UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E COMPUTAÇÃO CIENTÍFICA
DEPARTAMENTO DE MATEMÁTICA APLICADA

Beatriz Belucci

Modelamento sísmico por diferenças finitas: aplicação do pacote Devito para resolução da equação da onda[1]

Monografia apresentada ao Instituto de Matemática, Estatística e Computação Científica da Universidade Estadual de Campinas como parte dos requisitos para obtenção de créditos na disciplina Projeto Supervisionado, sob a orientação do(a) Prof. Dr. Ricardo Biloti.

Campinas
2022

[1]Este trabalho foi financiado pela DOW, quota 2021/2022.

1 Introdução

Este projeto tem por objetivo o estudo, e a utilização, do *framework* robusto para modelamento sísmico por diferenças finitas chamado Devito. Os estudos foram divididos em três etapas: estudo teórico da onda acústica em meio homogêneo e sua representação matemática, estudo teórico e aplicações do método de diferenças finitas e, por fim, o estudo e utilização do *framework* Devito.

O foco de estudo foram as ondas acústicas unidimensionais. Para o estudo inicial utilizamos Knobel (2000), que apresenta uma introdução às ondas progressivas e estacionárias, bem como exemplos de eventos que identificam o fenômeno da onda. Além disso, é apresentado a representação matemática da onda e suas soluções analíticas. Para enriquecer o estudo, foram utilizados os textos Biloti (2020) e Biloti (2022). Para o método de diferenças finitas utilizamos Wencai (2013), Porto (2020), Santos et al. (2019) e Biloti (2021). Por fim, com o propósito de desenvolver os conhecimentos sobre o Devito foi utilizado, principalmente, o texto de Louboutin et al. (2019), bem como o material apresentado em London et al. (2016). As rotinas que utilizam o Devito, implementadas durante o projeto, baseiam-se nos códigos contidos no GitHub[1] oficial do *framework*. O presente projeto foi executado com uma bolsa financiada pela DOW, quota 2021/2022.

2 Ondas acústicas em meio homogêneo

Ondas acústicas em meio homogêneo podem ser entendidas como uma pertubação que se desloca de uma parte à outra do meio com uma velocidade de propagação constante e reconhecível. As ondas que se propagam em apenas uma direção do espaço são chamadas ondas unidimensionais e são matematicamente representadas por funções de duas variáveis $u(x,t)$. Essas funções dependem de uma variável espacial x, que indica a coordenada espacial, e de uma variável temporal t, que representa o tempo. A propagação de ondas escalares unidimensionais é descrita pela *equação da onda*

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}, \tag{1}$$

onde, c representa a velocidade de propagação da onda, que pode ser constante ou variar com o espaço.

A equação da onda é uma equação diferencial parcial de segunda ordem, ou seja, é uma equação que depende das derivadas parciais da função que descreve uma onda com relação ao espaço e ao tempo. Como solução analítica para a equação da onda em meios homogêneos podemos ter, por exemplo, funções que representam ondas progressivas ou ondas periódicas. Focando, a princípio, nas ondas unidimensionais, uma onda progressiva é uma onda que se

[1] O repositório pode ser acessado em: https://github.com/devitocodes/devito.

propaga na direção positiva ou negativa do eixo x. Esse tipo de onda é representada por uma função f de uma variável, que é avaliada em $x \pm ct$. Ou seja, uma $f(x \pm ct)$ que é transladada em ct unidades, onde c é uma constante não nula que representa a velocidade de propagação da onda e t é o instante de tempo. Por ser transladada, o perfil da onda progressiva é preservado durante o movimento. Já as ondas periódicas são ondas progressivas que podem ser descritas como funções periódicas

$$u(x,t) = A\cos\left[k\left(x - \frac{\omega}{k}t\right)\right],$$

isto é, funções que repetem seus valores após um certo período de tempo. Note que, funções que são a soma finita de funções periódicas também são funções periódicas, logo, também são soluções da equação da onda. Entretanto, cabe ressaltar que ondas periódicas são soluções para a equação da onda apenas se satisfizerem uma relação de dispersão: uma relação entre o *número de onda* k e a *frequência angular* w. O número de onda representa o número de ciclos da onda periódica dentro de um intervalo de 2π no eixo x e a frequência angular representa o número de ciclos da onda periódica que passa em uma posição x fixa dentro de um intervalo de tempo 2π.

Ainda no contexto das ondas progressivas, temos que uma função

$$u(x,t) = F(x+ct) + G(x-ct), \tag{2}$$

onde F e G são funções de classe C^2, que é a soma de duas funções que representam ondas progressivas também é solução para a equação da onda e é dita *solução geral.* A demonstração de (2) está apresentada no Apêndice A e também pode ser vista em Knobel (2000). Essa solução geral pode ser utilizada para demonstrar a *solução de D'Alambert* para a equação da onda

$$u(x,t) = \frac{1}{2}[f(x-ct) + f(x+ct)] + \frac{1}{2c}\int_{x-ct}^{x+ct} g(s)ds, \tag{3}$$

onde $f(x) = u(x,0)$ e $g(x) = u_t(x,0)$. A solução de D'Alambert parte de um *problema de valor inicial (PVI)* para a resolução da equação da onda. Por problema de valor inicial temos o problema de resolver uma equação diferencial parcial a qual já conhecemos o valor da solução e o valor da derivada da solução no instante de tempo inicial, f e g respectivamente. De posse dessas informações, a partir da solução de D'Alambert podemos determinar a função que define a solução da equação da onda. No Apêndice B mostramos que (3) é válida, assim como pode ser visto em Knobel (2000).

3 Método de diferenças finitas

O método de diferenças finitas é utilizado para a resolução numérica de equações diferenciais parciais, como a equação da onda. Com esse método podemos determinar não a função que satisfaz a equação da onda, mas sim aproximações para os valores dessa função em determinados

pontos no espaço e no tempo.

Esse método faz uso de aproximações numéricas para derivadas e é utilizado para a resolução de *problemas de valor de contorno (PVC)*. Nesse método as derivadas que aparecem na equação diferencial são substituídas por aproximações usando apenas valores da função computados sobre uma malha. Assim como nos problemas de valor inicial, os problemas de valor de contorno também são problemas de resolução de equações diferenciais parciais, porém, aqui, precisam ser conhecidos os valores da solução nos pontos, não apenas iniciais, mas em todo o contorno da malha.

