GRMOT - strong GRound MOTion simulations

Warning : the documentation of the library is still under preparation.

General description

GRMOT generates simulated seismograms, including displacement, velocity, and accelerations, using the 3D discrete wavenumber representation method.
The code is designed for parallel execution, ensuring efficiency in large-scale simulations.
The library's core is implemented in Rust for high performance,
while a Python interface provides a user-friendly experience for researchers and engineers.

Citation

DOI 10.5281/zenodo.14900910

If you use this project in your research or work, please cite it as follows:

```
@software{Smaragdakis_GrMot_2025,
author = {Smaragdakis, Costas},
doi = {https://doi.org/10.5281/zenodo.14900909},
month = feb,
title = {{GrMot}},
url = {https://github.com/kesmarag/grmot},
version = {v0.9.0},
year = {2025}
}
```

Installation

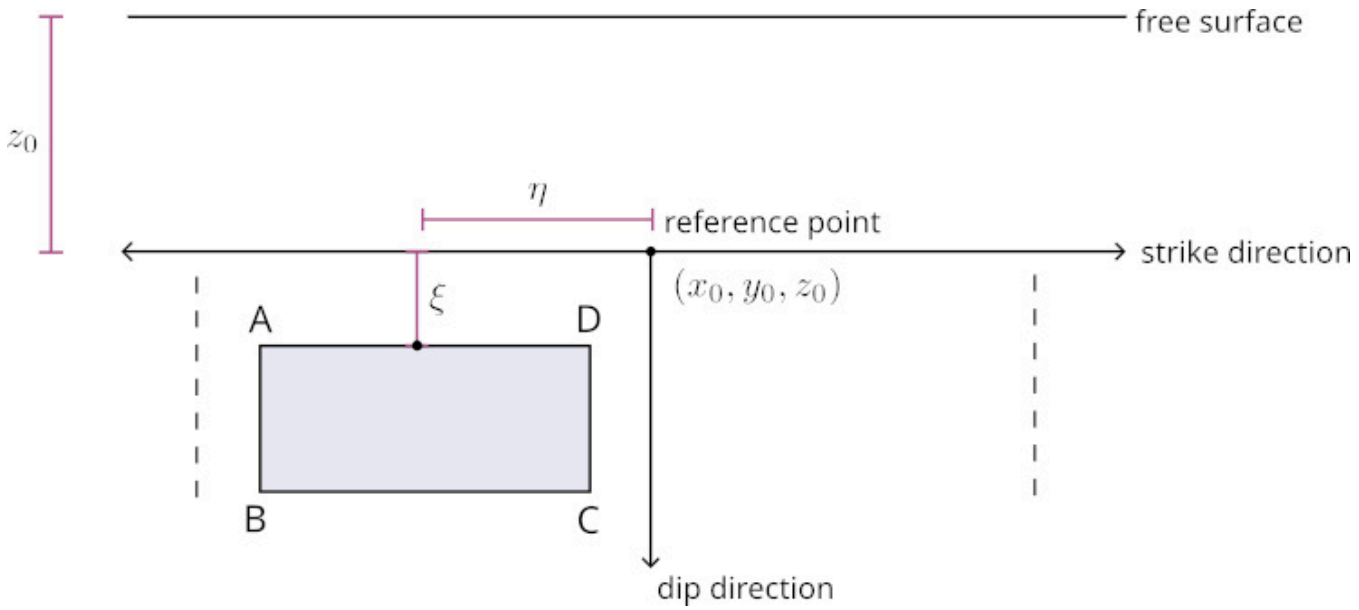
The GrMot library supports GNU/Linux and requires Python 3.8 or later. To install the appropriate version for your Python environment, run:

```
pip install $(python -c "import sys; version=f'{sys.version_info.major}{sys.version_info.minor}';
print(f'https://github.com/kesmarag/grmot/raw/main/target/wheels/grmot-0.9.0-cp{version}-cp{version}-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl')")
```

