MAP 3122 - Métodos Numéricos e Aplicações
Exercício Prático II

Bruno Borges Paschoalinoto (10687472)

Escola Politécnica da Universidade de São Paulo

Implementar a solução da equação de onda nas formas direta e inversa.

I. Introdução

Neste EP foi estudada a oposição entre problemas diretos e inversos. O objeto de estudo é a equação de onda em uma dimensão. Claro, foram adotadas simplificações, como condições de contorno ideais, fontes pontuais, e a homogeneidade do meio. O meio é discretizado em $n_x + 1$ pontos espaçados de Δx, e o tempo é discretizado em $n_t + 1$ intervalos de duração Δt.

O problema direto pode ser expresso da seguinte maneira: dadas as posições e intensidades de fontes no meio, computar a *pressão acústica* $u(x, t)$ da onda para cada ponto e instante discretizados, ou seja, a evolução do sistema.

O problema inverso usa a solução do direto para obter um resultado diferente. Dada a prssão acústica da onda em um só ponto x_r (chama-se esse dado de *sismograma*) em todos os instantes considerados, e sabendo que há K fontes nas posições $x_k\,(k = 1, 2, \ldots, K)$, calcular suas intensidades $a_k\,(k = 1, 2, \ldots, K)$.

Soluções para ambos os problemas foram implementadas em C, com alguns programas auxiliares em Python 3 para plotar os dados de saída usando `matplotlib`.

II. Estrutura do programa

O EP foi implementado largamente em C (padrão C89) (usando apenas as bibliotecas-padrão) e Python 3 (usando apenas as bibliotecas-padrão e `matplotlib`). O código está distribuído em vários arquivos de código na pasta `src`, e cada arquivo é de um dos seguintes tipos:

- Arquivos de cabeçalho C (declaram funções dos módulos)
- Arquivos de código C sem ponto de entrada (implementam as funções dos cabeçalhos)
- Arquivos de código C com ponto de entrada (implementam um dos exercícios)
- Arquivos de código Python 3 (não dependem dos demais, plotam saídas dos programas em C)

Como há uso extensivo de matrizes e métodos para resolver sistemas lineares nesse EP, optou-se por implementar essas funções compartilhadas em vários módulos em C. Sâo eles:

- `matrizes.h`: funções para criar e manipular matrizes (vetores bidimensionais) de `double`s.
- `sistemas.h`: funções para resolver sistemas linares (fatoração de Cholesky e SOR).
- `direto.h`: solução do problema direto (parâmetros → evolução do sistema).
- `inverso.h`: elaboração do sistema linear para solução do problema inverso.
- `ruido.h`: inserção de ruído num sismograma.
- `utils.h`: subrotinas matemáticas gerais usadas pelos demais módulos.

Todos esses `.h` têm um `.c` correspondente contendo as implementações das funções definidas (e outras internas, para legibilidade, mas irrelevantes para o módulo).

Esses módulos não contêm interação, apenas subrotinas a serem incluídas num programa completo, com ponto de entrada. Esses pontos de entrada ficam na pasta `src/mains`. Sendo assim, para compilar um programa específico do EP, basta compilar os módulos com uma das "mains". Isso torna o EP muito mais modular e fácil de analisar, pois há menos repetição de código, e este fica mais categorizado. As "mains" disponíveis são:

- `ex1.c`: recebe os parâmetros do problema direto e produz a evolução do sistema, escrevendo-a num arquivo.
- `ex2.c`: recebe os parâmetros do problema inverso, um arquivo de sismograma, e calcula as intensidades das fontes.

- `ex3.c`: recebe os parâmetros do problema inverso, um arquivo de sismograma, um valor para η, e calcula as soluções do problema inverso com e sem ruído, comparando-as.
- `ruidificador.c`: recebe um sismograma, alguns parâmetros de tempo, um valor para η e produz um outro sismograma, e informa na tela o nível de ruído resultante. Usado para produzir as comparações de sismogramas.

Além disso, nessa pasta também há dois programas em Python 3 capazes de plotar saídas dos programas em C, usados para produzir as imagens neste relatório. São eles:

- `plotar_ex1.py`: plota uma versão animada da solução do problema direto na tela, recebendo como parâmetro um arquivo de saída do programa `ex1`.
- `plotar_ruido.py`: recebe um arquivo de sismograma e um valor para η, e plota na tela uma comparação entre o sismograma com e sem ruído, além do nível de ruído resultante.

Como o processo de compilação pode ser tedioso, foi fornecido um `Makefile` com comandos prontos de compilação e de invocação. Todos os comandos prontos disponíveis podem ser vistos com o comando `make help` no terminal, a partir da pasta raiz do EP.

Na ausência do *GNU Make*, os comandos completos para compilação e alguns exemplos de invocações válidas para cada um dos três programas podem ser encontrados em `sem_make.txt`. Além disso, todos os programas informam suas opções de invocação ao serem chamados sem parâmetros.

III. Exercício 1: o problema direto

A. Formalização e modelo adotado

Antes de formalizar o problema direto, é preciso estabelecer alguns termos gerais. O meio $[0, X]$ é discretizado em $n_x + 1$ pontos, onde $x_j = j * \Delta x$, sendo $\Delta x = X/n_x$ e $j = 0, 1, 2, \ldots, n_x$. O mesmo acontece para o tempo $[0, T]$, discretizado em $n_t + 1$ instantes, onde $t_i = i * \Delta t$, sendo $\Delta t = T/n_t$ e $i = 0, 1, 2, \ldots, n_t$.

