CUSeisTut

Release 0.0.1

CUSeisTut team

Nov 23, 2022

CUSEISTUT

This document is developed and maintained by CUHK Seismology LAB.

Structure of this doc

1. Entry level
2. Intermediate level
3. Advanced level

Useful links

- CUHK EASC Graduate Division
- CUHK Cryosphere Lab

CHAPTER

ONE

INTRODUCTION

This CUSeisTut was created to provide guidance and training for skillsets needed by seismologists. Over ten modules make up the training process, which are organized into three levels: Entry, Intermediate, and Advanced. The modules are designed in a step-by-step format and is friendly to students who have no prior expertise in scientific research. After finishing the online training modules, students are anticipated to develop sophisticated hands-on skills in seismology.

The first module in the Entry Level introduces the basic operation commands in processing the text file in the Linux system. This module seeks to familiarize users with the Linux environment and operations in terminals. The second module focuses on the most widely used mapping tool in seismology, The Generic Mapping Tools (GMT). In that module, we provide instructions for creating a map with various elements, such as earthquake locations, coast lines, and topography.

Seismology is an observations-based science. The seismograms, the records of ground shakings, include information about earthquakes and subsurface structures. Therefore, processing seismic data is a fundamental skillset in seismology. The final module in Entry Level block explains how to use ObsPy, a tool based on Python, to process seismic waveform data

Users can explore more challenging questions in seismology after being equipped with the skillsets in the Entry Level. In the intermediate Level training, we provide guidance and codes to illustrate the procedure for detecting and locating earthquakes from seismic waveform data. Furthermore, we provide the Advanced Level block for users' interests, which focuses on two fundamental questions in seismology: how to resolve the earthquake source and earth structure. Understanding the Advanced Level modules requires more background knowledge in earthquake source physics and topography theories.

Your feedback will be helpful in developing and improving the CUSeisTut. Feel free to contact us by clicking the "Give us feedback here"under the Appendix.

CUHK Seismology Lab, 2022, Jan

How to help update/maintain this online document.

1.1 How to contribute

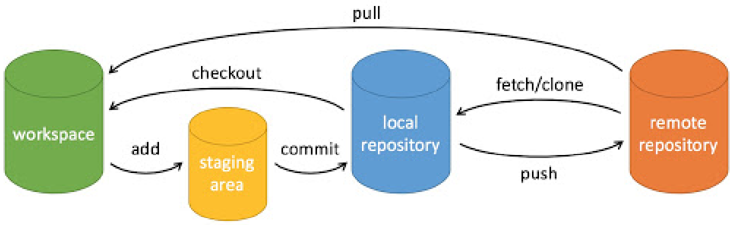
It's very easy for students to help make contributions to the CUSeisTut

Here are several steps based on github commands:

1. open a terminal in your computer
2. run: git clone https://github.com/JunhaoSong/cuseistut.git
3. enter the cloned directory named "cuseistut" (you will find a hidden local repository named ".git" in the cloned directory)

4. make modifications
5. after 4, run: git add <modified files/folders>
6. after 5, run: git commit -m “message”
7. after 6, run: git push (username and token are required, please contact Junhao Song for the temporary token.)

steps 5, 6, 7 can be easily understood based on the figure below.



After successfully pushing your modifications, check the updated CUSeisTut website

CHAPTER

TWO

UNIX COMMANDS

The tutorial is based on the Unix system. In this chapter, we will introduce some basic Unix commands that are useful in later modules. We will introduce basic commands related to Directory management, text file reading, Unix input and output redirection, and the Pipe command. the file used in this tutorial is the earthquake catalog and station list in the Banda Arc–Australian Plate Collision Zone (Jiang et al., 2022).

Developed by Po Wang LAM (Ryan) under the instruction of Han CHEN.

2.1 1 Target

Master the basic operation in the directory and file.

Understanding the input and output redirection and the Pipe command.

Make sub-catalogs and sub-station lists based on the Downloaded complete earthquake catalog and save the file in corresponding directories.

