INSTITUTO NACIONAL DE MATEMÁTICA PURA E APLICADA

TOMOGRAFIA SÍSMICA POR TEMPO DE PERCURSO: MODELAGEM, MÉTODOS NUMÉRICOS E IMPLEMENTAÇÃO

Autora: Gabriela Félix Brião

Orientador: Prof. Dr. Jorge P. Zubelli

2005

Resumo

Neste trabalho, nosso principal objetivo é estudar o problema inverso linear associado ao imageamento do subsolo compreendido entre dois poços já perfurados. A questão que se coloca é: Se possuirmos dados de tempo de percurso e dos raios emitidos entre um conjunto de fontes e receptores colocados estrategicamente em cada poço, como obter o perfil de velocidade? Para resolvê-la, utilizamos técnicas de álgebra linear numérica, otimização e análise numérica.

O trabalho compreende também um conjunto de exemplos que visam mostrar a eficiência dos métodos estudados, sobretudo o ART (Algebraic Reconstruction Technique) na tentativa de encontrar indícios da presença de hidrocarbonetos no solo.

Abstract

In this work, we are concerned with the linear inverse problem associated to seismic imaging of subsurface geophysical structures between bore holes. More precisely, if we are given traveltime data as well as the rays paths between a set of sources and receptors suitably located in each well, how do we reconstruct the velocity profile? To solve such problem, we use techniques from linear algebra, optimization and numerical analysis.

We provide a set of examples demostrating the efficiency of Algebraic Reconstruction Technique (ART) in the search of hydrocarbonet reservoirs.

Agradecimentos

- Agradeço aos meus familiares e amigos, em especial ao pessoal do Fluid pelo apoio.
- À CAPES e à ANP pelo suporte financeiro durante o mestrado.
- Agradeço também ao meu orientador professor Zubelli pelos vários sábados e domingos passados aqui no Impa.
- À Dayse Haime Pastore pela ajuda com o Matlab e no desenvolvimento do programa para cálculo do caminho percorrido.
- Ao Professor Christian Schaerer, pelas sugestões e por ter lido com cuidado este trabalho.

"Não utilize mais matemática do que os dados merecem."
Sven Treitel

Conteúdo

Introdução

O petróleo forma-se de material orgânico concentrado em um ambiente de sedimentos de baixa permeabilidade que é uma condição necessária para evitar a oxidação da matéria orgânica. Mais especificamente, o petróleo é uma mistura constituída predominantemente de hidrocarbonetos (composto químico constituído apenas por átomos de carbono e hidrogênio) e ocorre na natureza nos estados sólido, líquido e gasoso. [30]

A prospecção sísmica visa dois objetivos:

1. mapear o subsolo, classificando situações geológicas onde a acumulação de petróleo é possível;

2. avaliar dentre as formações rochosas, o local que tem mais chances de possuir o óleo.

O termo prospecção é utilizado para toda técnica empregada para localizar e calcular o valor econômico de jazidas minerais. As etapas que o constituem têm um custo reduzido comparado com o custo de perfuração de um único poço. Assim, a indústria petrolífera investe pesadamente no estudo de determinada região antes de iniciar o processo exploratório.

Estudaremos a técnica da tomografia sísmica ou sísmica "poço a poço", que consiste em obter dados de tempo de percurso de ondas acústicas emitidas por uma fonte localizada em um dos poços e captada por receptores no outro poço e, utilizar esses dados para imagear o subsolo. A região em estudo é dividida em pequenos pixels no caso bidimensional e em pequenos voxels no caso de querermos informações tridimensionais.

Como dados para a tomografia sísmica podemos citar dois: os de tempo de percurso e os de formato da onda. A tomografia por tempo de percurso apesar de nos dar uma resolução muito menor, computacionalmente é muito mais robusta e fácil de implementar que a tomografia de formato da onda.

O caminho percorrido pela onda sísmica é discretizado em função do número de células. Utilizando uma fonte em um poço e vários receptores em diferentes posições no outro poço é possível recobrir cada pixel com vários raios e

através de um tratamento computacional intenso podemos aproximar a velocidade em cada célula.

As frentes de ondas sísmicas geralmente não percorrem caminhos retilíneos como é o caso da tomografia médica. O que acontece é que o índice de refração de raios-x através do corpo humano é praticamente constante, muito diferente do que ocorre no subsolo terrestre, onde as ondas se comportam segundo a Lei de Snell.

Os dados de velocidade, bem como a interpretação das feições geológicas da região, são informações importantes para a determinação de acumulação de hidrocarbonetos no subsolo, o que nos indicaria uma possível reserva. Ver Figura 3.

Para a aquisição dos dados necessários para o problema inverso associado à tomografia por tempo de percurso, são utilizadas fontes de energia sísmica, como a dinamite, o vibrador em terra (ver Figura 2) e canhões de ar no mar.

Descreveremos o processo de inversão de dados sísmicos em simulações numéricas em cinco etapas.

1. Obtenção dos dados de tempo de percurso. Quando efetuamos simulações numéricas, tais dados podem ser simulados numericamente pela solução por diferenças finitas da equação eiconal associada ou por um método de minimização do tempo de percurso. Obtemos assim um vetor t de tamanho $m \times 1$, onde m é o produto do número de receptores pelo de detectores.

2. Denotando por l_{ij} o comprimento do i-ésimo caminho através da j-ésima célula, podemos escrever

$$t_i = \sum_{j=1}^{n} l_{ij} s_j,$$

onde s_j é o inverso da velocidade na j-ésima célula (supondo que a velocidade é constante em cada célula). Em notação matricial, $Ms = t$, onde $M = [l_{ij}]$.

Para obter a matriz M dos comprimentos dos raios, precisamos de algoritmos para calcular o traçado desses caminhos.

Exemplo:

Se numerarmos os caminhos da Figura 1 de cima para baixo e os pixels de baixo para cima e da esquerda para a direita, a matriz M representada a seguir tem como i-ésima linha os comprimentos do raio em cada pixel (representado nas colunas).

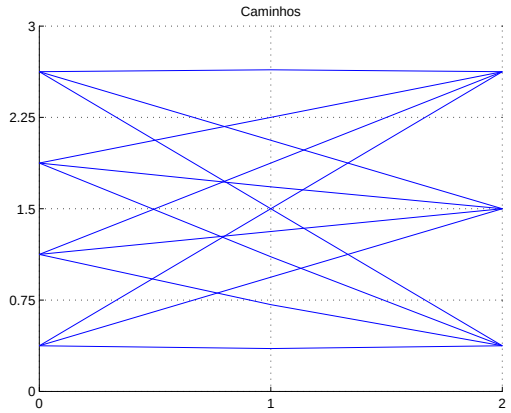

Figura 1: Um exemplo de percurso entre par fonte-receptor. Existem quatro fontes e três receptores linearmente espaçados.

$$M = \begin{pmatrix} 0 & 0 & 0 & 1.0001 & 0 & 0 & 0 & 1.0001 \\ 0 & 0 & 0.3825 & 0.7649 & 0 & 0 & 1.1473 & 0 \\ 0 & 0.0000 & 1.0035 & 0.5017 & 0.5017 & 1.0034 & 0 & 0 \\ 0 & 0 & 1.0664 & 0.0018 & 0 & 0 & 0 & 1.0678 \\ 0 & 0 & 1.0189 & 0 & 0 & 0 & 1.0160 & 0 \\ 0 & 0.6463 & 0.6152 & 0 & 0.6354 & 0.6033 & 0 & 0 \\ 0 & 0.6250 & 0.6250 & 0 & 0 & 0 & 0.6250 & 0.6250 \\ 0 & 1.0174 & 0 & 0 & 0 & 1.0174 & 0 & 0 \\ 0.0990 & 0.9828 & 0 & 0 & 1.0553 & 0 & 0 & 0 \\ 0.5017 & 1.0035 & 0.0000 & 0 & 0 & 0 & 1.0034 & 0.5017 \\ 0.7649 & 0.3825 & 0 & 0 & 0 & 1.1473 & 0 & 0 \\ 1.0003 & 0 & 0 & 0 & 1.0003 & 0 & 0 & 0 \end{pmatrix}$$

3. Escolhido um modelo inicial s' (geralmente através de fórmulas de retroprojeção), escrevemos,

$$t - t' = Ms - M's' \tag{1}$$

onde M' é a matriz associada ao modelo s'.

Linearizando a relação entre modelo e dados (1) se torna:

$$\delta t = M\delta s \tag{2}$$

Com isto, estamos implicitamente assumindo que pequenas perturbações no modelo não afetam o comprimento dos caminhos.

4. Utilizamos algum método numérico, como por exemplo, o dos mínimos quadrados amortizados, para obtermos solução:

$$\delta s = (M^{\mathrm{T}}M + \alpha I)^{-1}M^{\mathrm{T}}\delta t,$$

onde α é o parâmetro de regularização.

5. Avaliamos o modelo obtido.

O item 1) é uma área de pesquisa muito ativa. Foi abordado, por exemplo, na dissertação de mestrado de Leo Espin [10]. Ele consiste nos dados do problema direto.

O item 4), será o tema central deste trabalho. Mais especificamente, procuraremos modos eficientes e robustos de resolver a equação (2).

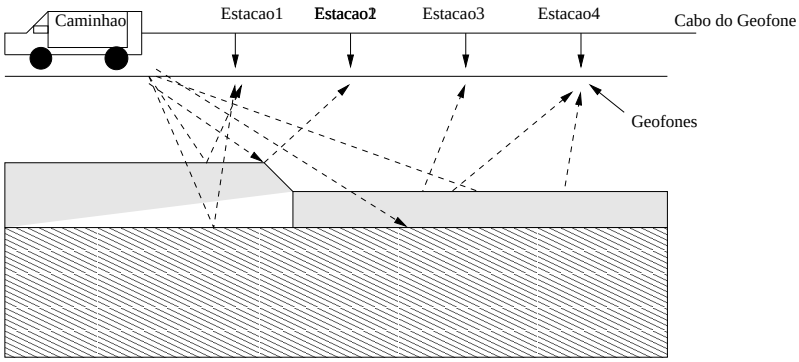

Figura 2: Exemplo de fonte muito utilizado. São produzidas vibrações de várias intensidades e receptores são acoplados junto ao chão de maneira a capturar as ondas resultantes das mesmas.

É natural que exista um limite de distância entre os poços para que possamos obter bons resultados. Este limite depende do que estamos tentando imagear e como estamos dispostos a fazê-lo (com qual tecnologia). Tipicamente, a distância entre as fontes em um experimento sísmico é de um metro, porém essa distância varia conforme o objetivo da imagem [19].

Ondas que se propagam em sólidos e líquidos são chamadas de ondas elásticas. Existem basicamente dois tipos: as ondas compressionais, mais conhecidas como ondas P, nas quais o deslocamento do meio se dá na mesma direção de propagação da energia e, as ondas de cisalhamento (ondas S), nas

quais o deslocamento do meio é perpendicular à direção de propagação da energia. Atualmente a indústria petrolífera utiliza somente as ondas P, sendo as ondas S alvo de intensas pesquisas. As técnicas matemáticas empregadas no presente trabalho podem em princípio ser aplicadas à ambas situações.

A velocidade de propagação das ondas sísmicas é função das constantes elásticas e da densidade do meio. Por isso depende, dentre outros, da temperatura, presença de microfraturas, porosidade (espaços vazios no interior das rochas).

A propagação de ondas elásticas é tipicamente descrita por equações hiperbólicas e se caracteriza por um domínio de influência associado à perturbação. Sendo assim surge naturalmente a noção de uma "frente de onda", ou seja, a fronteira (dependente do tempo) do suporte da onda associado a uma perturbação localizada no espaço.

É importante observar que a propagação das frentes de ondas elásticas é regida pelas mesmas leis da ótica geométrica.

A técnica de imageamento sísmico poço a poço ("borehole tomography") aproveita ao máximo os poços já perfurados obtendo dados utilizando elementos já disponíveis, o que pode diminuir custos na procura de reservas petrolíferas.

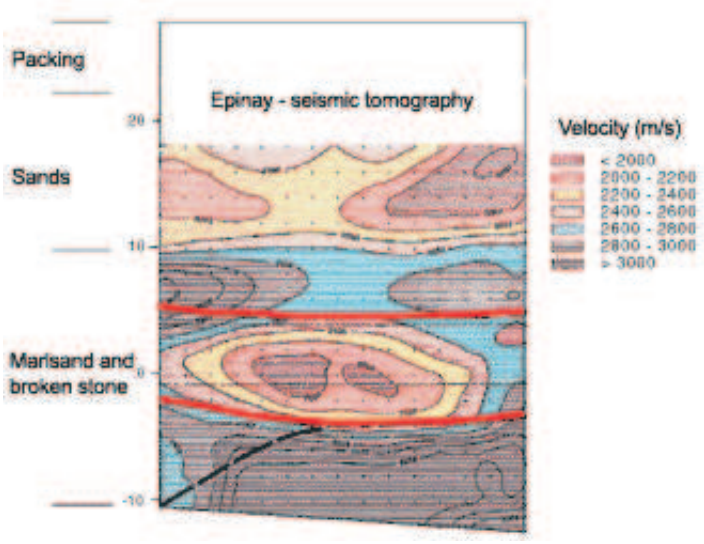

Figura 3: Tomograma sísmico. Com os dados de velocidade os geofísicos são capazes de inferir as formações geológicas no subsolo estudado formando o tomograma acima. [30]

O problema inverso associado à prospecção sísmica por tempo de percurso

é o de determinar a velocidade de propagação das ondas com base em medidas dos tempos de chegada das frentes de onda entre fontes e receptores. Com isto, os geólogos e geofísicos constroem seus tomogramas que recuperam a imagem do subsolo. Isto está exemplificado nas Figuras 3 e 4.

A dissertação está estruturada em três capítulos. No Capítulo 1, tratamos da modelagem física do problema, isto é, apresentamos os modelos geralmente utilizados no estudo da tomografia sísmica. Escolhendo um deles, a saber, o modelo da vagarosidade constante em cada pixel. Outro tema importante deste capítulo trata da distinção entre problemas diretos e problemas inversos. Em particular, o problema de resolver o sistema linear $Ms = t$ surge naturalmente em prospecção sísmica ao se discretizar o problema inverso, quando invariavelmente se recai num problema mal-posto com um número muito grande de variáveis e um número ainda maior de equações.

Finalmente, o Capítulo 1 é encerrado com exemplos de outros campos da ciência. Um desses exemplos é a medicina e, para citar outro, recentemente foi publicado que cientistas brasileiros estavam tentando reconstruir o rosto de múmias egípcias de mais de quatro mil anos, cujos sarcófagos não podiam ser violados. Para isto, era necessário todo um processo de reconstrução tomográfica. Ver site [27].

No Capítulo 2, desenvolvemos ferramentas teóricas e métodos numéricos que utilizamos para solucionar o problema de reconstrução. Começamos o capítulo apresentando a pseudo-inversa de Moore-Penrose, a qual nos dá uma solução ótima em um certo sentido para o problema em questão, porém muitas vezes não é muito prático calculá-la. Depois, dividimos o capítulo em duas seções importantes: métodos diretos e métodos iterativos. Os métodos iterativos são particularmente interessantes na resolução do problema inverso proposto, uma vez que trabalhamos com matrizes esparsas.

No último capítulo da dissertação, implementamos o método direto de decomposição QR e os métodos iterativos dos gradientes conjugados e ART. Estes últimos são muito utilizados no caso de matrizes mal-condicionadas [1].

Comparamos os métodos estudando um conjunto de exemplos e, para finalizar, apresentamos nossas conclusões.

[1]Por matrizes mal-condicionadas entendemos M tal que o número de condição $k(M) >> 1$ onde, $k(M) = \|M\|\|M^{-1}\|$. Este número nos dá uma forma de examinar como pertubações em M e t afetam a solução s

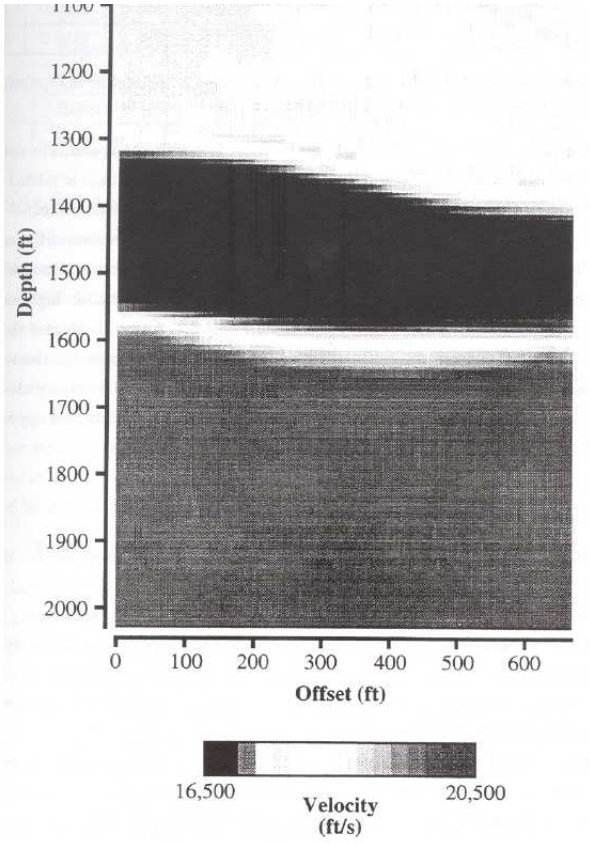

Figura 4: Um exemplo de tomograma sísmico relacionando profundidade com velocidade. [5]

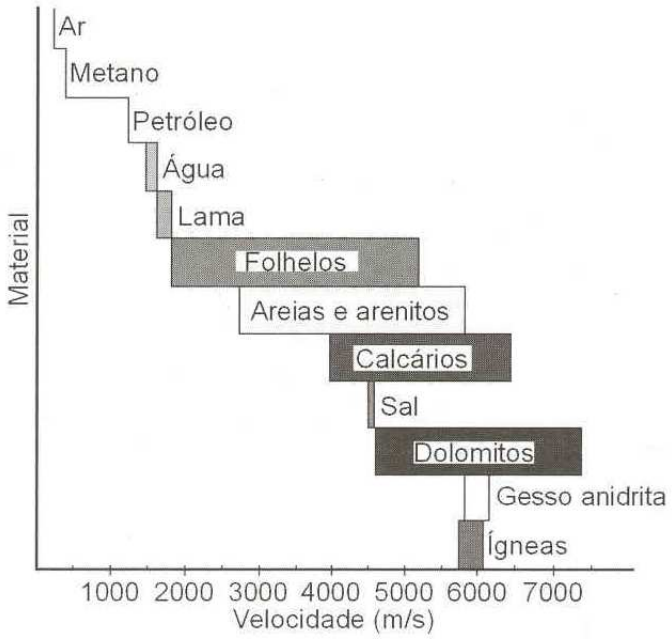

Figura 5: Gráfico de materiais de acordo com a velocidade. Assim, com a solução obtida pela tomografia de tempo de percurso, os geólogos e geofísicos podem inferir que tipo de material existe no subsolo.

Capítulo 1

Modelagem Física

O problema de reconstruir imagens por projeções, isto é, reconstruir uma função através de suas integrais ao longo de retas surgiu independentemente em diversos ramos da ciência tais como a geofísica, a astrofísica, a medicina dentre outros.

Provavelmente os exemplos que causaram maior impacto na vida moderna foram em prospecção sísmica e na tomografia computadorizada voltada para diagnósticos clínicos.

1.1 Reconstrução na Tomografia Sísmica de Tempo de Percurso

A palavra tomografia vem do prefixo grego "tomo"que quer dizer fatia, o que nos sugere uma reconstrução em 2-D. Mas o termo já é utilizado rotineiramente para se referir a reconstrução de imagens em 3-D, sobretudo pelos sismólogos e radiólogos.

A tomografia sísmica tem como preocupação, a reconstrução de imagens da estrutura terrestre. Dentre as técnicas existentes focalizaremos a que utiliza dados de tempo de percurso.

Iniciaremos com a descrição do Princípio de Fermat. Neste contexto, é natural introduzir a vagarosidade, ou seja, a inversa da velocidade.

Dada s uma distribuição contínua da vagarosidade $s(x)$, o *tempo de percurso* de um sinal ao longo de um possível caminho que liga a fonte que o emitiu ao receptor é dado por:

$$\tau^P(s) = \int_P s(x) dl^P = \int_P \frac{1}{v(x)} dl^P \tag{1.1}$$

onde dl^P denota o comprimento de arco ao longo do caminho P. Denotemos por Γ o conjunto de todos os possíveis caminhos ligando a fonte ao receptor.

O Princípio de Fermat diz que o caminho físico percorrido por uma onda entre dois pontos é aquele que minimiza o tempo de percurso.

$$\tau^*(s) = \min_{P\in\{\Gamma\}}\tau^P(s) \tag{1.2}$$

O funcional $P \mapsto \tau^P(s)$ de tempo de percurso é estacionário com respeito a pequenas pertubações no caminho de Fermat $P^*(s)$ no sentido do cálculo das variações [1]. Observe que (1.2) depende de forma não linear em s, como consequência do processo de minimização.

