INF2102 Projeto Final de Programação

Aluno: Andre Grigorio (1912698)

Prof. Orientador: Helio Lopes

A. Especificação do Programa

1. Introdução

1.1. Finalidade

A finalidade do programa é a de entregar uma ferramenta para análise e conversão de dados sísmicos como forma de preparação de *dataset* em etapa de pré-processamento anterior ao treinamento de redes neurais. O filtro terá como entrada um cubo sísmico e terá como saída a geração de estimativas dos atributos geométricos volumétricos *dip magnitude* e *dip azimuth*, que são de fundamental relevância para a estratigrafia sísmica, além de formarem a base para a derivação de outros atributos geométricos de grande importância para a avaliação da existência e potencial de migração de hidrocarbonetos.

1.2. Público-alvo

Uma ferramenta como a proposta por esse trabalho é de potencial interesse para estudiosos e profissionais da área de prospecção de hidrocarbonetos, que possam vir a utilizar os atributos gerados como meio de identificar regiões de interesse dos derivados de petróleo.

1.3. Escopo

O programa em questão recebe como entrada um cubo sísmico no formato SEG-Y, que é um dos padrões mais usados para armazenamento de dados geofísicos, desenvolvido pela *Society of Exploration Geophysicists* (SEG).

Os dados brutos de um cubo sísmico efetivamente representam o tempo gasto para que uma onde sísmica viaje a partir da fonte até um dado refletor geológico e retorne ao receptor na superfície da Terra, esse intervalo é chamado de *two-way travel time – TWT,* porém, considerando o objetivo de identificar fraturas e outras falhas geológicas de interesse, faz-se necessária a derivação de atributos geológicos mais complexos.

Um atributo sísmico de qualidade é qualquer medida sísmica derivada que guarda correlação direta com algum aspecto geológico ou propriedade de reservatório de hidrocarbonetos e permite uma visualização melhorada dessa região de interesse, o que facilita a análise por parte de um interpretador ou ainda permite o uso de técnicas de aprendizagem de máquina possibilitando o processamento automático de um grande volume de dados com boa acurácia quando comparada com um interpretador humano.

Os atributos volumétricos de *Dip magnitude* e *Dip Azimuth*, ou simplesmente *dip* e *azimuth*, são, questionavelmente, dos mais importantes para a análise estratigráfica pois, efetivamente

mapeiam diretamente os refletores existentes num cubo sísmico, permitindo a visualização das faltas mais súbitas. Infelizmente devido a distorções no modelo de velocidade, as estimativas de *dp* e *azimuth* derivadas de cubos sísmicos migrados no tempo divergem dos valores absolutos reais desses parâmetros para um dado refletor, contudo, considerando que para a análise de faltas as interpretações são feitas sobre as variações desses atributos, os erros inerentes são negligíveis.

2. Requisitos de Sistema

2.1. Requisitos Funcionais

- O sistema deve receber como entrada arquivo representativo de cubo sísmico no formato SEG-Y.
- O sistema deverá gerar na saída arquivos individuais para os atributos de *dip magnitude* e *dip azimuth*.
- O sistema deve gerar na saída diagrama 3D interativo que permita visualização ao longo das *inlines*, *crosslines*¸ e *timeslices*.

2.2. Requisitos Não Funcionais

- Para o cálculo dos atributos de *dip magnitude* e *dip azimuth* deve ser usada a abordagem *Complex Seismic Trace Analysis*.
- Os valores individuais nos cubos de saída devem estar no formato de ponto flutuante.
- Deve-se aproveitar de rotinas da linguagem para otimizar operações matemáticas.

B. Projeto do Programa

O programa foi desenvolvido na linguagem Python e, considerando que a solução do problema proposto consiste numa sequência de operações matemáticas, sua arquitetura é puramente linear e procedural visto não ser necessário reuso de código ao longo de sua execução.

De acordo com *Chopra*, *Marfurt* e *Barnes*, a técnica de *Complex Seismic Trace Analysis* para obtenção de *Dip Magnitude* e *Dip Azimuth* essencialmente consiste no tratamento sucessivo dos dados sísmicos até a obtenção do resultado final.

Numa primeira etapa é usada a transformada de Hilbert unidimensional ao longo do eixo de tempo para se obter a informação de fase do sinal e termos o traço complexo, e, por conseguinte sua fase instantânea.

Com a informação de fase instantânea, a frequência instantânea, ω, desse sinal é facilmente obtida pelo método das diferenças finitas de Euler.

Similarmente, a partir do cubo de fase instantânea, diferenciando-se ao longo dos eixos de *inline* e *crossline*, ao invés do eixo de tempo, obtêm-se os números de onda instantâneo k_x e k_y, respectivamente.

