1 = s_2$ e $\rho = 0$ no caso anterior). Então a matriz é obtida por

$$p_{ij} = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(i-k)^2 + (j-l)^2}{2\sigma^2}\right)$$

em que σ é o grau de dispersão e (k, l) é o centro. Na Figura 2.14 exemplificamos essa distribuição para três diferentes σ.

Outro efeito comum é o *Motion Blur*, ou borrão de movimento, causado por movimentos rápidos do objeto que está sendo capturado ou da câmera ou então quando a captura é feita por longa exposição. Ilustramos na Figura 2.15 a PSF do *Motion Blur* horizontal com diferentes intensidades r. Fenômeno análogo acontece em outras direções.

Em nossos exemplos usamos uma mesma PSF para embaçar todos os pixels da figura, quando isso acontece dizemos que o embaçamento é *espacialmente invariável* (HANSEN; NAGY; O'LEARY, 2006). Nem sempre é assim, mas é o que acontece na maioria das vezes, então neste trabalho tratamos apenas desse caso.

Finalmente, obtemos A colocando os elementos de P nas posições apropriadas, de tal forma que a A descreva a *convolução* da PSF com a imagem verdadeira. Para embasar, temos a definição 2.2.1 a seguir.

Figura 2.14: Exemplos de PSF gaussiana, com diferentes σ, e respectivas imagens embaçadas

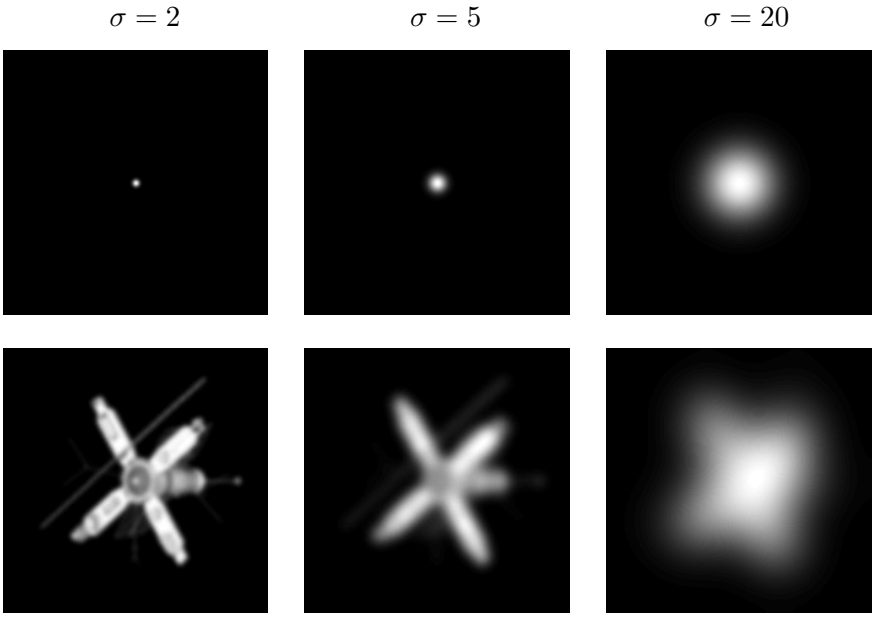
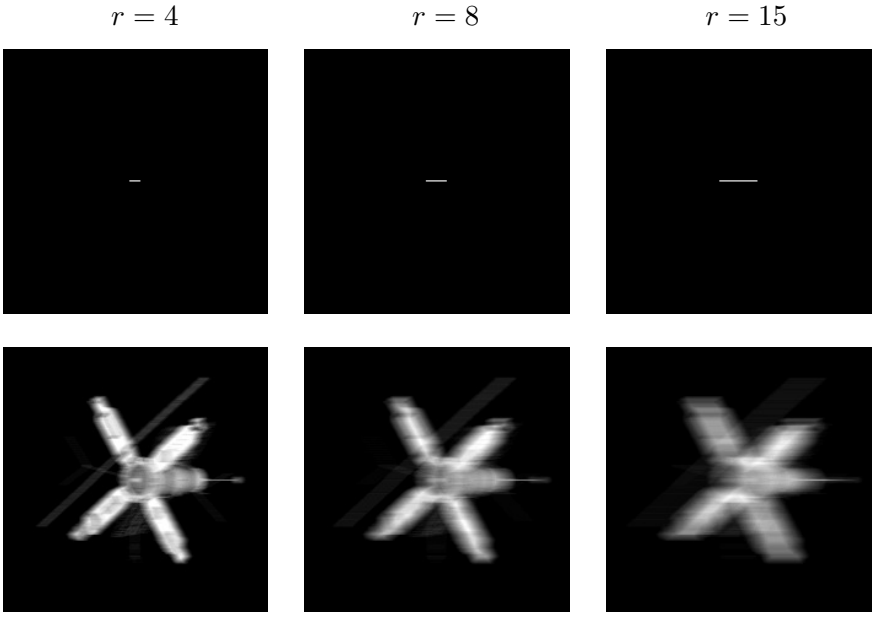


Figura 2.15: Exemplos de PSF *Motion Blur* horizontal, com diferentes intensidades, e respectivas imagens embaçadas



Definição 2.2.1. *Dadas duas funções p e x contínuas por partes para $s \geq 0$, a convolução de p e x é a função definida por:*

$$b(s) = \int_{-\infty}^{\infty} p(s-t)\ x(t)\ dt,$$

Também denotamos a convolução por $(p * x)(s)$ ou $p(s) * x(s)$.

Nesse caso, podemos interpretar cada valor de $b(s)$ como a média ponderada de $x(t)$, em que os pesos são dados por p (HANSEN; NAGY; O'LEARY, 2006). Essa é uma operação que tem propriedades associativa, comutativa, distributiva, entre outras.

No caso discreto, a convolução é obtida por uma soma finita de termos. Então, em nosso contexto, os pixels da imagem embaçada são obtidos pela soma ponderada dos pixels correspondentes e seus vizinhos na imagem real, sendo que os pesos são dados pela matriz PSF.

Por simplicidade, vamos exemplificar o caso de uma imagem unidimensional em que desprezamos as informações fora da fronteira da imagem, para outros tratamentos da fronteira sugerimos Hansen, Nagy e O'Leary (2006). Sejam a PSF e a imagem exata dadas pelas seguintes matrizes:

$$p = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \end{bmatrix} \quad \text{e} \quad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix},$$

a convolução de p com x é calculada conforme os seguintes passos:

- Giramos p em 180° (no caso unidimensional equivale a escrevermos seus elementos de baixo para cima);
- Alinhamos o centro da PSF com a i-ésima entrada de x;
- E fazemos as somas dos produtos dos pares correspondentes.

Assumindo que o centro da PSF é p_3, então, por exemplo, $b_1 = p_3x_1 + p_2x_2 + p_1x_3$ e a convolução pode ser escrita como o seguinte produto de matrizes:

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{bmatrix} = \begin{bmatrix} p_3 & p_2 & p_1 & & \\ p_4 & p_3 & p_2 & p_1 & \\ p_5 & p_4 & p_3 & p_2 & p_1 \\ & p_5 & p_4 & p_3 & p_2 \\ & & p_5 & p_4 & p_3 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}, \tag{2.4}$$

assim obtemos a matriz A, que é a matriz de coeficientes p_i de (2.4).

Analogamente, no caso bidimensional

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix} \quad \text{e} \quad X = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix},$$

em que p_{22} é o centro da PSF, temos por exemplo que

$$\begin{aligned} b_{11} = \;& p_{22}x_{11} + p_{21}x_{12} \\ + \;& p_{12}x_{21} + p_{11}x_{22} \end{aligned}$$

e

$$\begin{aligned} b_{22} = \;& p_{33}x_{11} + p_{32}x_{12} + p_{31}x_{13} \\ + \;& p_{23}x_{21} + p_{22}x_{22} + p_{21}x_{23} \\ + \;& p_{13}x_{31} + p_{12}x_{32} + p_{11}x_{33} \end{aligned}$$

então relacionamos $b = \text{vec}(B)$ e $x = \text{vec}(X)$ por

$$\begin{bmatrix} b_{11} \\ b_{21} \\ b_{31} \\ b_{12} \\ b_{22} \\ b_{32} \\ b_{13} \\ b_{23} \\ b_{33} \end{bmatrix} = \begin{bmatrix} p_{22} & p_{12} & & p_{21} & p_{11} & & & & \\ p_{32} & p_{22} & p_{12} & p_{31} & p_{21} & p_{11} & & & \\ & p_{32} & p_{22} & & p_{31} & p_{21} & & & \\ p_{23} & p_{13} & & p_{22} & p_{12} & & p_{21} & p_{11} & \\ p_{33} & p_{23} & p_{13} & p_{32} & p_{22} & p_{12} & p_{31} & p_{21} & p_{11} \\ & p_{33} & p_{23} & & p_{32} & p_{22} & & p_{31} & p_{21} \\ & & & p_{23} & p_{13} & & p_{22} & p_{12} & \\ & & & p_{33} & p_{23} & p_{13} & p_{32} & p_{22} & p_{12} \\ & & & & p_{33} & p_{23} & & p_{32} & p_{22} \end{bmatrix} \cdot \begin{bmatrix} x_{11} \\ x_{21} \\ x_{31} \\ x_{12} \\ x_{22} \\ x_{32} \\ x_{13} \\ x_{23} \\ x_{33} \end{bmatrix}. \qquad (2.5)$$

Para mais informações sobre PSF e a construção da matriz A recomendamos Hansen, Nagy e O'Leary (2006).

Portanto, neste capítulo vimos que uma imagem digital pode ser representada como uma matriz (ou mais, a depender do formato de cor), vimos também que muitas vezes os processos de embaçamento são lineares e podem ser descritos por PSFs, que representam

como um ponto de luz se espalha após embaçamento e, dada uma PSF, construímos a matriz A responsável por embaçar a imagem inteira. Então modelamos o problema de restauração de imagens como um sistema $Ax = b$ de grande porte, em que o embaçamento $A \in \mathbb{R}^{mn \times mn}$ e a imagem embaçada $b \in \mathbb{R}^{mn}$ são conhecidos.

No capítulo a seguir propomos dois métodos para resolver problemas inversos, dado que para esse tipo de problema é completamente inviável resolver por métodos diretos.

Capítulo 3

Métodos Iterativos

Vimos no capítulo anterior que uma das etapas finais do processo de restauração envolve a obtenção da solução de um sistema linear, representado matricialmente por

$$Ax = b, \tag{3.1}$$

que em um contexto mais genérico, A é a matriz de coeficientes $m \times n$ (neste trabalho contemplamos apenas o caso $m \geq n$ e A tem posto completo, ou seja, posto$(A) = n$), b é o vetor do lado direito ou vetor de dados e x é o vetor de incógnitas.

Teoricamente, a solução deste problema pode ser obtida como $x = A^{-1}b$ ou $x = A^{\dagger}b$, em que † denota a pseudo-inversa da matriz A – para mais informações sobre a pseudo-inversa, sugerimos ver o livro *Matrix Analysis and Applied Linear Algebra* (MEYER, 2000). Entretanto, para problemas de grande porte, a obtenção da solução é inviável por métodos diretos, tais como eliminação gaussiana (BURDEN; FAIRES; BURDEN, 2016) ou métodos baseados em fatorações matriciais como LU e QR. Alternativamente, apresentamos na sequência o conceito de métodos iterativos e métodos de projeção, sendo que nosso foco neste capítulo são dois populares métodos de projeção em subespaços: TSVD e LSQR.