Using GRMOT

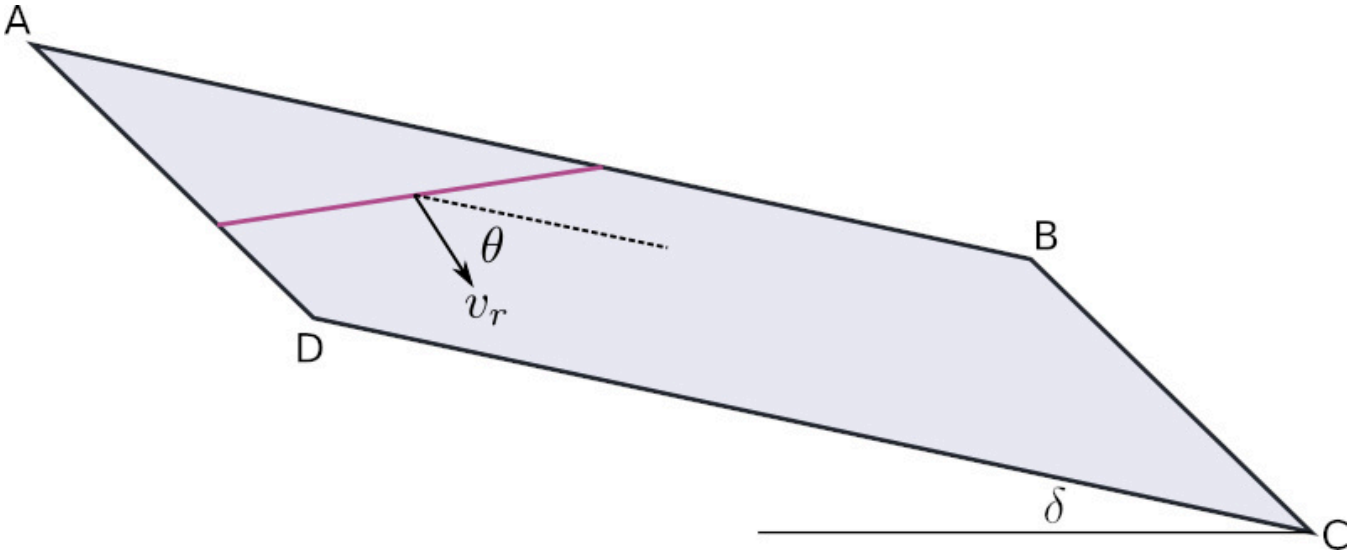
First, we create a reference sub-plane. The following parameters determine this sub-plane

- The top centre point of the fault (north ((x_0) in km), east ((y_0) in km), and depth ((z_0) in km)) with respect to a general reference point.
- The dip, strike and rake angles.

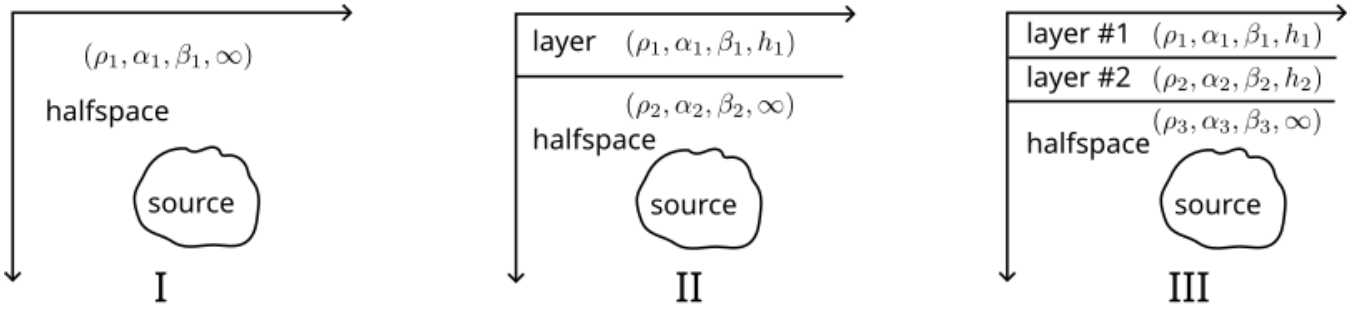


Next, we divide the fault into multiple subfaults (see rectangle ABCD) and define their properties as follows:

- Center coordinates relative to the reference fault point (in km).
- Dimensions: Length and width of each subfault (km).
- Rupture characteristics: Rupture velocity and orientation of the rupture front.
- Rupture timing: A piecewise linear function defining the rupture onset at each subfault.



The library provides three environmental setups as it is shown in the following image. In parenteses given the density, velocities of p- and s-waves and thickness for each medium.