2.2 2 Commands and concept included

Directory management: `pwd` `cd` `mkdir` `ls`

File reading: `more` `cat` `wc` `grep` `awk`

Input and output redirection: > <

Pipe: `|`

2.3 3 Commands usage and examples

2.3.1 3.1 Directory management

Before any operation, it is important to make sure you are at the correct working directory. First, to know which directory you are at, you can use the print working directory command `pwd`. It will print the current working directory as output in the terminal.

```
$ pwd
/current-directory     #for illustration only
```

You are now at `current-directory`. Then, you should change the current directory to the work directory you want, here we used the desktop as example, by the command change directory `cd`.

```
$ cd /home/user/desktop
$ pwd
/home/user/desktop
```

Note: The `/home/user/desktop` here is used as an example. please change it to your own directory accordingly.

Now you are at the `/home/user/desktop`. Before downloading the data, you may want to create an own directory for storing them. This can be done by a simple command `mkdir`.

```
$ mkdir Data_storage
```

The command `ls` list the content under current directory. Now run the list files `ls` command the make sure the `Data_storage` is created.

```
$ mkdir ls
Data_storage
```

Now you can change your working directory to `Data_storage`.

```
$ cd Data_storage
$ ls
```

After running the `ls`, no output was shown, which means that the directory is now empty.

Tip: Several common commands for dealing with directories

lslist files: List directories and file names

cdchange directorySwitch directory

pwdprint work directoryShow current directory

mkdirmake directoryCreate a new directory

rmdirremove directoryDelete an empty directory

cpcopy file: Copy a file or directory

rmremove: delete file or directory

mvmove file: Move files and directories, or change the names of files and directories

2.3.2 3.2 File reading

3.2.1 Browse documents

Now we can download the catalog file. The file could be download from `here`. After downloading the catalogue, it will first be stored in the `~/Downloads`. To move the files from `~/Downloads` to the directory you created, you may use the command `mv`

```
$ pwd
Home/user/desktop/Data_storage
$ mv ~/Downloads/Unix_command_Materials.tgz ./
```

(continues on next page)

(continued from previous page)

```
$ ls
Unix_command_Materials.tgz
```

Note: The directory `~/Downloads` here is suitable for Unix system. please change it to your own directory accordingly.

The file was now moved to your working directory. As the file is zipped as a .tgz file, it needs to be unzipped. It can be done by the command `tar`

```
$ tar -zxf Unix_command_Materials.tgz
$ ls
CUSeisTut Unix_command_Materials.tgz
```

The file was now unzipped and the directory `CUSeisTut` was create. Change your working directory to the `CUSeisTut` and use `ls` to check the contents.

```
$ cd CUSeisTut
$ ls
A_Detailed_EQ_Catalog  Stations_Info  TSR_Paper
$ ls A_Detailed_EQ_Catalog
banda_arc_catalog.txt
$ ls Stations_Info TSR_Paper
GE_3_stations.txt  YS_30_stations.txt
Stations_Info:
GE_3_stations.txt  YS_30_stations.txt

TSR_Paper:
tsr-2021041.1.pdf  tsr-2021041_supplement.docx
```

We will first look at the earthquake catalog `banda_arc_catalog.txt` stored in `A_Detailed_EQ_Catalog`, try to change your working directory to there.

