ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

RAFAEL VITAL RODRIGUES

TOMOGRAFIA POR ULTRASSOM EM MEIOS HETEROGÊNIOS

SÃO PAULO

2016

**RESUMO**

A proposta do trabalho é projetar um tomógrafo por ultrassom que seja capaz de visualizar as estruturas anatômicas na forma de cortes da seção transversal do tórax, um meio altamente heterogêneo. Devido à heterogeneidade do meio, as ondas sonoras que se propagam no corpo estão sujeitas a três fenômenos físicos, reflexão, refração e difração, o que torna a sua propagação complexa, o que proporciona que as técnicas de tomografia mais robustas e desenvolvidas, como, por exemplo, a dos raios-x, não possam ser aplicadas. Em uma tomografia por ultrassom, diversas fontes emitem separadamente sinais sonoros que passam através de um corpo, e os sinais propagados são coletados por múltiplos sensores, localizados ao redor do corpo. O método desenvolvido, por meio dos sinais coletados e a partir de uma estimativa inicial do corpo, visa a reconstruir iterativamente as propriedades sonoras do corpo, utilizando a técnica de propagação e *backpropagation* (PBP). Cada iteração do método PBP executa quatro passos principais: predição dos sinais recebidos nos sensores, calculados por meio da simulação da propagação das ondas sonoras que partem dos sensores e atravessam o corpo estimado; comparação entre os sinais coletados e preditos, preparando-os para a *backpropagation*; *backpropagation* dos sinais modificados, utilizando o mesmo corpo em que foi executada a propagação direta e aprimorando as propriedades sonoras do corpo, baseado nas ondas propagadas e retropropagadas. As propagações e as retropropagadas das ondas utilizam o modelo pseudo espectral *k-space* (*Kwave*). Como o método é iterativo e requer diversas simulações de propagação de onda, a sua resolução exige muitos cálculos e, consequentemente, um tempo de execução grande. Para sanar esse problema, um dispositivo gráfico (GPU) deve ser usado.

Palavras-chave: Tomografia por ultrassom; GPU; modelo não linear das equações de onda.

Sumário

[1. INTRODUÇÃO 4](#_Toc451717421)

[2. PROPOSTA 6](#_Toc451717422)

[3. O ESTADO DA ARTE 8](#_Toc451717423)

[3.1 Modelo de desdobramentos de raios [#1] 8](#_Toc451717424)

[3.2 Modelo baseado em equações de onda [#2] [#3] [#4] 9](#_Toc451717425)

[3.2.1 Aproximação de Born [#2] [#3] 10](#_Toc451717426)

[3.2.2 Método iterativo [#4] 14](#_Toc451717427)

[3.3 Modelo não linear da equação de onda [#5] 16](#_Toc451717428)

[3.3.1 Método pseudo espectral k-space e outras ferramentas numéricas 18](#_Toc451717429)

[3.3.2 Método pseudo espectral k-space aplicado no problema de propagação de ondas acústicas 21](#_Toc451717430)

[3.3.3 O K-Wave 23](#_Toc451717431)

[4. CUDA - KWAVE 27](#_Toc451717432)

[4.1 Código C++ 27](#_Toc451717433)

[4.1.1 Estrutura 28](#_Toc451717434)

[4.1.2 Classes e Objetos 33](#_Toc451717435)

[4.2 Código CUDA 36](#_Toc451717436)

[4.3 Validação 38](#_Toc451717437)

[4.3.1 Verificação dos resultados 38](#_Toc451717438)

[4.3.2 Análise de Desempenho 46](#_Toc451717439)

[4.3.3 Confronto do tempo de execução 48](#_Toc451717440)

[5. ALGORTIMO TOMOGRáFICO 50](#_Toc451717441)

[5.1 Etapas do Algoritmo 51](#_Toc451717442)

[5.1.1 Propagação e Retro-Propagação (1ª e 3ª etapa) 51](#_Toc451717443)

[5.1.2 2ª etapa (Interpolação) 52](#_Toc451717444)

[5.1.3 4ª etapa (Aprimoramento) 53](#_Toc451717445)

[5.2 Metodologia de teste 54](#_Toc451717446)

[5.2.1 Primeira Etapa 55](#_Toc451717447)

[5.2.2 Segunda Etapa 58](#_Toc451717448)

[5.2.3 Terceira Etapa 59](#_Toc451717449)

[5.3 Testes e Resultados 62](#_Toc451717450)

[6. BIBLIOGRAFIA 81](#_Toc451717451)

# INTRODUÇÃO

Na medicina moderna, uma classe de equipamentos muito usada pelos médicos para obter diagnósticos são os tomógrafos. Os tomógrafos têm como objetivo visualizar as estruturas internas do corpo do paciente de modo não invasivo, causando o menor dano possível ao paciente.

Originalmente os tomógrafos representavam a seção transversal de um objeto através de uma imagem 2D, por isso o nome originado do grego *tomos* (corte, pedaço ou fatia) e *graphein* (grafar). Mas hoje, muitas técnicas foram desenvolvidas de modo que se consegue adquirir a imagem 3D dos objetos.

Os tomógrafos funcionam da seguinte maneira, os transmissores emitem sinais que passam através de um objeto (corpo do paciente), sofrendo modificações durante o processo e, então, são detectados pelos receptores. Por meio das medidas adquiridas nos receptores, diversos cálculos são realizados com o objetivo de estimar como deve ser o corpo do paciente de modo que o par de sinais transmitidos e recebidos faça sentido, segundo as leis físicas e os modelos matemáticos. Esse tipo de problema é comumente chamado de problema inverso.

Os sinais são transmitidos de diversas formas. Na medicina, os tomógrafos mais clássicos usam raios-x para a transmissão, no entanto outras formas como fótons, pósitrons, raios gama, ondas eletromagnéticas e ondas sonoras são usados. Cada tipo de sinal origina uma imagem tomográfica diferente, por exemplo, a imagem tomográfica, gerada com raios x, mostra a absorção de radiação x de cada região do objeto e esta imagem é claramente diferente da imagem gerada por tomógrafos por ultrassom que mostra a velocidade do som em cada região.

Portanto, a variedade de tomógrafos traz grande benefício aos diagnósticos médicos e escolher qual é o melhor tipo de tomógrafo depende da aplicação, já que cada equipamento possui suas vantagens e suas desvantagens, com seus respectivos efeitos colaterais, sua área de atuação e seus custos.

As vantagens da tomografia por ultrassom residem no seu efeito colateral mínimo e no baixo custo, por isso existe um grande interesse em estudá-lo para ampliar a sua gama de utilização na medicina e fora dela. No entanto, as ondas sonoras propagam-se de forma complexa, causando grandes dificuldades em sua aplicação.

A propagação das ondas sonoras sofre efeito de três fenômenos: reflexão, refração e difração. A reflexão ocorre quando as ondas sonoras encontram um obstáculo e parte da sua energia retorna (não necessariamente na mesma direção da onda incidente), enquanto a refração é o fenômeno que descreve o desvio que a onda sofre quando atravessa uma interface de dois meios diversos, por exemplo, a agua e o ar. A difração é o encurvamento sofrido pelas ondas quando ela encontra obstáculos à propagação. Esses três efeitos ocorrem simultaneamente de forma intensa ou moderada segundo o meio que se encontra.

O problema direto, que é o problema da propagação das ondas sonoras, apesar de ser complexo, já foi muito estudado, e equações e métodos que o descrevem são bem conhecidos, robustos e consistentes. Por outro lado, o problema inverso (a tomografia) é um problema que está em grande parte em aberto.

Devido à complexidade dos problemas, diversas simplificações do modelo de propagação de onda podem ser feitas. Por exemplo, em um meio onde o principal fenômeno é a reflexão, pode-se ignorar os outros efeitos e considerar apenas a reflexão, simplificando o modelo e, consequentemente, o nível de complexidade do problema inverso. Esse modelo mais simples é largamente utilizado na medicina, e a sua aplicação mais conhecida é a ultrassonografia de bebês.

No entanto, as aplicações em que se pode considerar apenas o fenômeno da reflexão da onda são limitadas, e, por isso, técnicas que abrangem problemas mais elaborados são necessárias. É justamente, nesse cenário, que se trabalhará, onde os fenômenos de refração e da difração não podem ser ignorados.

Embora existam diversos métodos na literatura que se propõem a resolver o problema inverso da tomografia por ultrassom, ainda não existe uma solução definitiva para o problema. Além disso, a maior parte dos métodos atuais, que conseguem bons resultados, possuem limitações na heterogeneidade do objeto tomografado, pois a velocidade do som, ao longo do objeto, não pode variar muito.

# PROPOSTA

A proposta do trabalho é projetar um tomógrafo por ultrassom que seja capaz de tomografar a seção transversal do tórax, de modo semelhante aos tomógrafos das mamas [#1] (Hormati, Jovanovíc, Vetterli, 2010). A grande diferença é a alta heterogeneidade na região torácica, pois a velocidade do som no pulmão é bem diferente da velocidade nos ossos que é bem diferente da velocidade dos tecidos musculares, assim os efeitos dos fenômenos de difração e de refração intensificam-se.

O projeto concentra-se em desenvolver um método (algoritmo) que resolva o problema inverso em questão e em dimensionar como devem ser dispostos os transdutores (receptores e transmissores) do tomógrafo, mas apenas em um nível teórico, não entrando na viabilidade técnica dele, isto é, como seria a construção do aparelho e quais os componentes elétricos, mecânicos e materiais seriam usados em sua estrutura.

A ideia é desenvolver um algoritmo iterativo que, a partir da estimativa inicial da seção transversal torácica, identifique-a, com uma maior precisão, no algoritmo iterativo, quando muitos problemas diretos são resolvidos (propagação e retropropagação de onda). A resolução desses problemas diretos será efetuada, usando o método pseudo espectral *k-space*.

O *k-wave*, que é uma biblioteca, desenvolvida em *Matlab* e, posteriormente, adaptada para C++, implementa esse método e, no projeto, é empregado. O *k-wave* resolve o problema direto tanto para o caso 2D quanto para o caso 3D. Neste projeto, está-se interessado em obter somente um corte da seção transversal do objeto, ou seja, uma imagem 2D. Apesar de a simulação 2D ser suficiente para ter um problema inverso definido, usar-se-ão as simulações 3D, pois as ondas sonoras, diferentemente dos raios x, não estão restritas a um único plano. Assim, os transdutores estarão dispostos nas paredes externa de um cilindro que cobre o tórax (semelhante a um espartilho), mas apenas a seção transversal no centro desse cilindro será analisada.

O trabalho pode ser dividido em duas etapas. A primeira parte consiste em agilizar o processo de simulação do *k-wave*, usando GPU (*Graphics Processing Unit* - Unidade de Processamento Gráfico), e a segunda parte é a aplicação do *k-wave*, otimizado para criar um método que resolva o problema inverso de tomografia ultrassônica iterativamente.

Na primeira etapa, as atividades a serem desenvolvidas são relativamente simples. Em primeira instância, deve-se compreender como funciona a biblioteca de simulação e, a partir de sua versão C++, estendê-la para usar o poder de processamento do dispositivo gráfico. O *software* deve ser projetado de tal forma que o usuário da biblioteca estendida possa escolher livremente o uso do dispositivo gráfico ou do processador tradicional, qualquer modificação no código precisa ser realizada de modo simples, isto é, basta fazer a substituição de alguns parâmetros.

A biblioteca *k-wave* apresenta-se bem projetada nesse ponto, de modo que a sua modificação não deve trazer grandes problemas. Na realidade, o *k-wave* já possui a opção de utilização do dispositivo gráfico para as simulações, no entanto essa opção não está otimizada.

A otimização será voltada para computadores com placa de vídeo da NVIDEA, usando a plataforma de computação paralela CUDA (*Compute Unified Device Architecture*). Em princípio, o código será desenvolvido para máquinas simples, ou seja, máquinas com poder de processamento relativamente baixo, um computador pessoal com um ou dois dispositivos gráficos de médio porte, nada fora do comum, como, por exemplo, um *mainframe* com dezenas de placa de vídeo ou de equipamentos similares.

Embora a primeira parte seja mais simples que a segunda, se bem feita e se apresentar bons resultados, a sua contribuição para o estado da arte será significante.

A segunda parte é mais desafiadora do que a primeira. O principal objetivo é testar a viabilidade de um algoritmo iterativo muito semelhante ao MIMO, mas usando o *k-wave* para simular a propagação de onda. Inicialmente, criar-se-á o algoritmo para casos simples e paulatinamente acrescentar-se-á complexidade ao sistema. O primeiro caso será de um objeto 2D com baixo índice de heterogeneidade e com muitos transdutores. Uma vez que se tenha sucesso em obter a imagem de ultrassom desse objeto, pode-se alterar a complexidade de três maneiras:

* substituindo o objeto 2D por um 3D, lembrando que a imagem tomográfica de interesse é a seção transversal central do objeto;
* aumentando o nível de heterogeneidade;
* diminuindo o número de receptores e transmissores.

O intuito é simular esses três níveis de complexidade, separadamente e simultaneamente, localizando os limites dos algoritmos e as possíveis explicações para os efeitos e os artefatos que serão encontrados.

# O ESTADO DA ARTE

Os métodos de obtenção de imagens tomográficas por ultrassom podem ser classificados de duas maneiras: métodos que usam a equação da onda e métodos que usam a teoria de desdobramentos de raios (Bent ray model).

## Modelo de desdobramentos de raios [#1]

O modelo Bent ray considera que a propagação do som ocorre de forma análoga aos raios, isto é, são linhas perpendiculares à frente de onda que transportam energia acústica. Desse modo, trata-se a propagação acústica de forma semelhante aos raios óticos, portanto pode-se aplicar o princípio de Fermat (raio segue a menor distância entre dois pontos) e a Lei de Snell-Descartes (refração) para os raios acústicos.

Essa aproximação é válida desde que o comprimento de onda seja pequeno se comparada às dimensões corpo em estudo, ou seja, as frequências das ondas sonoras devem ser altas o suficiente. Em mamógrafos clínicos que aplicam o princípio de Bent-ray, as ondas sonoras incidentes possuem frequência de 1.5MHz.

Um problema que utiliza o modelo Bent ray pode ser formulado, usando o tempo de transmissão entre os pares de transmissores e receptores do tomógrafo. Esse tempo é definido por:

Onde é o caminho de propagação dos raios acústicos (princípio de Fermat e Lei de Snell-Descartes) e é a velocidade do som na posição do objeto. Desse modo, está-se interessado em, a partir das medidas do tempo de transmissão, obter os valores de do corpo. Para tanto, discretiza-se o corpo em N regiões, e cada região possui o seu atraso (inverso da velocidade ). Assim, pode-se modelar o problema como:

Onde é vetor do atraso das N regiões, é o vetor de tamanho M do tempo de propagação de cada medida e é a matriz que indica por quais regiões cada raio (onda sonora) atravessou.

Portanto, a solução que minimiza é a imagem tomográfica do corpo em questão.

## Modelo baseado em equações de onda [#2] [#3] [#4]

Outra classe de algoritmos de reconstrução tomográficas por ultrassom é a baseada na seguinte equação da onda:

Onde é a pressão de campo da onda no ponto e no tempo , é a velocidade do som em e é a forçante do sistema. Vale observar que essa equação não considera a atenuação ou a absorção da onda.

Com a equação da onda, é possível formular dois problemas:

* **Problema direto:** Dados , e as condições iniciais de quer-se descobrir . Esse problema pode ser usado para a propagação da onda ou para a propagação-retrograda (*backpropagation*) da onda. Para a propagação, usa-se, como condição inicial, e avança-se até o tempo , no caso contrário, impõe-se, como condição inicial, e retorna-se até
* **Problema inverso:** Dados e onde é o conjunto de pontos onde se dispõe das medidas de para todo e querer-se descobrir .

Para resolver o problema direto, basta solucionar a equação de onda. Na maioria dos casos, a solução analítica não existe, e uma solução numérica é usada. Assim, quando se resolve um problema numericamente, tem-se de levar em consideração os erros devido à discretização e ao próprio método numérico.

Contudo, aqui, o foco é no problema inverso, cuja solução exata não é única. Para resolver o problema inverso, diversos métodos foram propostos na literatura, então, ver-se-ão, brevemente, os métodos que usam a aproximação de Born (que simplifica a equação de onda), combinado com uma interpolação no domínio da frequência (Transformada de Fourier) e métodos iterativos que, partindo de uma solução aproximada do problema e de diversas soluções do problema direto (propagação e retropropagação), convergem para uma solução mais acurada do problema.

### **Aproximação de Born** [#2] [#3]

Antes de se falar do uso da aproximação de Born para as resoluções dos problemas direto e inverso, interpreta-se a equação da onda por outra perspectiva, isto é, pela equação de Helmholtz.

A partir da equação de onda com o termo forçante com sinal oposto (o que não causa nenhuma perda de generalidade) e admitindo que tanto o campo da onda quanto o termo forçando são separáveis e de frequência única[#7], pode-se inferir que:

A equação obtida representa o problema da propagação da onda em um meio heterogêneo (com número de onda que sofre a ação de uma forçante .

Considere-se, agora, outro problema onde as condições de contorno e as forçantes sejam iguais ao anterior, mas que o meio seja homogêneo, ou seja, com número de onda . Seguindo os mesmos passos e as considerações do problema inicial, tem-se que o novo problema que pode ser descrito por:

Onde a solução numérica de , dado uma forçante suficientemente regular, é conhecida e robusta. Pode-se solucionar esse problema, usando elementos finitos de Galerkin, aplicando as técnicas de resolução de um problema de difusão e transporte. [#9]

Subtraindo o problema heterogêneo do problema homogêneo, tem-se o seguinte: (omitiu-se o parâmetro por simplicidade)

Adicionando e passando o termo para o lado direito da equação:

Por fim, substituindo por por :

Obtém-se, assim, a equação de Helmholtz não homogênea, na qual a sua solução é a convolução de:

Onde é a função de Green, que é a solução da equação diferencial:

No entanto, o problema ainda está de difícil solução, pois depende de que não é conhecido, porque, na realidade, é o que se pretende estimar. Contudo, pode-se usar a aproximação de Born de primeira ordem.

A aproximação de Born de primeira ordem consiste em substituir por , o campo acústico de incidência, pois assume que a diferença entre eles é pequena o bastante. Lembrando que é obtido resolvendo a equação de onda para um meio homogêneo. Deste modo:

Essa equação define claramente o problema direto, pois pode-se obter o campo acústico de perturbação (), isto é, a propagação da onda, através do campo incidente (), do meio heterogêneo () e da função de Green ().

Com o () calcula-se o campo acústico (P) que é igual a:

Fazendo ainda duas considerações adicionais, uma em que se substitua por , que representará as propriedades acústicas do objeto tomografado, e outra onde se impõe que a forçante dos sistemas gere, em um meio homogêneo, uma onda sonora plana, isto é, que o campo incidente () seja igual a onde e Dessa forma, o modelo reduz a:

Indicando a transformada de Fourier de uma função por , onde e **,** e aplicando a transformada na equação, obtém-se:

Dado que a função de Green e sua transformada de função de Green para o caso 2D são respectivamente:

E, sendo a função de Hankel de primeira ordem, pode-se concluir:

Pela equação, nota-se que a transformada de Fourier de é a transformada de Fourier de escalado pela distância de da origem e possui pontos de singularidade quando a distância for igual a .

Até agora, tratou-se apenas do problema direto (no tempo e na frequência), ou seja, tratou-se o caso onde se conhece a onda incidente e o objeto e quer-se descobrir como se comporta. No entanto, o foco é na tomografia, o problema inverso, no qual se conhece a onda incidente e o campo acústico de perturbação () em um conjunto limitado de ponto e quer-se descobrir qual é o objeto .

Vai-se, então, derivar um modo de obter a partir de e de . Para tal, considera-se um cenário onde a direção da onda incidente é igual a e que se tenham sensores que capitem o sinal na posição . Logo, (usando a fórmula da transformada inversa de Fourier):

É possível demostrar, usando integração de contorno com respeito a variável , o seguinte:

Aplicando a transformada de Fourier em , obtém-se:

Assim, por meio da transformada de Fourier de , pode-se estimar os valores de em um arco circular.

A imagem a seguir mostra como são os arcos circulares que sobrepoem a transformada de Fourier do objeto. Vale notar que a direção da onda incidente (a linha tracejada) é genérica, consequentemente os receptores não estão localizados na posição e sim em uma posição perpendicular a onda plana incidente.

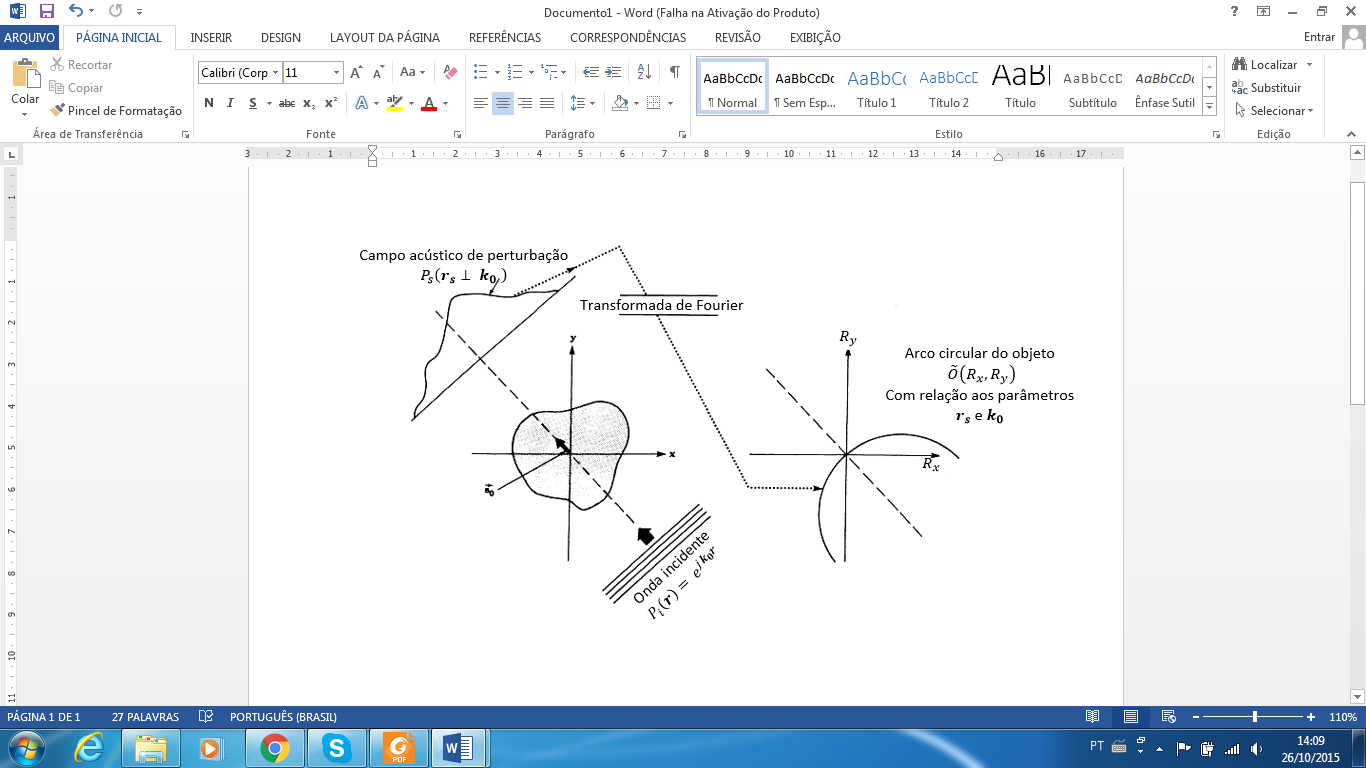


Figura . Projeção de uma onda plana (esquerda) Domina da transformada de Fourier da projeção (direita)

Fonte: [#4]

Portanto, para cada onda incidente de direção , obtém-se um arco de circunferência de diferente. Se se juntar cada um desses arcos de circunferência, ter-se-á uma gama de pontos cujo valor de é conhecido.

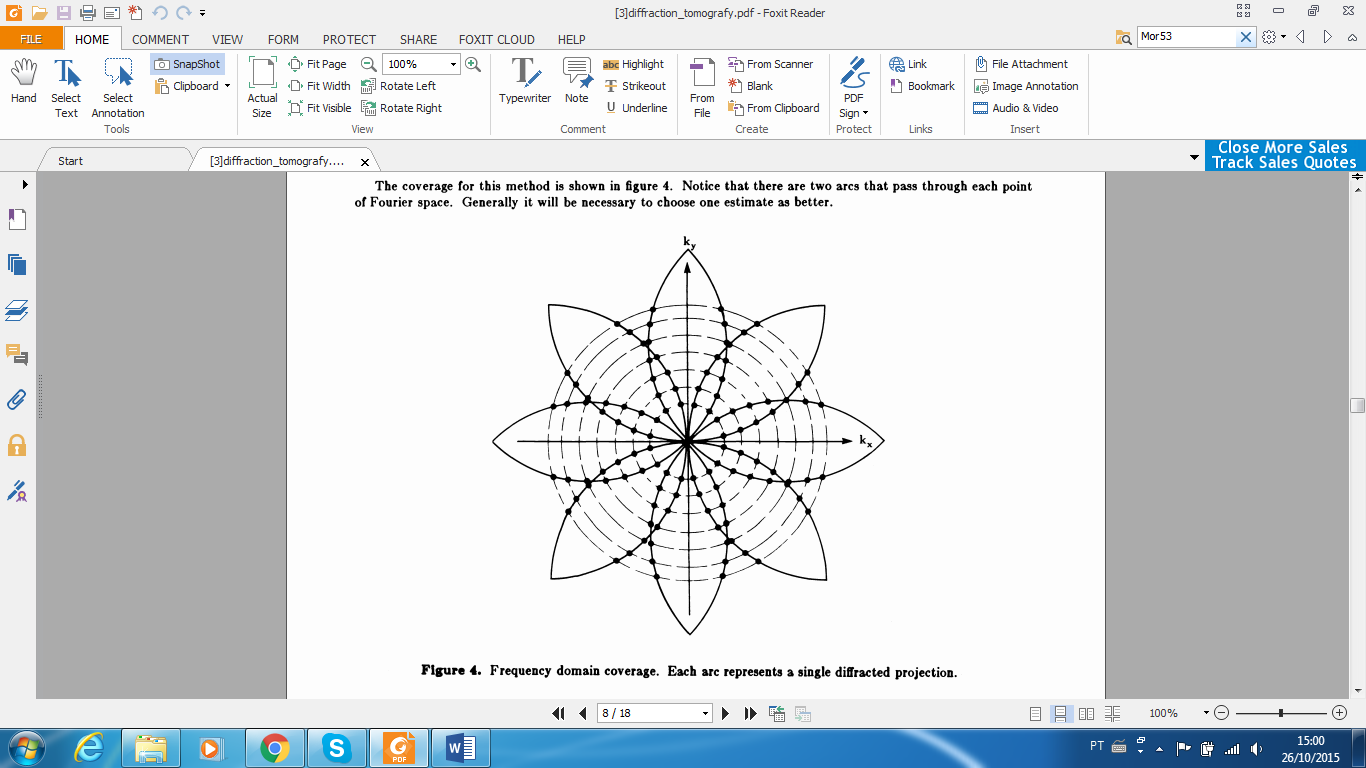


Figura . Os arcos de circunferência é o domínio (em frequência) dos dadas projetado. Já os anéis circulares são as coordenadas onde a interpolação é realizada

Fonte: [#4]

Com esses pontos, realiza-se uma interpolação de modo que seja conhecido em todo o domínio da frequência, assim, pode-se aplicar a transformada inversa de Fourier e obter a densidade acústica do objeto o que resolve o problema inverso, a tomografia.

Um último ponto a se tomar nota é a semelhança deste método de resolução do problema inverso com o Teorema das Projeções usado na tomografia por raio-x. Os dois métodos são similares: obtém-se uma curva de valores na frequência, interpola-se e aplica-se a transformada inversa. A diferença entre os métodos é que, no caso do raio-x (Teorema das Projeções), a curva de valores conhecidos de é uma reta perpendicular a que passa pela origem, ao contrário das ondas sonoros, onde a curva é um arco de circunferência.

### Método iterativo [#4]

Os métodos iterativos são métodos que, ao invés de usar os sinais sonoros dos transmissores e dos receptores em algum tipo de operador inverso, usam os sinais para refinar uma estimativa inicial desse objeto.

Assim, por meio do modelo de propagação de onda e da estimativa da velocidade do som em cada ponto do objeto, faz-se a simulação dos sinais sonoros que seriam recebidos pelos receptores, caso os sinais fossem transmitido através desse objeto estimado. Comparando os sinais simulados e os efetivamente recebidos, verifica-se se a estimativa do objeto é próxima o suficiente da solução, e, caso não seja, usam-se essas medidas (simulada e real) para refinar a estimativa das propriedades sonoras do objeto.

Pode-se representar a situação matematicamente como:

Onde é o operador que descreve a propagação das ondas transmitida pelos transmissores dado o objeto . ,e representa os sinais recebidos nos receptores.

Além disso, o avanço iterativo de é descrito como:

Onde é a imagem tomográfica do objeto, ao passo e é o operador adjunto de que é a derivada de Fréchet de .

A equação de evolução da solução pode ser interpretada da seguinte forma: partindo da solução , faz-se a simulação da propagação das ondas transmitidas, obtendo, assim, os sinais recebidos nos receptores , então, calcula-se a diferença entre os valores simulados e os efetivamente recebidos e, por fim, calcula-se , adicionado o resultado a . O cálculo de representa a etapa de retropropagação da onda.

Para simular o valor recebido nos receptores, deve-se resolver o problema direto, seguindo o modelo da equação de onda. Um modo de modelar o problema direto é considerar que os sinais transmitidos proveem do termo forçante da equação que age sobre a fronteira do objeto , e as condições de contorno são tais que:

Os valores nos receptores são:

E descreve o objeto com a relação:

Na qual é a velocidade do som no ponto ***r*** do objeto, e é velocidade do som no ar.

Por outro lado, a etapa de *backpropagation* não é um processo trivial. Usando a dualidade de operadores, minimização de incremento () e o efeito dos incrementos na equação da onda (), deduz-se o seguinte:

Onde é a solução da equação de onda na qual a forçante é nula, a velocidade do som no meio é e as condições de contorno são:

Para verificar as deduções e para compreender o método interativo formalmente, é fortemente sugerida a consulta do artigo [#4]. O mais importante é compreender que, em cada iteração, dois problemas diretos são resolvidos.

## Modelo não linear da equação de onda [#5]

Até agora, apresentaram-se possíveis soluções para o problema inverso e, em muitas dessas soluções, o modelo da equação de onda é fundamental, principalmente no caso dos métodos iterativos onde muitos problemas diretos (propagação da onda) são resolvidos.

A modelagem da equação de onda tem como princípio uma série de equações diferenciais parciais que descrevem o comportamento da pressão, da densidade e das velocidades das partículas no meio por onde a onda se propaga. Por exemplo, pode-se modelar uma onda com amplitude limitada que percorre um meio homogêneo sem perdas no processo, usando a velocidade acústica das partículas (), a pressão acústica (), a densidade acústica (), a densidade ambiente (), a velocidade isentrópica do som () de modo que as equações físicas respeitem as seguintes equações parciais:

Combinando essa equação, obtém-se a equação de onda, já apresentada anteriormente, em sua forma homogênea (sem o termo forçante).

Embora as duas formulações descrevam a propagação de onda, a primeira formulação é mais vantajosa, seja por razões numéricas, seja por questões de modelagem.

Partindo das EDP de primeira ordem, pode-se modificá-las ligeiramente acrescentando efeitos mais complexos na propagação de onda, como, por exemplo, meios heterogêneos, absorção da energia e não linearidade, resultando nas seguintes equações:

Com essa formulação, surgem 5 novos termos:

* o termo indica o efeito que os deslocamentos das partículas causam na propagação da onda, embora o termo seja influente no sistema, ele pode ser "ignorado" quando se resolvem as equações II e III simultaneamente, pois ele se anula com o termo ;
* é a parcela que mais influencia a propagação da onda, é um operador linear integral-derivativo composto por duas operações que descrevem a absorção acústica e a dispersão. Esses dois efeitos seguem lei de potência em frequência ) e precisam estar ambos presentes por razões de causalidade;
* os parâmetros (fator da lei potência) e (expoente da lei de potência) devem respeitar determinadas condições para serem válidos [#6], o que geralmente ocorre em tecidos biológicos.
* o termo considera a não linearidade convexa entre a velocidade das partículas e a velocidade da onda e
* por fim, é um parâmetro não linear que relaciona a velocidade do som com o efeito de amplitude finita.

Além desses termos, podem-se acrescentar as forçantes (forçante de força) na equação I e (forçante de massa) na equação II.

### Método pseudo espectral *k-space* e outras ferramentas numéricas

Existem muitos modos de resolver numericamente um sistema de equações diferenciais parciais. O melhor método para resolvê-los depende de muitos fatores como, por exemplo, o tamanho do domínio, as propriedades dos objetos, as condições de contorno e as forçantes do sistema.

Os métodos, baseados em diferenças e elementos finitosem geral, exigem 10 *grid points* por comprimento de onda, caso contrário a precisão do algoritmo é insuficiente. Se se forem cumprir esses requisitos, em um caso real, as dimensões do *grid* seriam tais que a memória do computador seria insuficiente para armazenar, e o tempo de execução da simulação seria incrivelmente alto.

Para ilustrar, imagine um transdutor que excite um objeto de com uma onda de frequência . Então, o número mínimo de *grid ponts*, para percorrer o objeto, em cada dimensão, deve ser 10 vezes o número de comprimentos de onda necessário para atravessá-lo, já que o número mínimo de harmônicas para se obter uma boa precisão. Assim, como o comprimento de onda é de e supondo que 3 harmônicas sejam suficientes para obter bons resultados, cada dimensão deve ter aproximadamente 30 comprimentos de onda e, consequentemente, pontos. Considerando agora que se tem que armazenar as informações de velocidade, de pressão e de densidade em matrizes 3D de ponto flutuante, a memória mínima requisitada é de 4GB.

Uma memória de 4 GB em si não é muito grande, mas, se aumentar a frequência da onda ou se quiser um pouco mais de precisão, a requisição da memória será maior e, possivelmente, intangível para computadores não especializados. Além disso, problemas de estabilidade numérica e mal condicionamento podem começar a surgir quando se aumenta o número de elementos das matrizes.

No entanto, se se usar o método espectral *k-space*, ao invés de elementos finitos de Galerkin ou de diferenças finitas, o montante de memória requisitada e o número de cálculos são menores. O método espectral *k-space* combina cálculo espectral das derivadas espaciais com a propagação temporal da onda, calculada no domínio da frequência espacial.

Por exemplo, no caso de uma equação da onda homogênea, isto é,

uma possível formulação para o método espectral é:

Onde é a transformada de Fourier 3D de . Nessa equação, pode-se ver que o laplaciano de foi “substituído” por que é o cálculo espectral das derivadas espaciais, originados pela propriedade da transformada de Fourier que transforma derivada em multiplicação:

Onde e é o vetor de onda.

Já a propagação temporal em frequência pode ser vista por meio da derivada segunda em tempo de , executada com a técnica de diferença finita central.