These parameters are crucial for simulating seismic wave propagation through different layers of the Earth's crust.

I. A halfspace

Create a python tupple as follows:

```
medium = ((rho_1, alpha_1, beta_1, 0),) # with 0 we mean halfspace (infinite thinkness)
```

II. A layer over a halfspace

Note: There are bugs in this setup, and we are currently working on fixing them.
For the time being, in the case of two layers, use the third setup (III.) with two identical upper media, each with a thickness of h_1/2.

```
medium = ((rho_1, alpha_1, beta_1, h_1),
          (rho_2, alpha_2, beta_2, 0),)
```

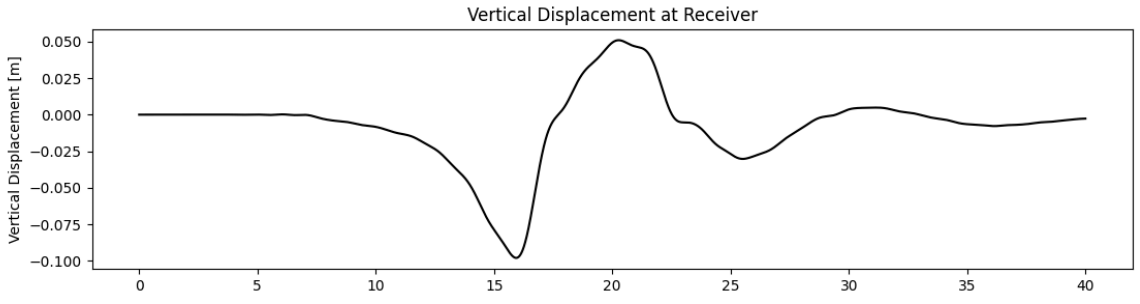
III. Two layers over a halfspace

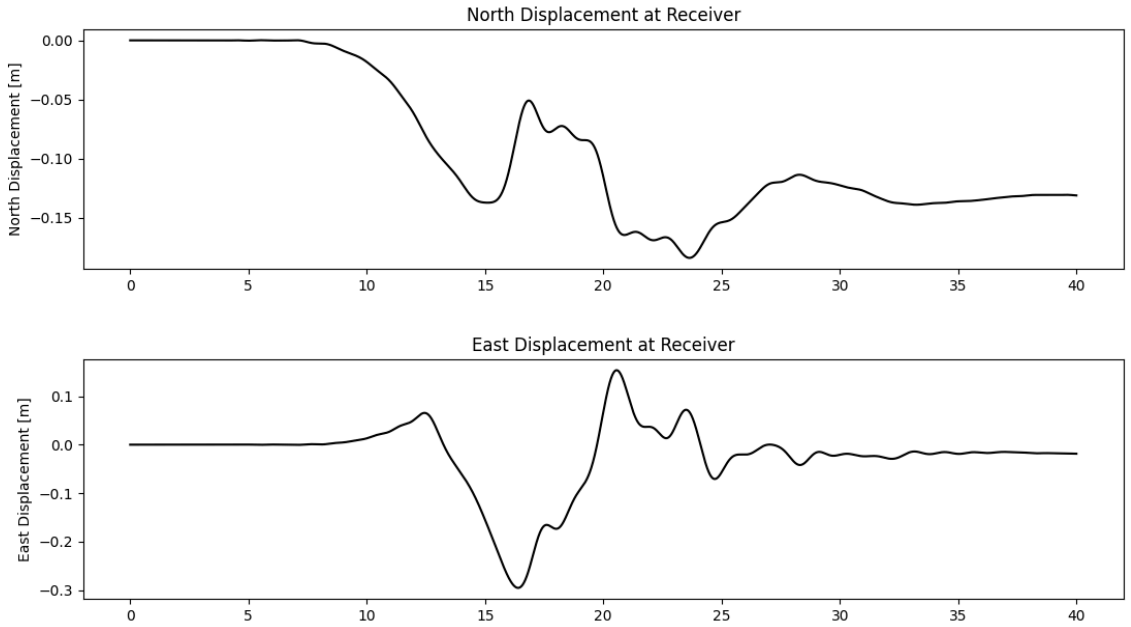
```
medium = ((rho_1, alpha_1, beta_1, h_1),
          (rho_2, alpha_2, beta_2, h_2),
          (rho_3, alpha_3, beta_3, 0),)
```