Uma forma de obter dados na tomografia sísmica é feita aproveitando a existência de poços já perfurados. Isto é, colocando transmissores em um dos poços e receptores em outro de maneira que são emitidas ondas entre os poços. Tais ondas podem ser sísmicas ou eletromagnéticas. Através da utilização de receptores apropriados mede-se o tempo de chegada das mesmas.

Esses dados, apesar de imprecisos e ruidosos, podem ser utilizados para obtermos informações sobre a composição do subsolo e a presença de hidrocarbonetos.

1.2 Problema Direto, Problema Inverso e Tomografia

Dentro de uma aproximação aceitável para muitas finalidades em geofísica, podemos modelar as ondas sísmicas como soluções da equação diferencial parcial:

$$\partial_t^2\phi - c^2(x)\Delta\phi = g(x,t)$$

onde $g(x,t)$ é a intensidade da pertubação num determinado ponto x e tempo t. Temos que $\phi(x,t)$ é a intensidade da onda no tempo t e posição x. Esta equação deve ser complementada com condições de contorno apropriadas.

O problema direto consiste em resolver a equação dado g, isto é, encontrar a solução ϕ. No caso da tomografia por tempo de percurso, a preocupação é determinar o tempo de percurso da frente de onda e o caminho percorrido pela mesma entre fonte e receptor. Para saber mais sobre as técnicas de resolução desse tipo de problema direto ver [24].

Já no problema inverso (Figura 1.1a), queremos obter informações sobre os coeficientes da equação utilizando dados sobre suas soluções em regiões distintas. As técnicas de problemas inversos são de grande interesse na prospecção sísmica, pois têm por objetivo, determinar o interior da região em estudo

somente com base em informação parcial dos dados no exterior da mesma. Assim, nos propomos a reconstruir os valores de $c(x)$ com base na solução da equação da onda medida na fronteira que delimita a região de interesse. Com essas informações em mãos, os geólogos e geofísicos podem inferir que tipo de material existe no subsolo estudado fazendo uso de informações como por exemplo na Figura 5.

Na inversão linear na tomografia por tempo de percurso, assumimos à priori que sabemos o traçado dos feixes que ligam fonte a receptor, o que é justificado por uma aproximação linear que ignora a dependência que os caminhos possuem da distribuição da vagarosidade (Princípio de Fermat).

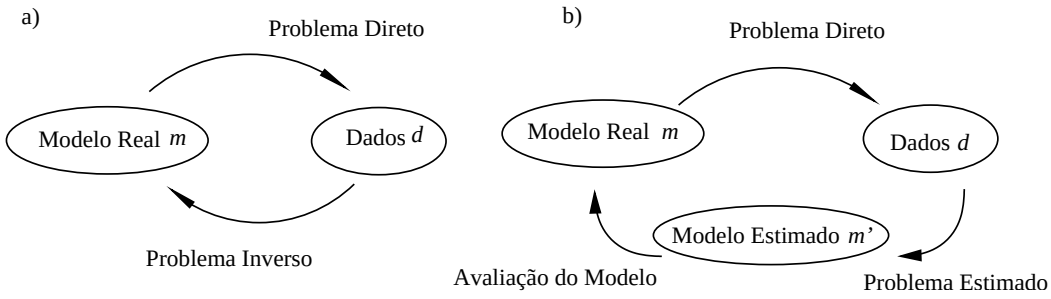

Figura 1.1: a) Problema Direto versus Problema Inverso; b) O problema inverso visto com duas etapas

Na Figura 1.1a descrevemos simplificadamente uma comparação entre o problema direto e o problema inverso. O problema inverso é relativamente mais complicado, uma vez que, em problemas reais, fixados os dados, podemos construir infinitos modelos que se adequam a estes mesmos dados. No problema inverso, muitas vezes não há essa unicidade levando dos dados ao modelo. Assim, chegamos a um esquema mais adequado à realidade na Figura 1.1b.

A não unicidade do problema inverso pode ser explicada pelo fato de possuirmos somente uma quantidade finita de dados coletados para obter um modelo que muitas vezes é uma função contínua de suas variáveis, o que significa, que o mesmo possui infinitos graus de liberdade. Por causa dessa limitação física da finitude dos dados, o modelo que alcançamos através dos dados coletados não é necessariamente o que modela a realidade.

São necessários dois passos na inversão para chegarmos a um modelo mais próximo da realidade. Isto é representado na Figura 1.1b. Vale salientar que em última análise o modelo verdadeiro não é sabido em problemas reais.

O primeiro passo seria então reconstruir um modelo m' utilizando os dados d. Uma vez feito isto, determinamos que propriedades o modelo m' preserva

do modelo real m e que tipo de erros e ruídos estão associados a ele, ou seja, fazemos uma avaliação do modelo.

1.3 Modelos (Representação da Estrutura)

Apresentaremos aqui duas maneiras de parametrizar a vagarosidade. O mais simples seria dividir a região em pequenos blocos (denominados pixels no caso 2-D e voxels no caso 3-D) e atribuir valores constantes à vagarosidade em cada bloco. Isto pode ser visto na Figura 1.2a.

Uma alternativa a este modelo é definir vagarosidade nos vértices da malha formada pela divisão da região em blocos (Figura 1.2b). Essa definição seria formulada em conjunto com uma função de interpolação. Um exemplo ilustrativo disto seria no contexto de tomografia local de terremotos, tal que para cada vértice (x, y, z) é utilizada uma interpolação trilinear (figura 1.2c):

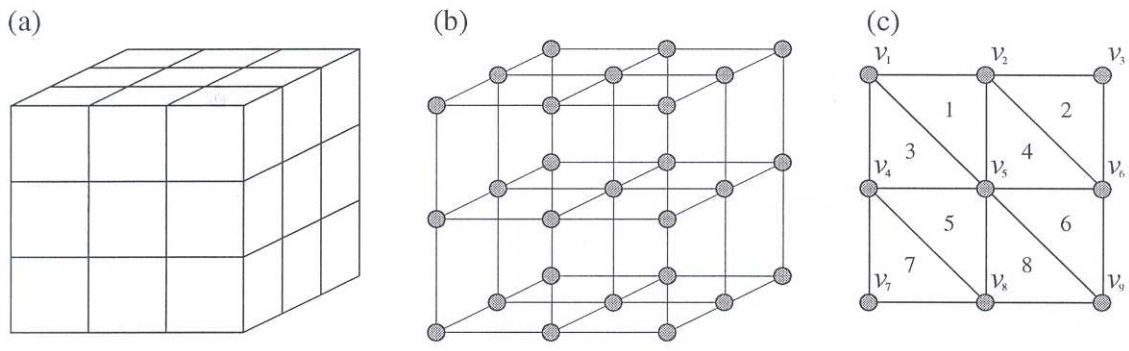

Figura 1.2: Modelos

$$v(x,y,z) = \sum_{i=1}^{2}\sum_{j=1}^{2}\sum_{k=1}^{2} V(x_i, y_j, z_k)\big(1 - |\frac{x - x_i}{x_2 - x_1}|\big)\big(1 - |\frac{y - y_j}{y_2 - y_1}|\big)\big(1 - |\frac{z - z_k}{z_2 - z_1}|\big)$$

onde $V(x_i, y_j, z_k)$ são os valores da velocidade nos oito vértices que cercam o vértice (x, y, z).

Neste trabalho estamos interessados em estudar o primeiro modelo acima.

Sendo assim, considere $t_1, \ldots, t_m$ conjunto de tempos de percurso entre fonte e receptor. Dado um modelo com n células, podemos escrever,

$$t_i = \sum_{j=1}^{n} l_{ij} s_j,$$

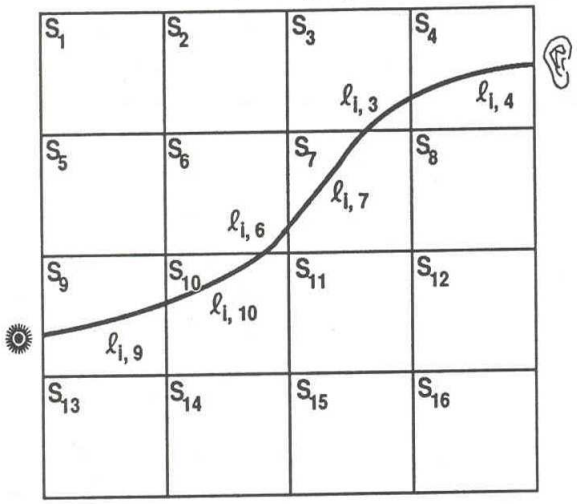

Figura 1.3: Tempo de percurso para a i-ésima frente de onda, onde está sendo utilizado o modelo discretizado com vagarosidade constante em cada pixel.

ou melhor, $Ms = t$. Onde M é a matriz formada pelo comprimento l_{ij} do i-ésimo raio que passa pela j-ésima célula e s é a vagarosidade (a nossa incógnita). Observe que

$$l_{ij} = \frac{\partial t_i}{\partial s_j}$$

e assim,

$$t_i = \frac{\partial t_i}{\partial s_1} s_1 + \frac{\partial t_i}{\partial s_2} s_2 + \cdots + \frac{\partial t_i}{\partial s_n} s_n.$$

Assim, discretizando o domínio da vagarosidade obtemos um sistema de equações lineares, onde a matriz do sistema é muito esparsa [1] porque cada raio intersecta somente uma pequena fração dos voxels da discretização (ver Figura 1.3). Neste trabalho o enfoque são os pixels em 2-D, cada raio intersecta algo da ordem de $m * n$ pixels em uma malha $m \times n$. Isso torna o problema particularmente atrativo para a utilização de soluções iterativas.

A matriz M contém todas as informações físicas e matemáticas que escolhemos para o modelo no problema dado. Assim, no caso da tomografia por tempo de percurso, a matriz M terá como suas componentes os dados do comprimento das trajetórias.

[1] A densidade de uma matriz é o número de elementos não nulos dividido pelo total de elementos da matriz. Se esse número for muito pequeno essa matriz é dita esparsa.

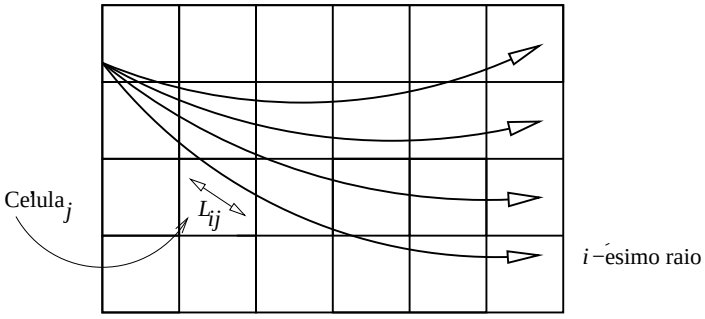

Figura 1.4: Diagrama de um experimento tomográfico. A matriz M geralmente é esparsa, pois existem células por onde não passam nenhum raio do experimento.

1.4 Regularização

Problemas em tomografia geralmente são mal-postos, isto é, problemas que falham, seja na existência de soluções, na unicidade dessas soluções ou mesmo que a solução não depende continuamente dos dados. Sendo assim, são utilizadas frequentemente técnicas de regularização para dar estabilidade ao problema [21]. Essas técnicas nos permitem solucionar não o problema original mas sim, um problema similar, porém mais robusto em relação a erros nos dados.

Considere o problema de resolver $Ms = t$. Do ponto de vista matemático, no caso de modelos lineares, esses problemas mal-postos se devem geralmente ao fato da matriz M possuir valores singulares nulos ou muito próximos de zero. Uma das formas de contornar isto seria acrescentar à matriz $M^{\mathrm{T}}M$ um múltiplo da matriz identidade de tal maneira que essa nova matriz possua somente valores singulares positivos, porém distantes do zero. De fato, considerando $B = M^{\mathrm{T}}M + \gamma I$, temos que se $\gamma \neq 0$, os autovalores de B ficam diferentes de zero (positivos).

Feito isto, podemos definir a solução de mínimos quadrados amortecidos do sistema original por:

$$s' = (M^{\mathrm{T}}M + \gamma I)^{-1}M^{\mathrm{T}}t$$

A escolha de um bom parâmetro γ é fundamental nos problemas mal-postos. O número γ é chamado de parâmetro de regularização.

A não existência ou a perda de unicidade das soluções se devem ao fato de $t \notin Im(M)$ ou a não injetividade da transformação M, respectivamente. Nesses casos, a utilização da pseudo-inversa $M^{\dagger}$ é o mais conveniente, conforme

estudado no Capítulo 2. A técnica de regularização de Tikhonov consiste em obter certas transformações dadas denotadas por $A_\lambda : \mathbb{R}^n \longrightarrow \mathbb{R}^m$, $\lambda > 0$, de tal forma,

$$\lim_{\lambda \to 0} A_\lambda t = M^\dagger t$$

onde $M^\dagger$ é a pseudo-inversa da matriz M.

Seja $t_\epsilon \in \mathbb{R}^n$ tal que $\|t - t_\epsilon\| \leqslant \epsilon$. E seja $\lambda(\epsilon)$ tal que, quando $\epsilon \to 0$

$$\lambda(\epsilon) \to 0$$

e

$$\|A_{\lambda(\epsilon)}\|\epsilon \to 0$$

assim,

$$\begin{aligned} \|A_{\lambda(\epsilon)} t_\epsilon - M^\dagger t\| &\leqslant \|A_{\lambda(\epsilon)} t_\epsilon - A_{\lambda(\epsilon)} t\| + \|A_{\lambda(\epsilon)} t - M^\dagger t\| \\ &\leqslant \|A_{\lambda(\epsilon)}\|\|t_\epsilon - t\| + \|A_{\lambda(\epsilon)} t - M^\dagger t\| \longrightarrow 0. \end{aligned}$$

Logo, se t_ϵ está próximo do tempo de percurso t então $A_{\lambda(\epsilon)} t_\epsilon$ está próximo da solução aproximada $M^\dagger t$ no sentido que veremos mais tarde no próximo capítulo.

Como exemplo desse método consideremos a parada em um processo iterativo. Seja então,

$$s^{(k+1)} = B_k s^{(k)} + C_k t$$

um processo iterativo e assuma que $s^{(k)} \to A^\dagger t$. Para cada $\lambda > 0$, seja $k(\lambda)$ índice tal que $k(\lambda) \to \infty$ quando $\lambda \to 0$. Então afirmamos que $A_\lambda t = s^{(k(\lambda))}$ é uma regularização, pois

$$\lim_{\lambda \to 0} A_\lambda t = \lim_{\lambda \to 0} s^{(k(\lambda))} = A^\dagger t$$

por hipótese.

Frequentemente no caso da tomografia por tempo de percurso a matriz M (associada aos tempos de percurso dos raios sobre as células) tem posto deficiente ou é extremamente mal-condicionada. Isto leva naturalmente a necessidade de regularização.

1.5 Exemplos em outras áreas

Nesta seção temos por objetivo apresentar diversas áreas científicas onde a reconstrução de imagens através de projeções se apresenta útil.

Uma delas é a astrofísica onde dados coletados de rochas enviadas para fora da atmosfera terrestre são utilizados para reconstruir a estrutura de uma supernova remanescente. [30]

Sem dúvida, o exemplo mais evidente ao público em geral é a medicina, onde a reconstrução tem papel fundamental nos diagnósticos clínicos.

Enviando um feixe de raios-x através do paciente são coletados dados que serão utilizados na recontrução da estrutura interna do corpo do paciente. A radiação se propaga dentro de uma boa aproximação em linha reta e é absorvida de acordo com:

$$\log(\frac{I}{I_0}) = -\int_L \mu(s)ds,$$

onde,

1. I_0 é a intensidade da radiação da fonte;
2. I é a intensidade da radiação medida pelo receptor;
3. L é a linha reta que liga fonte a receptor e
4. μ é o coeficiente de absorção.

O coeficiente de absorção μ é a variável de interesse que estamos tentando observar.

Fazendo um paralelo com o funcional de tempo de percurso, estamos lidando com o problema linear de obter uma grandeza, utilizando técnicas de resolução de sistemas lineares.

Nesse ponto, vale ressaltar que a tomografia de tempo de percurso na geofísica é altamente não linear o que a torna bem mais complicada do que a tomografia na medicina. Isso se deve ao fato que o percurso da frente de onda é uma curva que depende da vagarosidade. As ondas utilizadas para a coleta de dados na tomografia por tempo de percurso estão sujeitas à *Lei de Snell* e o índice de refração está longe de ser constante. Assim, não necessariamente, os percursos são em linha reta.

A Lei de Snell relaciona o índice de refração e o ângulo de incidência de uma onda incidindo em dois meios de diferentes vagarosidades (índice de refração). O índice de refração é dado por $n = \frac{c}{v}$, onde c é a velocidade da luz no vácuo (ou mais geralmente, da onda em um meio homogêneo).

Mais especificamente, a Lei de Snell é dada por

$$s_1 sen(\theta_1) = s_2 sen(\theta_2)$$

onde, θ_1 e θ_2 denotam os ângulos do raio de incidência e do raio refratado à normal ao plano que separa as duas regiões, respectivamente.

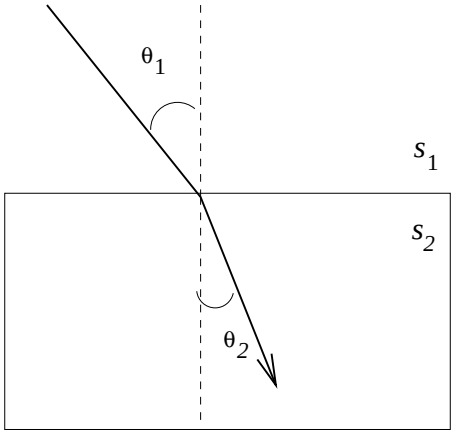

Figura 1.5: Lei de Snell

Uma aproximação aceitável para alguns problemas de tomografia por tempo de percurso é supor linhas retas por parte no percurso. Isto é computacionalmente muito vantajoso, uma vez que necessitamos tão somente da localização de fonte e receptor.

Os caminhos percorridos pela frente de onda geralmente não formam linhas retas, porém linearizaremos o problema. Iremos supor que estes caminhos *dentro de cada célula* formam uma linha reta como uma boa aproximação. A técnica de linearização da tomografia de tempo de percurso é baseada no princípio de Fermat, que tem como outra formulação a seguinte: o tempo de percurso ao longo de um raio não muda em primeira ordem quando esse raio é perturbado. Assim, comparativamente o problema da tomografia por tempo de percurso na geofísica se comporta da mesma maneira que o problema da tomografia na medicina.

Além dos exemplos citados acima, técnicas de tomografia são utilizadas na microscopia eletrônica e no imageamento dos oceanos.

1.6 Preliminares

Concluímos este capítulo com uma coletânea de resultados que serão importantes à seguir.

- Teorema dos valores singulares

  Teorema 1. *(Teorema dos valores singulares) Seja $A : E \longrightarrow F$ uma transformação linear de posto r entre espaços de dimensão finita, de dimensões m e n, respectivamente, com produto interno. Existem bases ortonormais $\{u_1, ..., u_m\} \subset E$ e $\{v_1, ..., v_n\} \subset F$ tais que $Au_i = \sigma_i v_i$ e $A^* v_i = \sigma_i u_i$ onde $\sigma_i > 0$ para $i = 1, 2 \ldots, r$ e $\sigma_i = 0$ para $r + 1 \leqslant i \leqslant \min n, m$.*

  Os números não-negativos $\sigma_1, \sigma_2, \ldots, \sigma_r, \ldots$ chamam-se os *valores singulares* da transformação linear A.

  A versão matricial deste teorema nos dá a famosa decomposição em valores singulares (SVD).

  Teorema 2. *Para toda $A \in M_{m\times n}(\mathbb{R})$ existem matrizes ortogonais $U \in M_m(\mathbb{R})$, $V \in M_n(\mathbb{R})$, tais que $U\mathrm{T}AV = \Sigma$, onde $\Sigma \in M_{m\times n}(\mathbb{R})$ é uma matriz diagonal, i.e., $d_{ij} = 0$ se $i \neq j$. Para $i = 1, \ldots, r = \mathrm{posto}(A)$ tem-se $d_{ii} = \sigma_i > 0$, e para $i > r$ tem-se $d_{ii} = 0$. Os números σ_i são os valores singulares, cuja existência foi assegurada no teorema dos valores singulares.*

  Com este teorema podemos decompor qualquer tipo de matriz A na seguinte forma $A = U\Sigma V^{\mathrm{T}}$.

- Equação eiconal

  A equação eiconal é dada por:

  $$|\nabla\phi| = s(x),$$

  esta equação surge naturalmente no estudo da equação da onda em um meio heterogêneo de vagarosidade $s(x)$, quando consideramos a teoria assintótica de altas frequências.

  O tempo de percurso de uma frente de onda pode ser modelado pela equação eiconal. Esta equação é não-linear de primeira ordem, o que exige um estudo das equações características associadas para determinar soluções.