Realizar a derivada espacial no domínio da frequência traz duas vantagens: uma que já citou-se anteriormente, a redução do número de *grid points* requerido por cada comprimento de onda, e a outra é uma maior precisão nos cálculo da derivada.

A primeira vem do fato que, usando a FFT para calcular a transformada de Fourier, apenas 2 pontos por comprimento de onda são requisitados. Então, tomando como comparação métodos clássicos de diferença finita, tem-se uma redução de 10 pontos por comprimento de onda para 2, o que ocasiona uma redução no uso da memória de 125 vezes.

A segunda vantagem ocorre porque, calculando a derivada no domínio da frequência, todos os pontos do intervalo são levados em consideração, ao contrário de elementos finitos que utilizam apenas elementos que estão na vizinhança.

No entanto, o modelo espectral apresentado exige que o *time step*seja pequeno para controlar o erro introduzido pela derivada no tempo e para garantir a convergência. Contudo, pode-se acrescentar um termo na equação para remover essa limitação do *time step* que é conhecido como operador *k-space*

e deve operar sobre o termo do denominador da equação, assim:

O método espectral combinado com o operador *k-space* é conhecido como método pseudo espectral. O método pseudo espectral, apesar de permitir que o *time step* seja maior, requer um maior uso computacional e introduz perdas na precisão, que, para ser reduzida, exige um *time step* inferior, ou seja, o método pseudo espectral aumenta a margem de estabilidade, mas reduz a precisão.

Outra técnica usada, para se obterem melhores resultados de estabilidade e de precisão, é usar *staggered grids* tanto no tempo quanto no espaço. No tempo, quando se usa *staggered grid*, a equação do tipo transforma-se em que é a equação deslocada. Já a *staggered grid* espacial, embora siga a mesma lógica que a temporal, tem uma ligeira diferença, pois o deslocamento ocorre na frequência, logo, usa-se a propriedade do deslocamento em frequência:

Um último ponto a ser tratado é que, quando se transforma o domínio espacial na frequência, a transformada de Fourier considera que o domínio espacial é periódico, então, se se houver uma onda com amplitude significativa na fronteira do domínio, introduzirá não apenas erros numéricos, mas também erros de interpretação física. Um modo de resolver o problema é considerar um domínio grande o suficiente de modo que a amplitude da onda na fronteira seja pequena o suficiente, contudo, essa abordagem exige muito mais cálculos desnecessários. Então, um modo de resolver o problema é usar uma técnica chamada *Perfectly Matched Layer* (PML).

O *Perfectly Matched Layer* consiste em adicionar um termo de absorção fictício perto da fronteira do domínio de modo que a absorção reduza a amplitude da onda de modo gradual, sem causar efeitos físicos inexistentes, como, por exemplo, reflexão. O PML pode ser modelado com a seguinte equação:

Onde é o termo de absorção fictícia, e deve ser tal que não cause efeito de reflexão e, quando longe das fronteiras, tenha um valor desprezível e, quando perto da fronteira, tenha valores mais elevados. Quando se usa o PML juntamente com a diferença finita em tempo e com *staggered grid*, a equação diferencial discreta é descrita por:

Onde são parâmetros que ditam como se comporta.

### **Método pseudo espectral *k-space* aplicado no problema de propagação de ondas acústicas**

No tópico anterior, viu-se como o método pseudo espectral *k-space* pode ser aplicado à equação de onda não homogenia de segunda ordem e também apresentou-se uma breve introdução das técnicas que trazem maior estabilidade e precisão para as soluções numéricas das equações diferenciais parciais. Agora, aplicar-se-ão essas técnicas no sistema de equações diferenciais não lineares de primeira ordem, introduzido em 3.3.

O primeiro passo é transformar as derivas espaciais tradicionais em uma multiplicação na frequência espacial, mas como as derivadas temporais são resolvidas no tempo, a derivada espacial que foi resolvida na frequência espacial deve retornar para o domínio espacial tradicional. Assim, para executar a primeira equação do sistema , deve-se executar dois passos:

Onde é cada uma das coordenadas .

A primeira observação a se fazer é com relação ao uso dos recursos citados anteriormente: o *staggered grid* no espaço ), no tempo (índice e ); o PML (); a forçante do sistema (); operado *k-space* (), agindo sobre . Vale destacar que o operador é diferente em cada ponto, devido à variação de e de .

A segunda equação do sistema será muito semelhante à primeira equação a menos de um fator dividindo o lado direito (lembrando que o termo não influencia a resolução do sistema):

A terceira equação do sistema , quando discretizada, é igual à sua versão continua (desconsiderando o efeito da mudança de por ). No entanto, como se separam as três coordenadas anteriormente, tem-se que juntá-las. Para tal, usa-se uma simples operação de soma.

Já a alteração de por é um pouco mais complexa e consiste em calcular o laplaciano no domínio da frequência.

Desse modo:

Note que essa equação exige o cálculo de , o que pode não ser interessante do ponto de vista de eficiência. Uma aproximação que simplifica a expressão é considerar a equação linear de conservação de massa e, então, substituir a derivada temporal de pela derivada espacial de , já calculada anteriormente, assim:

A equação do é estável por si só, logo não é necessária a aplicação do operador *k-space* (em .

Outro erro de precisão é cometido quando se substitui por , pois usa-se o valor de e não de que seria o ponto correto da substituição. Em geral, os dois erros cometidos nessa aproximação não influenciam de modo crucial os resultados, mas, se a precisão for imprescindível na aplicação, um *time step* menor resolve parcialmente o problema.

### O *k-Wave*

O *k-wave* é uma *toolbox* para Matlab de código aberto que aplica o método pseudo espectral *k-space* que simula a propagação da onda sonora. A *toolbox* foi planeja para simular a propagação de ondas acústicas em tecidos biológicos, mais especificamente tecidos humano. O *k-wave* interage com meios homogêneos e heterogêneos e resolve as equações de onda tanto com modelos lineares clássicos quanto com modelos mais complexos, abrangentes e não lineares.

Em uma aplicação do *k-wave*, o usuário deve definir quatro tipos de parâmetros para realizar uma simulação: os parâmetros relacionados ao *grid*, ao meio, às fontes/forçantes e aos sensores do sistema.

Os parâmetros do meio são estes: *sound speed(c)*, *density* *()*, BonA , *alpha power* e *alpha coeff*. Com exceção do parâmetro *alpha power*, que é um escalar, todos os outros podem ser escalares ou matrizes, e, caso sejam matrizes, o meio será não homogêneo, e as equações serão resolvidas matricialmente quando necessário. O parâmetro BonA, se definido, faz com que o sistema siga as equações não lineares. Já o parâmetro *alpha coeff* , se definido, considera o operador *(L)*, caso contrário, o operador é desconsiderado.

Os parâmetros relacionados às fontes indicam onde e como as forçantes interagem com o sistema, para tal, têm-se parâmetros de localização e de intensidade. As forçantes podem incidir tanto na densidade quanto em qualquer uma das coordenadas de velocidade .

Os parâmetros dos sensores mostram quais pontos do sistema terão as variáveis armazenadas durante a simulação.

Os parâmetros relacionados ao *grid* descrevem como são o *grid* espacial e o *grid* temporal. O *grid* espacial pode ter 1, 2 ou 3 dimensões (parâmetro dim), com número de pontos de *grid* () e espaçamento () diferentes em cada uma das dimensões, quer sejam espaçados uniformemente () ou não. Já o *grid* temporal tem dois parâmetros que o influenciam: o *time step* () e o número de *steps* ().

O *k-wave* tem uma série de funções que ajudam o usuário a definir os parâmetros, de modo que a simulação tenha os requisitos mínimos para funcionar adequadamente, como, por exemplo, a função *makeTime*, que, baseada nos parâmetros de *grid* espacial e da velocidade máxima do som no meio, calcula a largura do *time step* e o número de *steps*.

Com relação ao *grid* espacial é importante destacar o seu intervalo mínimo admissível (), pois o intervalo mínimo está intimamente ligado à frequência espacial máxima (teorema de Nyquist) que, por sua vez, está ligada à frequência temporal das ondas sonoras do problema. Assim, o intervalo mínimo admissível é:

Onde é a frequência temporal das ondas sonoras, é a velocidade mínima do som no meio e é o número de *grid points* por comprimento de onda, que, neste caso, é 2.

Definidos corretamente os parâmetros, o usuário inicia a simulação, usando apenas um comando e não precisa fazer nenhuma outra operação.

Embora o *k-wave* seja uma ferramenta excelente para a simulação da propagação de ondas acústicas, ele apresenta uma restrição no requisito desempenho, já que o *k-wave* foi desenvolvido em Matlab, uma plataforma que preza mais pela facilidade de programação do que pelo uso eficiente do *hardware*.

Visando a essa limitação da ferramenta, o grupo desenvolvedor do *k-wave* propôs duas soluções para otimizar o tempo de simulação: usar uma linguagem de programação mais eficiente, como, por exemplo, C++ e explorar a capacidade dos dispositivos gráficos do computador.

O código C++, desenvolvido pela equipe, não é uma biblioteca exatamente igual ao *k-wave/*Matlab só programada em C++, mas é um recurso a mais da biblioteca que otimiza a resolução das equações diferenciais e os cálculos numéricos envolvidos. Toda a parte de *interface* entre o usuário e o algoritmo, isto é, a inserção de parâmetros e o controle, são feitos por meio do Matlab, ou seja, existe uma dependência dos códigos C++ e dos códigos Matlab.

Para resolver o problema, usando o código C++, a única diferença para o usuário é que, ao invés de iniciar a simulação com o comando *kspaceFirstOrder3D*, deve usar o comando *kspaceFirstOrder3DC*. O próprio código é responsável por copiar as informações do “ambiente Matlab “para o “ambiente C++”. A *interface* do Matlab com o C++ é feita por meio de arquivos de entrada e de saída que usam o formato *Hierarchical Data Format HDF5.*

Embora exista uma dependência entre C++ e Matlab, ela pode ser contornada, pois pode-se rodar o código C++ a partir dos arquivos de configuração*.* Assim, pode-se usar o Matlab para gerar arquivos de configuração e, por meio do terminal, executar o código C++, passando, como entrada, o arquivo de configuração criado. Mas, nesse caso, perdem-se alguns recursos oferecidos pelo Matlab, como, por exemplo, a interface gráfica.

O suporte do *kwave* ao dispositivo gráfico acontece por meio de *toolboxes* do Matlab que interpretam o código que foi escrito para rodar em um processador serial e que o “traduz” para rodar no dispositivo gráfico. Mas, como as *toolboxes* são desenvolvidas para interpretar códigos genéricos, as suas traduções não são otimizadas, e o ganho de desempenho pode ser baixo ou nem existir. As *toolboxes* que fazem essa tradução são as seguintes: *MATLAB Parallel Computing Toolbox, Accelereyes Jacket* e *GPUmat*. Para o usuário instruir o *k-wave* a executar a simulação, usando o dispositivo gráfico, e não a CPU, ele deve alterar dois parâmetros na chama da simulação. No caso de querer usar a *MATLAB Parallel Computing Toolbox*, os parâmetros adicionais são *DataCast* seguido de *gpuArray-single*.

# ***CUDA – K-WAVE***

O *k-wave* tem duas vertentes que melhoram o tempo de execução da simulação: uma que usa o dispositivo gráfico por meio das *toolboxes* do Matlab e outra que “transfere” as operações custosas de cálculos para uma linguagem de programação mais eficiente (C++).

O que se fará é combinar essas duas vertentes, quando, usando uma linguagem mais eficiente, desenvolver-se-á um código que resolva o problema de propagação da onda e que use eficientemente o dispositivo gráfico.

A ‘linguagem de programação’ escolhida para essa tarefa é o CUDA. O CUDA é uma plataforma de computação paralela e um modelo de programação criado pela NVIDIA que permite que o dispositivo gráfico da NVIDIA opere com pontos flutuantes. Além disso, o CUDA está relacionado com a eficiente linguagem de programação C++.

Portanto, partir-se-á do código C++, já desenvolvido pela equipe do *k-wave*, alterando-o onde necessário de modo que as funções e os cálculos custosos sejam executados no dispositivo gráfico, enquanto que a interface de entradas dos dados permaneça a mesma.

Assim, o primeiro passo deve ser a compreensão completa do algoritmo C++, sua interface, seus objetos e como ele se comunica com o Matlab.

## Código C++

Dividir-se-á a explicação do código em duas partes: uma voltada ao algoritmo de resolução da equação diferencial e outra voltada à apresentação das classes que compõe o algoritmo. Com essa informação em mãos, estar-se-á apto a alterar o código onde for necessário, de modo que ele opere, usando o dispositivo gráfico como a fonte principal de processamento.

### Estrutura

O algoritmo é dividido em 4 fases.

**Fase de criação e de carregamento**: Nesta fase, duas operações são realizadas. A primeira operação é responsável por criar os objetos e por reservar a memória de todas as variáveis que são usadas ao longo do algoritmo, independentemente da homogeneidade do meio ou da uniformidade do *grid*. Todas as informações, que o algoritmo precisa, isto é, quais variáveis são usadas e o tamanho de cada variável, são conhecidas a *priori* e estão contidas no arquivo de configuração *HDF5.*

A segunda operação é responsável por carregar as informações contidas no arquivo de configuração *HDF5* para a memória do sistema. As informações carregadas podem ser tanto as condições iniciais da simulação quanto o estado das variáveis armazenadas em arquivo de *checkpoint*.

O *checkpoint* é um recurso na qual o programa faz *backups* dos estados das variáveis do algoritmo durante a simulação, de modo que, caso ocorra algum problema técnico e a simulação seja interrompida, possa-se reiniciar a simulação a partir do *checkpoint*. Devido ao grande volumo de dados, o *checkpoint* não pode ser salvo a cada *time step* simulado e deve ser executado após um número coerente de iterações.

**Fase de pré-processamento:** A fase de pré-processamento executa as tarefas que são computadas uma única vez e que influenciam diretamente na fase principal. As suas tarefas são estas:

* caso os sensores, os transdutores e as fontes usem máscaras para especificar suas localizações, os índices (as coordenadas) estão com notação que partem do 1, mas, para funcionar em C++, os índices devem partir do zero.
* cálculo de ; esta divisão é feita na fase de pré-processamento de modo que ela seja efetuada apenas uma vez durante a simulação e não em toda iteração. A tabela a seguir mostra, em quais casos, o pré-processamento é vantajoso e aplicável e quantas operações estão envolvidas. É importante destacar que, devido ao *staggered grid* em espaço, cada ponto do *grid* possui quatro (um para cada coordenada da velocidade e um para o cálculo dos mas apenas os referente às velocidades são pré-calculados, por isso o fator 3 na tabela)**.**

|  |  |  |
| --- | --- | --- |
|  | Grid | Operações |
| Escalar (homogêneo) | Uniforme | 3 |
| Escalar (homogêneo) | Não uniforme | Não aplicável |
| Matriz (heterogêneo) | Uniforme |  |
| Matriz (heterogêneo) | Não uniforme |  |

Claramente o uso do dispositivo gráfico é útil nos casos em que é heterogêneo. Outro ponto relevante é que, no caso em que é homogêneo e o grid é não uniforme, o pré-cálculo poderia ser feito, mas, se fosse feito, ocuparia mais espaço na memória, pois passaria de um escalar para uma matriz, desse modo, o *trade-off* ganho em computação x memoria usado não compensa.

* geração da matriz do operador *k-space* calculada em pontos. Logo, o uso da GPU será vantajoso.
* caso seja usada a lei de absorção, os valores de , e podem ser pré-calculados. Lembrando que e são sempre matrizes de dimensões pontos e , podem ser escalares ou matrizes, dependendo se o meio é homogêneo ou heterogêneo. Como nos outros casos, os cálculos são matriciais e podem ser efetuados no dispositivo gráfico eficientemente.
* cálculo de . Esta é a última tarefa executada pelo pré-processamento e substitui o valor de pelo seu quadrado na memória. Tal ação tem por objetivo evitar cálculos desnecessários de , já que, a partir desse ponto, o valor de não é mais usado.

**Fase principal:** a fase principal é onde o algoritmo resolve o problema da propagação da onda, a fase principal tem caráter repetitivo condicionado pelo *looping* *for* que executa as interações até atingir o tempo especificado.

O *looping* principal é dividido em 7 operações:

1. cálculo de ;
2. adição da forçante ;
3. cálculo de ;
4. cálculo de ;
5. adição da forçante ;
6. cálculo de ;
7. salvamento dos dados dos sensores.

A **primeira operação** que calcula equivale às equações I e II do sistema e, portanto, possui duas etapas. Na primeira etapa, calcula-se a transformada de Fourier 3D de , realiza-se uma série de multiplicações matriciais ponto a ponto no domínio de Fourier e, por fim, obtém-se por meio da transformada inversa de Fourier. Esse processo é realizado três vezes no caso 3D.

A segunda etapa consiste em atualizar o valor de , usando o valor calculado na iteração anterior (), o valor calculado na etapa anterior e a condição de fronteira PML. No entanto, dependendo se o meio é homogêneo, ou não, e se o grid é uniforme, ou não, as operações são ligeiramente diferentes. São três funções diferentes que executam essa etapa, a tabela a seguir mostra exatamente as condições de cada função. Porém, independente do caso, as funções aplicam-se em matrizes, e o uso do dispositivo gráfico é vantajoso.

|  |  |  |
| --- | --- | --- |
|  | Grid | Função |
| Escalar (homogêneo) | Uniforme | *scalar\_uniform* |
| Escalar (homogêneo) | Não uniforme | *scalar\_nonuniform* |
| Matriz (heterogêneo) | Uniforme |  |
| Matriz (heterogêneo) | Não uniforme |  |

A **segunda operação** é simples e consiste apenas na aplicação da forçante , quer seja por condição de Dirichlet, quer seja por adição. Embora as operações sejam matriciais, a intensidade não é alta o suficiente, e a GPU não acrescenta um *speed up* relevante.

A **terceira operação** é muito semelhante à primeira etapa da primeira operação e equivale à terceira equação do sistema. Portanto, para cada uma das coordenadas, são calculados a transformada de Fourier 3D de , uma série de multiplicações matriciais ponto a ponto e uma transformada inversa. Além disso, no caso em que o *grid* não é uniforme, uma operação extra de multiplicação matricial ponto a ponto de com os parâmetros do *grid* é executa. Dada à natureza matricial, o dispositivo gráfico beneficia-se nas operações:

A **quarta operação** é equivalente à equação quatro do sistema e é semelhante à segunda etapa da operação 1. Essa operação possui quatro variantes que, estruturalmente, não são muito distintas entre si. As variantes surgem de duas situações: da linearidade ou não do modelo e da homogeneidade ou não do meio com relação à .

A **quinta operação** é análoga à segunda operação, com a diferença da entrada e de saída, que são referentes aos ao invés dos .

A **sexta operação** equivale às equações V, VI e VII do sistema e, consequentemente, é relativamente complexa, pois exige muitos cálculos. Essa operação possui 8 variações, cada uma delas é significantemente diferente uma da outra, e uma explicação separada é conveniente. Na realidade, são quatro grandes variações, e cada variação possui duas outras variações mais simples. Como em todos os casos, cálculos matriciais são usados, e a GPU beneficia-se nas operações:

* a primeira variação ocorre quando se usa o modelo linear sem absorção. Este é o modelo mais simples, e a equação mostra os cálculos exatos que são efetuados, lembrando que pode ser uma matriz (meio heterogêneo) ou um escalar (meio homogêneo).
* a segunda variação ocorre quando se usa o modelo não linear sem absorção. A equação que rege essa condição é muito semelhante à anterior apenas com uma parcela a mais que calcula . Lembrado que e podem ser matrizes (meio heterogêneo) ou escalares (meio homogêneo).
* a terceira variação ocorre quando se usa o modelo linear com absorção. A equação que rege essa condição é mais custosa, pois compreende o operador que realiza cálculos no domínio da frequência, logo são efetuadas transformadas de Fourier diretas e inversas. Vale lembrar que e foram pré calculados que e estão armazenados na memória e que e podem ser matrizes (meio heterogêneo) ou escalares (meio homogêneo).
* a quarta variação ocorre quando se usa o modelo não linear com absorção. Esta variação é uma combinação das variações dois e três. A equação que a rege é

A **sétima operação** consiste apenas em salvar as informações dos sensores em um arquivo *HDF5.* Os sensores são pontos do *grid* onde as variáveis são salvas em todas as iterações. Porém, existe um problema quando se deseja salvar as velocidades das partículas , pois, devido ao *staggered grid*, as velocidades são deslocadas do ponto central do *grid*, dessa forma, se se deseja salvar o valor “certo” da velocidade, deve-se calcular o deslocamento dessa velocidade para o centro do *grid*.

Este cálculo é complexo, pois exige que cada coordenada da velocidade volte para o domínio da frequência, por meio da transformada de Fourier 1D, que seja multiplicada pelo fator de deslocamento e, por fim, retorne ao domínio do tempo, para, então, ser salva no arquivo *HDF5.* Como esse processo exige poder computacional, a placa de vídeo é vantajosa.

No entanto, salvar as componentes dos sensores é uma tarefa complicada para o dispositivo gráfico, pois exige que as informações, contidas na memória de dispositivo, sejam copiadas para o computador, que, por sua vez, salva as variáveis no arquivo *HDF5.* Contudo, o CUDA possui um recurso chamado *stream* que permite que o dispositivo gráfico faça transferência de informação entre a GPU e o PC simultaneamente ao processamento dados na GPU. Portanto, a tarefa que salva as informações não causa prejuízos no desempenho do algoritmo.

**Fase de pós-processamento:** a fase de pós-processamento é bem simples e consiste de salvar certas informações em arquivos HDF5 para posterior análise, seja na forma de *checkpoints*, seja na forma de documento final. Nos dois casos, a única tarefa que o dispositivo gráfico deve fazer é copiar as informações que estão na sua memória para o *host* (CPU) e o próprio código C++ cuida de salvar as variáveis no HDF5.

### Classes e objetos

O algoritmo C++ possui as seguintes classes de objetos:

* ***MatrixClasses*:** não é uma classe, mas sim um conjunto de classes que estão fisicamente contidas na pasta *MatrixClasses*. Os objetos desta classe representam as variáveis matriciais do problema. As matrizes podem ser tanto índices inteiros, números reais ou números complexos. Os métodos destas classes executam as tarefas de acesso de leitura e escrita e de alocação de memória.

Os objetos de número complexos, além dos métodos de acesso e alocação, possuem métodos que executam a transformada de Fourier 1D e 3D. Já a classe UXYZ\_Matrix, extensivamente aos métodos comuns, possui métodos que executam partes do algoritmo, pois mais especificamente computa a equação II do sistema discretizado.

* **TParameters**:esta classe gera um objeto *singleton* que armazena os parâmetros que regem a simulação, entre eles, podem-se destacar os parâmetros que dizem a qual tipo de equação será usada, com ou sem absorção, linear ou não, homogêneo e heterógeno, além dos parâmetros que especificam o tamanho do domínio e do *grid* e os seus respectivos incrementos.
* **HDF5**:esta classe é um *wraper* da biblioteca HDF5 para o código C++ desenvolvido. A classe tem como objetivo fazer a *interface* entre o Matlab e o código C++, pois a *interface* é realizada por meio de um arquivo ‘texto’ do tipo HDF5, ou seja, a passagem é realizada, utilizando o disco rígido.
* **MatrixContainer**: classe *singleton* que contém os objetos do tipo ***MatrixClasses*,** isto é, todos os objetos que representam alguma grandeza física na simulação, velocidade, pressão, densidade, *kappa* e outras. Os métodos desta classe são responsáveis pela criação das matrizes, com as suas respectivas dimensões e tipos, pelo carregamento das informações oportunas e pelo acesso das matrizes pela classe principal.
* **KSpaceFirstOrder3DSolver**: é a classe principal do problema, ou seja, é a classe que executa o algoritmo em si, possui métodos que chamam os métodos das classes subordinadas (carregamento, criação, salvamento). Mas o principal método é o **compute** que computa todas as fases do algoritmo descritas no item 4.1.1.

O pseudocódigo do método compute é o seguinte:

void TKSpaceFirstOrder3DSolver::Compute(){

PreProcessingPhase();

Compute\_MainLoop();

PostPorcessing();

}

Por sua vez, o pseudocódigo do método Compute\_MainLoop é:

void TKSpaceFirstOrder3DSolver::Compute\_MainLoop(){

while (tempo final não for atingido) {

Compute\_uxyz();

Add\_u\_source();

Compute\_duxyz();

if (modelo não linear){

Compute\_rhoxyz\_nonlinear();

}else{

Compute\_rhoxyz\_linear();

}

Add\_p\_source();

if (modelo não linear ()){

Compute\_new\_p\_nonlinear();

}else{

Compute\_new\_p\_linear();

}

if (tempo inicial){

Calculate initial pressure();

}

StoreSensorData();

Tempo++;

}

}

Internamente, os métodos que compõe o método Compute\_MainLoop(), isto é, os métodos Compute\_uxyz() ,Compute\_rhoxyz\_nonlinear() e os outros, possuem variações segundo a homogeneidade e a absorção do meio.

## Código CUDA

Um código CUDA pode ser dividido em 4 tarefas. A primeira delas é o armazenamento das variáveis no dispositivo gráfico, isto é, o código faz uma cópia das variáveis que estão na memória do computador para a memória do dispositivo, cuja operação é bem simples e consiste em apenas algumas linhas de códigos.

A segunda tarefa consiste no complementar da primeira, ou seja, é a cópia das variáveis que estão no dispositivo gráfico para o computador.

A terceira tarefa é relativa ao processamento dos dados já alocados na placa de vídeo. Esta é uma tarefa mais complexa, pois o modo como as operações (fft, multiplicação, soma) são executadas no dispositivo gráfico são diferentes de como são executados nos processadores tradicionais. As operações devem aproveitar ao máximo o alinhamento da memória e dos *threads* a fim de se obter melhor desempenho. Quando uma função é executada no dispositivo gráfico, ela é chamada de *kernel* e, quando ela é executada, utiliza-se o termo lançar o *kernel*.

Por fim, tem-se a quarta tarefa que é responsável por controlar quando as tarefas 1, 2 e 3 são realizadas.

Felizmente o código C++, desenvolvido pela equipe do *k-wave*, possui essas quatro tarefas bem definidas o que facilitou, significantemente, a adaptação do código C++ para o código CUDA.

A adaptação CUDA do *k-wave* possui os seguintes códigos/objetos:

* **MatrixContainerG:** objeto que herda de *matrixContainer* e é responsável por alocar e desalocar a memória no dispositivo gráfico e por copiar as variáveis do computador para a placa de vídeo. Logo, executa a primeira e a segunda tarefa. O funcionamento deste objeto é bem simples, já que foi aproveitada a estrutura do *matrixContainer*. Os seus principais métodos são os seguintes:
* *createGpuPtr()* – aloca todos os vetores que serão utilizados no dispositivo gráfico. A decisão de quais vetores são alocados foi feita por *matrixContainer*;
* *freeGpuPtr()* – libera as memórias alocadas;
* *copyHostDevice(TMatrixID MatrixID)* – copia o vetor do computador (*host*) para o dispositivo gráfico (*device*), baseado no nome do vetor;
* *copyDeviceHost(TMatrixID MatrixID)* – copia o vetor do dispositivo gráfico (*device*) para o computador (*host*), baseado no nome do vetor;
* *getGpuPtr(TMatrixID MatrixID)* – obtém o ponteiro do vetor alocado no dispositivo gráfico, baseado no nome do vetor. Este método é usado para selecionar o vetor que será processado durante o *kernel*.
* **Kernel Config:** configura o número de *thread* por bloco e o número de blocos a serem utilizados quando lançamos os *kernels*. Esta função considera fisicamente o modelo do dispositivo gráfico e decide os parâmetros que oferecem o melhor desempenho.
* **KSpace3DG:** classe principal do código que executa a terceira e quarta tarefa, ou seja, é responsável tanto pelo controle e lançamento dos *kernels*, quanto pela definição desses *kernels*. Esta classe é análoga à classe *KSpace3D* e possui praticamente as mesmas funções, porém, ao invés das funções executarem as operações no processador do computador, executa as operações na placa de vídeo.

Os *kernels* podem ser divididos em dois tipos: os *kernels* que executam a transformada de Fourier e os *kernels* que executam as operações básicas (soma, multiplicação e divisão).

As operações de FFT são realizadas por meio da biblioteca cuFFT. Anteriormente, eram feitas com a biblioteca fftw3f. Os funcionamentos das duas bibliotecas são muito semelhantes e a adaptação é fácil.

Já as operações básicas são feitas em baixo nível, isto é, os *kernels* foram modelados especificamente para otimizar as operações envolvidas, alinhando a memória perfeitamente de modo a realizar o menor número possível de leituras e de escrita.

O modo de funcionamento dos *kernels* possui 4 variações, segundo o tipo de operação vetorial requisitado. Os tipos de operação vetorial são os seguintes:

* **Tipo 1**: quando um vetor 3D é operado com outro vetor 3D, posição a posição. Neste caso, os vetores já estão alinhados, logo um único ‘*for*’ resolve todo o problema.
* **Tipo 2**: quando um vetor 3D é operado com outro vetor 1D () na coordenada de modo que () seja igual a:

Observe que se deve fazer apenas uma leitura de para cada leituras de , ou seja, quando realizar a operação, deve-se ter dois laços *for* encadeados, o primeiro para ler a coordenada de e o outro para as coordenadas e de .

* **Tipo 3**: quando um vetor 3D é operado com outro vetor 1D () na coordenada de modo que () seja igual a:

Análogo ao tipo 2, deve-se ter dois laços *for* encadeados, no entanto, este caso apresenta uma dificuldade a mais, pois logo após ler , tem-se que transpô-lo, já que representa, na prática, uma coluna, mas, ao ler na memória do dispositivo, ele é uma linha. Então, tem-se que ler em um laço *for,* transpô-lo e, então, operar para cada e de

* **Tipo 4:** quando um vetor 3D é operado com um outro vetor 1D () na coordenada de modo que () seja igual a:

As operações do tipo 4 são executadas de modo análogo às operações do tipo 3.

## Validação

A validação do *K-wave* otimizado para rodar em GPU da *nvidia* está dividida em duas etapas. Na primeira etapa, far-se-á a verificação da consistência dos resultados simulados, comparando as simulações executas por meio dos códigos escritos em Matlab e C++, otimizado com OMP com o código desenvolvido em CUDA.

Na segunda etapa, analisar-se-á o tempo de execução dos códigos, verificando se o dispositivo gráfico é um bom instrumento para diminuir o tempo de simulação da propagação de uma onda.

### Verificação dos resultados

A verificação dos resultados consiste em comparar, em cada iteração, os valores das diversas variáveis (velocidade da partícula , pressão e densidade ) e dos diversos modelos (linear, não linear, com absorção ou sem absorção) e em garantir que os códigos desenvolvidos pela equipe do *K-wave*, consequentemente testado e aprovado pela comunidade, e o código desenvolvido em CUDA tenham exatamente o mesmo resultado.

A metodologia escolhida baseia-se em três baterias de testes.

A primeira bateria testa o modelo mais complexo, isto é, o modelo não linear com absorção em meio não homogêneo em e , e, a cada instante de tempo, compara os valores de (nos três eixos), de (nos três eixos) e de (em todo o domínio).

Essa primeira bateria, foi implementada da seguinte maneira: no código CUDA desenvolvido, o algoritmo executa, paralelamente e “sincronamente”, as instruções do código C++ (com OMP) e do código CUDA, assim, a cada novo valor de ou calculado, uma comparação ponto a ponto é executada, e, caso alguma diferença seja notada, o programa emite um alerta.

Ao comparar as variáveis, certo cuidado é requisitado, pois elas são do tipo *float***,** logo diferentes arquiteturas e diferentes sequências de cálculos comutativos e associativos geram resultados levemente diferentes, dessa forma, uma comparação *bit* a *bit* é impossível. Logo, deve-se usar um critério baseado na diferença relativa para concluir se dois números são iguais ou não. O valor de foi escolhido como limiar da condição de igualdade, e a diferença relativa foi calculada por meio da seguinte fórmula:

Superado com sucesso a primeira bateria, inicia-se a segunda bateria, que visa a testar os diferentes tipos de fontes do sistema. As fontes do sistema podem ser de quatro tipos:

* pressão inicial, usada para a simulação de pulsos acústicos, a função principal do *k-wave*;
* fonte de pressão ou de massa. É uma fonte que, a cada instante de tempo, define um valor de pressão para o ponto no qual fonte pertence. As fontes podem ser *dirichlet* ou aditivas, além disso, todas as fontes do sistema podem impor os mesmos valores de pressão ou valores diferentes para cada uma delas. Logo, têm-se quatro variações para as fontes de pressões, na qual todas devem ser testadas;
* fonte de velocidade das partículas ou de força. Análogo à fonte de pressão, mas, ao invés de pressão, controla a velocidade das partículas nas três coordenadas;
* transdutores: classe escrita em Matlab que facilita o processo de criação das fontes do sistema. Todas as funcionalidades estão contidas na própria classe. A interação da classe com o código ocorre de forma direta sem a presença nenhuma bifurcação (condição if). Portanto, apenas um teste deve ser realizado a fim de comprovar a sua boa funcionalidade.

Na segunda bateria, testa-se cada uma das fontes, usando os *scripts* exemplos do *k-wave*. As simulações foram executadas tanto no Matlab quanto no código desenvolvido e os valores das pressões recebidos nos sensores foram comparados.

A excitação do modelo por pressão inicial foi testada por meio do exemplo *example\_ivp\_3D\_simulation*, que consiste na propagação de duas esferas de pressão e de um *array* de sensores retilíneos. Os resultados do teste estão representados na figura 4.1 e na figura 4.2.