Before processing the file, we would like to ensure the file is compatible and contains the desired data. There are several ways for reading a text file from the terminal, we would like to introduce two methods: `more` `cat`

```
$ cd A_Detailed_EQ_Catalog
$ ls
banda_arc_catalog.txt
$ more banda_arc_catalog.txt
 indx year mon day time sec_relative_to_day  res      lat     lon    dep    mag  visual_flag hypodd_flag
    2 2014 03 18 16:56:34.260000 60994.2600    0.697   -9.092  124.191   82.1   3.1 1 1
    3 2014 03 19 14:39:17.472600 52757.4726    1.593   -8.519  126.329   23.2   3.0 1 0
    4 2014 03 20 15:32:39.914100 55959.9141    0.706   -7.482  127.900  192.6   3.7 1 0
    5 2014 03 21 15:10:35.760000 54635.7600    1.090   -8.936  124.325   81.2   1.6 1 1
    6 2014 03 21 18:16:37.720199 65797.7202    1.898   -8.928  125.775   69.8   2.3 1 0
    7 2014 03 22 08:18:17.449799 29897.4498    1.095   -8.974  125.587   -4.5   3.0 1 0
    8 2014 03 22 13:57:21.227599 50241.2276    0.792   -9.891  123.950   62.9   3.9 1 0
    9 2014 03 22 15:43:47.217700 56627.2177    1.546   -8.747  122.528  114.1   2.4 1 0
   10 2014 03 22 19:23:44.019000 69824.0190    0.642   -7.383  128.098  218.4   2.9 1 0
   11 2014 03 23 04:37:50.889999 16670.8900    0.787   -9.359  124.140   57.7   3.3 1 1
   12 2014 03 23 12:16:37.705200 44197.7052    1.017  -10.549  123.603   -4.1   3.3 1 0
   13 2014 03 24 15:56:41.793600 57401.7936    1.512   -7.118  127.153  152.0   3.0 1 0
```

(continues on next page)

(continued from previous page)

```
  14 2014 03 24 17:16:01.760000 62161.7600    1.904   -9.108  124.246   68.7   3.1 1 1
  17 2014 03 25 17:14:15.798098 62055.7981    0.768   -7.407  126.617  199.0   2.5 1 0
  18 2014 03 25 18:57:23.260000 68243.2600    0.606   -9.000  124.130   75.6   1.8 1 1
  19 2014 03 26 06:32:57.887500 23577.8875    0.851   -7.449  127.497  166.7   2.6 1 0
  23 2014 03 29 20:31:47.700000 73907.7000    0.734   -9.381  123.590   82.9   3.0 1 1
  24 2014 03 31 10:56:15.241300 39375.2413    0.973   -7.424  126.049   23.6   4.1 1 0
  26 2014 04 01 01:00:36.254898  3636.2549    0.932   -7.614  127.268  188.9   2.9 1 0
  27 2014 04 02 03:01:25.829400 10885.8294    1.684   -7.507  122.368   21.8   4.4 1 0
  28 2014 04 02 15:26:23.424199 55583.4242    0.827   -9.073  124.145   37.0   2.4 1 0
  29 2014 04 02 18:20:08.600000 66008.6000    0.840   -8.897  124.130   78.2   2.2 1 1
  30 2014 04 02 19:06:23.708500 68783.7085    1.137   -9.317  120.447  131.6   3.3 1 0
  31 2014 04 03 11:44:25.794800 42265.7948    0.842   -7.779  128.245  218.4   3.5 1 0
  33 2014 04 04 07:06:34.235300 25594.2353    1.358   -7.970  123.634  119.6   3.6 1 0
  34 2014 04 04 11:18:03.472600 40683.4726    0.769   -8.941  124.175   14.9   2.4 1 0
  35 2014 04 04 16:50:22.464900 60622.4649    2.933   -8.011  127.448  115.0   2.2 1 0
  36 2014 04 05 02:37:26.165300  9446.1653    1.564   -8.899  125.963   -4.5   2.4 1 0
  --More--(0%)
```

The more command outputs the first few rows of the file, we can press “Enter” to show more lines. and Press ‘Ctrl+C’ to exit the command.

```
$ cat banda_arc_catalog.txt
   ......
   28683 2018 12 29 12:28:05.790000 44885.7900    0.860   -8.220  123.825  180.2   3.5 1 1
   28684 2018 12 29 15:32:39.982800 55959.9828    1.553   -8.846  124.025   66.1   2.0 1 0
   28686 2018 12 29 20:29:02.320000 73742.3200    0.652  -10.036  123.314   31.6   2.3 1 1
   28688 2018 12 30 03:43:30.899300 13410.8993    1.217   -7.994  128.068  183.4   3.7 1 0
   28689 2018 12 30 04:10:53.381499 15053.3815    1.544   -9.896  118.925   33.8   5.0 1 0
   28692 2018 12 30 14:47:19.025200 53239.0252    0.865   -7.788  127.986  176.9   3.5 1 0
   28695 2018 12 30 18:21:30.339600 66090.3396    0.942   -9.970  123.299   60.6   2.6 1 0
   28696 2018 12 30 20:22:25.646099 73345.6461    0.928   -8.069  123.234  206.4   3.2 1 0
   28697 2018 12 31 04:36:34.280000 16594.2800    0.769   -8.808  124.321   96.4   2.5 1 1
   28699 2018 12 31 19:13:00.751600 69180.7516    0.983   -8.879  123.539    5.2   2.9 1 0
```