  Note que se resolvermos a equação eiconal em *uma dimensão* obtemos um funcional τ dado por:

$$\tau(x) = \int_0^x s(l)dl,$$

o qual representa o tempo de percurso de uma frente de onda da origem até o ponto x.

Capítulo 2

Métodos Numéricos Associados ao Problema

O objetivo deste capítulo é apresentar métodos para solucionar o problema inverso associado à tomografia linear que sejam ao mesmo tempo rápidos, estáveis e fáceis de implementar. Iniciaremos com alguns resultados teóricos que são importantes para a análise dos algorítmos.

Observamos que apesar de vários métodos serem apresentados, implementaremos somente alguns destes. Os métodos escolhidos para formar o conjunto de exemplos descritos no Capítulo 3 são: o método QR, o método dos gradientes conjugados e o método ART.

2.1 A pseudo-inversa de Moore-Penrose

Como sabemos, nem toda matriz possui uma inversa. Nesta seção iremos generalizar o conceito de inversa de modo que toda matriz real (inclusive as retangulares) possuam essa inversa generalizada, a qual chamaremos de pseudo-inversa de Moore-Penrose.

Por ser uma generalização, é natural construir a pseudo-inversa de modo que satisfaça o maior número possível de propriedades de uma inversa. Além disso, desejamos que no caso de M ser uma matriz não-singular, a inversa e a pseudo-inversa coincidam.

A pseudo-inversa possui um papel estratégico na resolução de sistemas lineares inconsistentes, o que é muito comum na tomografia sísmica bem como em outras áreas. Quando é impossível achar $s \in \mathbb{R}^n$ tal que $Ms = t$, a inversa generalizada nos permite encontrar entre todos os possíveis vetores $s_0 \in \mathbb{R}^n$ cujo erro $\|Ms_0 - t\|$ é o menor possível o de menor norma. Assim, podemos obter uma solução aproximada no seguinte sentido:

Definição 1. *Seja a matriz $M \in \mathbb{R}^{m\times n}$. A inversa generalizada de M é a única matriz $M^\dagger$ que satisfaz:*

1. $MM^\dagger M = M$;
2. $M^\dagger MM^\dagger = M^\dagger$;
3. $(MM^\dagger)^{\mathrm{T}} = MM^\dagger$;
4. $(M^\dagger M)^{\mathrm{T}} = M^\dagger M$.

Essas expressões são frequentemente chamadas de condições de Penrose. No Apêndice A, mostramos que se uma matriz X satisfaz todas essas equações então $X = M^\dagger$ (unicidade).

Apesar de resolver o problema teoricamente, o cálculo da pseudo-inversa, na maioria dos casos, é muito caro do ponto de vista computacional. Por exemplo, no caso de uma matriz quadrada $n \times n$ é da ordem de n^3 operações. [12]

Os métodos que veremos a seguir podem ser analisados mais convenientemente em termos de sua convergência para a inversa generalizada. Muitas vezes encontramos inversas aproximadas, que não satisfazem as quatro condições de Penrose, mas que de alguma forma se aproximam da pseudo-inversa.

Note que se uma inversa aproximada satisfaz as condições de Penrose, então, por unicidade, é a pseudo-inversa (ver apêndice A).

Definição 2. *Para que s_0 seja uma solução ótima (aproximada) da equação*

$$Ms = t,$$

uma das duas condições têm que ser satisfeitas para todo s:

a) $\|Ms - t\| > \|Ms_0 - t\|$,

b) $\|Ms - t\| = \|Ms_0 - t\| \quad e \quad \|s\| \geqslant \|s_0\|$.

Teorema 3. *$M^\dagger t$ é a única solução aproximada da equação $Ms = t$ na norma $l_2(\mathbb{R}^n)$.*

Para demonstração ver [23].

Veremos agora o significado das condições de Penrose. Para isto, definiremos as matrizes de resolução que nos ajudarão a analisar o grau de proximidade da solução ao modelo s utilizando a pseudo-inversa $M^\dagger$. Em particular, definiremos também matrizes análogas para problemas onde a inversa aproximada X é utilizada no lugar de $M^\dagger$.

Definição 3. *As matrizes de resolução são dadas por:*

$$\begin{aligned} R_{mod} &= M^\dagger M \\ R_d &= MM^\dagger \end{aligned}$$

Observe que $R_{mod}^2 = R_{mod}$ e $R_d^2 = R_d$ e que $R_{mod}^{\perp} = R_{mod}$ e $R_d^{\perp} = R_d$. Logo R_d é projeção ortogonal sobre a imagem de M (resp. R_{mod} é projeção ortogonal sobre o complemento ortogonal do núcleo de M).

A inversa generalizada (Penrose) nos dá a melhor escolha dentre todas as inversas aproximadas no sentido de que suas matrizes de resolução são as mais próximas da identidade. Se a matriz de resolução é a identidade então encontramos uma inversa à esquerda no caso da matriz de resolução do modelo e, no outro caso, uma inversa à direita.

As **matrizes efetivas de resolução** são aquelas onde a inversa aproximada X toma o lugar de $M^{\dagger}$, isto é,

$$\begin{aligned} \mathcal{E}_{mod} &= XM \\ \mathcal{E}_{dad} &= MX. \end{aligned}$$

Para exemplificar o nosso interesse nas matrizes de resolução, considere

$$X = (M^{\mathrm{T}} F^{-1} M + \mu G)^{-1} M^{\mathrm{T}} F^{-1}$$

onde F e G são matrizes positivas e diagonais, $m \times m$ e $n \times n$ respectivamente. Afirmamos que X é uma inversa aproximada de M.

Para simplificar, consideremos o caso onde $F = G = I$. Então, decompondo $\mathcal{E}_{mod}$ em valores singulares obtemos,

$$\mathcal{E}_{mod} = XM = (M^{\mathrm{T}} M + \mu I)^{-1} M^{\mathrm{T}} M = \sum_{j=1}^{r} \frac{\lambda_j^2}{\lambda_j^2 + \mu} z_j z_j^{\mathrm{T}}$$

A matriz efetiva de resolução $\mathcal{E}_{mod}$ está relacionada com a matriz de resolução R_{mod} da seguinte forma:

$$\sum_{j=1}^{r} z_j z_j^{\mathrm{T}} = \mathcal{E}_{mod} + \mu \sum_{j=1}^{r} \frac{1}{\lambda_j^2 + \mu} z_j z_j^{\mathrm{T}}$$

e, assim, obtemos

$$\mathcal{E}_{mod} = R_{mod} + \sum_{j=1}^{r} \frac{\mu}{\lambda_j^2 + \mu} z_j z_j^{\mathrm{T}}$$

Logo, $\mathcal{E}_{mod} \to R_{mod}$ quando $\mu \to 0^+$. Da mesma forma, obtemos resultado análogo para a matriz $\mathcal{E}_{dad}$, o que nos mostra que esta inversa aproximada se aproxima da inversa generalizada de Penrose.

A simetria para as matrizes de resolução é uma propriedade desejável. Assim, as duas últimas condições de Penrose significariam que as matrizes efetivas de resoluçao são simétricas.

As condições $XMX = X$ e $MXM = M$ formam de alguma maneira condições de unicidade como veremos a seguir. No primeiro caso, temos uma condição de unicidade sobre a inversa aproximada X. Para ver que as condições de Penrose excluem a possibilidade de termos contribuições do núcleo de M, o que poderia fazer com que perdêssemos a unicidade, considere a decomposição em valores singulares de M.

$$M = \sum_i \sigma_i u_i v_i^{\mathrm{T}}$$

e

$$X = \sum_i \sigma_i^{-1} v_i u_i^{\mathrm{T}} + \epsilon_0 v_0 u_0^{\mathrm{T}}$$

onde $Mv_0 = 0$ e $u_0^{\mathrm{T}} M = 0$.

A condição $XMX = X$ nos faz concluir que $\epsilon_0 = 0$, pois o lado esquerdo nos diz que não podemos ter contribuições do tipo $v_0 u_0^{\mathrm{T}}$.

De maneira análoga, a condição de Penrose $XMX = M$ é um tipo de condição de unicidade sobre M.

Qualquer vetor no espaço dos modelos pode ser descrito como:

$$s = \sum_{i=1}^{r} \alpha_i v_i + s_0. \tag{2.1}$$

onde s_0 está no núcleo à direita de M.

Similarmente, um vetor de dados pode ser descrito como:

$$t = \sum_{i=1}^{r} \tau_i u_i + t_0. \tag{2.2}$$

onde t_0 é um vetor do núcleo à esquerda de M.

Assim, as matrizes de resolução retiram da decomposição dos vetores, (2.1) e (2.2), dos vetores do modelo e dos dados, a parte correspondente ao núcleo à direita e à esquerda, respectivamente, da matriz M.

A maneira mais utilizada para se encontrar a inversa generalizada da matriz M é a decomposição em valores singulares. Assim, se

$$M = \sum_{i=0}^{r} \sigma_i u_i v_i^{\mathrm{T}},$$

Afirmamos então que a pseudo-inversa de Moore-Penrose pode ser expressa por,

$$M^{\dagger} = \sum_{i=0}^{r} \lambda_i^{-1} z_i y_i^{\mathrm{T}}.$$

De fato, sejam então U e V matrizes ortogonais e Σ matriz diagonal tal que $M = U\Sigma V^{\mathrm{T}}$ seja a decomposição de M em valores singulares. Queremos mostrar que

$$M^{\dagger} = V\Sigma^{\dagger}U^{\mathrm{T}}.$$

Para isto, mostraremos que $M^{\dagger}$ satisfaz as condições de Penrose. Verificaremos (1):

$$MM^{\dagger}M = M,$$

pois

$$MM^{\dagger}M = U\Sigma V^{\mathrm{T}}V\Sigma^{\dagger}U^{\mathrm{T}}U\Sigma V^{\mathrm{T}} = U\Sigma\Sigma^{\dagger}\Sigma V^{\mathrm{T}} = U\Sigma V^{\mathrm{T}} = M.$$

A terceira igualdade segue-se porque $\Sigma^{\dagger}$ é evidentemente a pseudo-inversa de Σ, e essa inversa generalizada tem em sua diagonal o inverso dos elementos não nulos da diagonal de Σ.

A demonstração que as outras condições são válidas é análoga.

2.2 Métodos Diretos de Resolução

Os métodos diretos de resolução são aqueles que envolvem um número finito de operações elementares. Em geral, esses métodos utilizam algum tipo de fatoração da matriz M em questão.

Métodos desse tipo podem ser pouco práticos se a matriz for muito grande e esparsa como é o caso das aplicações à geofísica. Na próxima seção veremos métodos mais aplicáveis aos problemas reais em tomografia.

O método dos mínimos quadrados usual corresponde em minimizar o funcional

$$\psi(s) = (t - Ms)^{\mathrm{T}}(t - Ms)$$

O modelos s que minimiza esse funcional satisfaz a chamada equação normal

$$M^{\mathrm{T}}Ms = M^{\mathrm{T}}t$$

Se na inversão discretizada, temos mais dados que número de células de vagarosidade constante, o problema é dito *superdeterminado*. Caso a situação se inverta, dizemos que o problema é *subterminado*. Se tivermos tantos dados quanto células e além disso M tiver posto máximo, o problema tem solução

única. No caso em que o posto de M é maior ou igual a dimensão do vetor t, temos que

$$s = (M^{\mathrm{T}}M)^{-1}M^{\mathrm{T}}t$$

É muito mais comum na geofísica lidarmos com um problema superdeterminado. Porém, existem casos, em que é interessante usar somente uma parte dos dados que possuímos. Assim, que os problemas pouco determinados também são interessantes no estudo da tomografia por tempo de percurso.

2.2.1 Eliminação Gaussiana

Considere o sistema $Ms = t$. A eliminação gaussiana é um algoritmo que sob certas hipóteses dá uma decomposição da matriz quadrada M em um produto LU onde L é uma matriz triangular inferior ("lower triangular") e U é uma matriz triangular superior ("upper triangular") [1].

Voltando ao problema, é especialmente interessante encontrar uma decomposição LU. Pois se escrevemos o sistema na forma $LUs = t$, então $Us = L^{-1}t$ e podemos encontrar s da seguinte forma:

1. Resolvendo o sistema $Ly = t$.
2. Resolvendo o sistema $Us = y$.

Note que desde que os elementos das diagonais sejam não nulos, estes sistemas são solúveis pois são sistemas triangulares.

A maior vantagem computacional da decomposição LU ocorre quando se tem de resolver um grande número de equações com a mesma matriz M e com muitos vetores s. Uma vez dispondo dessa decomposição não é preciso repetir muitas vezes o processo de eliminação gaussiana que leva da ordem de $\frac{1}{3}n^3$ operações. Já o processo (1) e (2) levam da ordem de n^2 operações. Para mais detalhes ver [12]

2.2.2 A decomposição QR

Da Álgebra Linear, conhecemos o processo de Gram-Schmidt, o qual é útil quando é dado um conjunto de vetores linearmente independente $\{v_1, \ldots, v_n\}$ e queremos determinar um conjunto ortonormal de vetores $\{u_1, \ldots, u_n\}$ que gerem o mesmo subespaço, isto é:

$$\mathrm{span}\{v_1, \ldots, v_n\} = \mathrm{span}\{u_1, \ldots, u_n\}$$

[1] Mais geralmente, pode ser necessário utilizar uma matriz de permutação P para obtermos uma decomposição $PM = LU$.

Com este método temos ainda que,

$$\text{span}\{v_1, \ldots, v_k\} = \text{span}\{u_1, \ldots, u_k\}, \quad (k = 1, \ldots, n)$$

Consideremos uma base $\{v_1, \ldots, v_n\}$ do espaço vetorial V:

1. Seja $w_1 = v_1$.
2. Definamos w_2 por:
$$w_2 = v_2 - \frac{< v_2, w_1 >}{< w_1, w_1 >} w_1$$
Notemos que $\frac{< v_2, w_1 >}{< w_1, w_1 >} w_1$ é o vetor projeção ortogonal do vetor v_2 sobre o vetor w_1.
3. Em geral:
$$w_n = v_n - \frac{< v_n, w_{n-1} >}{< w_{n-1}, w_{n-1} >} w_{n-1} - \ldots \cdots - \frac{< v_n, w_1 >}{< w_1, w_1 >} w_1$$
4. Para obtermos uma base ortonormal basta normalizarmos os vetores w_i, com i variando de $1, \ldots, n$, i. e.,
$$u_1 = \frac{w_1}{\|w_1\|}, \ldots, u_n = \frac{w_n}{\|w_n\|}$$

O processo acima é chamado de processo de ortogonalização de Gram-Schmidt.

Agora precisamos recuperar os vetores originais $\{v_1, \ldots, v_n\}$ como combinações lineares na base ortonormal encontrada pelo algoritmo.

$$u_1 = \frac{w_1}{\|w_1\|} = \frac{v_1}{\|v_1\|} \longrightarrow v_1 = \|v_1\| u_1,$$

e:

$$u_2 = \frac{v_2 - \frac{< v_2, w_1 >}{< w_1, w_1 >} w_1}{\|w_2\|} \longrightarrow v_2 = \|w_2\| u_2 + \frac{< v_2, w_1 >}{\|w_1\|} u_1$$

e, portanto, v_n é a combinação linear:

$$v_n = \|w_n\| u_n + \frac{< v_n, w_{n-1} >}{\|w_{n-1}\|} u_{n-1} + \cdots + \frac{< v_n, w_1 >}{\|w_1\|} u_1.$$

Assim,

$$\begin{pmatrix} v_1 & \dots & v_n \end{pmatrix} = \begin{pmatrix} u_1 & \dots & u_n \end{pmatrix} \begin{pmatrix} \|w_1\| & \frac{< v_2, w_1 >}{\|w_1\|} & \dots \\ \vdots & \ddots & \vdots \\ 0 & \dots & \|w_n\| \end{pmatrix}$$

Logo, qualquer matriz A com colunas linearmente independentes pode ser fatorada no produto $A = QR$, onde as colunas de Q são ortonormais e R é triangular superior inversível. Se A é quadrada, então Q e R também o são, e Q torna-se uma matriz ortogonal.

Algoritmo 1 Decomposição QR

{M é uma matriz $m \times n$}
for $k = 1$ to n **do**
  $s_{ik} \leftarrow v_k^T w_i$ $\{i = 1, \dots, k-1\}$
  $z_k \leftarrow v_k - \sum_{i=1}^{k-1} s_{ik} w_i$
  $r_{kk} \leftarrow \left(z_k^T z_k\right)^{1/2}$
  $q_k \leftarrow z_k / r_{kk}$
  $r_{ik} \leftarrow s_{ik} / r_{kk}$
end for
{Na *k-ésima* etapa do algoritmo são geradas as *k-ésimas* colunas de Q e de R}

É muito vantajoso trabalhar com matrizes ortogonais, já que para invertê-las basta calcular a transposta. E também algoritmicamente falando (erros devidos a ponto flutuante) estes algoritmos são mais estáveis que a decomposição LU.

No caso do sistema ser inconsistente com $\text{posto}(M) = r$ onde n é o número de colunas, o problema dos mínimos quadrados torna-se:

$$s = (M^{\mathrm{T}}M)^{-1}M^{\mathrm{T}}t = (R^{\mathrm{T}}Q^{\mathrm{T}}QR)^{-1}R^{\mathrm{T}}Q^{\mathrm{T}}t = R^{-1}(R^{\mathrm{T}})^{-1}R^{\mathrm{T}}Q^{\mathrm{T}}t = R^{-1}Q^{\mathrm{T}}t$$

e assim basta calcular a multiplicação $Q^{\mathrm{T}}t$ e depois encontrar a solução do sistema (se necessário no sentido de mínimos quadrados):

$$Rs = Q^{\mathrm{T}}t,$$

o que é simples.

Esse é um excelente método para resolver sistemas de equações lineares, apesar da eliminação gaussiana ser o mais utilizado na prática, pois utiliza somente a metade das operações numéricas necessárias na decomposição QR. Para mais detalhes ver [12].

Apesar do método QR ser mais custoso computacionalmente, no problema inverso é mais interessante do que a Eliminação Gaussiana por ser mais estável, pois utiliza multiplicações por matrizes ortogonais.

2.3 Métodos iterativos

Um método iterativo é um processo em que se obtém uma sequência de soluções aproximadas do problema, tal que cada termo da sequência $\{x_n\}_{n=1}^{\infty}$ é obtido a partir dos anteriores. Além disso, para n suficientemente grande, estamos cada vez mais próximos da solução exata do problema.

De um modo geral, em um método iterativo, decompomos a matriz M como diferença de outras duas matrizes, de maneira que o problema original $Ms = t$ é reescrito na forma equivalente,

$$Ps = Qs + t,$$

onde $M = P - Q$.

Assim, podemos obter um processo iterativo da seguinte forma: escolhemos um vetor inicial s_0 e iteramos

$$s_k = Gs_{k-1} + c$$

onde $G = P^{-1}Q$ e $c = P^{-1}t$.

Vamos agora estabelecer critérios para a convergência do método iterativo.

Teorema 4. *O algoritmo $s_{k+1} = Gs_k + c$ converge para uma solução de $Ms = t$, qualquer que seja o vetor inicial s_0, se todos os autovalores de G têm módulo menor que* 1.

Demonstração Temos que $s_k = G^k s_0 + (I - G^k)(I - G)^{-1}c$, assim,

$$\lim_{k\to\infty} s_k = (I - G)^{-1}c$$

pois, temos que $\lim_{k\to\infty} G^k s_0 = 0$, pois por hipótese G tem seus autovalores com módulo menor do que 1.

Concluímos então que $\lim_{k\to\infty} s_k$ existe e chamemos de s esse limite. Então s satisfaz a equação $s = Gs+c$, a qual é equivalente a $Ms = t$, logo s é a solução do sistema.

Seja $A \in \mathbb{R}^{n\times n}$ uma matriz que não possui elementos nulos na diagonal. É claro que podemos transformar essa matriz numa soma de uma matriz P diagonal com uma matriz Q, onde P possui a diagonal de A e Q possui os outros elementos de A, porém com diagonal nula. Assim, escrevemos a matriz $A = P - Q$, com P invertível. Chamamos este método iterativo de método de Jacobi. Para o método de Gauss-Seidel, basta considerar a seguinte decomposição de $A = P - Q$, onde a matriz P é uma matriz triangular inferior composta pelos elementos de A, inclusive os de sua diagonal, e Q é matriz triangular superior com diagonal nula.

Nos métodos iterativos sempre necessitamos de um ponto inicial para começar os algoritmos. Na tomografia é usual estabelecer esse ponto através de fórmulas de retroprojeção.

A retroprojeção é um processo muito simples porém nos dá uma solução distante da original. Para se ter uma idéia intuitiva do que acontece, considere a média da vagarosidade ao longo do i-ésimo raio:

$$< s >_i = \frac{t_i}{L_i}$$

onde t_i é o tempo de percurso do i-ésimo raio e L_i é o comprimento desse mesmo raio entre fonte e receptor.

