#NBVAL\_IGNORE\_OUTPUT  
# Adding ignore due to (probably an np notebook magic) bug  
import numpy as np  
import matplotlib.pyplot as plt  
%matplotlib inline  
import pandas as pd

from devito import \*

#NBVAL\_IGNORE\_OUTPUT  
from examples.seismic import Model, plot\_velocity

from examples.seismic import Model, plot\_velocity

import numpy as np  
import pandas as pd  
  
def capas\_pre\_devito(vel\_original, no\_capas, shape\_z):  
 '''  
 Función para calcular la matriz de capas de entrada para hacer el campo de velocidades para devito  
 La entrada vel\_original debe ser un dataframe con las siguientes columnas ['DEPTH', 'ACOUSTIC\_INTERVAL\_VEL', 'DELTA\_DEPTH']  
 no\_capas: Las capas que deseo para mi modelo  
 Shape\_z= la dimención de  
 '''  
   
   
 depth\_max = vel\_original['DEPTH'].max()  
 intervalo = round(depth\_max / (no\_capas - 1), 2)  
   
 new\_depth = np.arange(0, depth\_max, intervalo)  
 new\_df = pd.DataFrame({'DEPTH': new\_depth})  
   
 # Interpolación de los valores de velocidad acústica  
 new\_df['ACOUSTIC\_INTERVAL\_VEL'] = np.interp(new\_df['DEPTH'], vel\_original['DEPTH'], vel\_original['ACOUSTIC\_INTERVAL\_VEL'])  
   
 # Cálculo de CAPA\_INFERIOR  
 new\_df['CAPA\_INFERIOR'] = new\_df['DEPTH'] + intervalo / 2  
 new\_df.iloc[-1, new\_df.columns.get\_loc('CAPA\_INFERIOR')] = new\_df['DEPTH'].iloc[-1]  
   
 # Cálculo de FILA\_INF  
 new\_df['FILA\_INF'] = np.round(new\_df['CAPA\_INFERIOR'] \* (shape\_z - 1) / depth\_max + 1).astype(int)  
   
 # Creación de las capas con FILA\_INF y ACOUSTIC\_INTERVAL\_VEL  
 capas = list(zip(new\_df['FILA\_INF'], new\_df['ACOUSTIC\_INTERVAL\_VEL'] / 1000))  
   
 return capas

import numpy as np  
import pandas as pd  
  
def capas\_pre\_devito(vel\_original, no\_capas, shape\_z):  
 '''  
 Función para calcular la matriz de capas de entrada para hacer el campo de velocidades para devito  
 La entrada vel\_original debe ser un dataframe con las siguientes columnas ['DEPTH', 'ACOUSTIC\_INTERVAL\_VEL', 'DELTA\_DEPTH']  
 no\_capas: Las capas que deseo para mi modelo  
 Shape\_z= la dimención de  
 '''  
   
   
 depth\_max = vel\_original['DEPTH'].max()  
 intervalo = round(depth\_max / (no\_capas - 1), 2)  
 vel\_final= vel\_original['ACOUSTIC\_INTERVAL\_VEL'].iloc[-1]  
   
 new\_depth = np.arange(0, depth\_max, intervalo)  
 new\_df = pd.DataFrame({'DEPTH': new\_depth})  
   
 # Interpolación de los valores de velocidad acústica  
 new\_df['ACOUSTIC\_INTERVAL\_VEL'] = np.interp(new\_df['DEPTH'], vel\_original['DEPTH'], vel\_original['ACOUSTIC\_INTERVAL\_VEL'])  
   
 # Cálculo de CAPA\_INFERIOR  
 new\_df['CAPA\_INFERIOR'] = new\_df['DEPTH'] + intervalo / 2  
 new\_df.iloc[-1, new\_df.columns.get\_loc('CAPA\_INFERIOR')] = new\_df['DEPTH'].iloc[-1]  
   
 # Cálculo de FILA\_INF  
 new\_df['FILA\_INF'] = np.round(new\_df['CAPA\_INFERIOR'] \* (shape\_z - 1) / depth\_max + 1).astype(int)  
 if len(new\_df) !=no\_capas:  
   
   
 last\_row = new\_df.iloc[-1]  
 new\_row = pd.DataFrame([last\_row], columns=new\_df.columns)  
 new\_row['FILA\_INF']=shape\_z  
 new\_row['ACOUSTIC\_INTERVAL\_VEL']=vel\_final  
 new\_row['DEPTH']=depth\_max  
 new\_row['CAPA\_INFERIOR']=depth\_max  
 new\_df = pd.concat([new\_df, new\_row], ignore\_index=True)  
   
   
   
 # Creación de las capas con FILA\_INF y ACOUSTIC\_INTERVAL\_VEL  
 capas = list(zip(new\_df['FILA\_INF'], new\_df['ACOUSTIC\_INTERVAL\_VEL'] / 1000))  
   
 #return new\_df  
 return capas

def campo\_velocidades\_devito(shape, capas):  
 """  
 Genera un campo de velocidades con un número variable de capas.  
  
 Parameters:  
 - shape: Tuple que define la forma de la matriz (número de filas, número de columnas).  
 - capas: Lista de tuplas, donde cada tupla contiene dos valores:  
 (profundidad\_final, velocidad), siendo profundidad\_final la columna final de la capa  
 y velocidad la velocidad asignada a esa capa.  
  
