Tips for installations

***In case of problems with installations, please contact Matteo Di Giovanni, Soumen Koley, or Jan Harms.***

If you want to run python on your PC/laptop, we strongly suggest to use Anaconda virtual environments and not to mess with native Python on your machine. For installation instructions and suggestions on how to create a Python virtual environment, please visit the documentation at anaconda.com.

We also suggest not to use the latest version of Python available for Anaconda but to create a virtual environment specifying a Python version of 3.7 or 3.8 to avoid conflicts with modules not up to date with the latest release.

Once you created and activated your virtual environment, all the needed Python modules can be downloaded and installed by simply typing “conda install package\_name” in your terminal.

The following modules are needed for our work:

RUMOR

* scipy
* numpy
* matplotlib
* configparser
* mpmath

Seismic analyses

* obspy
* numpy
* matplotlib
* scipy
* netCDF4
* xarray
* basemap
* motuclient

Please note that this document is not intended as an ObsPy manual, but only as a tutorial for showing some basic functionalities. Always refer to the official documentation for detailed instructions at <https://github.com/obspy/obspy/wiki>.

In addition, a convenient tool for this workshop will be Jupyterlab. Python code can be easily modified and executed in blocks in a Jupyterlab environment much like Matlab code. For installation instructions see, for example, <https://jupyter.org/install>. For people working under Windows, the easiest installation is via “pip install jupyterlab”. It just requires the installation of Python to provide the pip command.

How to download and read seismic data using ObsPy

Obspy is a convenient Python module to read, save and manipulate seismic data. It provides lots of useful built-in functions that help facilitate the analysis of seismic data.

First of all, Obspy can read seismic data saved in mostly any format (.SAC, .MSEED, etc...). All the compatible formats and the instructions on how to read the files can be found in the official documentation.

Here, we will focus only on a couple of examples backed up by the attached Jupyter Notebook.

## Read data straight from a file

If you have a file named path\_to\_file/seismic\_data.mseed, just use the function read:

from obspy import read

st = read(‘path\_to\_file/seismic\_data.mseed’)

print(st)

st.plot()

This will fill the variable st with the seismic data stream based on what is stored in the file. To remove the response of the seismometer, Obspy also has a convenient function that reads the response of the seismometer from an appropriate file:

from obspy import read\_inventory

inv = read\_inventory(‘path\_to\_inventory\_file’)

st.remove\_response(inventory = inv, output = ‘ACC’, water\_level = 60)

For the details of the functionalities of these functions, please refer to the official Obspy documentation.

Nevertheless, reading data specifying the file name can be cumbersome if you have to read a huge amount of data from many files. This can be facilitated by the use of the SeisComP Data Structure (SDS).

## Read data using SDS on local storage

The SDS provides a basic level of standardization and portability to softwares that need direct access to data files.

Given data from a seismic station from a given year and characterized by the codes NETWORK, STATION, LOCATION, CHANNEL,the basic directory and file layouts are defined as:

PATH\_TO\_SDS/Year/NETWORK/STATION/CHANNEL.TYPE/NETWORK.STATION.LOCATION.CHANNEL.TYPE.YEAR.DAY

PATH\_TO\_SDS : arbitrary base directory

YEAR : 4 digit year

NET : Network code/identifier, up to 8 characters, no spaces

STA : Station code/identifier, up to 8 characters, no spaces

CHAN : Channel code/identifier, up to 8 characters, no spaces

TYPE : 1 characters indicating the data type, recommended types are:

'D' - Waveform data

'E' - Detection data

'L' - Log data

'T' - Timing data

'C' - Calibration data

'R' - Response data

'O' - Opaque data

LOC : Location identifier, up to 8 characters, no spaces

DAY : 3 digit day of year, padded with zeros

The dots, '.', in the file names must always be present regardless if neighboring fields are empty.