Uma primeira aproximação para o valor constante s_j (valor da vagarosidade na j-ésima célula) seria a média de todos os valores $< s >_i$ dos caminhos que passam pela célula j. Assim obtemos:

$$s_j = \frac{1}{\sum sign(l_{ij})} \sum_{i=1}^{n} sign(l_{ij}) \frac{t_i}{L_i}$$

onde $sign(l_{ij})$ vale 0 caso l_{ij} seja 0 e vale 1, caso contrário.

2.3.1 Iteração simples

O que chamaremos de iteração simples é um caso particular de uma iteração de Richardson que é dada por

$$x^{k+1} = x^k + f(x^k).$$

Na iteração simples consideraremos [2]

$$s^{(k+1)} = s^{(k)} + M^{\mathrm{T}}(t - Ms^{(k)}).$$

Seja $r = \mathrm{posto}(M)$, para que analisemos que tipo de condições precisamos para que a sequência de iterações convirja, começamos por reconhecer que na equação $Ms = t$, os vetores s e t podem ser expandidos em termos dos vetores singulares à esquerda e à direita de M.

$$t = \sum_{i=1}^{r} \tau_i y_i + t_0.$$

e

$$s = \sum_{i=1}^{r} \sigma_i z_i + s_0.$$

[2] Esta iteração está associada ao método de Landweber no contexto não linear.

onde $z_i^{\mathrm{T}} s_0 = y_i^{\mathrm{T}} t_0 = 0$ e $\tau_i = y_i^{\mathrm{T}} t$, $\sigma_i = z_i^{\mathrm{T}} s$, $\forall i = 1, \ldots, r$.

Assim, a sequência de iterações acima é equivalente à

$$\begin{aligned} \sigma_i^{k+1} &= \sigma_i^k + \lambda_i(\tau_i - \lambda_i \sigma_i^k) \\ &= \lambda_i \tau_i + (1 - \lambda_i^2)\sigma_i^k \\ &= \lambda_i \tau_i + (1 - \lambda_i^2)[\lambda_i \tau_i + (1 - \lambda_i^2)\sigma_i^{k-1}] \end{aligned}$$

Então, obtemos a seguinte série:

$$\begin{aligned} \sigma_i^{k+1} &= [1 + (1 - \lambda_i^2) + (1 - \lambda_i^2)^2 + \cdots + (1 - \lambda_i^2)^k]\lambda_i \tau_i + (1 - \lambda_i^2)\sigma_i^0 \\ &= \frac{1 - (1 - \lambda_i^2)^{k+1}}{1 - (1 - \lambda_i^2)} \lambda_i \tau_i + (1 - \lambda_i^2)\sigma_i^0 \\ &= \frac{1 - (1 - \lambda_i^2)^{k+1}}{\lambda_i} \sigma_i + (1 - \lambda_i^2)\sigma_i^0. \end{aligned}$$

Supondo que $-\sqrt{2} < \lambda_i < \sqrt{2}$, então a sequência converge. Note que a sequência converge para a solução de $Ms = t$ no sentido dos mínimos quadrados. A iteração simples é um método bem mais simples de implementar que os gradientes conjugados e por isso é algumas vezes utilizado para resolver problemas de tomografia linear.

2.3.2 Gradientes Conjugados

Vamos assumir nesta seção que A é simétrica e positiva definida, isto é,

$$A^{\mathrm{T}} = A \tag{2.3}$$

e

$$x^{\mathrm{T}} A x > 0, \quad para \quad x \neq 0. \tag{2.4}$$

Note que A definida como acima é invertível.

Lema 1. *Seja A simétrica positiva definida, então solucionar $As = b$ é equivalente à minimizar a forma quadrática $q(s) = s^{\mathrm{T}} A s - 2 s^{\mathrm{T}} b$.*

Para a demonstração do lema basta observar que se A satisfaz (2.3) e (2.4) temos que q é convexa e assim minimizá-la é equivalente ao gradiente ser nulo no único ponto de mínimo existente. Acontece que a solução desse problema é o mesmo que encontrar a solução de $As = b$.

Iremos minimizar q ao longo de uma sequência de retas, sempre escolhendo uma direção $v^{(k)}$ conveniente.

$$s^{(k+1)} = s^{(k)} + t_k v^{(k)}$$

onde,

$$t_k = \text{argmin}_\tau q(s^{(k)} + \tau v^{(k)})$$

No método da descida rápida escolhemos $v^{(k)}$ como o gradiente negativo de q em $s^{(k)}$, isso nos leva a que esse gradiente aponte na direção do residual $r_k = b - As^{(k)}$. Esse método é raramente utilizado, pois pode convergir muito lentamente.

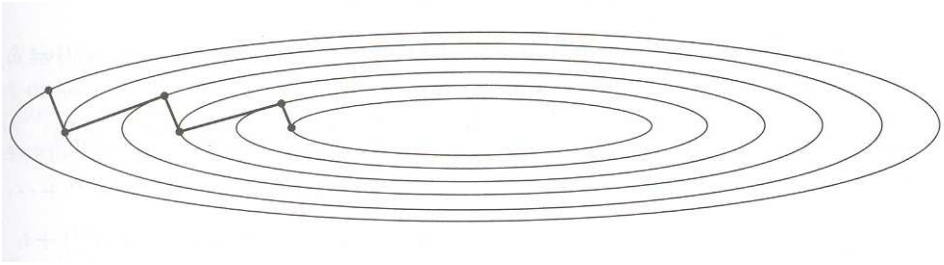

Figura 2.1: O método da descida rápida

A estratégia no método das direções conjugadas é escolher vetores A-ortonormais, ou seja, vetores $u^{(1)}, \ldots, u^{(n)}$ tais que

$$(u^{(i)})^{\mathrm{T}} A u^{(j)} = 0$$

se $i \neq j$. Esses vetores também são conhecidos como direções A-conjugadas.

Para gerar direções A-conjugadas considere dados $\{\xi_0, \ldots, \xi_k\} \in \mathbb{R}^n$ conjunto de vetores linearmente independentes. Pelo processo de Gram-Schmidt podemos construir vetores ortogonais $v^{(k)}$ tais que:

$$\text{span}\{\xi_0, \ldots, \xi_i\} = \text{span}\{v^{(0)}, \ldots, v^{(i)}\}$$

para todo $i \leqslant k$.

Assim, seja $v^{(0)} = \xi_0$ e suponha já obtidos $v^{(0)}, \ldots, v^{(i)}$ A-conjugados, $i < k$. Utilizando Gram-Schmidt, temos:

$$v^{(i+1)} = \xi^{i+1} + \sum_{j=0}^{i} \gamma_{i+1,j} v^{(j)}$$

e escolhemos $\gamma_{i+1,j}$ tal que $(v^{(i+1)})^{\mathrm{T}} A v^{(m)} = 0$, para $0 \leqslant m \leqslant i$.

Para isto,

$$0 = (v^{(i+1)})^{\mathrm{T}} A v^{(m)} = (\xi^{i+1})^{\mathrm{T}} A v^{(m)} + \sum_{j=0}^{i} \gamma_{i+1,j} (v^{(j)})^{\mathrm{T}} A v^{(m)}$$

Por hipótese temos que $\sum_{j=0}^{i} \gamma_{i+1,j} (v^{(j)})^{\mathrm{T}} A v^{(m)} = \gamma_{i+1,m} (v^{(m)})^{\mathrm{T}} A v^{(m)}$ e, assim,

$$\gamma_{i+1,m} = -\frac{(\xi^{i+1})^{\mathrm{T}} A v^{(m)}}{(v^{(m)})^{\mathrm{T}} A v^{(m)}}.$$

para cada $0 \leqslant m \leqslant i$.

O principal resultado sobre o método das direções conjugadas é a propriedade de que em até n iterações obteremos a solução do problema.

Teorema 5. *A solução do problema de minimizar a função quadrática q sujeita a condição $s \in M_k$ é $s^{(k+1)}$, onde*

$$M_k = \{s; \quad s = s^{(0)} + v, com \quad v \in \operatorname{span}\{v^{(0)}, \ldots, v^{(k)}\}\}$$

Note que $M_{n-1} = \mathbb{R}^n$ e por isso temos a solução em no máximo n iterações.

Demonstração Seja $s^{(k+1)} = s^{(k)} + t_k v^{(k)}$, onde t_k é tal que

$$q(s^{(k)} + t_k v^{(k)}) = \min_{\tau} q(s^{(k)} + \tau v^{(k)}).$$

Assim, se $\phi_i(\tau) = q(s^{(i)} + \tau v^{(i)})$, temos

$$0 = \phi_i'(t_i) = \nabla q(s^{(i)} + t_i v^{(i)})^{\mathrm{T}} v^{(i)} = \nabla q(s^{(i+1)})^{\mathrm{T}} v^{(i)}$$

Note que para $i \leqslant k$.

$$\begin{aligned}
\nabla q(s^{(k+1)})^{\mathrm{T}} v^{(i)} &= (A(s^{(k+1)}) - b)^{\mathrm{T}} v^{(i)} = (A(s^{i+1} + \sum_{j=i+1}^{k} t_j v^{(j)}) - b)^{\mathrm{T}} v^{(i)} \\
&= (s^{(i+1)} + \sum_{j=i+1}^{k} \gamma_j (v^{(j)})^{\mathrm{T}} A v^{(i)} - b^{\mathrm{T}} v^{(i)} \\
&= (s^{(i+1)})^{\mathrm{T}} A v^{(i)} - b^{\mathrm{T}} v^{(i)} = \nabla q(s^{(i+1)})^{\mathrm{T}} v^{(i)} = 0
\end{aligned}$$

Minimizar $q(s)$ com $s \in M_k$ é equivalente a encontrar $\gamma = (\gamma_1, \ldots, \gamma_k) \in \mathbb{R}^k$ tal que $q(s^{(0)} + \sum_{i=1}^{k} \gamma_i v^{(i)})$ seja o menor possível. Como q é convexa, sabemos que possui único mínimo e que é dado pelo ponto onde o gradiente se anula.

Pelo que vimos acima $\gamma_i = t_i$. Logo, o mínimo de q em M_k é $s^{(k+1)}$.

Feito isto, trataremos agora do método que dá título a esta seção, o método dos gradientes conjugados. Aqui, consideramos o conjunto linearmente independente $\{\xi_0, \ldots, \xi_k\}$, que é construído ao longo das iterações e é composto pelos gradientes de q em pontos a determinar.

Algoritmo 2 Método dos Gradientes Conjugados

$s^{(0)} \leftarrow 0$
for $k = 1$ to n **do**
  $r_{k-1} \leftarrow As^{(k-1)} - b$
  if $r_{k-1} = 0$ **then**
    set $s = s^{(k-1)}$ and quit
  else
    if $k = 1$ **then**
      $v^{(k)} = r_o$
    else
      $\alpha_k = \left(v^{(k-1)}\right)^T r_{k-1} / \left(v^{(k-1)}\right)^T Av^{(k-1)}$
      $s^{(k)} = s^{(k-1)} - \alpha_k v^{(k)}$
      $v^{(k)} = -r_k + \frac{r_k^T Av^{(k)}}{\left(v^{(k)}\right)^T Av^{(k)}} v^{(k)}$
    end if
  end if
end for

Iremos considerar o problema $Ms = t$ no sentido dos mínimos quadrados, $As = b$ onde $A = M^{\mathrm{T}}M$ e $b = M^{\mathrm{T}}t$.

Assim queremos encontrar t_k que minimize a forma quadrática na direção $s^{(k+1)} = s^{(k)} + t_k v^{(k)}$, ou seja, tal que $q(s^{(k)} + t_k v^{(k)}) \leqslant q(s^{(k)})$.

Para isto, considere então

$$r_k = As^{(k)} - b$$

e

$$h(\tau) = q(s^{(k)} + t_k v^{(k)}) - q(s^{(k)}).$$

Utilizando a forma quadrática q, obtemos que:

$$q(\tau) = 2\tau < v^{(k)}, r_k > + \tau^2 < v^{(k)}, Av^{(k)} >= 0$$

ou seja,

$$t_k = \tau = -\frac{< r_k, v^{(k)} >}{< v^{(k)}, Av^{(k)} >}$$

e como,

$$r_{k+1} = r_k + t_k A v^{(k)}$$

temos

$$< r_{k+1}, v^{(k)} >= 0.$$

Resumindo, o método dos gradientes conjugados segue o seguinte esquema:

$$\begin{aligned} s^{(k+1)} &= s^{(k)} - \frac{(v^{(k)})^{\mathrm{T}} r_k}{(v^{(k)})^{\mathrm{T}} A v^{(k)}} v^{(k)} \\ r_{k+1} &= A s^{(k+1)} - b \\ v^{(k+1)} &= -r_{k+1} + \frac{r_{k+1}^{\mathrm{T}} A v^{(k)}}{(v^{(k)})^{\mathrm{T}} A v^{(k)}} v^{(k)} \end{aligned}$$

2.3.3 SOR (Successive Over-Relaxation)

Escrevendo o sistema $Ms = t$ de maneira equivalente como,

$$Qs = (Q - M)s + t$$

e, iterando da seguinte forma:

$$s^{(k)} = (I - Q^{-1}M)s^{(k-1)} + Q^{-1}t = Gs^{(k-1)} + c$$

onde $G = I - Q^{-1}M$ e $c = Q^{-1}t$.

O método SOR pode ser caracterizado com a escolha:

$$\begin{aligned} &Q = \omega^{-1}(D + \omega L), \\ &G = (D + \omega L)^{-1}(-\omega U + (1 - \omega)D). \end{aligned}$$

onde $M = D + L + U$, com D matriz diagonal formada pelos elementos da diagonal de M, L a matriz triangular inferior com diagonal nula e U matriz triangular superior dos elementos de M com diagonal nula.

Na prática, este método é um passo a mais no método de Gauss-Seidel. Assim se considerarmos $\tilde{x}_i^{(k)}$ como a k-ésima iterada de Gauss-Seidel, podemos calcular a k-ésima iterada como

$$x_i^{(k)} = x_i^{(k-1)} + \omega \left(\tilde{x}_i^{(k)} - x_i^{(k-1)} \right), \tag{2.5}$$

onde w é um parâmetro de relaxação. Observe que se $w = 1$ então retornamos para a iterada de Gauss-Seidel.

Os seguintes teoremas apresentam resultados sobre a convergência do método SOR.

Teorema 6. *(Kahan - condição necessária) Para que haja convergência do método SOR, qualquer que seja a iterada inicial, é necessário que ω esteja no intervalo $(0, 2)$.*

Para uma prova ver [17].

Teorema 7. *(Ostrowski-Reich - condição suficiente) Se a matriz A for simétrica e definida positiva, e considerarmos ω no intervalo $(0, 2)$, há convergência do método SOR, qualquer que seja a iterada inicial.*

Para uma prova ver [22] e [25].

2.3.4 ART (Algebraic Reconstruction Technique)

Considere o problema $As = b$, onde $A = M^{\mathrm{T}}M$ e $b = M^{\mathrm{T}}t$ e sejam a_i as linhas da matriz A. A sequência de iterações desse método de reconstrução é dada por:

$$s^{(i)} = Q_i s^{(i-1)} + \frac{b_i}{a_i^{\mathrm{T}} a_i} a_i \quad para \quad 1 \leqslant i \leqslant n. \tag{2.6}$$

onde Q_i é o operador projeção no complemento ortogonal do vetor a_i, ou seja,

$$Q_i = I - \frac{a_i a_i^{\mathrm{T}}}{a_i^{\mathrm{T}} a_i}$$

Algoritmo 3 ART - $\texttt{ART}(M, t, tol)$

Input $M,\ t,\ tol$
{M é uma matriz $m \times n$}
{$A = M^T M$ e $b = M^T t$ }
$s^{(0)} = 0$
$teste = 0$
while $(teste = 0)$ **do**
  $sold \leftarrow s$
  for $i = 1$ to n **do**
    $s^{(i)} \leftarrow Q_i s^{(i-1)} + \frac{b_i}{a_i^T a_i} a_i$
  end for
  if $\|s - sold\| \leq tol$ **then**
    $teste \leftarrow 1$
  end if
end while

Um ciclo do ART é dado quando percorremos todas as linhas da matriz A. Para simplificar a notação, quando mudamos de ciclo, o qual tem n passos,

podemos reescrever

$$s^{(i)} = Q_{i'} s^{(i-1)} + \frac{b_{i'}}{a_{i'}^{\mathrm{T}} a_{i'}} a_{i'} \quad para \quad 1 \leqslant i' \leqslant n, \quad i \geqslant 1$$

O número de iteração ou o ciclo em que estamos é dado por $k = [\frac{i}{n}]$.

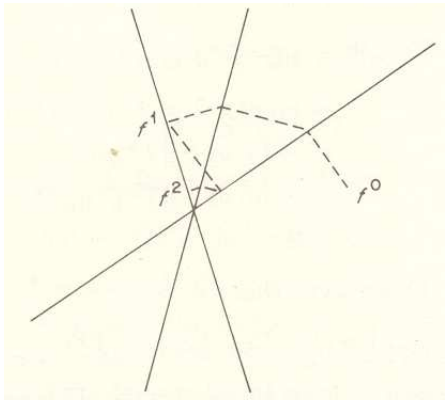

Figura 2.2: O Método de ART para três equações em $\mathbb{R}^2$, onde $f^k = s^{(k)}$

Sabemos que as soluções de um sistema linear não homogêneo estão na interseção das variedades afins lineares formadas por cada linha do sistema. Em cada ciclo, este algoritmo projeta em cada uma dessas variedades lineares afins. De fato, tomemos s_0 qualquer para iniciar o algoritmo, o próximo termo da iteração s_1 está na variedade afim formada pela primeira linha do sistema, pois a solução do sistema $(a_1)^{\mathrm{T}} s = b_1$ é dada pela solução geral do sistema homogêneo associado mais uma solução particular do sistema. Em (2.6) a primeira parcela à direita é uma solução homogênea e a segunda parcela é solução particular. Logo s_1 é solução de $(a_1)^{\mathrm{T}} s = b_1$, o que significa que está nessa variedade linear afim.

Agora, vamos dar condições para que o algoritmo convirja para uma solução. Para isto, observe que

$$\begin{aligned} s^{(n)} &= Q_n s^{(n-1)} + \frac{b_n}{a_n^{\mathrm{T}} a_n} a_n \\ &= Q_n (Q_{n-1} s^{(n-2)}) + \frac{b_{n-1}}{a_{n-1}^{\mathrm{T}} a_{n-1}} a_{n-1}) + \frac{b_n}{a_n^{\mathrm{T}} a_n} a_n \\ &= Q_n Q_{n-1} s^{(n-2)} + Q_n \frac{b_{n-1}}{a_{n-1}^{\mathrm{T}} a_{n-1}} a_{n-1} + \frac{b_n}{a_n^{\mathrm{T}} a_n} a_n \end{aligned}$$

Assim, é fácil ver que $s^{(n)}$ pode ser reescrito como:

$$\begin{aligned} s^{(n)} &= Q_n \dots Q_1 s^{(0)} + b_1 Q_n \dots Q_2 \frac{a_n}{(a_n)^{\mathrm{T}} a_n} + \dots b_{n-1} Q_n \frac{a_{n-1}}{a_{n-1}^{\mathrm{T}} a_{n-1}} + \frac{b_n}{(a_n)^{\mathrm{T}} a_n} a_n \\ &= Q s^{(0)} + Rb \end{aligned}$$

onde $Q = Q_n \dots Q_1$ e $R = (\frac{Q_n \dots Q_1 a_1}{a_1^{\mathrm{T}} a_1}, \dots, \frac{1}{a_n^{\mathrm{T}} a_n} a_n)$.

É simples obter $s^{(kn)}$ utilizando que $Qs^{(0)} + Rb$,

$$s^{(kn)} = \sum_{p=0}^{k-1} Q^p Rb + Q^k s^{(0)}$$

onde k é o número do ciclo.

Então se

$$\lim_{k \to \infty} \sum_{p=0}^{k-1} Q^p R = A^{\dagger}$$

e

$$\lim_{k \to \infty} \sum_{p=0}^{k-1} Q^k s_0 = 0 \quad ou \quad v,$$

onde $v \in N(A)$, o método converge para a solução aproximada $A^{\dagger} b$. Como visto antes, esta é a solução ótima com a norma de Frobenius.

Foi observado que o método de ART pode não convergir caso A seja singular ou os dados sejam inconsistentes. Para mais detalhes ver [3, 9, 11, 13].

Capítulo 3

Implementação dos Métodos

Neste capítulo implementaremos alguns dos métodos estudados no capítulo anterior. Mais especificamente, daremos um conjunto de exemplos e estudaremos como cada método escolhido se comporta. Estudaremos sua eficiência para matrizes encontradas em experimentos geofísicos. Faremos também uma comparação entre os tempos de processamento de cada um.