 Returns:  
 - Vel\_Devito: Matriz numpy con el campo de velocidades.  
 """  
 Vel\_Devito = np.empty(shape, dtype=np.float32)  
   
 # Inicializar el índice inicial en 0  
 indice\_inicial = 0  
   
 for profundidad\_final, velocidad in capas:  
 Vel\_Devito[:, indice\_inicial:profundidad\_final] = velocidad  
 indice\_inicial = profundidad\_final  
   
 return Vel\_Devito

def print\_registro(model,cant\_capas):  
   
 #NBVAL\_IGNORE\_OUTPUT  
 from examples.seismic import RickerSource  
 f0 = 0.010 # Source peak frequency is 10Hz (0.010 kHz)  
 src = RickerSource(name='src', grid=model.grid, f0=f0,  
 npoint=1, time\_range=time\_range)  
  
 # First, position source centrally in all dimensions, then set depth  
 src.coordinates.data[0, :] = np.array(model.domain\_size) \* .5  
 src.coordinates.data[0, -1] = 20. # Depth is 20m  
  
 # We can plot the time signature to see the wavelet  
 #src.show()  
  
  
  
 f0 = 0.010 # Source peak frequency is 10Hz (0.010 kHz)  
 src = RickerSource(name='src', grid=model.grid, f0=f0,  
 npoint=1, time\_range=time\_range)  
  
 # First, position source centrally in all dimensions, then set depth  
 src.coordinates.data[0, :] = np.array(model.domain\_size) \* .5  
 src.coordinates.data[0, -1] = 20. # Depth is 20m  
  
  
 # We can plot the time signature to see the wavelet  
 #src.show()  
  
  
  
 f0 = 0.010 # Source peak frequency is 10Hz (0.010 kHz)  
 src = RickerSource(name='src', grid=model.grid, f0=f0,  
 npoint=1, time\_range=time\_range)  
  
 # First, position source centrally in all dimensions, then set depth  
 src.coordinates.data[0, :] = np.array(model.domain\_size) \* .5  
 src.coordinates.data[0, -1] = 20. # Depth is 20m  
  
 # We can plot the time signature to see the wavelet  
 #src.show()  
  
  
 #NBVAL\_IGNORE\_OUTPUT  
 from examples.seismic import Receiver  
  
 # Create symbol for 101 receivers  
 rec = Receiver(name='rec', grid=model.grid, npoint=101, time\_range=time\_range)  
  
 # Prescribe even spacing for receivers along the x-axis  
 rec.coordinates.data[:, 0] = np.linspace(0, model.domain\_size[0], num=101)  
 rec.coordinates.data[:, 1] = 20. # Depth is 20m  
  
 # We can now show the source and receivers within our domain:  
 # Red dot: Source location  
 # Green dots: Receiver locations (every 4th point)  
 #plot\_velocity(model, source=src.coordinates.data,  
 # receiver=rec.coordinates.data[::4, :])  
  
  
  
 from devito import TimeFunction  
 # Define the wavefield with the size of the model and the time dimension  
 u = TimeFunction(name="u", grid=model.grid, time\_order=2, space\_order=2)  
  
 # We can now write the PDE  
 pde = model.m \* u.dt2 - u.laplace + model.damp \* u.dt  
  
 # The PDE representation is as on paper  
 #pde  
  
  
 # This discrete PDE can be solved in a time-marching way updating u(t+dt) from the previous time step  
 # Devito as a shortcut for u(t+dt) which is u.forward. We can then rewrite the PDE as   
 # a time marching updating equation known as a stencil using customized SymPy functions  
 from devito import Eq, solve  
  
 stencil = Eq(u.forward, solve(pde, u.forward))  
 #stencil  
  
  
  
 # Finally we define the source injection and receiver read function to generate the corresponding code  
 src\_term = src.inject(field=u.forward, expr=src \* dt\*\*2 / model.m)  
  
 # Create interpolation expression for receivers  
 rec\_term = rec.interpolate(expr=u.forward)  
  
  
 #NBVAL\_IGNORE\_OUTPUT  
 from devito import Operator  
  
 op = Operator([stencil] + src\_term + rec\_term, subs=model.spacing\_map)  
  
  
 #NBVAL\_IGNORE\_OUTPUT  
 op(time=time\_range.num-1, dt=model.critical\_dt)  
  
  
 #NBVAL\_IGNORE\_OUTPUT  
 from examples.seismic import plot\_shotrecord  
   
   
 fig, ax = plt.subplots(figsize=(4, 2))  
  
 # Configurar el texto  
 ax.text(0.5, 0.5, f'Registro correspondiente a un modelo de {cant\_capas} capas', fontsize=30, ha='center', va='center', color='blue')  
  
 # Configurar el estilo del gráfico  
 ax.set\_xlim(0, 1)  
 ax.set\_ylim(0, 1)  
 ax.axis('off') # Ocultar ejes  
  
 # Mostrar el gráfico  
 plt.show()  
   
   
  
 plot\_shotrecord(rec.data, model, t0, tn)

Vel\_Pozo= pd.read\_csv('vel\_pozo\_edit.csv')

depth\_max=Vel\_Pozo.DEPTH.max()  
depth\_max

np.float64(2506.17228)

# Define a physical size  
shape = (101, 101) # Number of grid point (nx, nz)  
dz=depth\_max/(shape[1]-1)  
dz

np.float64(25.0617228)

dx=dz  
ncapas=2

capas=capas\_pre\_devito(Vel\_Pozo,ncapas,shape[1])  
capas

[(51, 2.1070824), (101, 4.42668286624203)]

# Define a physical size  
  
spacing = (dx, dz) # Grid spacing in m. The domain size is now 1km by 1km  
origin = (0., 0.) # What is the location of the top left corner. This is necessary to define  
# the absolute location of the source and receivers