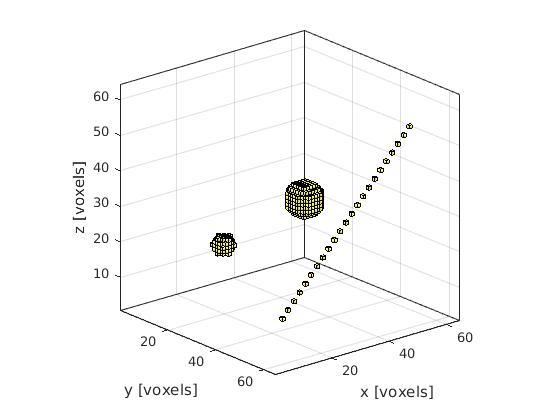
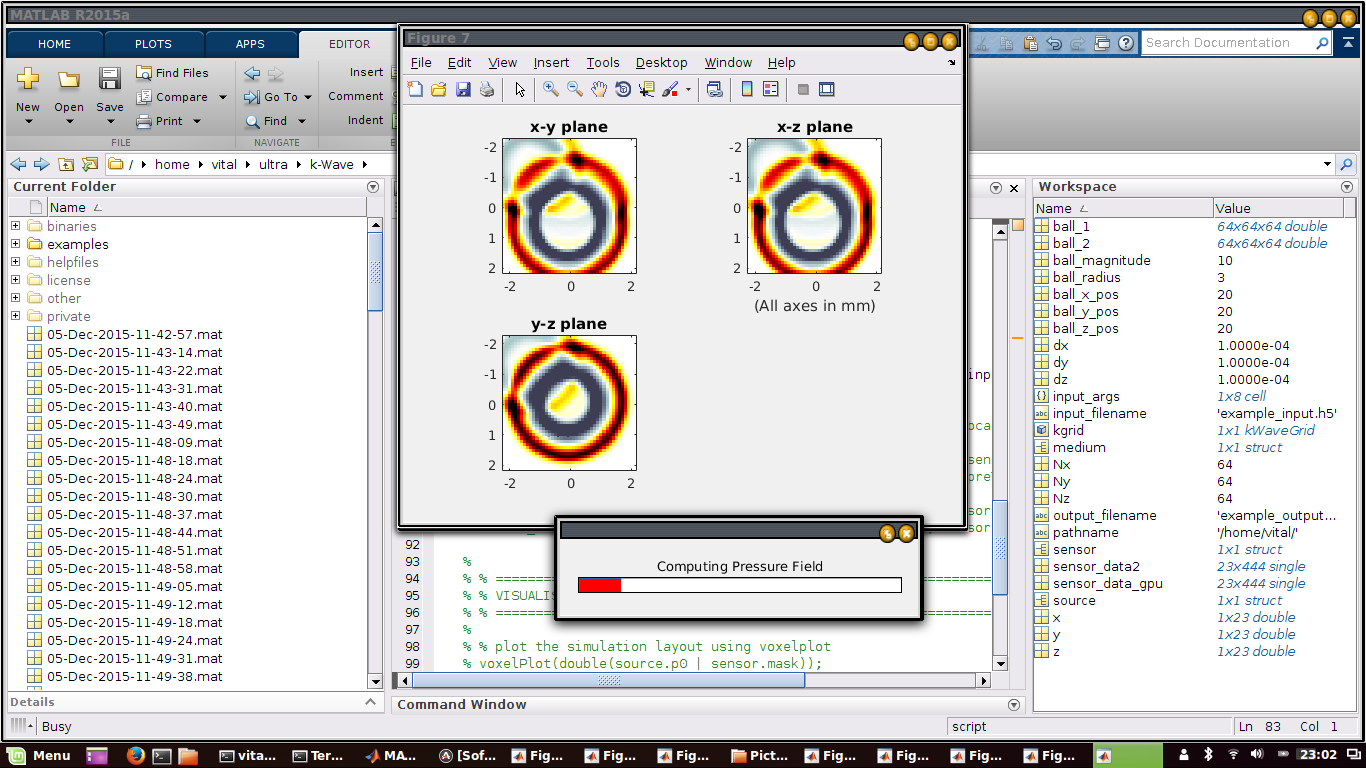
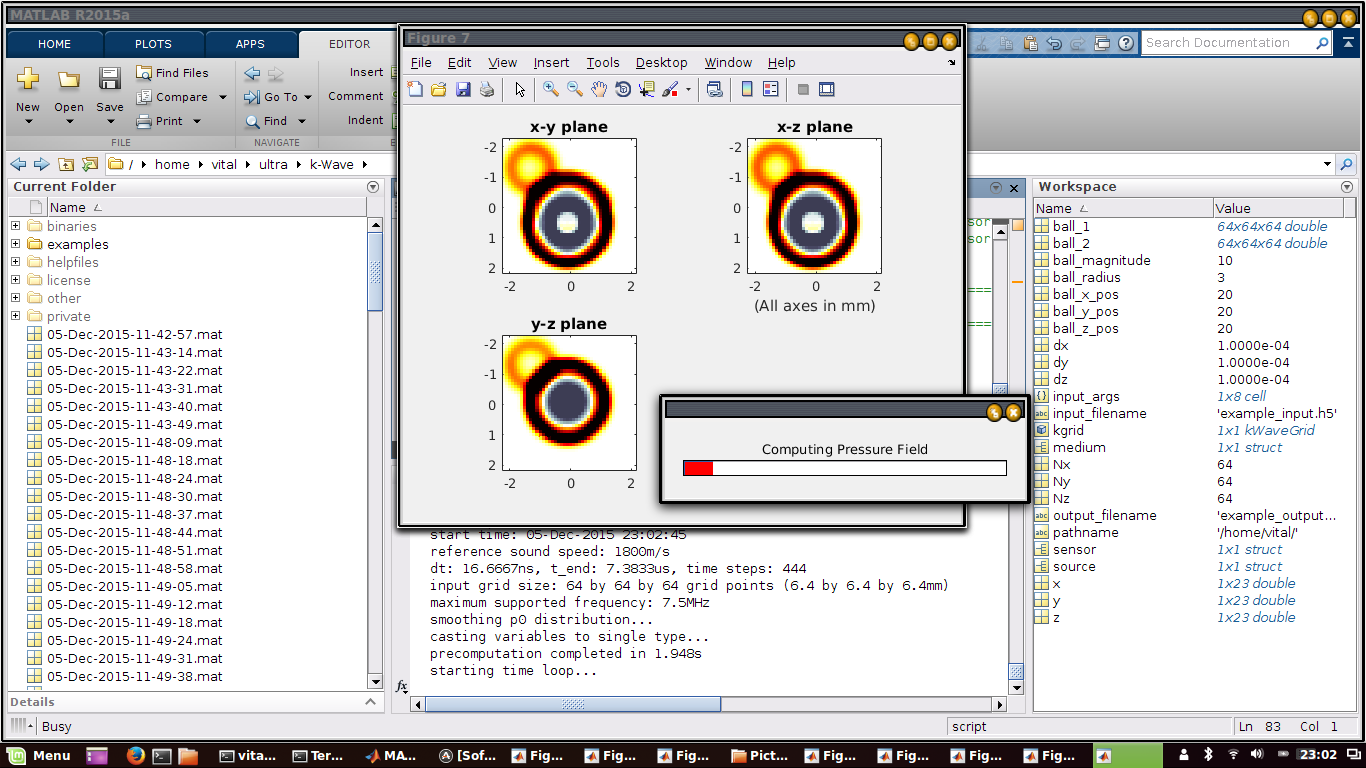


Figura 4. – Simulação da propagação da onda resultante de duas esferas de pressão inicial a) Grid 3D que mostra a localização das esferas de pressão inicial e o array de sensores colineares. b), c) e d) mostram a propagação da onda em três instantes de tempo diferentes, onde

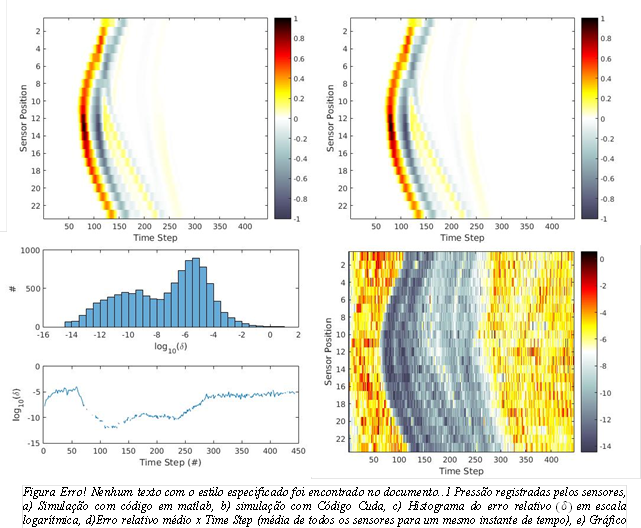


Figura . Pressão registradas pelos sensores, a) Simulação com código em matlab, b) simulação com Código CUDA, c) Histograma do erro relativo em escala logarítmica, d)Erro relativo médio x Time Step (média de todos os sensores para um mesmo instante de tempo), e) Gráfico 2D com o erro relativo em escala logarítmica.

A figura 4.2 indica que os resultados das duas simulações são muito semelhantes e uma análise visual não é possível para concluir a qualidade da simulação com o dispositivo gráfico. Pelo histograma, vê-se que, na maioria dos casos, o erro relativo é menor que , divergindo um pouco do erro de que é o que se esperava. Contudo, o erro ligeiramente maior é totalmente aceitável, pois, na figura 4.2, constata-se que onde o sinal é mais significativo, isto é, onde o sinal tem o maior módulo, o erro relativo é muito baixo, menor que .

A análise da fonte, usando transdutores, é muito parecida com a anterior, tanto que se usou o exemplo do *k-wave, ‘example\_us\_beam\_pattern’*, e compararam-se os valores de pressão nos receptores. Os erros, nesse caso, foram todos menores que , garantindo, assim, que a implementação dos transdutores como fonte está correta.

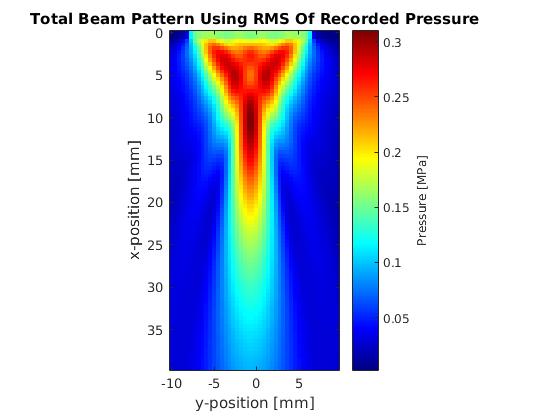
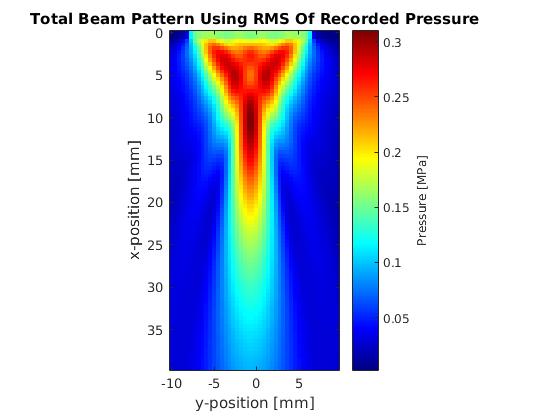
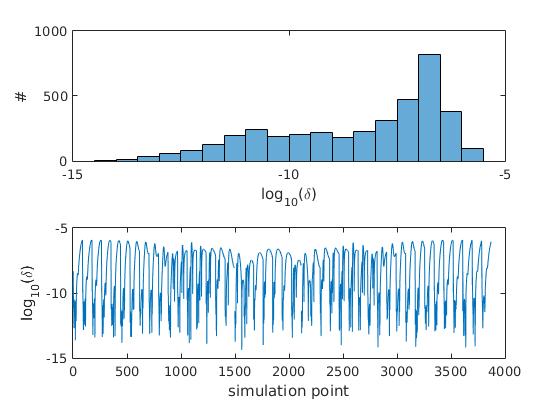
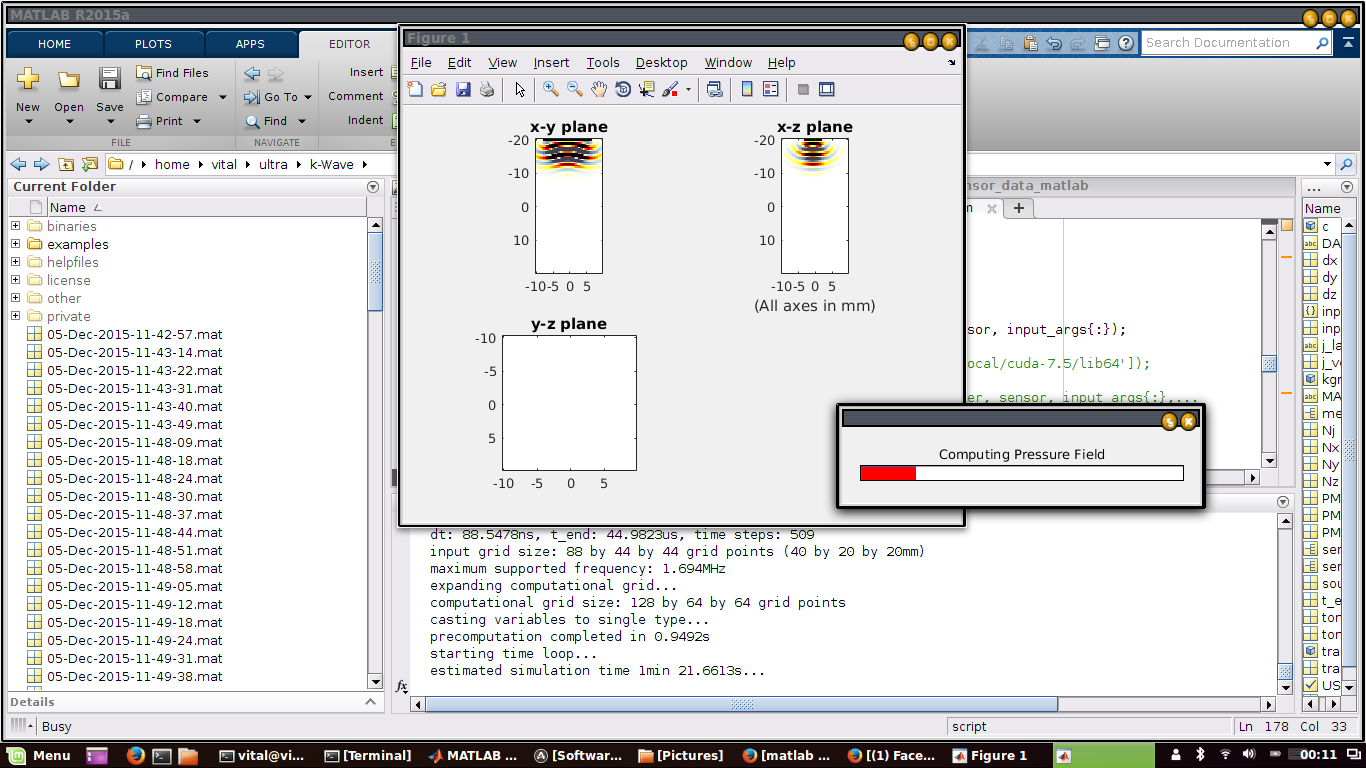
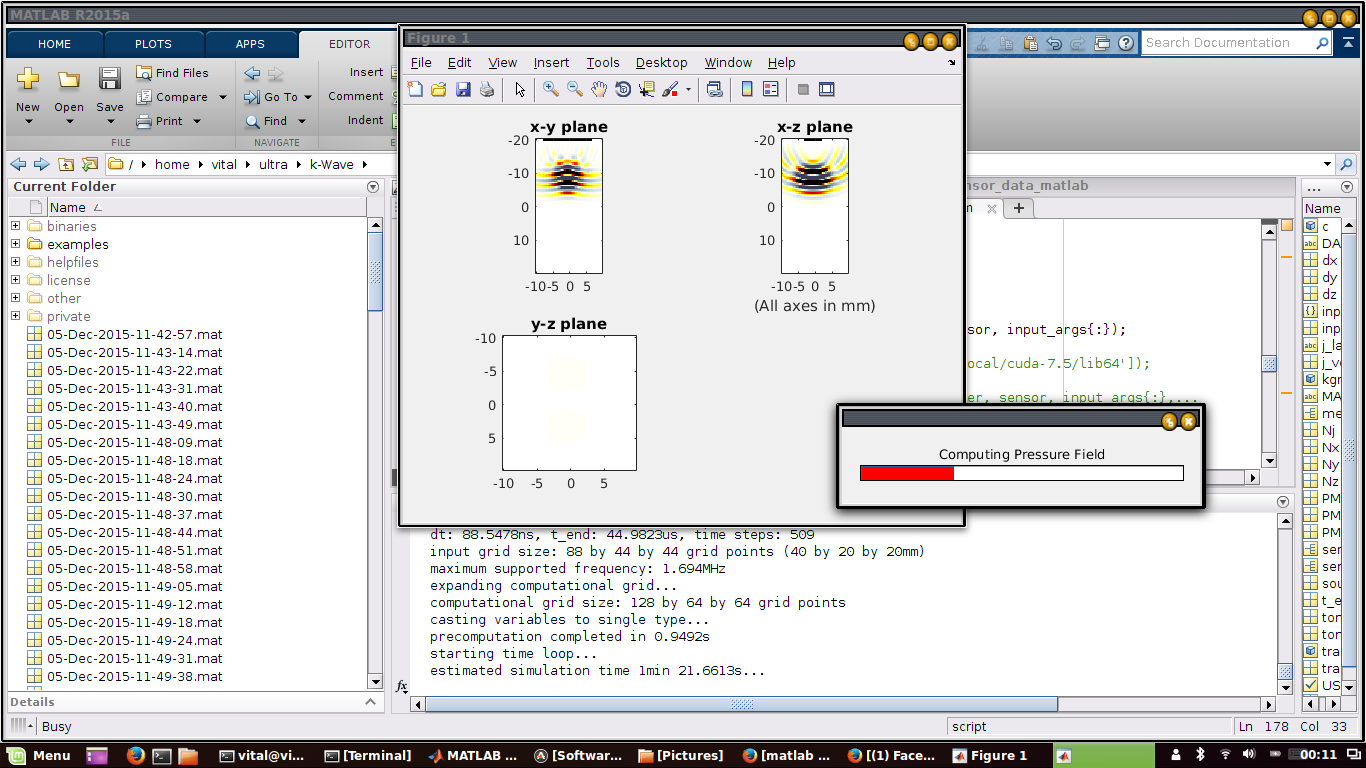


Figura . Simulação usando transdutores como fonte. a),b) e c) propagação da onda em três instantes de tempo diferentes, onde , d) Histograma dos erros relativos em escala logarítmica e) RMS de pressão em uma fatia do domínio calculado com o código matlab f) RMS de pressão em uma fatia do domínio calculado com o código CUDA

O teste para as fontes pontuais do sistema, quer seja a fonte de pressão, quer seja a fonte de velocidade das partículas , exige um esforço maior, pois são quatro diferentes condições. Para tal, foi desenvolvido um algoritmo que testa essas quatro condições automaticamente, gerando um histograma e um gráfico de análise temporal para cada caso. De modo semelhante à simulação das condições iniciais de pressão, foi criado um cenário onde se tem um *array* colinear de sensores cruzando o domínio 3D e duas fontes pontuais dispostas em pontos opostos do domínio que excitam o sistema com sinais senoidais.

Cada tipo de fonte possui quatro variações de simulações, são elas:

* *dirichlet* e iguais entre si;
* aditiva e iguais entre si;
* *dirichlet* e diferente entre si;
* aditiva e diferente entre si.

Os gráficos 4.4 e 4.5 mostram o comportamento do erro.

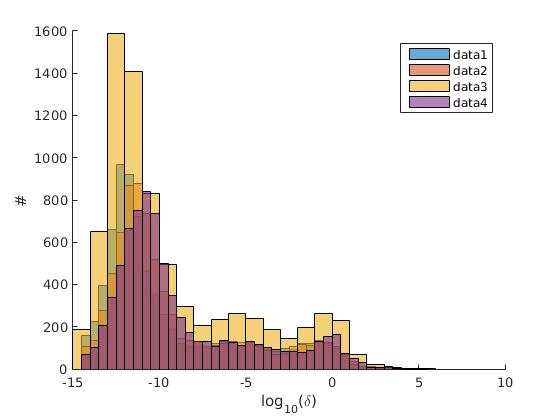
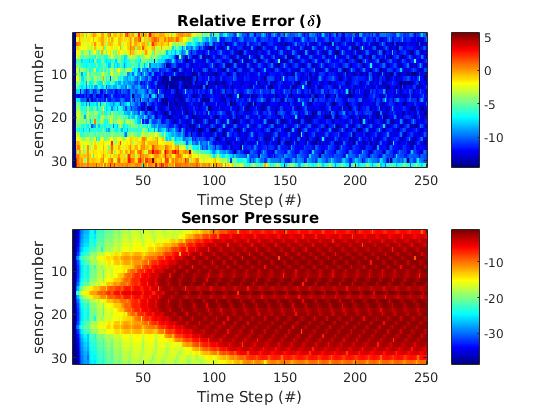
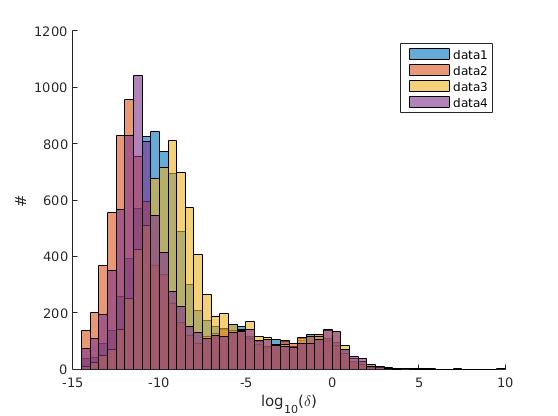
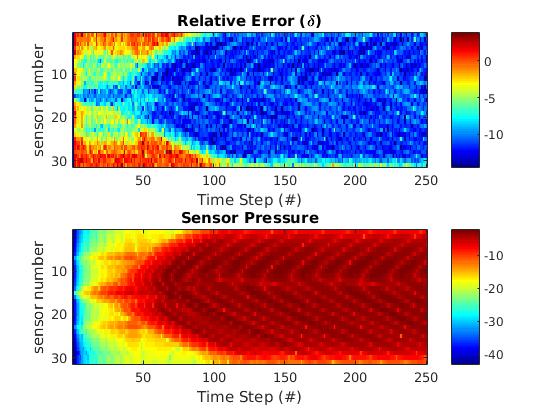


Figura 4. Simulação com fontes de pressão. a) Histograma dos quatro tipos simulações de fonte de pressão, b) Erro relativo das pressões registradas pelos sensores ao longo do tempo (escala logarítmica), c) Pressão registrada prelos sensores ao longo do tempo (escala logarítmica). b) e c) foram simulados com fontes aditivas e diferentes entre si.

Figura 4. Simulação com fontes de velocidade de partícula. a) Histograma dos quatro tipos simulações de fonte de velocidade de partícula, b) Erro relativo das pressões registradas pelos sensores ao longo do tempo (escala logarítmica), c) Pressão registrada prelos sensores ao longo do tempo (escala logarítmica). b) e c) foram simulados com fontes aditivas e diferentes entre si.



As figuras 4.4 e 4.5 são muito semelhantes entre si, o que é esperado, pois mudança na pressão causa mudança na velocidade da partícula e vice-versa. Os histogramas dos quatro casos (tanto com fonte de pressão como com fonte de velocidade) têm uma grande concentração do erro relativo para valores baixos, no entanto existem muitos pontos que possuem um erro elevado de ou mais. Esses erros relativos elevados são justificados, pois ocorrem quando o valor da pressão é extremamente pequeno, , assim, as variáveis, que são pontos flutuantes, perdem a consistência numérica, e o erro tende a aumentar.

Embora a segunda bateria de testes não tenha atingido a meta de erros menores que , em todas as medidas, os resultados confirmam o bom funcionamento do código, pois as discrepâncias são justificadas por imprecisão numérica dos pontos flutuantes (*float*).

A terceira bateria de teste é responsável por verificar o bom funcionamento dos modelos. Os modelos são baseados em 6 parâmetros: linearidade e não linearidade, com absorção ou sem absorção, heterogeneidade em densidade , heterogeneidade em velocidade (), heterogeneidade nos coeficientes de absorção () e heterogeneidade no coeficiente de não linearidade (BonA). Portanto, 36 simulações são necessárias para testar todas as funções do *k-wave*.

O cenário que testa os modelos é composto de um *array* de sensores colineares que cruza todo o domínio, como indica a figura 4.6, e de uma fonte de pressão inicial pontual, localizada no centro do domínio, que excita o sistema com um pulso. A figura 4.7 mostra um exemplo dos sinais recebido pelos sensores em uma simulação, que usa um modelo não linear sem absorção com heterogeneidade em densidade.

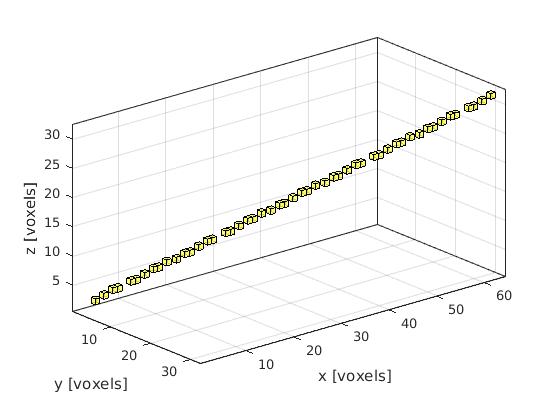


Figura 4.6 Array de sensores

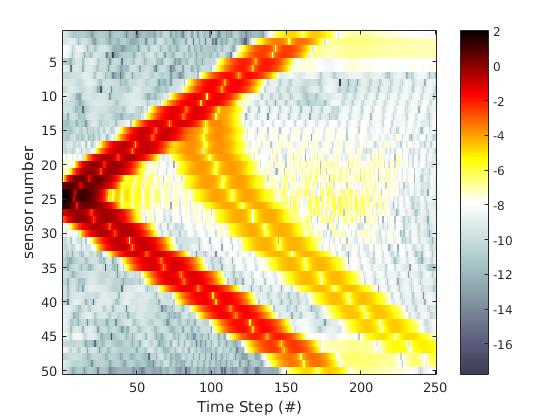


Figura 4.7 Sinais de pressão recebidos pelos sensores (em escala logarítmica) em uma simulação com modelo não linear sem absorção com heterogeneidade na densidade ().

A relação entre as pressões, calculada por meio do código Matlab e do código CUDA, está representada na figura 4.8. Os histogramas da esquerda mostram que, assim como nas simulações das fontes de pressões, o erro relativo é, em geral, baixo, isto é, a maioria dos erros é menor que , no entanto, muitos são erros significativamente altos. Contudo, se se filtrar os histogramas, calculando o erro relativo apenas dos sinais que tenham uma amplitude minimamente significativa (), os histogramas computados apresentam ótimos resultados, com todos os erros relativos menores que e concentrados entre e .

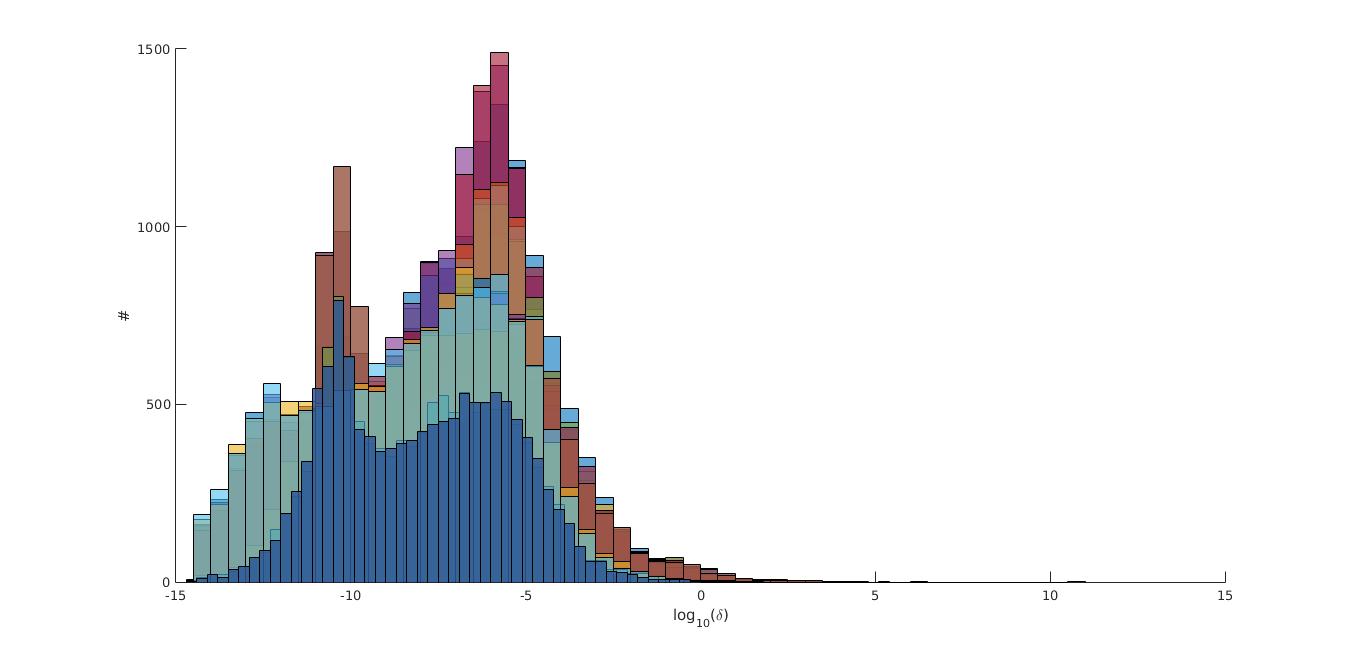
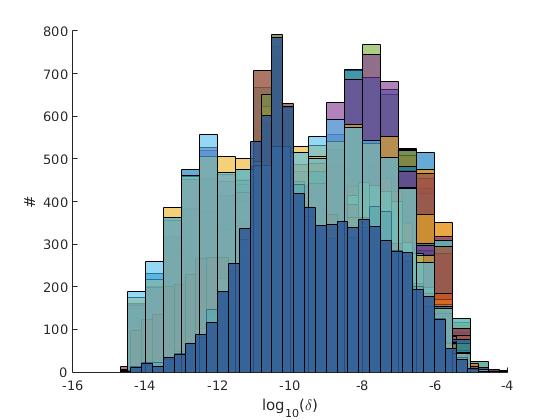


Figura 4. Histogramas do erro relativo de pressão dos 36 modelos sobrepostos. a) Todos os valores de pressões adquiridos pelos sensores estão sendo usados. b) Apenas os valores com amplitude significativa são computados.

Após essas três baterias de testes, têm-se informações suficientes para afirmar que o algoritmo desenvolvido está correto, ou seja, os resultados das simulações são iguais ao algoritmo desenvolvido pelo *k-wave*. Dessa forma, pode-se passar para o teste de desempenho do algoritmo e verificar se houve melhora no desempenho.

Todos os testes, realizados nesta fase, analisaram as pressões coletados nos sensores. Como os outros parâmetros (velocidade da partícula e densidade) dependem da pressão e a pressão depende desses parâmetros, verificando-se a assertividade de um, garante-se a assertividade do outro.

Em todo momento que se trata de erro relativo, está-se falando dessas grandezas em módulo e elevadas ao quadrado. Logo, quando se diz que o erro relativo é de , o erro é, na realidade, . O mesmo raciocínio aplica-se para os valores de pressões, pois eles estão em módulo e elevados ao quadrado.

### Análise de desempenho

A análise de desempenho consiste em verificar qual é a parte do algoritmo mais custosa e como ela pode ser modificada para acelerar o processamento. Para tal, usa-se a ferramenta *Nvidia Visual Profiler* que analisa todos os *kernel’s*, identificando quais são os mais custosos e os seus respectivos pontos fracos.

A figura 4.9 mostra que o tempo de execução do algoritmo CUDA desenvolvido depende fortemente da transformada de Fourier, tanto que o retângulo negro na figura seleciona quais são os *kernels* responsáveis por executar a FFT e a IFFT. Logo, do código está otimizada ao “máximo”, já que a biblioteca CuFFT é o que se tem de mais veloz em termos de transformada de Fourier em dispositivos gráficos.

Portanto, mesmo que o restante do código tenha alguma falha de desempenho, o *speed up* do código está limitado à velocidade da transformada de Fourier.

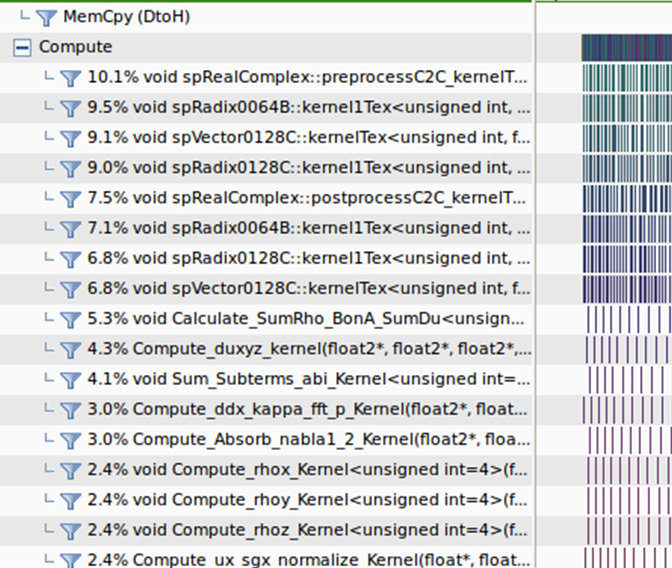


Figura 4. Nvidia Visual Profiler – Kernel time Analysis. Os kerneis selecionados pelo retângulo negro realizam as operações da FFT e IFFT

Dos *kernels* desenvolvidos neste TCC, apenas dois deles apresentam problemas relevantes de desempenho. Todos os outros *kernels* cumprem todos os requisitos básicos para o funcionamento ótimo: não estão limitados por registradores ou por memória compartilhada, não geram conflitos de banco e o acesso à memória global é feito de modo coalescente*.*

Os *kernels* que apresentam problemas são *Compute\_duxy* e *Compute ddx\_kappa\_fft\_p*. Ambos possuem problemas de registradores e de acesso coalescente à memória global. Reduzir o número de registradores é difícil e pode não trazer os benefícios desejados, pois ele está apenas limitando à ocupacidade do dispositivo, o que não quer dizer que o dispositivo não opere na sua capacidade de *hardware* máxima.

Já o acesso coalescente à memória global é um problema mais relevante. O problema ocorre, pois esses dois *kernels* operam no domínio da frequência. Nesse domínio, a coordenada , a coordenada de mudança mais rápida, possui um número ímpar de elementos , o que gera um desalinhamento quando se carrega e se armazena dados na memória global.

Além do desalinhamento causado pela disparidade do domínio, a leitura das variáveis complexas não está sendo feita de modo eficiente, pois, para cada variável complexa, duas leituras estão sendo executadas, uma para a parte real e outra para a parte complexa. Se se usar a memória compartilhada como *buffer*, apenas uma leitura seria necessária para acessar cada variável complexa.

Fisicamente, o desempenho do *hardware* está sendo limitado pela taxa de transferência da memória global (Figura 4.10). Não existe uma solução algorítmica para esse problema, mas o que se pode fazer para solucioná-lo é usar um *hardware* mais potente.

Porém, já que alguns dos *kernels* carregam dados da memória global apenas para a leitura, existe uma pequena alteração que pode reduzir levemente o gargalo no acesso à memória global. Por meio de um recurso chamado de textura, que usufrui das capacidades do hardware de modo diverso, pode-se alocar as variáveis usadas somente para leitura em “memoria global texturizada” ao invés da memória global simples.