Salientamos que o método de decomposição QR não é muito útil no caso de matrizes de grande porte. Esse tipo de decomposição quando feito com uma matriz esparsa com um número muito grande de coeficientes nos dá uma matriz densa Q para a qual não é possível o armazenamento na máquina uma vez que ultrapassa o limite de memória.

Em nossos testes, a parte computacionalmente mais custosa é a de resolver o problema direto de encontrar o percurso das ondas geradas para o imageamento. Para isto, tem-se de escolher o melhor algoritmo de traçado de caminhos para o modelo adotado. Neste trabalho, o modelo adotado é o de células de vagarosidade constante, sendo assim, um algoritmo vantajoso para o traçado é o método de "bending", ou método de Prothero, Taylor e Eickemeyer [5].

O método de "bending"de maneira simplificada consiste em perturbar raios retilíneos que ligam um par de fonte e receptor fixos. Isto é feito de forma que consigamos obter uma boa aproximação para os percursos dos raios e consequentemente a matriz $M = M_{rs}$, que indica para cada percurso r, o comprimento do mesmo dentro da célula s.

Assim, primeiramente, consideremos a reta que liga os pontos $(0, y_i)$ e (L, y_f), isto é:

$$y_0(x) = y_i(1 - \frac{x}{L}) + y_f\frac{x}{L}$$

e a perturbamos com uma função $y_\delta(x)$ dada por:

$$y_\delta(x) = \sum_{j=1}^{k} a_j \sin\left(\frac{j\pi x}{L}\right)$$

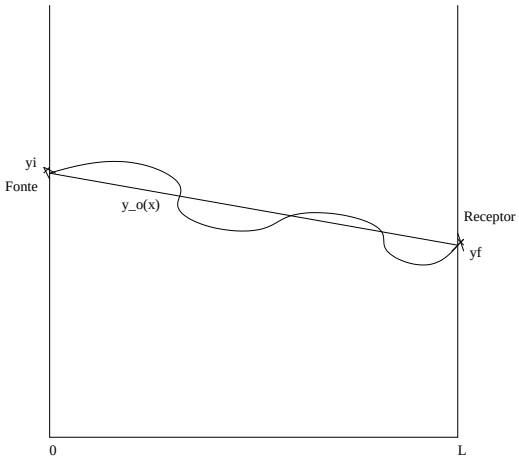

Figura 3.1: Método de "bending"

Observe que a perturbação $y_\delta(x)$ foi escolhida assim dentre outros motivos por manter fixos os pontos iniciais e finais dados.

Sabemos que se P é o percurso ligando a fonte y_i ao receptor y_f então

$$t = \int_P s(x) dl^P$$

é o tempo de percurso da onda entre y_i e y_f , onde dl^P denota a distância infinitesimal ao longo do caminho P. Lembrando que o percurso físico do raio é dado por um mínimo do tempo de percurso quando P percorre todos os possíveis caminhos que ligam a fonte ao receptor, a idéia é minimizar na variável $a \in \mathbb{R}^k$ a função $t = t(a)$. Escrevendo o tempo de percurso em termos de $y(x)$, temos

$$t(a) = \int_0^L s(x, y(x)) \sqrt{1 + \left(\frac{dy}{dx}\right)^2} dx$$

De posse desse vetor $a \in \mathbb{R}^k$, obtemos o caminho aproximado ao caminho de Fermat que liga a fonte ao receptor escolhidos.

Neste trabalho, a matriz M calculada no problema direto será utilizada na solução do problema inverso. Em situações mais realísticas, como por exemplo

quando dispomos somente de dados reais, a matriz M é determinada através de um processo iterativo onde diversas aproximações são feitas do modelo subjacente.

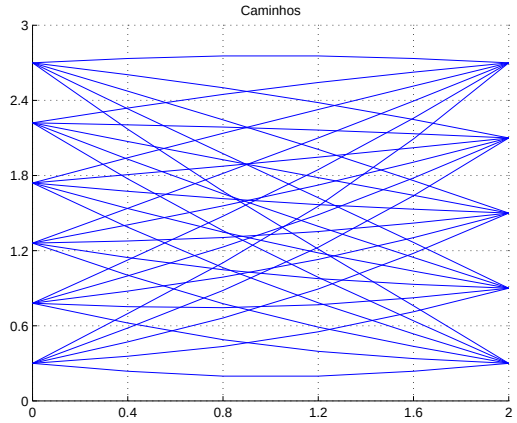

Figura 3.2: Um exemplo pequeno de percursos ótimos entre fontes e receptores. Aqui, a distância entre os poços é de duas unidades de medida e a profundidade dos poços é de três unidades de medida.

Assim, utilizamos o programa *qnewton* (gentilmente cedido por Dayse H. Pastore) de minimização com busca linear para dentre todos os percursos possíveis entre y_i e y_f, obtermos o de menor tempo de percurso. Para passar as informações sobre o que queremos minimizar criamos a subrotina *bend*.

O programa *matrix* se ocupa de duas coisas, construir o gráfico de todos os caminhos que ligam cada fonte a cada receptor e construir a matriz M de comprimento de percursos em cada pixel.

Apesar de construir a matriz M, não faz sentido guardá-la por causa do grande porte dos problemas. Então tivemos um esforço adicional para programar os algoritmos utilizando M como uma função.

Para ordenar os pixels utilizamos o padrão de cima para baixo e da direita para esquerda como mostra a Figura 3.3. Denotamos $g2$ a quantidade de partições no eixo da profundidade e g a quantidade de partições no eixo da distância entre os poços. Assim possuímos um total de $g * g2$ pixels.

O programa *comprimento* é subrotina do programa *matrix*. Este programa utiliza essencialmente a fórmula de distância entre dois pontos, fazendo assim uma aproximação dos comprimentos em cada pixel.

Para ordenar os caminhos fixamos cada fonte, sempre de cima para baixo e a ligamos com todos os recptores de cima para baixo. Assim a quantidade de

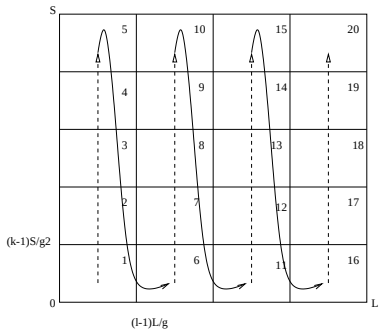

Figura 3.3: Forma utilizada no trabalho para ordenar os pixels.

caminhos é o produto da quantidade de fontes pela quantidade de receptores.

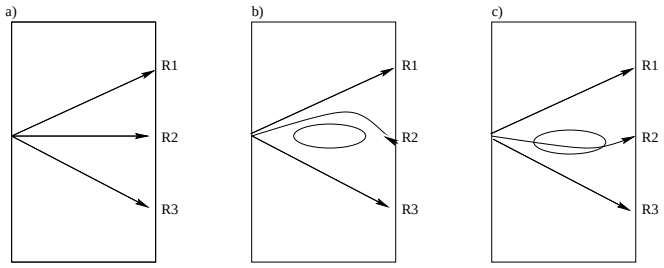

Figura 3.4: a) Meio homogêneo, com isto os raios que ligam fontes a receptores são linhas retas; b) Meio com região de baixa velocidade, o que faz com que os raios contornem essa região; c) Meio com região de baixa velocidade.

É importante salientar que em todos os exemplos que daremos a seguir, existe uma área de cobertura aonde a reconstrução é mais bem sucedida, pois esta região mais centrada é naturalmente mais iluminada por raios do que as regiões mais periféricas. Isto é mostrado na Figura 3.2, por exemplo. Esta área é também conhecida como região de interesse.

Implementamos o algoritmo ART, o qual nos exemplos mais adiante foi mais eficiente no tempo de execução do que o outro método utilizado nos exemplos, os gradientes conjugados. Também implementamos o método QR utilizando a matriz M. Isto como já dissemos utiliza muita memória para problemas de grande porte, o que o torna inviável para matrizes muito grandes.

Utilizamos o pacote do Matlab de gradientes conjugados. Devido ao fato que os problemas em questão são mal-postos no sentido de que o condicionamento dos mesmos é muito alto (podendo até ser infinito), se faz necessário a

utilização de métodos de regularização. Para tal, no caso de método de gradientes conjugados buscamos a solução não das equações normais

$$M'Mx = M't,$$

mas sim de

$$(M'M + \eta I)x = M't,$$

onde η é o parâmetro de regularização. Isto tem duas vantagens: regularização do problema e tornar a equação que queremos resolver simétrica, e portanto tratável pelos algoritmos clássicos de gradientes conjugados.

Com o objetivo de testar o desempenho dos métodos de solução do problema inverso, introduzimos ruídos nos dados. Isto é feito, adicionando-se ao vetor t um vetor αn, onde $\alpha > 0$ denota o nível de ruído (tipicamente 0.001, 0.01) e n é uma variável pseudo-aleatória uniformemente distribuída no intervalo $[0, 1]$.

Para regularizar o método ART, optamos por fazer poucas iterações, onde a vagarosidade inicial é obtida por uma fórmula de retroprojeção.

O conjunto de exemplos a seguir foi realizado com 35×35 pixels e 40×40 pares de fontes e receptores. Fizemos em cada exemplo um experimento sem ruído e outro com ruído de 1%. Nas tabelas comparamos erros relativos na região de interesse, utilizando cada método.

Exemplo 1

Neste exemplo, supomos que o meio estudado possui alguma formação geológica no centro deste.

Ruído	Erro Relativo		
	ART	CG	QR
0%	0.0879	0.0452	0.0843
1%	0.0597	0.0425	0.1776

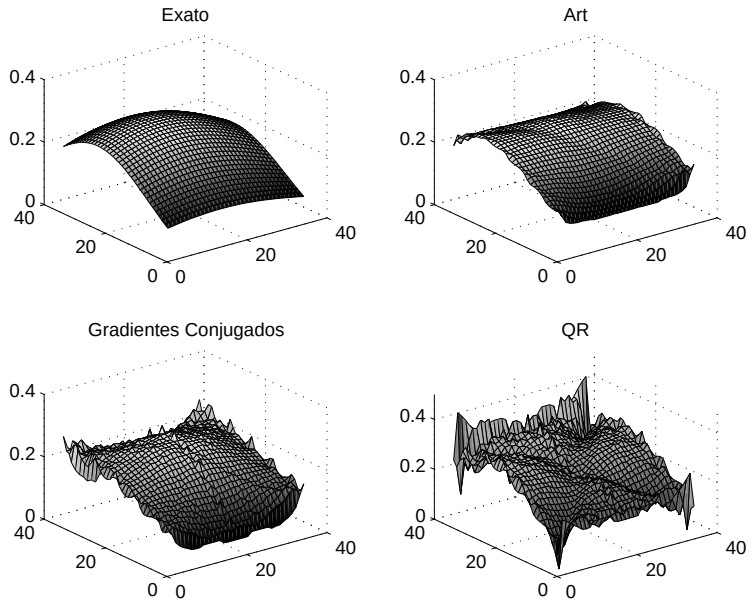

Figura 3.5: A vagarosidade em 3-D sem ruído

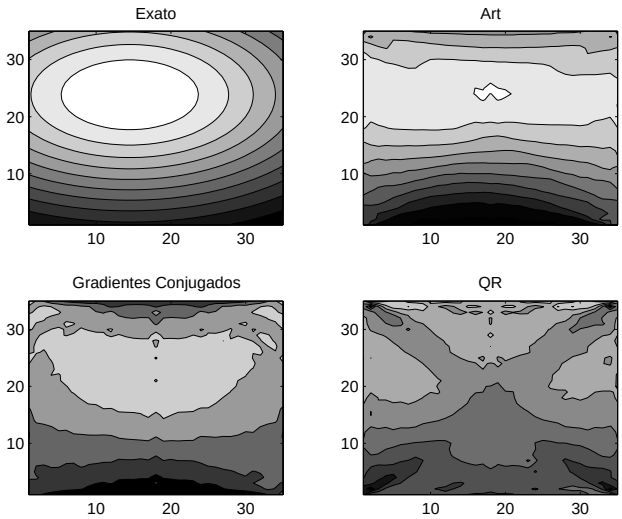

Figura 3.6: Curvas de nível na região de interesse sem ruído

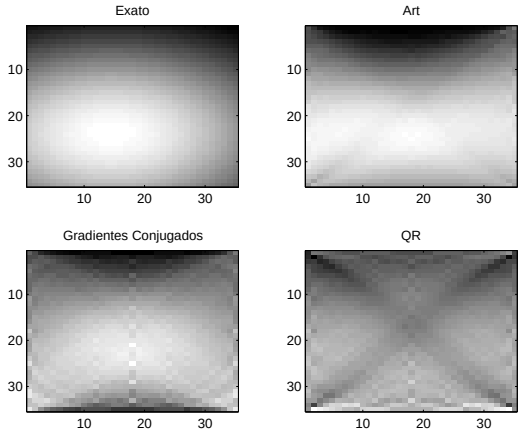

Figura 3.7: Intensidade da vagarosidade na região de interesse sem ruído

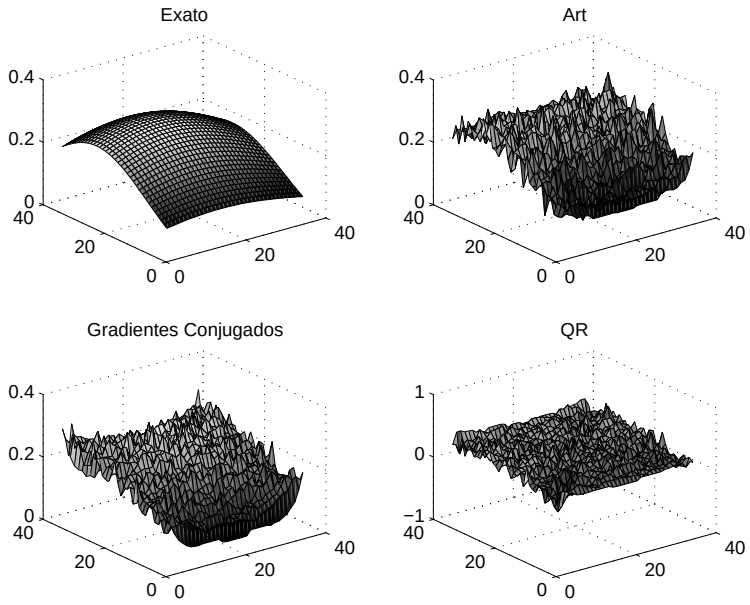

Figura 3.8: A vagarosidade em 3-D com 1% de ruído

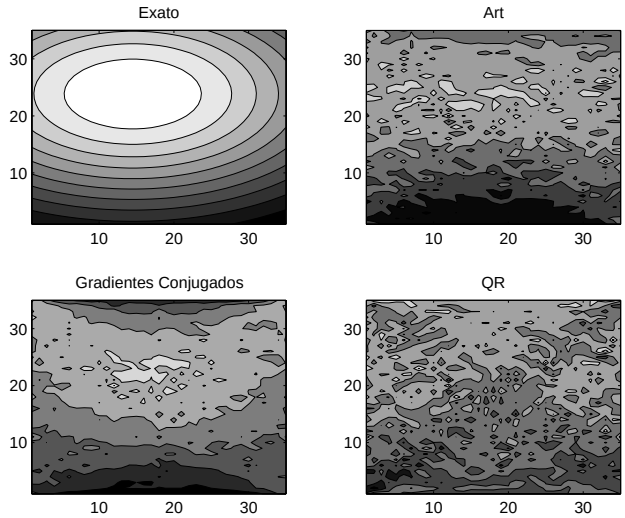

Figura 3.9: Curvas de nível na região de interesse com 1% de ruído

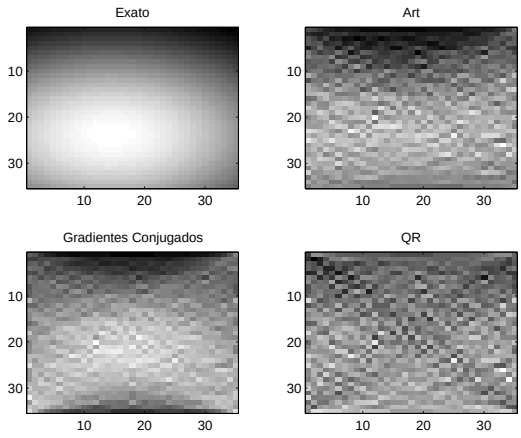

Figura 3.10: Intensidade da vagarosidade na região de interesse com 1% de ruído

Exemplo 2

Neste exemplo, supomos que o meio estudado possui três formações geológicas mais para o centro.

Ruído	Erro Relativo		
	ART	CG	QR
0%	0.0758	0.0320	0.0791
1%	0.0559	0.0184	0.0744

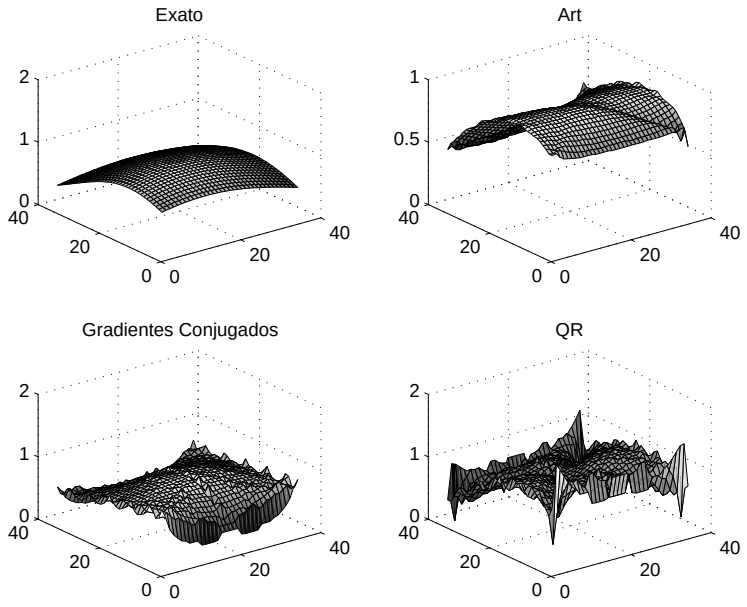

Figura 3.11: A vagarosidade em 3-D sem ruído

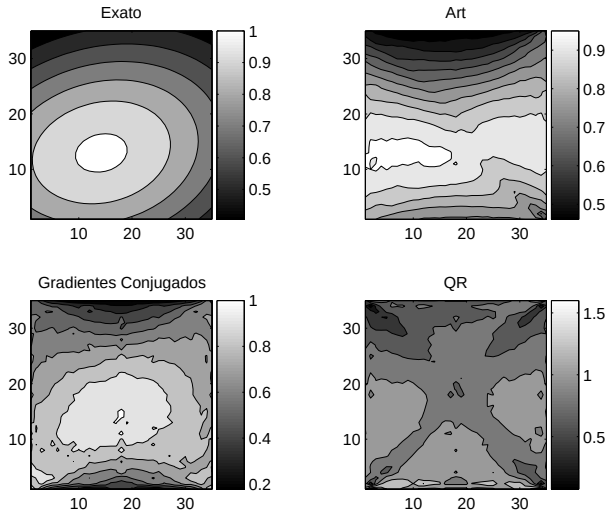

Figura 3.12: Curvas de nível na região de interesse sem ruído

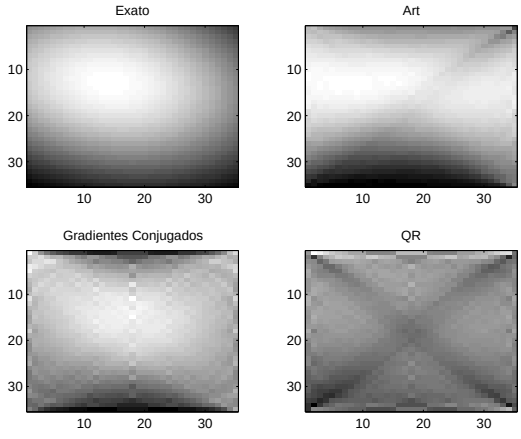

Figura 3.13: Intensidade da vagarosidade na região de interesse sem ruído

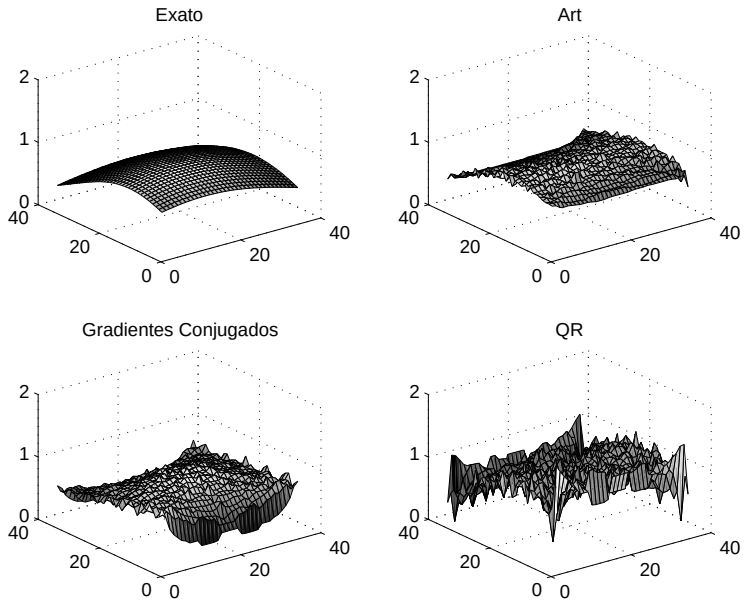

Figura 3.14: A vagarosidade em 3-D com 1% de ruído

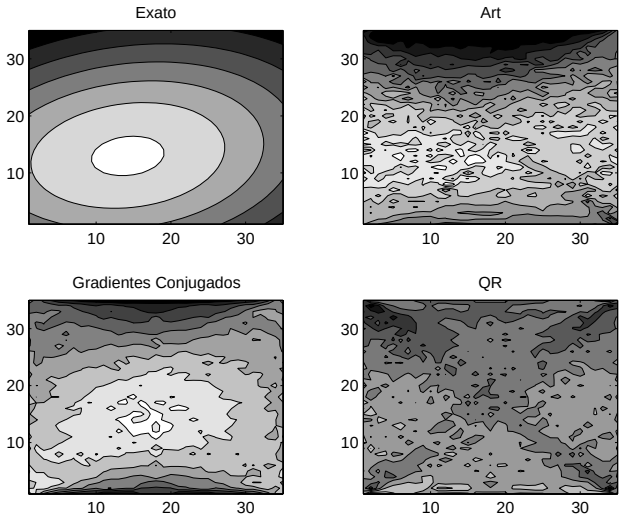

Figura 3.15: Curvas de nível na região de interesse com 1% de ruído

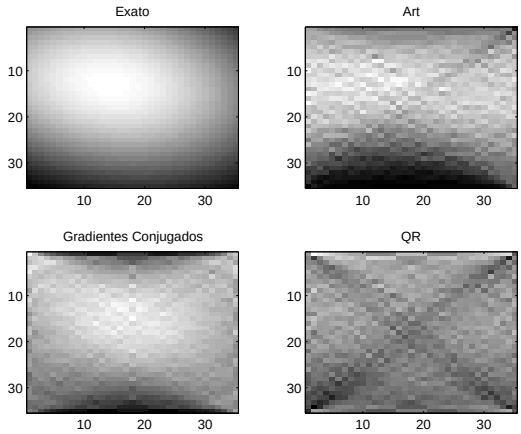

Figura 3.16: Intensidade da vagarosidade na região de interesse com 1% de ruído

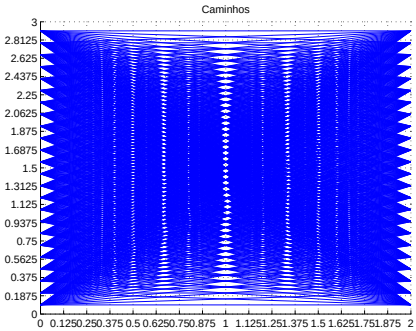

Figura 3.17: Os percursos dos raios entre os poços

Exemplo 3

Neste exemplo a vagarosidade real é dada pela função coseno. Aqui estamos lidando com um meio estratificado.