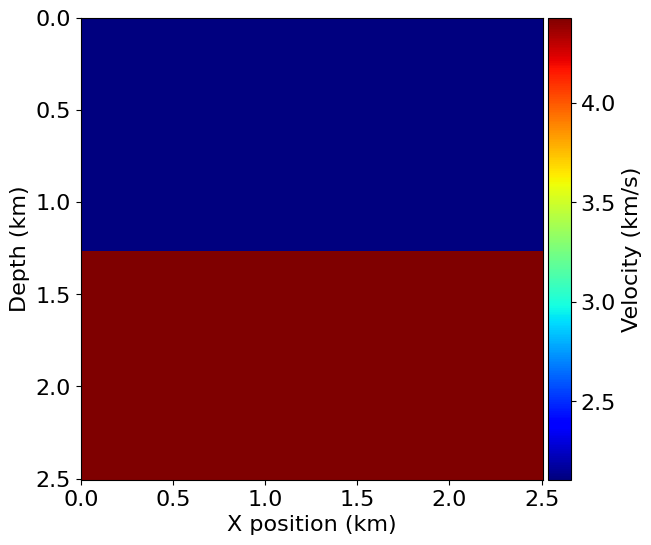
capas=capas\_pre\_devito(Vel\_Pozo,ncapas,shape[1])  
capas

[(51, 2.1070824), (101, 4.42668286624203)]

v=v = campo\_velocidades\_devito(shape, capas)

# With the velocity and model size defined, we can create the seismic model that  
# encapsulates this properties. We also define the size of the absorbing layer as 10 grid points  
model = Model(vp=v, origin=origin, shape=shape, spacing=spacing,  
 space\_order=2, nbl=10, bcs="damp")  
  
plot\_velocity(model)

Operator `initdamp` ran in 0.01 s

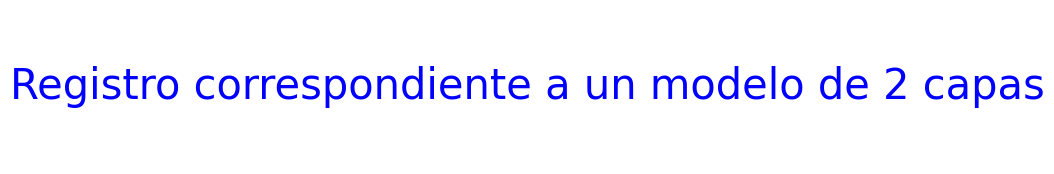


png

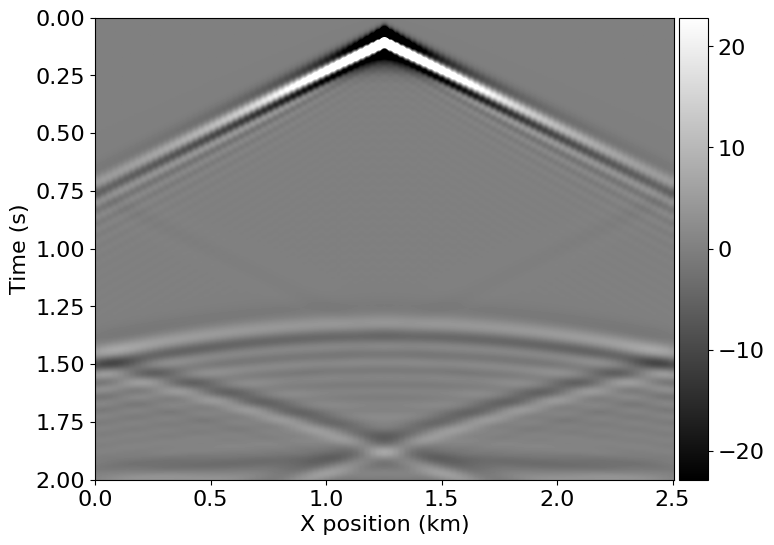
from examples.seismic import TimeAxis  
  
t0 = 0. # Simulation starts a t=0  
tn = 2000. # Simulation last 1 second (1000 ms)  
dt = model.critical\_dt # Time step from model grid spacing  
  
time\_range = TimeAxis(start=t0, stop=tn, step=dt)

print\_registro(model,2)

Operator `Kernel` ran in 0.01 s



png

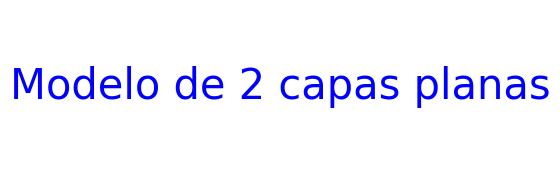


png

# Recurrencia

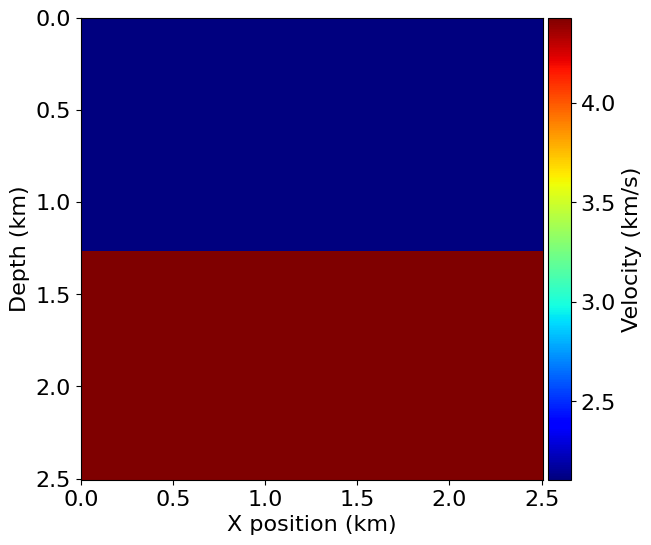
Vel\_Pozo= pd.read\_csv('vel\_pozo\_edit.csv')  
depth\_max=Vel\_Pozo.DEPTH.max()  
depth\_max  
shape = (101, 101) # Number of grid point (nx, nz)  
dz=depth\_max/(shape[1]-1)  
dz  
# Define a physical size  
dx=dz  
  
  
spacing = (dx, dz) # Grid spacing in m. The domain size is now 1km by 1km  
origin = (0., 0.) # What is the location of the top left corner. This is necessary to define  
# the absolute location of the source and receivers