As aproximações para as derivadas são construídas a partir do polinômio de Taylor. Se considerarmos os polinômios de Taylor de segunda e quarta ordem, define-se uma aproximação centrada para as derivadas de primeira e segunda ordem de uma função $u(x,t)$. A demonstração dessas aproximações serão omitidas neste relatório. As aproximações centradas para as derivadas parciais utilizadas no projeto são

$$\frac{\partial u(x_i,t_j)}{\partial t} \approx \frac{u_{i,j+1} - u_{i,j-1}}{2\Delta t}, \tag{4}$$

$$\frac{\partial^2 u(x_i,t_j)}{\partial t^2} \approx \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{\Delta t^2}, \tag{5}$$

$$\frac{\partial^2 u(x_i,t_j)}{\partial x^2} \approx \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{\Delta x^2}, \tag{6}$$

onde $u_{i,j}$ representa o valor da função $u(x,t)$ no ponto (x_i,t_j) da malha e Δx e Δt são, respectivamente, os passos em x e t da malha, ou seja, $\Delta x = x_{i+1} - x_i$ e $\Delta t = t_{j+1} - t_j$.

Neste projeto, o método de diferenças finitas foi utilizado para resolução do problema

$$\begin{aligned}
u_{tt} &= c^2 u_{xx},\ a < x < b,\ t > 0,\\
u(x,0) &= f(x),\ a < x < b,\\
u_t(x,0) &= g(x),\ a < x < b,\\
u(a,t) &= u_a,\ t > 0,\\
u(b,t) &= u_b,\ t > 0,
\end{aligned}$$

onde, temos um PVC no domínio espacial e um PVI no domínio temporal. A partir do problema de interesse e das aproximações para as derivadas parciais, podemos reescrever a equação da onda de acordo com o método de diferenças finitas. Substituindo (5) e (6) na equação da onda unidimensional em meio homogêneo, chegamos na equação de diferenças finitas que define as aproximações de $u(x,t)$ sobre os pontos da malha,

$$\frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{\Delta t^2} = c^2 \left[\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{\Delta x^2} \right]. \tag{7}$$

Rearranjando os termos de (7),

$$u_{i,j-1} - 2u_{i,j} + u_{i,j+1} = \left(\frac{c^2\Delta t^2}{\Delta x^2}\right)\left[u_{i-1,j} - 2u_{i,j} + u_{i+1,j}\right]. \tag{8}$$

Supondo conhecidos os valores os valores das aproximações $u_{i,j}$ para todo i no tempo t_j, para determinar $u_{i,j+1}$, podemos então organizar (8) como

$$\begin{aligned} u_{i,j+1} &= \left(\frac{c^2\Delta t^2}{\Delta x^2}\right)\left[u_{i-1,j} - 2u_{i,j} + u_{i+1,j}\right] - u_{i,t-1} + 2u_{i,t} \\ &= \left(\frac{c^2\Delta t^2}{\Delta x^2}\right)\left[u_{i-1,j} + u_{i+1,j}\right] - 2u_{i,j}\left(\frac{c^2\Delta t^2}{\Delta x^2}\right) - u_{i,j-1} + 2u_{i,j} \\ &= \left(\frac{c^2\Delta t^2}{\Delta x^2}\right)\left[u_{i-1,j} + u_{i+1,j}\right] + 2u_{i,j}\left(1 - \left(\frac{c^2\Delta t^2}{\Delta x^2}\right)\right) - u_{i,j-1}. \end{aligned}$$

Tomando $\Phi \equiv \frac{c^2\Delta t^2}{\Delta x^2}$, a equação acima pode ser reescrita finalmente como

$$u_{i,j+1} = \left[u_{i-1,j} + u_{i+1,j}\right]\Phi + 2u_{i,j}\left(1 - \Phi\right) - u_{i,j-1}. \tag{9}$$

Note que, quando $t = 0$, temos

$$u_{i,1} = \left[u_{i-1,0} + u_{i+1,0}\right]\Phi + 2u_{i,0}\left(1 - \Phi\right) - u_{i,-1}. \tag{10}$$

Como t_{-1} não pertence à malha temporal, podemos aproximar o termo $u_{i,-1}$ a partir da aproximação para a primeira derivada de $u(x,t)$ e a condição inicial para tal derivada. Sabendo que, pelas condições iniciais do problema,

$$\frac{\partial u(x,0)}{\partial t} = g(x) \tag{11}$$

e, pela aproximação por diferenças finitas,

$$\frac{\partial u(x,0)}{\partial t} \approx \frac{u_{i,j+1} - u_{i,j-1}}{2\Delta t},$$

igualando (11) e (4) e tomando $g(x_i) = g_i$, podemos escrever

$$\frac{u_{i,j+1} - u_{i,j-1}}{2\Delta t} = g_i.$$

Logo,

$$u_{i,-1} = u_{i,1} - 2\Delta t g_i. \tag{12}$$

Substituindo (12) em (10),

$$u_{i,1} = [u_{i-1,0} + u_{i+1,0}]\,\Phi + 2u_{i,0}\,(1-\Phi) - [u_{i,1} - 2h_t g_i] \tag{13}$$

$$= [u_{i-1,0} + u_{i+1,0}]\,\Phi + 2u_{i,0}\,(1-\Phi) - u_{i,1} + 2h_t g_i. \tag{14}$$

Além disso, novamente pelas condições inicias do problema, sabemos que

$$u(x,0) = f(x),$$

então, tomando $f(x_i) = f_i$ podemos escrever

$$u_{i,0} = f_i, \tag{15}$$

e assim, substituindo (15), (14) pode ser reescrita como

$$u_{i,1} = [f_{i-1} + f_{i+1}]\,\Phi + 2f_i\,(1-\Phi) + 2h_t g_i - u_{i,1} \tag{16}$$

$$= [f_{i-1} + f_{i+1}]\,\frac{\Phi}{2} + f_i\,(1-\Phi) + h_t g_i. \tag{17}$$

Então, quando $t = 0$, a equação (9) do método de diferenças finitas pode ser escrita como (17).