Ruído	Erro Relativo		
	ART	CG	QR
0%	0.0078	0.0507	0.0761
1%	0.0154	0.0611	0.0792

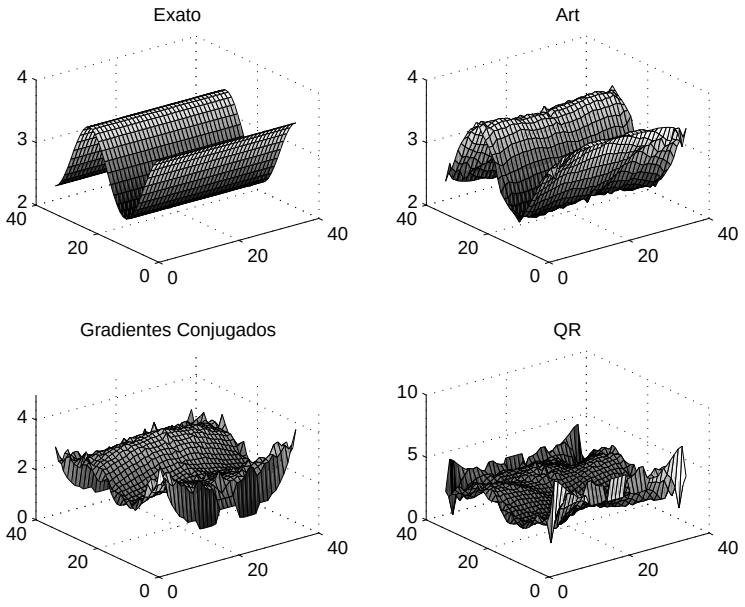

Figura 3.18: A vagarosidade em 3-D sem ruído

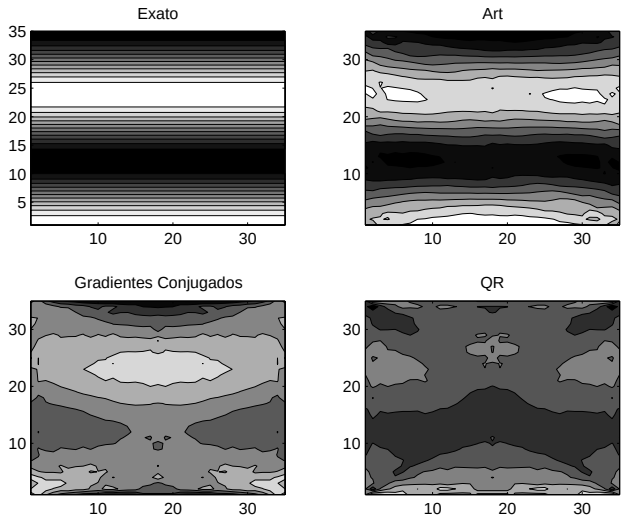

Figura 3.19: Curvas de nível na região de interesse sem ruído

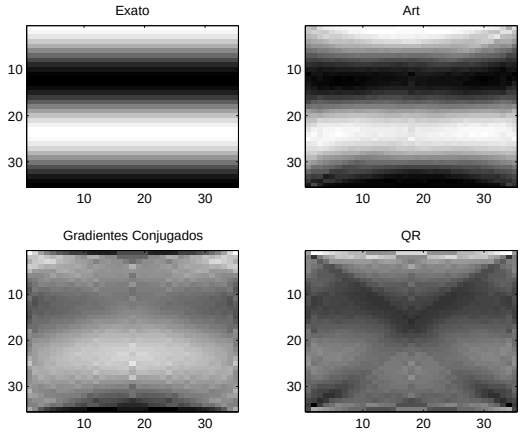

Figura 3.20: Intensidade da vagarosidade na região de interesse sem ruído

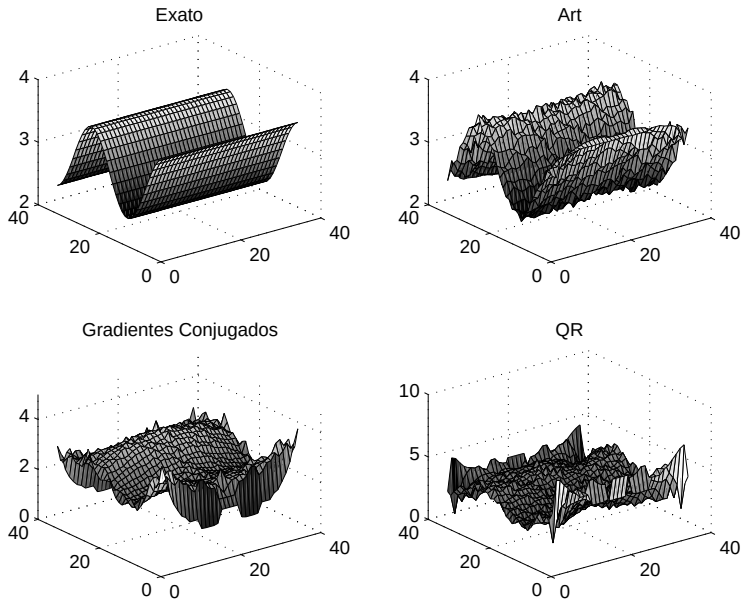

Figura 3.21: A vagarosidade em 3-D com 1% de ruído

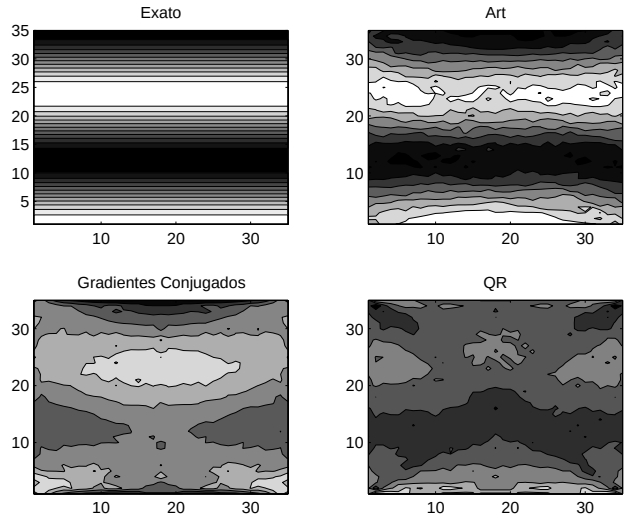

Figura 3.22: Curvas de nível na região de interesse com 1% de ruído

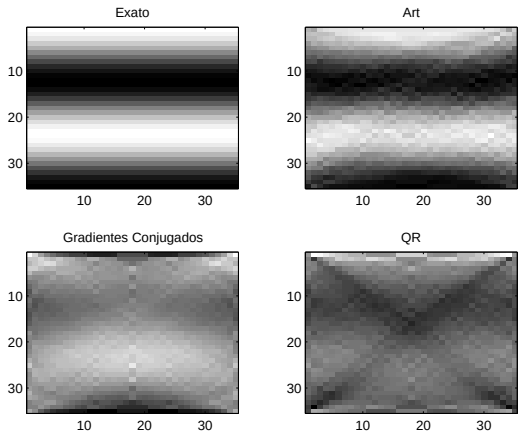

Figura 3.23: Intensidade da vagarosidade na região de interesse com 1% de ruído

Conclusões

Neste trabalho desenvolvemos um estudo sobre técnicas numéricas para atacarmos o problema de tomografia por tempo de percurso para imageamento entre poços ("bore-hole tomography"). Este problema, de grande utilidade na indústria petrolífera, é sabidamente difícil pelo seu péssimo condicionamento. Portanto nos valemos de técnicas da teoria de problemas inversos e do conceito fundamental de condicionamento.

Iniciamos com uma revisão da modelagem física e das ferramentas de álgebra linear numérica necessárias para o estudo do problema. Após uma revisão do conceito de pseudo-inversa de Moore-Penrose, e de alguns métodos diretos, de interesse teórico, como eliminação gaussiana e decomposição QR, apresentamos diversos métodos iterativos que são comumente usados na prática para problemas de grande porte.

Desenvolvemos e implementamos os seguintes algoritmos para o estudo do problema de tomografia de tempo de percurso:

- QR
- Gradientes Conjugados
- ART

Comparamos os tempos de execução dos algoritmos e para os métodos em questão obtivemos a seguinte tabela:

Pixels	Tempo de Execução (s)		
	ART	CG	QR
10X10	108	146	0.05
15X15	241	319	0.06
20X20	434	463	0.17
25X25	671	720	0.7
30X30	972	815	0.87

Aqui estamos considerando o EXEMPLO 1 com quantidade fixa de 32 fontes ao longo de um poço e 32 receptores ao longo do outro.

Observamos que, como era de se esperar, os sistemas associados ao problema de tomografia por tempo de percurso são extremamente mal condicionados. Portanto, os mesmos requerem necessariamente regularização, o que foi usado em todos os exemplos.

Observamos também que a região intermediária é melhor reconstruída que as regiões periféricas, isso se deve à iluminação nas regiões periféricas ser menor. Por este motivo, calculamos o erro relativo somente nessa região de melhor iluminação.

Vale salientar, que tivemos que optar por um tempo de execução maior para não comprometer muita memória do computador, uma vez que os problemas são de grande porte. O que fizemos foi notar que no problema $Ms = t$, podemos ver o vetor Ms como uma função aplicada em s. Isto ocasionou ganho de espaço no disco uma vez que não precisamos guardar a matriz M dos comprimentos dos percursos. O método QR possui tempo de execução menor em baixas resoluções, porém utilizamos a matriz M neste método o que para resoluções mais altas faz com que se torne inviável. A decomposição QR não se apresentou como um bom método neste problema, seu erro relativo sempre foi bem grande, mesmo na região de interesse.

Para o tamanho dos problemas em questão, os resultados do método dos gradientes conjugados parecem ter um melhor comportamento que os outros métodos, isto porém requer maior análise uma vez que métodos do tipo ART são extensamente usados em tomografia computadorizada.

Estamos trabalhando em baixa resolução, porém em experimentos com uma resolução um pouco mais alta pudemos observar que o erro relativo do método ART vai diminuindo enquanto que nos outros dois métodos esse erro tende a aumentar.

O tema da comparação entre as diversas técnicas de reconstrução, e em particular métodos do tipo ART, recebeu tremenda atenção na literatura em tomografia clássica. Uma continuação natural do presente trabalho seria a utilização dos programas aqui desenvolvidos para uma comparação abrangente nos critérios fundamentais de tempo de execução, quantidade de memória, e qualidade das reconstruções. Isto deve ser feito em particular, usando-se uma resolução bem mais alta do que aquela apresentada aqui, bem como se possível com dados reais de campo. Tal comparação detalhada foge ao escopo deste trabalho.

Apêndice A

Programas

Utilizamos Matlab versão $R14$ para todos os programas deste trabalho. Neste apêndice apresentamos as subrotinas e o programa principal implementados.

```
%PROGRAMA PRINCIPAL

% yi=vetor das fontes (coluna)
% yf=vetor dos receptores (coluna)
% L=distancia entre os pocos
% g=quantidade de intervalos de divisao no eixo x
% S=profundidade de um poco
% g2=quantidade de intervalos de divisao no eixo y
% tol=tolerancia
% int=numero de iteracoes
% graf=1;
% a=vetor linha para iniciar a minimizacao
% f=vetor coluna de g*g2 entradas
% gt=vetor dos tempos de percurso
% cam=multi-matriz com os comprimentos dos caminhos
% T2=variavel global que aparece no AdRd para guardar
% os comprimentos dados par fonte-receptor,
% com i-esimo pixel fixado
% omega=parametro de relaxacao no ART
% eta=constante de regularizacao (equacoes normais)
% x0=abscissa que centra a perturbacao do background
% da vagarosidade
% y0=ordenada que centra a perturbacao do background
% da vagarosidade
% g10=background da vagarosidade
% A=constante da gaussiana
```

```
% sigma=constante da gaussiana
%
% obs: Neste programa trabalhamos com tres exemplos
% distintos.
%
tic
%
% Variaveis globais:
%
global yi yf L g S g2 tol int graf a cam omega f gt T2
%
global eta  x0 y0 g10 A sigma
%
L=2;g=35;S=3;g2=35;a=zeros(1,2);tol=0.0001;
int=200;graf=1;omega=0.2;fonte=40;recept=40;
yi=linspace(S-(S/(2*g2)),S/(2*g2),fonte)';
yf=linspace(S-(S/(2*g2)),S/(2*g2),recept)';
g10=3;
%
% Exemplo 1
%
% A=0.3;
% x0=0.8; y0=2.0;
%
% Exemplo 2
%
A=[0.1 0.2 0.1];
x0=[0.2 0.8 1.5]; y0=[0.3 1.0 2.3];
%
% Exemplo 3
%
% A=0.2;
% x0=1; y0=2;
%
sigma=2;
%
% Rodar o problema direto
%
[m,gt1]=matrix; % Figura 1
title('Caminhos');
%
```

```
% Guardar os dados
%
save m;

%
% adicionar ruido
%
eta1=0.01;
%
%
% Aplicar Metodos
%
%    1. Chamar o bend e gerar o array de Y com
%      size = [l2 , l1, Ndivisoes ]
%       (este vetor passa como GLOBAL!)
[cam,gt]=feval('achary');
save gt;
%
toc
disp('Pronta a primeira parte')
%
%     2. Chamar o art, gradientes conjugados e as equacoes
% normais para resolver o problema
%
eta=0.2; % constante de regularizacao (variavel global)
%
[ga,ga1,deltas]=showslow(L,g,g2,S); % vagarosidade teste
save ga;
save ga1;
save deltas;
s0=3.*ones(g*g2,1);
u5=length(yi)*length(yf);
for i=1:u5
    t0(i,1)=Rd(s0,i);
end
% t0=m*s0; % problema m(deltas)=gt-m(s0), onde m(s0)=t0
gt=gt.*(1.+ eta1.*rand(size(gt)));
gt=gt-t0;
tic
%f=zeros(g*g2,1);
f=back(m,gt);
```

```
sn=artz;  % Aplicando ART
toc
disp('Pronto Metodo Art')
tic
for i=1:g*g2
    ttt(i,1)=AdRd(gt,i);
end
maxit = 150;
sn2=cgs(@(x)Freg(x),ttt,tol,maxit); % Aplicando metodo dos
% gradientes conjugados
toc
disp('Pronto Metodo dos Gradientes Conjugados')
tic
b=m'*gt;
A = (m'*m+eta*eye(g*g2));
R = triu(qr(A));
          sn3 = R\(R'\(A'*b));
          r = b - A*sn3;
          e = R\(R'\(A'*sn3));
          sn3 = sn3 + e;
toc
disp('Pronto QR')
tic
%
%  Transformar as solucoes de vetores para funcoes de duas
% variaveis constante na malha
%
m_1=0;
for j=1:g
    for i=1:g2
        m_1=m_1+1;
         ss(i,j)=sn(m_1);
         ss2(i,j)=sn2(m_1);
         ss3(i,j)=sn3(m_1);
     end
end
%
% %     3. Apresentar os resultados
%
% % Estamos resolvendo o novo problema  m(deltas)= gt
%
```

```
ga1=ga1';
figure % Figura 2
subplot(2,2,1)
surf(ga-g10);
title('Exato');
subplot(2,2,2)
surf(ss)
title('Art');
subplot(2,2,3)
surf(ss2)
title('Gradientes Conjugados')
subplot(2,2,4)
surf(ss3)
title('QR')
colormap(gray)
figure
subplot(2,2,1) % Figura 3
contourf(ga-g10)
title('Exato');
subplot(2,2,2)
contourf(ss)
title('Art');
subplot(2,2,3)
contourf(ss2)
title('Gradientes Conjugados')
subplot(2,2,4)
contourf(ss3)
title('QR')
colormap(gray)
figure
colormap(gray)
jpzj=floor(g/3);
jpzi=floor(g2/3);
jpzgmax=2*jpzj;
jpzg2max=2*jpzi;
subplot(2,2,1) % Figura 4
rrr=ga-g10;
contourf(rrr(jpzi:jpzg2max,jpzj:jpzgmax))
title('Exato')
subplot(2,2,2)
contourf(ss(jpzi:jpzg2max,jpzj:jpzgmax))
```

```
title('Art');
subplot(2,2,3)
contourf(ss2(jpzi:jpzg2max,jpzj:jpzgmax))
title('Gradientes Conjugados')
subplot(2,2,4)
contourf(ss3(jpzi:jpzg2max,jpzj:jpzgmax))
title('QR')
subplot(2,2,1) % Figura 5
gl=ga-g10;
imagesc(gl,[min(gl(:)) max(gl(:))]); colormap(gray)
title('Exato');
subplot(2,2,2)
imagesc(ss,[min(ss(:)) max(ss(:))]); colormap(gray)
title('Art');
subplot(2,2,3)
imagesc(ss2,[min(ss2(:)) max(ss2(:))]); colormap(gray)
title('Gradientes Conjugados')
subplot(2,2,4)
imagesc(ss3,[min(ss3(:)) max(ss3(:))]); colormap(gray)
title('QR')
colormap(gray)
% Erros Relativos na regiao de interesse
e_r1=norm(rrr(jpzi:jpzg2max,jpzj:jpzgmax)-...
ss(jpzi:jpzg2max,jpzj:jpzgmax))/...
norm(rrr(jpzi:jpzg2max,jpzj:jpzgmax))
e_r2=norm(rrr(jpzi:jpzg2max,jpzj:jpzgmax)-...
ss2(jpzi:jpzg2max,jpzj:jpzgmax))/...
norm(rrr(jpzi:jpzg2max,jpzj:jpzgmax))
e_r3=norm(rrr(jpzi:jpzg2max,jpzj:jpzgmax)-...
ss3(jpzi:jpzg2max,jpzj:jpzgmax))/...
norm(rrr(jpzi:jpzg2max,jpzj:jpzgmax))
toc
disp('Resultados')
```