A fonte, no problema original, é uma função $f(x, t)$, nula para todo $x \neq x_c$. Contudo, como representações de ponto-flutuante não costumam ser exatas, é bem possível que não haja um $x_j = x_c$ para a maioria das escolhas de n_x. Sendo assim, optou-se por definir j_c como o inteiro mais próximo de $\frac{x_c}{\Delta x}$, e f_i^j assumindo um valor nulo para todo $j \neq j_c$.

B. Implementação

A solução contínua $u(x, t)$ para a equação de onda pode ser aproximada por u_i^j, de acordo com uma relação de recorrência:

$$u_i^0 = u_i^{n_x} = 0 \tag{1}$$

$$u_0^j = u_1^j = 0 \tag{2}$$

$$u_i^j = -u_{i-2}^j + 2\left(1 - \alpha^2\right) u_{i-1}^j + \alpha^2 \left(u_{i-1}^{j+1} + u_{i-1}^{j-1}\right) + \Delta t^2 f_{i-1}^j \tag{3}$$

A condição inicial (1) é a condição de extremidades fixas, e a (2) indica que tanto a solução quanto sua primeira derivada são nulas no instante inicial. A relação de recorrência (3) é usada para calcular os valores nos instantes subsequentes ($i > 1$).

Em `direto.c`, é implementada a relação de recorrência e uma função que inicializa uma matriz e a preenche com os valores u_j^i. Todo valor calculado, mesmo que em passos intermediários, é preservado na matriz; e o ponto de entrada das chamadas subsequentes é sempre próximo dos já calculados, mantendo o tamanho do *stack* limitado a duas ou três chamadas, e a implementação, eficiente.

C. Resultados e observações

Ao invés de observar imagens paradas, foi feito um programa em Python 3 capaz de animar na tela toda a evolução do sistema, lendo o arquivo produzido pelo programa C `ex1`. Pode ser facilmente observado um comportamento semelhante ao de uma corda (sem gravidade nem atrito, claro) oscilando com as pontas fixas. Contudo, nem todas as combinações de n_x e n_t produzem um resultado satisfatório, e o enunciado propôs que fossem explorados os valores limítrofes para a estabilidade.

Tal investigação seguiu o procedimento indicado: para valores fixos dos demais parâmetros do sistema, foi-se diminuindo n_t até a solução degenerar (no caso, assumir valores muito altos, nulos, detecção de infinidades ou `NaN`s). Foi observado um valor limítrofe a partir do qual a solução degenera, a depender apenas de c e n_x, sendo este mantido sempre em 100.

Para $c^2 = 10$, o menor valor de n_t que produz uma solução estável é 317. Para $c^2 = 20$, é 448. A condição de estabilidade CFL dá a seguinte relação entre os parâmetros para garantir estabilidade:

$$c\frac{\Delta t}{\Delta x} \leq C_{max} \tag{4}$$

A partir dela, pode-se isolar o n_t crítico (menor valor antes da instabilidade):

$$c\frac{\frac{T}{n_t}}{\frac{X}{n_x}} \leq C_{max} \tag{5}$$

$$c\frac{Tn_x}{Xn_t} \leq C_{max} \tag{6}$$

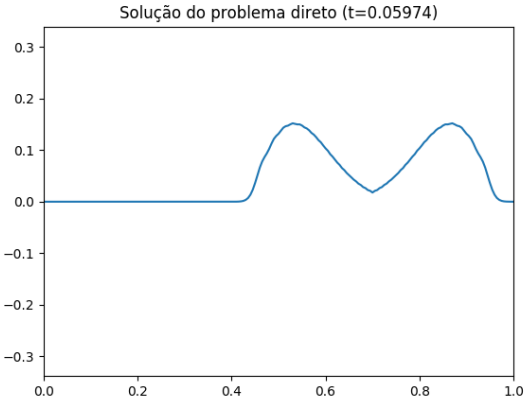
$$c\frac{Tn_x}{XC_{max}} \leq n_t \tag{7}$$

O menor valor que satisfaz a inequação acima é:

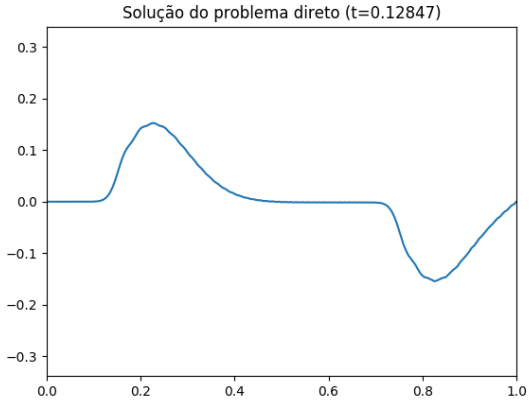
$$n_{t,crit} = \left\lceil c\frac{Tn_x}{XC_{max}} \right\rceil \tag{8}$$

Os valores de $n_{t,crit}$ observados condizem com um $C_{max} = 1$, um valor esperado para métodos explícitos de maneira geral.