Por fim, para garantir a convergência do método é preciso que ele tenha estabilidade. Para que isso ocorra, segundo Porto (2020), é necessário que a condição de *Courant-Friedrichs-Lewy (CFL)* seja satisfeita. Essa condição é satisfeita quando os espaçamentos das malhas temporal e espacial satisfazem a seguinte relação

$$\frac{c^2 \Delta x^2}{\Delta t^2} < 1. \tag{18}$$

Entendendo o contexto e as equações do método de diferenças finitas, podemos agora entender a aplicação do método computacionalmente, utilizando o pacote Devito.

4 Devito

O *framework* Devito é um pacote escrito na linguagem de programação *Python* e tem por objetivo o cômputo de estênceis otimizados para o método de diferenças finitas para equação da onda. Esse módulo parte de definições de problemas simbólicos de alto nível. Um dos principais usos do pacote Devito é a criação de núcleos de propagação de ondas para serem usados em problemas de modelamento e inversão sísmica. Neste projeto, o pacote Devito foi utilizado para resolver a equação da onda unidimensional em meio homogêneo a partir do método de diferenças finitas de segunda ordem. Esta seção descreve detalhadamente como o pacote Devito pode ser instalado e como ele foi utilizado nesse projeto. Vale ressaltar que nesse projeto toda a implementação e utilização do pacote Devito foi feita no sistema operacional *Linux*.

4.1 Instalação

Primeiramente, para obter o pacote Devito é necessário clonar o repositório do projeto oficial do *framework*. Para isso basta usar o comando

```
pip install --user git+https://github.com/devitocodes/devito.git
```

no terminal.

Note que, para o comando funcionar, é necessário que estejam instalados na máquina a linguagem Python e o pacote de instalação *Pip*. Além disso, também é necessário que a máquina possua um compilador *C/C++* instalado[2]. Tudo isso é padrão de instalação do Linux.

Feito isso, o pacote Devito foi instalado e está salvo dentro do diretório usado para o clone. Quando o pacote é clonado, todos os pacotes Python necessários para o bom funcionamento do Devito são automaticamente instalados na máquina, ou seja, com o clone do repositório já temos tudo o que precisamos para começar.

Para conseguir acessar as ferramentas do pacote durante a implementação e funcionamento das rotinas, basta importar o Devito para o projeto. Para isso, são suficientes os seguintes códigos,

```
import devito
from devito import *
```

que importam o pacote Devito e, do pacote Devito, importam todas as suas ferramentas.

4.2 Estruturas

Para obter uma solução numérica para a equação da onda, algumas estruturas utilizadas no Devito são de extrema importância. A fim de esclarecer o funcionamento do *framework*, bem como o funcionamento e desenvolvimento dessas estruturas, esta seção detalha algumas ferramentas do pacote que são necessárias na implementação de uma rotina de modelamento da propagação da onda.

4.2.1 Malha

Para o desenvolvimento do método de diferenças finitas, é necessário, primeiramente, discretizar o domínio em que queremos obter uma solução para a equação da onda. Essa discretização é feita na geração da malha. Para gerar a malha que discretiza o domínio usamos a ferramenta `Grid()`. A malha pode ser definida usando o código

```
grid = Grid(shape, extent)
```

onde o parâmetro `shape` representa a discretização das dimensões espaciais da malha, ou seja, quantos pontos queremos que a malha tenha em cada direção. A malha pode ter até 3 dimensões

[2]Outras formas de instalação podem ser consultadas em: https://www.devitoproject.org/devito/download.html

espaciais. Já o parâmetro `extent` representa a extensão real do domínio a ser discretizado, em metros. Por exemplo, com o código

```
grid = Grid(shape=(1000), extent=(100))
```

é armazenado na variável `grid` uma malha que discretiza um domínio unidimensional com 100 metros de extensão na direção x, particionado-o em 1000 pontos. No Devito, não é necessário discretizar a dimensão temporal.

Até o momento, a malha é apenas uma estrutura e não armazena os valores da aproximação da solução. Os valores da aproximação da solução nos pontos da malha são armazenados na estrutura `data`, que veremos na subseção 4.2.3.

4.2.2 Funções simbólicas

Com o domínio discretizado, agora é necessário definir a função a qual aproximaremos seus valores nos pontos da malha. O Devito faz uso de expressões matemáticas simbólicas, então para facilitar a visualização, podemos utilizar os códigos

```
from sympy import pprint
init_printing(use_latex=True)
```

que permitem visualização, no terminal, das expressões matemáticas definidas. No âmbito desse projeto, o objetivo é definir uma função que varia no tempo e espaço e que será aproximada no domínio discretizado e estruturado pela malha previamente definida. Para isso, podemos utilizar a função `TimeFunction()`, como no seguinte código.

```
u = TimeFunction(name='u', grid=grid, time_order=2, space_order=2)
```

Dentre os parâmetros da função `TimeFunction()`, para definir a função $u(x,t)$ são passados quatro deles: `name`, `grid`, `time_order` e `space_order`. No parâmetro `name` é estabelecido o nome da função que está sendo definida. Passando o parâmetro `grid`, a malha onde a função será avaliada é indicada. E, por fim, com os parâmetros `time_order` e `space_order`, determinam-se a ordem de discretização das derivadas temporais e espaciais, respectivamente.

4.2.3 Data

A função $u(x,t)$ definida tem uma outra estrutura associada, a `data`, que serve para armazenar os valores da função nos pontos da malha. A critério de facilitar o entendimento, pode-se visualizar essa estrutura como uma matriz de 3 linhas e Nx colunas, sendo Nx a quantidade de pontos da malha na direção x. Cada uma das linhas dessa matriz armazena os valores de $u(x,t)$ em um instante de tempo. No método de diferenças finitas de segunda ordem, para aproximar $u_{i,j+1}$, precisam ser conhecidos os valores $u_{i,j}$, $u_{i,j-1}$, $u_{i-1,j}$ e $u_{i+1,j}$, ou seja, são esses, t_{j-1}, t_j e t_{j+1}, os três instantes de tempo que são armazenados em `data`. A cada iteração esses valores são renovados, mas, vale aqui ressaltar, que o armazenamento dos valores da função na estrutura `data` é rotativo, ou seja, os instantes de tempo não assumem posições fixas dentro

da estrutura. O entendimento dessa rotatividade vem do entendimento do funcionamento da função `Operator()`, que será apresentado na subseção 4.2.7.

