pySeismicFMM: Python based travel time calculation in regular 2D and 3D grids in Cartesian and geographic coordinates using Fast Marching Method

Marcin POLKOWSKI

Institute of Geophysics, Faculty of Physics, University of Warsaw, Poland (marcin@marcinpolkowski.com)

UNIVERSITY OF WARSAW

Abstract

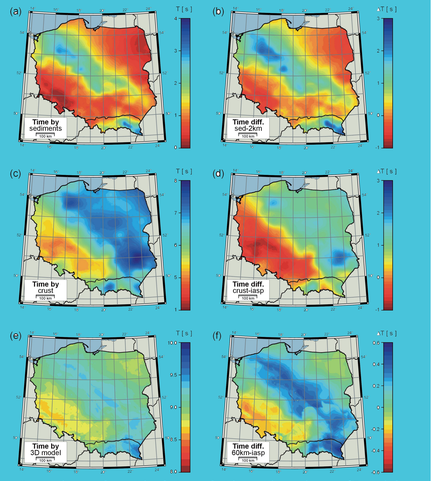
Seismic wave travel time calculation is the most common numerical operation in seismology. The most efficient is travel time calculation in 1D velocity model – for given source, receiver depths and angular distance time is calculated within fraction of a second. Unfortunately, in most cases 1D is not enough to encounter differentiating local and regional structures. Whenever possible travel time through 3D velocity model has to be calculated. It can be achieved using ray calculation or time propagation in space. While single ray path calculation is quick it is complicated to find the ray path that connects source with the receiver. Time propagation in space using Fast Marching Method seems more efficient in most cases, especially when there are multiple receivers. In this presentation a Python module pySeismicFMM is presented – simple and very efficient tool for calculating travel time from sources to receivers. Calculation requires regular 2D or 3D velocity grid either in Cartesian or geographic coordinates. On desktop class computer calculation speed is 200k grid cells per second. Calculation has to be performed once for every source location and provides travel time to all receivers. pySeismicFMM is free and open source. Development of this tool is a part of authors PhD thesis. National Science Centre Poland provided financial support for this work via NCN grant DEC-2011/02/A/ST10/00284.

This work is part of PhD thesis. Full source code of pySeismicFMM will be published later this year.

Model and efficiency

pySeismicFMM is designed to work efficiently on regular 2D and 3D grids and was prepared mainly to work with 3D P-wave velocity model of Poland (Grad et. al 2015). The model covers area of Poland from topography to 60 km deep. Total grid size is 631 x 536 x 6261 (2.117.570.376) cells. In 1.390.642.955 P-wave velocity is provided. To allow correct simulation 3D model has always to be of convex shape.

Single simulation provides traveltime to all grid cells in the model for given source location. Source can be located in one or more grid cells with proper time. For example, an interface can be assumed as source for reflected wave. Single simulation requires about 20GB of operating memory and takes 2 hours to complete. By average about 200.000 grid cells are calculated every second on modern workstation. Unfortunately, due to algorithm construction, it is not possible to migrate to GPU or any other parallel solution.

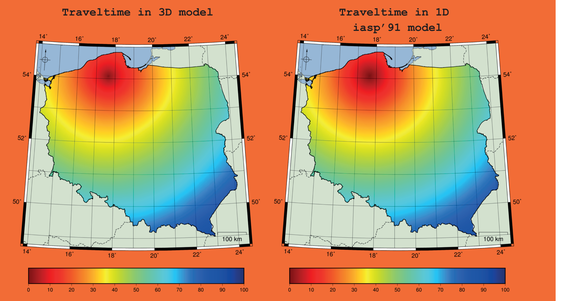


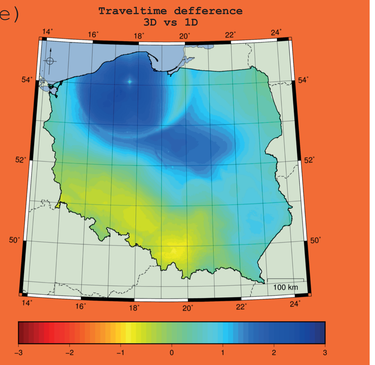
Comparison of vertical travel times through our 3D model and other models. (a) Vertical pass time through sedimentary cover in the 3D model and (b) time difference in relation to homogeneous 2-km-thick layer with a velocity of 3 km/s. (c) Vertical pass time through crust in the 3D model and (d) time difference to the iasp91 model. (e) Vertical pass time through the whole 3D 60-km-thick model and (f) time difference to 60-km-thick of the iasp91 model.

pySeismicFMM Example:

```
from SeismicFMM import SeismicFMM3D
import numpy

myFMM = SeismicFMM3D()
myFMM.SetModelSize(631, 536, 6261)
myFMM.SetGridSize(numpy.deg2rad(0.01), numpy.deg2rad(0.02), 10)
myFMM.ReadVelocityModel("MODELVPF.bin", numpy.single)
myFMM.ReadLatVector("lat.bin", numpy.double)
myFMM.ReadLonVector("lon.bin", numpy.double)
myFMM.ReadHVector("H.bin", numpy.double)
myFMM.CreateCalculationVariables()
myFMM.SetSource(192,538,244)
myFMM.Do(numpy.sum(myFMM.model_velocity > 0))
myFMM.Save("PythonTime.bin")
```





This presentation participates in OSPP

Outstanding Student Poster & PICO Contest

Application

pySeismicFMM is designed to be used in studying of local seismicity. High resolution 3D seismic velocity model of the region of Poland and efficient travel time calculation allows to locate local events with superior accuracy in both quake location and depth.

Other methods

```
myFMM.SetLatVector(Lat)
myFMM.SetLonVector(Lon)
myFMM.SetHVector(H)
myFMM.GetVelocityAtXYZ(lat, lon, z)
myFMM.GetTimeAtXYZ(lat, lon, z)
myFMM.SetSourceGeo(lat, lon, z)
```