A partir da frequência e números de onda instantâneos, são obtidos os *apparent time dips* p e q, conforme as expressões a seguir:

$$p = \frac{k_x}{\omega} \qquad\qquad q = \frac{k_y}{\omega}$$

Finalmente, *Dip Magnitude*, ∅, e *Dip Azimuth*, s, são obtidos pelas operações seguintes:

$$\emptyset = ATAN2(q, p) \qquad\qquad s = \sqrt{(p^2 + q^2)}$$

O programa é organizado em uma única classe, chamada *Cube*, que, em sua instanciação inicial, configura diversos atributos na forma de constantes e variáveis associadas ao cubo sísmico em tratamento. Adicionalmente, associados à referida classe existem cinco métodos que executados em sequência executam os procedimentos necessários para obtenção dos atributos de *Dip Magnitude* e *Dip Azimuth*, conforme descrito acima.

Cube
-seismic_cube: ndarray -ILINE: int -XLINE: int -TSLICE: int -DIL: float -DXL: float -DTS: float
-calc_analytic() -calc_inst_freq() -calc_inst_wavenumb_kx() -calc_inst_wavenumb_ky() -calc_dip()

C. Código fonte

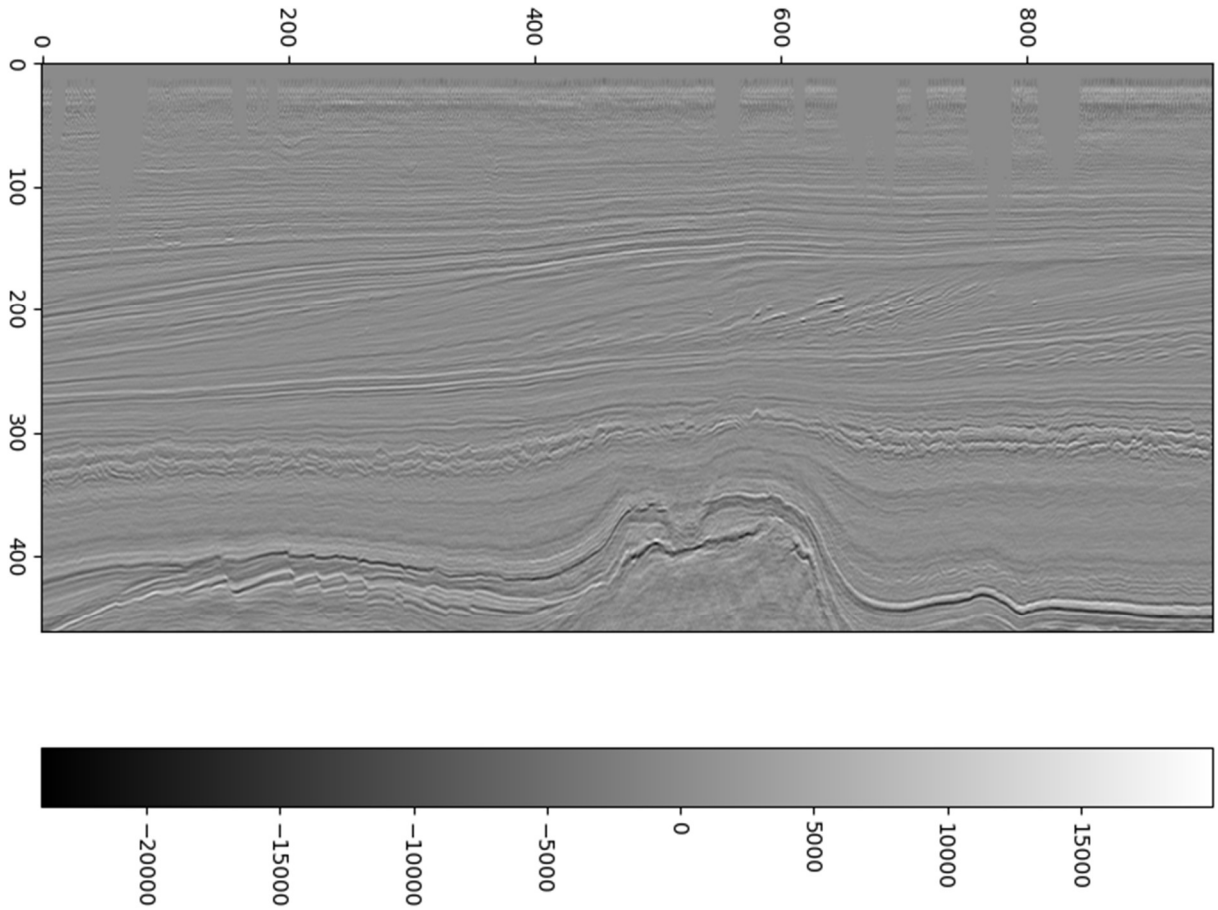
```
import os
from os import path
import numpy as np
import scipy.signal
import scipy.io
import segyio
from mayavi import mlab


def main():


    dutch = Cube('Dutch Government_F3_entire_8bit seismic.segy', 'output_cubes')
    dutch.calc_analytic()
    dutch.calc_inst_freq()
    dutch.calc_inst_wavenumb_kx()
    dutch.calc_inst_wavenumb_ky()
    dutch.calc_dip()


# Como saída visual é usada a plataforma mayavi para exibir um cubo 3D interativo
    data = dutch.seismic_cube
    source = mlab.pipeline.scalar_field(data)
    source.spacing = [1, 1, -1]
    for axis in ['x', 'z']:
        plane = mlab.pipeline.image_plane_widget(source,
                                                 plane_orientation='{}_axes'.format(axis),
                                                 slice_index=100, colormap='gray')
        plane.module_manager.scalar_lut_manager.reverse_lut = True
    mlab.outline()
    mlab.show()


class Cube:

    # Definições básicas do cubo sísmico a ser processado, incluindo nome do arquivo em disco, número de inlines, crosslines
    # e timeslices. As constantes DIL, DXL e DTS representam as granularidade de distâncias e tempo, respectivamente eixos
    # x, y e z
    # Autor: Andre Grigorio
    def __init__(self, cube_filename, output_directory):
        seismic_cube_file = segyio.open(cube_filename)
        self.seismic_cube = segyio.tools.cube(seismic_cube_file)
        self.ILINE = seismic_cube_file.ilines.size
        self.XLINE = seismic_cube_file.xlines.size
        self.TSLICE = seismic_cube_file.samples.size
        self.DIL = 25 / self.ILINE
        self.DXL = 25 / self.XLINE
        self.DTS = segyio.tools.dt(seismic_cube_file) / 1000000

        if not os.path.isdir(output_directory):
            os.mkdir(output_directory)
        self.cubo_anal_file = os.path.join(output_directory, 'cubo_anal.npy')
        self.cubo_inst_frequ_file = os.path.join(output_directory, 'cubo_inst_frequ.npy')
        self.cubo_inst_wavenumb_kx_file = os.path.join(output_directory, 'cubo_inst_wavenumb_kx.npy')
        self.cubo_inst_wavenumb_ky_file = os.path.join(output_directory, 'cubo_inst_wavenumb_ky.npy')
        self.cubo_dip_mag_file = os.path.join(output_directory, 'cubo_dip_mag.npy')
        self.cubo_dip_az_file = os.path.join(output_directory, 'cubo_dip_az.npy')
        self.cubo_inst_dip_p_file = os.path.join(output_directory, 'cubo_inst_dip_p.npy')
        self.cubo_inst_dip_q_file = os.path.join(output_directory, 'cubo_inst_dip_q.npy')

    # A partir do cubo sísmico é obtido o cubo analítico, com a componente em quadratura do sinal, usando-se a transformada
    # de Hilbert. A transformada é executada unidimensionalmente, ao longo do eixo de tempo, para cada par inline e
    # crossline
    # Autor: Andre Grigorio