3.1 O que são métodos iterativos e métodos de projeção?

Dado o problema da equação (3.1) desejamos obter, se existir, uma solução para o sistema, isto é, encontrar x^* que satisfaz a igualdade ou então, caso não exista solução,

encontrar x_{LS} solução do problema de mínimos quadrados

$$x_{\text{LS}} = \underset{x \in \mathbb{R}^n}{\operatorname{argmin}} \|b - Ax\|_2, \tag{3.2}$$

nesse caso, queremos minimizar a norma do resíduo $r = b - Ax$.

Existem várias formas de resolver tanto o problema (3.1) quanto o problema (3.2). Entre elas estão os métodos iterativos, que basicamente são algoritmos que geram uma sequência de vetores que converge para a solução desejada, isto é, dada uma aproximação inicial x_0, a cada nova iteração obtemos um novo x_k e $x_k \to x^*$ ou $x_k \to x_{\text{LS}}$. Entretanto, muitas das vezes os métodos só convergem sob determinadas condições. Por exemplo, iterações de Landweber formam uma sequência obtida por $x_{k+1} = x_k + \gamma A^T(b - Ax_k)$, em que $\gamma \in \mathbb{R}$ e usualmente $x_0 = 0$, nesse caso a convergência depende do parâmetro γ, isto é, a sequência converge para x_{LS} se $0 < \gamma < \frac{2}{\overline{\gamma}}$, em que $\overline{\gamma}$ é o maior autovalor da matriz A^TA (DAX, 1990). Ou o método SOR no qual a sequência é construída como,

$$[x_k]_i = (1-\omega)[x_{k-1}]_i + \frac{\omega}{a_{ii}}\left[b_i - \sum_{j=1}^{i-1} a_{ij}[x_k]_j - \sum_{j=i+1}^{n} a_{ij}[x_{k-1}]_j\right]$$

em que $[x_k]_i$ denota o i-ésimo elemento do vetor x_k. Neste caso, a convergência é garantida se, e somente se, $0 < \omega < 2$ (HADJIDIMOS, 1999).

Uma classe particular de métodos iterativos são os métodos de projeção em subespaços. Considere, no $\mathbb{R}^n$, uma sequência de subespaços $V_1,\ V_2, \cdots, V_k$ encaixantes, isto é, $V_k \subset V_{k+1}$ e $\dim(V_k) = k$. O objetivo dos métodos de projeção é construir uma sequência de vetores $x_1, x_2, ..., x_k$ tal que

$$x_k = \underset{x \in V_k}{\operatorname{argmin}} \|b - Ax\|_2. \tag{3.3}$$

Por serem subespaços encaixantes, à medida que aumentamos a dimensão do subespaço, a norma do resíduo $r_k = b - Ax_k$ torna-se menor ou se mantém, isto é, x_k minimiza a norma do resíduo em V_k.

Teorema 3.1.1. *Seja x_k, $k \geq 1$, dada por (3.3), então $\|r_{k+1}\|_2 \leq \|r_k\|_2$ para $k \geq 1$.*

Demonstração. Como x_{k+1} satisfaz

$$x_{k+1} = \underset{x \in V_{k+1}}{\operatorname{argmin}} \|b - Ax\|_2$$

então

$$\|r_{k+1}\|_2 = \|b - Ax_{k+1}\|_2 \leq \|b - Ax\|_2, \ \forall x \in V_{k+1}.$$

Como $V_k \subset V_{k+1}$, então $x_k \in V_{k+1}$, logo

$$\|r_{k+1}\|_2 \leq \|b - Ax_k\|_2 = \|r_k\|_2.$$

□

A seguir temos um teorema que garante a convergência da sequência x_k.

Teorema 3.1.2. *Seja V_k uma sequência de subespaços encaixantes, isto é, $V_k \subset V_{k+1}$ e $dim(V_k) = k$. Se x_k é a sequência definida por (3.3), então $x_k \to x_{LS}$ quando $k \to n$.*

Demonstração. Como $V_k \subset \mathbb{R}^n$, e $\dim(V_k) = k$, então se $k = n$ temos $V_n = \mathbb{R}^n$, logo

$$x_n = \underset{x \in V_n}{\operatorname{argmin}} \|b - Ax\|_2 = \underset{x \in \mathbb{R}^n}{\operatorname{argmin}} \|b - Ax\|_2 = x_{\mathrm{LS}}.$$

□

Existem diversos métodos na literatura e aqui descreveremos a seguir dois métodos clássicos.

3.2 Solução de mínimos quadrados em termos da SVD e o método da TSVD

Um método para encontrar a solução de mínimos quadrados é através da decomposição em valores singulares (do inglês, *Singular Value Decomposition* - SVD) da matriz A, que mostraremos a seguir, mas primeiro apresentamos um outro resultado útil.

Teorema 3.2.1. *Se $V_1 \in \mathbb{R}^{n \times r}$, $r < n$, tem colunas ortonormais, então existe $V_2 \in \mathbb{R}^{n \times (n-r)}$, de forma que $V = [\ V_1 \mid V_2\ ]$ é ortogonal.*

Demonstração. Se $V = [\ v_1 \mid \cdots \mid v_n\ ] \in \mathbb{R}^{n \times n}$ é uma matriz ortogonal, então o conjunto de v_i para $i = 1, 2, \cdots, n$ forma uma base ortonormal para R^n. Logo, se $V_1 = [\ v_1 \mid \cdots \mid v_r\ ]$ tem colunas ortonormais, podemos construir o complemento ortogonal de V_1 para o $\mathbb{R}^n$ (STEINBRUCH; WINTERLE, 1987), que é $V_2 = [\ v_{r+1} \mid \cdots \mid v_n\ ]$, tal que $V = [\ V_1 \mid V_2\ ]$ seja ortogonal. □

Teorema 3.2.2 (SVD)**.** *Se A é uma matriz real de ordem $m \times n$, então existem matrizes ortogonais*

$$U = \left[\begin{array}{ccc} u_1 & \cdots & u_m \end{array} \right] \in \mathbb{R}^{m \times m} \quad e \quad V = \left[\begin{array}{ccc} v_1 & \cdots & v_n \end{array} \right] \in \mathbb{R}^{n \times n}$$

tais que

$$U^T A V = \Sigma = diag(\sigma_1, \sigma_2, \cdots, \sigma_p) \in \mathbb{R}^{m \times n}, \quad p = min\{m, n\} \tag{3.4}$$

em que $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0$.

Demonstração. Dados $x \in \mathbb{R}^n$ e $y \in \mathbb{R}^m$ vetores unitários (na norma euclidiana) que satisfazem $Ax = \sigma y$, em que $\sigma = \|A\|_2$. Então, pelo Teorema 3.2.1, existem $V_2 \in \mathbb{R}^{n \times (n-1)}$ e $U_2 \in \mathbb{R}^{m \times (m-1)}$, tais que $V = [\ x \mid V_2\ ] \in \mathbb{R}^{n \times n}$ e $U = [\ y \mid U_2\ ] \in \mathbb{R}^{m \times m}$ são ortogonais. Disso podemos dizer que

$$U^T A V = \left[\begin{array}{cc} \sigma & w^T \\ 0 & B \end{array} \right] \equiv A_1$$

em que $w \in \mathbb{R}^{n-1}$ e $B \in \mathbb{R}^{(m-1) \times (n-1)}$. Então

$$\left\| A_1 \left[\begin{array}{c} \sigma \\ w \end{array} \right] \right\|_2^2 = \left\| \left[\begin{array}{c} \sigma^2 + w^T w \\ Bw \end{array} \right] \right\|_2^2 \geq (\sigma^2 + w^T w)^2,$$

logo $\|A_1\|_2^2 \geq (\sigma^2 + w^T w)$. Porém, pela construção, $\sigma^2 = \|A\|_2^2 = \|A_1\|_2^2$, portanto temos que $w = 0$. Por indução, a demonstração está completa. □

Como em nosso contexto $m \geq n$ e A tem posto completo, então $\sigma_1 \geq \cdots \geq \sigma_n > 0$. Nesse teorema os σ_i são os valores singulares de A, isto é, são as raízes quadradas dos autovalores não nulos de AA^T e $A^T A$; os u_i são os vetores singulares à esquerda de A, ou seja, os autovetores de AA^T e; os v_i são os vetores singulares à direita, nesse caso, autovetores de $A^T A$.

Tendo em vista que U e V são matrizes ortogonais, isto é, $U^{-1} = U^T$ e $V^{-1} = V^T$, temos, da equação (3.4), que

$$U^T A V = \Sigma \Rightarrow U U^T A V V^T = U \Sigma V^T \Rightarrow A = U \Sigma V^T.$$

Como a matriz A tem posto completo, então Σ também tem posto completo e, conse-

quentemente, $A^\dagger = (U\Sigma V^T)^\dagger = V\Sigma^\dagger U^T$.

No caso da matriz Σ temos

$$\Sigma^\dagger = \text{diag}\left(\frac{1}{\sigma_1}, \frac{1}{\sigma_2}, \cdots, \frac{1}{\sigma_n}\right) \in \mathbb{R}^{n\times m} = \begin{bmatrix} \frac{1}{\sigma_1} & 0 & \cdots & 0 & \cdots & 0 \\ 0 & \frac{1}{\sigma_2} & \cdots & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{1}{\sigma_n} & \cdots & 0 \end{bmatrix}.$$

Podemos usar a SVD da matriz A para encontrar a solução do problema de minimização (3.2), ou, no caso de existência de infinitas soluções, aquela que tem a menor norma. Ressaltamos que como A tem posto completo, então o problema (3.2) admite única solução.

Note que encontrar a solução do problema de minimização (3.2) é equivalente a encontrar a soluções das chamadas equações normais

$$A^T Ax = A^T b. \tag{3.5}$$

Se a matriz A tem posto completo, então $A^T A$ também tem posto completo e é, portanto, não singular, logo

$$x_{\text{LS}} = \underset{x\in\mathbb{R}^n}{\text{argmin}} \|b - Ax\|_2 \iff x_{\text{LS}} = (A^T A)^{-1} A^T b.$$

Como a pseudo-inversa da matriz A é

$$A^\dagger = (A^T A)^{-1} A^T,$$

então

$$x_{\text{LS}} = A^\dagger b.$$

Assim, aplicando a SVD da matriz A, encontramos

$$Ax = b \Rightarrow x_{\text{LS}} = A^\dagger b \Rightarrow x_{\text{LS}} = V\Sigma^\dagger U^T b = \sum_{i=1}^{n} \frac{u_i^T b}{\sigma_i} v_i. \tag{3.6}$$

Perceba que se A possui valores singulares σ_i muito pequenos, especialmente se $\sigma_i \ll |u_i^T b|$, então $\dfrac{|u_i^T b|}{\sigma_i} \gg 1$. A consequência disso é que a solução pode tornar-se sensível a pequenas perturbações em b, podendo gerar demasiada perturbação na solução x_{LS}, ou

seja, $x_{\text{exato}} = A^{\dagger} b_{\text{exato}}$ muito diferente de $x_{\text{LS}} = A^{\dagger} b$. Como na maioria dos problemas reais os dados possuem algum ruído e, então

$$x_{\text{LS}} = \sum_{i=1}^{n} \frac{u_i^T (b_{\text{exato}} + e)}{\sigma_i} v_i = \underbrace{\sum_{i=1}^{n} \frac{u_i^T b_{\text{exato}}}{\sigma_i} v_i}_{\text{Solução exata}} + \underbrace{\sum_{i=1}^{n} \frac{u_i^T e}{\sigma_i} v_i}_{\text{Erro}}. \tag{3.7}$$