The cat command will pop the whole content at once.

We could count the total lines and characters of the catalog by using the wc command.

```
$ wc -l banda_arc_catalog.txt
19075 banda_arc_catalog.txt
$ wc -c banda_arc_catalog.txt
1735843 banda_arc_catalog.txt
```

The parameters -l and -c are parameters that choose the output. -l means count the lines of the file and -c means count the total characters of the file. The catalog is a very large catalog with 19074 events (first line of the file indicates the contents).

Tip: The `$?` could be used to represent specific column, such as `$1` represents the 1st culumn, `$2` represents the 2nd column, `$NF` represents the last column.

After viewing the original catalog, some processes can be done to divide the catalog for the analysis. The target is to generate sub-divided catalogs based on different properties like time and magnitude. To achieve this, some operational commands need to be used. `grep` and `awk` is two simple commands for searching certain content from the file.

3.2.1 Extract the text content

The `grep` is a command used to search for texts and strings. It output all the rows that contain the searched character.

For example, when we would like to search earthquakes that occurred in 2014, we can use the following commands.

```
 1  $ grep '2014' banda_arc_catalog.txt
 2    ...
 3    2707 2014 12 31 13:46:44.510000 49604.5100    1.210   -8.959  123.953   91.9   3.2 1 1
 4    2709 2014 12 31 15:50:22.129999 57022.1300    0.599   -8.116  120.694    3.2   2.1 1 1
 5    2710 2014 12 31 15:53:45.350000 57225.3500    1.305   -9.457  119.644   64.3   2.2 1 1
 6    2711 2014 12 31 16:22:48.430000 58968.4300    1.488   -9.476  120.123   75.6   1.9 1 1
 7    2712 2014 12 31 18:16:21.280000 65781.2800    1.329  -10.439  123.626  110.1   2.4 1 1
 8    2714 2014 12 31 19:07:44.320000 68864.3200    1.350   -9.601  119.909   55.2   1.8 1 1
 9    2715 2014 12 31 19:14:17.627898 69257.6279    1.265   -9.331  124.087    4.9   2.5 1 0
10    2716 2014 12 31 19:19:49.720000 69589.7200    1.649   -9.662  119.824   24.4   1.9 1 1
11    2717 2014 12 31 19:47:27.840000 71247.8400    1.068   -9.146  118.864   51.9   2.6 1 1
12    2718 2014 12 31 20:24:31.120000 73471.1200    1.724   -9.072  123.987   73.2   1.9 1 1
13    2720 2014 12 31 20:33:22.680000 74002.6800    1.350   -8.293  120.601   39.5   2.1 1 1
14    2721 2014 12 31 20:40:36.260000 74436.2600    0.917  -10.070  119.152   13.2   2.2 1 1
15    2722 2014 12 31 22:46:19.320000 81979.3200    1.516   -9.511  120.082   70.5   1.9 1 1
16    6638 2015 05 07 06:06:54.990000 22014.9900    0.837   -9.458  125.033    7.5   2.7 1 1
17    6771 2015 05 12 06:06:54.222800 22014.2228    0.684   -8.800  120.459  109.0   2.1 1 0
18   12014 2015 11 04 19:28:46.320000 70126.3200    1.223   -8.298  125.076    5.2   2.4 1 1
19   20141 2016 06 18 18:32:52.840000 66772.8400    0.836   -8.311  123.929  176.8   1.9 1 1
20   20142 2016 06 18 19:14:26.900000 69266.9000    1.531   -9.441  124.789    4.1   1.8 1 1
21   20143 2016 06 18 20:03:26.445700 72206.4457    2.369   -8.408  126.589   -3.2   2.6 1 0
22   20144 2016 06 18 20:31:17.911600 73877.9116    1.388   -8.957  119.781   21.3   2.0 1 0
23   20147 2016 06 19 01:38:58.808900  5938.8089    1.081  -11.094  118.986  110.4   3.2 1 0
24   20148 2016 06 19 05:31:08.009999 19868.0100    1.257   -9.681  119.794   56.3   2.3 1 1
25   22014 2016 09 11 08:59:33.940000 32373.9400    1.514   -8.672  118.465  119.5   2.6 1 1
26   27150 2018 07 25 10:21:49.201498 37309.2015    0.947   -8.532  126.669   69.8   3.4 1 0
```