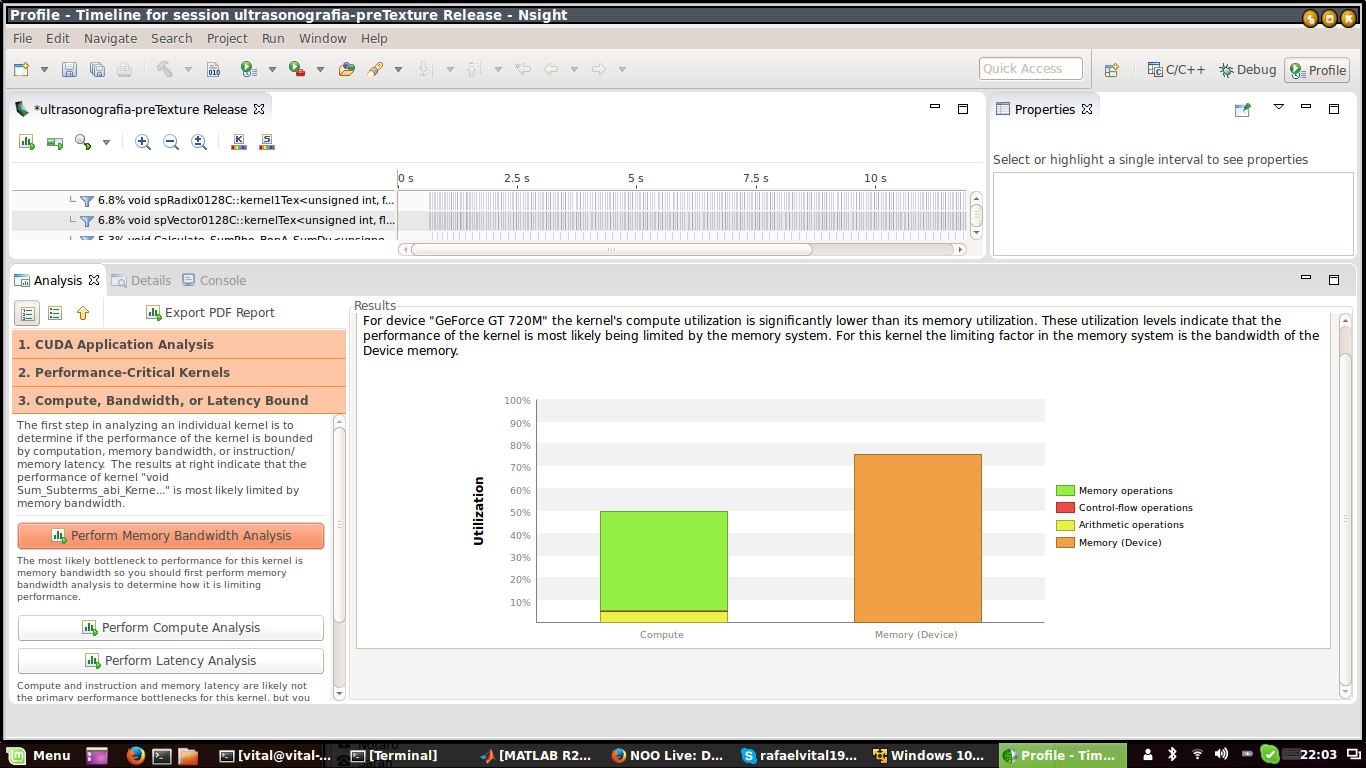


Figura 4.10 Utilização dos recursos computacionais do dispositivo gráfico

### Confronto do tempo de execução

O confronto do tempo de execução consiste em rodar as diferentes versões do modelo de simulação mais custoso com diferentes tamanhos de *grid* e identificar qual plataforma é mais adequada para cada dimensão de *grid*. O *hardware* usado para a simulação é um *notebook* Asus equipado com processador Intel i7-3537U 2.00 Ghz, 3ª geração, com 8Gb de memória RAM e placa de vídeo Nvidia Geforce 720M 2Gb DDR3.

As dimensões de *grid* e seus respectivos tempos de execução são as seguintes:

|  |  |  |  |
| --- | --- | --- | --- |
| *Grid* | Tempo Matlab (s) | Tempo C++ OMP (s) | Tempo CUDA (s) |
| 32 x 32 x 32 | 4,13 | 2,7 | 2,8 |
| 64 x 32 x 32 | 6,01 | 2,4 | 2,6 |
| 64 x 64 x 64 | 26,8 | 5,2 | 6,7 |
| 128 x 64 x 64 | 55,2 | 10,3 | 11,8 |
| 256 x 256 x 64 | 519,6 | 65,7 | 85,03 |
| 512 x 512 x 512 | - | 285,5 | 329, |
| 1024 x 1024 x 32 | - | 137,4 | 181,1 |
| 32 x 1024 x 1024 | - | 155,9 | 220,8 |
| 1024 x 32 x 1024 | - | 157,4 | 180,8 |

A tabela mostra que, embora a versão desenvolvida em CUDA seja mais veloz que a verão Matlab, seu desempenho não consegue superar o tempo de execução do *k-wave*, codificado em C++ e otimizado com OMP.

Contudo, o confronto realizado não é justo, pois a placa de vídeo utilizada para o teste tem um desempenho muito aquém do que se espera, pois as placas GeForce foram criadas especificamente para gráficos e, embora permitam a realização de cálculos científicos, elas não apresentam bons resultados para tal finalidade. Além disso, a desempenho da placa de vídeo 720M é baixa se comparada com outros modelos GeForce o que não ocorre com o processador, que possui ótimas especificações para o seu custo.

Portanto, para uma comparação mais justa, devem-se usar dispositivos gráficos da linha Tesla, específicos para cálculo científico.

# ALGORTIMO TOMOGRáFICO

O algoritmo tomográfico desenvolvido é um algoritmo iterativo muito semelhante ao [#4]. A grande diferença é que esse algoritmo usa o *k-wave* para simular a propagação e a retropropagação das ondas. Então, em um primeiro momento, assim como em [#4], o algoritmo funciona apenas para modelos lineares, sem absorção, semi-heterogêneo em velocidade e homogêneo em densidade .

Lembra-se que uma tomografia por ultrassom pode ser modelada como o problema inverso de uma coleção de equações de propagação da onda, pois pode-se escrevê-las com o operador matemático.

Onde é o operador que descreve a propagação das ondas emitidas pela fonte , dado o objeto , e representa os sinais recebidos pelos receptores, ou seja, a -ésima projeção de . Assim, o problema em questão resume-se em definir o objeto , por meio da coleção de projeções e do operador .

A abordagem iterativa consiste em usar o operador adjunto para aprimorar a estimativa do objeto tomografado, de modo que o corpo evolua da seguinte forma. [#4]

Onde é a estimativa do corpo ao passo , a estimativa inicial, o operador adjunto de que é a derivada de Fréchet de

Em termos físicos, o operador representa a simulação da propagação de onda emitida pela -ésima fonte na -ésima estimativa de . Já o termo é a diferença entre o sinal real medido () diante da emissão do sinal no corpo real e o sinal predito na simulação

Assim um algoritmo iterativo é composto por quatro etapas:

* predição dos sinais recebidos nos sensores, calculados por meio da simulação da propagação das ondas sonoras que partem dos emissores e que atravessam o corpo estimado;
* comparação entre os sinais coletados e preditos, gerando sinais residuais a partir de uma técnica de interpolação capaz de reduzir o número de sensores requisitados;
* retropropagação dos sinais residuais por meio do mesmo corpo em que foi executada a propagação direta;
* aprimoramento das propriedades sonoras do corpo, baseado nas ondas propagadas e retropropagadas.

Nos próximos tópicos, far-se-á uma breve explicação de cada uma das etapas, suas motivações e como cada uma delas foi implementada.

Logo após, será explicada a metodologia de testes e os resultados obtidos pelo algoritmo.

Posteriormente, far-se-á uma explicação da estrutura do código Matlab desenvolvido que implementa o algoritmo em questão.

## Etapas do Algoritmo

As etapas do algoritmo serão apresentadas nas próximas três seções. Devido à similaridade entre a primeira e terceira etapa, ambas serão explicadas na mesma seção.

### Propagação e retropropagação (1ª e 3ª etapa)

No algoritmo, as propagações e as retropropagações das ondas são simuladas utilizando o *K-wave*. Como explicado nas seções anteriores, o *K-wave* aplica o método pseudo espectral *k-space* para resolver as equações de propagação de onda, modeladas com as seguintes equações diferenciais de primeira ordem:

Onde é velocidade acústicas das partículas, a pressão acústica, a densidade acústica, a densidade ambiente, a velocidade isentrópica do som, fator da velocidade e a forçante de massa. O método pseudo espectral consiste basicamente em resolver as derivadas espaciais no domínio de Fourier.

A propagação de onda da -ésima fonte consiste na resolução do sistema de equação diferencial anterior limitado às seguintes condições de contorno, sobre e às condições iniciais em .

Onde é a pressão acústica, é o sinal aplicado e é a localização da fonte . Enquanto o problema de retropropagação tem as seguintes condições:

Onde é o residual que se deseja retropropagar e é o campo de pressão gerado por

### 2ª etapa (interpolação)

A segunda etapa consiste em calcular a diferença entre o sinal propagado () e o sinal medido (). Porém, pode ocorrer (por razões econômicas e de implementação) que o número de sensores seja limitado e que não cubra todo o contorno do domínio (Figura 5.1). Nesse caso, antes de fazer a subtração entre os sinais, é necessário interpolar o sinal medido de modo que ele exista em todo o contorno do domínio (o sinal predito existirá em todo o contorno, já que ele foi simulado durante a propagação). A existência do sinal em todo o contorno é requisito essencial para a etapa de retropropagação, pois os sensores, na etapa de propagação, serão emissores com condição de Dirichlet durante a retropropagação.

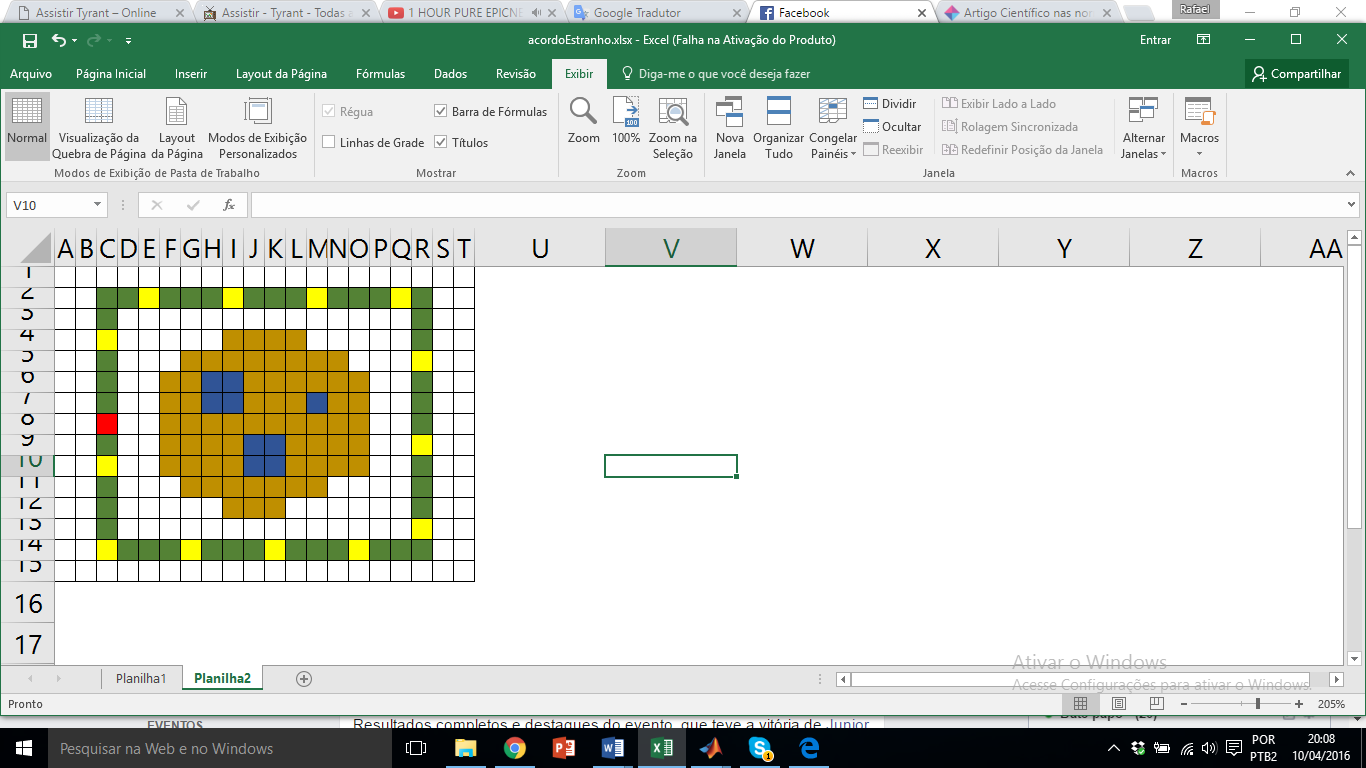
****

Figura 5.1 Exemplificação do domínio de simulação. Grid marrom e azul, seção do corpo que possui diferentes velocidade de propagação. Grid vermelho, j-ésima fonte emissora. Grid amarelo, receptores. Grid verde, coordenadas que fecham o contorno do corpo e que necessitam de interpolação dos dados

A interpolação usada é linear e leva em consideração os dois sensores ‘reais’ mais próximos ao sensor ‘fictício’ a ser interpolado. Além disso, a interpolação deve considerar o atraso do sinal recebido. O atraso na leitura ocorre, pois a distância e a velocidade entre cada par emissor-sensor são diferentes. Esse atraso é calculado por meio da integral de linha:

Onde S é o percurso que liga diretamente o emissor ao sensor e é a velocidade de propagação no ponto pertencente ao percurso S. Assim, o sinal interpolado é calculado com a fórmula:

Onde é o sinal recebido pelo sensor durante a simulação da fonte e os pesos e são inversamente proporcionais à distância entre os respectivos sensores, sendo que

### 4ª etapa (aprimoramento)

A quarta etapa é onde se calcula o aprimoramento do coeficiente do corpo. Em [#4], deduz-se que uma aproximação para a operação é:

Onde é o fator de velocidade, é o laplaciano da pressão propagada na primeira etapa e é a pressão retropropagada na terceira etapa. Assim, têm-se todas as ferramentas para calcular iterativamente a partir de .

Contudo, resultados experimentais mostraram que essa aproximação de , quando usado juntamente com o modelo pseudo espectral, gera efeitos que invalidam a aproximação. Porém, seguindo a regra de pequenos incrementos no aprimoramento de , uma modificação foi realizada para estabilizar o processo.

Essa modificação consiste em controlar a amplitude de , que se chamará e de sua média espacial . Assim, ao invés de aprimorar o corpo com valores entre 5 e 15, por exemplo, o aprimoramento ocorre entre -1 e 2. O controle acontece por meio dos coeficientes e que modificam .

Onde é constante durante todo o processo, e é um fator próximo de 1 que varia a cada iteração segundo a expressão Onde lag é o valor de que maximiza a correlação entre os sinais medidos e predito .

## Metodologia de teste

A metodologia de teste para o algoritmo é baseada em um phantom que representa o torso de um ser humano. O phantom deve respeitar os mesmos requisitos do algoritmo, isto é, linear, sem absorção, semi-heterogêneo em velocidade e homogêneo em densidade .

O phantom é nada mais que um conjunto de matrizes e escalares que quantizam as propriedades sonoras e físicas do objeto. Como se tem heterogeneidade apenas na velocidade de propagação do som (), uma única matriz consegue representar o objeto, assim, quando se referir ao phantom, está-se se referindo a matriz dos coeficientes . Mas, não se pode esquecer que a densidade também é importante para caracterizar o objeto, porém ela é constante e é representada como um escalar, juntamente com a dimensão do objeto e do passo de discretização espacial.

Os testes verificarão a funcionalidade do algoritmo e os seus limites. A verificação dá-se por meio de uma análise qualitativa e quantitativa. Qualitativamente, há a comparação visual do phantom com a imagem tomográfica gerada pelo algoritmo. Entende-se como imagem tomográfica, a imagem gerada a partir da matriz dos coeficientes de velocidade do som em cada ponto do objeto. A comparação entre as imagens, além de visual, pode ser quantizada usando métricas, como, por exemplo, o MSE (*mean square error*, erro quadrático médio) e o NMSE (*normalized mean square error*, erro quadrático normalizado).

Além da comparação entre os coeficientes , pode-se analisar a correlação e o valor residual entre os sinais de pressão, obtidos nos receptores, em cada iteração do algoritmo, com os sinais de pressões, obtidos na simulação do phantom.

Cada teste é composto de três etapas: a primeira etapa é composta pela criação do phantom e da definição de sensores e de receptores e pela aquisição dos ‘dados reais’; a segunda é onde se executa o algoritmo de resolução do problema inverso e a terceira é onde se geram os resultados para análise (alguns dos resultados para análise são gerados durante a segunda etapa).

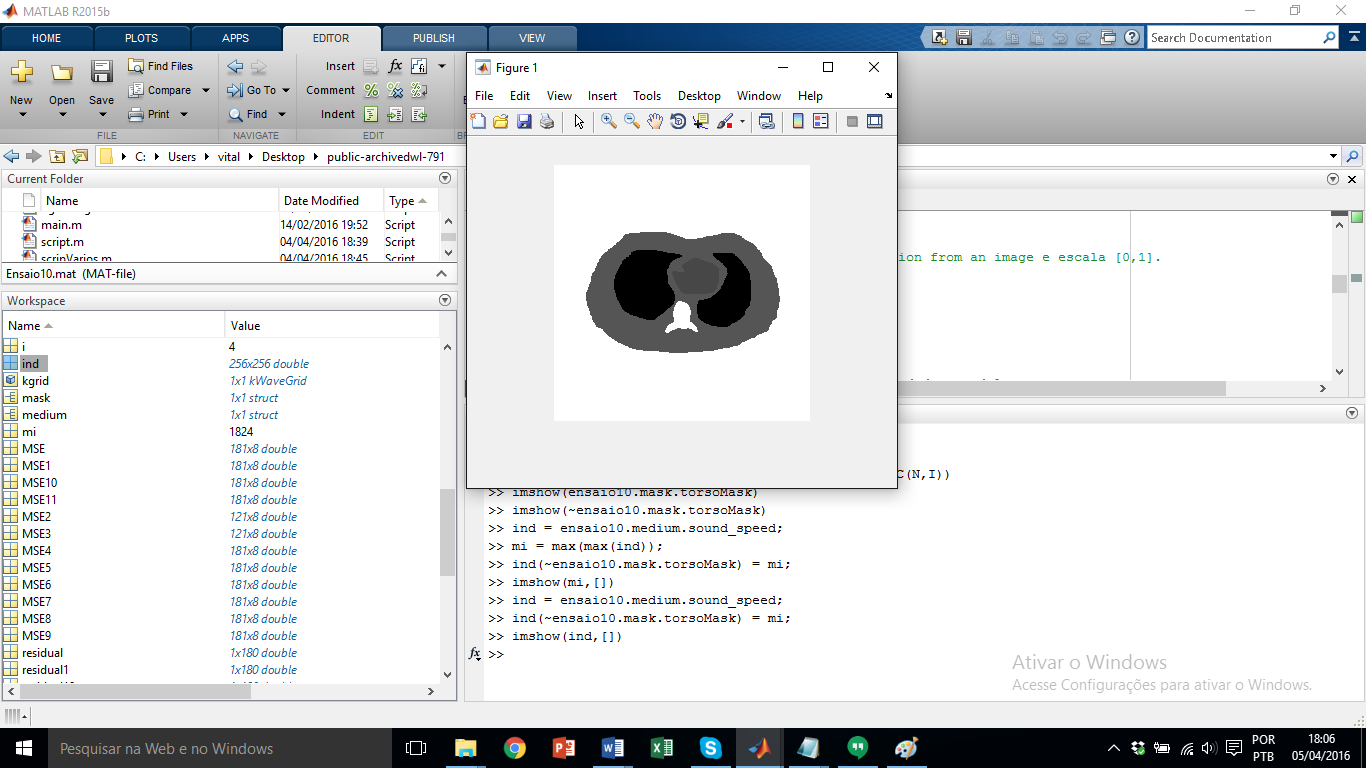
### Primeira etapa

Na primeira etapa, o objetivo é criar o problema para ser resolvido na etapa sucessiva. Assim, cria-se um *phantom* e realizam-se diversas simulações de modo a obter os dados que se chamarão de ‘reais’ (embora eles sejam simulados). Como esses dados devem ser o mais próximo possível da realidade, a ideia é que as simulações tenham passos de discretização pequenos, minimizando os erros numéricos e computacionais.

O ***phantom*** usado para validar o algoritmo, é composto por 4 seções, cada seção representa diferentes órgãos do corpo humano (coração, pulmão, coluna e meio/torso). Todos os *phantons* usados têm a mesma forma, isto é, cada seção tem os locais predeterminados, o que se altera em cada *phantom* é o coeficiente de propagação do som de cada uma das seções (toda a seção tem o mesmo coeficiente).

Os coeficientes dos órgãos seguem um parâmetro que indica o contraste da velocidade de propagação. Os coeficientes seguem as seguintes regras:

Outros parâmetros que definem o *phantom* são a dimensão do próprio *phantom*, a discretização espacial e a densidade . Em todos os testes, a densidade e a dimensão do *phantom* são fixos, enquanto que o passo e a discretização variam de acordo com a necessidade. A figura 5.2 mostra um exemplo de *phantom* gerado.



meio

torso

coluna

pulmão

coração

Figura 5.2 Exemplo do phantom. Embora na figura, o coeficiente do torso e do meio pareçam diferentes no algoritmo eles são os mesmos.

Assim, a especificação do *phantom* é dada pelos seguintes parâmetros:

: define o número de passos de discretização na direção X e Y;

: define o contraste entre os coeficientes de velocidade de propagação;

Tamanho do *phantom* em metros

A definição dos **sensores** é simples e consiste em definir quantos sensores serão usados e onde eles estarão. Nos experimentos, foram escolhidas duas maneiras de colocar os sensores: uma cobrindo o torso e outra formando um anel quadrado no limite do PML.

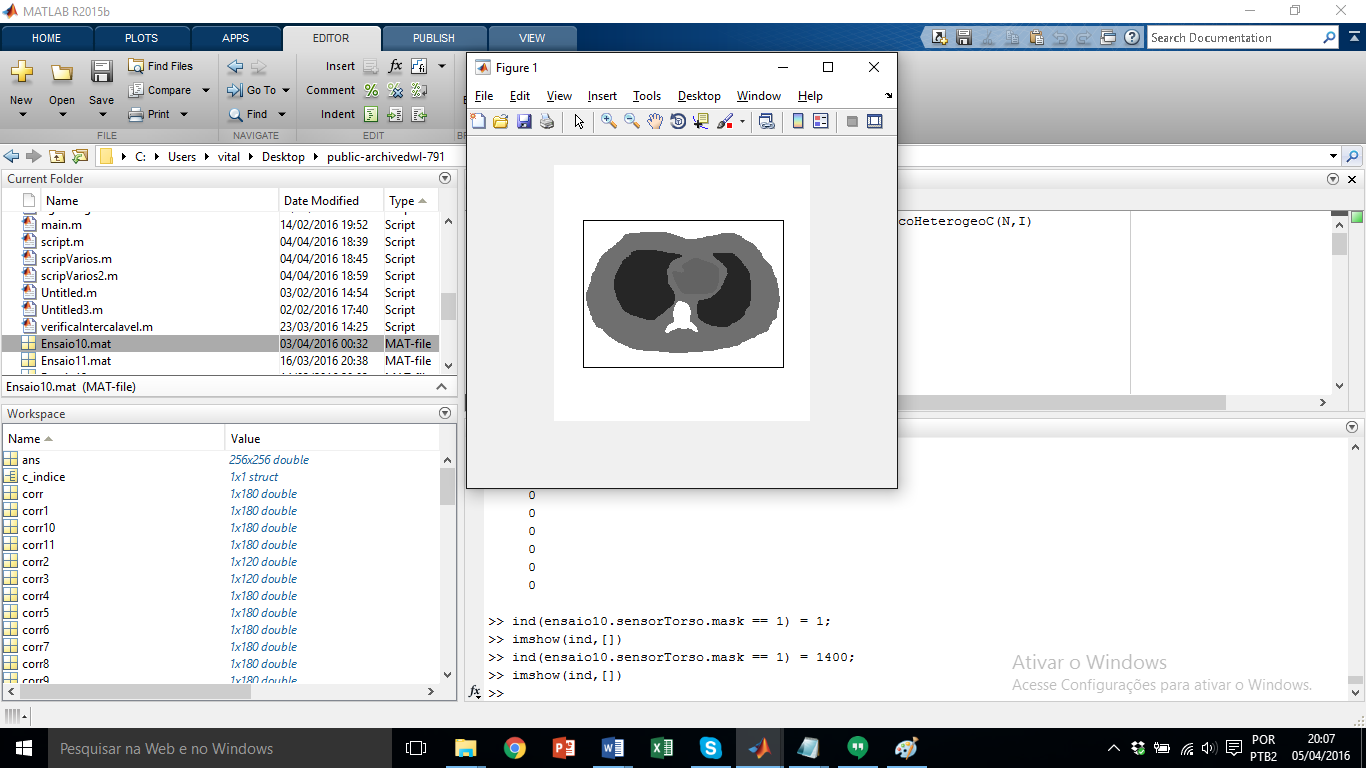
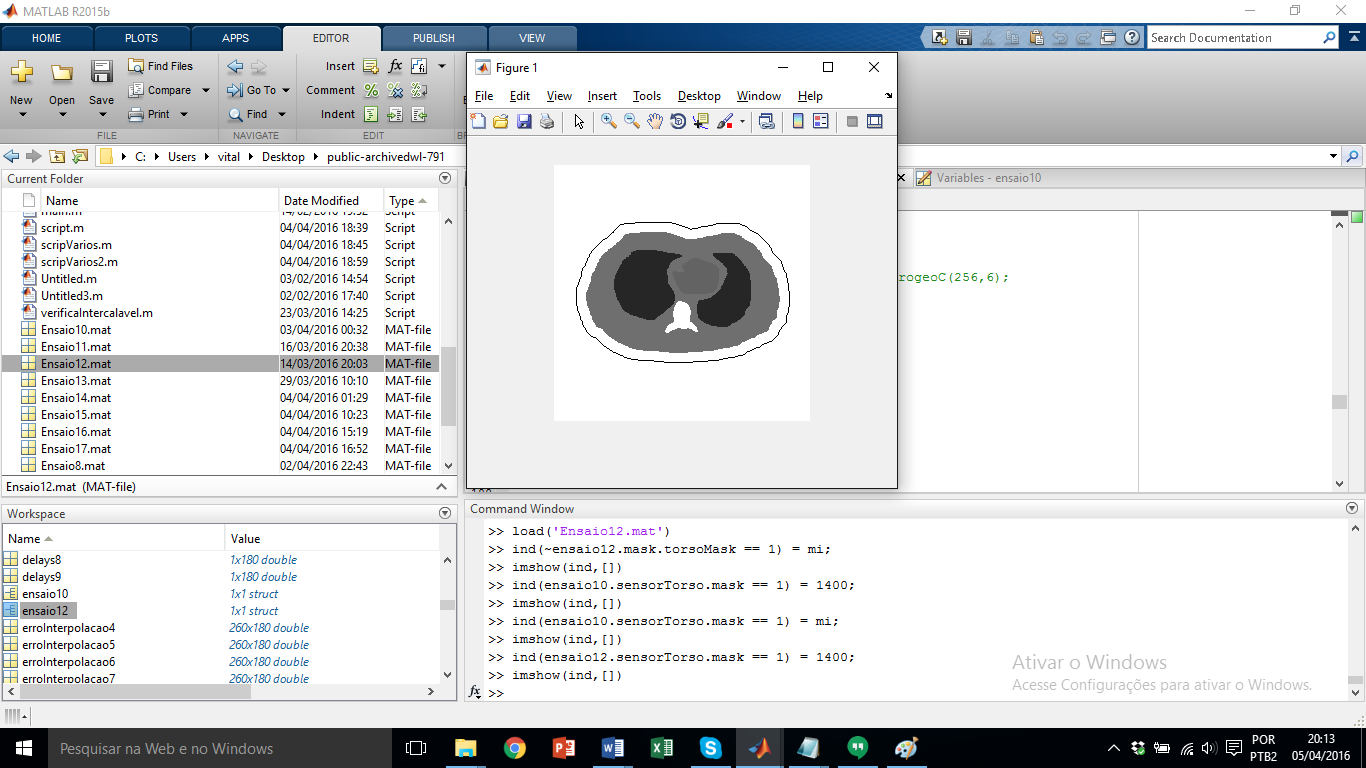


Figura . - Localização dos sensores. Sensores no limite do PML (esquerda). Cobrindo o torso (direita).

A definição das **fontes** é semelhante à definição de sensores, porém as fontes são pontuais, diferentemente dos sensores que são contínuas (na medida em que o *grid* discreto permite). As fontes devem estar na mesma coordenada que os sensores, isto é, se os sensores formam um anel quadrado, as fontes devem estar posicionadas nesse anel.

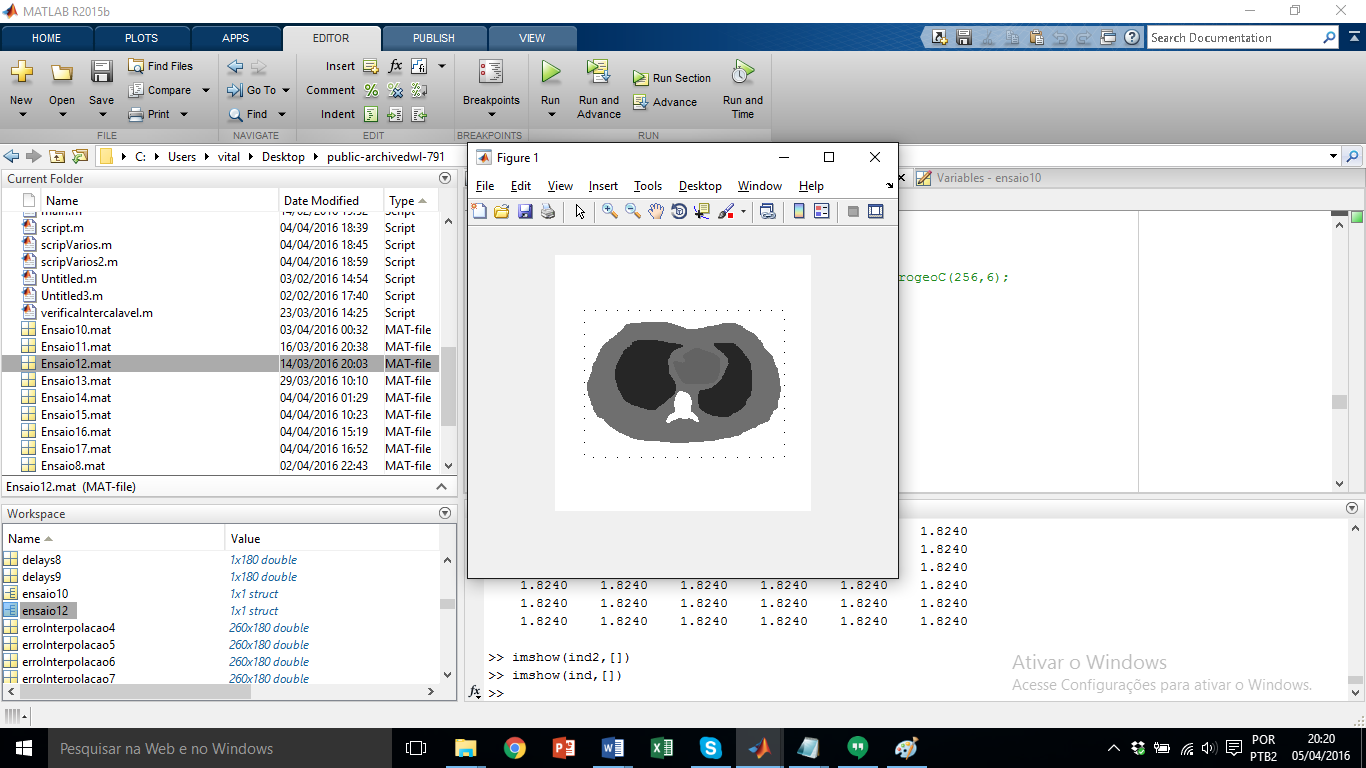


Figura 5.4 Localização das fontes

Além de definir a localização da fonte, ainda tem-se que definir qual é a sua função de incidência. Essa é uma etapa muito importante, pois o espectro da função deve ser limitado, porque, quanto maior for o espectro da função, menor deve ser a discretização espacial. A função escolhida, para incitar o sistema, é um impulso senoidal filtrado de modo que 90% da potência do sinal tenha frequência menor que 187khz, assim, o mínimo *grid* admissível para simulação deve possuir, ao menos, 96 divisões por dimensão.