Com o intuito de abater parte do erro da solução, o método da SVD truncada (no inglês, *Truncated SVD* - TSVD) consiste em construir a solução x_k no subespaço formado pelos k primeiros vetores singulares da matriz A, ou seja, em $\mathcal{V}_k = span\{v_1, v_2, \cdots, v_k\}$, então

$$x_k = \underset{x \in \mathcal{V}_k}{\operatorname{argmin}} \|b - Ax\|_2. \tag{3.8}$$

Perceba que se $x \in \mathcal{V}_k$, então x é combinação linear dos vetores da base $\{v_1, v_2, \cdots, v_k\}$, ou seja, $x = \begin{bmatrix} v_1 & v_2 & \cdots & v_k \end{bmatrix} y = V_k\, y$, para algum $y \in \mathbb{R}^k$. Logo,

$$x_k = \underset{x \in \mathcal{V}_k}{\operatorname{argmin}} \|b - Ax\|_2 = V_k\ y_k \iff y_k = \underset{y \in \mathbb{R}^k}{\operatorname{argmin}} \|b - AV_k y\|_2. \tag{3.9}$$

Nessas condições é interessante trabalhar com a SVD de A particionada como

$$A = \begin{bmatrix} U_k & U_{m-k} \end{bmatrix} \begin{bmatrix} \Sigma_k & 0 \\ 0 & \Sigma_0 \end{bmatrix} \begin{bmatrix} V_k^T \\ V_{n-k}^T \end{bmatrix} \tag{3.10}$$

em que U_k e V_k são matrizes formadas pelas k-ésimas primeiras colunas das respectivas matrizes, U_{m-k} e V_{n-k} são os demais elementos e $\Sigma_0 = \text{diag}(\sigma_{k+1}, \cdots, \sigma_n) \in \mathbb{R}^{(m-k)\times(n-k)}$. Então y_k pode ser obtido por

$$
\begin{aligned}
y_k &= (AV_k)^\dagger b \\
&= \left(\begin{bmatrix} U_k & U_{m-k} \end{bmatrix} \begin{bmatrix} \Sigma_k & 0 \\ 0 & \Sigma_0 \end{bmatrix} \begin{bmatrix} V_k^T \\ V_{n-k}^T \end{bmatrix} [\, V_k \,] \right)^\dagger b \\
&= \left(\begin{bmatrix} U_k & U_{m-k} \end{bmatrix} \begin{bmatrix} \Sigma_k & 0 \\ 0 & \Sigma_0 \end{bmatrix} \begin{bmatrix} \mathbb{I}_k \\ 0 \end{bmatrix} \right)^\dagger b \\
&= \left(\begin{bmatrix} U_k & U_{m-k} \end{bmatrix} \begin{bmatrix} \Sigma_k \\ 0 \end{bmatrix} \right)^\dagger b \\
&= (U_k \Sigma_k)^\dagger b \\
&= \Sigma_k^{-1} U_k^T b
\end{aligned}
\tag{3.11}
$$

e, finalmente, a k-ésima solução TSVD é

$$
x_k = V_k \, y_k = V_k \Sigma_k^{-1} U_k^T b. \tag{3.12}
$$

Abrindo as contas,

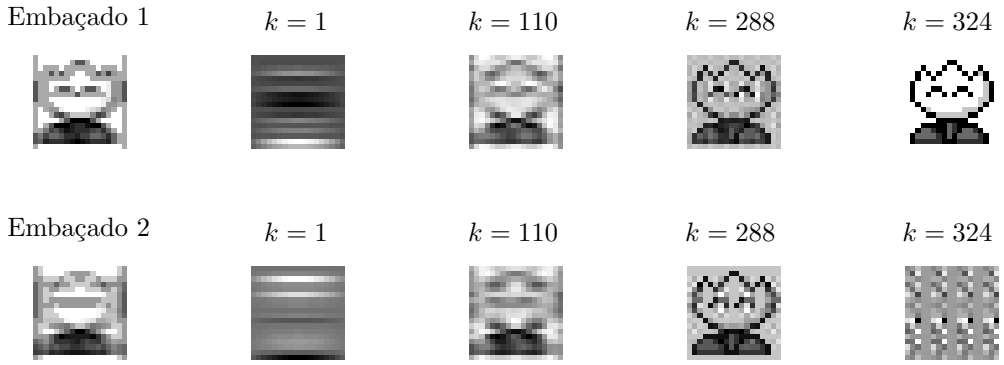
$$
\begin{aligned}
x_k &= V_k \Sigma_k^{-1} U_k^T b \\
&= \begin{bmatrix} \left(\dfrac{v_{11}u_{11}}{\sigma_1} + \cdots + \dfrac{v_{1k}u_{1k}}{\sigma_k} \right) b_1 + \cdots + \left(\dfrac{v_{11}u_{m1}}{\sigma_1} + \cdots + \dfrac{v_{1k}u_{mk}}{\sigma_k} \right) b_m \\ \vdots \\ \left(\dfrac{v_{n1}u_{11}}{\sigma_1} + \cdots + \dfrac{v_{nk}u_{1k}}{\sigma_k} \right) b_1 + \cdots + \left(\dfrac{v_{n1}u_{m1}}{\sigma_1} + \cdots + \dfrac{v_{nk}u_{mk}}{\sigma_k} \right) b_m \end{bmatrix} \\
&= \begin{bmatrix} \dfrac{u_1^T b}{\sigma_1} v_{11} + \cdots + \dfrac{u_k^T b}{\sigma_k} v_{1k} \\ \vdots \\ \dfrac{u_1^T b}{\sigma_1} v_{n1} + \cdots + \dfrac{u_k^T b}{\sigma_k} v_{nk} \end{bmatrix} \\
&= \frac{u_1^T b}{\sigma_1} v_1 + \cdots + \frac{u_k^T b}{\sigma_k} v_k = \sum_{i=1}^{k} \frac{u_i^T b}{\sigma_i} v_i,
\end{aligned}
$$

e note que este último somatório é similar à equação (3.6), ou seja, se truncarmos o

somatório da equação (3.6) em $k < n$, obtemos a k-ésima solução TSVD.

Para exemplificar, aplicamos o método em dois problemas de restauração, $A_1x = b_1$ e $A_2x = b_2$. Ambos foram gerados de uma figura pequena de 18×18 pixels que passou por processos de embaçamentos diferentes (matrizes A_1 e A_2 diferentes) que simulam uma imagem "tremida", o resultado é apresentado na Figura 3.1.

Figura 3.1: Comparação de algumas soluções do método TSVD para dois problemas diferentes



Em ambos os casos temos um sistema da forma $Ax = b$, com A de ordem 324×324 e o ruído nos dados é proveniente apenas de arredondamentos ou truncamentos numéricos. Na Figura 3.1 apresentamos para cada problema a imagem embaçada e quatro soluções TSVD, para $k = 1, 110, 288$ e 324, sendo a solução com $k = 324$ a de menor erro relativo no problema 1 e a solução com $k = 288$ a de menor erro relativo no problema 2.

Perceba que as soluções ficam mais próximas da imagem original à medida que aumentamos o número de iterações k. Entretanto, no segundo caso, as últimas iterações sofrem muito com a influência do ruído dos dados, mesmo sendo decorrente apenas de arredondamentos. Isso acontece pois, no primeiro caso, o maior valor singular da matriz é aproximadamente 0,99, o menor valor singular 0,03 e o número de condição da matriz é de aproximadamente 31,66, ou seja, é uma matriz relativamente bem-condicionada e, aliado ao fato de termos dados com quase nada de erros/ruídos, então da equação (3.7) percebemos que a contribuição do vetor de ruídos e é bastante insignificante e o melhor resultado é obtido com $k = 324$. Em contrapartida, no segundo caso, os extremos dos valores singulares são aproximadamente 0,97 e $4{,}49 \times 10^{-18}$ (menor que a precisão da máquina) e o número de condição da matriz fica em torno de $2{,}17 \times 10^{17}$, revelando que a matriz é mal-condicionada e tornando a solução truncada mais interessante. Aqui, pelo fato dos menores valores singulares serem extremamente próximos de zero, isto faz com

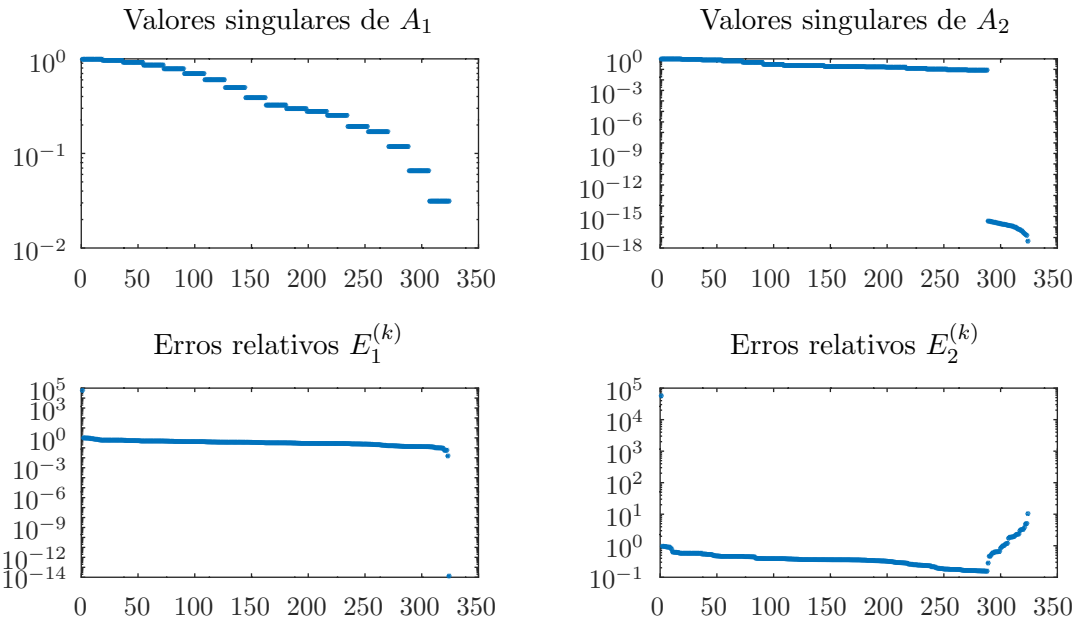
que as parcelas do ruído e associadas aos menores valores singulares tenham forte contribuição na solução de mínimos quadrados. A ideia do método TSVD é tentar construir uma solução que "filtre" o ruído nos dados de modo a não incluir as contribuições das componentes de e associadas aos menores valores singulares, mesmo que isso implique em não capturar parte da informação de b_{exato}.

Na Figura 3.2 apresentamos os valores singulares das matrizes A_1 e A_2 responsáveis pelas distorções 1 e 2, respectivamente, e os erros relativos entre a k-ésima solução obtida com o método TSVD e a imagem original, ou seja,

$$E_1^{(k)} = \frac{\|x_1^{(k)} - x_{\text{exato}}\|}{\|x_{\text{exato}}\|} \quad \text{e} \quad E_2^{(k)} = \frac{\|x_2^{(k)} - x_{\text{exato}}\|}{\|x_{\text{exato}}\|},$$

em que $E_1^{(k)}$ e $E_2^{(k)}$ denotam os k-ésimos erros relativos dos problemas 1 e 2, respectivamente, e $x_1^{(k)}$ e $x_2^{(k)}$ representam as k-ésimas soluções em cada um dos problemas.

Figura 3.2: Paralelo entre valores singulares de A_1 e A_2 e erros relativos E_1 e E_2



Pela distribuição dos valores singulares, fica ainda mais evidente o comportamento do método. Enquanto com A_1 os valores singulares ficam "controlados", o erro relativo se mantém de acordo, porém, no segundo caso, de $\sigma_{289} = 3{,}6892 \times 10^{-16}$ em diante, a contribuição do ruído e é dominante, provocando uma perturbação significativa nas soluções.