Following a test case that implements the final example (Fig. 6) from the paper **Discrete Wave Number Representation of Elastic Wave Fields in Three Space Dimensions**, Journal of Geophysical Research, Vol. 84, No. B7, by Michael Bouchon.

```
from grmot import Fault
import numpy as np
import matplotlib.pyplot as plt

x_fault = 5.0
y_fault = 0.0
z_fault = 1.0
x_receiver, y_receiver = 7.0, 1.0

sources = [((3, 10, 0, 0, 2, 270 * np.pi / 180), [(0, 1.0)])]

angles = (90.0 * np.pi / 180.0, 0.0 * np.pi / 180.0, 180.0 * np.pi / 180.0)
```

```
fpars = (1 / 10, 5)
N = 1 / fpars[0] * (4 * fpars[1])
print(N)

medium = ((2.4, 2.5, 1.4, 0.5), (2.4, 2.5, 1.4, 0.5), (2.8, 5.0, 2.8, 0))

rvel = 0
conf = (300, 300, 200.0, 200.0, 1.0)
loc = (x_fault, y_fault, z_fault)
fault = Fault(angles, loc, fpars, medium, conf)

receivers = [(x_receiver, y_receiver)]

north, east, vertical, _, _, _, _, _, _ = fault.simulate(sources, receivers, 8192)

t = np.linspace(0, 1 / fpars[0] * 4, 8192)

# Vertical Displacement Plot
plt.figure(figsize=(10, 3))
plt.plot(t, vertical[0],'black')
plt.ylabel('Vertical Displacement [m]')
plt.xlabel('time [s]')
plt.title("Vertical Displacement at Receiver")
plt.show()

# North Displacement Plot
plt.figure(figsize=(10, 3))
plt.plot(t, north[0],'black')
plt.ylabel('North Displacement [m]')
plt.xlabel('time [s]')
plt.title("North Displacement at Receiver")
plt.show()

# East Displacement Plot
plt.figure(figsize=(10, 3))
plt.plot(t, east[0],'black')
plt.ylabel('East Displacement [m]')
plt.xlabel('time [s]')
plt.title("East Displacement at Receiver")
plt.show()
```





Approximation of an Elliptical Crack

This function `approx_elliptical_crack` approximates an elliptical rupture on a fault by considering a set of rectangular sub-faults. The rupture nucleates at an internal point of an elliptical crack and propagates in a self-similar manner. The instantaneous elliptical rupture front moves toward the crack barrier at a constant velocity.

This elliptical kinematic rupture model was first introduced by Burridge and Willis. Here, we approximate the rupture using rectangular sub-faults.

Function Signature

```
# L, W, dl, radius_xi, radius_eta, xi, eta, coef, delay, nxi, neta, vr, code
approx_elliptical_crack(crack_params)
```