Algoritmo matrix

```
function [m,t]=matrix
%
%  Obtem a matriz de comprimentos dos caminhos percorridos
% entre fontes e receptores em cada pixel.
%
% Variaveis globais:
        %
        % yi=vetor das fontes (coluna)
        % yf=vetor dos receptores (coluna)
        % L=distancia entre os pocos
        % g=quantidade de intervalos de divisao no eixo x
        % S=profundidade de um poco
        % g2=quantidade de intervalos de divisao no eixo y
        % tol=tolerancia
        % int=numero de iteracoes
        % graf=1;
        % a=vetor linha para iniciar a minimizacao
        % omega=parametro de relaxacao no ART
        % f=vetor coluna de g*g2 entradas
        % gt=vetor dos tempos de percurso
        % cam=multi-matriz com os comprimentos dos caminhos
        %
global yi yf L g S g2 tol int graf a cam omega f gt
%
[l,m2]=size(yi);
[l1,m1]=size(yf);
h=1;
%
% figura dos caminhos
%
hold
grid on
```

```
axis([0 L 0 S])
set(gca,'xtick',[0:L/g:L])
set(gca,'ytick',[0:S/g2:S])
%
for i=1:l
    for j=1:l1
        [xmin,f0,g0,ind,k] = feval('qnewton','bend',L,a,yi(i),...
        yf(j),g,tol,int,graf);
        q=plot([0:L/g:L],g0);
        mm=feval('benderro1',g0);
        m(h,:)= mm;
        t(h,1)=f0;
        h=1+h;
        end
end
```

Algoritmo wolfe00

Este algoritmo foi gentilmente cedido por Dayse Pastore.

```
function [xx,ff,gg,hh,isim,mode] = wolfe00(simul,...
L,y,g1,n,x,f,g,d,t,dtmin,w1,w2,isim,imp)

% [xx,ff,gg,mode] = wolfe00 (simul,n,x,f,g,d,t,...
% dtmin,w1,w2,iter,imp)
%
% Recherche lineaire de Wolfe.
% Le simulateur est appele par [f,g,indic]=
% simul(indic,x,iter).
% Tous les vecteurs en E/S sont des
% vecteurs-colonne.
%
% En entree
%
%   simul: chaine de caractere donnant le
%   nom du simulateur, appele avec indic=4
%   n: dimension de la variable x
%   x: variable
%   f: valeur fonction
%   g: gradient de f
%   d: direction de descente
%   t: pas initial
%   dtmin: variation minimale du pas
%   w1, w2: coefficient d'Armijo et de Wolfe
%   isim: nombre d'appels au simulateur avant
%   entree
%   imp=: detail d'impression
% 0, rien n'est imprime
% 1, tout est imprime
%
```

```
% En sortie
%
%   xx: nouveau x (si mode=0)
%   ff: fonction en xx (si mode=0)
%   gg: gradient en xx (si mode=0)
%   isim: nombre d'appels au simulateur en sortant
%   mode (mode de sortie):
%     0: nouveau point xx trouve,
%     1: d n'est pas une direction de descente,
%     3: l'utilisateur veut s'arreter
%     4: la simulation a echoue
%     6: fin sur dtmin.

if imp~=0 & imp~=1, imp=0;end;
% parametres
 sgi=0.01;   % sauvegarde pour l'interpolation, le
             %nouveau pas doit etre dans
             % l'intervalle [tg+sgi*(td-tg),...
             %tg+(1-sgi)*(td-tg)]
  sge=10;    % sauvegardes pour l'extrapolation,
             %le nouveau pas doit etre dans
  sgee=100;  % l'intervalle [sge*t,sgee*t]

% initialisation

  ithd=-1;   % indice de l'iteration precedente
  %donnant un x hors domaine

% tester si d est de descente

pente=g'*d;
if pente >= 0 & imp==1,
  fprintf('  wolfe00: >>> d n''est pas de descente')
  fprintf(', pente = %10.3e\n\n',pente)
  mode=1;
  return
end
w1pente=w1*pente;
w2pente=w2*pente;

% impression initiale
```

```
if imp==1,
fprintf('\n  RL de Wolfe: w2*pente = %12.5e\n\n',w2pente);
fprintf('    pas   Dfct   ppente   pente par DF\n');
end;

% iterations

tg=0; td=-1;
for it=1:100000

  % nouveau point
  xx=x+t*d;

  % appel simulateur
  [ff,gg,hh,indic]=...
  eval([simul,'(L,xx,y(1),y(2),g1)']);isim=isim+1;

  % xx hors domaine
  if indic == -1 & imp==1,
    fprintf('   %7.5f]  hors domaine\n',t);
    if it == ithd+1, divhd=divhd*2; else, divhd=2; end
    t=(t+tg)/divhd;
    ithd=it;

  % si l'utilisateur veut s'arreter ou
  % si le simulateur n'a pas pu calculer ff en xx
  elseif indic > 0,
     mode=2+indic;
     return

  % sinon on se lance dans les tests d'Armijo et de
  % Wolfe
  else

    % test d'Armijo
    if ff <= f + t * w1pente,

      ppente=gg'*d;
  if imp==1,
      fprintf('  [%7.5f   %12.5e   %12.5e   %12.5e\n',...
      t,ff-f,ppente,(ff-f)/t);
```

```
end;

   % test de Wolfe
   if ppente >= w2pente % on a un point de Wolfe
     mode=0;
     return

   else  % on a un nouveau pas gauche
     tg=t;
     if td < 0,   % alors on fait de l'extrapolation
         % quadratique

       % essai de pas
       a0=f;
       a1=pente;
       a2=(3*ff-t*ppente-3*a0-2*a1*t)/t^2;
       a3=(ppente-pente-2*a2*t)/t^2/3;
       rad=a2^2-3*a1*a3;
       if a3 ~= 0,
         if rad >= 0,   % le min du modele convient
           tt=(-a2+sqrt(rad))/a3/3;
         else
           tt=0;
         end
       elseif a2 > 0
         tt=-a1/a2/2;
       else
         tt=0;
       end

       % sauvegarde
       if tt < sge*t
         t=sge*t;
       elseif tt > sgee*t
         t=sgee*t;
       else
         t=tt;
       end

     else       % on fait de l'interpolation
                % quadratique
```

```
          % essai de pas
          a0=f;
          a1=pente;
          a2=(3*ff-t*ppente-3*a0-2*a1*t)/t^2;
          a3=(ppente-pente-2*a2*t)/t^2/3;
          rad=a2^2-3*a1*a3;
          if a3 > 0,  % on prend le plus grand min
            tt=(-a2+sqrt(rad))/a3/3;
          else
            if a2 > 0 & rad > 0,   % le premier point
                % stationnaire est le min
              tt=(-a2+sqrt(rad))/a3/3;
            else
              tt=sgee*t;
            end
          end

          % sauvegarde
          if tt < tg+sgi*(td-tg),
            t=tg+sgi*(td-tg);
          elseif tt > tg+(1-sgi)*(td-tg),
            t=tg+(1-sgi)*(td-tg);
          else
            t=tt;
          end

        end
      end

    else  % test d'Armijo non verifie, on a un
          % nouveau pas droit
if imp==1,
      fprintf('   %7.5f]  %12.5e   %12.5e\n',t,...
      ff-f,(ff-f)/t);
end;
      td=t;

      % nouveau t par interpolation quadratique!!!!!!
      % on pourrait faire cubique
      tt=-pente*t^2/(ff-f-pente*t)/2;
```

```
      % on sait que tt < 0.5*t/(1-w1)
      if tt < sgi*t,
        t=sgi*t;
      elseif tt > (1-sgi)*t,
        t=(1-sgi)*t;
      else
        t=tt;
      end

% fprintf('\nt = %12.5e,   tt = %12.5e\n',t,tt)

    end
  end

  % test sur dtmin

  if td > 0 & td-tg < dtmin,
     mode=6;
if imp==1,
     fprintf('\nwolfe00: >>> La RL a echoue sur dtmin\n\n')
end;
     return
  end
end
```

Algoritmo qnewton

Este algoritmo foi gentilmente cedido por Dayse Pastore.

```
function [xmin,f,h,ind,k] = qnewton(simul,L,a,yi,yf,g,tol,int,graf)

% Metodo quase Newton com BUSCA LINEAR DE WOLFE
%
% Sintaxe:
% [xmin, ind, k] = qnewton(simul,x0, tol,int,graf)
%
% onde:
% Entradas:
% simul=nome do arquivo .m contindo o simulador
% x0=ponto inicial para a busca do minimo de f
%     tol=tolerancia
%       int=numero maximo de interacoes
%       graf=indicador se e para plotar ou nao f(xk)
%       graf==0 plotar
%       graf<>0 nao plotar
% Saidas:
% xmin=ponto que minimiza a funcao f
% ind=indicador do andamento do processo de
%       minimizacao:
% 0: tudo foi bem
% >0: avaliacao impossivel
% k=numero de iteracoes
% hold;
x0=a;
[m,n]=size(x0);
if n>1 & m>1
    disp('x0 nao e um vetor'); return;
end
if max(abs(x0))==inf
```

```
    disp('x0 nao esta bem definido'); return;
end
if n>1
    x0 = x0';
    a=a';
end
%
[f0, g0, y,ind] = eval([simul,'(L,a,yi,yf,g)']);
%
k=1;
x = a;
f = f0;
g1 = g0;
isim=1;
h=y;
W=eye(max(n,m));
if norm(g0)  <= tol ...  % condicao de parada
        xmin=x;
    return
end
while ind == 0 & k<int
    if size(a, 1) == 1      % x e vetor 1xn
        seq(k,:)=a;
    elseif size(a, 2) == 1   % x e vetor nx1
        seq(k,:) = a';
    end
    d=-W*g1; % definicao da direcao de descida
    %
    %%%%%% Busca linear %%%%%%%%
    %
    n=max(size(a));dtmin=1.d-5;w1=0.01;w2=0.99;out=0;
    t0=1;a0=a;g2=g1;
    [a,f,g1,h,isim,ind] =...
        wolfe00(simul,L,[yi,yf],g,n,a,f,g1,d,t0,...
    dtmin,w1,w2,isim,out);
    %
    xmin=a;vf(k)=f;
    %
    %     0: novo ponto xx encontrado,
    %     1: d nao eh uma direcao de descida
    %     3: o usuario pode parar
```

```
    %     4: a simulacao fracassou
    %     6: fim em dtmin.
    if ind==1
        disp('d nao e uma direcao de descida');
        if graf==0
            plot(vf);
        end
        return;
    elseif ind==3
        disp('o usuario pode parar');
        if graf==0
            plot(vf);
        end
        return;
    elseif ind==4
        disp('a simulacao fracassou');
        if graf==0
            plot(vf);
        end
        return;
    elseif ind==6
        %disp('fim em dtmin');
        if graf==0
            plot(vf);
        end
        return;
    end

    if norm(g1) / norm(g0) <= tol ...  % condicao de parada
            return
    end
    k = k + 1;
    s=a-a0;
    y=g1-g2;
    W=W-(s*y'*W+W*(y*s'))/(y'*s)+ ...
    (1+(y'*(W*y))/(y'*s))*(s*s')/(y'*s);
end
if graf==0
    plot(vf);
end
```

Algoritmo bend

```
function  [f0, g0, y, ind]=bend(L,a,y_i,y_f,g)
%
% Programa para obter o caminho de Fermat.
% Subrotina do programa qnewton.
%
% Variaveis globais:
        %
        % x0=abscissa que centra a perturbacao do
        % background da vagarosidade
        % y0=ordenada que centra a perturbacao do
        % background da vagarosidade
        % g10=background da vagarosidade
        % A=constante da gaussiana
        % sigma=constante da gaussiana
        %
global x0 y0 g10 A sigma
%
a=a';
[l,k]=size(a);
x=[0:L/g:L];
y=y_i.*(1-x/L)+y_f.*x/L+a*sin((pi.*[1:1:k]'*x)/L);
s=([1:1:k].*pi.*a)*cos((pi.*[1:1:k]'*x)/L)/L;
dy=((y_f-y_i)/L  + s);
% gaussiana exemplo 1
%
% ga=g10+ A*exp(-(((x-x0).^2)+ ((y-y0).^2))/(2*sigma));
%
% gaussiana exemplo 2
%
ga=g10.*(1 + A(1).*(exp(-(((x-x0(1)).^2)+...
((y-y0(1)).^2))/(2*sigma)))'+...
A(2).*(exp(-(((x-x0(2)).^2)+...
```

```
((y-y0(2)).^2))/(2*sigma)))'+ ...
A(3).*(exp(-(((x-x0(3)).^2)+...
((y-y0(3)).^2))/(2*sigma)))');
ga=ga';
%
% gaussiana exemplo 3
%
%  ga=g10.*(2+A.*cos(2*pi.*(y-y0)/L));
%
% integrando
%
f=ga.*(sqrt(1+dy.^2));
%
% integral de f
f0=(L/g)*(sum(f(2:g))+(f(1)+f(g+1))/2);
%
% exemplo 1 (derivada de f(x,a))
%
% for i=1:k
% g1(:,i)=((pi*i/L).*(sqrt(1./(1+dy.^2))).*...
% ga.*dy.*cos((i*pi/L).*x))'+ ...
% ((g10-ga).*(((y-y0))/sigma).*...
% sin((i*pi/L).*x).*(sqrt(1+dy.^2)))';
% end
%
% exemplo 2 (derivada de f(x,a))
%
for i=1:k
g1(:,i)=((pi*i/L).*(sqrt(1./(1+dy.^2))).*...
ga.*dy.*cos((i*pi/L).*x))'+ ...
(A(1).*exp(-(((x-x0(1)).^2)+...
((y-y0(1)).^2))/(2*sigma)).* ...
((y-y0(1))/sigma)+A(2).*...
exp(-(((x-x0(2)).^2)+ ...
((y-y0(2)).^2))/(2*sigma)).*...
((y-y0(2))/sigma)+ ...
A(3).*exp(-(((x-x0(3)).^2)+...
((y-y0(3)).^2))/(2*sigma)).*...
((y-y0(3))/sigma).*sin((i*pi/L).*x).*...
(sqrt(1+dy.^2)))';
end
```

```
%
%exemplo 3 (derivada de f(x,a))
%
% for i=1:k
% g1(:,i)=((pi*i/L).*(sqrt(1./(1+dy.^2))).*...
% ga.*dy.*cos((i*pi/L).*x))'+ ...
% (g10.*A.*sin(2*pi.*(y-y0)/L).*...
% 2*pi/L.*sin((i*pi/L).*x).*(sqrt(1+dy.^2)))';
% end

% integral de g1
g0=((L/g)*(sum(g1(2:g,:))+(g1(1,:)+...
g1(g+1,:))./2))';
%
ind=0;
```

Algoritmo comprimento

```
function m=benderro1(y)
%
%  Subrotina que calcula os comprimentos dos
% caminhos utilizando geometria analitica.
%
% Variaveis globais:
        %
        % yi=vetor das fontes (coluna)
        % yf=vetor dos receptores (coluna)
        % L=distancia entre os pocos
        % g=quantidade de intervalos de divisao no eixo x
        % S=profundidade de um poco
        % g2=quantidade de intervalos de divisao no eixo y
        % tol=tolerancia
        % int=numero de iteracoes
        % graf=1;
        % a=vetor linha para iniciar a minimizacao
        % omega=parametro de relaxacao no ART
        % f=vetor coluna de g*g2 entradas
        % gt=vetor dos tempos de percurso
        % cam=multi-matriz com os comprimentos dos caminhos
        %
global yi yf L g S g2 tol int graf a cam omega f gt
%
n=1;
%
for l=1:g
    R=y(l+1)-y(l);
    if R>0
        for k=1:g2
            if (k-1)*S/g2 <= y(l) & y(l)< k*S/g2
                if (k-1)*S/g2 <= y(l+1) & y(l+1)<= k*S/g2
```

```
                m(n)=sqrt([L/g,R]*[L/g,R]');
                n=n+1;
            else
                x=(L*(k*S/g2-y(l+1)+l*R))/(g*R);
                m(n)= sqrt([x-(l-1)*L/g,k*S/g2-y(l)]*...
                [x-(l-1)*L/g,k*S/g2-y(l)]');
                n=n+1;
            end
        elseif y(l)< (k-1)*S/g2
            if y(l+1)>= k*S/g2
                xk=(L*((k-1)*S/g2-y(l+1)+l*R))/(g*R);
                xk1=(L*(k*S/g2-y(l+1)+l*R))/(g*R);
                m(n)= sqrt([xk1-xk,S/g2]*[xk1-xk,S/g2]');
                n=n+1;
            elseif (k-1)*S/g2 <= y(l+1) & y(l+1)<= k*S/g2
                x=(L*((k-1)*S/g2-y(l+1)+l*R))/(g*R);
                m(n)= sqrt([x-l*L/g,(k-1)*S/g2-y(l+1)]*...
                [x-l*L/g,(k-1)*S/g2-y(l+1)]');
                n=n+1;
            else
                m(n)=0;
                n=n+1;
            end
        else
            m(n)=0;
            n=n+1;
        end
    end
elseif R<0
    for k=1:g2
        if (k-1)*S/g2 < y(l) & y(l)<= k*S/g2
            if (k-1)*S/g2 <= y(l+1) & y(l+1)<= k*S/g2
                m(n)=sqrt([L/g,R]*[L/g,R]');
                n=n+1;
            else
                 x=(L*((k-1)*S/g2-y(l+1)+l*R))/(g*R);
                 m(n)= sqrt([x-(l-1)*L/g,(k-1)*S/g2-y(l)]*...
                 [x-(l-1)*L/g,(k-1)*S/g2-y(l)]');
                 n=n+1;
            end
        elseif y(l)> k*S/g2
```

```
                if y(l+1)<= (k-1)*S/g2
                    xk=(L*(k*S/g2-y(l+1)+l*R))/(g*R);
                    xk1=(L*((k-1)*S/g2-y(l+1)+l*R))/(g*R);
                    m(n)= sqrt([xk-xk1,S/g2]*[xk-xk1,S/g2]');
                    n=n+1;
                elseif (k-1)*S/g2 <= y(l+1) & y(l+1)<= k*S/g2
                    x=(L*(k*S/g2-y(l+1)+l*R))/(g*R);
                    m(n)= sqrt([x-l*L/g,k*S/g2-y(l+1)]*...
                    [x-l*L/g,k*S/g2-y(l+1)]');
                    n=n+1;
                else
                    m(n)=0;
                    n=n+1;
                end
            else
                m(n)=0;
                n=n+1;
            end
        end
    else
        for k=1:g2
            if (k-1)*S/g2 <= y(l) & y(l)<= k*S/g2
                m(n)=L/g;
                n=n+1;
            else
                m(n)=0;
                n=n+1;
            end
        end
    end
end
```

Algoritmo achary

```
function [y,t]=achary
%
% OUTPUTS:
        %  Matriz y(l,k) que tem como entradas o vetor y
        % que tem tamanho g+1.
        %  A outra saida t e o tempo de percurso minimizado
        % entre todas as fontes e receptores respectivos.
%
% Variaveis globais:
        %
        % yi=vetor das fontes (coluna)
        % yf=vetor dos receptores (coluna)
        % L=distancia entre os pocos
        % g=quantidade de intervalos de divisao no eixo x
        % S=profundidade de um poco
        % g2=quantidade de intervalos de divisao no eixo y
        % tol=tolerancia
        % int=numero de iteracoes
        % graf=1;
        % a=vetor linha para iniciar a minimizacao
        %
global yi yf L g S g2 tol int graf a
%
[l2,m2]=size(yi);
[l1,m1]=size(yf);
%
n=1;
for l=1:l2
    for ks=1:l1
        [xmin,f0,g0,ind,k] = feval('qnewton','bend',L,a,yi(l),...
        yf(ks),g,tol,int,graf);
        for m=1:g+1
```

```
            y(l,ks,m)=g0(m);
        end
        t(n,1)=f0;
        n=n+1;
    end
end
```