Apesar do bom desempenho desse método, calcular a SVD se torna uma tarefa com-

putacionalmente cara a medida que n aumenta, conforme mostramos na Tabela 3.1, em que apresentamos um paralelo entre a dimensão da matriz A e o tempo para calcular a sua SVD via Octave (EATON et al., 2020) instalado em uma máquina que possui processador de 2,5 GHz Dual-Core Intel Core i5 e memória RAM de 10 GB.

Tabela 3.1: Tempo de processamento da SVD

Dimensão	Tempo (s)
324×324	1,35
576×576	1,78
1024×1024	14,78
1600×1600	42,30
2500×2500	148,72

Por conta disso, para resolver esse tipo de problema muitas vezes é necessário recorrer a outros métodos que são computacionalmente mais baratos.

3.3 O método LSQR

Sabemos que encontrar uma solução de mínimos é equivalente a encontrar a solução das equações normais 3.5.

O método LSQR contrói soluções iteradas em um tipo particular de subespaços associados à matriz A^TA e ao vetor A^Tb, mais especificamente em subespaços de Krylov cuja definição apresentamos a seguir.

Definição 3.3.1. *Seja A uma matriz quadrada de ordem n e um vetor $b \in \mathbb{R}^n$, o subespaço de Krylov de ordem k associado ao par (A, b) é $\mathcal{K}_k(A, b) = \text{span}\{b, Ab, A^2b, \cdots, A^{k-1}b\}$.*

Conforme mencionado anteriormente, o método LSQR constrói soluções nos subespaços de Krylov $\mathcal{K}_k(A^TA, A^Tb)$, ou seja,

$$x_k = \underset{x \in \mathcal{K}_k}{\text{argmin}} \|b - Ax\|_2$$

e, por construção, temos $K_k \subset K_{k+1}$.

Um dos pontos centrais do algoritmo LSQR é a bidiagonalização de Golub e Kahan (1965) que tem o propósito de reduzir uma matriz A a uma estrutura bidiagonal superior.

Colocamos o teorema aqui por conveniência.

Teorema 3.3.2. *Seja A uma matriz $m \times n$ com elementos complexos. Então A pode ser escrita como*

$$A = PJQ^*,$$

em que P e Q são matrizes unitárias e J é bidiagonal superior.

Demonstração. A demonstração do teorema pode ser encontrada em Golub e Kahan (1965). □

Deste resultado temos duas formas de bidiagonalização, uma superior e uma inferior. Como queremos apenas a redução a forma bidiagonal inferior, não abordaremos a superior.

Perceba que definindo u_1 adequadamente, recursivamente é possível obter os vetores v_1, u_2, v_2, $\cdots$, v_k e u_{k+1} e os respectivos $\alpha_1, \cdots, \alpha_k, \beta_2, \cdots, \beta_{k+1}$ da seguinte maneira

$$q_j = A^T u_j - \beta_j v_{j-1}, \quad \alpha_j = \|q_j\|_2 \quad \text{e} \quad v_j = \alpha_j^{-1} q_j, \tag{3.13}$$

$$p_j = A v_j - \alpha_j u_j, \quad \beta_{j+1} = \|p_j\|_2 \quad \text{e} \quad u_{j+1} = \beta_{j+1}^{-1} p_j. \tag{3.14}$$

Por ser um algoritmo iterativo, para $k = 1, \cdots, n$ buscamos encontrar na k-ésima iteração as matrizes

$$U_k = [\, u_1 \; u_2 \; \cdots \; u_{k+1} \,] \in \mathbb{R}^{m \times (k+1)},$$

$$V_k = [\, v_1 \; v_2 \; \cdots \; v_k \,] \in \mathbb{R}^{n \times k} \quad \text{e}$$

$$B_k = \begin{bmatrix} \alpha_1 & & & \\ \beta_2 & \alpha_2 & & \\ & \beta_3 & \ddots & \\ & & \ddots & \alpha_k \\ & & & \beta_{k+1} \end{bmatrix} \in \mathbb{R}^{(k+1) \times k},$$

então, pelo forma como a decomposição é construida, para qualquer $j \in \{1, 2, \cdots, n\}$, vale que

$$A v_j = \alpha_j u_j + \beta_{j+1} u_{j+1}, \tag{3.15}$$

disso podemos escrever que

$$AV_k = U_k B_k \tag{3.16}$$

e impondo que $\beta_1 v_0 \equiv 0$ e $\alpha_{k+1} v_{k+1} \equiv 0$, para qualquer $j \in \{1, 2, \cdots, n+1\}$ vale que

$$A^T u_j = \beta_j v_{j-1} + \alpha_j v_j, \tag{3.17}$$

então podemos escrever

$$A^T U_{k+1} = V_k B_k^T + \alpha_{k+1} v_{k+1} e_{k+1}^T \tag{3.18}$$

em que e_{k+1} é o vetor $e_{k+1} = [\underbrace{0 \ \cdots \ 0}_{k \text{ vezes}} \ 1]^T$.

Por conveniência para o objetivo do método, fixamos

$$\beta_1 u_1 = b \tag{3.19}$$

e como $\|u_1\|_2 = 1$, pois é ortonormal, consideramos $u_1 = \frac{b}{\|b\|_2}$ e $\beta_1 = \|b\|_2$.

Então pela forma que u_j e v_j são construídos, temos que $u_j \in \hat{\mathcal{K}}_j(AA^T, b)$ e $v_j \in \check{\mathcal{K}}_j(A^T A, A^T b)$ sendo

$$\hat{\mathcal{K}}_j(AA^T, b) = \text{span}\{b, (AA^T)b, \cdots, (AA^T)^{j-1} b\} \tag{3.20}$$

$$\check{\mathcal{K}}_j(A^T A, A^T b) = \text{span}\{A^T b, (A^T A)A^T b, \cdots, (A^T A)^{j-1} A^T b\} \tag{3.21}$$

os subespaços de Krylov associados às matrizes AA^T e $A^T A$, respectivamente.

Com esses insumos, o método LSQR constrói uma sequência de aproximações $x_k \in \check{\mathcal{K}}_k(A^T A, A^T b)$ para o problema de mínimos quadrados, que nesse caso fica restrito a estes subespaços, ou seja,

$$x_k = \underset{x \in \check{\mathcal{K}}_k}{\text{argmin}} \|b - Ax\|_2.$$

Como $\check{\mathcal{K}}_k(A^T A, A^T b) = \text{span}(V_k)$, então para algum $y_k \in \mathbb{R}^k$ obtemos

$$x_k = V_k y_k. \tag{3.22}$$

Logo, o resíduo r_k associado a esta solução é

$$r_k = b - Ax_k \overset{(3.22)}{=} b - AV_k y_k \overset{(3.16)}{=} b - U_k B_k y_k \overset{(3.19)}{=} \beta_1 u_1 - U_k B_k y_k = \beta_1 U_k e_1 - U_k B_k y_k,$$

em que e_1 é o vetor canônico $[1, 0, 0, \cdots, 0]^T \in \mathbb{R}^k$, e a norma do resíduo é

$$\|r_k\|_2 = \|\beta_1 U_k e_1 - U_k B_k y_k\|_2 = \|\beta_1 e_1 - B_k y_k\|_2, \tag{3.23}$$

pois as colunas da matriz U_k são ortonormais.

Mas das equações (3.22) e (3.23), obtemos o problema de encontrar um $y_k \in \mathbb{R}^k$ tal que

$$y_k = \underset{y \in \mathbb{R}^k}{\operatorname{argmin}} \|\beta_1 e_1 - B_k y\|_2. \tag{3.24}$$

Finalmente usamos a decomposição QR da matriz bidiagonal inferior B_k para resolver esse último problema (e consequentemente o problema de encontrar x_k). Agora o objetivo é encontrar $Q_k \in \mathbb{R}^{(k+1)\times(k+1)}$ e uma matriz bidiagonal superior $R_k \in \mathbb{R}^{k\times k}$ tal que

$$Q_k B_k = \overline{R}_k = \begin{pmatrix} R_k \\ \\ 0 \end{pmatrix} \in \mathbb{R}^{(k+1)\times k}. \tag{3.25}$$

Consequentemente

$$Q_k(\beta_1 e_1) = \begin{pmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_{k-1} \\ \overline{\phi}_{k+1} \end{pmatrix} = \begin{pmatrix} f_k \\ \\ \overline{\phi}_{k+1} \end{pmatrix} \in \mathbb{R}^{k+1}.$$

Tendo em vista que o produto $Q_k B_k$ visa eliminar todos os elementos β_i da matriz B_k, obtemos

$$R_k = \begin{pmatrix} \rho_1 & \theta_1 & & & \\ & \rho_2 & \theta_2 & & \\ & & \ddots & \ddots & \\ & & & \rho_{k-1} & \theta_{k-1} \\ & & & & \rho_k \end{pmatrix},$$

e a matriz Q_k pode ser obtida do produto de rotações de Givens, $Q_k = G_{k,k+1} G_{k-1,k} \cdots G_{1,2}$,

em que

$$G_{1,2} = \begin{pmatrix} c_1 & s_1 & & & \\ -s_1 & c_1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{pmatrix}, \cdots, G_{j,j+1} = \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & c_j & s_j & & \\ & & -s_j & c_j & & \\ & & & & \ddots & \\ & & & & & 1 \end{pmatrix},$$

com c_j e s_j dados por

$$c_j = \frac{\alpha_j}{\sqrt{\alpha_j^2 + \beta_{j+1}^2}} \quad \text{e} \quad s_j = \frac{\beta_{j+1}}{\sqrt{\alpha_j^2 + \beta_{j+1}^2}}.$$

Deste modo, obtemos o vetor solução y_k da seguinte forma

$$R_k y_k = f_k \Rightarrow y_k = R_k^{-1} f_k \tag{3.26}$$

e, considerando que $Q_k^T Q_k = I$ e $B_k = Q_k^T \bar{R}_k$, o resíduo obtido é

$$\begin{aligned} r_k &= \beta_1 e_1 - B_k y_k \\ &= Q_k^T Q_k (\beta_1 e_1) - Q_k^T \bar{R}_k R_k^{-1} f_k \\ &= Q_k^T \begin{pmatrix} f_k \\ \bar{\phi}_{k+1} \end{pmatrix} - Q_k^T \begin{pmatrix} I_k \\ 0 \end{pmatrix} f_k \\ &= Q_k^T \begin{pmatrix} 0 \\ \bar{\phi}_{k+1} \end{pmatrix} \\ &\Downarrow \\ \|r_k\|_2 &= |\bar{\phi}_{k+1}|. \end{aligned} \tag{3.27}$$

Um fato importante a ser ressaltado é que a cada iteração é possível "reaproveitar" a fatoração QR anterior, isso porque da iteração $k-1$ para a k, a matriz B_k aumenta em uma linha e uma coluna, que exige mais uma rotação para zerar o elemento da posição $(k+1, k)$, mas todo o resto da decomposição se mantém. Esse último β_{k+1} que precisa

ser zerado sofre alteração da última rotação da decomposição anterior e depois é zerado com a aplicação de outra rotação $G_{k,k+1}$ resultando na última coluna de R_k. Enquanto para $Q_k(\beta_1 e_1)$ basta multiplicarmos $G_{k,k+1}$ e $\begin{bmatrix} f_k^T & \overline{\phi}_k & 0 \end{bmatrix}^T$ que provoca mudança apenas nos elementos $\overline{\phi}_k$ e 0. Como $\overline{\phi}_{k+1} = -s_k\,\overline{\phi}_k$, então a norma do resíduo é

$$\|r_k\|_2 \overset{(3.27)}{=} |\overline{\phi}_{k+1}| = |s_k\overline{\phi}_k| = \cdots = |s_k s_{k-1}\cdots s_1\beta_1|. \tag{3.28}$$

Além disso, perceba que $|s_k| = |\operatorname{sen}\theta_k| \leq 1$, então $\|r_k\|_2 = |s_k\overline{\phi}_k| \leq |\overline{\phi}_k| = |r_{k-1}|$, ou seja, a normal do resíduo é não-crescente, corroborando o teorema 3.1.1.