The command gives an output in the terminal with all rows including '2014'. But here are some problem, some row contains '2014', but the '2014' does not represent the year 2014 (e.g., line 16 to line 26).

The `awk` is a more powerful tool for manipulating data and producing reports. The awk command allows the user to use variables, numeric functions, string functions, and logical operators.

Tip: General command: awk ['pattern {action}'] [file_name]

[pattern] : indicate where to execute the action, for example, NR>10 means lines > 10

[action] : the default action is to print out all lines fulfilled the pattern, but the action can also be more specific with different input like calculation.

[file_name] : the file to process

The `-F` parameter could be used to specify the delimiter. such as `awk -F "[|]" [‘pattern action’] [file_name]` specify the `|` as the delimiter. By default, the delimiter is `Space`.

```
$ awk 'NR<20{if ($11>3.0) print;}' banda_arc_catalog.txt
 indx year mon day time sec_relative_to_day  res      lat     lon    dep    mag  visual_
↪flag hypodd_flag
   2 2014 03 18 16:56:34.260000 60994.2600    0.697   -9.092  124.191   82.1   3.1 1 1
   4 2014 03 20 15:32:39.914100 55959.9141    0.706   -7.482  127.900  192.6   3.7 1 0
   8 2014 03 22 13:57:21.227599 50241.2276    0.792   -9.891  123.950   62.9   3.9 1 0
  11 2014 03 23 04:37:50.889999 16670.8900    0.787   -9.359  124.140   57.7   3.3 1 1
  12 2014 03 23 12:16:37.705200 44197.7052    1.017  -10.549  123.603   -4.1   3.3 1 0
  14 2014 03 24 17:16:01.760000 62161.7600    1.904   -9.108  124.246   68.7   3.1 1 1
  24 2014 03 31 10:56:15.241300 39375.2413    0.973   -7.424  126.049   23.6   4.1 1 0
```

In this example, the pattern is `NR>20` and the action is `if($11>3.0) print`. In a readable way, it means for the first 20 lines, print all rows where their column 11($11) is larger than 3.0. With the physical meaning of each column, the output is all events with a magnitude over 3 within the first 19 events

We could print the column that we are interested in separately by adding the column index after `print`.

```
$ awk 'NR<20{if ($11>3.0) print $1,$2,$3,$4,$5,$8,$9,$10;}' banda_arc_catalog.txt
indx year mon day time lat lon dep
2 2014 03 18 16:56:34.260000 -9.092 124.191 82.1
4 2014 03 20 15:32:39.914100 -7.482 127.900 192.6
8 2014 03 22 13:57:21.227599 -9.891 123.950 62.9
11 2014 03 23 04:37:50.889999 -9.359 124.140 57.7
12 2014 03 23 12:16:37.705200 -10.549 123.603 -4.1
14 2014 03 24 17:16:01.760000 -9.108 124.246 68.7
24 2014 03 31 10:56:15.241300 -7.424 126.049 23.6
```

2.3.3 3.3 File output and input

> and < is the Output and Input Redirection in Unix. Most Unix system commands take input from your terminal and send the resulting output back to your terminal, as what shown in above examples.