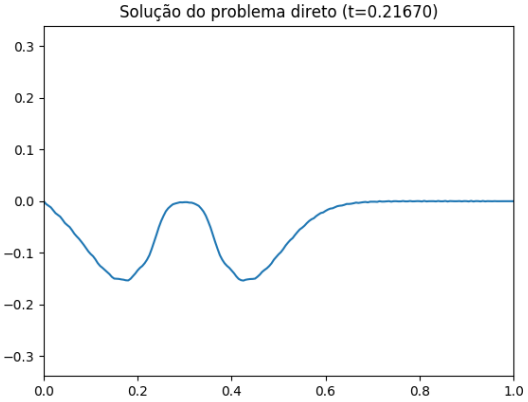
Seguem algumas imagens da solução com $c^2 = 20$, $\beta = 10$, $n_x = 200$, e $n_t = 4000$.



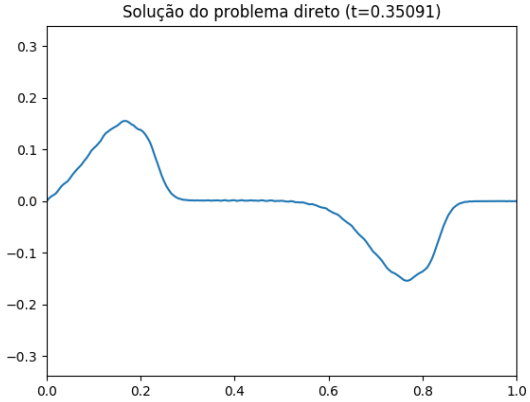
(a)



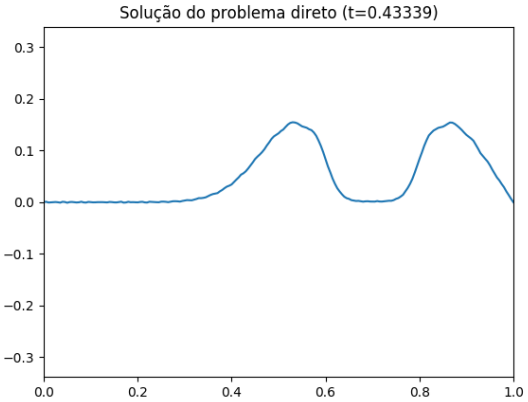
(b)



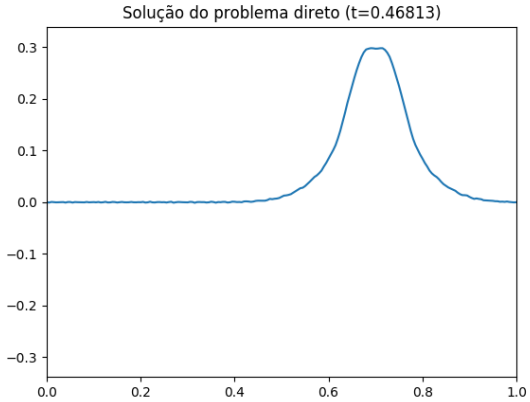
(c)



(d)



(e)



(f)

É possível ver a animação acima com o comando `make ex1-anim-20`. Também há um comando semelhante para a situação $c^2 = 10$: `make ex1-anim-10`.

IV. Exercício 2: o problema inverso

A. Formalização e modelo adotado

O exercício 2 consiste na resolução do problema inverso proposto. Consideramos que há K fontes, cada qual numa posição conhecida x_k com intensidade desconhecida a_k^*. Pelo princípio da superposição, a evolução resultante do sistema pode ser considerada uma combinação linear das soluções para cada uma das fontes isoladamente, tomando como coeficientes as intensidades a_k^*.

Formalmente, o problema consiste em, munidos dos parâmetros do sistema (c, β, n_x), as posições x_k das K fontes e um sismograma a partir de um ponto conhecido x_r, estimar as intensidades a_k^* de cada fonte.

Semelhantemente ao x_c no caso do exercício 1, foi necessário adequar x_r e cada x_k à discretização do espaço, calculando-se j_r e j_k de cada fonte. Além disso, é considerado um período de tempo reduzido $[t_i, t_f]$, e não o sismograma inteiro sempre.

O problema então torna-se um de minimização de erro, sendo este a diferença entre o sismograma dado e um que pode ser elaborado a partir das intensidades encontradas. Para tal, empregou-se o Método dos Mínimos Quadrados (MMQ).

B. Montagem do sistema linear

O problema inverso então assume a seguinte forma: munido do sismograma d_r em x_r, as posições x_k das fontes, os parâmetros do sistema, e um período de tempo $[t_i, t_f]$ a considerar, elaborar o sistema linear $Ba = c$, resolvê-lo usando um dos métodos propostos, e ter como vetor das intensidades aproximadas a_k a solução.

A montagem do sistema linear é precedida pela elaboração de K sismogramas "isolados" d_k, cada qual representando o comportamento do ponto x_k (ou melhor, seu vizinho discreto mais próximo) sob a influência exclusiva da k-ésima fonte. Todos esses sismogramas são armazenados como vetores-coluna, assim como d_r.