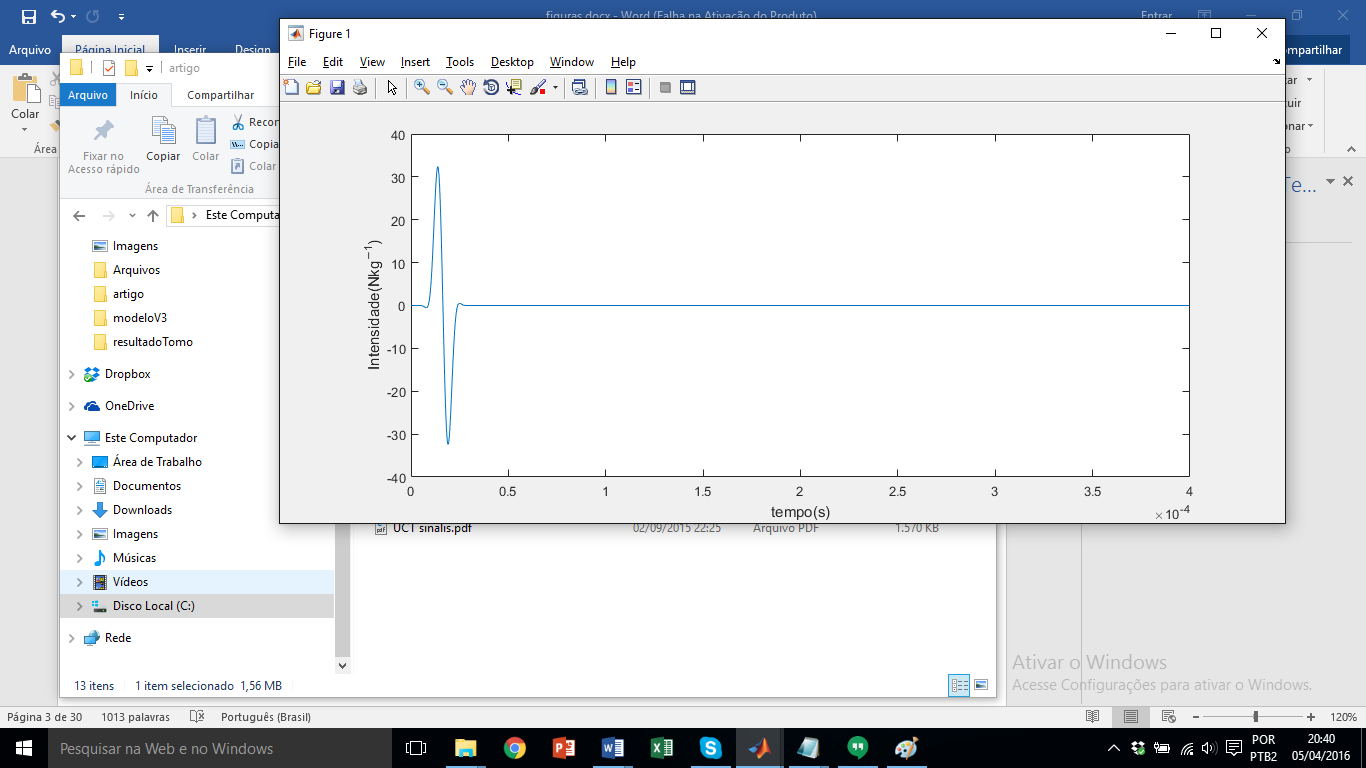


Figura 5.5 Função de incidência

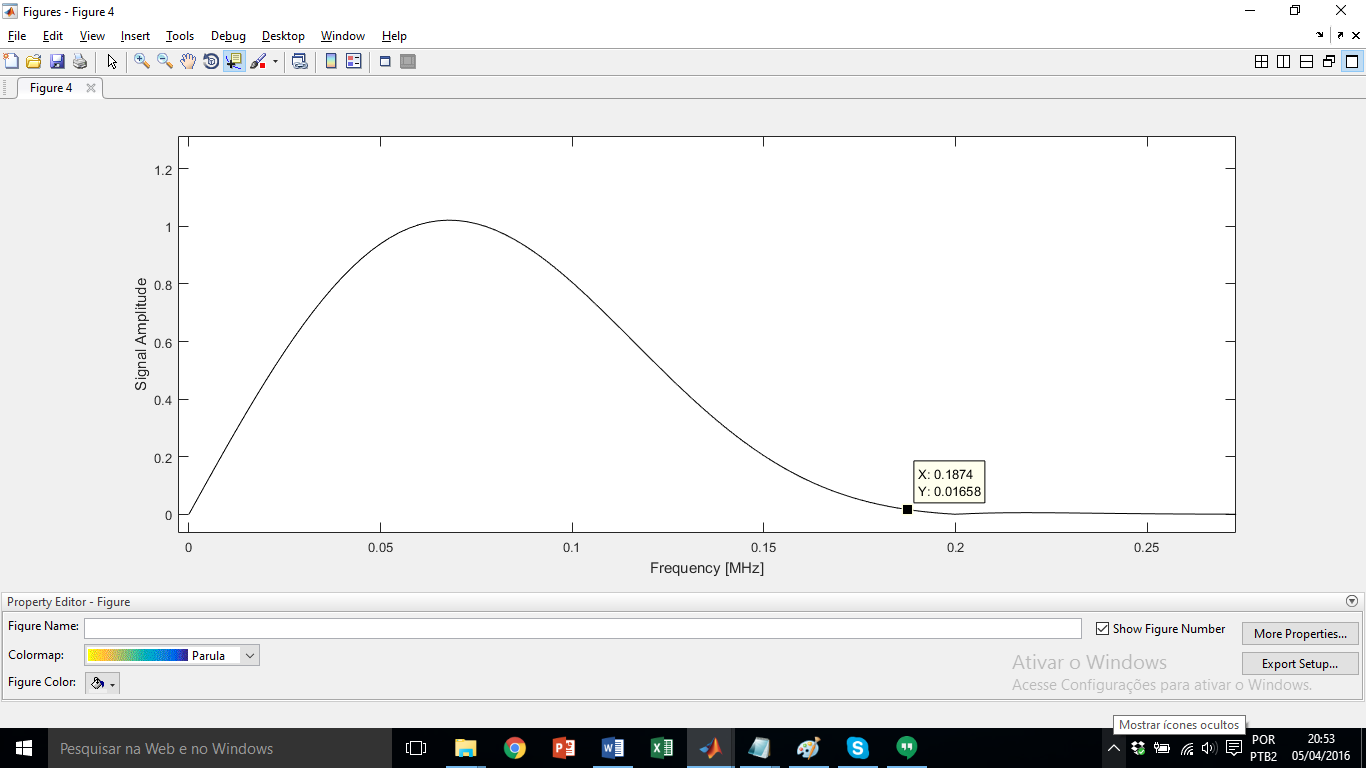


Figura 5.6 Espectro da função de incidência

Por fim, os últimos parâmetros que se devem definir são **o tempo de simulação** e o **intervalo de tempo** .

O tempo de simulação é fixo, uma vez que o tamanho do *phantom* é fixo e, na média, a velocidade de propagação do som no *phantom* é constante. No caso, usa-se **,** que é tempo suficiente para uma onda sonora partir da fonte, refletir no extremo oposto do domínio e retornar para a posição da fonte emissora.

Já o intervalo de tempo depende da velocidade de propagação do *phantom*, máxima e mínima, e do passo de discretização espacial. Para calcular o intervalo de tempo adequado, foi usada a função do *k-wave ‘maketime’* com número **CFL = 0.1**, o que proporciona uma simulação mais refinada.

Definidos todos os parâmetros, podem-se executar as simulações da propagação das ondas, concluindo a primeira etapa, na qual chamamos por **ensaio.** Vale lembrar que devemos executar uma simulação para cada fonte emissora.

### Segunda etapa

A segunda fase consiste em resolver o problema inverso, partindo dos dados adquiridos nos ensaios da primeira etapa. A resolução do problema depende de alguns parâmetros e são justamente eles que se quer pôr a prova durante a validação do algoritmo.

Os primeiros parâmetros são as **discretizações temporais e espaciais** que serão usadas para solucionar o problema inverso. Uma discretização muito fina aumenta o tempo de simulação a tempos impraticáveis, enquanto uma discritização grossa prejudica os resultados numéricos obtidos. Como o número de divisões espaciais limita a frequência, a discretização mínima é de 96 divisões, embora se possa simular com 64 divisões (ignorando o fato que a frequência extrapola ligeiramente os limites estabelecidos pelo *k-wave*).

Os próximos parâmetros de interesse são o número de **iterações** e de **varreduras**. Uma iteração considera o aprimoramento do meio, causada por uma das fontes emissoras. Uma varredura consiste no aprimoramento do meio, causada por todas as fontes emissoras. Assim, uma varredura é composta de um número de iterações igual ao número de fontes emissoras.

Ainda tem-se que definir o **número de sensores** e o **número de fontes** que se usará. Embora o ensaiodefina uma quantidade de sensores suficientes para preencher continuamente o contorno do torso, no mundo real, não se dispõe de tantos sensores. Logo, em uma simulação realística, deve-se considerar que existem somente alguns sensores e que se deve interpolar os dados para obter um contorno completo do torso, conforme explicado no algoritmo.

Por fim, tem-se que definir o **meio inicial** do qual o algoritmo partirá. No primeiro momento, considerar-se-á a densidade conhecida e apenas a velocidade de propagação deve ser aprimorada. O meio inicial pode ser de três tipos: um meio neutro, isto é, todos os valores são impostos por um valor neutro, por exemplo, (velocidade média nos músculos); um meio na qual as formas dos órgãos são iguais a do *phantom*, mas o coeficiente é diferente; como o oposto, o coeficiente é conhecido e igual, mas a forma distinta.

Não se podem desconsiderar os dois parâmetros internos do algoritmo, os fatores  e **.**

### Terceira Etapa

A terceira etapa é onde se analisam os resultados. O primeiro passo é comparar, visualmente, os resultados, obtidos pelo algoritmo, com o *phantom* original, logo após, calculam-se o MSE e o NMSE para cada região (órgão) do corpo topografado, obtendo, assim, um valor analítico e menos subjetivo.

****

Figura 5.7 Imagem tomografica obtida (esquerda). Phanton Original (Direita)

O MSE e o NMSE são calculados da seguinte forma:

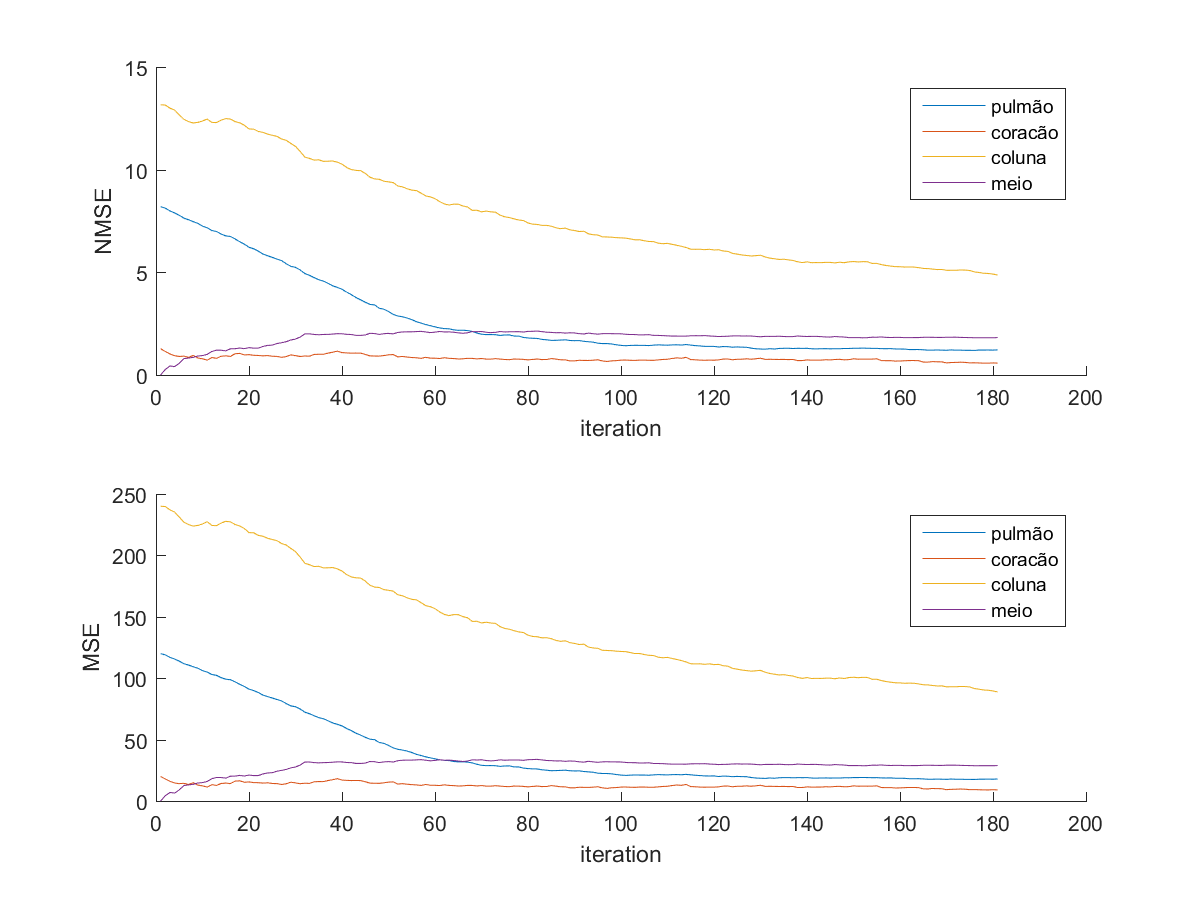


Figura 5.8 Exemplo da evolução do NMSE e do MSE com o número de iterações

Outra forma de mensurar a qualidade da imagem tomográfica é usando a correlação e o valor residual entre os ‘sinais reais’ (*phantom*) e os sinais simulados (imagem tomográfica). O valor **residual** entre os sinais é calculado da seguinte forma:

Onde, a cada o valor da pressão mensurada pelos sensores do *phantom* (p e pelos sensores da imagem tomográfica (são diferentes, pois, a cada iteração, a fonte emissora é diferente, logo o sinal recebido, nos sensores, também serão diferentes.

O sinal , além da alteração, devido à fonte emissora, muda conforme o algoritmo aprimora a identificação do meio (corpo). Por isso, o valor residual é um parâmetro muito variante e deve-se analisar a tendência dele, isto é, se o valor residual sofre uma redução na média. A figura a seguir exemplifica o residual de acordo com o número de iterações.

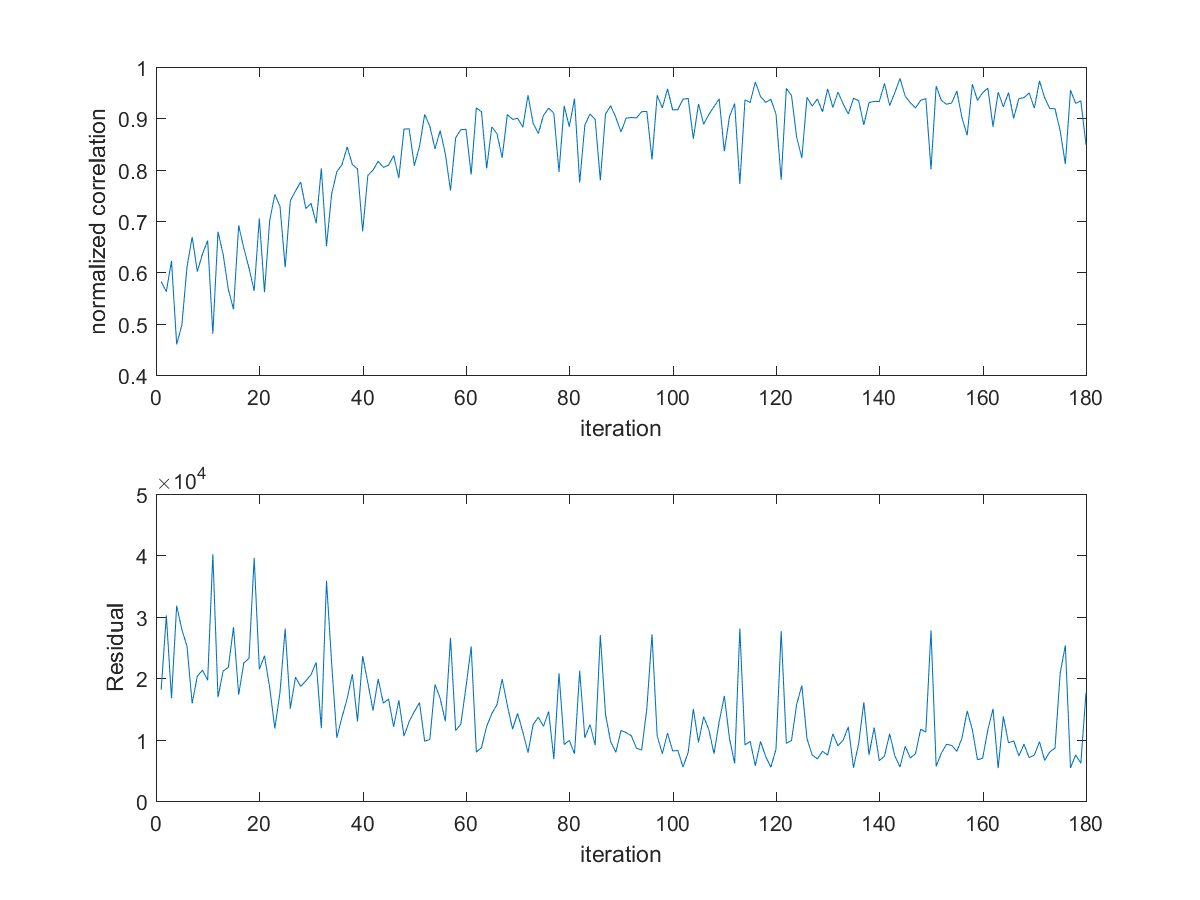


Figura 5.9 Exemplo da evolução do valor residual com o número de iterações

Por sua vez, a **correlação normalizada** é calculada da seguinte forma:

Onde varia entre e , e a somatória no tempo deve respeitar os limites dos vetores temporais na simulação. A correlação sofre os mesmos efeitos que o valor residual e, em sua análise, deve ser levada a tendência dela. A seguir, apresenta-se um exemplo da evolução da correlação em função do número de iterações.

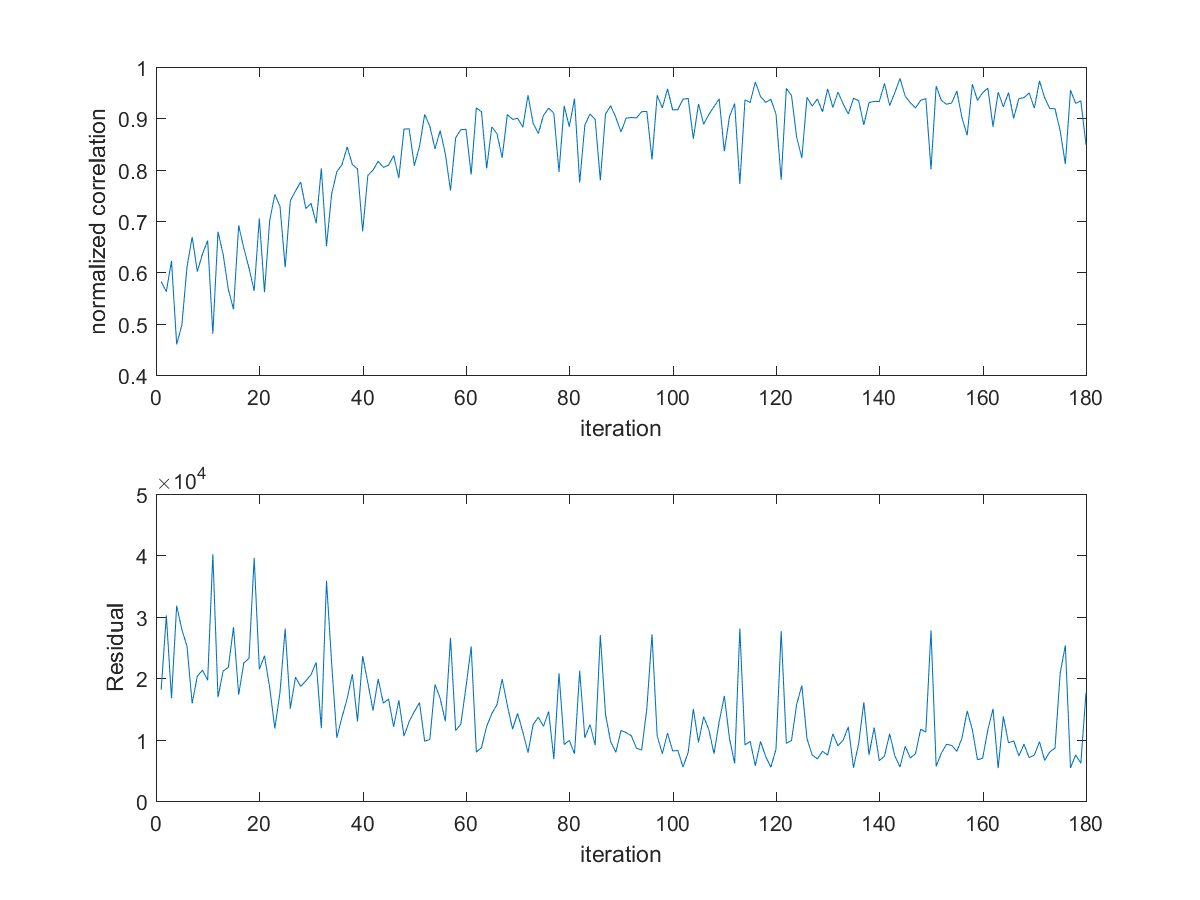


Figura 5.10 Exemplo da evolução da correlação normalizada com o número de iterações

Com relação à correlação, vale destacar que o valor de , que maximiza a correlação, em geral, é zero (principalmente quando a imagem tomográfica se aproxima do *phantom* ‘real’), como pode ser visto no gráfico a seguir.

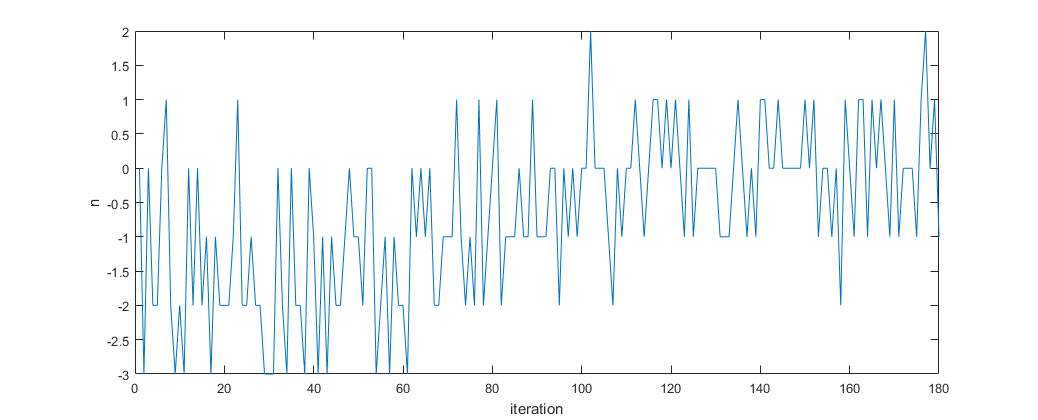


Figura 5.11 Evolução do índice de deslocamento que maximiza a correlação com o número de iterações

Comparando as métricas de valor residual e de correlação, constata-se que ambas seguem um comportamento parecido, mas que a correlação é mais marcante, pois apresenta uma maior variação quanto mais a imagem tomografia se aproxima do *phantom*. Além disso, a ‘instabilidade’ da correlação é proporcionalmente menor que a ‘instabilidade’ do valor residual. Por ‘instabilidade’ entendem-se os bicos do gráfico.

## Testes e Resultados

Os testes têm por objetivo verificar a funcionalidade e os limites do algoritmo.

A primeira bateria de teste visa a verificar até qual valor de contraste o algoritmo, partindo de um meio inicial neutro, consegue recuperar o *phantom*. Para tanto, realizam-se simulações com diferentes valores de e deixam-se os demais valores constantes.

A segunda bateria de teste visa a verificar a convergência do algoritmo, partindo de um meio com a mesma forma que o *phantom*, mas com coeficiente da velocidade de propagação do som diferente. Para tanto, realizam-se simulações com *phantoms* com diferentes , partindo de *phantoms* com outros valores de , mas os demais parâmetros permanecem constantes.

A terceira bateria de teste visa a verificar a convergência do algoritmo, partindo de um meio com os mesmos coeficientes da velocidade de propagação do som que o do *phantom* original, mas com a forma ligeiramente diferente. Para tanto, realizam-se simulações que partem de um *phantom* deformado, isto é, muda-se o domínio de cada uma das regiões, mas mantêm-se os coeficientes delas.

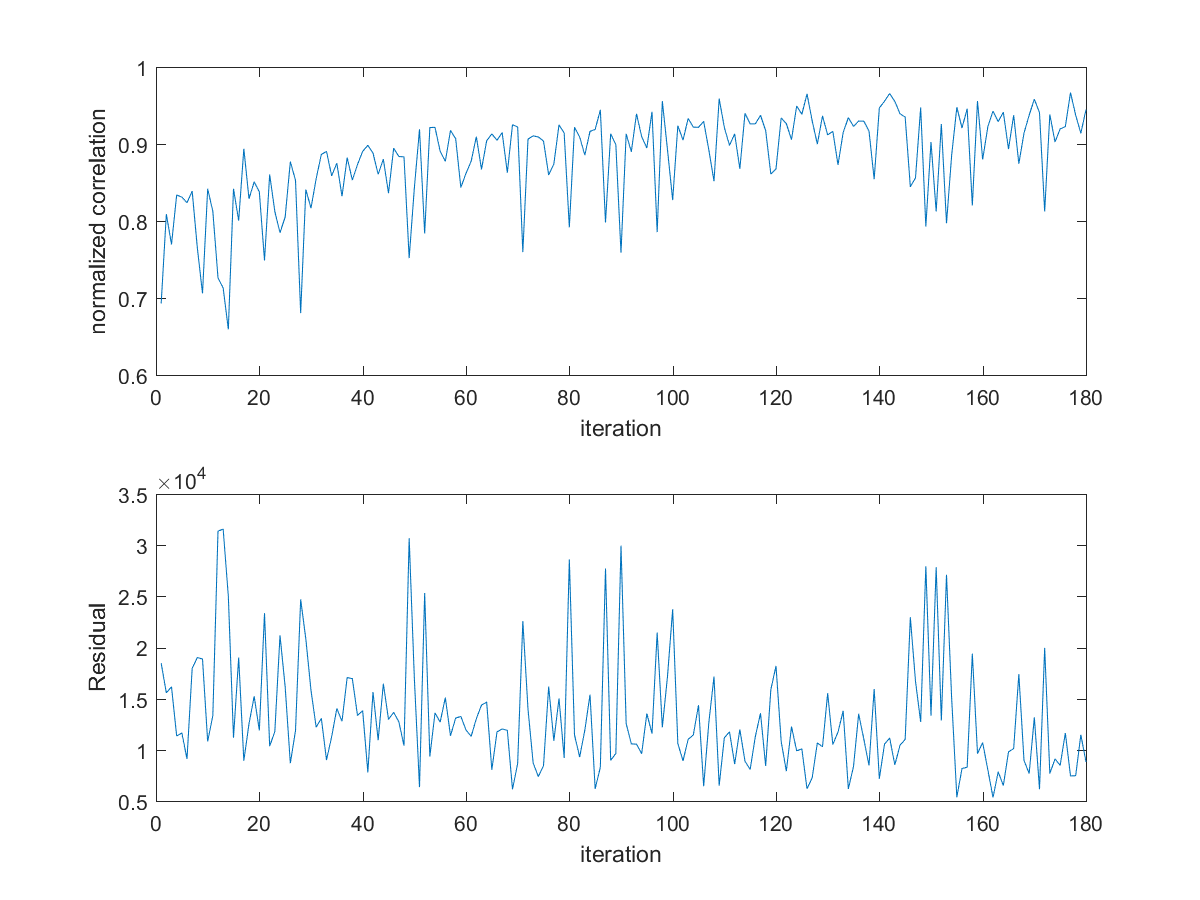
A quarta bateria de teste visa a verificar a influência do número de sensores no algoritmo e se o método de interpolação desenvolvido é adequado. Para tanto, realizam-se simulações com diferentes números de sensores e mantêm-se os demais parâmetros constantes.

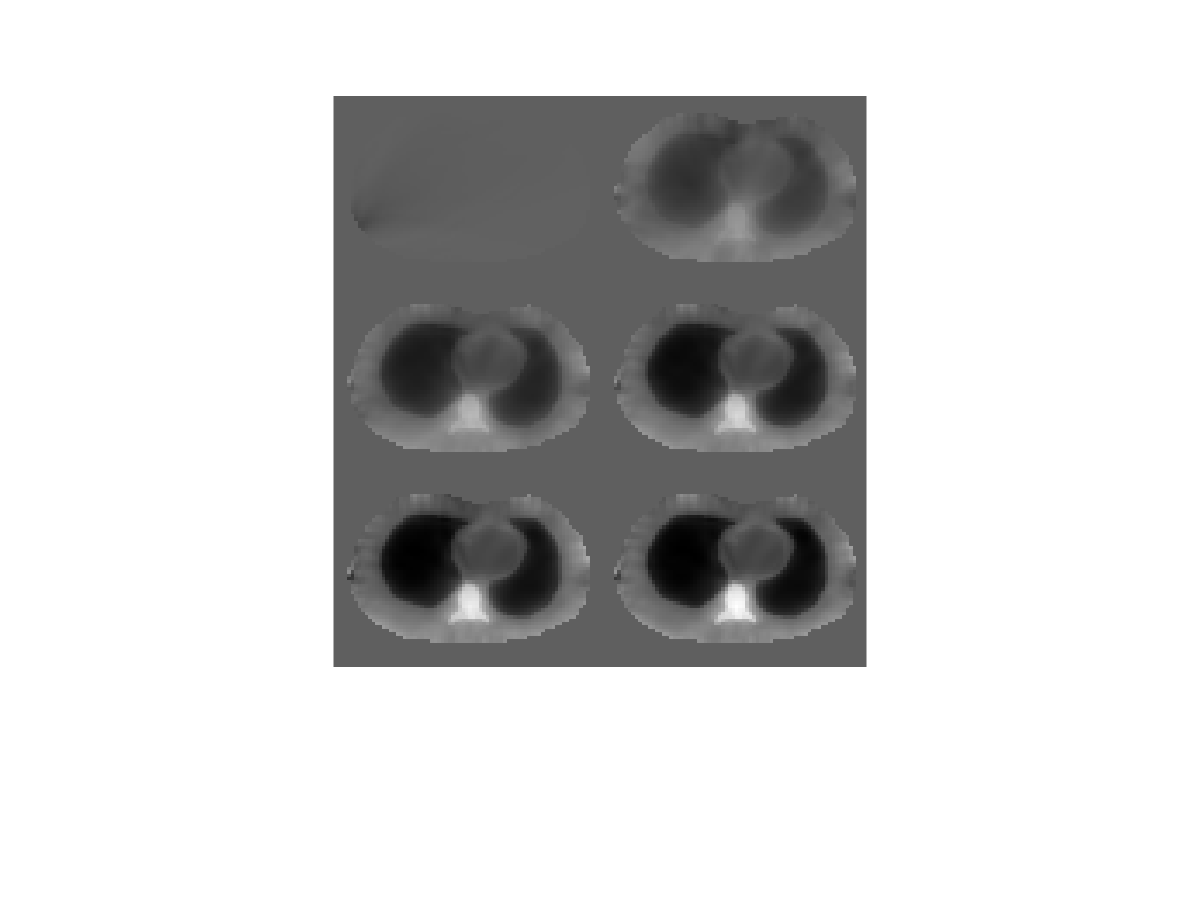
A quinta bateria de teste visa a verificar a influência do número de fontes emissoras no algoritmo. Para tanto, realizam-se simulações com diferentes números de emissores e mantêm-se os demais parâmetros constantes.

Os **resultados desses testes** foram os seguintes:

**1ª Bateria –** Teste de Contraste , partindo do meio neutro**:**

|  |  |
| --- | --- |
| **Parâmetros constantes** | |
| N *phantom* | 256 |
| CFL *phantom* | 0.1 |
| N algoritmo | 96 |
| CFL algoritmo | 0.3 |
| Tipo de Borda | Anel quadrado |
| nSenores | Continuo (694) |
| nFontes | 60 |
| nIterações | 180 |
| Meio inicial | Neutro |
| Fator media | 0.025 |
|  | 3 |
| **Parâmetros alterados** | |
| *phantom* | 3, 6 e 9 |

 Resultado



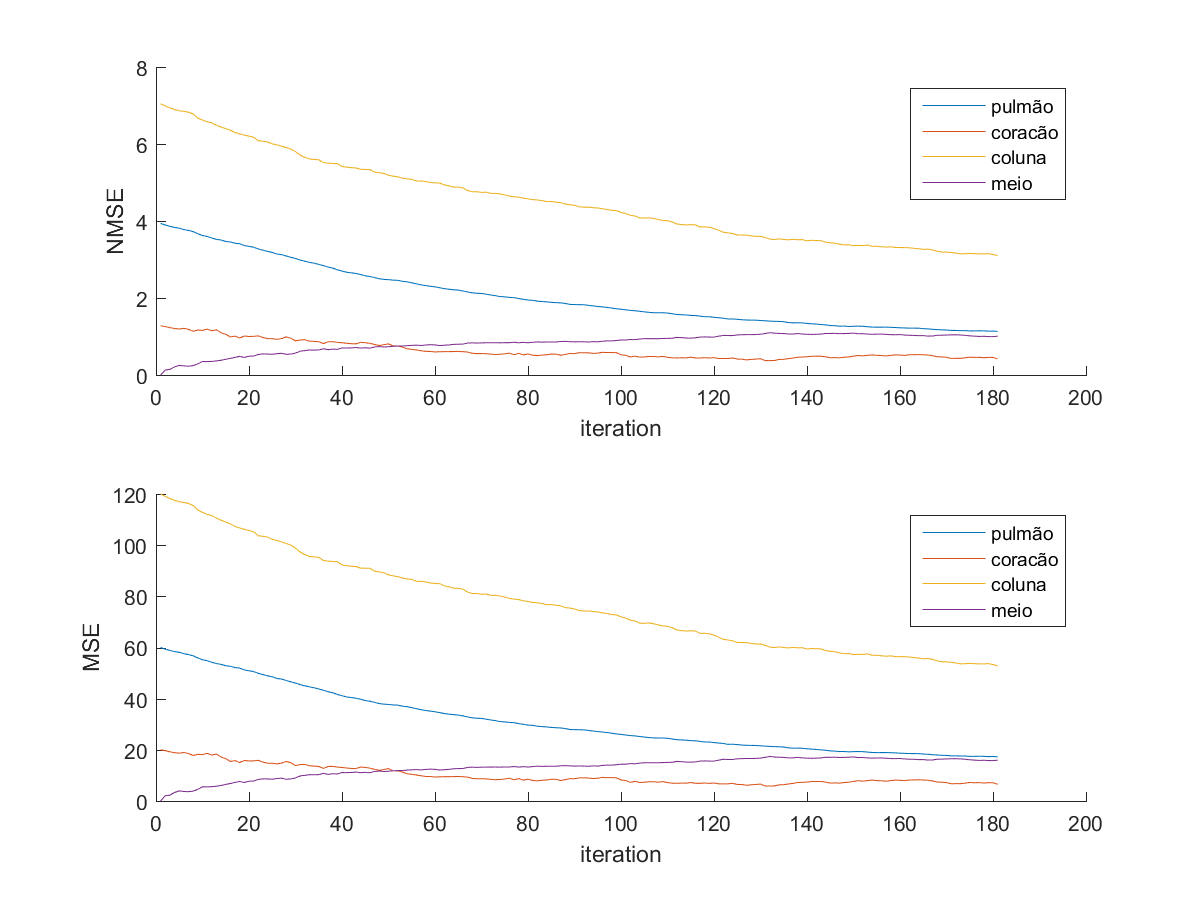
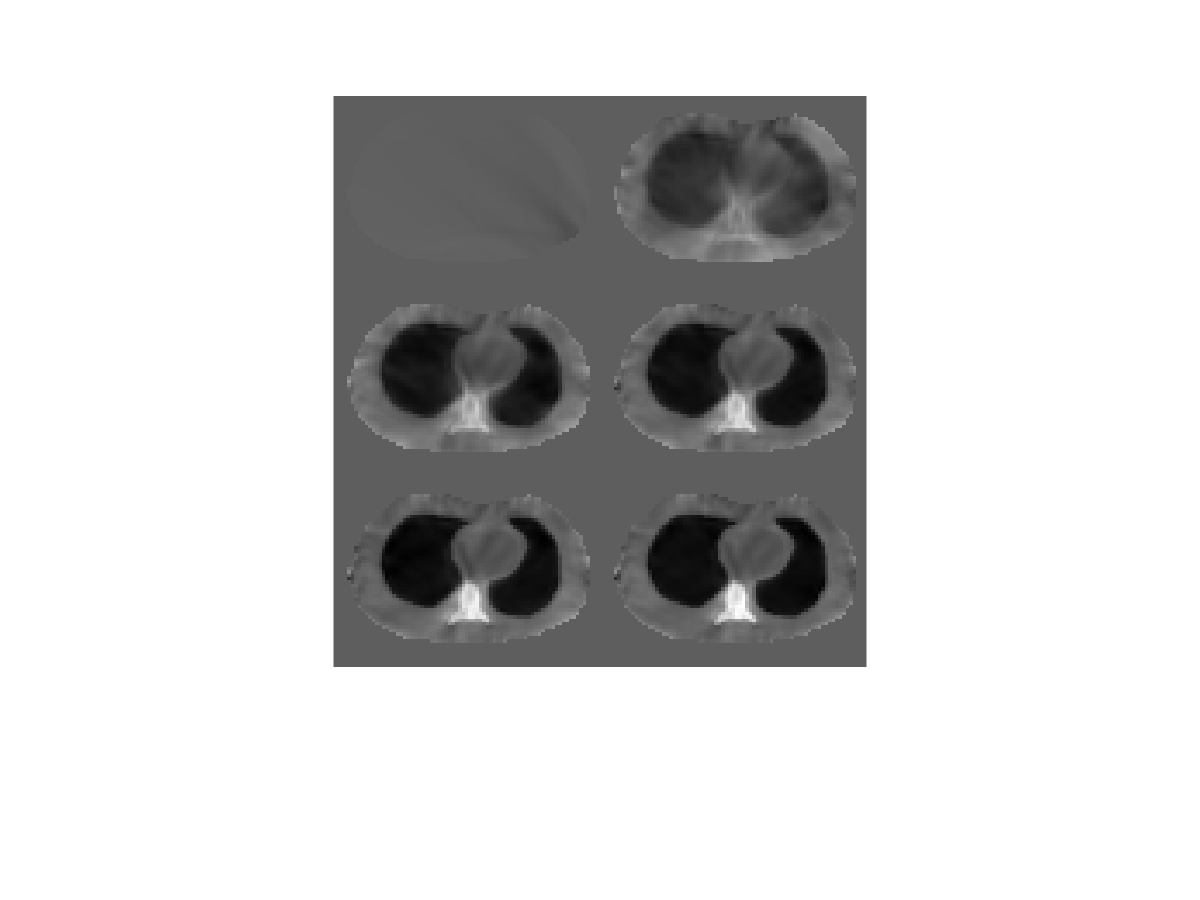
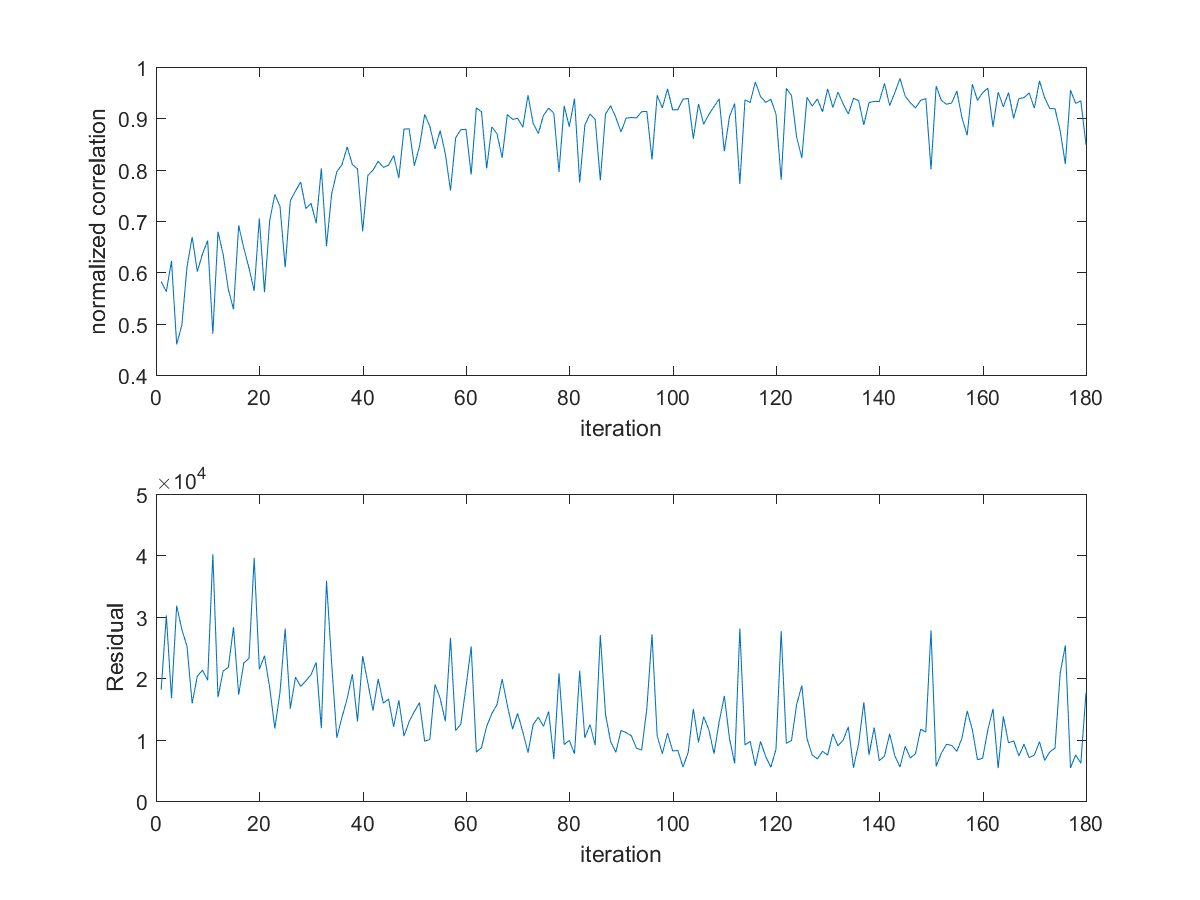