    def calc_analytic(self):
        if not path.exists(self.cubo_anal_file):
            self.cubo_anal = scipy.signal.hilbert(self.seismic_cube[:, :, :self.TSLICE])
            np.save(self.cubo_anal_file, self.cubo_anal)
        else:
            self.cubo_anal = np.load(self.cubo_anal_file)

    # A partir do cubo analítico é extraída a informação de fase do sinal e subsequentemente a frequência instantânea
    # é calculada pela diferenciação ao longo do eixo de tempo. Como a diferenciação usa o método de forward difference,
    # a última matriz do cubo é uma réplica da penúltima para que se mantenha a mesma ordem do objeto tridimensional
    # Autor: Andre Grigorio
    def calc_inst_freq(self):
        if not path.exists(self.cubo_inst_frequ_file):
            self.cubo_inst_frequ = np.zeros((self.ILINE, self.XLINE, self.TSLICE))
            self.cubo_inst_frequ[:, :, :-1] = np.diff(np.angle(self.cubo_anal))
            self.cubo_inst_frequ[:, :, -1] = self.cubo_inst_frequ[:, :, -2]
            np.save(self.cubo_inst_frequ_file, self.cubo_inst_frequ)
        else:
            self.cubo_inst_frequ = np.load(self.cubo_inst_frequ_file)
```

```
  # Similarmente ao procedimento anterior, são calculados os cubos de número de onda instantâneos kx
e ky,
    # diferenciando-se ao longo dos eixos x e y, respectivamente
    # Autor: Andre Grigorio
    def calc_inst_wavenumb_kx(self):
        if not path.exists(self.cubo_inst_wavenumb_kx_file):
            self.cubo_inst_wavenumb_kx = np.zeros((self.ILINE, self.XLINE, self.TSLICE))
            self.cubo_inst_wavenumb_kx[:-1, :, :] = np.diff(np.angle(self.cubo_anal), axis=-3)
            self.cubo_inst_wavenumb_kx[-1, :, :] = self.cubo_inst_wavenumb_kx[-2, :, :]
            np.save(self.cubo_inst_wavenumb_kx_file, self.cubo_inst_wavenumb_kx)
        else:
            self.cubo_inst_wavenumb_kx = np.load(self.cubo_inst_wavenumb_kx_file)

    def calc_inst_wavenumb_ky(self):
        if not path.exists(self.cubo_inst_wavenumb_ky_file):
            self.cubo_inst_wavenumb_ky = np.zeros((self.ILINE, self.XLINE, self.TSLICE))
            self.cubo_inst_wavenumb_ky[:, :-1, :] = np.diff(np.angle(self.cubo_anal), axis=-2)
            self.cubo_inst_wavenumb_ky[:, -1, :] = self.cubo_inst_wavenumb_ky[:, -2, :]
            np.save(self.cubo_inst_wavenumb_ky_file, self.cubo_inst_wavenumb_ky)
        else:
            self.cubo_inst_wavenumb_ky = np.load(self.cubo_inst_wavenumb_ky_file)

    #  Com os cubos de frequência e número de ondas instantâneos, são calculados os cubos de dip
aparentes, e, finalmente,
    #  os cubos de dip magnitude e dip azimuth
    # Autor: Andre Grigorio
    def calc_dip(self):
        if (not path.exists(self.cubo_dip_mag_file)) or (not path.exists(self.cubo_dip_az_file)) or
(
        not path.exists(self.cubo_inst_dip_p_file)) or (not path.exists(self.cubo_inst_dip_q_file)):
            self.cubo_inst_dip_p = self.cubo_inst_wavenumb_kx / self.cubo_inst_frequ
            self.cubo_inst_dip_q = self.cubo_inst_wavenumb_ky / self.cubo_inst_frequ
            self.cubo_dip_mag = np.sqrt(np.square(self.cubo_inst_dip_p) +
np.square(self.cubo_inst_dip_q))
            self.cubo_dip_az = np.arctan2(self.cubo_inst_dip_q, self.cubo_inst_dip_p)
            np.save(self.cubo_inst_dip_p_file, self.cubo_inst_dip_p)
            np.save(self.cubo_inst_dip_q_file, self.cubo_inst_dip_q)
            np.save(self.cubo_dip_mag_file, self.cubo_dip_mag)
            np.save(self.cubo_dip_az_file, self.cubo_dip_az)
        else:
            self.cubo_inst_dip_p = np.load(self.cubo_inst_dip_p_file)
            self.cubo_inst_dip_q = np.load(self.cubo_inst_dip_q_file)
            self.cubo_dip_mag = np.load(self.cubo_dip_mag_file)
            self.cubo_dip_az = np.load(self.cubo_dip_az_file)

if __name__ == "__main__":
    main()
```