Para acessarmos a estrutura `data` da função, podemos usar o código

```
u.data
```

ou, para acessar o valor da função em apenas um instante de tempo, podemos usar o código indicando qual das linhas da `data` queremos. Para acessar a primeira linha, por exemplo, usamos o seguinte código.

```
u.data[0]
```

Também podemos acessar o valor em um ponto específico. Para isso, basta indicar a linha e a coluna de interesse, como no código a seguir, em que acessamos o primeiro elemento da primeira linha da `data`.

```
u.data[0][0]
```

4.2.4 Equação da onda simbólica

A próxima etapa é definir a expressão simbólica da equação diferencial que queremos resolver. No caso, vamos definir a equação da onda unidimensional em meio homogêneo. Para tal, utilizamos o código

```
equacao_onda = (1 / c ** 2) * u.dt2 - u.dx2
```

onde, onde `c` é uma função simbólica que define velocidade de propagação. Com esse código é possível definir as expressões matemáticas simbólicas das derivadas parciais de segunda ordem espaciais e temporais de uma função facilmente, usando as funções `dt2` e `dx2`.

4.2.5 Estêncil

A próxima etapa é definir a equação (9) do método de diferenças finitas. Para tal vamos utilizar os comandos

```
import sympy
from sympy import Eq
lado_esquerdo = u.forward
lado_direito = solve(equacao_onda, u.forward)
stencil = Eq(lado_esquerdo, lado_direito)
```

que utilizam função `Eq()`, que é uma ferramenta do pacote *Sympy* do Python, que já é instalado junto com o Devito. Essa função é utilizada para definir equações simbólicas. No primeiro parâmetro da função passamos a expressão que queremos que esteja do lado esquerdo da equação e no segundo parâmetro, a do lado direito da equação. Com o parâmetro `u.forward` criamos a expressão simbólica para $u(x, t + \Delta_t)$, ou seja, $u_{i,j+1}$. Agora, com o parâmetro `solve(equacao_onda, u.forward)` estamos criando o estêncil do método de diferenças finitas para a equação que definimos, ou seja, o lado direito de (9).

4.2.6 Condições de contorno

A últimas definições que precisam ser feitas são as condições de contorno. Para o problema que estamos resolvendo, o valor da função $u(x,t)$ nas bordas do domínio são constantes. Logo, basta usarmos os códigos

```
t_s = grid.stepping_dim
bc = [Eq(u[t_s + 1, 0], u_a)]
bc += [Eq(u[t_s + 1, Nx], u_b)]
```

para definir as duas equações simbólicas que representam o valor de $u(x,t)$ nas bordas da malha, ou seja, $u_{0,j+1} = u_a$ e $u_{x_n,j+1} = u_b$. Outros tipos de condições de contorno também podem ser definidas.

4.2.7 Operador

Depois de definidas todas as expressões necessárias, vamos criar um operador, que é uma estrutura que pode ser utilizada como uma função, que, ao ser chamada, aplica o estêncil considerando os dados já definidos. Para isso, a função `Operator()` é utilizada, recebendo como parâmetro o estêncil e as condições de contorno. Para criar o operador e chamá-lo, usamos os seguintes códigos.

```
op = Operator([stencil]+bc)
op.apply(time_m=1, time_M=Nt, dt=dt)
```

Como parâmetros para aplicar o operador, passamos a quantidade iterações no tempo e o passo Δt. Para definir os tempos, passamos o passo mínimo `time_m` e o passo máximo `time_M` e, para definir Δt passamos o parâmetro dt, onde $dt = \Delta t$. Vale ressaltar que os valores que passamos nos parâmetros `time_m` e `time_M` são os índices da discretização temporal e não os tempos em si. Por exemplo, se temos `time_m` $= 1$ e `time_M`=20, significa que vamos percorrer os tempo entre t_1 e t_{20}.

Quando o operador é aplicado, a cada iteração ele vai atualizando os valores armazenados na estrutura `data` e, como mencionado na seção 4.2.3, esse armazenamento é rotativo. Quando aplicamos o operador, na verdade estamos gerando e compilando um código otimizado em linguagem C. Com o comando `print(op.ccode)`, imprimimos na tela esse código C que foi gerado e podemos visualizar como o armazenamento funciona e entender a ordem em que os valores são salvos. Não há necessidade de entender o código em C para utilizar o Devito, pois a ferramenta é construída com o objetivo de facilitar o trabalho do usuário, mas é importante entender a forma como os dados são armazenados para que possamos acessar corretamente os dados de interesse. Analisando o funcionamento do operador entendemos que os valores da função calculados em cada iteração são sempre salvos na posição `u.data[t2]`. O índice t_2 é calculado como $t_2 = (it + 1)\%3$, sendo o operador % o resto da divisão e it o índice do tempo. Por exemplo, das condições iniciais do problema, conhecemos o valor de $u_{i,0}$. Esse valor é do instante de tempo t_0, logo, o índice do tempo é $it = 0$. Usando a relação para o cálculo do

índice de armazenamento em `data`, sabemos que $u_{i,0}$ será armazenado na posição $t_2 = 1$. Como no esquema a seguir

$$\begin{bmatrix} [\quad\quad] \\ [\quad u_{i,0} \quad] \\ [\quad\quad] \end{bmatrix}.$$

Na primeira iteração vamos calcular $u_{i,1}$, ou seja, estamos no instante t_1 de tempo, logo, temos que $it = 1$ e $t_2 = 2$. Então,

$$\begin{bmatrix} [\quad\quad] \\ [\quad u_{i,0} \quad] \\ [\quad u_{i,1} \quad] \end{bmatrix}.$$

Seguindo dessa forma, na segunda iteração $t_2 = 0$, logo,

$$\begin{bmatrix} [\quad u_{i,2} \quad] \\ [\quad u_{i,0} \quad] \\ [\quad u_{i,1} \quad] \end{bmatrix},$$

e assim sucessivamente. Para mais detalhes, o código em C gerado nas implementações desse projeto pode ser visto no Apêndice D.