If the notation > `file` is appended to any command, the output of that command will be written to the file instead of your terminal.

```
$ echo "hello word"
hello word
$echo "hello world" > test.txt
cat test.txt
hello word
```

In the second command, > was appended to the echo command, so the output was written to the file ‘test.txt’ rather than the terminal.

Note: `echo`: output the original content to the screen if it has no special meaning; if the output content has a special meaning, the output will print its meaning.

> will generate a new file if the file does not exist and will write over the file if the file already exists!

Note: Types of Redirection

1. Overwrite

> standard output, < standard input

2. Appends

>> standard output, << standard input

We could now use `awk` to extract the earthquake based on various criterion and save the output as sub-catalog by using >.

```
$ mkdir earthquake-2014
$ awk '{if ($2==2014) print;}' banda_arc_catalog.txt > earthquake-2014/banda_arc_catalog-2014.txt
$ cd earthquake-2014
$ ls
banda_arc_catalog-2014.txt
$ wc -l banda_arc_catalog-2014.txt
1871 banda_arc_catalog-2014.txt
```

Here we make a directory `earthquake-2014` and extract the earthquakes that occurred in 2014 (e.g. $2==2014) and save the sub-catalog into file 'banda_arc_catalog-2014.txt' under the directory. We than count the events number by using `wc -l`

< will direct the file as a standard input to the command, for example

```
$ echo line1 > test.txt
$ echo line2 >> test.txt
$ while read line
$ do
$ echo $line
$ done < test.txt
line1
line2
```

Here we use < to direct the file `test.txt` as a standard input for the command `while read line`. the command will read the file line by line as `$line`.

2.3.4 3.4 The Pipe in Unix

Pipe is used to redirect the output of a command as the input another command

```
$ grep 2014 banda_arc_catalog.txt | wc -l
1882
$ awk '{if ($2==2014) print;}' banda_arc_catalog.txt | wc -l
1871
```

In this example, we used the pipe `|` to redirect the output of a command `grep` and `awk` as the input of command `wc -l`.

2.4 4 Exercise

4.1. Make a directory `earthquake-Mag` under `A_Detailed_EQ_Catalog`. Extract the year, mon, day, time, lat, lon, dep, and mag of earthquakes with Magnitude between 4 to 6 and save the output to file `earthquake-Mag-4-6.txt`. Count the number of the extracted earthquakes.

4.2. Make a station list file with only station 'ALRB' and save it as `Substation.lst` . Only Network, station, latitude, longitude, elevation are needed. Append same information of station 'PPLA' to the Subastaion.lst.

2.5 References

Jiang C, Zhang P, White M C A, et al. A Detailed Earthquake Catalog for Banda Arc–Australian Plate Collision Zone Using Machine-Learning Phase Picker and an Automated Workflow[J]. The Seismic Record, 2022, 2(1): 1-10.