Muticapa=[2,4,5,8,10,30,40,80]  
for ncapas in Muticapa:  
   
 capas=capas\_pre\_devito(Vel\_Pozo,ncapas,shape[1])  
   
 v=v = campo\_velocidades\_devito(shape, capas)  
 # With the velocity and model size defined, we can create the seismic model that  
 # encapsulates this properties. We also define the size of the absorbing layer as 10 grid points  
   
 fig, ax = plt.subplots(figsize=(2, 2)) # Ajusta el tamaño de la figura aquí  
  
 # Configurar el texto  
 ax.text(0.5, 0.5, f'Modelo de {ncapas} capas planas', fontsize=30, ha='center', va='center', color='blue')  
  
 # Configurar el estilo del gráfico  
 ax.set\_xlim(0, 1)  
 ax.set\_ylim(0, 1)  
 ax.axis('off') # Ocultar ejes  
  
 # Mostrar el gráfico  
 plt.show()  
   
   
   
   
   
 model = Model(vp=v, origin=origin, shape=shape, spacing=spacing,  
 space\_order=2, nbl=10, bcs="damp")  
  
 plot\_velocity(model)

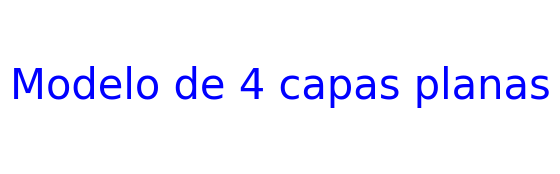


png

Operator `initdamp` ran in 0.01 s

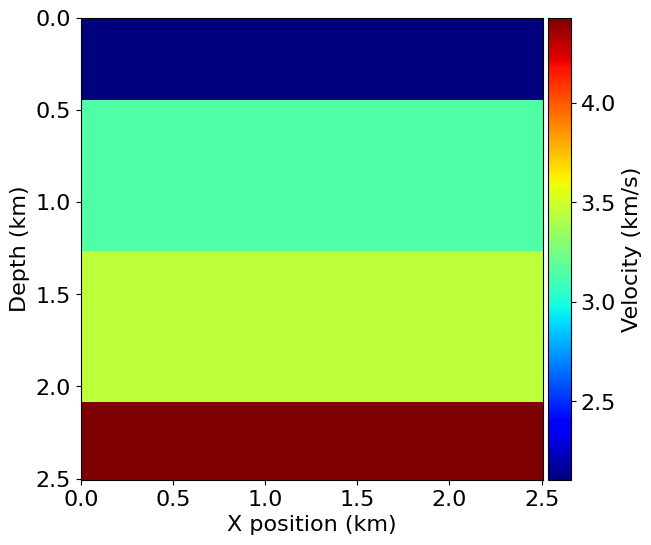


png

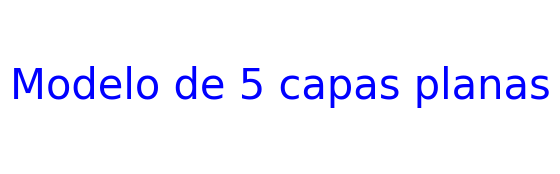


png

Operator `initdamp` ran in 0.01 s

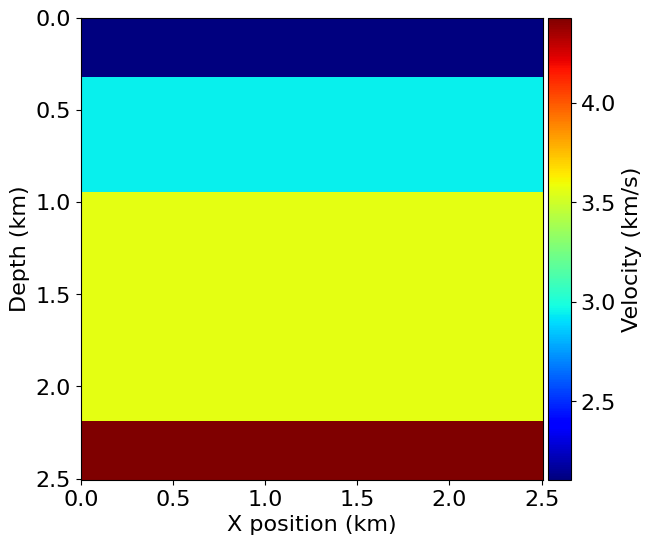


png



png

Operator `initdamp` ran in 0.01 s

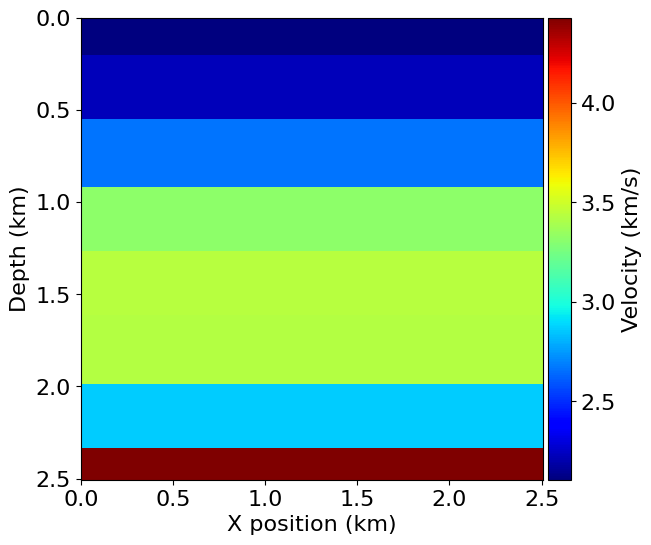


png



png

Operator `initdamp` ran in 0.01 s

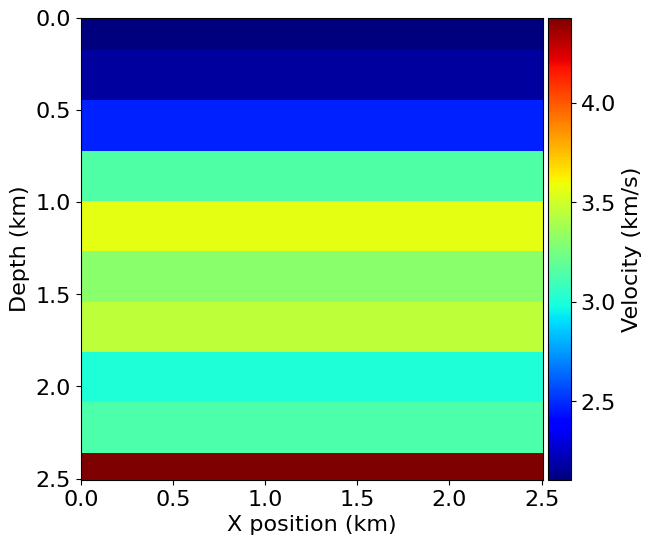


png



png

Operator `initdamp` ran in 0.01 s

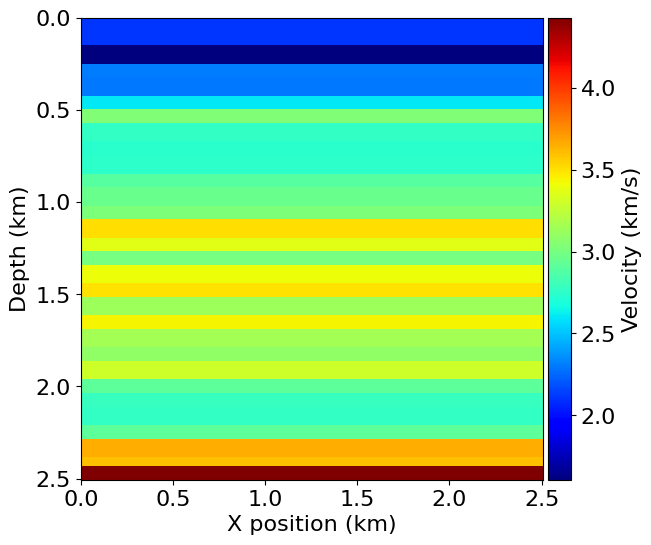


png



png

Operator `initdamp` ran in 0.01 s

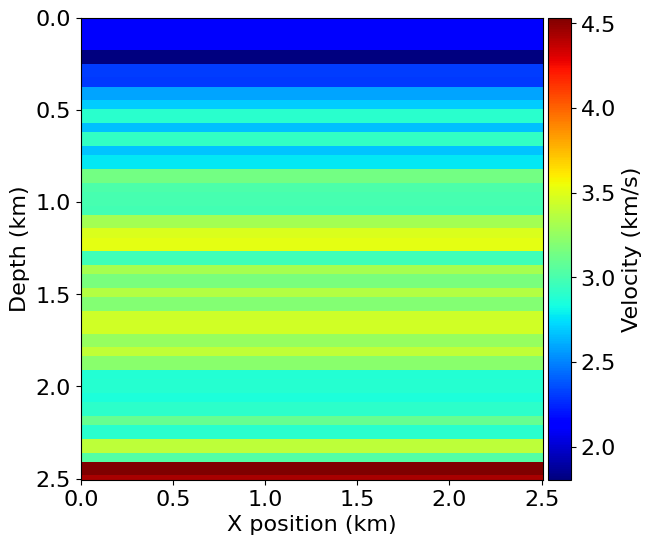


png



png

Operator `initdamp` ran in 0.01 s

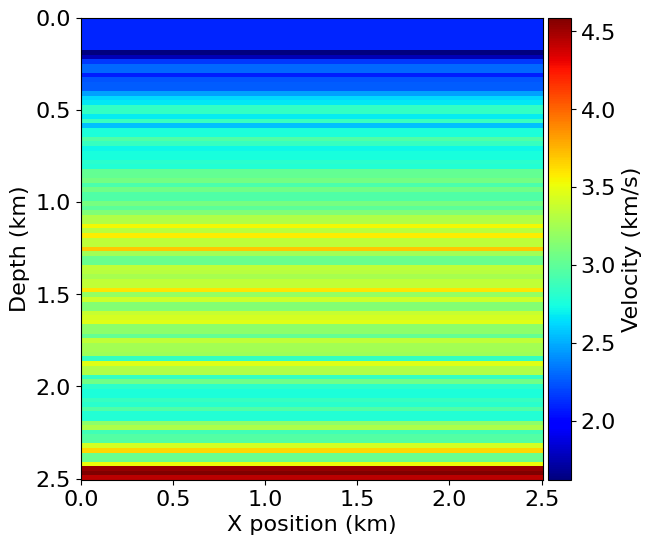


png



png

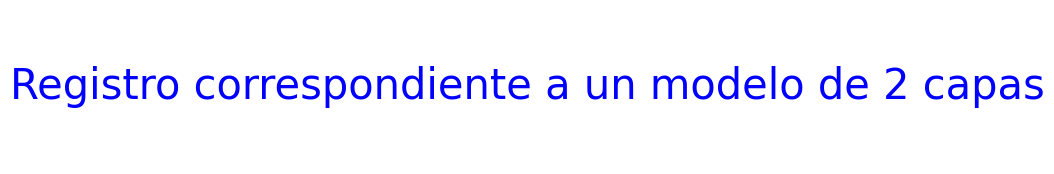