In this way, reading data on your local machine becomes straightforward and easier to handle and you don’t need to know the name of the files beforehand, just where they are stored:

from obspy.clients.filesystems.sds import Client

from obspy import UTCDateTime

from obspy import read\_inventory

root = ‘PATH\_TO\_SDS’

client = Client(root)

Start = UTCDateTime(‘2020-4-1T00:00’)

Stop = UTCDateTime(‘2020-4-2T23:59’)

inv = read\_inventory(‘path\_to\_inventory\_file’)

st = client.get\_waveforms(station = STATION, channel = CHANNEL, network = NETWORK, location = LOCATION, starttime = Start, endtime = Stop)

st.remove\_response(inventory = inv, output = ‘ACC’, water\_level = 60)

print(st)

st.plot()

## Read data using FDSN webservice

Most of the geophysical research institutes around the world participate in the IRIS FDSN webservice that allows to download and read public seismic data, e.g. from national seismic networks, directly on your computer.

First of all, to visualize all the institutions participating to FDSN use:

from obspy.clients.fdsn.header import URL\_MAPPINGS

for key in sorted(URL\_MAPPINGS.keys()):

print("{0:<11} {1}".format(key, URL\_MAPPINGS[key]))

This will print all the codes of the available service providers. To obtain seismic data:

from obspy.clients.filesystems.fdsn import Client

from obspy import UTCDateTime

client = Client(‘INGV’) #EXAMPLE OF ONE SERVICE PROVIDER

Start = UTCDateTime(‘2020-4-1T00:00’)

Stop = UTCDateTime(‘2020-4-2T23:59’)

inv = client.get\_stations(network = NETWORK, station = STATION, location = LOCATION, channel = CHANNEL, level = ‘response’)

st = client.get\_waveforms(station = STATION, channel = CHANNEL, network = NETWORK, location = LOCATION, starttime = Start, endtime = Stop)

st.remove\_response(inventory = inv, output = ‘ACC’, water\_level = 60)

print(st)

st.plot()

The client.get\_station() method allows you to download the response of the detector.

## Attached Jupyter Notebook

Since you may need some sample seismic data to start making the analyses for Sos Enattos, the ‘read\_and\_save.ipynb’ notebook shows examples of how to download seismic data from FDSN since one of the seismic stations at the site is part of the INGV national seismic network as IV.SENA.

The notebook is divided into two sections:

1. Download small data chunks and store them in a stream variable to manipulate them;
2. Make a mass download, save the data in a SDS data structure in your local machine and read them. This is the best option if you plan to download a huge amount of data.

## Inventory files

Inventory files are files that contain the station metadata and are used to deconvolve the data stream with the response of the detector. Inventory files can be saved in multiple formats but the most common are .dseed and .xml.

If you are using the FDSN webservice and you want to save the station metadata once for all, you can save it by just using:

from obspy.clients.filesystems.fdsn import Client

client = Client(‘INGV’) #EXAMPLE OF ONE SERVICE PROVIDER

inv = client.get\_stations(network = NETWORK, station = STATION, location = LOCATION, channel = CHANNEL, level = ‘response’)

inv.write("example.xml", format="STATIONXML")

And to read it:

from obspy import read\_inventory

inv = read\_inventory(‘example.xml’)

Obviously, the metadata in the inventory must coincide with the network, station, location and channel codes in the data stream, otherwise the remove\_response method will fail.

If you know which detector and data-logger are installed at the station and other details, you can also build an inventory file on your own because the response of the most common sensors can be downloaded by Obspy. An example of this procedure can be found in the create\_inventory\_file.ipynb notebook.

For the tutorials on how to access all the detailed information in the inventory, please refer to Obspy documentation.