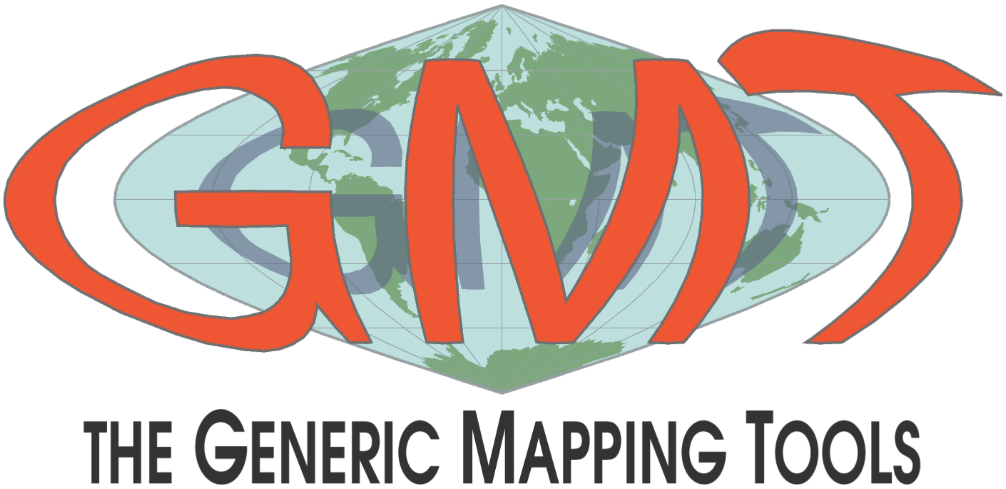
CHAPTER THREE

GENERIC MAPPING TOOLS - TOPOGRAPHIC MAP

3.1 Brief introduction

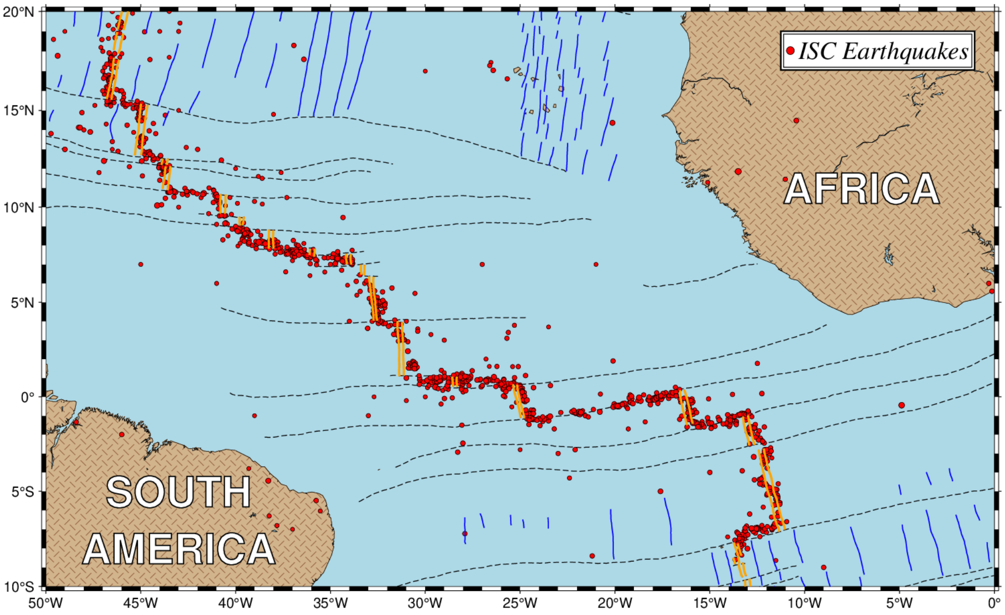
3.1.1 What is GMT?

1. The Generic Mapping Tools (GMT) is one of the most popular mapping softwares in the world;
2. It is widely used across the Earth, Ocean, and Planetary sciences and beyond;
3. It is a free, open source software licensed under the GNU LGPL license.



3.1.2 What can GMT do?

1. **Process data of different format.** For example, time series (seismograph), xy data (fault trace), xyz data(geographic data), and so on.
2. **Generate publication-quality illustrations.** Here is an example:



3. **Automate workflows.** If script is provided and GMT is installed, one figure can be plotted by one command.
4. **Make animations.** For example, this is an animation showing Pacific Earthquakes in 2018 made by GMT.

3.1.3 How to install GMT?

Note:

There are lots of ways to install GMT on your computer.
Here we strongly suggest beginners to install it by conda.

The following steps have been tested successfully for Linux, macOS, and window subsystem for linux.

1. **Install** miniconda
2. **Open a terminal and run these commands sequentially**

Warning: Exclude $ and start without whitespace!

```
$ conda create --name gmt-env
$ conda activate gmt-env
$ conda install gmt -c conda-forge
```

3. **Check if it has been successfully installed**

```
$ gmt --version
6.3.0
```

3.1.4 Start with three simple examples

1. Run this single command and look what will happen

```
$ gmt coast -Rg -JH15c -Gpurple -Baf -B+t"My First Plot" -pdf,png GlobalMap1
```