Finalmente, substituindo (3.26) em (3.22)

$$x_k = V_k y_k = V_k R_k^{-1} f_k.$$

Porém dessa forma é necessário armazenar os vetores $v_1, \cdots, v_k$ para calcular x_k e dependendo das dimensões do problema, isso pode ser computacionalmente muito "caro". Entretanto, Paige e Saunders (1982) apresentam uma relação de recorrência para calcular x_k a partir de x_{k-1}, usando $Z_k = V_k R_k^{-1}$. Nesse caso, definindo $z_0 = x_0 = 0$, iterativamente obtemos

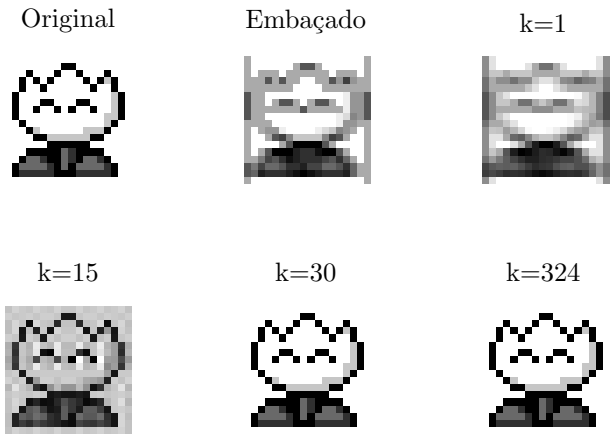
$$z_k = \frac{1}{\rho_k}(v_k - \theta_k z_{k-1}) \quad \text{e} \quad x_k = x_{k-1} + \phi_k z_k.$$

Cabe salientar que toda essa teoria quando aplicada na prática, fica sujeita a precisão da máquina e, consequentemente, pode gerar resultados não esperados e perda de ortogonalidade dos vetores u_i e v_i, principalmente se k for grande. Neste caso podemos reortogonalizar os vetores u_i e v_i pelo processo de Gram-Schmidt ou Gram-Schmidt modificado.

Aplicando o método na mesma figura que usamos de exemplo na seção anterior, obtemos um resultado satisfatório para um k pequeno, se comparado ao método TSVD - conforme mostra a Figura 3.3.

Neste caso, à medida que a dimensão do subespaço aumenta, a solução torna-se melhor, entretanto a partir de $k = 30$ a mudança na solução é imperceptível, portanto, do ponto de vista operacional, não precisamos da solução "exata" do sistema linear e o algoritmo pode ser interrompido, isto é, $x_{30} \approx x_{\text{LS}}$. Salientamos que critérios de parada não serão discutidos neste trabalho, mas para mais informações sugerimos Borges, Bazán e Cunha (2015).

Figura 3.3: Ilustração de algumas soluções do método LSQR



Para ilustrar como o método LSQR é menos custoso que o TSVD, apresentamos a seguir a tabela 3.2 que indica o tempo decorrido para resolver a quantidade indicada de soluções/iterações para problemas de restauração com diferentes quantidades de variáveis. Essa tabela pode ser comparada com a Tabela 3.1 em que apresentamos o tempo de processamento da SVD.

Tabela 3.2: Tempo de processamento da LSQR

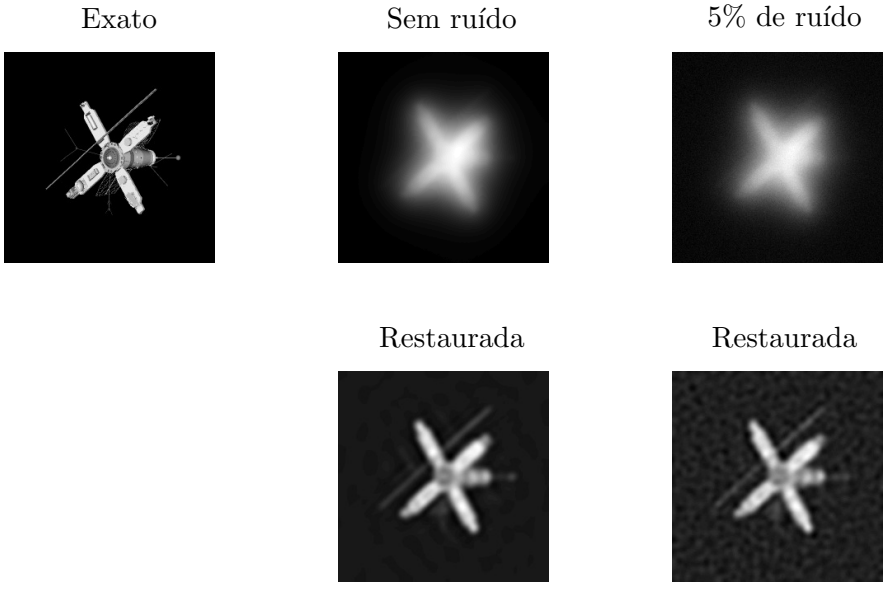
Dimensão	Tempo (s)	Iterações
324×324	0,07	324
1600×1600	0,43	1600
10.000×10.000	3,86	3000
65.536×65.536	21,26	3000
262.144×262.144	104,43	3000

Conforme o exposto, podemos dizer que para imagens pequenas todas as iterações possíveis são obtidas quase que instantâneamente, mas à medida que o problema aumenta deixa de ser interessante esgotar as iterações e mesmo assim obtivemos resultados excelentes. Por exemplo, no problema 262.144×262.144 do teste, antes na iteração 1.000 o algoritmo encontrou uma solução com pequeno erro relativo de $0,0040987$, não sendo necessário realizar as outras 2.000 iterações.

3.4 Exemplos de aplicações em problemas com ruído

A fim de entender qual é o efeito do ruído dos dados na prática, aplicamos o algoritmo LSQR em problemas de 65.536 variáveis gerados a partir da imagem de um satélite de 256×256 pixels. Com o problema exato (exceto por erros de arredondamento) proveniente do pacote *RestoreTools* (NAGY et al., 2007) que fornece uma PSF semelhante à da turbulência atmosférica apresenta na seção 2.2, foram criados outros três problemas com níveis de ruído relativo nos dados de 0,1%, 1% e 5%. Na Figura 3.4 apresentamos a imagem exata, como também o problema sem ruído e com ruído de 5%, os casos intermediários são semelhantes. Por questões de processamento, não executamos os algoritmos até o último passo possível. Foram realizadas apenas 500 iterações em cada caso e conseguimos perceber características interessantes. A média de tempo para obter as 500 soluções iteradas para cada um dos problemas foi de aproximadamente 39s.

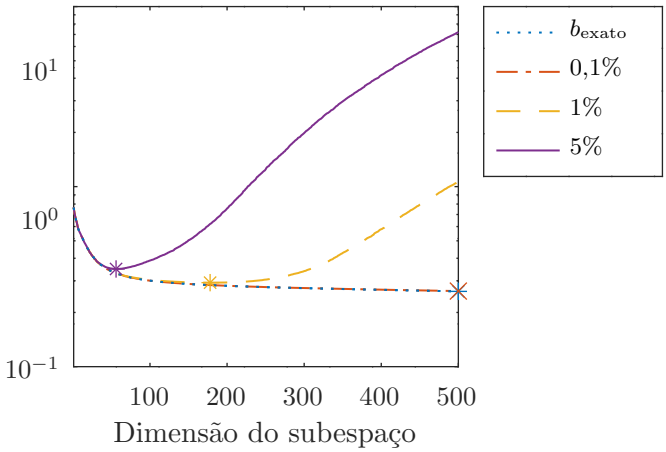
Figura 3.4: À esquerda está a imagem exata do satélite, no meio a imagem embaçada e a respectiva melhor solução e à direita a imagem embaçada com 5% de ruído e a respectiva melhor solução



Na Figura 3.5 apresentamos um gráfico com os erros relativos das soluções de cada subespaço para cada um dos problemas com diferentes níveis de ruído.

Percebemos que as linhas que representam os erros relativos do problema com dados exatos e do problema com ruído de 0,1% ficam praticamente sobrepostas, o que nos leva a concluir que pelo menos para as primeiras iterações um ruído pequeno nos dados

Figura 3.5: Gráfico de comparação dos erros relativos das soluções LSQR do problema do Satélite exato e com diferentes níveis de ruído (eixo y em escala logarítmica)



não provoca muita interferência na solução. Em contrapartida, à medida que o nível de ruído aumenta, mais rapidamente atingimos o limite de iterações em que a solução iterada melhora (em relação a solução exata), entretanto após esse limite a contribuição associada ao ruído pode interferir drasticamente nas soluções, pois a cada nova iteração temos x_k cada vez mais próximo de x_{LS} e, como vimos, x_{LS} é dominada pelo ruído e.

Para deixar mais explícito, na Tabela (3.3) apresentamos para cada um dos problemas do satélite o menor erro relativo (valor aproximado com 10 casas decimais), a dimensão do subespaço da melhor solução e o tempo de processamento.

Tabela 3.3: Menor erro relativo, dimensão do subespaço e tempo de processamento de cada um dos problemas do satélite

Ruído	Erro mínimo	k	Tempo (s)
0	0,2618549133	500	38
0,1%	0,2629424992	500	36
1%	0,2928567106	178	41
5%	0,3487907997	56	40

Analogamente, fizemos esse experimento para uma segunda imagem, esta com 512 × 512 pixels. Novamente criamos 4 problemas, sendo que a matriz de embaçamento A nesse caso foi criada a partir de uma PSF fora de foco, a apresentada na seção 2.2, e executamos o algoritmo até a iteração 500.

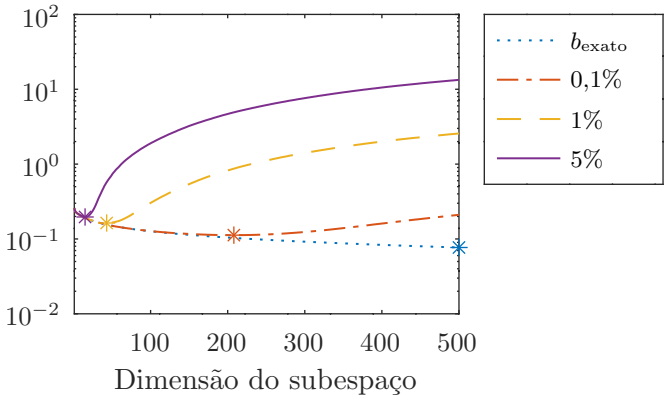
Como podemos ver na Figura 3.6, por ser uma imagem com mais detalhes, a solução

Figura 3.6: À esquerda está a imagem exata do Pirata, no meio a imagem embaçada e a respectiva melhor solução e à direita a imagem embaçada com 5% de ruído e a respectiva melhor solução

do problema com 5% de ruído acaba sendo muito mais "poluída" e desagradável.

Nesse caso, o gráfico da Figura 3.7 deixa mais nítido como o ruído dos dados afeta as soluções rapidamente.