Operator `initdamp` ran in 0.01 s



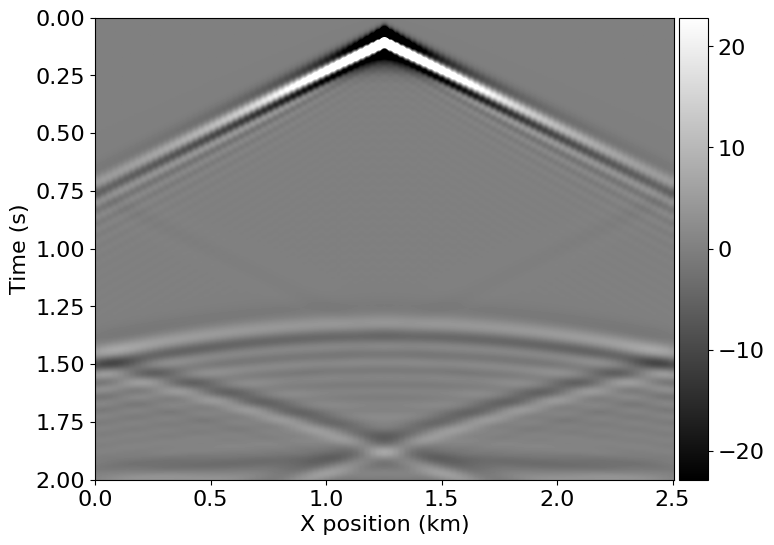
png

for ncapas in Muticapa:  
   
 capas=capas\_pre\_devito(Vel\_Pozo,ncapas,shape[1])  
   
 v=v = campo\_velocidades\_devito(shape, capas)  
 # With the velocity and model size defined, we can create the seismic model that  
 # encapsulates this properties. We also define the size of the absorbing layer as 10 grid points  
   
   
   
   
   
   
 model = Model(vp=v, origin=origin, shape=shape, spacing=spacing,  
 space\_order=2, nbl=10, bcs="damp")  
  
  
  
 from examples.seismic import TimeAxis  
  
 t0 = 0. # Simulation starts a t=0  
 tn = 2000. # Simulation last 1 second (1000 ms)  
 dt = model.critical\_dt # Time step from model grid spacing  
  
 time\_range = TimeAxis(start=t0, stop=tn, step=dt)  
 print\_registro(model,ncapas)