5 Resultados e discussão

Primeiramente, a partir dos estudos teóricos sobre as ondas e os estudos sobre o método de diferenças finitas foi implementada, a fim de enriquecer o entendimento sobre o assunto, uma rotina em *Octave* para a resolução da equação da onda unidimensional em meio homogêneo a partir do método de diferenças finitas. Nessa rotina, a malha e todas equações necessárias para método de diferenças finitas foram implementadas de acordo com a teoria apresentada na Seção 3. Essa implementação foi utilizada também como estudo, então partiu-se de uma solução analítica para comparar os resultados esperados com os resultados obtidos, permitindo um melhor entendimento das soluções.

Além disso, a partir de todo entendimento adquirido ao longos dos estudos sobre o pacote Devito e suas ferramentas, foi implementada, em Python, uma rotina para resolução da equação da onda unidimensional em meio homogêneo utilizando as ferramentas do pacote Devito. Para testar a funcionalidade do Devito, novamente partiu-se de uma solução analítica. Foi assumido uma solução da forma $u(x,t) = f(x-ct)$, ou seja, uma onda progressiva propagando na direção positiva do eixo x. A rotina implementada pode ser vista no Apêndice C. A implementação seguiu as orientações propostas na Seção 4, porém foi acrescentada uma nova funcionalidade para o código, a chamada de uma função para plotar os gráficos das soluções obtidas, facilitando a análise dos resultados. Essa função também pode ser vista no Apêndice C.

A rotina foi utilizada para resolver o problema de valor de contorno

$$\begin{aligned} u_{tt} &= c^2 u_{xx}, \quad -20 < x < 20, \ t > 0, \\ u(x,0) &= f(x), \quad -20 < x < 20, \\ u_t(x,0) &= g(x), \quad -20 < x < 20, \\ u(-20,t) &= 0, \ t > 0, \\ u(20,t) &= 0, \ t > 0, \end{aligned}$$

onde,

$$f(x) = \begin{cases} \cos^2(x), & |x| < \pi/2, \\ 0, & \text{caso contrário,} \end{cases}$$

e,

$$g(x) = \begin{cases} -2\cos(x)\sin(x), & |x| < \pi/2, \\ 0, & \text{caso contrário.} \end{cases}$$

Para a geração da malha, consideramos $x \in [-20, 20]$ e $t \in [0, 25]$, com o passo $\Delta t = 0.004$ s. Além disso, definimos o número de Courant como sendo $C = 0.5$, logo, satisfazendo a relação (18), a discretização espacial do domínio foi feita com passo $\Delta x = 0.008$ m. Por fim, determinamos a velocidade de propagação da onda como sendo $c = 1$ m/s.

Como estamos utilizando o método de diferenças finitas de segunda ordem, precisamos, para começar, dos valores da equação da onda em dois pontos da malha: $u_{i,0}$ e $u_{i,1}$. Então, como argumento para função que chama a rotina implementada, passamos, não só o valor $f(x)$, como também $f(x - ct_1)$. Baseando-se na teoria estudada e na solução analítica pré determinada, o resultado obtido pelo operador do Devito foi como o esperado. Os gráficos da Figura 1 ilustram o resultado. Graficamente vemos que, como a borda utilizada é fixa e não absorvente, a onda bate na borda direita da malha e reflete com fase invertida. Além disso, como as duas extremidades da borda são fixas e não absorventes, toda vez que a onda bate em uma das extremidades essa reflexão ocorre e ela se mantém até atingir o tempo máximo de simulação, uma vez que, no exemplo criado, também não está sendo considerado o amortecimento.

O principal resultado obtido ao longo desse projeto, atingindo o objetivo inicial, foi o entendimento e a utilização do pacote Devito. Essa etapa do projeto foi a que demandou mais tempo e esforço, uma vez que grande parte das documentações e exemplos relacionados ao Devito, que puderam ser encontrados, não foram apresentadas de forma clara e eficiente, dificultando as implementação de rotinas para resolução de problemas básicos. Devido a tais circunstâncias, ao longo do projeto ficou claro que uma explicação detalhada e passo a passo sobre a utilização do pacote, partindo de uma situação básica, era necessária. Com isso, o objetivo principal desse projeto foi, não apenas mostrar os resultados obtidos com as simulações, como também servir como um guia para quem deseja começar a utilizar o *framework*. A utilização mecânica das ferramentas do Devito é mais simples de ser entendida, mas quando é necessário fazer adaptações

ou acessar dados, é muito útil ter um entendimento claro de como as funções que são utilizadas nas rotinas trabalham. Saber como as funções acessam os dados necessários e armazenam os dados simulados, facilita a conferência dos resultados, permitindo notar inconsistências ou divergências, e facilita também a análise gráfica, já que, conhecendo a forma de armazenamento, conseguimos acessar os resultados exatamente nos pontos que queremos da malha. As principais e mais relevantes informações obtidas foram detalhadas nas seções anteriores.

6 Considerações finais

Em conclusão, os resultados obtidos até o momento foram satisfatórios. Inicialmente houve muita dificuldade na compreensão do funcionamento das estruturas do pacote Devito. A principal dificuldade surgiu na adaptação, para um problema simples, das implementações consultadas. Todas essas implementações tinham um formato, de certa forma, padrão e nenhuma delas deixava claro como as estruturas funcionavam, o que impediu o entendimento do porquê, a principio, os resultados obtidos não tinham qualidade. A estrutura mais difícil de entender foi a `Operator`, uma vez que não era sabido como operador faz uso da estrutura `data`, o que impediu a visualização e o acesso aos resultados, impossibilitando o entendimento do que estava acontecendo. A partir do momento em que foi possível acessar o código em C do operador, tudo ficou mais claro e o entendimento da rotatividade de armazenamento permitiu que o estudo avançasse. A partir do momento em que as estruturas foram compreendidas, foi possível notar como as estruturas do Devito funcionam em harmonia umas com as outras e como facilitam a simulação. Esse projeto ainda não está concluído. Com o conhecimento estabelecido, pretende-se adaptar a rotina implementada para problemas que envolvem um maior nível de complexidade, como ondas bidimensionais e bordas absorventes.