O objetivo é preencher a matriz quadrada de lado K, B e o vetor-coluna de dimensão K, c, representando o sistema linear $Ba = c$. Ambos giram em torno do mesmo conceito geral, a integral de um produto. Considerando a discretização do tempo, foi usada a aproximação do trapézio para integrais, e os limites de tempo t_i e t_f serão doravante representados por seus índices discretos i_{min} e i_{max}.

$$\int_{t_i}^{t_f} h(t)dt = \left(\frac{h_{i_{min}} + h_{i_{max}}}{2} + \sum_{i=i_{min}+1}^{i_{max}-1} h_i \right) \Delta t = \mathrm{I}\left(i_{min}, i_{max}, h, \Delta t\right) \tag{9}$$

O preenchimento da matriz B se fez da seguinte maneira:

$$B_{pq} = \int_{t_i}^{t_f} d_p(t) d_q(t) dt = \mathrm{I}\left(i_{min}, i_{max}, d_p \odot d_q, \Delta t\right) \tag{10}$$

Note-se que $A \odot B$ representa o produto de Hadamard (entrada-a-entrada) entre duas matrizes (no caso, vetores-coluna). Como a matriz B é simétrica, uma otimização simples é só calcular os valores para $p \leq q$ e "espelhar" os valores calculados quando $p \neq q$. Já o vetor de valores independentes c é construído de maneira similar, onde o sismograma dado d_r substitui um dos "isolados":

$$c_p = \int_{t_i}^{t_f} d_p(t) d_r(t) dt = \mathrm{I}\left(i_{min}, i_{max}, d_p \odot d_r, \Delta t\right) \tag{11}$$

O sistema linear do MMQ montado como acima, ao ser resolvido, produzirá o vetor a com a aproximação encontrada para as intensidades das fontes. Resta, então, a escolha de um método numérico para resolver o sistema montado e sua implementação, discutidos na subseção a seguir.

C. Resolução do sistema linear

O enunciado propôs que fossem implementados dois métodos para resolver o sistema linear do MMQ: fatoração de Cholesky e SOR (*successive over-relaxation*). Em seguida, foi pedido que se comparassse a precisão dos dois métodos. Nesta subseção, será discutida apenas a implementação, sendo a comparação parte da discussão na próxima subseção.

Primeiro, a fatoração de Cholesky. Ela não é um método de resolução de sistema linear por si só, ela consiste em decompor uma matriz hermitiana (ou simplesmente simétrica, como só se está trabalhando com números reais) e positiva-definida no produto de uma triangular inferior e sua transjugada (ou simplesmente transposta, pelo mesmo motivo). Isso permite que um sistema linear simétrico $Ba = c$ seja tornado equivalente a resolver dois sistemas triangulares, cujas resoluções por *back-substitution* são triviais e rápidas.

O procedimento formal é o seguinte:

1. Começa-se com o sistema $Ba = c$. Após a fatoração de Cholesky, ele pode ser escrito como $LL^\intercal a = c$.
2. Seja $y = L^\intercal a$. O sistema triangular inferior $Ly = c$ pode ser resolvido rapidamente, determinando-se y.
3. Conhecendo-se y, pode-se resolver o sistema triangular superior $L^\intercal a = y$, determinando-se a.

Já o método iterativo SOR é mais simples de explicar. É um método geral, cuja taxa de convergência é condicionada numa boa escolha do *parâmetro de relaxamento* ω. Infelizmente, é difícil saber o valor ótimo de ω de antemão, então costuma-se usar estimativas. Contudo, é sabido que o método não converge para $\omega \notin (0, 2)$.

Quanto ao procedimento implementado, parte-se da "solução" inicial $a^0 = \vec{0}$, e usa-se a seguinte fórmula para computar a próxima iteração:

$$a_k^{(i+1)} = (1-\omega)a_k^{(i)} + \frac{\omega}{B_{kk}}\left(c_k - \sum_{j<k} B_{kj}a_j^{(i+1)} - \sum_{j>k} B_{kj}a_j^{(i)}\right), \quad k = 1, 2, \ldots, K. \tag{12}$$

A explicação dessa fórmula pode ser encontrada nas referências. O enunciado também impôs que devem ser feitas 10^4 iterações e que deve ser usado um ω igual a 1.6.

D. Resultados e observações

Para o problema inverso, duas métricas de erro são definidas: o resíduo e o erro L2. O resíduo representa o "erro da reconstrução", ele mede o quanto o cenário descrito pelas intensidades encontradas diverge do cenário real. Por outro lado, o erro L2 é uma mensura direta do quanto as intensidades calculadas diferem das reais; ele é meramente a norma do vetor $a_k - a_k^*$.

$$\text{resíduo} = \frac{1}{2\,(t_f - t_i)}\int_{t_i}^{t_f} \left(d(t) - d_r(t)\right)^2 dt, \quad e \quad \text{erro}_{L2} = \sqrt{\sum_{k=1}^{K}\left(a_k - a_k^*\right)^2} \tag{13}$$

Note-se que $d(t)$ é o sismograma construído a partir das intensidades calculadas a_k, e a_k^* são as intensidades "reais" (quando conhecias, claro). Como as intensidades reais para os sismogramas `dr3.txt` e `dr10.txt` foram fornecidas, foi possível calcular ambas as métricas de erro para eles, apresentadas na tabela abaixo:

Sismograma	Método	erro_{L2}	resíduo
`dr3.txt`	Cholesky	$3.78255 \cdot 10^{-12}$	$1.65288 \cdot 10^{-23}$
`dr3.txt`	SOR	$3.78268 \cdot 10^{-12}$	$1.65286 \cdot 10^{-23}$
`dr10.txt`	Cholesky	$4.87011 \cdot 10^{-11}$	$1.73706 \cdot 10^{-24}$
`dr10.txt`	SOR	$4.9382 \cdot 10^{-11}$	$1.73703 \cdot 10^{-24}$

Aparentemente, tanto a fatoração de Cholesky quanto o SOR produzem resultados precisos, com diferenças marginais nas métricas de erro. Talvez as diferenças entre os métodos, tanto em termos de performance quanto de precisão, pudessem se tornar mais perceptíveis numa situação em que o tamanho dos sistemas envolvidos fosse maior. No caso, só se trabalhou com $K \in \{3, 10, 20\}$.

Para os dois primeiros sismogramas, a precisão dos resultados foi tamanha que, ao se imprimir os valores do vetor a com o formatador `%g` do C, o resultado aparentava ser exato, sendo necessário imprimir com precisão completa (`%.17f`) ou calcular outras métricas de erro, como desvios percentuais com relação ao a_k^*.

Já para o sismograma `dr20.txt`, não foram dadas as intensidades reais das fontes, sendo assim, a métrica erro_{L2} não pode ser calculada, e a corretude dos resultados só pode ser indicada por um valor baixo do resíduo. Os valores encontrados são os seguintes:

Ordem original	Ordem crescente
6.5478	-17.188
-17.188	-11.701
21.044	-11.508
12.966	-6.1658
14.626	-4.9931
-3.8704	-3.8704
-2.3858	-2.3858
-6.1658	0.46808
37.358	6.5478
-11.701	9.5876
65.233	11.496
-11.508	11.094
22.582	12.966
-4.9931	12.28
9.5876	14.626
20.311	21.044
0.46808	20.311
11.496	22.582
12.28	37.358
11.094	65.233

No cálculo que trouxe os valores acima, o resíduo foi de aproximadamente $6.13512 \cdot 10^{-4}$.

V. Exercício 3: ruído e seus efeitos

A. Inserção do ruído e determinação do seu nível

O terceiro e último exercício pede que o sismograma seja alterado via uma função que distorce cada entrada numa parte de um sismograma (novamente, não será considerado o sismograma inteiro). Além disso, só será usado o sismograma `dr10.txt`, alterado de acordo com a fórmula que se segue:

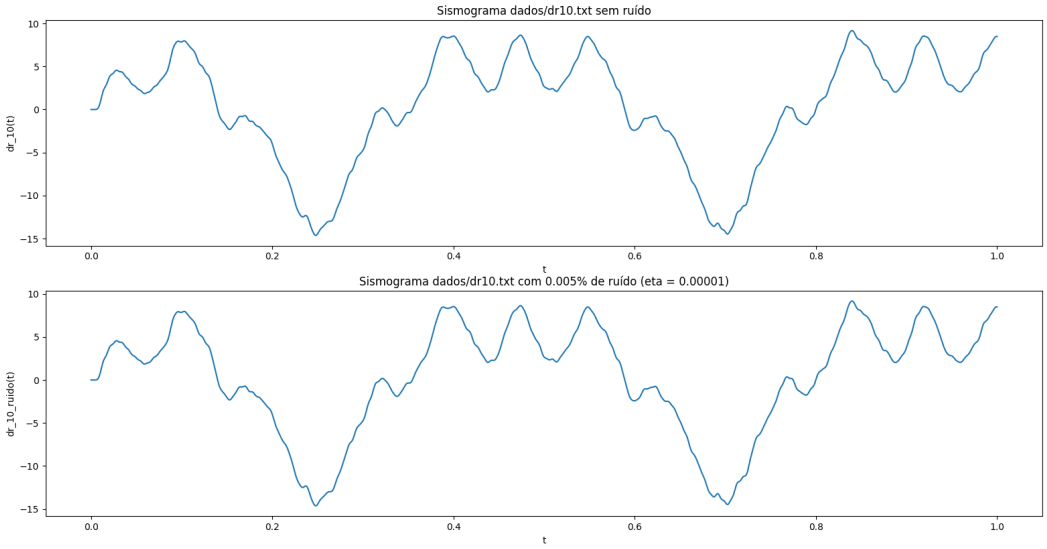
$$d_r^{ruido}(t) = (1 + \eta \cdot M \cdot v(t))\, d_r^{10}(t), \quad \text{onde} \quad M = \max_{s \in [t_i, t_f]} d_r^{10}(s) \quad \text{e} \quad v(t) \quad \text{é um gerador aleatório em } [-1, 1] \tag{14}$$

Após a inserção do ruído numa parte do sismograma, uma métrica pode ser calculada sobre o sismograma resultante para quantificar a diferença entre ele e o original:

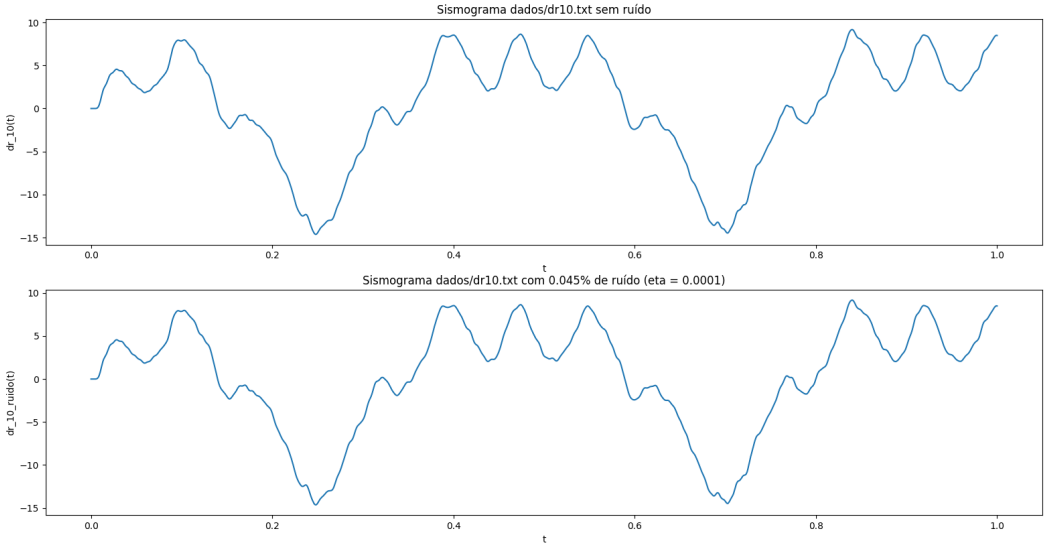
$$\text{ruido} = 100 \frac{\int_{t_i}^{t_f} \left| d_r^{10}(t) - d_r^{ruido}(t) \right| dt}{\int_{t_i}^{t_f} \left| d_r^{10}(t) \right| dt} \tag{15}$$

Como sempre, todas as integrais neste relatório estão sendo calculadas no programa pela fórmula dos trapézios descrita em (9), adequando-se à discretização do tempo. Assim sendo, a importância dessa métrica se deve ao fato de que níveis "razoáveis" de ruído (não impeditivos à boa reconstrução com resíduo baixo) são visualmente imperceptíveis.

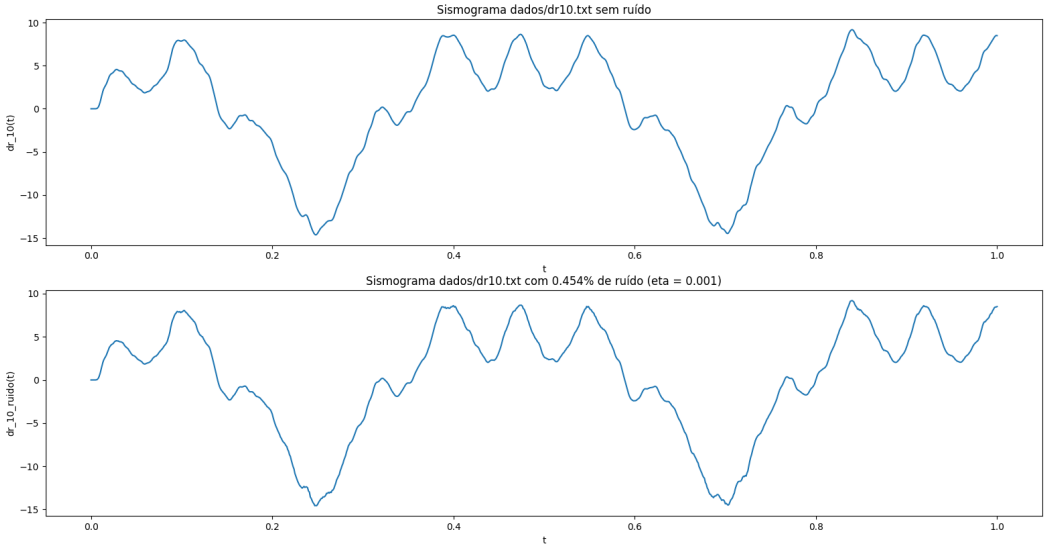
A efeito de exemplo, basta ver as imagens nas páginas seguintes: só é possível perceber "de longe" uma diferença visual no sismograma nas três últimas imagens, com valores maiores de η. Entretanto, como vai ser discutido a seguir, os valores menores e visualmente imperceptíveis da primeira página já provocaram grandes degradações na reconstrução, com valores altos de resíduo e desvios percentuais significativos com respeito a a_k^*.