Figura 3.7: Gráfico de comparação dos erros relativos das soluções LSQR do problema do Pirata exato e com diferentes níveis de ruído (eixo y em escala logarítmica)



No problema com 5% de ruído, com apenas 15 iterações obtemos o menor erro relativo, a partir daí a contribuição do ruído e domina a solução. Já no problema com apenas 0,1% a melhor solução é obtida no subespaço de dimensão 208, diferente do problema do satélite em que o erro decresceu até a última iteração executada, mas o erro relativo se mantém

controlado até a iteração 500.

Condensamos as principais informações do experimento na Tabela 3.4 que consta os valores dos menores erros relativos, a dimensão do subespaço em que se obteve a melhor solução e tempo de processamento das 500 iterações para cada um dos problemas do pirata.

Tabela 3.4: Menor erro relativo, dimensão do subespaço e tempo de processamento de cada um dos problemas do pirata

Ruído	Erro mínimo	k	Tempo (s)
0	0,0769178348	500	161
0,1%	0,1124281242	208	173
1%	0,1634833324	43	177
5%	0,1966136110	15	169

Sendo assim, concluímos que o método LSQR é um método adequado para trabalhar com problemas de grande porte por conta de sua eficiência em encontrar soluções significativamente boas com poucas iterações, mesmo quando temos ruído nos dados.

Nesse capítulo também exploramos o método TSVD, mas por conta da dependência da Decomposição em Valores Singulares, esse método é muito mais custoso e exige que sejam executadas quase todas as iterações para obter uma solução adequada.

Então, considerando a hipótese de abordar o problema de restauração de imagens no ensino superior, em que desconhecemos o poder computacional das máquinas, acreditamos ser mais viável a aplicação do método LSQR, conforme propomos no capítulo a seguir.

Capítulo 4

Uma proposta de aplicação no Ensino Superior

Neste capítulo apresentamos uma proposta de sequência de aulas para uma primeira disciplina de Álgebra Linear em que os assuntos de restauração de imagens podem ser tratados. Nosso público alvo são os alunos dos cursos de graduação em matemática e física, em que essa disciplina é ofertada, conforme consta nas respectivas Diretrizes Curriculares Nacionais desses cursos, e se estende também aos cursos de engenharia que têm essa matéria no currículo.

Tendo em vista a gama de conceitos que podem ser abordados no estudo de restauração de imagens, que abre margem para falar sobre assuntos como sistemas lineares, solução de mínimos quadrados, espaço e subespaço vetorial, matrizes, autovalores e autovetores, entre outros, sugerimos uma sequência de oito aulas para explorar alguns desses assuntos. Para essa proposta, consideramos os currículos dos cursos da Universidade Federal de Santa Catarina – UFSC, em que a disciplina de Álgebra Linear é ofertada no segundo ou terceiro semestre.

Nosso foco nessa sequência é tratar desde Sistemas Lineares até Espaço Vetorial e Base, passar rapidamente pelo conceito do problema de Mínimos Quadrados e finalizar com o problema prático de restauração de imagem. E para deixar os assuntos mais encadeados e fluidos, aconselhamos que o professor prepare os alunos desde o início, comentando previamente sobre a atividade final à medida que o conteúdo se desenvolve.

Inicialmente recomendamos tratar dos assuntos básicos de Sistemas Lineares, permeando por definições, propriedades, exemplos, sistema na forma matricial e o método de

solução por Eliminação Gaussiana (Método de Escalonamento). Nessa etapa, ao falar de classificação de Sistemas Lineares, é conveniente que o professor deixe explícito que quando o sistema é impossível, podemos considerar, por exemplo, a solução de mínimos quadrados que será apresentada mais adiante. Estimamos duas aulas para isso, preferencialmente em dias separados.

Na sequência, sugerimos abordar os conteúdos de Espaço Vetorial e Subespaço Vetorial, novamente trazendo definições, propriedades e exemplos. Inclusive, a ideia de vetor deve ser bem explorada, visto que os alunos já conhecem esse termo das aulas do ensino médio (COIMBRA, 2008), mas na Álgebra Linear esse termo é usado de forma mais ampla. Então é importante frizar que, a depender do Espaço, um vetor pode ser uma matriz (como uma imagem) ou polinômio, por exemplo. Sugerimos duas aulas nessa etapa.

Feito isso, em uma aula, cabe entrar nos assuntos de combinação linear, subespaço gerado, dependência e independência linear e base. Um questão que muita das vezes gera confusão e que aqui pode ser melhor explorada é a dimensão do subespaço, que muitas vezes os alunos associam, por exemplo, a quantidade de entradas do vetor, o que não é correto.

Posteriormente, podemos abordar os conceitos de imagem que apresentamos no capítulo 2, a começar pela representação digital da imagem. Lincando com a aula anterior, nesse contexto é viável apresentar aos alunos a ideia de que, por exemplo, uma imagem em tons de cinza de 4 pixels (2×2) pode ser representada como

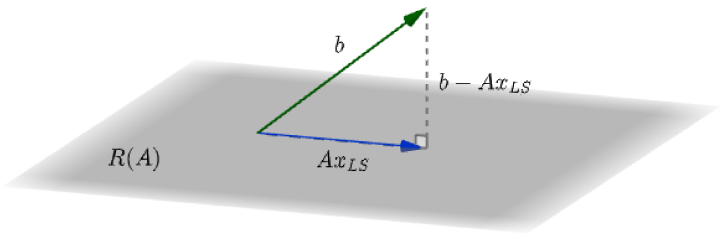
$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} a & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & b \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ c & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & d \end{bmatrix},$$

assim, considerando o caso em que todos os pixels são diferentes de zero, cada uma dessas imagens que representa um pixel pode ser interpretada como um vetor de uma base do $\mathbb{R}^{2\times 2}$ e a imagem completa é obtida como uma combinação linear destas imagens que representam um único pixel. E podemos generalizar essa ideia para dimensões maiores. Na sequência podemos dar uma ideia de modelos de embaçamento e por fim apresentar o problema de restauração, que será explorado posteriormente. Uma aula é suficiente.

Em seguida, podemos usar uma aula para expor o assunto de mínimos quadrados e projeção em subespaço. Considerando que a essa altura a turma ainda não tem muitos

insumos matemáticos, como norma e ortogonalidade, então sugerimos que a abordagem seja mais visual, isto é, apresentando a noção geométrica. Nesse caso, como ainda não podemos falar em "norma do vetor", uma opção é usar informalmente a expressão "tamanho do vetor" ou "módulo do vetor" para dizer que o objetivo por trás do método de mínimos quadrados é "encontrar o vetor x, tal que o tamanho do vetor $b - Ax$ é o menor possível". Ilustramos na Figura 4.1 a ideia geométrica da solução de mínimos quadrados, em que o

Figura 4.1: Interpretação geométrica da solução de mínimos quadrados



vetor b no espaço $\mathbb{R}^3$ é representado em verde, o subespaço de 2 dimensões (plano cinza) representa o espaço gerado pelas colunas da matriz A, a projeção ortogonal do vetor b nesse subespaço é o vetor Ax_{LS} (vetor azul) e o resíduo é o vetor $b - Ax_{\text{LS}}$, indicado pelo tracejado, que é o vetor cuja norma (tamanho, módulo) queremos minimizar.

Finalmente, temos uma aula cujo objeto é estudar como resolver os problemas de restauração de imagem, apresentado aulas antes. Aqui cabe frisar que para problemas de dimensão grande como esses, é inviável resolver o sistema via métodos diretos, como a Eliminação Gaussiana, consequentemente precisamos usar outros tipos de método, nesse caso vamos usar o método de projeção LSQR. Então apresentamos o Subespaço de Krylov e explicamos que para o problema $Ax = b$ o método LSQR constrói uma sequência de soluções $x_k \in \check{\mathcal{K}}_k(A^TA, A^Tb)$, isto é, uma sequência de soluções sobre os Subespaços de Krylov associados a matriz A^TA e ao vetor A^Tb, portanto o problema fica restrito a estes subespaços, ou seja,

$$x_k = \underset{x \in \check{\mathcal{K}}_k}{\operatorname{argmin}} \|b - Ax\|_2.$$

Teoricamente, essa sequência converge para a solução de mínimos quadrados, pois a medida que a dimensão do subespaço da solução aumenta Ax_k fica mais "próximo" de b. Mas vale ressaltar que na prática, temos a influência do ruído nos dados, que são erros numéricos de medida, arredondamento e afins, então nem sempre a solução no maior subespaço é a melhor. Entretanto, o método LSQR tem a vantagem de sofrer pouca influência do

ruído no início, por isso é importante interromper o processo, abrindo mão de parte da informação que seria restaurada, mas garantindo uma solução com menos ruído.

Nessa aula é importante ter interação com algum software em que possa mostrar exemplos práticos de restauração, ponderando sobre o comportamento das soluções nos subespaços de diferentes dimensões. Se for viável, nesta oportunidade o professor pode deslocar a turma para o laboratório de informática para que os alunos possam experienciar o processo e fazer seus próprios testes, deixamos nos anexos algumas rotinas para facilitar o processo. Para finalizar esta etapa sugerimos alguns questionamentos que apresentamos na seção a seguir.

Em resumo, a proposta de sequência de aulas é apresentada na Tabela 4.1.

Tabela 4.1: Proposta de sequência de aulas

Aulas	Conteúdo
1	Sistemas Lineares: definição e propriedades
1	Sistemas Lineares: Eliminação Gaussiana
2	Espaço Vetorial e Subespaço Vetorial
1	Combinação Linear, Subespaço Gerado, Dependência e Independência Linear e Base
1	Imagens e o Problema de Embaçamento
1	Mínimos Quadrados e Projeção em Subespaço
1	Subespaço de Krylov e Método LSQR

4.1 Exercícios propostos

Para fixação e melhor compreensão dos assuntos tratados, sugerimos alguns exercícios para serem trabalhados em aula.

1. Escolha uma imagem colorida qualquer e exiba separadamente as três camadas de cor RGB com auxílio da função *imagesc*. Considerando sua representação matricial, podemos dizer que essa imagem pertence a que espaço vetorial? Dica: Para que uma só cor apareça é necessário zerar as entradas das outras duas camadas de cor.

   Espera-se um resultado semelhante ao da Figura 2.5, nesse caso e a dimensão seria $\mathbb{R}^{960\times640\times3}$*, sendo uma matriz tridimensional.*

2. Assinale as afirmações a seguir com “V” para verdadeiro e “F” para falso.

   ( ) Quanto mais pixels tem uma imagem, mais nítida ela é.

   F - imagens com mais pixels são potencialmente melhores, entretanto fenômenos de embaçamento diminuem a nitidez de imagens de qualquer dimensão.

   ( ) Uma “imagem exata” não tem nenhuma perca de informação.

   F - até no caso da imagem mais “perfeita”, há perca de informações na discretização da imagem digital, pois não dispomos de precisão infinita.

   ( ) É possível modelar fenômenos de embaçamento por expressões matemáticas.

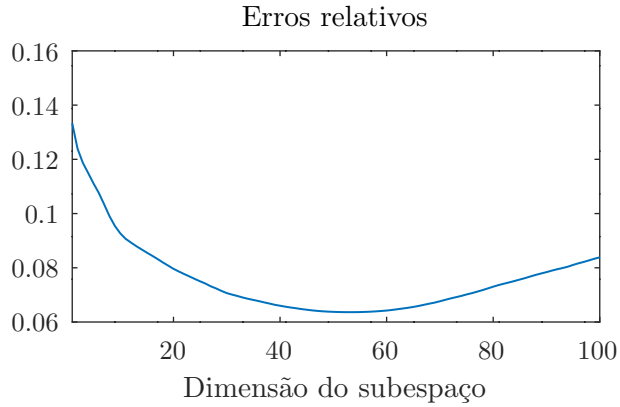
   V - falamos sobre isso na seção 2.2.