Algoritmo showslow

```
function [ga,ga1,deltas]= showslow(L,g,g2,S)
%
%  Transforma a vagarosidade real do problema em
% um vetor e em uma funcao de duas variaveis
%
% Variaveis globais:
       %
       % x0=abscissa que centra a perturbacao do
       % background
       % da vagarosidade
       % y0=ordenada que centra a perturbacao do
       % background
       % da vagarosidade
       % g10=background da vagarosidade
       % A=constante da gaussiana
       % sigma=constante da gaussiana
       %
global x0 y0 g10 A sigma
%
dx1=L/(2*g);
dy1=S/(2*g2);
m_1=0;
%
% Exemplo 1
% for j=1:g
%     for i=1:g2
%         m_1=m_1+1;
%         ga(i,j)= g10+ A*exp(-((((j*L/g - dx1)-x0).^2)+ ...
%             (((i*S/g2 - dy1)-y0).^2))/(2*sigma));
%         deltas(i,j)=A*exp(-((((j*L/g - dx1)-x0).^2)+ ...
%             (((i*S/g2 - dy1)-y0).^2))/(2*sigma));
%         ga1(m_1)=ga(i,j);
```

```
%     end
% end
%
% Exemplo 2
%
for j=1:g
    for i=1:g2
        m_1=m_1+1;
        ga(i,j)=g10.*(1 + A(1)* ...
          (exp(-((((j*L/g - dx1)-x0(1)).^2)+ ...
          (((i*S/g2 - dy1)-y0(1)).^2))/(2*sigma)))'+...
          A(2)*(exp(-((((j*L/g - dx1)-x0(2)).^2)+ ...
          (((i*S/g2 - dy1)-y0(2)).^2))/(2*sigma)))'+ ...
          A(3)*(exp(-((((j*L/g - dx1)-x0(3)).^2)+...
          (((i*S/g2 - dy1)-y0(3)).^2))/(2*sigma)))');
        deltas(i,j)=g10.*(A(1)* ...
          (exp(-((((j*L/g - dx1)-x0(1)).^2)+ ...
          (((i*S/g2 - dy1)-y0(1)).^2))/(2*sigma)))'+...
          A(2)*(exp(-((((j*L/g - dx1)-x0(2)).^2)+ ...
          (((i*S/g2 - dy1)-y0(2)).^2))/(2*sigma)))'+ ...
          A(3)*(exp(-((((j*L/g - dx1)-x0(3)).^2)+...
          (((i*S/g2 - dy1)-y0(3)).^2))/(2*sigma)))');
        ga1(m_1)=ga(i,j);
    end
end
%
% Exemplo 3

% for j=1:g
%     for i=1:g2
%         m_1=m_1+1;
%         ga(i,j)= g10.*(2 +A.*cos(2*pi.*((i*S/g2 - dy1)-y0)/L));
%         deltas(i,j)=g10.*A.*cos(2*pi.*((i*S/g2 - dy1)-y0)/L);
%         ga1(m_1)=ga(i,j);
%     end
% end
```

Algoritmo back

```
function s=back(M, t)
[m1,n1]=size(M);
L=sum(M');
for j=1:n1
    for i=1:m1
        if M(i,j)>0
            sgn(i)=1;
            k(i)=t(i)/L(i);
        else
            sgn(i)=0;
            k(i)=0;
        end
    end
    N=sum(sgn);
    if N~=0
        s(j)= 1/N * sum(k);
    else
        s(j)=0;
    end
end
s=s';
```

Algoritmo art

```
function y = artz

%  Aplica o metodo ART ao problema R * f = g
% onde R: x|->y e definida pela funcao
% y(i) = Rd(x,i)
%
% Variaveis globais:
        %
        % yi=vetor das fontes (coluna)
        % yf=vetor dos receptores (coluna)
        % L=distancia entre os pocos
        % g=quantidade de intervalos de divisao
        % no eixo x
        % S=profundidade de um poco
        % g2=quantidade de intervalos de divisao
        % no eixo y
        % tol=tolerancia
        % int=numero de iteracoes
        % graf=1;
        % a=vetor linha para iniciar a minimizacao
        % f=vetor coluna de g*g2 entradas
        % gt=vetor dos tempos de percurso
        % cam=multi-matriz com os comprimentos
        % dos caminhos
        % T2=variavel global que aparece no AdRd
        % para guardar os
        % comprimentos dados par fonte-receptor,
        % com i-esimo pixel fixado
        % omega=parametro de relaxacao no ART
        %
global yi yf L g S g2 tol int graf a omega f gt T2
%
```

```
p = length(f);
glength = length(gt);
% Comecamos computando os valores de R(j)*R(j)'
tmp = zeros(glength,1);
for j=1:glength
    tmp(j,1)=1; % Neste momento tmp torna-se e_j
    for i=1:p
        gtmp(i,1) = AdRd(tmp,i);
    end
    % INR(j) = 1/norm(gtmp)^2; %
    INR(j) = 1/((gtmp')*gtmp);
    tmp(j,1)=0; % Reseta a zero tmp
end
        k=0; % Conta o numero de iteracoes externas
        relat=2*tol;
while (k<5)&(relat>tol) % (k<int)
    ftmp=f; % Salva f para esta iteracao externa
    for j=1:glength
        tmp(j,1)=1; % Neste momento tmp torna-se e_j
        for i=1:p
            gtmp(i,1) = AdRd1(tmp,i);
        end
        tmp(j,1)=0; % Reseta a zero tmp
        f = f + omega*INR(j)*(gt(j,1)-Rd(f,j))*gtmp;
    end
    k=k+1;
    relat=norm(f-ftmp)/norm(f); % Pode ser retirado para
    % melhorar performance
end
y=f;
tmres=0
for j=1:glength
    tmres=tmres+(gt(j,1)-Rd(f,j))^2;
end

tmres=sqrt(tmres)/norm(gt)
```

Algoritmo AdRd

```
function xj = adRd(y,j)
% Function adRd(y,j)
% Computes the j-th component of the ADJOINT of R applied to the vect
% global vecsize
vecsize = length(y);
alfa=0.2; % ATTENTION THIS SHOULD BE CONSISTENT WITH Rd
beta=0.1; % ATTENTION THIS SHOULD BE CONSISTENT WITH Rd
% This is a tridiagonal matrix
if j==1
    xj=y(1)+alfa*y(2);
elseif j~=vecsize
% for i=2:(vecsize-1),
    xj=y(j)+alfa*y(j+1)+beta*y(j-1);
% end
else
xj=y(vecsize)+beta*y(vecsize-1);
end
%
% AQUI DEVERA ENTRAR A ADJUNTA
```

Algoritmo AdRd1

```
function w=AdRd1(s,i) %calcula a i-esima coordenada de R'*s
%s=vetor coluna de tamanho l1*l2
global yi yf L g S g2 tol int graf a cam omega f gt T2
[l2,m2]=size(yi);
[l1,m1]=size(yf);
l5=l1*l2;
% n=1;
%  Estamos calculando no i-esimo pixel, qual o comprimento
% de cada caminho que passa por ele.
T=0;
for j=1:l5
    if s(j)==0
        T=T;
    else
        %  Calculamos agora de qual fonte este j caminho
        % partiu
        lc=ceil(j/l1);
        %  Calculamos agora em qual receptor este j caminho
        % chegou
        if mod(j,l1)==0
            kc=l1;
        else
            kc=mod(j,l1);
        end
        T=T+T2(lc,kc,i)*s(j);
    end
end
w=T;
```

Algoritmo tempo1

```
function T10=tempo1(y,i)
%
% Este programa visa calcular o comprimento do
%caminho y na i-esima celula.
%
global yi yf L g S g2 tol int graf a cam omega f gt
%
l=ceil(i/g2);
if mod(i,g2)==0
    k=g2;
else
    k=mod(i,g2);
end
R=y(l+1)-y(l);
if R>0
    if (k-1)*S/g2 <= y(l) & y(l)< k*S/g2
        if (k-1)*S/g2 <= y(l+1) & y(l+1)<= k*S/g2
            T=sqrt([L/g,R]*[L/g,R]');
        else
            x=(L*(k*S/g2-y(l+1)+l*R))/(g*R);
            T= sqrt([x-(l-1)*L/g,k*S/g2-y(l)]*...
            [x-(l-1)*L/g,k*S/g2-y(l)]');
        end
    elseif y(l)< (k-1)*S/g2
        if y(l+1)>= k*S/g2
            xk=(L*((k-1)*S/g2-y(l+1)+l*R))/(g*R);
            xk1=(L*(k*S/g2-y(l+1)+l*R))/(g*R);
            T= sqrt([xk1-xk,S/g2]*[xk1-xk,S/g2]');

        elseif (k-1)*S/g2 <= y(l+1) & y(l+1)<= k*S/g2
            x=(L*((k-1)*S/g2-y(l+1)+l*R))/(g*R);
            T= sqrt([x-l*L/g,(k-1)*S/g2-y(l+1)]*...
```

```
            [x-l*L/g,(k-1)*S/g2-y(l+1)]');
        else
            T=0;
        end
    else
        T=0;
    end
elseif R<0
    if (k-1)*S/g2 < y(l) & y(l)<= k*S/g2
        if (k-1)*S/g2 <= y(l+1) & y(l+1)<= k*S/g2
            T=sqrt([L/g,R]*[L/g,R]');
        else
            x=(L*((k-1)*S/g2-y(l+1)+l*R))/(g*R);
            T= sqrt([x-(l-1)*L/g,(k-1)*S/g2-y(l)]*...
            [x-(l-1)*L/g,(k-1)*S/g2-y(l)]');
        end
    elseif y(l)> k*S/g2
        if y(l+1)<= (k-1)*S/g2
            xk=(L*(k*S/g2-y(l+1)+l*R))/(g*R);
            xk1=(L*((k-1)*S/g2-y(l+1)+l*R))/(g*R);
            T= sqrt([xk-xk1,S/g2]*[xk-xk1,S/g2]');
        elseif (k-1)*S/g2 <= y(l+1) & y(l+1)<= k*S/g2
            x=(L*(k*S/g2-y(l+1)+l*R))/(g*R);
            T= sqrt([x-l*L/g,k*S/g2-y(l+1)]*...
            [x-l*L/g,k*S/g2-y(l+1)]');
        else
            T=0;
        end
    else
        T=0;
    end
else
        if (k-1)*S/g2 <= y(l) & y(l)<= k*S/g2
            T=L/g;

        else
            T=0;
        end
end
T10=T;
```

Algoritmo Rd

```
function z=Rd(s,i)
%
% Calcula a i-esima coordenada de R*s
%
% Variaveis globais:
        %
        % yi=vetor das fontes (coluna)
        % yf=vetor dos receptores (coluna)
        % L=distancia entre os pocos
        % g=quantidade de intervalos de divisao no
        % eixo x
        % S=profundidade de um poco
        % g2=quantidade de intervalos de divisao no
        % eixo y
        % tol=tolerancia
        % int=numero de iteracoes
        % graf=1;
        % a=vetor linha para iniciar a minimizacao
        % cam=multi-matriz com os comprimentos dos
        % caminhos
        %
global yi yf L g S g2 tol int graf a cam
%
[l2,m2]=size(yi);
[l1,m1]=size(yf);
%  Calcularemos agora de qual fonte o i-esimo caminho
% partiu
l=ceil(i/l1);
%  Calcularemos agora em qual receptor o i-esimo caminho
% chegou
if mod(i,l1)==0
    k=l1;
```

```
else
k=mod(i,l1);
end
%
%  Aqui vamos para o algoritmo tempo que uma vez tendo
% o caminho, calcula para o i-esimo caminho, o produto
% dessa linha i-esimo caminho (dos comprimentos desse
% caminho em cada pixel) por s.
z=feval('tempo',cam(l,k,:),s);
```

Algoritmo tempo

```
function T=tempo(y,s)
%
% s=vetor coluna com g*g2 entradas
%
global yi yf L g S g2 tol int graf a cam
%
%  Dado caminho y, este programa calcula o produto
% do vetor dos comprimentos em cada pixel desse caminho
% com o vetor s. Isso nos da um numero
%
n=1;
T=0;
for l=1:g
    R=y(l+1)-y(l);
    if R>0
        for k=1:g2
            if (k-1)*S/g2 <= y(l) & y(l)< k*S/g2
                if (k-1)*S/g2 <= y(l+1) & y(l+1)<= k*S/g2
                    m(n)=sqrt([L/g,R]*[L/g,R]');
                    T=T + (m(n)*s(n));
                    n=n+1;
                else
                    x=(L*(k*S/g2-y(l+1)+l*R))/(g*R);
                    m(n)= sqrt([x-(l-1)*L/g,k*S/g2-y(l)]*...
                    [x-(l-1)*L/g,k*S/g2-y(l)]');
                    T=T + (m(n)*s(n));
                    n=n+1;
                end
            elseif y(l)< (k-1)*S/g2
                if y(l+1)>= k*S/g2
                    xk=(L*((k-1)*S/g2-y(l+1)+l*R))/(g*R);
                    xk1=(L*(k*S/g2-y(l+1)+l*R))/(g*R);
```

```
                m(n)= sqrt([xk1-xk,S/g2]*[xk1-xk,S/g2]');
                T=T + (m(n)*s(n));
                n=n+1;
            elseif (k-1)*S/g2 <= y(l+1) & y(l+1)<= k*S/g2
                x=(L*((k-1)*S/g2-y(l+1)+l*R))/(g*R);
                m(n)= sqrt([x-l*L/g,(k-1)*S/g2-y(l+1)]*...
                [x-l*L/g,(k-1)*S/g2-y(l+1)]');
                T=T + (m(n)*s(n));
                n=n+1;
            else
                m(n)=0;
                n=n+1;
            end
        else
            m(n)=0;
            n=n+1;
        end
    end
elseif R<0
    for k=1:g2
        if (k-1)*S/g2 < y(l) & y(l)<= k*S/g2
            if (k-1)*S/g2 <= y(l+1) & y(l+1)<= k*S/g2
                m(n)=sqrt([L/g,R]*[L/g,R]');
                T=T + (m(n)*s(n));
                n=n+1;
            else
                 x=(L*((k-1)*S/g2-y(l+1)+l*R))/(g*R);
                 m(n)= sqrt([x-(l-1)*L/g,(k-1)*S/g2-y(l)]*...
                 [x-(l-1)*L/g,(k-1)*S/g2-y(l)]');
                 T=T + (m(n)*s(n));
                 n=n+1;
            end
        elseif y(l)> k*S/g2
            if y(l+1)<= (k-1)*S/g2
                xk=(L*(k*S/g2-y(l+1)+l*R))/(g*R);
                xk1=(L*((k-1)*S/g2-y(l+1)+l*R))/(g*R);
                m(n)= sqrt([xk-xk1,S/g2]*[xk-xk1,S/g2]');
                T=T + (m(n)*s(n));
                n=n+1;
            elseif (k-1)*S/g2 <= y(l+1) & y(l+1)<= k*S/g2
                x=(L*(k*S/g2-y(l+1)+l*R))/(g*R);
```

```
                    m(n)= sqrt([x-l*L/g,k*S/g2-y(l+1)]*...
                    [x-l*L/g,k*S/g2-y(l+1)]');
                    T=T + (m(n)*s(n));
                    n=n+1;
                else
                    m(n)=0;
                    n=n+1;
                end
            else
                m(n)=0;
                n=n+1;
            end
        end
    else
        for k=1:g2
            if (k-1)*S/g2 <= y(l) & y(l)<= k*S/g2
                m(n)=L/g;
                T=T + (m(n)*s(n));
                n=n+1;
            else
                m(n)=0;
                n=n+1;
            end
        end
    end
end
```

Apêndice B

Unicidade da pseudo-inversa

Nosso objetivo aqui é mostrar que existe uma única matriz X satisfazendo o seguinte sistema de equações:

$$\begin{aligned} MXM &= M && \text{(B.1)} \\ XMX &= X && \text{(B.2)} \\ (MX)^t &= MX && \text{(B.3)} \\ (XM)^t &= XM && \text{(B.4)} \end{aligned}$$

onde M é uma matriz real qualquer (possivelmente retangular). Como já sabemos esse sistema de equações é chamado de condições de Penrose.

Proposição 1. *Existe única solução X do sistema de equações:*

$$\begin{aligned} MXM &= M && \text{(B.5)} \\ (MX)^t &= MX && \text{(B.6)} \\ (XM)^t &= XM && \text{(B.7)} \end{aligned}$$

para qualquer matriz M de posto máximo.

Demonstração Sejam X e Y duas soluções do sistema acima. De B.1 obtemos que

$$MXM = MYM = M,$$

e, assim,

$$M(X-Y)M = 0. \tag{B.8}$$

multiplicando B.8 por $(X-Y)$ à esquerda obtemos:

$$0 = (X-Y)M(X-Y)M = [(X-Y)M]^t(X-Y)M,$$

onde a segunda igualdade segue de B.4, o que implica $(X-Y)M = 0$. Assim as colunas de $(X - Y)$ se encontram no núcleo de M. Analogamente, utilizando B.3 obtemos:

$$(X - Y)M = 0,$$

mostrando que (X-Y) está no núcleo à esquerda de M.

Como M tem posto máximo, $(X - Y)$ tem que ser a matriz nula.

Teorema 8. *As condições de Penrose possuem uma única solução X para qualquer matriz real M.*

Demonstração Suponha X e Y soluções do sistema de equações. Note que acrescentamos somente a equação $XMX = X$ às equações da proposição anterior.

Da prova da proposição anterior obtemos:

$$MX = MY$$

e

$$XM = YM.$$

Assim,

$$XMX = YMX = YMY$$

logo,

$$X = YMX = Y.$$

Bibliografia

[1] V. I. Arnol'd. *Mathematical methods of classical mechanics*. New York. Springer-Verlag, 1978.

[2] Owe Axelsson. Iterative solution methods. *Cambridge*, 1994.

[3] Johann Baumeister and Antonio Leitao. *Topics in inverse problems*. COLÓQUIO BRASILEIRO DE MATEMÁTICA. IMPA, 2005.

[4] James G. Berryman. Fermat's principle and nonlinear traveltime tomography. *Phys. Rev. Lett.*, 62(25):2953–2956, 1989.

[5] James G. Berryman. Lecture notes on nonlinear inversion and tomography, i. borehole seismic tomography. 1991.

[6] James G. Berryman. Analysis of approximate inverses in tomography. I. Resolution analysis of common inverses. *Optim. Eng.*, 1(1):87–115, 2000.

[7] James G. Berryman. Analysis of approximate inverses in tomography. II. Iterative inverses. *Optim. Eng.*, 1(4):437–473 (2001), 2000.

[8] Jon F. Claerbout. *Basic Earth Imaging (version 2.4)*. Stanford University, 2001.

[9] K. A. Dines and R. J. Lytke. Computerized geophysical tomography. *Proc. IEEE*, 67:1065–1073, 1979.

[10] Leonardo X. Espín Estevez. *A study of the solution of the eiconal equation for the computation of traveltimes*. Tese de Mestrado do Impa, 2003.

[11] P. Gilbert. Iterative methods for the three-dimensional reconstruction of an object from projections. *J. Theor. Biol.*, 36:105–117, 1972.

[12] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. ed. Baltimore. Johns Hopkins University Press, 1996. 3rd edition.

[13] R. Gordon. A tutorial on art (algebraic reconstruction techniques). *IEEE Trans. Nucl. Sci.*, NS-21:78–93, 1974.

[14] J. M. Harris and A. G. Ramm. Inversion of the acoustic well to well data. *Appl. Math. Lett.*, 1(2):127–131, 1988.

[15] Jerry M. Harris and Guan Y. Wang. *Croswell Seismic Geo-tomography*. Seismic Tomography Project, 1995.

[16] Gabor T. Herman. *Image reconstruction from projections: the fundamentals of computerized tomography*. Computer science and applied mathematics. Academic Press, 1980. 3rd edition.

[17] W. Kahan. Gauss-seidel methods of solving large systems of linear equations. *Tese de doutorado*, 1958.

[18] E. W. Kincaid, David R.; Cheney. *Numerical Anlysis: mathematics of scientific computing*. Pacific Grove, 1996. 2nd edition.

[19] Bruce Marion and Satinder. An interview with jerry harris, 2002.

[20] F. Natterer. *The Mathematics of Computerized Tomography*. John Wiley and Sons. 1986. 1st edition.

[21] F. Natterer and F. Wubbeling. *Mathematical in Image Reconstruction*. 2000. 1st edition.

[22] A. M. Ostrowski. On the linear iteration procedures for symmetric matrices. *Rend. Mat. e appl.*, 14:140–163, 1954.

[23] R. Penrose. On best approximate solutions of linear matrix equations. *Proc. Cambridge Philos. Soc.*, 52:17–19, 1955.

[24] M. Rawlinson, N.; Sambridge. Seismic traveltime tomography of the crust and lithosphere. *Advances in Geophysics*, 46(46):81–198, 2003.

[25] James G. Reich. On the convergence of the classical iterative method of solving linear simultaneous equations. *Ann. Math. Statistics*, 20:448–451, 1949.

[26] Gerard T. Schuster. Basics of exploration seismology and tomography. *Stanford Mathematical Geophysics Summer School Lectures*, 1998.

[27] Site. http://www.int.gov.br/novo/menus/mumia.html.

[28] R. Snieder and J. Trampert. Inverse problems in geophysics. *Wavefield inversion*, pages 119–190, 1999.

[29] Gilbert Strang. *Linear Algebra and its applications*. San Diego, 1988. 3rd edition.

[30] José Eduardo Thomas. *Fundamentos de Engenhara de Petróleo*. Editora Interciência, 2001.

[31] Raphael Yuster and Uri Zwick. Fast sparse matrix multiplication. *Proceedings of the 12th Annual European Symposium on Algorithms*, pages 604–615, 2004.

[32] Jorge P. Zubelli. *An introduction to inverse problems. Examples, methods and questions*. 22º Colóquio Brasileiro de Matemática. [22nd Brazilian Mathematics Colloquium]. Instituto de Matemática Pura e Aplicada (IMPA), Rio de Janeiro, 1999. Appendix A by the author and Luis Orlando Castellano Pérez.