Operator `initdamp` ran in 0.01 s  
Operator `Kernel` ran in 0.01 s

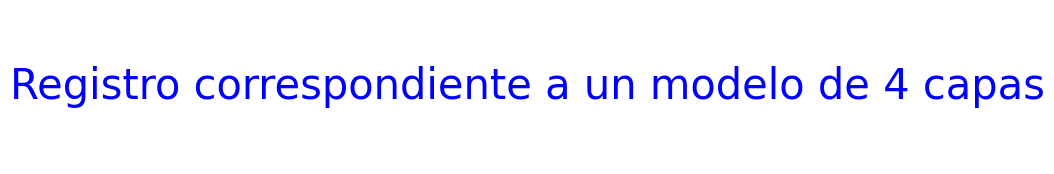


png

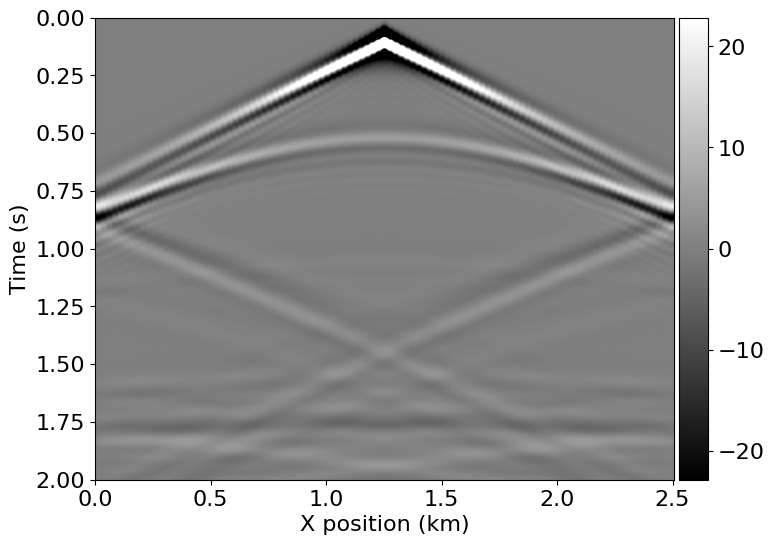


png

Operator `initdamp` ran in 0.01 s  
Operator `Kernel` ran in 0.01 s

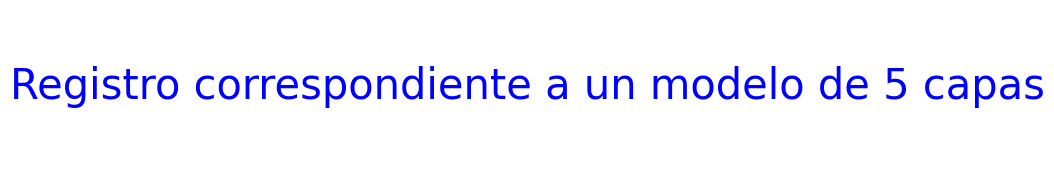


png

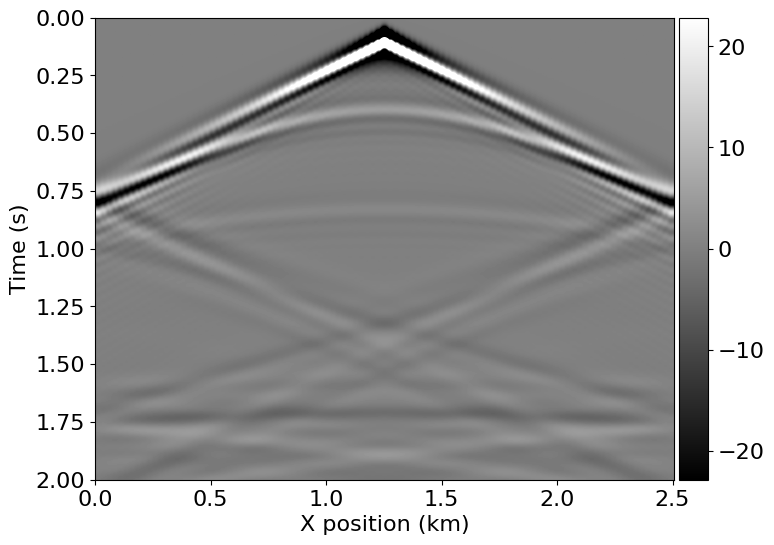


png

Operator `initdamp` ran in 0.01 s  
Operator `Kernel` ran in 0.01 s

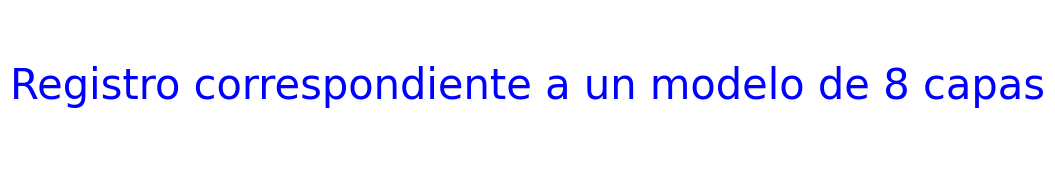


png

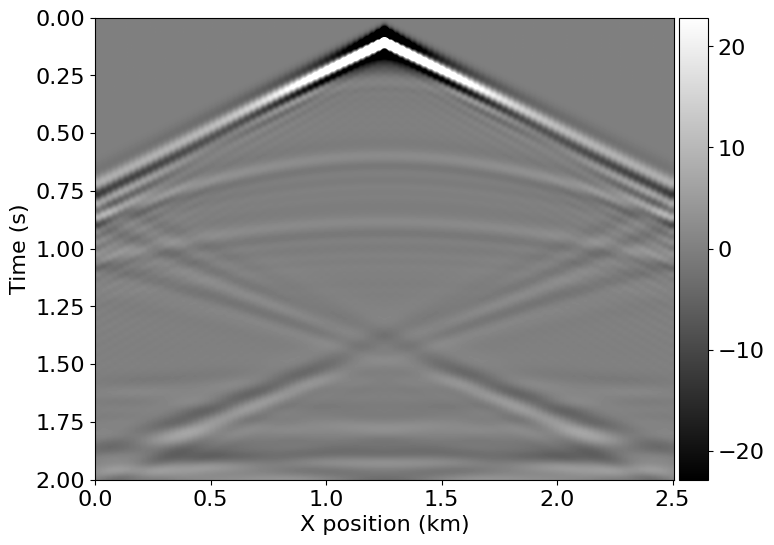


png

Operator `initdamp` ran in 0.01 s  
Operator `Kernel` ran in 0.01 s