3. Informalmente, o que significa “embaçamento espacialmente invariável”?

   Significa que todos os pixels da imagem passam pelo mesmo processo de embaçamento, em outras palavras, usamos a mesma PSF para embaçar toda a figura.

4. Para o problema do Relógio (ver orientações do Apêndice A), construa o gráfico dos erros relativos das soluções do método LSQR para $k = 1$ até 100. Sugestão: use a função *lsqr_b* do Anexo III para obter as soluções e a função *plot* para criar o gráfico.

Figura 4.2: Solução do exercício 4: Gráfico dos erros relativos



5. No exercício anterior, para qual subespaço temos o melhor resultado? Mostre a figura embaçada e a melhor figura restaurada.

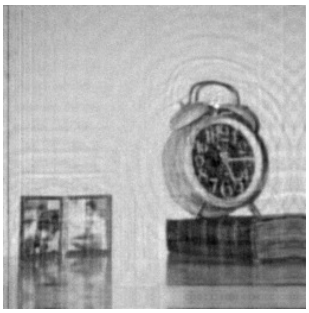
   A melhor solução é obtida no subespaço de dimensão 53, vide Figura 4.3.

6. Construa três problemas a partir de diferentes PSFs e com 4% de ruído. Exiba as imagens embaçadas.

Figura 4.3: Solução do exercício 5: Imagem embaçada ao lado da imagem restaurada

Imagem embaçada

Melhor solução, $k = 53$



Resposta individual, nela os embaçamentos podem ser criados com auxiliou das funções em anexo oblur.m ou mblur.m ou criando PSF próprias (no Apêndice A tem mais orientações sobre) ou ainda usando a função psfGauss.m da pasta TestDada do pacote RestoreTools.

7. Crie uma tabela que indique o erro mínimo e o k da melhor solução do método LSQR para cada um dos problemas do exercício anterior. O que você pode concluir disso?

   Resposta individual.

8. Seja $A = [a_1 \cdots a_n]$ uma matriz $m \times n$ em que a_i, com $i = 1, \cdots, n$, denota a i-ésima coluna de A. O subespaço gerado pelas colunas da matriz é $R(A) = \text{span}\{a_1, \cdots, a_n\}$. Podemos dizer que $R(A) = \mathbb{R}^m$? Por quê?

   Não, porque não há garantia de que entre os vetores temos um conjunto LI de m elementos.

9. Seja A uma matriz $m \times n$ e $b \in \mathbb{R}^m$, podemos afirmar que $b \in R(A)$?

   Não.

10. Dê um exemplo de uma matriz $A \in \mathbb{R}^{3\times3}$ e vetores $b_1, b_2 \in \mathbb{R}^3$, tais que $b_1 \in R(A)$ e $b_2 \notin R(A)$. Geometricamente, como podemos interpretar isto?

    Esperamos que os alunos encontrem uma matriz A com pelo menos uma coluna não nula e as demais múltiplas dessa ou A com duas colunas LI e a terceira uma combinação linear delas, b_1 pode ser uma das colunas da matriz e b_2 pode ser qualquer vetor fora ao espaço gerado de $R(A)$. Apesar de A possuir 3 colunas, elas

são LD e, consequentemente, $R(A) \neq \mathbb{R}^3$, nesse caso $R(A)$ pode ser interpretado geometricamente como um plano (como na Figura 4.1) ou uma reta no espaço $\mathbb{R}^3$.

11. Seja uma matriz $A \in \mathbb{R}^{m \times n}$ e $b \in \mathbb{R}^m$.

    a) Mostre que o conjunto

    $$N(A) = \{x \in \mathbb{R}^n / Ax = 0\}$$

    é um subespaço vetorial.

    Basta verificar as propriedades de subespaço vetorial.

    b) Mostre que se x_0 é uma solução do sistema linear $Ax = b$, então qualquer vetor da forma

    $$x = x_0 + x^*, \text{ para todo } x^* \in N(A)$$

    também é solução.

    A ideia é mostrar que $A(x_0+x^*) = b$. De fato, por ser linear $A(x_0+x^*) = Ax_0+Ax^*$, como x_0 é solução e $x^* \in N(A)$, então $Ax_0 + Ax^* = b + 0 = b$, logo $x = x_0 + x^*$ é solução.

Capítulo 5

Considerações Finais

Existem diversas áreas em que os problemas inversos estão presentes e se faz necessário o uso das mais diversas ferramentas provenientes da Álgebra Linear para conseguirmos obter soluções computacionalmente aplicáveis. Neste trabalho buscamos unir o útil ao agradável, explorando problemas práticos de restauração de imagens nas aulas de graduação para ilustrar alguns dos principais conceitos de Álgebra Linear, em especial o conceito de Subespaços, incentivando os alunos a expandirem seu entendimento a respeito. Estimulamos também a interação com software livre Octave para aplicação dos métodos, que por ventura pode despertar o interesse do aluno a explorar a ferramenta para outros fins.

Para fundamentar a proposta, dedicamos um capítulo para discorrer brevemente sobre como se dão os processos de captura de imagem e representação digital e trouxemos alguns modelos de processos físicos de embaçamento, que são parte importante da construção do problema de restauração.

Visto que nesse tipo de problema temos que resolver sistemas lineares com milhares ou milhões de variáveis e, portanto, seria inviável de resolver por métodos diretos, apresentamos dois poderosos métodos iterativos de para calcular soluções úteis.

No primeiro método usamos a decomposição em valores singulares da matriz de coeficientes A. Porém vimos que se A possui valores singulares muito pequenos, a solução torna-se instável a pequenas perturbações nos dados e por isso é mais interessante truncá-la, isto é, desprezando uma parcela dos dados associada aos menores valores singulares, chamamos esse método de TSVD. O ponto negativo desse método é a dificuldade em obter a decomposição em valores singulares da matriz, alternativamente apresentamos o método LSQR. Esse por sua vez é um método de projeção que constrói uma sequência de soluções

para problemas de mínimos quadrados restritos aos Subespaços de Krylov associados à matriz A^TA e ao vetor A^Tb. Temos uma seção destinada ao método LSQR em que entramos em detalhes na forma como este usa a bidiagonalização inferior de Golub e Kahan (1965) na matriz de coeficientes A para obter problemas de mínimos quadrados sobre os Subespaços de Krylov, que são resolvidos usando a fatoração QR e rotações de Givens. Para fechar esse capítulo, exibimos os resultados de alguns experimentos numéricos com o método LSQR que mostram a tendência de comportamento das soluções iteradas em problemas com e sem ruído e também registramos os tempos de processamento, que evidenciam que o método é computacionalmente viável para ser aplicado em sala de aula, visto que seu processamento é rápido.

Finalmente, propomos uma sequência de oito aulas com sugestões de como os problemas de restauração de imagens podem ser apresentados para turmas de graduação em sua primeira disciplina de Álgebra Linear, mesmo que nesse momento do curso os alunos ainda não tenham ferramentas suficientes para compreender os fundamentos por trás do método, julgamos a experiência apropriada, tornando a formação mais flexível abrangendo teoria e prática. E para apoiar o professor, deixamos uma lista de exercícios relacionada ao assunto com as devidas orientações de como reproduzir os problemas que envolvem programação.

Como sugestão de trabalho futuro, podemos aplicar a proposta em turmas reais e colher informações sobre os impactos dessa iniciativa sobre a disciplina. Além disso, seria interessante também avaliar a resposta dos alunos ao aplicar assuntos como esses em outros momentos do curso, por exemplo, aliando o método TSVD ao conteúdo de autovalores e autovetores.

Bibliografia

AHRENS, H. *The inverse problem of Magnetocardiography*. Dissertação (Mestrado) — Christian-Albrechts-Universität zu Kiel, 2015.

BAUMEISTER, J.; LEITÃO, A. *Topics in Inverse Problems*. [S.l.]: IMPA, 2005. ISBN 85-244-0224-5.

BORGES, L. S.; BAZÁN, F. S. V.; CUNHA, M. Automatic stopping rule for iterative methods in discrete ill-posed problems. *Computational and Applied Mathematics*, v. 34, p. 1175–1197, 2015.

BURDEN, R. L.; FAIRES, D. J.; BURDEN, A. M. *Análise numérica*. 3. ed. [S.l.]: Cengage Learning, 2016. v. 10. ISBN 10-85-221-2341-1.

COIMBRA, J. L. *Alguns Aspectos Problemáticos Relacionados ao Ensino-Aprendizagem da Álgebra Linear*. Dissertação (Mestrado) — Universidade Federal do Pará, 2008.

COLTON, D.; COYLE, J.; MONK, P. Recent developments in inverse acoustic scattering theory. *SIAM Review*, v. 42, n. 3, p. 369–414, 09 2000.

DAX, A. The convergence of linear stationary iterative processes for solving singular unstructured systems of linear equations. *SIAM Review*, v. 32, n. 4, p. 611–635, 12 1990.

EATON, J. W. et al. *GNU Octave version 6.1.0 manual: a high-level interactive language for numerical computations*. [S.l.], 2020. Disponível em: <https://www.gnu.org/software/octave/doc/v6.1.0/>. Acesso em: 24 set. 2021.

FURTADO, A. L. C. *Dificuldades na Aprendizagem de Conceitos Abstratos da Álgebra Linear*. Dissertação (Mestrado) — Universidade Federal do Rio de Janeiro, 2010.

GOLUB, G. H.; KAHAN, W. Calculating the singular values and pseudo-inverse of a matrix. *SIAM Journal on Numerical Analysis*, v. 2, n. 2, p. 205 – 224, 1965.

HADJIDIMOS, A. Successive overrelaxation (sor) and related methods. *Journal of Computational and Applied Mathematics*, v. 123, n. 2000, p. 177 – 199, 09 1999.

HANSEN, P. C.; NAGY, J. G.; O'LEARY, D. P. *Deblurring Images*. [S.l.]: Society for Industrial and Applied Mathematics, 2006.

KIRSCH, A. *An introduction to the mathematical theory of inverse problems*. 2. ed. [S.l.]: Springer-Verlag New York, 2011. (Applied Mathematical Sciences 120).

MARTINS, J. S. *Solução do Problema Inverso da Tomografia por Impedância Elétrica Utilizando o Simulated Annealing: Uma Nova Abordagem*. Tese (Doutorado) — Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, 01 2016.

MEYER, C. D. *Matrix Analysis and Applied Linear Algebra*. [S.l.]: Cambridge University Press, 2000.

NAGY, J.; PALMER, K.; PERRONE, L. Iterative methods for image deblurring: a matlab object-oriented approach. *Numerical Algorithms*, v. 36, p. 73–93, 12 2004.

NAGY, J. et al. *RestoreTools*. 1.4. ed. [S.l.], 2007. Disponível em: <http://www.mathcs.emory.edu/~nagy/RestoreTools/>. Acesso em: 11 ago. 2021.

PAIGE, C. C.; SAUNDERS, M. A. Lsqr: An algorithm for sparse linear equations and sparse least squares. *ACM Transactions on Mathematical Software*, v. 8, p. 43 – 71, March 1982.

STEINBRUCH, A.; WINTERLE, P. *Algebra Linear*. 2. ed. [S.l.]: Pearson Makron Books, 1987.

ZHANG, D.; GUO, Y. Some recent developments in the unique determinations in phaseless inverse acoustic scattering theory. *Electronic Research Archive*, v. 29, n. 2, p. 2149–2165, 06 2021.