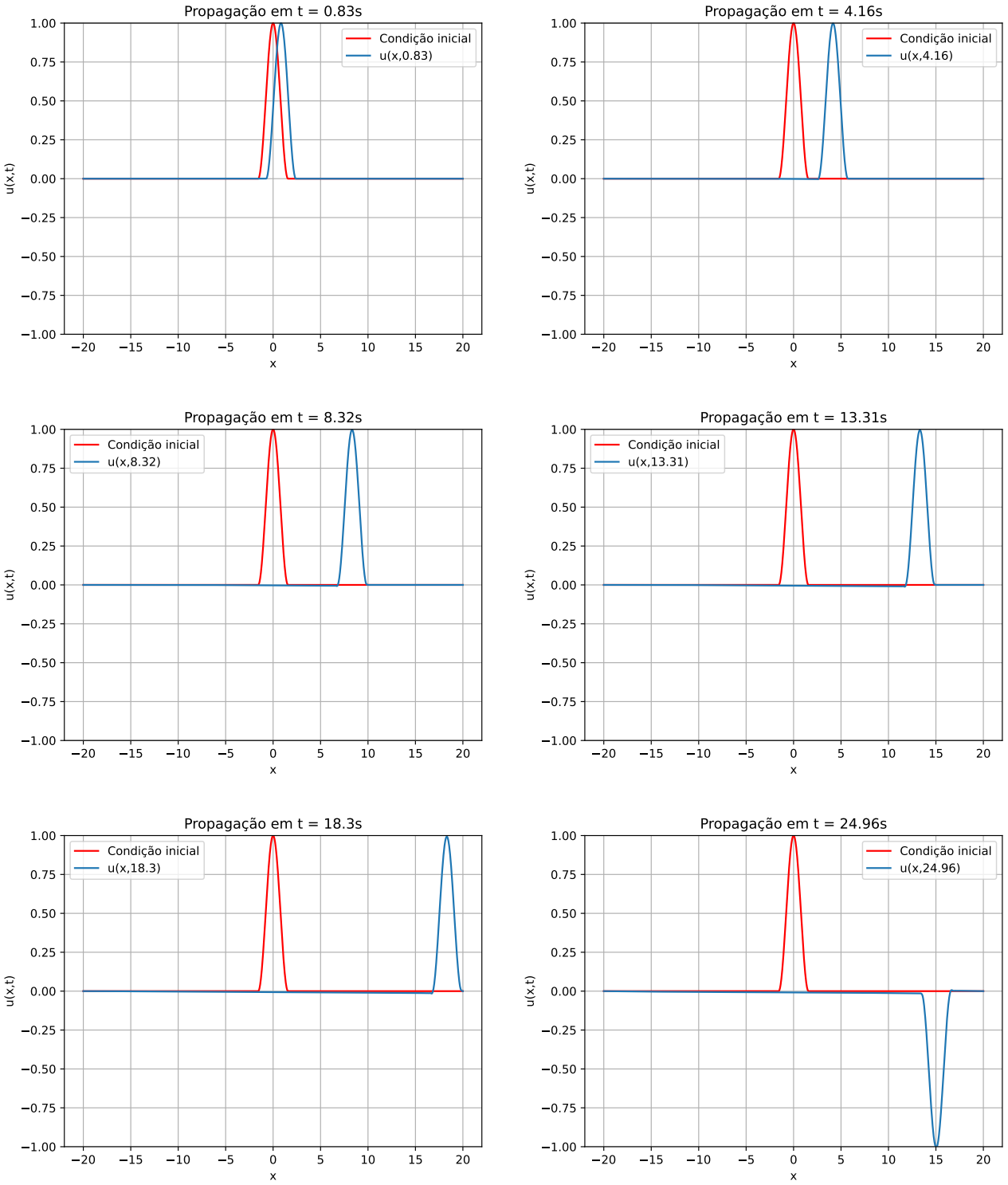

Figura 1: Solução da equação da onda unidimensional em meio homogêneo obtida pelo Devito.

A Solução geral da equação da onda

Vamos mostrar, primeiramente, que a equação da onda

$$u_{tt} = v^2 u_{xx},\ v > 0 \tag{19}$$

admite soluções das formas $f(x - ct)$ e $f(x + ct)$, onde f é uma função de duas variáveis.

Tomando $u(x, t) = f(x - ct)$, temos que

$$u_t(x, t) = -cf'(x - ct),$$

$$u_x(x, t) = f'(x - ct).$$

Derivando novamente,

$$u_{tt}(x, t) = c^2 f''(x - ct),$$

$$u_{xx}(x, t) = f''(x - ct).$$

Substituindo as derivadas de segunda ordem na equação (19), temos

$$c^2 f''(x - ct) = v^2 f''(x - ct),$$

e, reorganizando os termos,

$$(c^2 - v^2) f''(x - ct) = 0.$$

Note que, da equação acima, temos duas possibilidades

$$c^2 = v^2$$

ou,

$$f''(x - ct) = 0.$$

Logo, para o caso em que $c^2 = v^2$, se $c = \pm\sqrt{v}$ e f for uma função duas vezes diferenciável e não constante, concluímos que as funções

$$f(x - \sqrt{v}t),$$

$$f(x + \sqrt{v}t),$$

são soluções para a equação (19). E, para o caso em que $f''(x - ct) = 0$ concluímos que toda função do tipo

$$f(x - ct) = a + b(x - ct),$$

é solução de (19), para todo a, b e c, tal que $b \neq 0$ e $c > 0$.

■

Agora, sabendo que funções do tipo $f(x \pm ct)$ são soluções de (28), vamos mostrar que funções do tipo

$$u(x,t) = F(x-ct) + G(x+ct),$$

que são a soma de funções que representam ondas progressivas, também são soluções de (28).