Download sea waves data

Sea waves data can be downloaded from copernicus.eu. First of all, be sure to register and get an account. Then, navigate as follows:

1. On the homepage of copernicus.eu, click on the icon “Marine”
2. Click on “Discover Marine”
3. Click on “Ocean Products”
4. Select “Mediterranean Sea” in the dropdown menu “Regional domain”
5. Click on the dataset [MEDSEA\_ANALYSISFORECAST\_WAV\_006\_017](https://resources.marine.copernicus.eu/?option=com_csw&view=details&product_id=MEDSEA_ANALYSISFORECAST_WAV_006_017)
6. Click on “Download product”
7. In the dropdown menu choose the first option

At this point you will be prompted to a page in which you can select what you want to download, like the coordinates of the region of interest, the period of interest and the variables that you need. Sea wave height is given by the variable “VHM0”. Click on “Download” at the bottom of the page. The data will be downloaded in a .nc format file.

Please note that you have unlimited downloads available, but each download has only a limited maximum downloadable file size.Therefore, you can download data from a short period with all the variables or only one variable from a longer period. For example 3 months of continuous VHM0 data reach the maximum downloadable file size.

To avoid going through the navigation of the copernicus website everytime, data can be downloaded also thorough python command that needs the motuclient module:

python -m motuclient --motu http://nrt.cmems-du.eu/motu-web/Motu --service-id MEDSEA\_ANALYSISFORECAST\_WAV\_006\_017-TDS --product-id med-hcmr-wav-an-fc-h --longitude-min 0 --longitude-max 17 --latitude-min 35 --latitude-max 45 --date-min "2021-1-1 00:00:00" --date-max "2021-3-31 23:00:00" --variable VMDR\_WW --out-dir="output\_directory" --out-name="name\_of\_file.nc" --user your\_user --pwd your\_password

Concerning VHM0, the data consists of a time series per each point in the coordinates grid of the region of interest. In the case of the Mediterranean sea, the grid spacing is of about 7 km.

Calculate PSD with ObsPy

ObsPy has a built-in function to calculate, save and show the PSD using the method of McNamara et al. 1994. The advantage of calculating the PSD with Python is that they can be easily manipulated. The drawback is that, if on one hand the smoothing allows one to easily study the general trend of the spectra, this method applies a very heavy smoothing that reduces the resolution of the spectra.

Among the scripts provided, npz.py shows how to calculate and save the PSD with ObsPy. The example calculates the PSDs over 1 hour segments; the parameters of the calculation can obviously be changed (see ObsPy documentation). The script npz.py shows how to read the saved PSDs for plotting and manipulation. The notebook show\_PPSD.ipynb shows how to retrieve the PSD saved in the files and show them in a histogram-like plot.

Once you stored the PSDs in a variable, you can access them using the following methods:

* .\_period\_binning[0]: the period bins of the spectra;
* .\_times\_processed: the corresponding GPS times of the spectra;
* .\_binned\_psds: the actual spectra.

In particular, .\_times\_processed is a 1-D array with length equal to the total number of PSD segments. .\_binned\_psds is a 2-D array with x dimension equal to the total number of PSD segments and y dimension equal to the period binning.

The method s0z\_psd7,g2,g3,g4 = s0Z.extract\_psd\_values(period=4.5) extracts the time series of the PSD at a given period (4.5 s in this case). To extract the times in a readable format do the following: tztime0 = [t.datetime for t in s0Z.times\_processed].

If you do not wish to calculate the PSD using ObsPy, we also provide a script for calculating the spectra using standard Python libraries that you can modify according to your needs.

Up to now, all the scripts in the Notebooks folder are set to work with PSD calculated using ObsPy’s internal methods.

A .tar.gz file with the PSD calculated with ObsPy is included in this Drive.

Calculate polarization

To calculate the polarization of seismic noise we provide two scripts that use two different methods:

* Tanimoto (2006)
* Flinn (1965)

The first method was especially designed to calculate microseismic noise polarization using a single station. Since this method is not available as part of internal functions, the scripts implements the procedure from scratch.

On the other hand, the second method is more general and can be applied to any range of frequencies. Furthemore, it is implemented in ObsPy’s libraries and therefore the script is much easier to read.

The two procedures output results consistent with each other.