Figura 5.12 Resultado da primeira bateria de teste (I=3)

Resultado

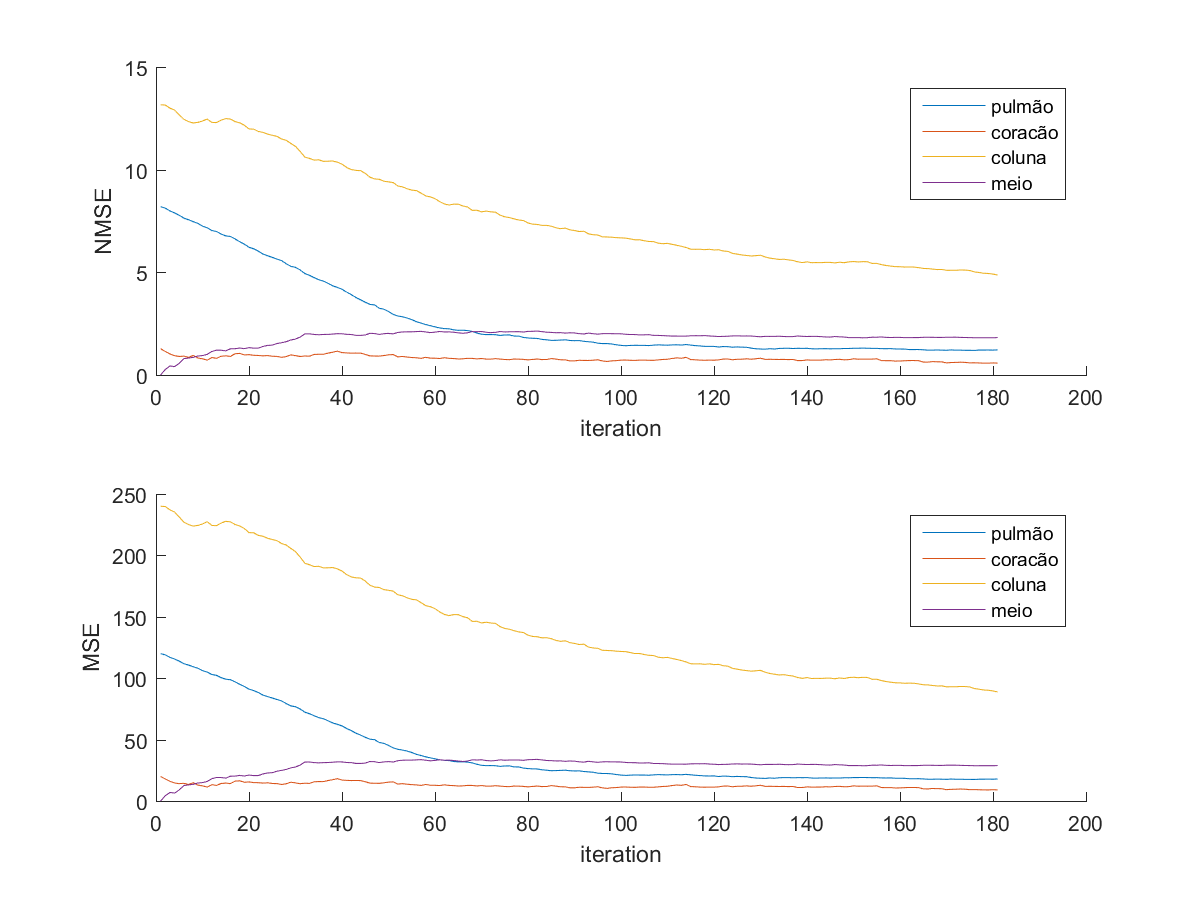
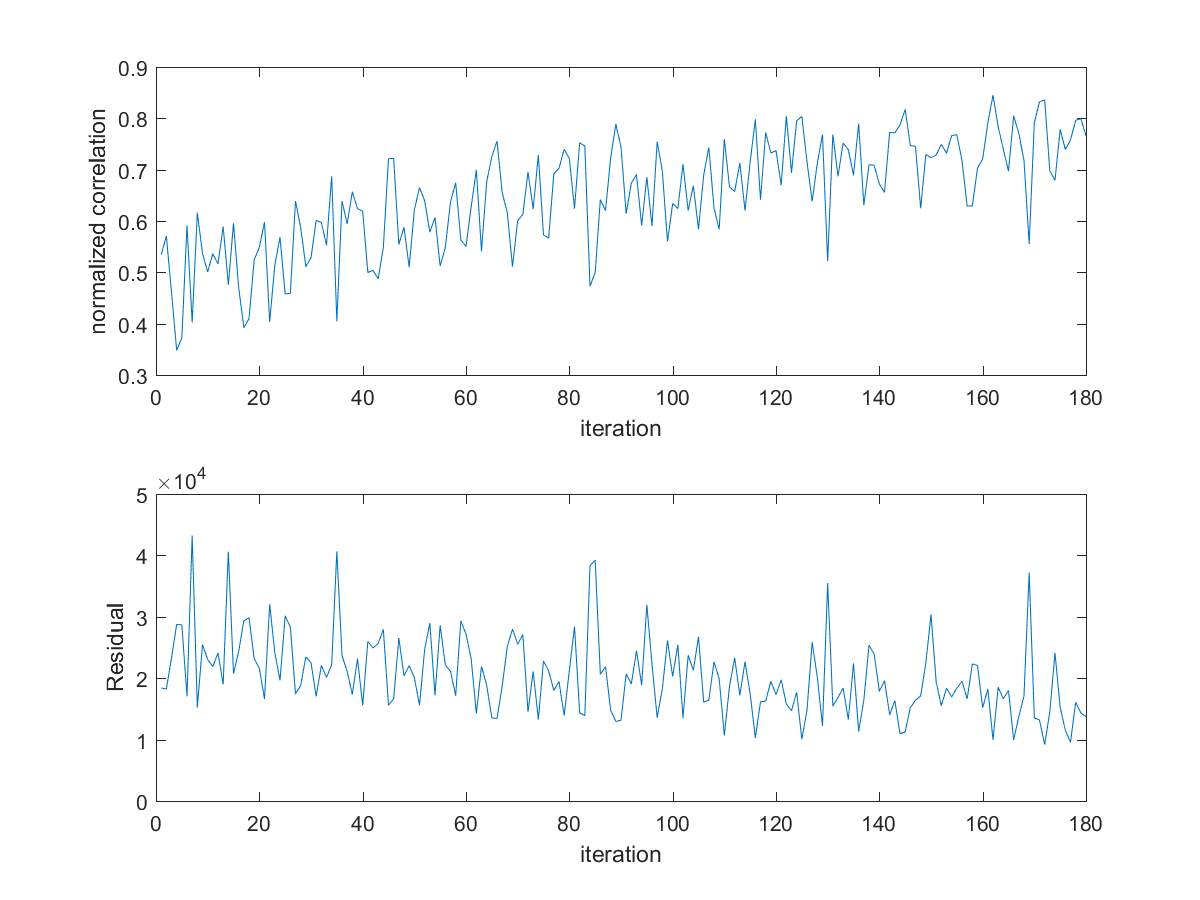


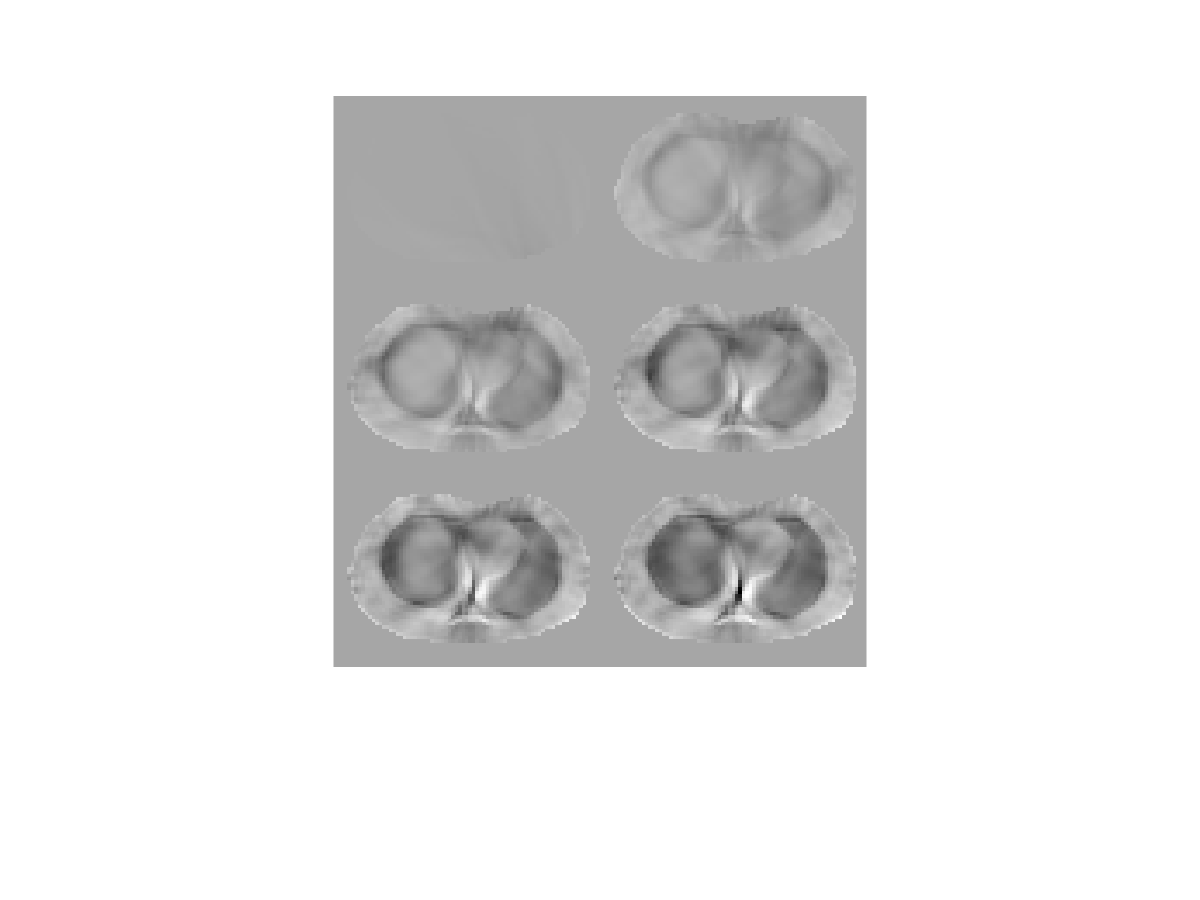
Figura 5.13 Resultado da primeira bateria de teste (I=6)

Nos testes para , vê-se que o algoritmo foi capaz de recuperar o *phantom* original com bastante qualidade. O MSE e o NMSE de todas as regiões ficaram menores que 6%. A correlação atingiu valores maiores que 0.9, e o residual sofreu uma redução de 150%.

Além disso, pode-se dizer que o algoritmo tem uma boa capacidade para encontrar os valores do coeficiente que são menores que a média do meio (no caso a região pulmonar), enquanto, para índices maiores, ele não possui a mesma capacidade. Basta notar que a redução do NMSE do pulmão foi de 8% para menos de 3%, ‘confundindo-se’ com o erro do meio e do coração, esse erro é o mínimo que o algoritmo consegue atingir. Já a coluna, apesar de ser bem identificável na imagem e do seu NMSE reduzir relativamente bastante do original (8%), ainda tem o seu erro alto (6%), se comparada às outras regiões.

Outro ponto a se destacar é que, conforme aumenta o contraste da imagem, o coração e o meio tendem a se misturar na imagem visualizada e, como o algoritmo possui um erro de aproximadamente 5%, essas duas regiões tendem a se misturar, impossibilitando a separação (identificação) das duas regiões. Esse é o ponto mais crítico do algoritmo e do processo de tomografia por ultrassom.

Resultado



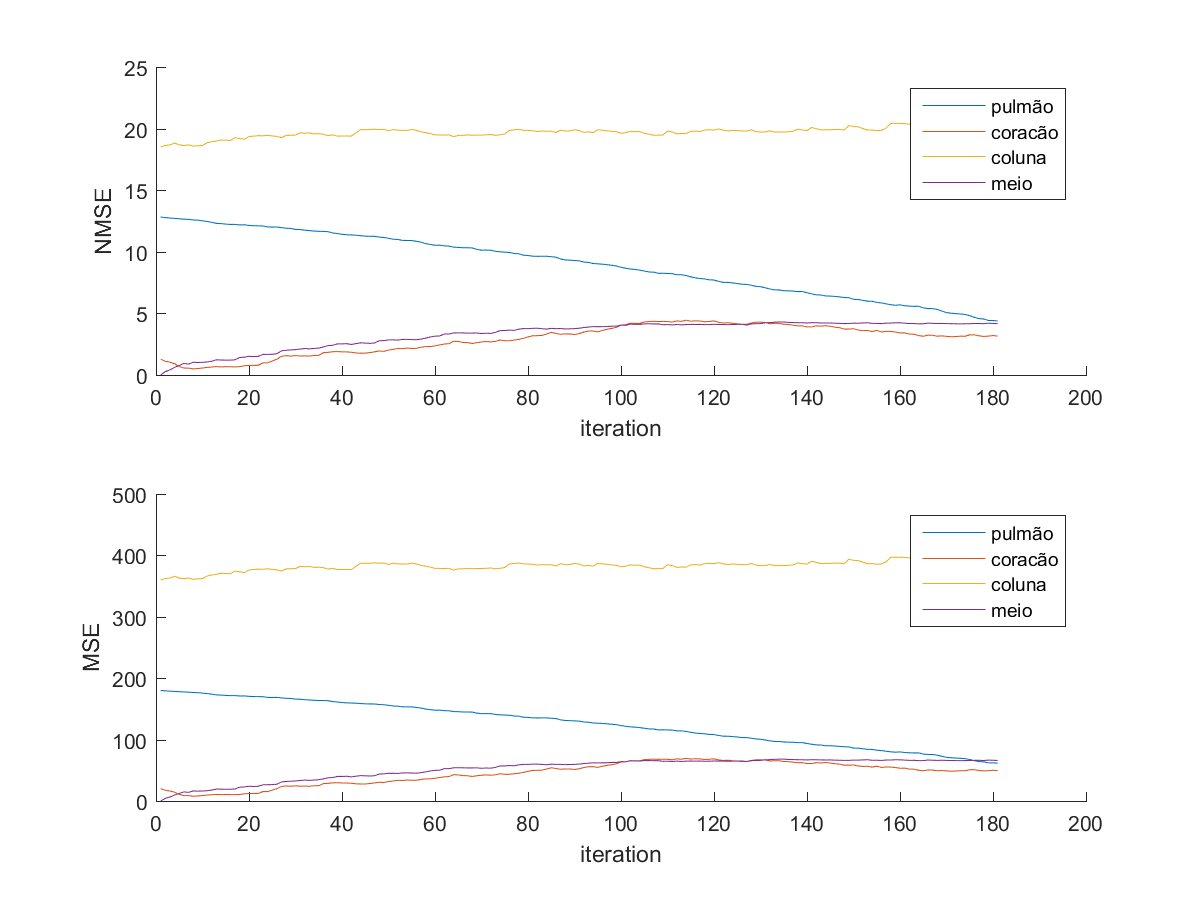


Figura 5.14 Resultado da primeira bateria de teste (I=9)

O teste para deixa muito evidente a fraqueza do algoritmo com relação ao contraste do *phantom*. Embora o algoritmo esteja convergindo no sentido da minimização do residual e na maximização da correlação, a imagem tomográfica final não está, pois, visualmente, pode-se afirmar que a coluna não foi identificada, e, em seu lugar, tem um artefato. Contudo, o algoritmo foi capaz de identificar o pulmão (quem tem o coeficiente menor do que o meio), encontrando um valor significativo para o coeficiente , uma vez que o NMSE da região pulmonar chegou à ordem de 5%.

**2ª Bateria –** Teste de Contraste , partindo do meio com a mesma forma que o *phantom* mais com um contraste diferente.

|  |  |
| --- | --- |
| **Parâmetros constantes** | |
| N *phantom* | 256 |
| CFL *phantom* | 0.1 |
| N algoritmo | 96 |
| CFL algoritmo | 0.3 |
| Tipo de Borda | Anel quadrado |
| nSenores | Continuo (694) |
| nFontes | 60 |
| nIterações | 180 |
| Fator media | 0.025 |
|  | 3 |
| **Parâmetros alterados** | |
| *phantom* | 6 e 15 |
| Meio inicial | I = 6, I = 9, I = 12, I = 15 |

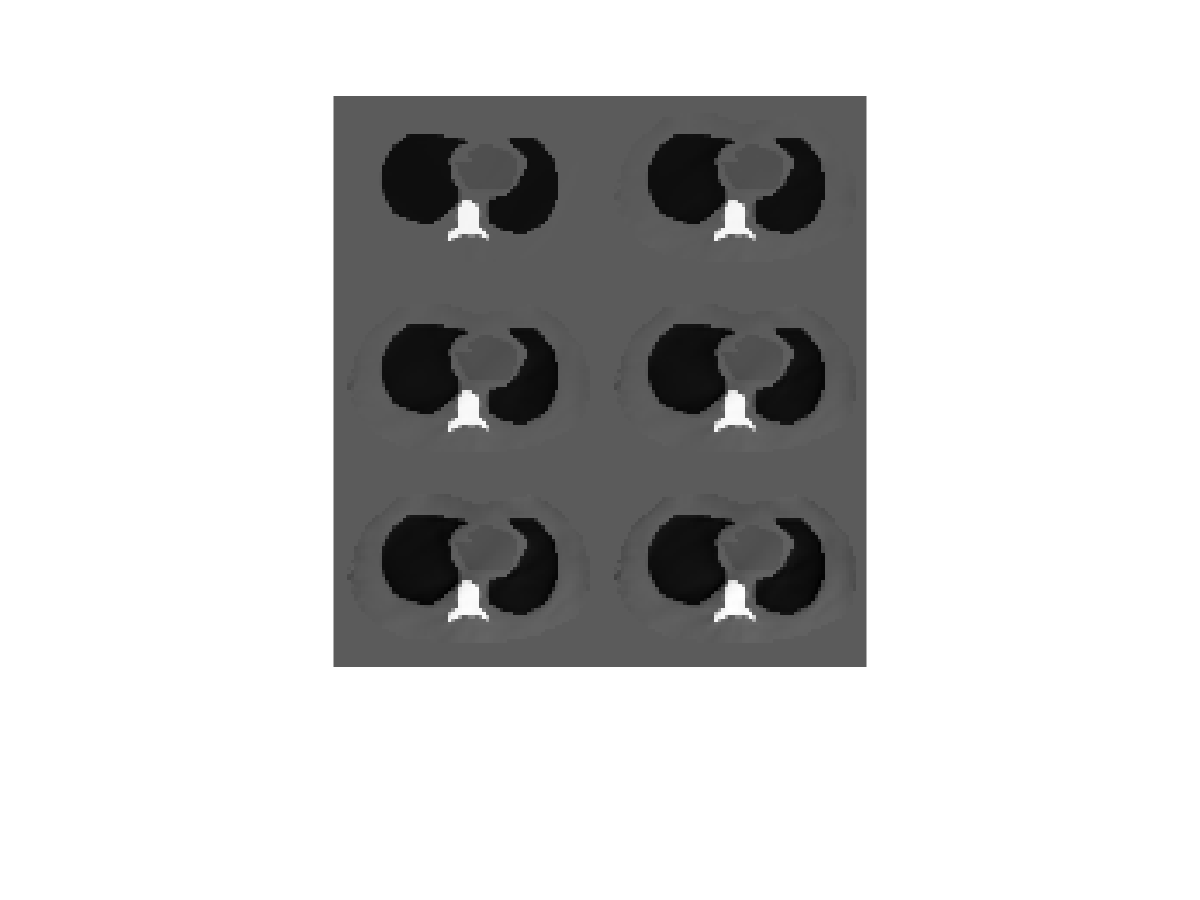
Esse primeiro teste (Figura 5.15) revela que o método é estável, quando se parte de uma imagem tomográfica igual ao *phantom* original, desde que ambos tenham níveis de contraste que respeitem o limite do algoritmo, o que ficará mais evidente após analisar os próximos testes (Figura 5.16).

Vale observar que os erros NMSE e MSE não partem do zero, o que era esperado, já que o meio inicial é igual ao meio do *phantom*, porém, devido à discretização espacial e ao modo como o algoritmo interpola/redimensiona as matrizes, surge erros numéricos que são presentes em todas os testes realizadas.

Outro ponto a se destacar é que este experimento mostra que o algoritmo não converge fortemente, isto é, apesar da solução encontrada ser a solução exata, o algoritmo continua ‘aprimorando’ o meio, porém ele acaba deteriorando a solução ao invés de melhorá-la.

Isso se deve ao fato que a solução do problema inverso não é única e ela não é necessariamente um atrator e pode inclusive ser uma solução instável.

Resultado de = 6 e 6



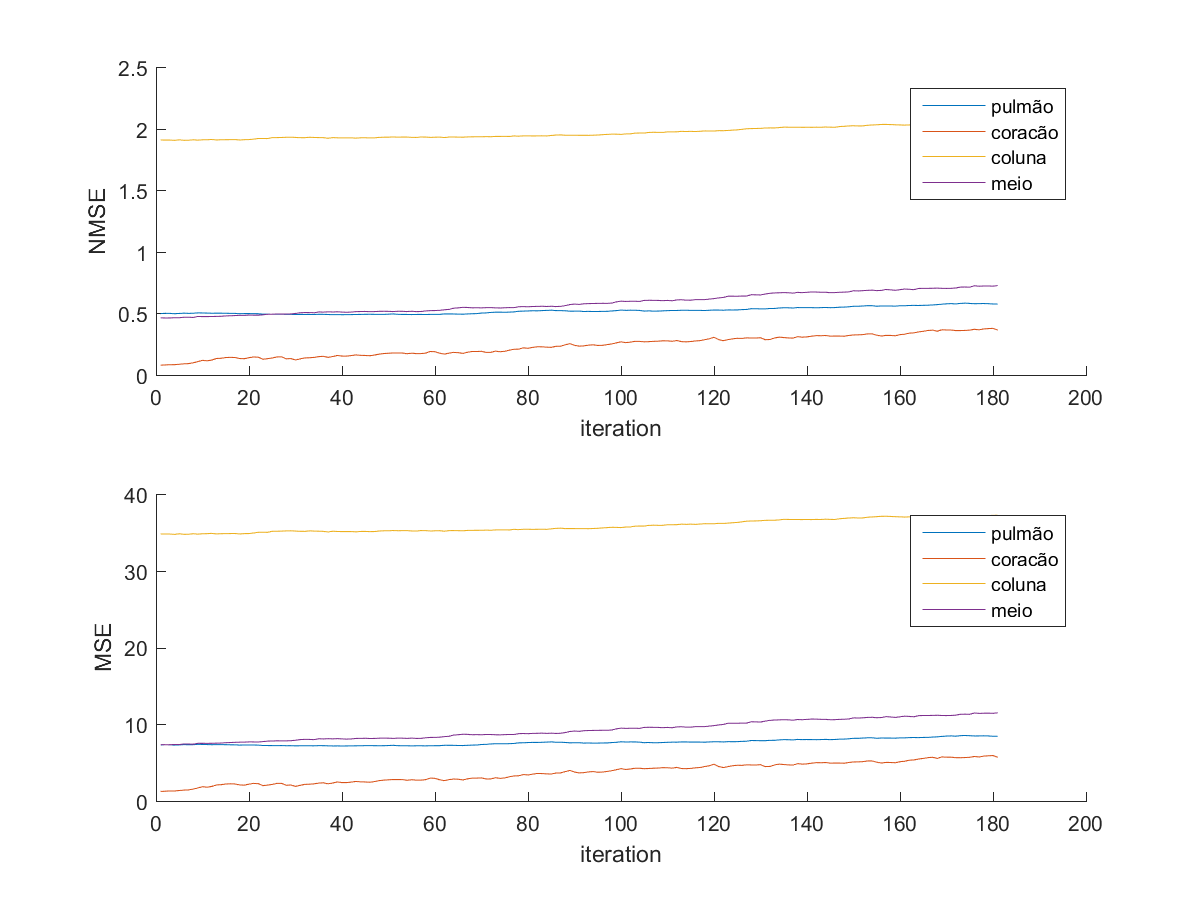
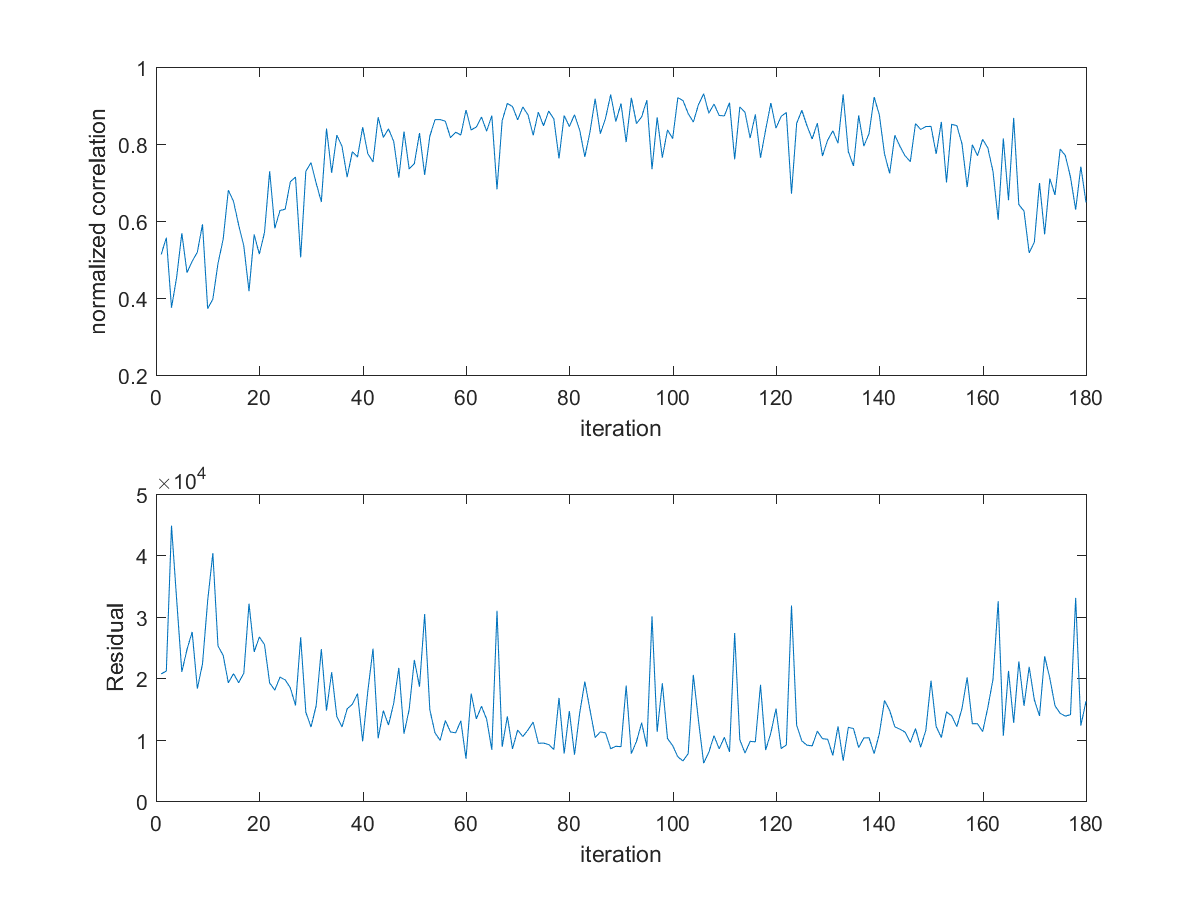
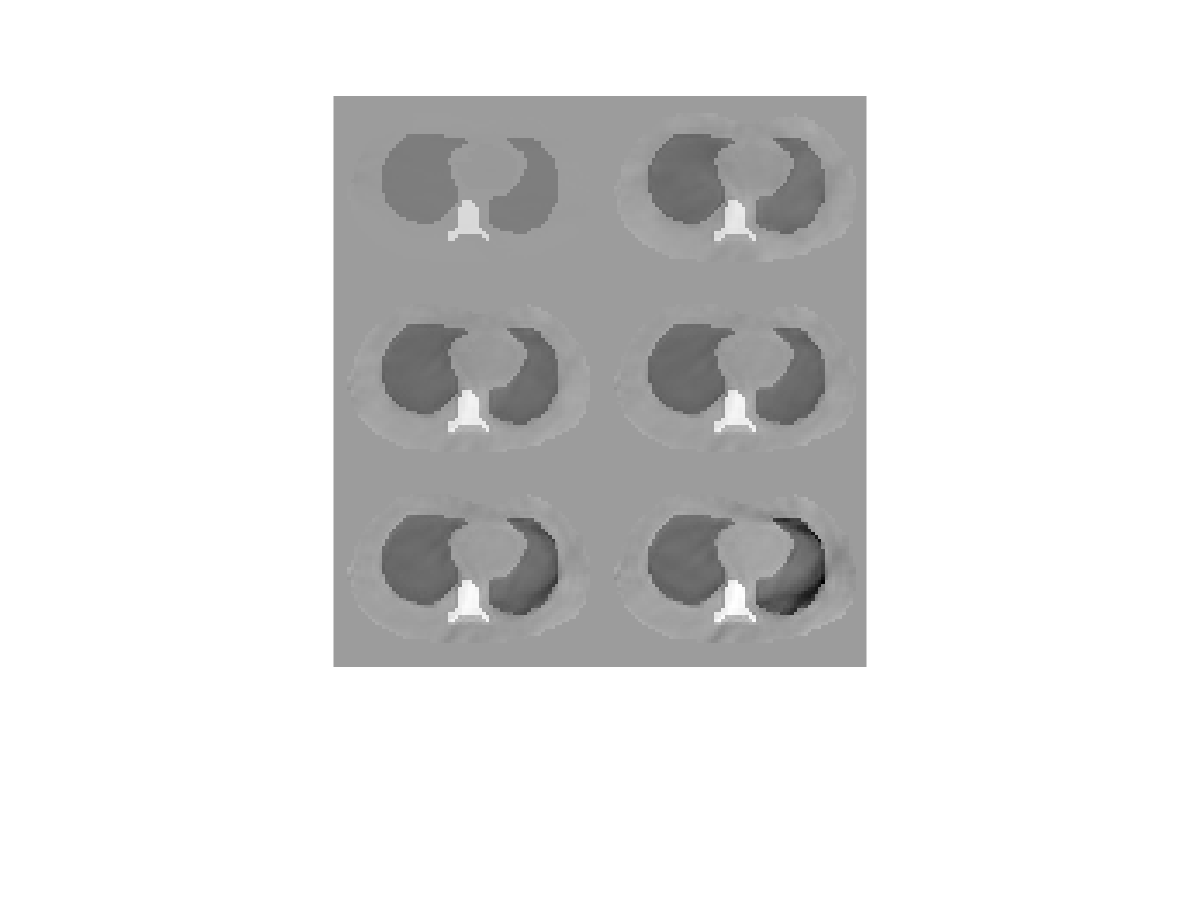