Sendo $f(x-ct)$ e $g(x+ct)$ soluções de (19), vamos tomar $A = x - ct$ e $B = x + ct$. Agora, definindo

$$u(x,t) = U(A,B),$$

onde U é uma função de A e B, podemos escrever

$$u_t(x,t) = U_A(x-ct)_t + U_B(x+ct)_t = -cU_A + cU_B,$$

$$u_x(x,t) = U_A(x-ct)_x + U_B(x+ct)_x = U_A + U_B,$$

sendo U_A e U_B, as derivadas parciais de U com relação a A e B, respectivamente. Derivando novamente,

$$\begin{aligned} u_{tt}(x,t) &= -c\left(U_{AA}(x-ct)_t + U_{AB}(x+ct)_t\right) + c\left(U_{BA}(x-ct)_t + U_{BB}(x+ct)_t\right) \\ &= c\left(cU_{AA} - cU_{AB}\right) + c\left(-cU_{BA} + cU_{BB}\right) \\ &= c^2\left(U_{AA} - U_{AB}\right) + c^2\left(U_{BB} - U_{BA}\right) \\ &= c^2U_{AA} - 2c^2U_{AB} + c^2U_{BB}, \end{aligned}$$

$$\begin{aligned} u_{xx}(x,t) &= U_{AA}(x-ct)_x + U_{AB}(x+ct)_x + U_{BA}(x-ct)_x + U_{BB}(x+ct)_x \\ &= U_{AA} + U_{AB} + U_{BA} + U_{BB} \\ &= U_{AA} + 2U_{AB} + U_{BB}, \end{aligned}$$

então, substituindo as derivadas acima na equação da onda, temos

$$c^2U_{AA} - 2c^2U_{AB} + c^2U_{BB} = c^2\left(U_{AA} + 2U_{AB} + U_{BB}\right).$$

Reorganizando os termos,

$$-2c^2U_{AB} = 2c^2U_{AB},$$

e então,

$$U_{AB} = 0.$$

Sabendo que $(U_A)_B = 0$, podemos concluir que U_A não é uma função de B, ou seja, $U_A = \phi(A)$. Logo, podemos escrever

$$\int U_A dA = \int \phi(A) dA,$$

e então,

$$U(A,B) = F(A) + G(B), \tag{20}$$

onde $F(A)$ é a primitiva de $\phi(A)$ e $G(B)$ é a constante de integração com respeito a A. Substituindo A e B em (20), concluímos que

$$u(x,t) = U(A,B) = F(x-ct) + G(x+ct)$$

■

B Solução de D'Alambert

Vamos mostrar que, dado um PVI

$$\begin{aligned} u_{tt} &= c^2 u_{xx}, -\infty < x < \infty, t > 0, \\ u(x,0) &= f(x), -\infty < x < \infty, \\ u_t(x,0) &= g(x), -\infty < x < \infty, \end{aligned}$$

a solução da equação da onda pode ser escrita como

$$u(x,t) = \frac{1}{2}[f(x-ct) + f(x+ct)] + \frac{1}{2c}\int_{x-ct}^{x+ct} g(s)ds. \tag{21}$$

Tomando a solução geral da equação $u_{tt} = c^2 u_{xx}$ como a soma de duas ondas progressivas

$$u(x,t) = F(x-ct) + G(x+ct), \tag{22}$$

podemos reescrever (22) utilizando as condições de contorno. Sabendo que $u(x,0) = f(x)$ e $u_t(x,0) = g(x)$, temos que

$$u(x,0) = F(x) + G(x) = f(x) \text{ e,} \tag{23}$$

$$u_t(x,t) = -cF'(x-ct) + cG'(x+ct). \tag{24}$$

Logo,

$$u_t(x,0) = -cF'(x) + cG'(x) = g(x). \tag{25}$$

Então, de (25) temos,

$$-F'(x) + G'(x) = \frac{1}{c}g(x). \tag{26}$$

Integrando ambos os lados de (26),

$$\int_0^x -F'(x) + G'(x)ds = \int_0^x \frac{1}{c}g(s)ds, \tag{27}$$

temos,

$$[-F(x) + G(x)]_0^x = \frac{1}{c}\int_0^x g(s)ds, \tag{28}$$

reorganizando os termos,

$$-F(x) + G(x) = -F(0) + G(0) + \frac{1}{c}\int_0^x g(s)ds. \tag{29}$$

Agora, de (23) e (29) temos o sistema

$$\begin{cases} F(x) + G(x) &= f(x), \\ -F(x) + G(x) &= -F(0) + G(0) + \frac{1}{c}\int_0^x g(s)ds. \end{cases} \tag{30}$$

Somando as duas equações do sistema (30), temos

$$G(x) = \frac{1}{2}f(x) + \frac{1}{2}[-F(0) + G(0)] + \frac{1}{2c}\int_0^x g(s)ds. \tag{31}$$

Subtraindo as duas equações do sistema (30), temos

$$F(x) = \frac{1}{2}f(x) - \frac{1}{2}[-F(0) + G(0)] - \frac{1}{2c}\int_0^x g(s)ds. \tag{32}$$

Substituindo os valores de $F(x)$ e $G(x)$ na solução geral do PVI,

$$\begin{aligned} u(x,t) &= F(x-ct) + G(x+ct) \\ &= \frac{1}{2}f(x-ct) - \frac{1}{2}[-F(0) + G(0)] - \frac{1}{2c}\int_0^{x-ct} g(s)ds \\ &+ \frac{1}{2}f(x+ct) + \frac{1}{2}[-F(0) + G(0)] + \frac{1}{2c}\int_0^{x+ct} g(s)ds \\ &= \frac{1}{2}[f(x-ct) + f(x+ct))] + \frac{1}{2c}\int_{x-ct}^{x+ct} g(s)ds. \end{aligned}$$