Sismograma dados/dr10.txt sem ruído
dr_10(t)
t
Sismograma dados/dr10.txt com 0.005% de ruído (eta = 0.00001)
dr_10_ruido(t)
t

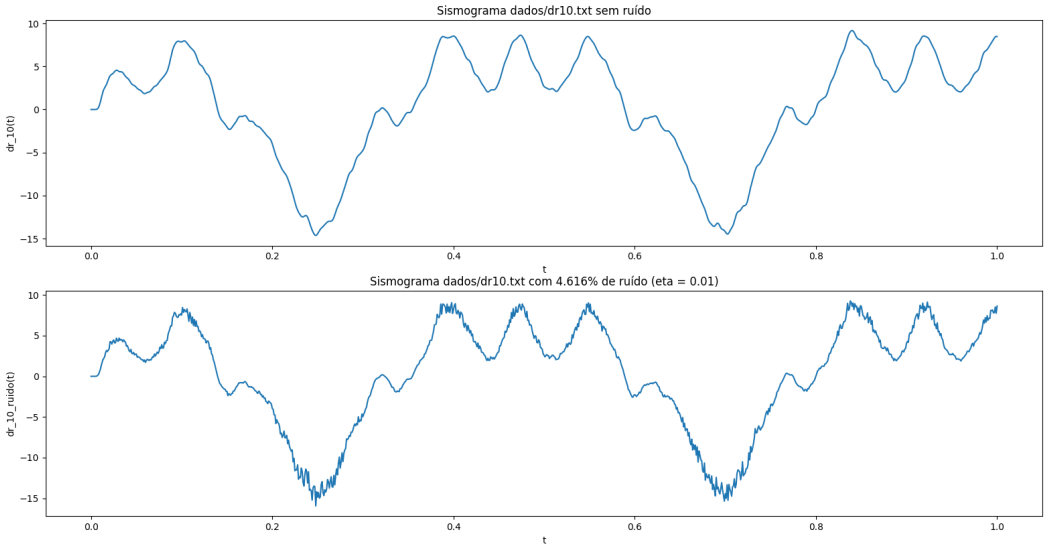
(a)


Sismograma dados/dr10.txt sem ruído
dr_10(t)
t
Sismograma dados/dr10.txt com 0.045% de ruído (eta = 0.0001)
dr_10_ruido(t)
t

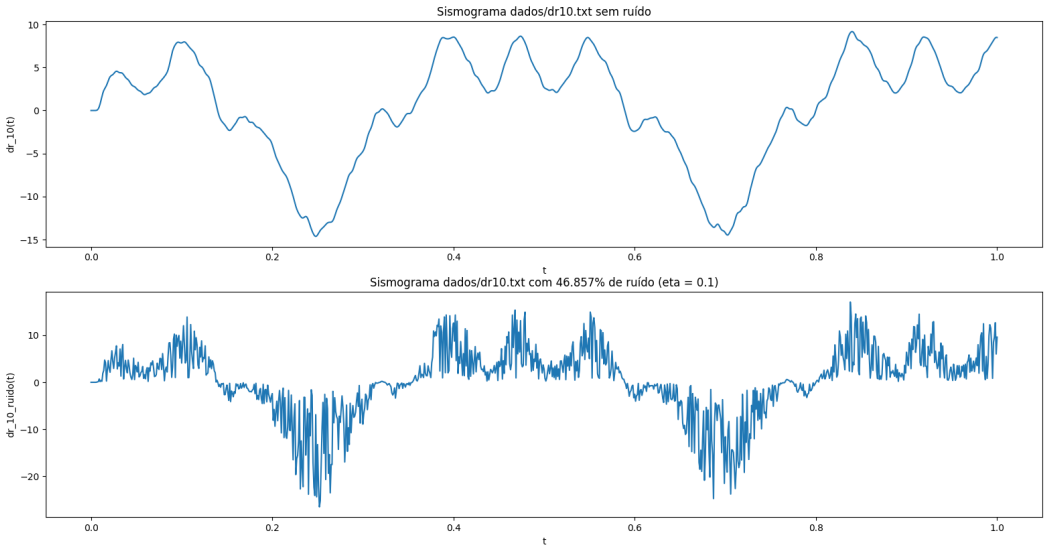
(b)


Sismograma dados/dr10.txt sem ruído
dr_10(t)
t
Sismograma dados/dr10.txt com 0.454% de ruído (eta = 0.001)
dr_10_ruido(t)
t

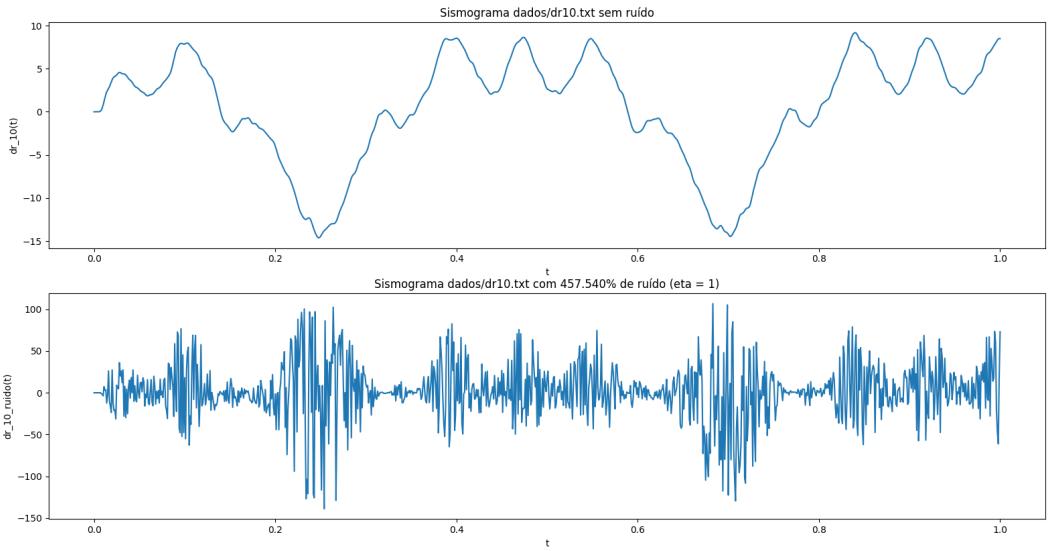
(c)