Apêndice A

Orientações sobre os exercícios

Para criar exercícios de restauração de imagem podemos usar os scripts *mblur.m*, do Apêndice I, ou *oblur.m*, do Apêndice II, para criar matrizes A de embaçamento. Nesse caso, o primeiro cria o efeito de borrão de movimento (como da Figura 2.15) e o segundo gera um efeito fora de foco, porém o espalhamento do ponto se dá como um quadrado, ao invés de um círculo como na Figura 2.11. Outra opção é criarmos PSFs próprias e para obter a matriz A usamos a função *psfMatrix* do pacote *RestoreTools* versão 1.4 (NAGY et al., 2007), disponível para download no site <http://www.mathcs.emory.edu/~nagy/RestoreTools/>.

No exercício do Relógio, da Seção 4.1, utilizamos a figura "Clock", disponível para download em <https://sipi.usc.edu/database/database.php?volume=misc>, e aplicamos o embaçamento Fora de Foco apresentado na Seção 2.2. Para reproduzir o exercício, em uma pasta do seu computador copie os scrips *foraDeFoco.m* (Apêndice C) e *exercicioRelogio.m* (Apêndice C), dentro da mesma pasta faça o download da imagem "Clock" e do pacote *RestoreTools*, citados anteriormente. Então basta abrir o Octave na pasta com os códigos e executar o *exercicioRelogio.m* para obter b e A, entre outras variáveis. Para disponibilizar aos alunos, sugerimos enviar por e-mail um arquivo compactado da pasta com todos os códigos.

No Apêndice D sugerimos um código para resolver o exercício do Relógio, para que funcione corretamente ele deve ser salvo na pasta juntamente com o script *lsqr_b.m* do Anexo III e as variáveis A e b devem estar criadas no ambiente de trabalho do Octave, ou seja, o script depende das variáveis do *exercicioRelogio.m* e por isso deve ser executado após ele.

Apêndice B

Função de embaçamento *foraDeFoco.m*

```
function P = foraDeFoco(N, raio, k, l)
  % foraDeFoco retorna uma matriz P quadrada NxN usada para criar a PSF
  %
  % P = foraDeFoco(N, raio, k, l)
  %
  % O raio define o "tamanho" do desfoque.
  % As variáveis (k,l) são as coordenadas do centro do embaçamento

  P = zeros(N,N);
  raio2 = raio^2;
  nivelEmbacamento = 1/(pi * raio2);

  if (nargin < 3), k = N/2, l = N/2; end
  if raio > k
    iMin = 1;
  else
    iMin = k-raio2;
  endif
  if N < k+raio2
    iMax = N;
  else
    iMax = k+raio2;
  endif
  if raio > l
    jMin = 1;
  else
    jMin = l-raio2;
```

```
  endif
  if N < l+raio2
    jMax = N;
  else
    jMax = l+raio2;
  endif

  for i = iMin:iMax
    for j = jMin:jMax
      if( (i-k)^2 + (j-l)^2 <= raio^2 )
        P(i,j) = nivelEmbacamento;
      endif
    endfor
  endfor
endfunction
```

Apêndice C

Script *exercicioRelogio.m*

```
clear
pkg load image
path(path,"./RestoreTools/Classes")

# Carregando dados
x = imread("clock.tiff");
x = im2double(x(:));
N = 256;
P = foraDeFoco(N, 10);
A = psfMatrix(P);
b = A*x;

# Acrescimo de ruido aos dados
randn("seed",1);
e = randn(size(b));
e = e/norm(e)*norm(b);
b = b+e*0.0015;
```

Apêndice D

Script *solucaoExercicioRelogio.m*

```
# Solucoes
n = 100;
xLSQR = lsqr_b(A,b,n);

# Erros relativos
erroRelativo = zeros(1,n);
for i = 1:n
  erroRelativo(1,i) = norm(x-xLSQR(:,i))/norm(x);
endfor

[menorErroRelativo, kMenorErro] = min(erroRelativo);

# Grafico
figure(1)
plot(1:n,erroRelativo,"LineWidth",0.75)
axis([1 n])
xlabel("Dimensão do subespaço")
title "Erros relativos"

# Imagem embacada e imagem restaurada
figure(2)
subplot(1,2,1)
imagesc(reshape(b,N,N)),colormap(gray)
axis square
axis off
title "Imagem embaçada"
```

```
subplot(1,2,2)
imagesc(reshape(xLSQR(:,kMenorErro),N,N)),colormap(gray)
axis square
axis off
title (sprintf ("Melhor solução, k=%d", kMenorErro))
```

Anexo I

Função de embaçamento *mblur.m*

```
function T = mblur(N,bandw,xy);
%MBLUR  Create a block Toeplitz matrix that models motion deblurring.
%
% function T = mblur(N,bandw,xy);
%
% The matrix T an N*N-by-N*N symmetric, block Toeplitz matrix that
   models
% blurring of an N-by-N image by a linear motion blur along the x- or y-
   axis.
% It is stored in sparse matrix format.
%
% The bandwidth, specified by the integer bandw, detemines the "length"
% of the deblurring, in the sense that bandw is the half-bandwidth of
% the matrix.  Hence, the total "length" of the deblurring is 2*bandw-1.
% If bandw is not specified, bandw = 3 is used.
%
% The direction of the motion blurring is determined by the third
% argument being either 'x' og 'y'.  The default is 'y',

% Per Christian Hansen, IMM, 09/03/97.

% Initialization.
if (nargin < 2), bandw = 3; end
bandw = min(bandw,N);
if (nargin < 3), xy = "y"; end

% Construct T via Kronecker product.
```

```
T = spdiags(ones(N,2*bandw-1),[-bandw+1:bandw-1],N,N)/(2*bandw-1);
if (xy=="x")
  T = kron(T,speye(N));
elseif (xy=="y")
  T = kron(speye(N),T);
else
  error("Illegal third argument")
end
```

Anexo II

Função de embaçamento *oblur.m*

```
function T = oblur(N,bandw);
%OBLUR  Create a block Toeplitz matrix that models our-of-focus
   deblurring.
%
% function T = oblur(N,bandw);
%
% The matrix T an N*N-by-N*N symmetric, block Toeplitz matrix that
   models
% out-of-foucs blurring of an N-by-N image, stored in sparse matrix
   format.
%
% The bandwidth, specified by the integer bandw, detemines the "size"
% of the deblurring, in the sense that bandw is the half-bandwidth of
% the matrix.  If bandw is not specified, bandw = 3 is used.

% Per Christian Hansen, IMM, 09/03/97.

% Initialization.
if (nargin < 2), bandw = 3; end
bandw = min(bandw,N);

% Construct T via Kronecker product.
T = spdiags(ones(N,2*bandw-1),[-bandw+1:bandw-1],N,N)/(2*bandw-1);
T = kron(T,T);
```

Anexo III

Método *lsqr_b.m*

```
function [X,rho,eta,F] = lsqr_b(A,b,k,reorth,s)
%LSQR_B Solution of least squares problems by Lanczos bidiagonalization.
%
% [X,rho,eta,F] = lsqr_b(A,b,k,reorth,s)
%
% Performs k steps of the LSQR Lanczos bidiagonalization algorithm
% applied to the system
%    min || A x - b || .
% The routine returns all k solutions, stored as columns of
% the matrix X.  The solution norm and residual norm are returned
% in eta and rho, respectively.
%
% If the singular values s are also provided, lsqr computes the
% filter factors associated with each step and stores them columnwise
% in the matrix F.
%
% Reorthogonalization is controlled by means of reorth:
%    reorth = 0 : no reorthogonalization (default),
%    reorth = 1 : reorthogonalization by means of MGS.

% Reference: C. C. Paige & M. A. Saunders, "LSQR: an algorithm for
% sparse linear equations and sparse least squares", ACM Trans.
% Math. Software 8 (1982), 43-71.

% Per Christian Hansen, IMM, April 8, 2001.

% The fudge threshold is used to prevent filter factors from exploding.
```

```
fudge_thr = 1e-4;

% Initialization.
if (k < 1), error("Number of steps k must be positive"), end

if (nargin==3), reorth = 0; end
%if (nargout==4 & nargin<5), error('Too few input arguments'), end
[m,n] = size(A); X = zeros(n,k);
if (reorth==0)
  UV = 0;
elseif (reorth==1)
  U = zeros(m,k); V = zeros(n,k); UV = 1;
  if (k>=n), error("No. of iterations must satisfy k < n"), end
else
  error("Illegal reorth")
end
if (nargout > 1)
  eta = zeros(k,1); rho = eta;
  c2 = -1; s2 = 0; xnorm = 0; z = 0;
end
if (nargin==5)
  ls = length(s);
  F = zeros(ls,k); Fv = zeros(ls,1); Fw = Fv;
  s = s.^2;
end

% Prepare for LSQR iteration.
v = zeros(n,1); x = v; beta = norm(b);
if (beta==0), error("Right-hand side must be nonzero"), end
u = b/beta; if (UV), U(:,1) = u; end
r = A'*u; alpha = norm(r); % A'*u;
v = r/alpha; if (UV), V(:,1) = v; end
phi_bar = beta; rho_bar = alpha; w = v;
if (nargin==5), Fv = s/(alpha*beta); Fw = Fv; end

% Perform Lanczos bidiagonalization with/without reorthogonalization.
for i=2:k+1

  alpha_old = alpha; beta_old = beta;
```

```
% Compute A*v - alpha*u.
p = A*v - alpha*u;
if (reorth==0)
  beta = norm(p); u = p/beta;
else
  for j=1:i-1, p = p - (U(:,j)'*p)*U(:,j); end
  beta = norm(p); u = p/beta;
end

% Compute A'*u - beta*v.
r = A'*u - beta*v;
if (reorth==0)
  alpha = norm(r); v = r/alpha;
else
  for j=1:i-1, r = r - (V(:,j)'*r)*V(:,j); end
  alpha = norm(r); v = r/alpha;
end

% Store U and V if necessary.
if (UV), U(:,i) = u; V(:,i) = v; end

% Construct and apply orthogonal transformation.
rrho = norm([rho_bar,beta]); c1 = rho_bar/rrho;
s1 = beta/rrho; theta = s1*alpha; rho_bar = -c1*alpha;
phi = c1*phi_bar; phi_bar = s1*phi_bar;

% Compute solution norm and residual norm if necessary;
if (nargout > 1)
  delta = s2*rrho; gamma_bar = -c2*rrho; rhs = phi - delta*z;
  z_bar = rhs/gamma_bar; eta(i-1) = norm([xnorm,z_bar]);
  gamma = norm([gamma_bar,theta]);
  c2 = gamma_bar/gamma; s2 = theta/gamma;
  z = rhs/gamma; xnorm = norm([xnorm,z]);
  rho(i-1) = abs(phi_bar);
end

% If required, compute the filter factors.
if (nargin==5)
```

```
    if (i==2)
      Fv_old = Fv;
      Fv = Fv.*(s - beta^2 - alpha_old^2)/(alpha*beta);
      F(:,i-1) = (phi/rrho)*Fw;
    else
      tmp = Fv;
      Fv = (Fv.*(s - beta^2 - alpha_old^2) - ...
                  Fv_old*alpha_old*beta_old)/(alpha*beta);
      Fv_old = tmp;
      F(:,i-1) = F(:,i-2) + (phi/rrho)*Fw;
    end
    if (i > 3)
      f = find(abs(F(:,i-2)-1) < fudge_thr & abs(F(:,i-3)-1) < fudge_thr
   );
      if ~isempty(f), F(f,i-1) = ones(length(f),1); end
    end
    Fw = Fv - (theta/rrho)*Fw;

  end

  % Update the solution.
  x = x + (phi/rrho)*w; w = v - (theta/rrho)*w;
  X(:,i-1) = x;

end
```