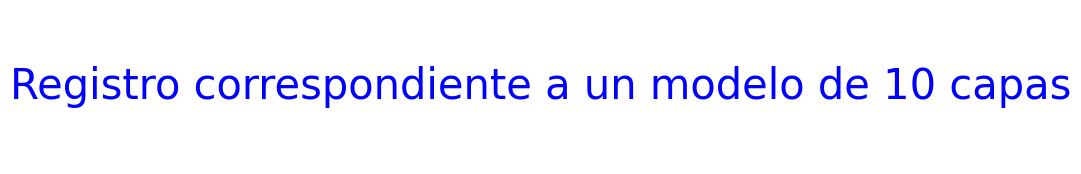


png

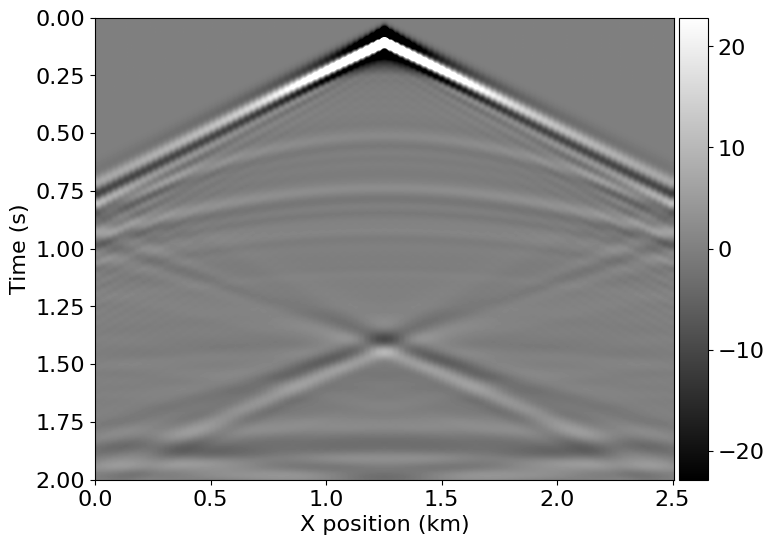


png

Operator `initdamp` ran in 0.01 s  
Operator `Kernel` ran in 0.01 s



png

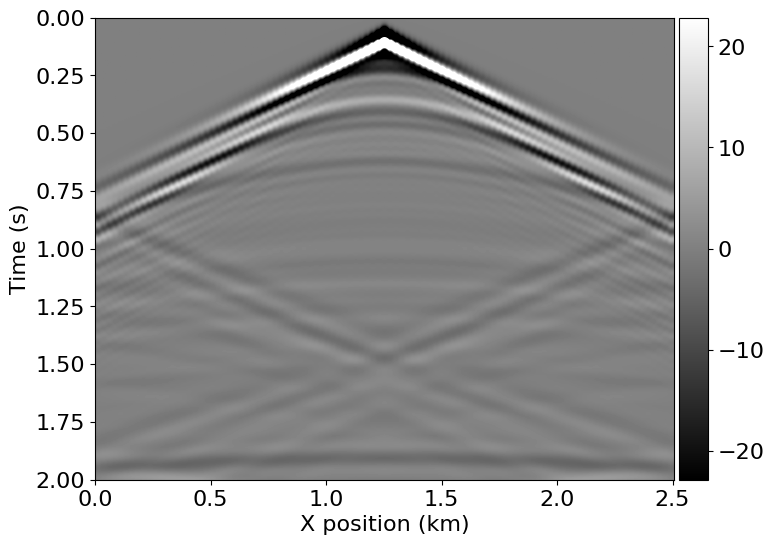


png

Operator `initdamp` ran in 0.01 s  
Operator `Kernel` ran in 0.01 s



png

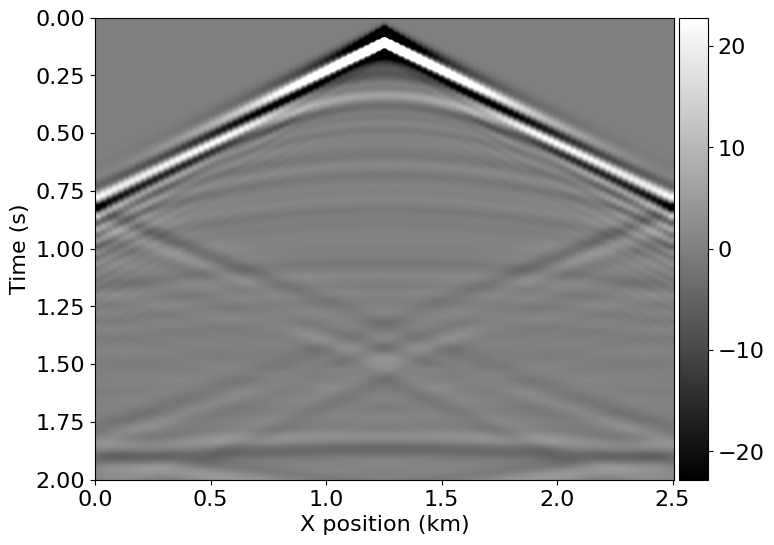


png

Operator `initdamp` ran in 0.01 s  
Operator `Kernel` ran in 0.01 s



png

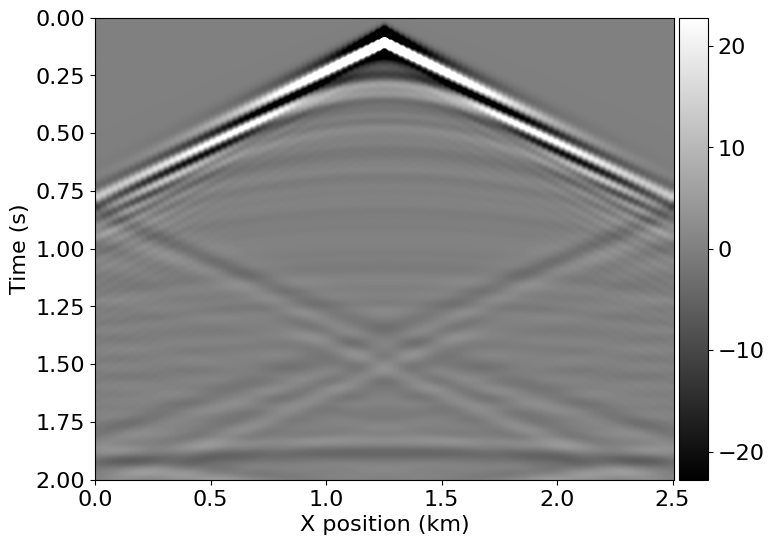


png

Operator `initdamp` ran in 0.01 s  
Operator `Kernel` ran in 0.01 s



png



png