Figura 5.15 Resultado da segunda bateria de teste. = 6 e 6

Resultado de = 15 e 9



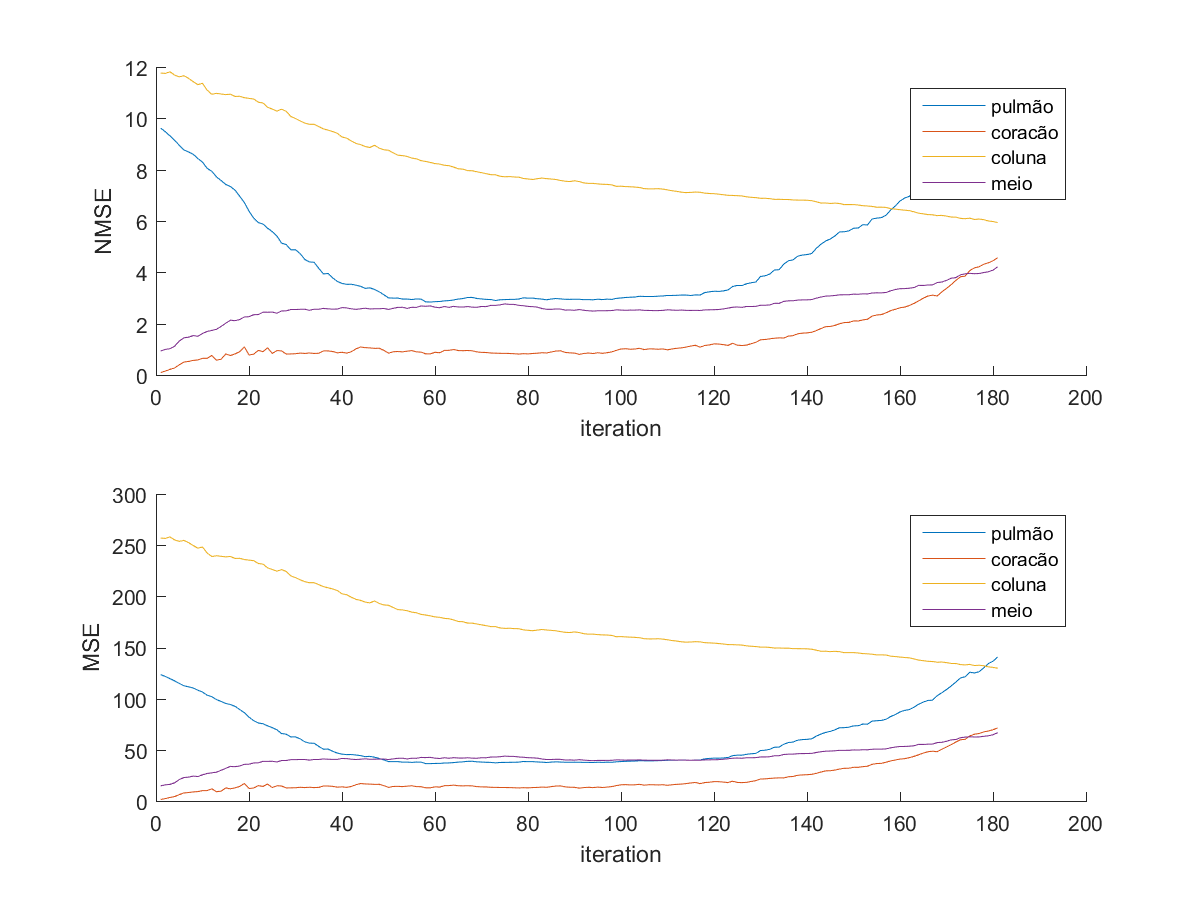
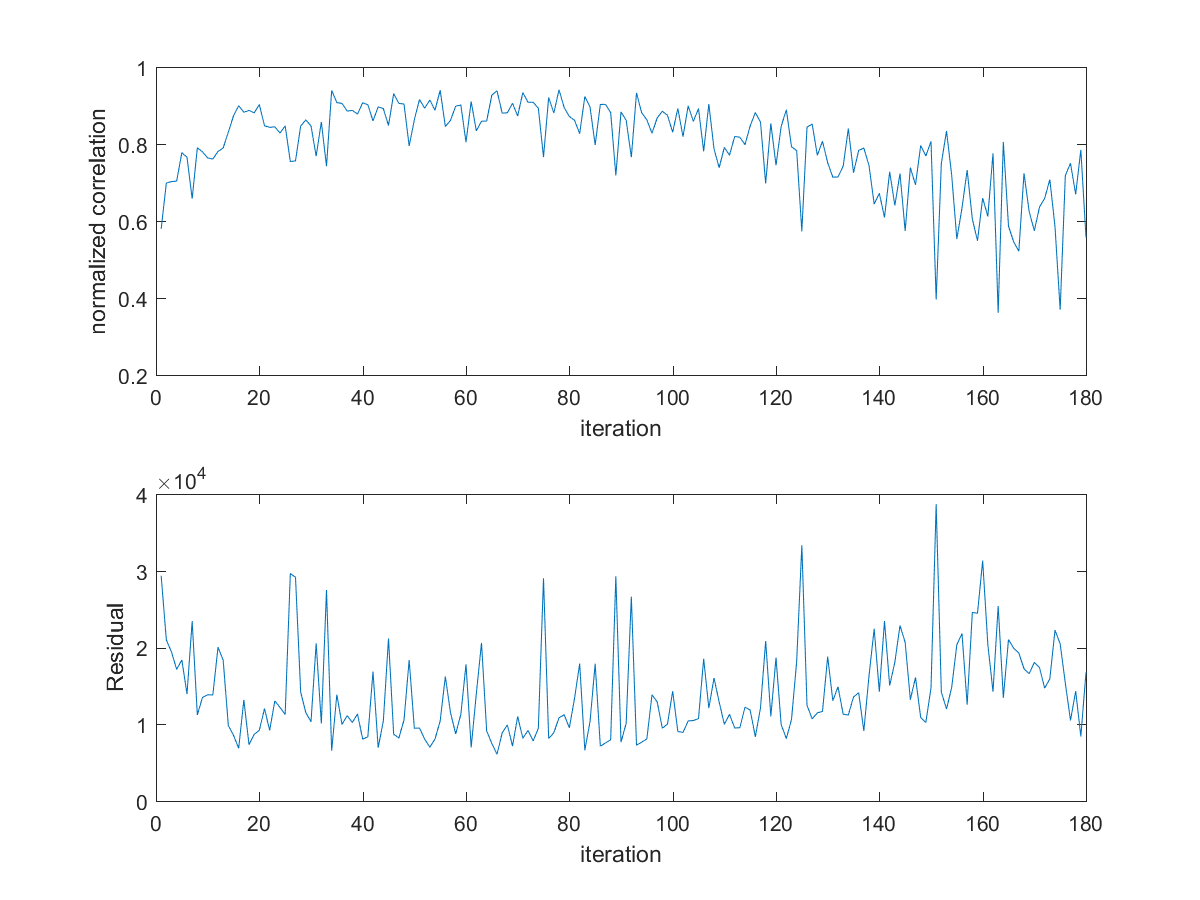
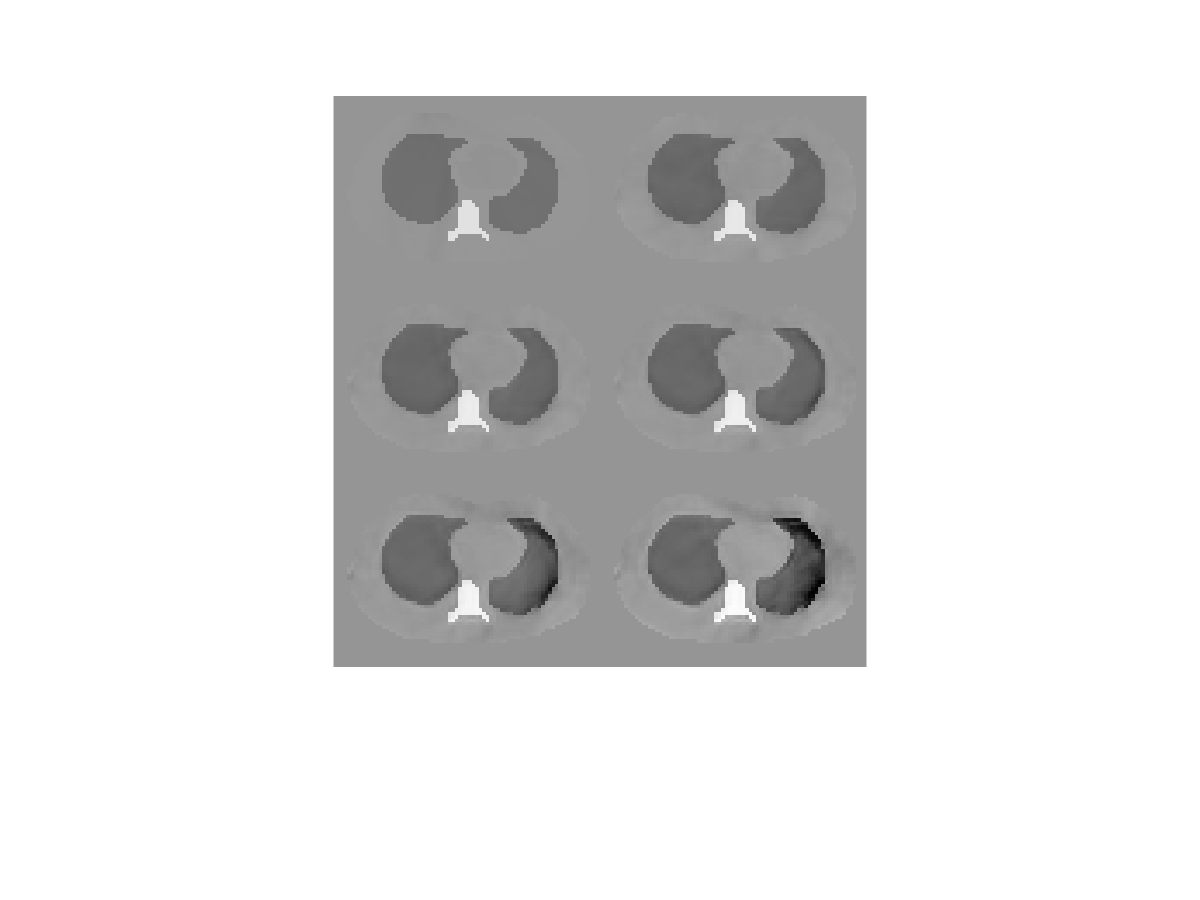


Figura 5.16 Resultado da segunda bateria de teste. e

Esse experimento foi feito para verificar a hipótese de que o algoritmo consegue convergir para altos contrastes, desde que se parta de um meio cujas diferenças entre os índices do *phantom* de ensaio (original) e do *phantom* inicial sejam menores que o limite de contrastes do algoritmo (I=6). Assim, o *phantom* inicial tem contrastes e o *phantom* do ensaio e a diferença entre ambos é I=6.

Contudo, essa hipótese pode ser descartada, pois, mesmo que o algoritmo tenha apresentado bons resultados inicialmente, começaram a surgir artefatos, a partir da iteração 110, que impediram o algoritmo de chegar a uma solução razoável.

Resultado de = 15 e 12



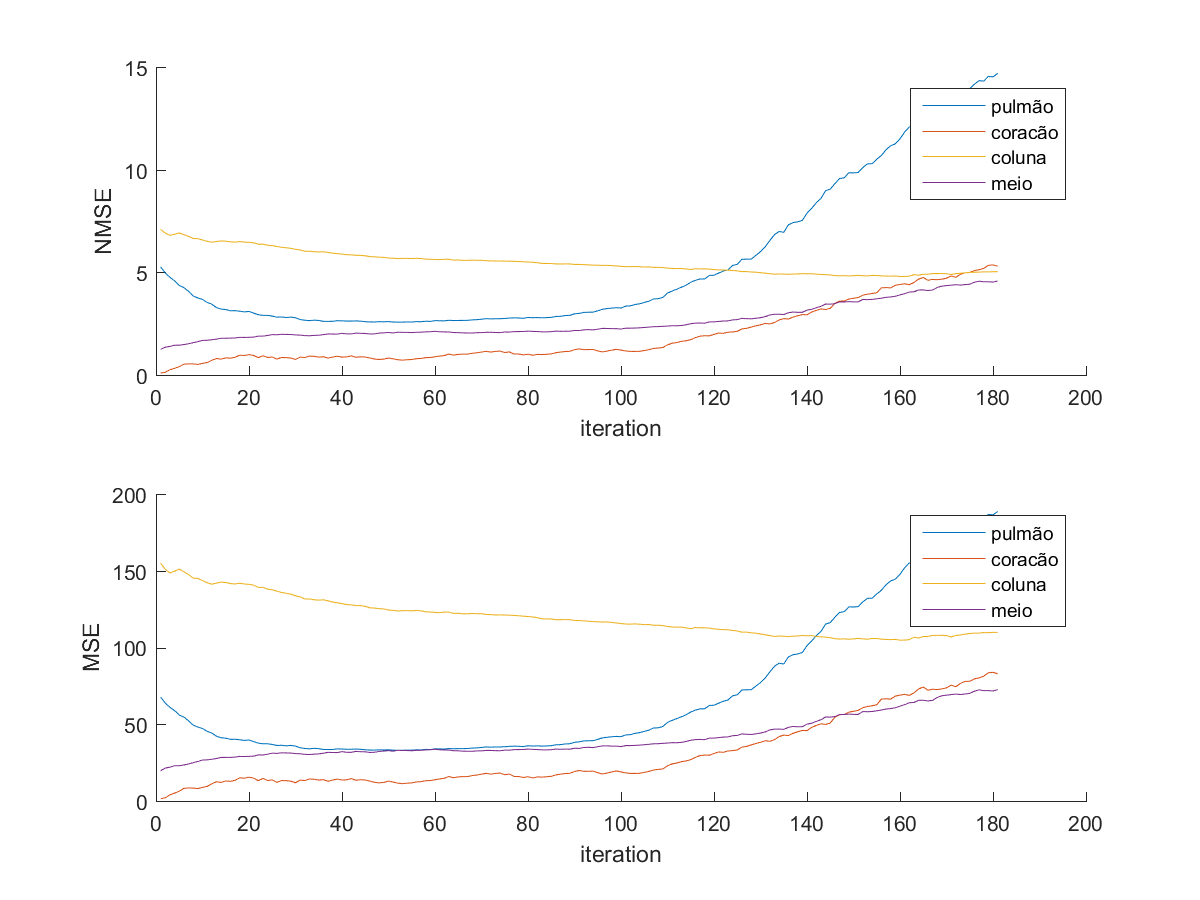
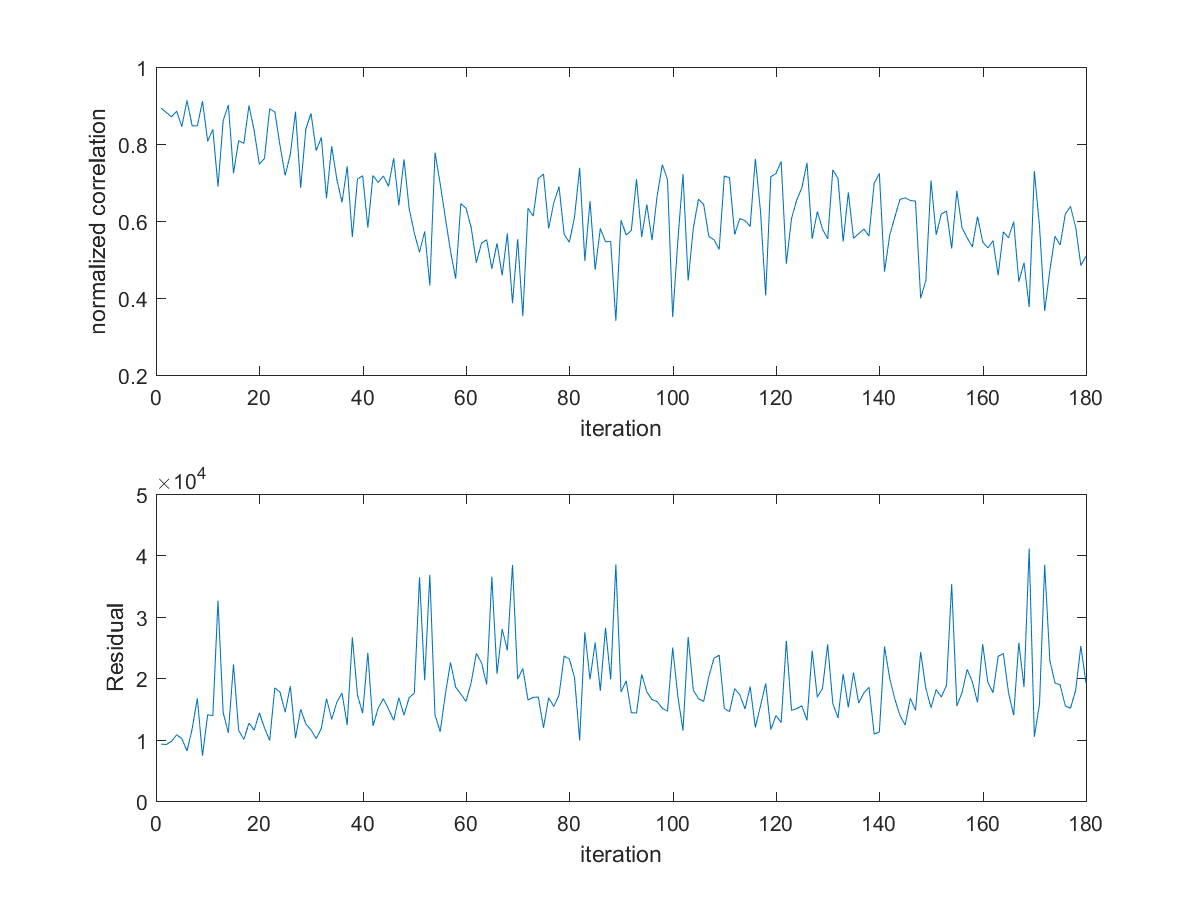
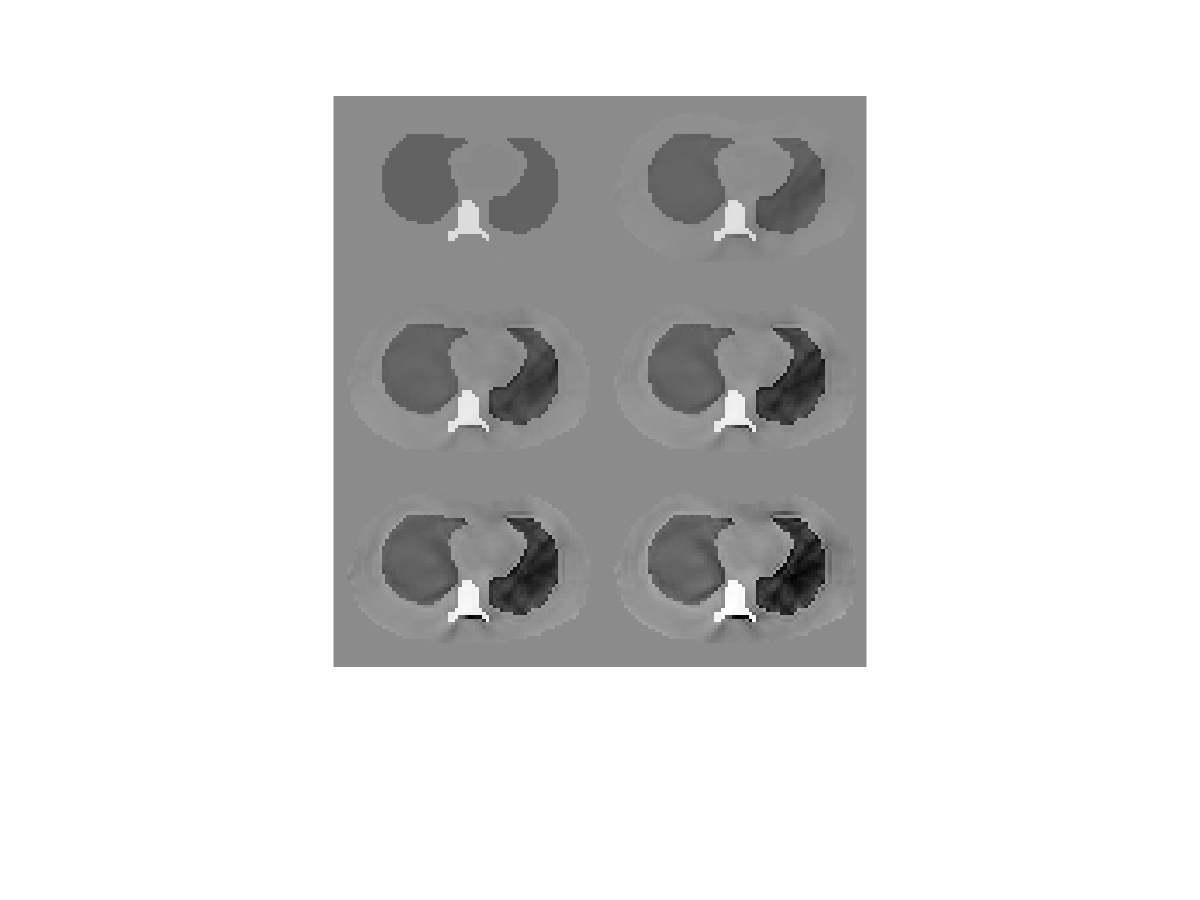


Figura 5.17Resultado da segunda bateria de teste. e

De modo semelhante ao experimento com 9, o algoritmo divergiu no processo, e artefatos surgiram na imagem. Em comparação com o experimento 9, a iteração na qual o processo começa a divergir é menor, por volta da iteração de número 85.

É importante observar que o artefato que surgiu, uma mancha no pulmão esquerdo, é o mesmo que no experimento com 9. Estudar a causa do surgimento desse artefato é um grande passo para o aprimoramento do algoritmo.

Resultado de = 15 e 15



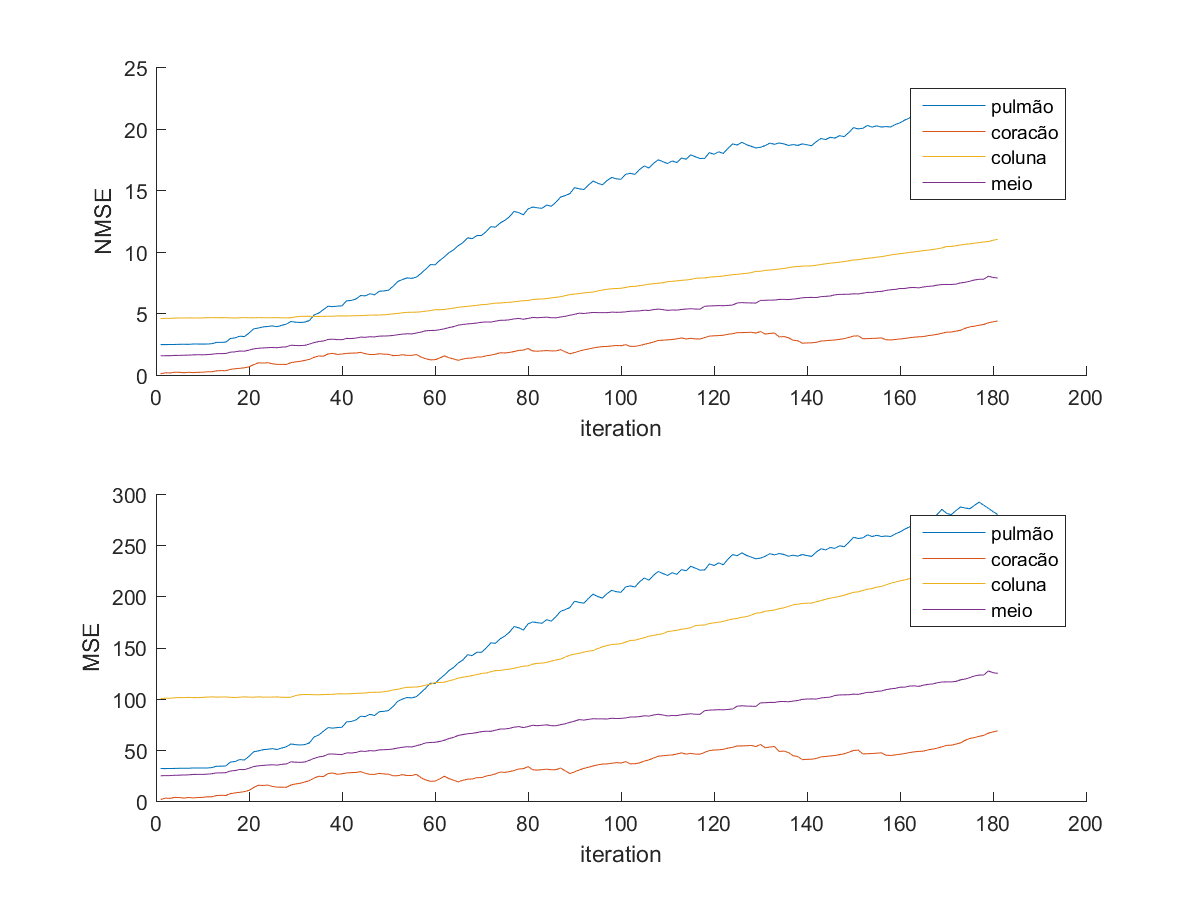


Figura 5.18 Resultado da segunda bateria de teste. e

Da mesma forma que para 9 e 12, o algoritmo divergiu, e, neste caso, foi bem precocemente, por volta da iteração de número 20. Além do mais, esse é um teste que mostra a incapacidade do algoritmo de funcionar em altos contrastes, pois, mesmo que ele encontre a solução do problema inverso, essa solução seria instável.

**3ª Bateria –** Teste de forma, partindo do meio com o mesmo contraste e consequentemente, os mesmos coeficientes de velocidade de propagação do som, mas com forma ligeiramente diferente.

|  |  |
| --- | --- |
| **Parâmetros constantes** | |
| N *phantom* | 256 |
| CFL *phantom* | 0.1 |
| *phantom* | 6 |
| N algoritmo | 96 |
| CFL algoritmo | 0.3 |
| Tipo de Borda | Anel quadrado |
| nSenores | Continuo (694) |
| nFontes | 60 |
| nIterações | 180 |
| Fator media | 0.025 |
|  | 3 |
| **Parâmetros alterados** | |
| Meio inicial | Figura 5.19 |

Resultado: Meio inicial com a seguinte forma

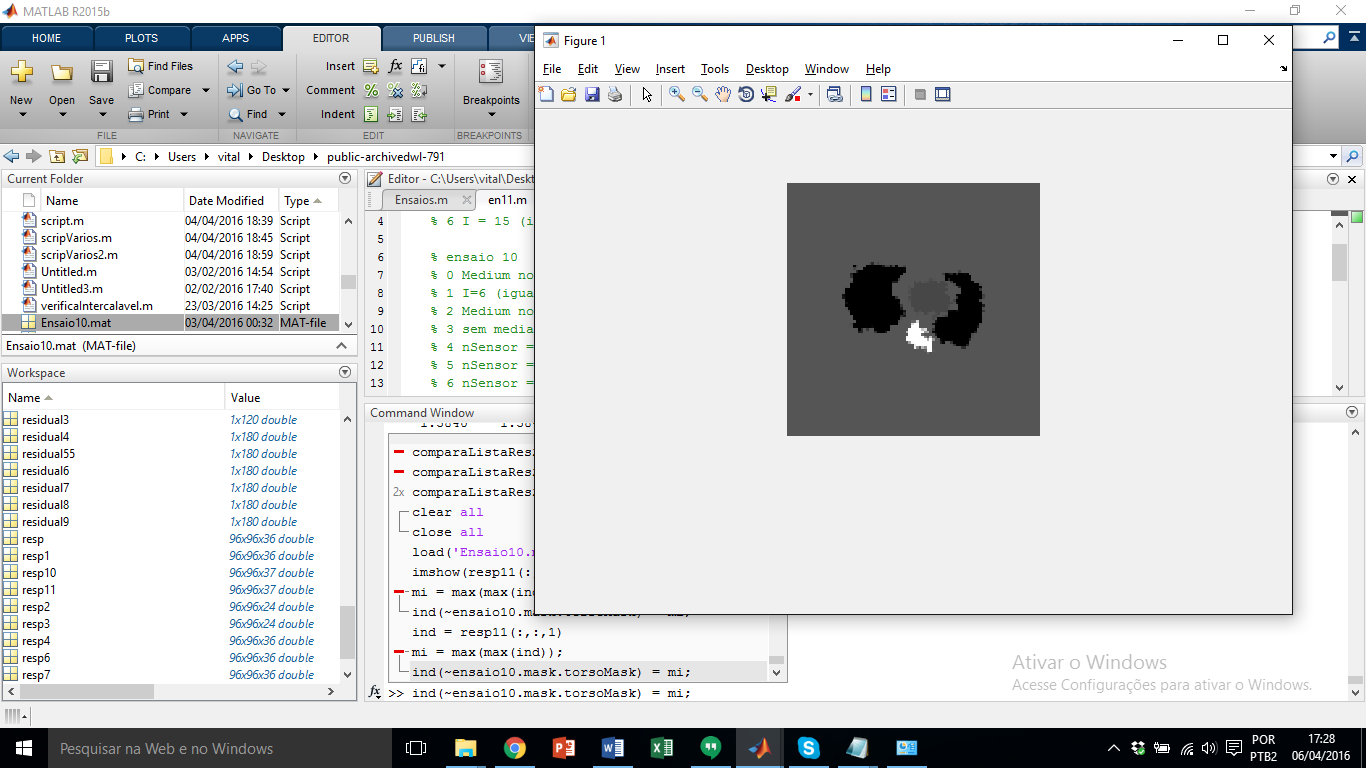
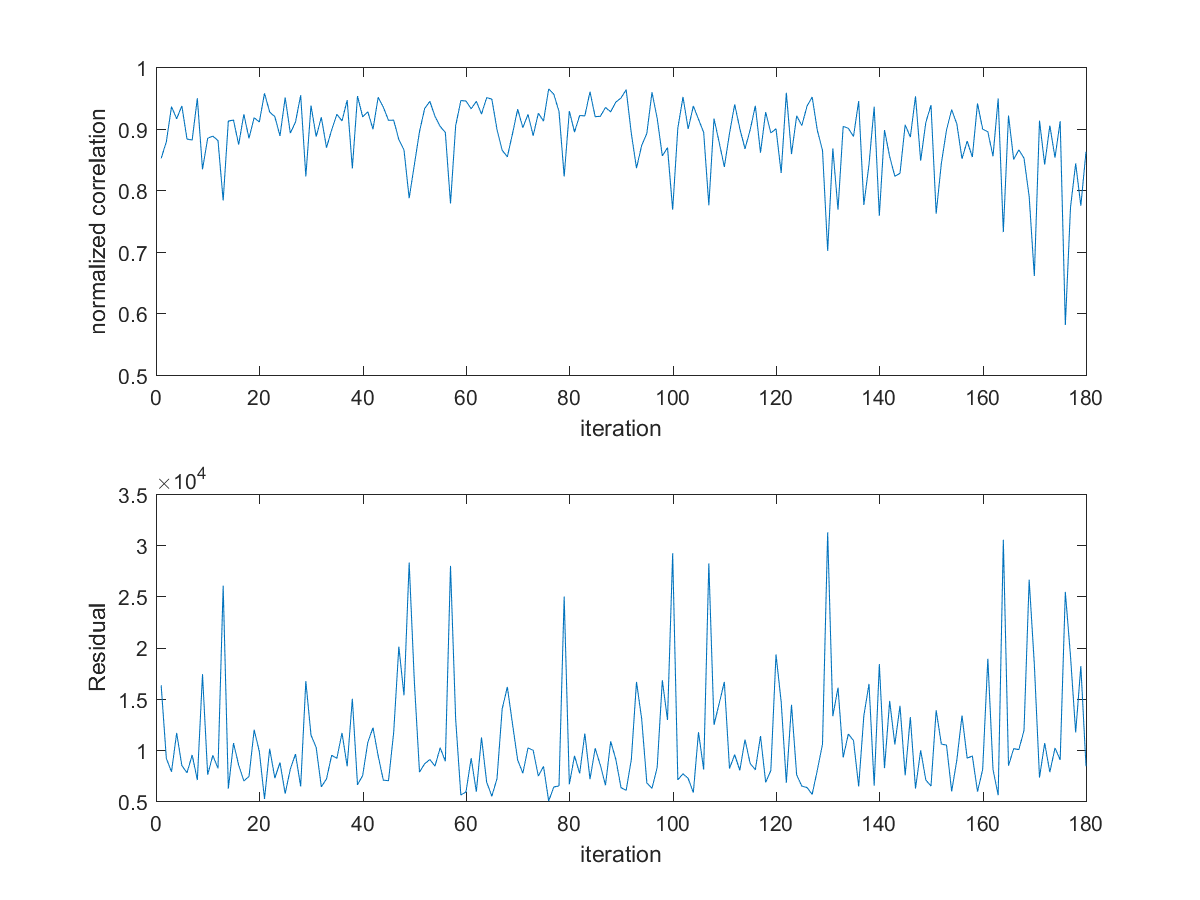


Figura 5.19 Imagem inicial deformada



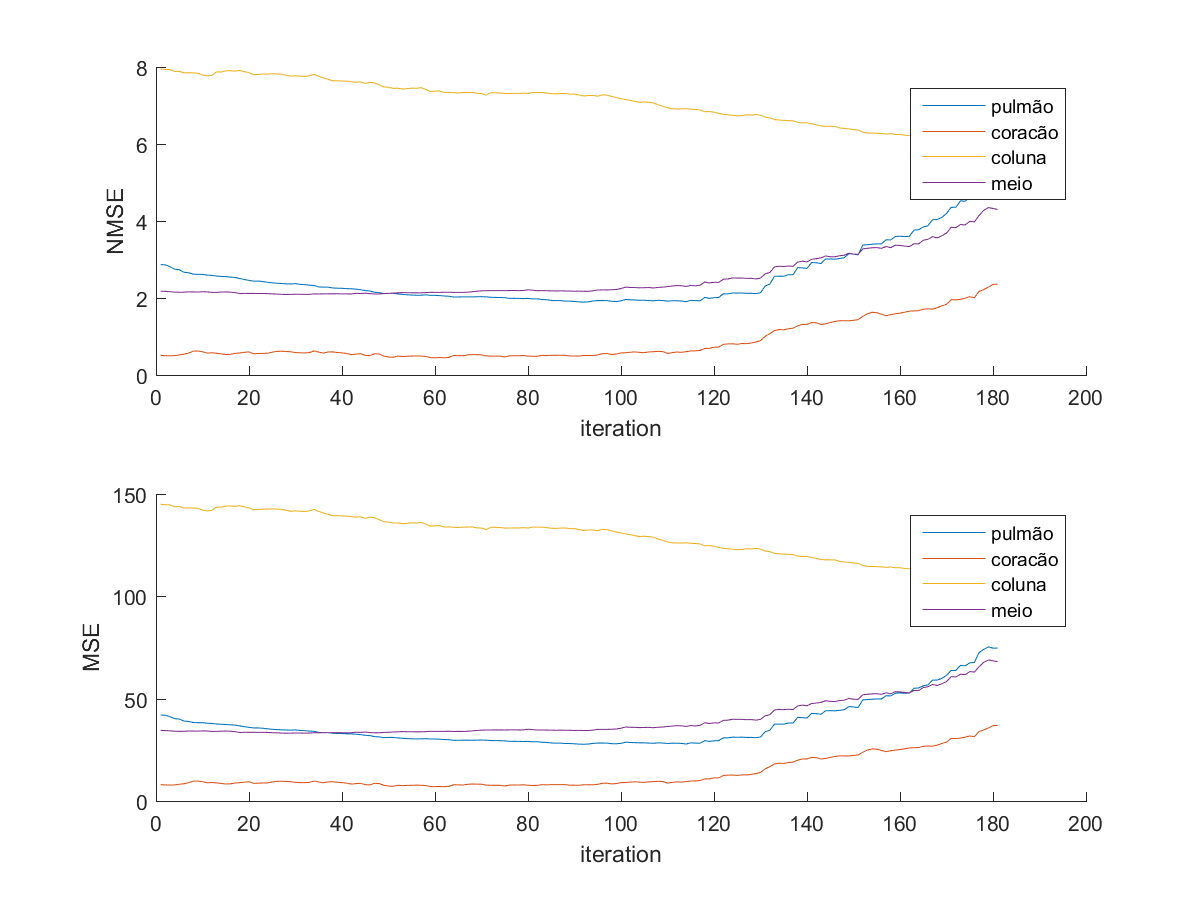


Figura 5.20- Resultado da terceira bateria

Inicializar o meio com uma imagem tomografia semelhante à original, mas com forma ligeiramente diferente, seria algo muito interessante para melhorar a *performance* e o uso do algoritmo. No teste, pode-se perceber que, inicialmente, o algoritmo consegue identificar as diferenças existentes entre o *phantom* inicial e o *phantom* do ensaio (ver os círculos na figura 5.21 ). Mas, com o avanço das iterações, um artefato surge, prejudicando o algoritmo.