Parameters

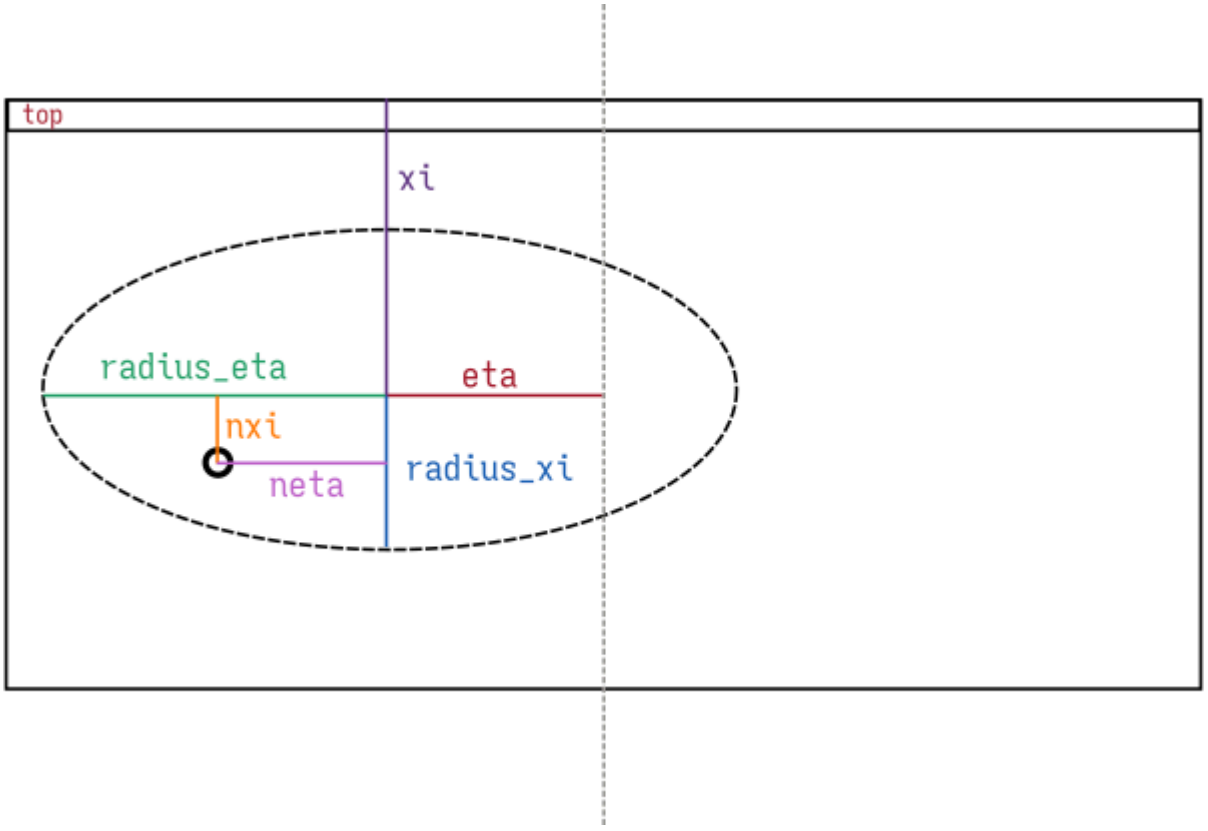
- `L` (float): Length of the fault.
- `W` (float): Width of the fault.
- `dl` (float): Grid spacing for discretization.
- `radius_xi` (float): ξ-axis radius of the elliptical crack.
- `radius_eta` (float): η-axis radius of the elliptical crack.
- `xi` (float): ξ-coordinate of the center of the elliptical crack.
- `eta` (float): η-coordinate of the center of the elliptical crack.
- `coef` (float): Scaling factor.
- `delay` (float): Initial time delay of rupture.
- `nxi` (float): ξ-direction of the nucleation point.

- `neta` (float): η-direction of the nucleation point.
- `vr` (float): Rupture velocity (km/s).
- `code` (str): Unique identifier for the fault model.

Returns

- `source_i` (list): List of rupture details.
- `m0it` (float): Total moment release.
- `maxslip` (numpy array): Maximum slip distribution.
- `ruptvel` (numpy array): Rupture velocity distribution.
- `theta0` (numpy array): Initial rupture angle distribution.
- `code` (str): Fault model identifier.

The following image illustrates the key parameters of the approx_elliptical_crack function.