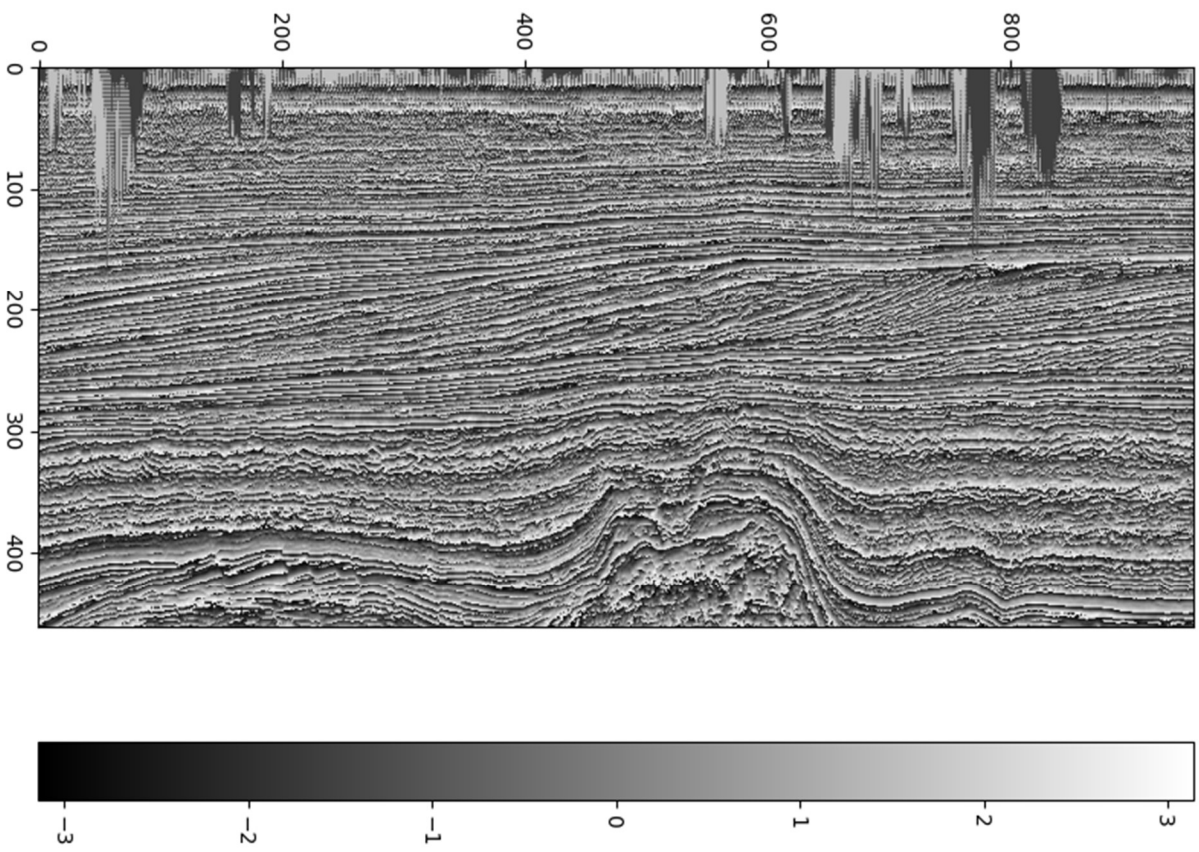
D. Roteiro de teste efetuado

Para teste do programa foram utilizados os dados sísmicos do conhecido *Dutch F3*. Nas imagens a seguir pode ser observada a evolução do algoritmo:

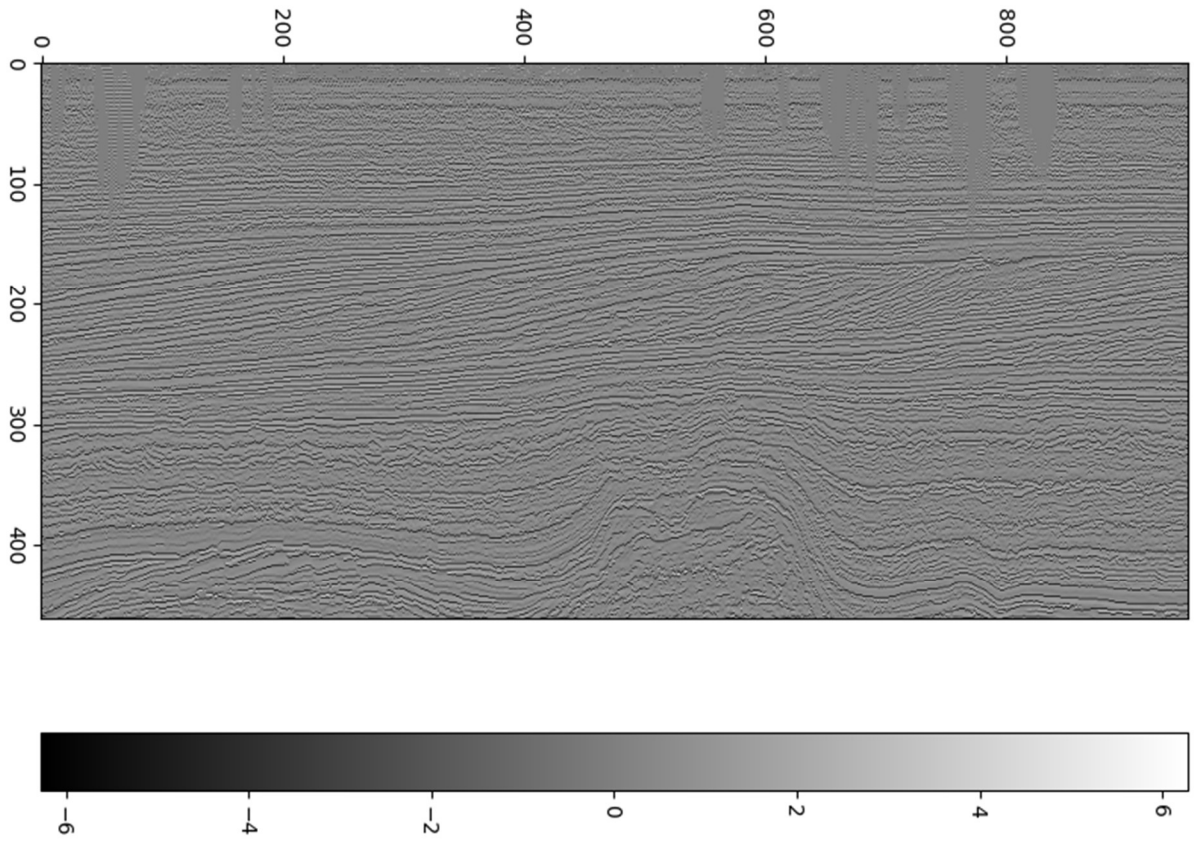
Dados sísmicos não tratados



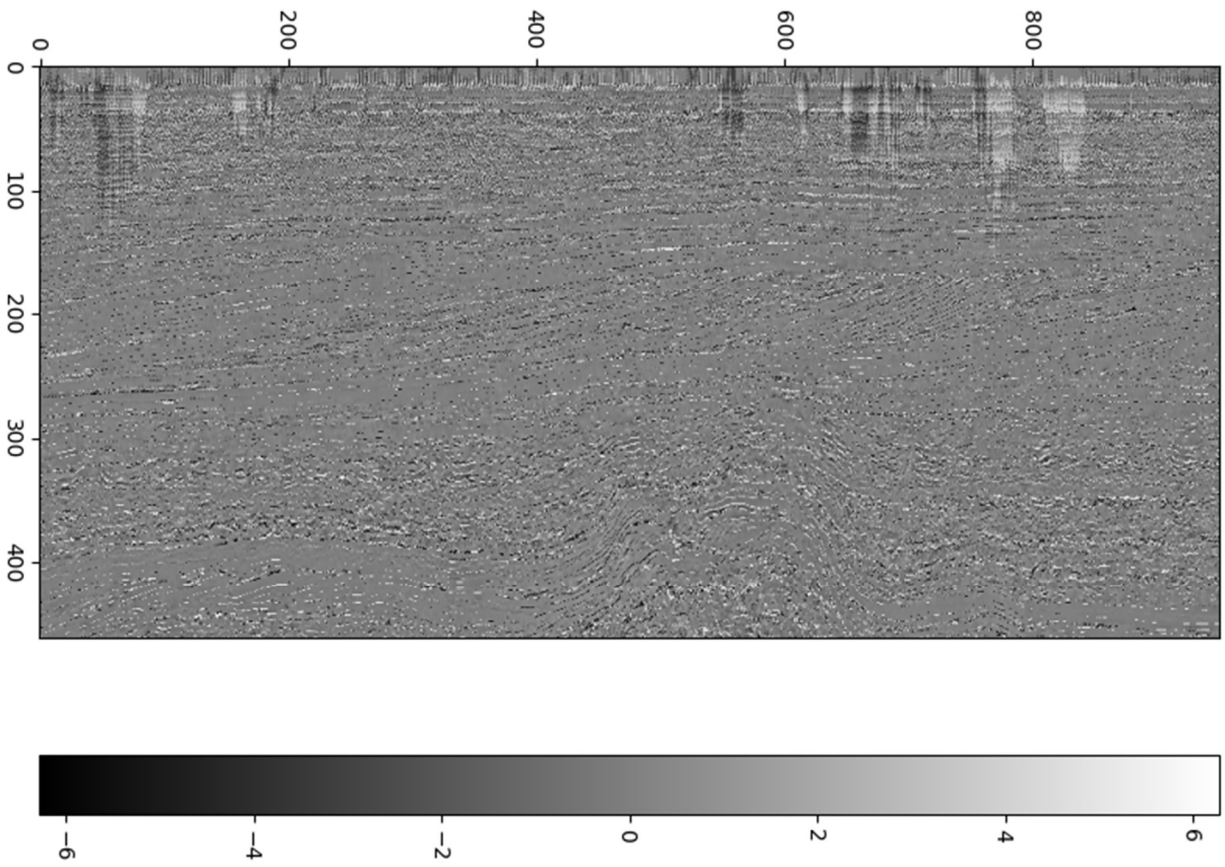
Fase instantânea



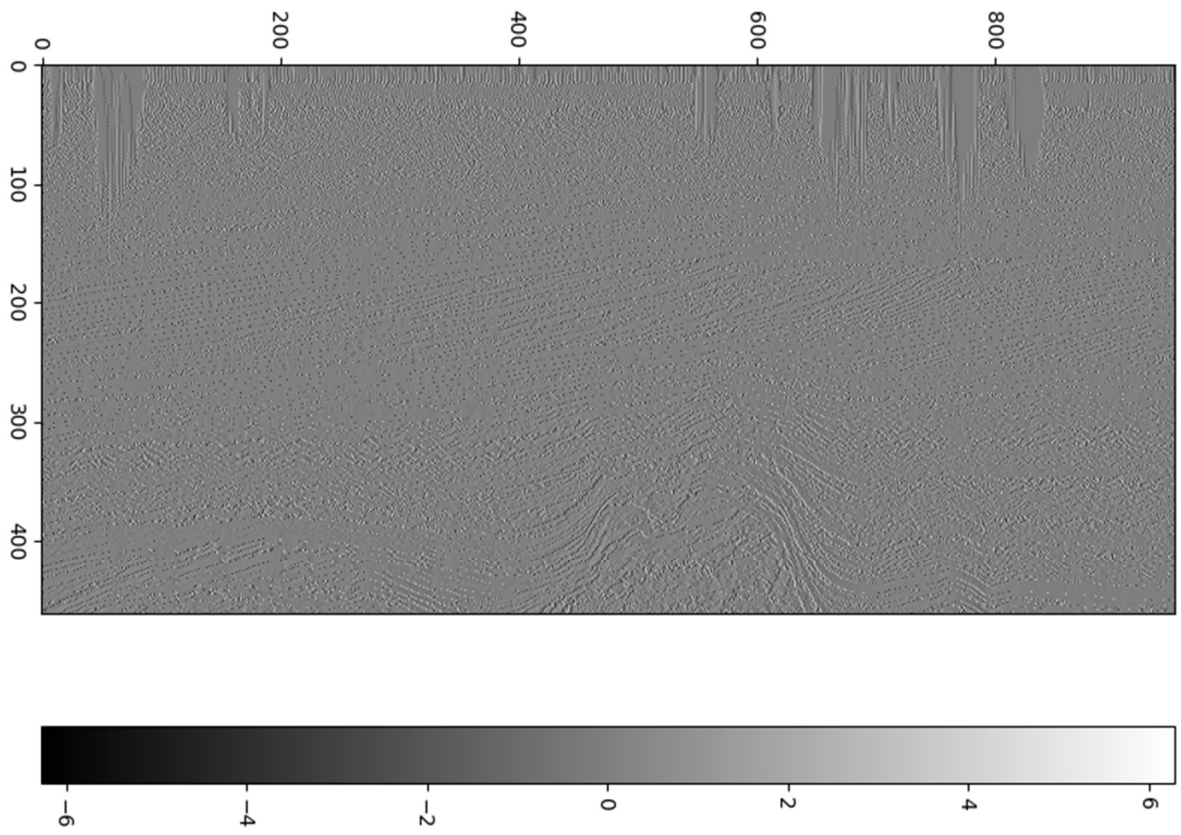
Frequência instantânea

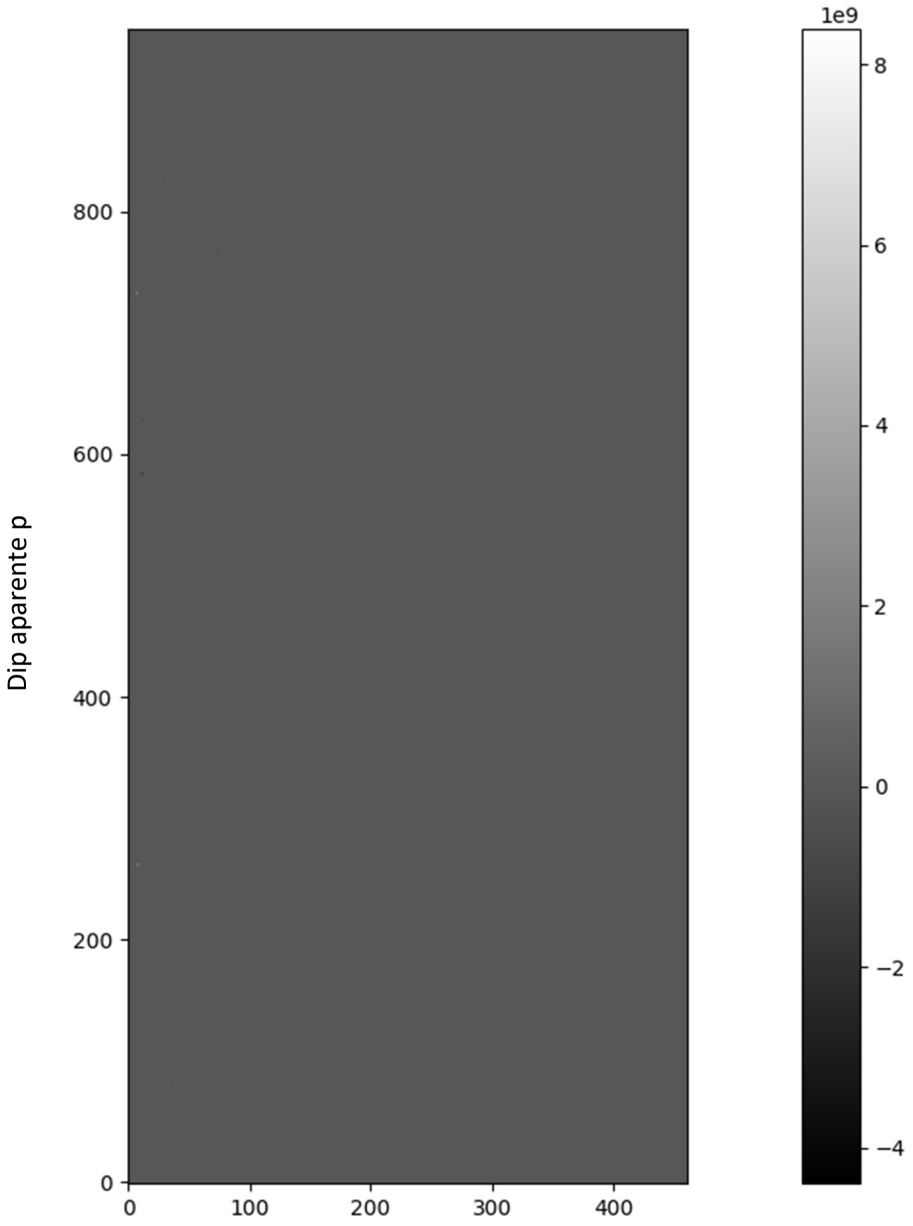


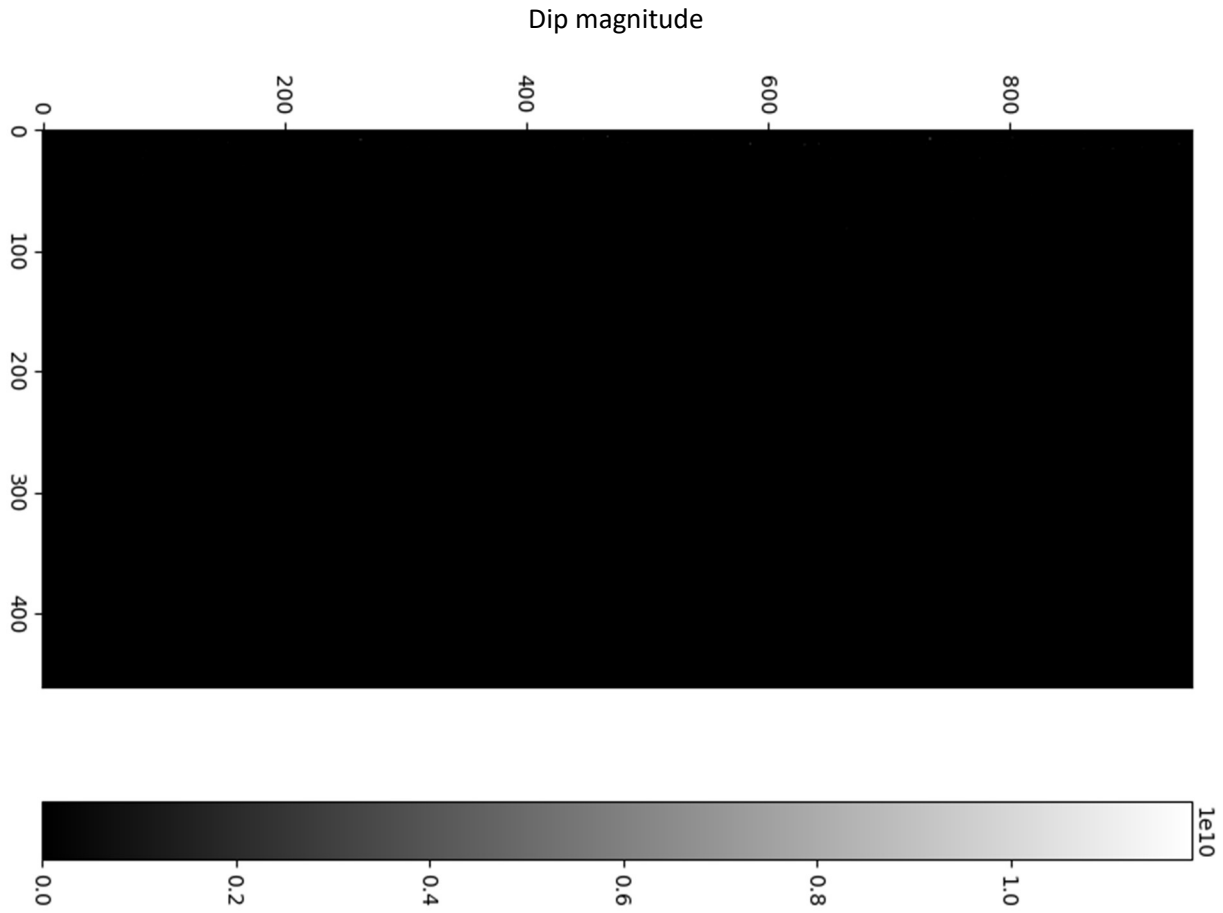
Número de onda instantâneo kx



Número de onda instantâneo ky




Dip aparente p
0
200
400
600
800
0
100
200
300
400
1e9
8
6
4
2
0
−2
−4



E. Documentação para o usuário

A utilização do programa consiste basicamente nos seguintes passos:

- Colocação do arquivo de dados sísmicos no formato SEG-Y no mesmo diretório do arquivo de código fonte do programa, *main.py*, escrito em Python;
- Alteração da linha 13 do código, no arquivo *main.py*, alterando os parâmetros relativos ao nome do arquivo de dados sísmicos, como descrito no passo anterior, e diretório de armazenamento dos cubos de saída, em negrito no exemplo abaixo.
  Ex.: *dutch = Cube('**Dutch Government_F3_entire_8bit seismic.segy**', '**output_cubes**')*

Note que, para fins didáticos, e para acelerar execuções subsequentes, os cubos intermediários são mantidos em disco, de forma que é necessário reservar espaço equivalente a cerca de dezesseis vezes o tamanho do arquivo de dados sísmicos original.

F. Referências

Chopra, S., and K. J. Marfurt, 2014, Seismic Attribute for Prospect Identification and Reservoir Characterization.

Marfurt, K. J., 2006, Robust estimate of 3D reflector dip and azimuth.

Barnes, A. E., 2007, A tutorial on complex seismic trace analysis.