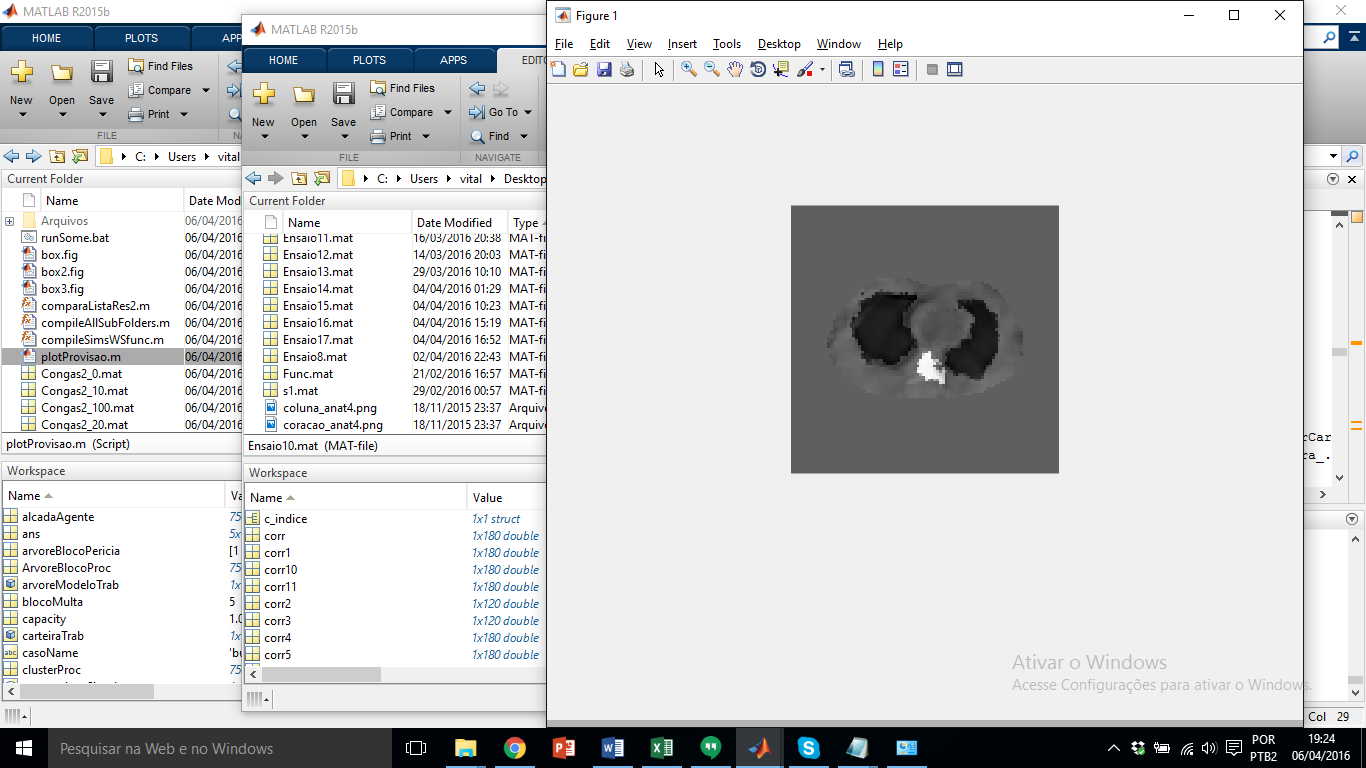


Figura 5.21 Reconstrução com meio inicial deformado

**4ª Bateria –** Teste com diferentes números de sensores, interpolando os dados necessários, já que o algoritmo exige que todas as informações de variação de pressão em torno de um contorno do torso sejam conhecidas.

|  |  |
| --- | --- |
| **Parâmetros constantes** | |
| N *phantom* | 256 |
| CFL *phantom* | 0.1 |
| *phantom* | 6 |
| N algoritmo | 96 |
| CFL algoritmo | 0.3 |
| Tipo de Borda | Anel quadrado |
| Meio inicial | Neutro (1854) |
| nFontes | 60 |
| nIterações | 180 |
| Fator media | 0.025 |
|  | 3 |
| **Parâmetros alterados** | |
|  | 60,40,30,20,10 |

Comparação do resultado diferentes números de sensores

|  |  |
| --- | --- |
|  |  |
| C:\Users\vital\Desktop\resultadoTomo\10a.png | 11a |
|  |  |
| 12a | 13a |

*Figura 5.22 Comparação visual da quarta bateria de teste*

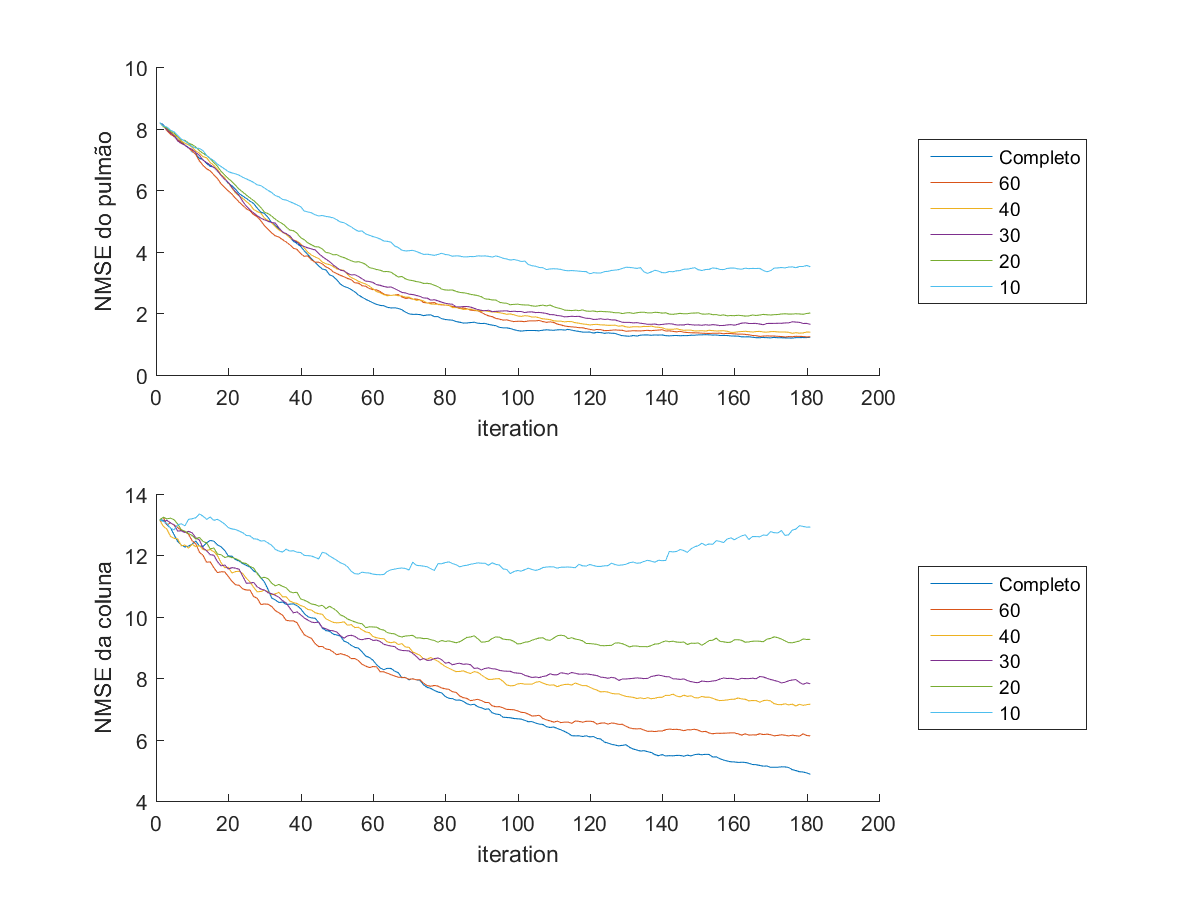


Figura 5.23 Comparação entre o NMSE do pulmão e da coluna para diferentes números de sensores (Quarta bateria)

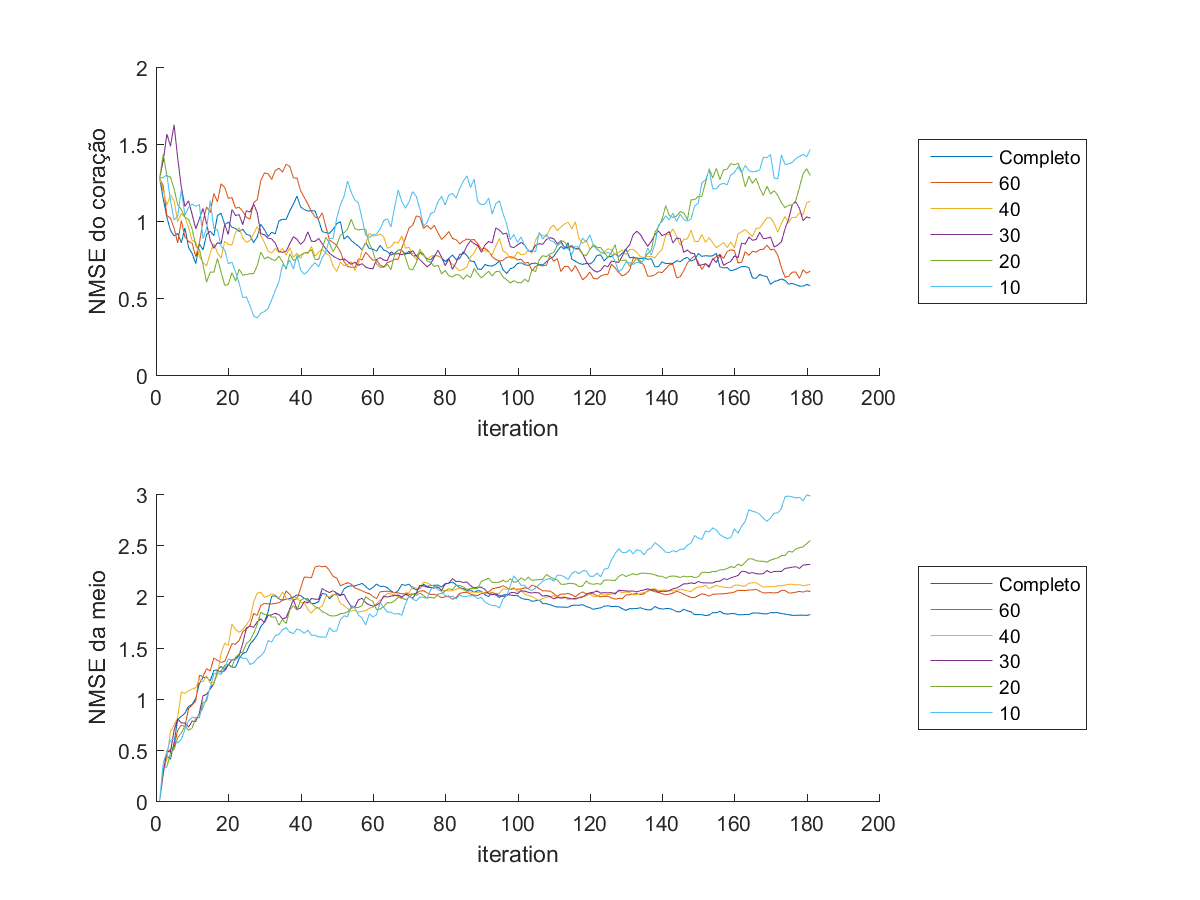


Figura 5.24 Comparação entre o NMSE do coração e do meio para diferentes números de sensores (Quarta bateria)

Os resultados mostram que a técnica de interpolação dos sensores é uma aproximação muito boa da realidade. Desconsiderando a região da coluna, ao se usar 40 sensores ou mais, o NMSE é comparável ao NMSE do algoritmo quando se usa um contorno completo de sensores.

Mesmo usando 20 ou 30 sensores, a aproximação continua suficiente boa para recuperar a *phantom*.

**5ª Bateria –** Teste com diferentes números de fontes emissoras.

|  |  |
| --- | --- |
| **Parâmetros constantes** | |
| N *phantom* | 256 |
| CFL *phantom* | 0.1 |
| *phantom* | 6 |
| N algoritmo | 96 |
| CFL algoritmo | 0.3 |
| Tipo de Borda | Anel quadrado |
| Meio inicial | Neutro (1854) |
| nSensores | Continuo (694) |
| nIterações | 180 |
| Fator media | 0.025 |
|  | 3 |
| **Parâmetros alterados** | |
| nFontes | 60,40,30,20,10 |

Comparação do resultado diferentes números de fontes emissores

|  |  |
| --- | --- |
|  |  |
| 1a | 15a |
|  |  |
| 15a | 17a |
|  | Original |
| 17a | 7a |

*Figura 5.25 Comparação visual com diferentes números de fontes emissores (Quinta Bateria)*

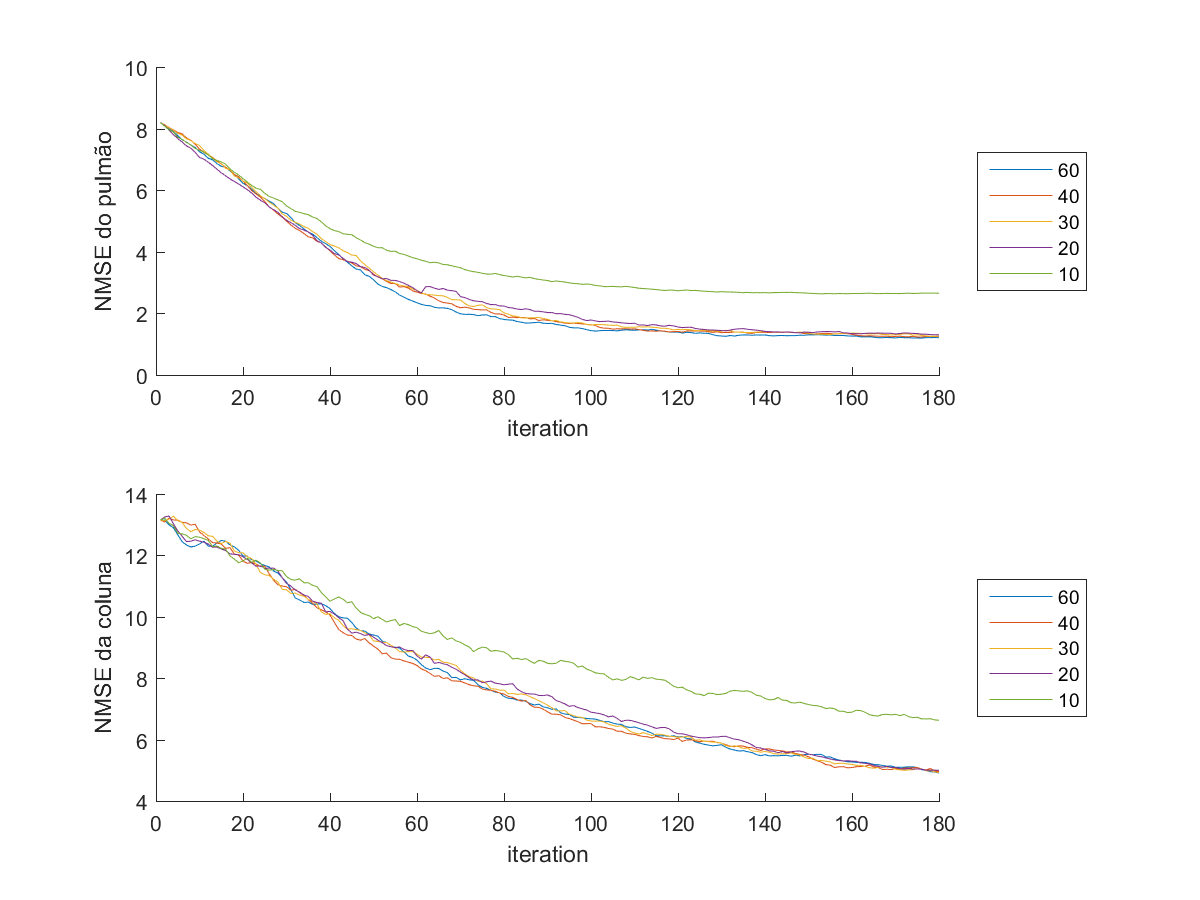


Figura 5.26 Comparação entre o NMSE do pulmão e da coluna para diferentes números de fontes (Quinta bateria)

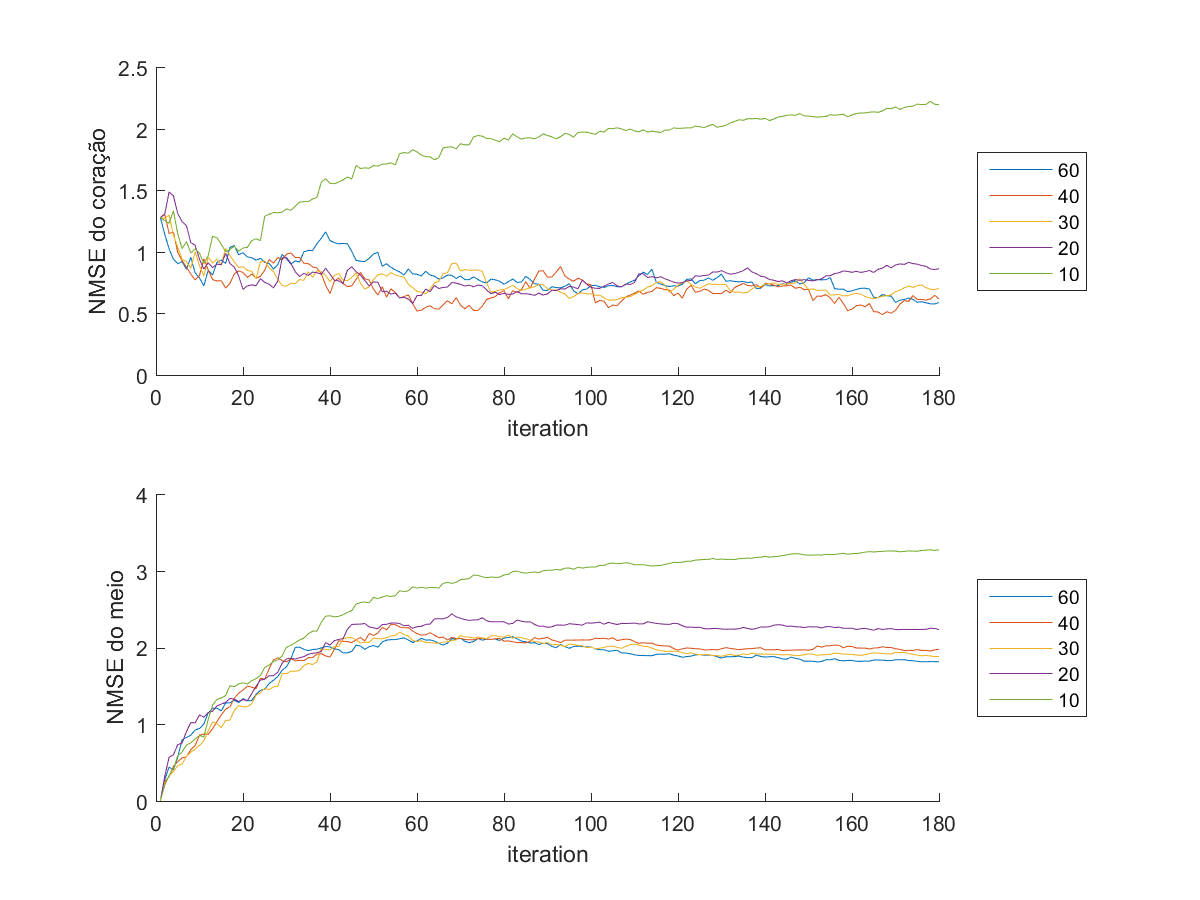


Figura 5.27 Comparação entre o NMSE do coração e do meio para diferentes números de sensores (Quinta bateria)

A influência do número de fontes emissoras é bem interessante, usando uma quantidade mínima de 20 fontes emissoras, o MSE e o NMSE dos diferentes experimentos são muito semelhantes e próximos entre si.

Porém, a análise visual mostra que o MSE não é uma métrica que, por si só, seja suficiente para julgar a qualidade da imagem tomográfica, pois a imagem com 30 ou 20 fontes emissoras é mais ‘manchada’ que a imagem com 40 ou 60 fontes emissores.

Algo importante para dizer é que o número de iterações em cada experimento são os mesmos, mas o número de varreduras não. Em todos os testes, foram feitas 180 iterações, mas, quando se usou 60 fontes, isso equivale a 3 varreduras, com 30 fontes, a 6 varreduras. Mas o mais relevante para se notar é que, para qualquer número de sensores, o gráfico do erro acompanhou o número de iterações e não o número de varreduras, ou seja, o que interessa para o algoritmo é que tenha alguma dupla de emissores-sensores para ser usado no aprimoramento, se essa dupla já foi usada 3 ou 5 vezes pouco influencia.

Assim, se os dados coletados durante alguma simulação estiverem comprometidos, pode-se executar o algoritmo sem esses dados, sem disturbar a imagem final.

Os dados podem estar comprometidos por diversas razões, como, por exemplo, muito ruído ou a posição da fonte emissora seja tal que o sinal gerado seja muito complexo para o algoritmo tratar ou tal que o algoritmo crie artefatos ao usar esses dados.

## Estrutura do código

O código que implementa o algoritmo de tomografia foi implementado em Matlab, uma vez que a ferramenta base do algoritmo o *k-wave* foi implementado nessa mesma linguagem.

Existem basicamente três tipos de código:

Os códigos responsáveis pela geração dos ensaios

Os códigos responsáveis pela execução do algoritmo de resolução do problema inverso

Os códigos que exibem a resposta do algoritmo.

### Códigos de geração dos ensaios

Para a geração do ensaio tem-se que usar duas funções: A função **poucoHeterogeoC** e a função **gerarEnsaio.**

A função **poucoHeterogeoC** cria um *phantom* com uma baixa heterogeneidade em velocidade e homogenia nos outros aspectos (densidade (), absorção ()) a partir das imagens de cada um dos órgãos da região torácica (torso, pulmões, coração e coluna). Nessa função os parâmetros de entrada são o índice de contraste (I) que indica o contraste de cada um dos órgãos e o tamanho do do *phantom* de saída.

O *phantom* criado é representado pelos 4 objetos de saída. O kgrid*, q*ue é o mesmo kgrid que o *k-wave* requisita para simular a propagação de onda; o medium, idem ao anterior; o mask que é uma que contém as máscaras de cada uma das 5 regiões (torso, pulmão, coração, coluna e meio); e o c\_indice que contém os a velocidade do som em cada uma das 5 regiões.

A função **gerarEnsaio** gera os sinais reais recebidos nos sensores em torno do torso a partir de uma função de excitação. As entradas da função são: kgrid; medium; func, que é a função de excitação; tipoBorda, que indica se os sensores estão presos ao torso ou formam um anel quadrado em torno ao torso; e nSource, que diz o número de fontes emissoras igualmente espaçadas que estão ao longo da borda.

Internamente, a função gerarEnsaio, usa duas outras funções, a função **defineSouce,** que define as fontes emissoras, e a função **defineSensor**, que define os sensores.

Os resultados obtidos nos sensores em gerarEnsaio junto com os objetos obtidos em poucoHeterogeoC criam a estrutura ensaio que será usada nos próximos códigos.

### Códigos do problema inverso

Existem duas maneiras de resolver o problema inverso, uma que considerar que todos os sensores que estão em torno do torso existem e serão usados na resolução do problema inverso. Nesse caso a interpolação dos dados nos sensores é desnecessária.

A segunda maneira é considerar que existem sensores apenas em alguns pontos da borda do torso. Assim, deve-se interpolar os dados disponíveis de modo que tenhamos informações em toda a borda do torso.

No primeiro caso deve-se usar as funções **controladorTomografiaSimples** e **tomografyIteration,** enquanto no segundo caso usa-se as funções **controladorTomografiaInterpolacao, interpolateSensorData** e **tomografyIteration.**

A função **controladorTomografiaSimples** é responsável por definir as condições iniciais do problema inverso e por controlar a resolução deste. As condições de resolução compreendem a definição do kgrid de resolução, dos sensores, das fontes emissoras e do meio inicial.

Considera-se que o do problema inverso é menor que do ensaio e que os sensores e as fontes emissoras do problema estão localizados na mesma posição cartesiana que as fontes e sensores do ensaio. Para as fontes emissores isso não é um problema, mas para os sensores sim. Pois como o kgrid do problema é menor que a do ensaio, os sensores (números e contínuos) sobrepõem-se quando rescaldos para o grid menor. Portanto, o algoritmo deve decimar alguns sensores.

O meio inicial que o algoritmo tomará como partida pode ser de 3 tipos: uma aproximação do objeto do ensaio, um meio neutro ou matriz com o índice de velocidade (c).

A aproximação do objeto é obtida quando a entrada cinicial é igual a 0, nesse caso, a função **aproximacaoInicialPoucoHeterogeneoC** será chamada criando um meio inicial semelhante em forma e em índice ao objeto do ensaio. A semelhança entre o meio e o objeto é definida através das variáveis, iterationForma, deslocamento e cIntensity, internas a função. Portanto um hardconding é necessário para altera-las.

O meio neutro é ativado quando cinicial é igual a 1 e a matriz será usado caso cinicial seja diferente de 0 ou 1.

A função controladorTomografiaInterpolacao, além das operações executas pela função controladorTomografiaSimples, ainda define quais são os sensores do ensaio que serão efetivamente usados e interpola os dados desses sensores.

Os sensores efetivamente usados representam os sensores que existiriam caso a simulação fosse real. Pois na realidade não dispomos de sensores contínuos no torso. Porém o algoritmo exige que existam sensores em todo o entorno do torso. Logo, faz-se uma interpolação nos dados coletados nos sensores efetivamente usado de modo que os dados sejam espalhados para todos os sensores do torso. Para tal, usa-se a função **interpolateSensorData.**

O controle do problema inverso é feito por meio de um laço de repetição *for* que chama a função **tomografyIteration** e com a saída dessa função calcula o aprimoramento do meio.

A função **tomografyIteration** é responsável por executar a propagação da onda , a retropagação do resíduo e o cálculo de . Sendo que este valor calculado é a saída da função. Usando essa saída, a função controladorTomografia, calcula o aprimoramento do meio através dos coeficientes e .

Destaca-se que a função **tomografyIteration** tem como entrada os dados reais de todos os sensores na borda do torso, logo a interpolação deve ocorrer antes da chamada da função. Assim, no caso interpolante o laço for executa três tarefas(enquanto no caso simples, apenas as duas últimas são realizadas):

* Interpola os dados dos sensores efetivamente, espalhando a informação para todos os sensores na borda do torso (função interpolateSensorData)
* Calcula a integral de (função tomografyIteration)
* Calcula o aprimoramento

### Códigos de resposta

Os códigos que geram respostas são bem simples e consistem de uma função e um script. A função é o calculam-se que calcula o MSE e o NMSE de cada uma das regiões (pulmões, coração, coluna e meio) partindo do ensaio e da resposta do algoritmo.

O script apenas gera os gráficos do MSE, do NMSE da correlação e do residual do algoritmo para cada iteração.

Abaixo segue uma figura que indica todos os códigos e as dependência que eles têm entre si. As caixas com traçado continuo representam os códigos de maior importância, enquanto os tracejados são os códigos auxiliares.

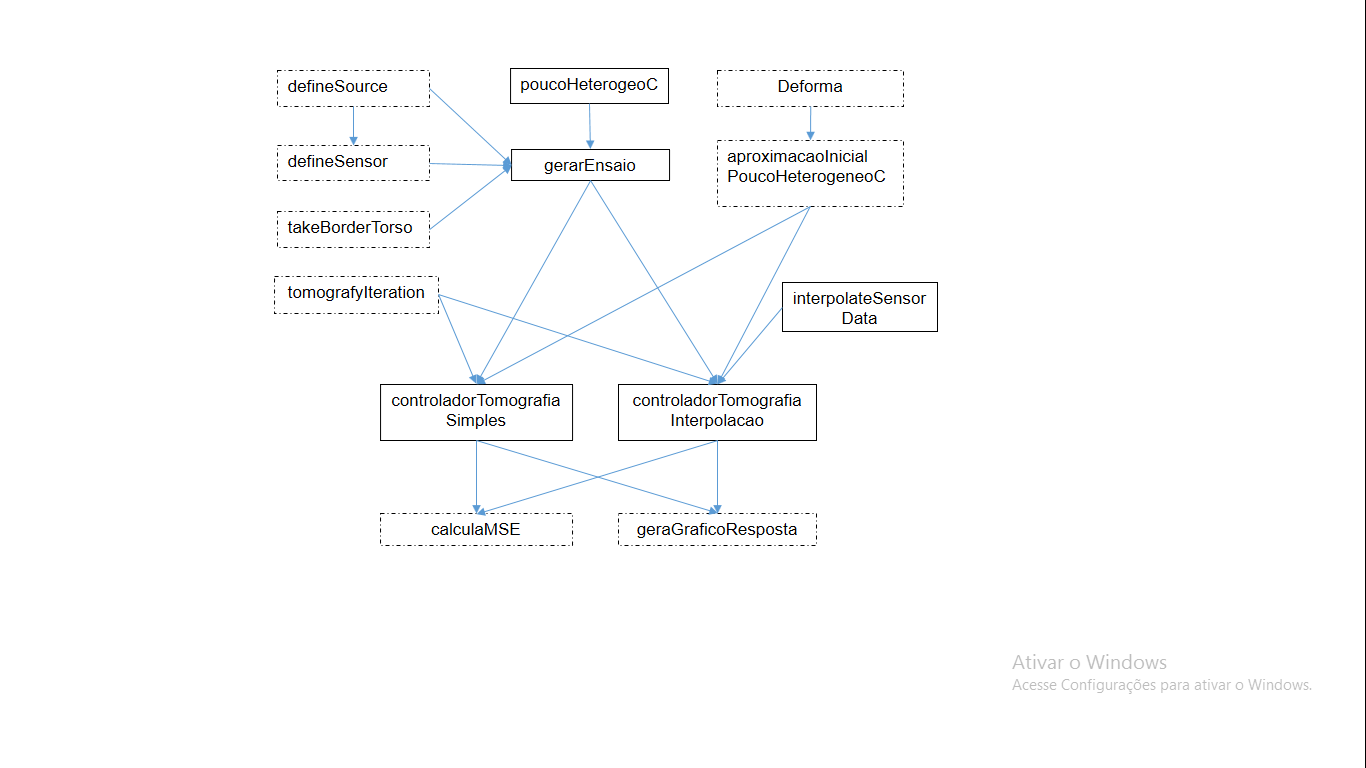


Figura . Diagrama de blocos da estrutura do código do algoritmo de reconstrução tomográfico

# BIBLIOGRAFIA

[#1] A. Hormati, I. Jovanovíc, O. Roy, M. VetterliRobust Ultrasound Travel-time Tomography Using the Bent Ray Model. Proc. of SPIE Vol. 7629 76290I-1. (2010)

[#2] P. Lasaygues, R. Guillermin, J. Lefebvre. Ultrasonic Computed Tomography. Bone Quantitative Ultrasound, Springer, pp.441-459, 2010.

[#3] Malcolm Slaney and A.C.Kak. Diffraction Tomography (1989)

[#4] C. Dong, Yuanwei Jin MIMO Nonlinear Ultrasonic Tomography by Propagation and Backpropagation Method. ieee transactions on image processing, vol. 22, no. 3, march 2013

[#5] K-wave user manual

[#6] B. E. Treeby and B. T. Cox, “Modeling power law absorption and dispersion for acoustic propagation using the fractional Laplacian,” J. Acoust. Soc. Am., vol. 127, no. 5, pp. 2741– 2748, 2010.

[#7] F. Simonetti, “Multiple scattering: The key to unravel the subwavelength world from the far-field pattern of a scattered wave,” Phys. Rev. E, vol. 73, no. 3, p. 036619, Mar 2006.

[#8] J Jaros, B. Treeby and A. Rendell. Use of Multiple GPUs on Shared Memory Multiprocessors for Ultrasound Propagation Simulations, Conferences in Research and Practice  
in Information Technology (CRPIT), Vol. 127 (2012)

[#9] Livro de analise numerica