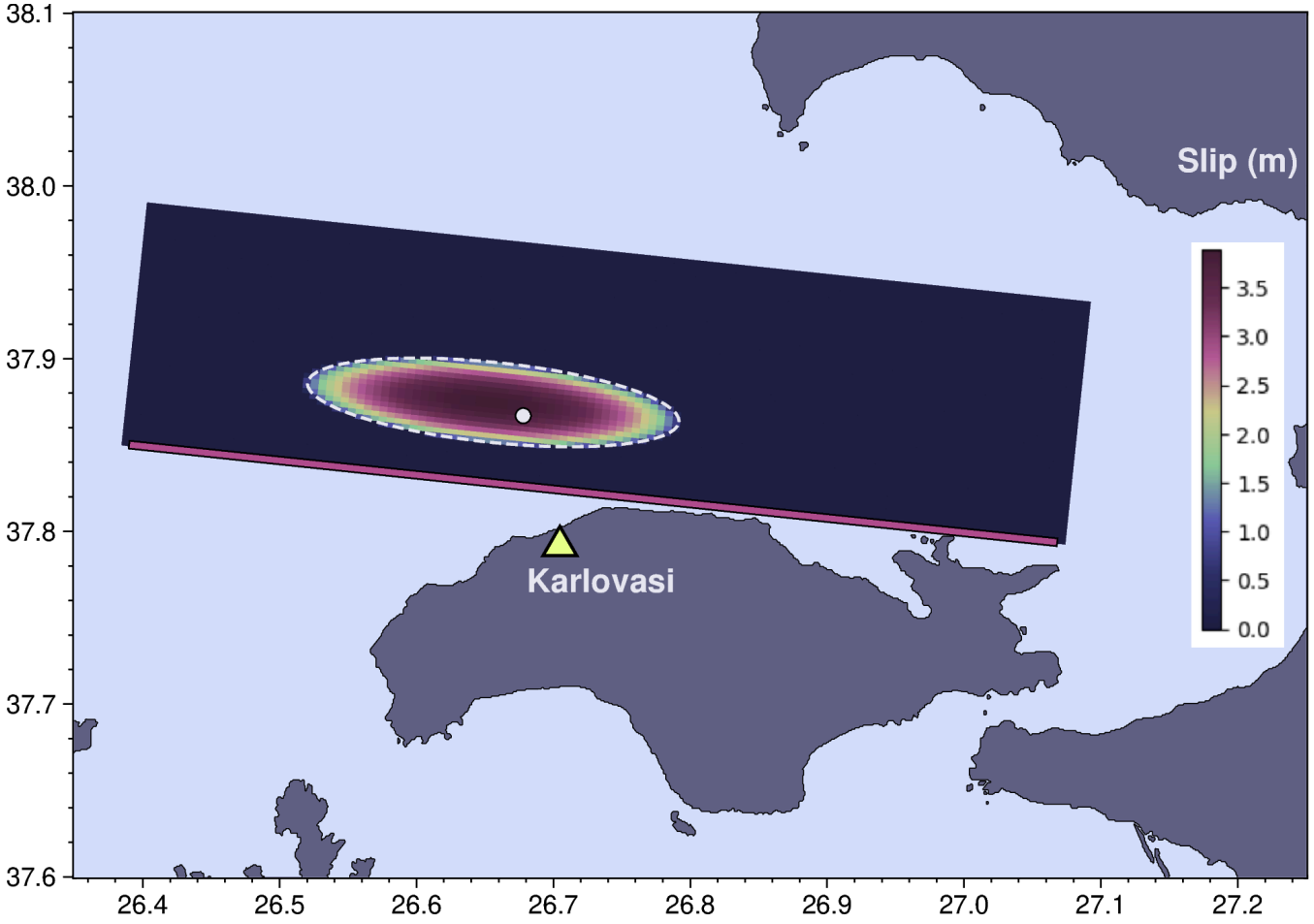


The nucleation point is denoted by the white dot.

Test Case: Synthetic Earthquake Simulation Near Samos Island, Greece

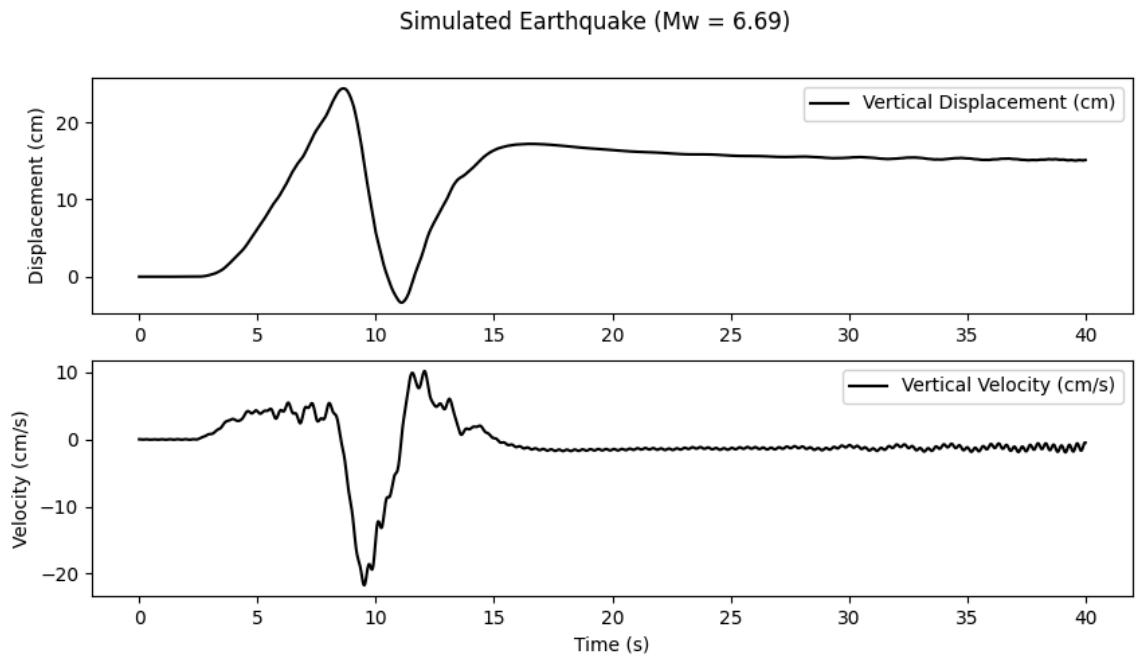
We aim to simulate a hypothetical earthquake on the same fault that ruptured during the 2020 Samos earthquake, evaluating its potential impact on Karlovasi main square.