Sismograma dados/dr10.txt sem ruído
dr_10(t)
t
Sismograma dados/dr10.txt com 4.616% de ruído (eta = 0.01)
dr_10_ruido(t)
t

(a)


Sismograma dados/dr10.txt sem ruído
dr_10(t)
t
Sismograma dados/dr10.txt com 46.857% de ruído (eta = 0.1)
dr_10_ruido(t)
t

(b)


Sismograma dados/dr10.txt sem ruído
dr_10(t)
t
Sismograma dados/dr10.txt com 457.540% de ruído (eta = 1)
dr_10_ruido(t)
t

(c)

B. Efeitos do ruído sobre a reconstrução

O enunciado afirma que, mesmo para valores pequenos de η, a reconstrução já será bastante degradada, e isso se confirma nas métricas de erro. A tabela abaixo apresenta uma comparação entre elas para alguns valores de η sobre o sismograma `dr10.txt`.

Ruído (η)	Nível de ruído	erro_{L2}	resíduo
0	0%	$4.87011 \cdot 10^{-11}$	$1.73706 \cdot 10^{-24}$
10^{-5}	0.00393%	0.0326536	$7.77749 \cdot 10^{-09}$
10^{-4}	0.0437%	0.283331	$9.83404 \cdot 10^{-07}$
10^{-3}	0.426%	3.39261	$4.89546 \cdot 10^{-05}$

Há uma relação de crescimento aparente entre as métricas de erro e η: uma década do η aumenta ambas as métricas de erro em aproximadamente uma década também. Todavia, essa relação provavelmente não é sustentável, devido ao fator aleatório envolvido. Para valores maiores de η, o fator aleatório vai predominar, descartando praticamente toda a informação do sismograma, como na última imagem da página anterior, então os resultados não vão conter quase nenhuma informação relevante.

Não são apenas as métricas de erro que indicam o efeito do ruído. Os próprios valores retornados já aparentam forte degradação. A tabela abaixo foi construída usando os mesmos parâmetros da anterior, só que o enfoque agora são os valores retornados.

$\eta = 0$	$\eta = 10^{-5}$	$\eta = 10^{-4}$	$\eta = 10^{-3}$
7.3	7.33334	7.27494	7.72967
2.4	2.39558	2.57713	1.16364
5.7	5.70217	5.80661	5.79692
4.7	4.70515	4.65134	5.24453
0.1	0.101249	0.125669	0.65959
20	19.9975	19.9804	19.2839
5.1	5.10082	5.10133	5.23164
6.1	6.09967	6.09461	6.09663
2.8	2.8001	2.80175	2.7963
15.3	15.2998	15.2987	15.3088

Essa tabela mostra um efeito do ruído que as duas métricas de erro não revelariam sozinhas. As fontes de intensidades menores tendem a ser mais afetadas pelo ruído: com $\eta = 10^{-3}$, as de intensidade acima de 4 sofreram quase todas desvios percentuais de não mais do que 1.5%, enquanto a de 0.1 variou em quase 800%.

Obviamente, não é uma relação fechada, perfeita: ainda comparando a primeira e quarta colunas, a fonte de intensidade 15.3 variou -0.03% (a menor variação percentual em módulo), mas a de 20 variou -3.52% — a 4ª maior variação percentual em módulo. A aleatoriedade do ruído ainda predomina, e nem foi levada em conta, ou mesmo discutida a influência das posições das fontes, dos demais parâmetros como c e β, etc.

De qualquer maneira, o padrão é visível e reprodutível: várias execuções com os mesmos parâmetros (variando só a semente do gerador pseudoaleatório, a qual é derivada do relógio interno do computador no instante da execução) consistentemente mostram as fontes de menor intensidade sofrendo os maiores desvios percentuais.

Referências

[1] Guilherme Caminha, "The CFL Condition and How to Choose Your Timestep Size." https://www.simscale.com/blog/2017/08/cfl-condition/, 2019. [Online; acessado em 11 de abril de 2020].

[2] Wikipedia, "Cholesky decomposition — Wikipedia, the free encyclopedia." http://en.wikipedia.org/w/index.php?title=Cholesky\%20decomposition&oldid=950029494, 2020. [Online; acessado em 11 de abril de 2020].

[3] Wikipedia, "Successive over-relaxation — Wikipedia, the free encyclopedia." http://en.wikipedia.org/w/index.php?title=Successive\%20over-relaxation&oldid=929345712, 2020. [Online; acessado em 11 de abril de 2020].

[4] N. Black e S. Moore, "Successive Overrelaxation Method — Wolfram MathWorld." https://mathworld.wolfram.com/SuccessiveOverrelaxationMethod.html, 2004. [Online; acessado em 11 de abril de 2020].