■

C Rotina para resolução da equação da onda

```
from sympy import Eq, solve, init_printing, pprint
init_printing(use_latex=True)
from grafico import grafico
import numpy as np
from devito import Grid, TimeFunction, Function, Eq, solve, Operator
import matplotlib.pyplot as plt

def solver(f_0, f_1, c, a, b, dt, C, T, u_a, u_b, delta_grafico):
    """
    Rotina para resolver a equacao da onda unimensional u_tt=c^2*u_xx no
   dominio (a,b)x(0,T], utilizando as ferramentas do pacote Devito.
    :param f_0: Condicao inicial: valor da funcao u(x,0),
    :param f_1: Condicao inicial: valor da funcao no segundo ponto temporal
   da malha u_{i,1},
    :param c: Velocidade de propagacao da onda. Pode ser uma constante ou
   uma funcao de x,
    :param a: Limite esquerdo da intervalo espacial,
    :param b: Limite direito da intervalo espacial,
    :param dt: Variacao temporal,
    :param C: Numero de Courant,
    :param T: Limite maximo de tempo (em segundos),
    :param u_a: Valor da funcao quando x = a,
    :param u_b: Valor da funcao quando x = b,
    :param delta_grafico: Intervalo em que os graficos serao plotados,
    :return:
    """

    # DEFININDO A MALHA

    Nt = int(round(T / dt))
    t = np.linspace(0, Nt * dt, Nt + 1)
    dx = dt * c / float(C)
    Nx = int(round(b-a / dx))
    x = np.linspace(a, b, Nx + 1)
    grid = Grid(shape=(Nx + 1, ), extent=(b - a, ))

    # PASSO TEMPORAL

    dt = t[1] - t[0]

    # CRIANDO A FUNCAO u(x,t)

    u = TimeFunction(name='u', grid=grid, time_order=2, space_order=2)
    u.data[1, :] = f_0
    u.data[2, :] = f_1
```

```
    # DEFININDO A EQUACAO DA ONDA

    equacao_onda = (1 / c ** 2) * u.dt2 - u.dx2

    # DEFININDO OS ESTENCIL

    stencil = Eq(u.forward, solve(equacao_onda, u.forward))

    # DEFININDO AS CONDICOES DE CONTORNO
    bc = [Eq(u[t_s + 1, 0], u_a)]
    bc += [Eq(u[t_s + 1, Nx], u_b)]

    # DEFININDO O OPERADOR
    op = Operator([stencil] + bc)
    n = 0
    for it in range(2, Nt):
        index = (it + 1) % 3
        op.apply(time_m=it, time_M=it, dt=dt)
        if it % delta_grafico == 0:
            n += 1
            grafico(x, u.data[index], it, n)
    return

def grafico(x, u, it, n, t, f_0):
    tt = t[it]
    tt = round(tt, 2)
    plt.plot(x, f_0, color='r', label = 'Condicao inicial')
    plt.plot(x, u, label = fr'u(x,{tt})')
    plt.ylim(-1,1)
    plt.legend()
    plt.grid()
    plt.title(fr'Propagacao em t = {tt}s')
    plt.xlabel('x')
    plt.ylabel(fr'u(x,t)')
    plt.savefig()
    plt.close()
```

Listagem 1: Rotina para resolução da equação da onda unidimensional em meio homogêneo.

D Código em C da função `Operator()`

```
#define _POSIX_C_SOURCE 200809L
#define START_TIMER(S) struct timeval start_ ## S , end_ ## S ; gettimeofday
    (&start_ ## S , NULL);
#define STOP_TIMER(S,T) gettimeofday(&end_ ## S, NULL); T->S += (double)(
    end_ ## S .tv_sec-start_ ## S.tv_sec)+(double)(end_ ## S .tv_usec-start_
    ## S .tv_usec)/1000000;
#include "stdlib.h"
#include "math.h"
#include "sys/time.h"
#include "xmmintrin.h"
#include "pmmintrin.h"
struct dataobj
{
  void *restrict data;
  int * size;
  int * npsize;
  int * dsize;
  int * hsize;
  int * hofs;
  int * oofs;
} ;
struct profiler
{
  double section0;
} ;
int Kernel(const float dt, const float h_x, struct dataobj *restrict u_vec,
    const int time_M, const int time_m, const int x_M, const int x_m, struct
    profiler * timers)
{
  float (*restrict u)[u_vec->size[1]] __attribute__ ((aligned (64))) = (
    float (*)[u_vec->size[1]]) u_vec->data;
  /* Flush denormal numbers to zero in hardware */
  _MM_SET_DENORMALS_ZERO_MODE(_MM_DENORMALS_ZERO_ON);
  _MM_SET_FLUSH_ZERO_MODE(_MM_FLUSH_ZERO_ON);
  float r0 = 1.0F/(dt*dt);
  float r1 = 1.0F/(h_x*h_x);
  for (int time = time_m, t0 = (time)%(3), t1 = (time + 2)%(3), t2 = (time +
    1)%(3); time <= time_M; time += 1, t0 = (time)%(3), t1 = (time + 2)%(3),
    t2 = (time + 1)%(3))
  {
    /* Begin section0 */
    START_TIMER(section0)
    for (int x = x_m; x <= x_M; x += 1)
    {
      float r2 = -2.0F*u[t0][x + 2];
```

```
      u[t2][x + 2] = (dt*dt)*(-r0*r2 - r0*u[t1][x + 2] + r1*r2 + r1*u[t0][x
   + 1] + r1*u[t0][x + 3]);
    }
    STOP_TIMER(section0,timers)
    /* End section0 */
    u[t2][2] = 0;
    u[t2][2522] = 0;
  }
  return 0;
}
```

Referências

Biloti, R. (2020). Propagação de ondas sísmicas. http://www.ime.unicamp.br/~biloti/geo/propagacao.pdf (acessado em 31/01/2022).

Biloti, R. (2021). Diferenças finitas. https://www.ime.unicamp.br/~biloti/an/211/edo-07.html (acessado em 10/02/2022).

Biloti, R. (2022). Processamento sísmico. http://www.ime.unicamp.br/~biloti/geo/notas.pdf (acessado em 31/01/2022).

Knobel, R. (2000). *An introduction to the mathematical theory of waves*, volume 3. American Mathematical Soc.

London, I. C. e others (2016). Devito: Symbolic finite difference computation.

Louboutin, M., Lange, M., Luporini, F., Kukreja, N., Witte, P. A., Herrmann, F. J., Velesko, P., e Gorman, G. J. (2019). Devito (v3.1.0): an embedded domain-specific language for finite differences and geophysical exploration. *Geoscientific Model Development*, 12(3):1165–1187.

Porto, D. L. B. (2020). Uma simulação numérica da equação da onda unidimensional pelo método das diferenças finitas: formulação explícita.

Santos, G. M. d., Ávila, M. d., Neto, H. R., e Neto, A. d. S. (2019). *Análise numérica da equação da onda unidimensional em regime transiente pelo método explícito*. Atena Editora.

Wencai, Y. (2013). *Reflection Seismology: theory, data processing and interpretation*. Elsevier.