Below is the map of the synthetic rupture, accompanied by the Python code that implements the simulation.




```
from grmot import approx_elliptical_crack, Fault, latlon_to_km
import numpy as np
import matplotlib.pyplot as plt

it = [24.0, 60.0, 0.5, 4, 12,  8,  -7,  0.6,  0, -1,  2, 2.0, 'Crack']

source, m0, m, r, t, code = approx_elliptical_crack(it)

lat_fault, lon_fault, z_fault = 37.822, 26.740, 1.0
x_fault, y_fault = latlon_to_km(lat_fault, lon_fault)

loc = (x_fault, y_fault, z_fault)

angles = (50*np.pi/180., 276*np.pi/180., -90.*np.pi/180.)
fpars = (1/40, 3.0)
medium = ((2.4, 3.7, 2.25, 0.5),
          (2.5, 4.6, 2.7, 0.5),
          (2.6, 5.4, 3.2, 0.0),)

conf = (300, 300, 250, 250, 1.0)

receivers_db = {
    'KARLOVASI_SQUARE': (37.7916, 26.7048)
}

dir_name = './samos_hyp'
```

```
for receiver_name in receivers_db:
    receivers=[]
    x_receiver, y_receiver = latlon_to_km(receivers_db[receiver_name][0],
receivers_db[receiver_name][1])
    receivers.append((x_receiver, y_receiver))
    dn,de,dv,vn,ve,vv,an,ae,av = fault.simulate(source, receivers, 2048)
    np.savez(dir_name + '/' + receiver_name + '_' + code + '.npz',
              dn=dn[0], de=de[0], dv=dv[0],
              vn=vn[0], ve=ve[0], vv=vv[0],
              an=an[0], ae=ae[0], av=av[0],
              m0=m0)

# Load and analyze the simulated data
karlovasi = np.load('./samos_hyp/KARLOVASI_SQUARE_' + code + '.npz')

# Calculate moment magnitude
mw = 2 * np.log10(karlovasi['m0'] * 10**7) / 3 - 10.7

t = np.linspace(0,40,2048)

# Plot results
plt.figure(figsize=(10, 5))
plt.subplot(2, 1, 1)
plt.plot(t, 100*karlovasi['dv'], label='Vertical Displacement (cm)', color='#3548cf')
plt.legend()
plt.ylabel('Displacement (cm)')

plt.subplot(2, 1, 2)
plt.plot(t, 100*karlovasi['vv'], label='Vertical Velocity (cm/s)', color='#3548cf')
plt.legend()
plt.xlabel('Time (s)')
plt.ylabel('Velocity (cm/s)')

plt.suptitle(f'Simulated Earthquake (Mw = {mw:.2f})')
plt.savefig('simulated.png')
```



Undocumented Functions

The library includes several undocumented functions. Please refer to the source code for details on these functions.

https://github.com/kesmarag/grmot/blob/main/grmot/utils.py

License

GRMOT is distributed as free software (GPL-v3) in the hope that it will useful, but without any warranty.

Acknowledgements

I would like to acknowledge Professor Apostolos Papageorgiou for his guidance and valuable advice.