2. Run the following commands step-by-step and compare the difference

```
$ gmt begin GlobalMap2 png,pdf
$ gmt coast -Rg -JH15c -Gpurple -Baf -B+t"My Second Plot"
$ gmt end show
```

3. Run the same commands using Shell script and compare the difference

```
$ cat GlobalMap3.sh
#!/bin/sh
gmt begin GlobalMap3 png,pdf
gmt coast -Rg -JH15c -Gpurple -Baf -B+t"My Third Plot"
gmt end show
$ sh GlobalMap3.sh
```

No difference found! And do you think which way is better, especially for very complex figures? There is no doubt that the third way, (i..e., based on script), is better. Since we can simply modify the script and re-run the script to refine the figure.

3.1.5 Basic structure of GMT script

Now let's take a further look at the third example in the previous part

1. `#!/bin/sh` specifies **sh** as the command language interpreter.

Tip:

$ which sh
If the output is /dir/to/sh, then replace `#!/bin/sh` by `#!/dir/to/sh`

2. `gmt begin GlobalMap3 png,pdf` initiates a new GMT session. The output figure name is `GlobalMap3`. The output figure format are `png` and `pdf`.

Note:

The sequence is important
That is: gmt begin <figure name> <format1,format2,…,formatN>

3. `gmt coast -Rg -JH15c -Gpurple -Baf -B+t"My Third Plot"` adds the first layer. If it's followed by other commands, as shown in later parts, this layer will be overlaid by new layers.

Note:

The general structure of these "adding-layer" commands is:
gmt <command> -<option1> -<option2> ... -<optionN>

4. `gmt end show` terminates the GMT session. If `show` exists, the produced figure of the first format will be opened by default viewer.

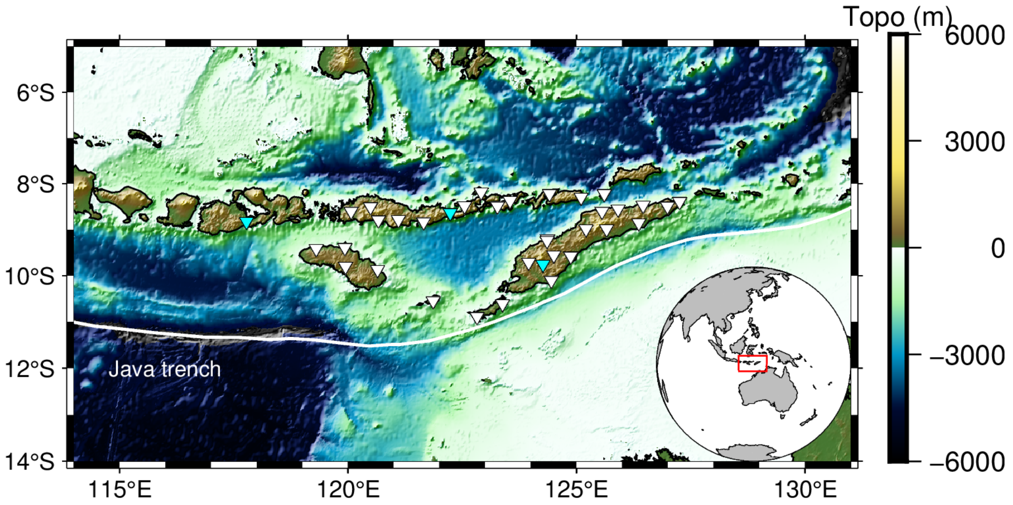
There are a lot of GMT commands and much much much more options. Here, this tutorial aims to give GMT beginners very quick training. Therefore, we will show readers how to generate figures from public seismic data using some commonly used commands. More specificlly, It includes: 1. Plotting topographic map 2. Plotting earthquake catalog 3. Plotting cross sections.

3.2 Plotting topographic map

3.2.1 Preview

After learning this part, you will be able to create the figure below or similar ones.

The figure below is modified from Figure 1 in this paper.



It is generated using the following commands:

```
#!/bin/sh

gmt begin Banda_Arc_Region png

# constructing the frame
gmt basemap -JM10c -R114/131/-14/-5 -BWeSn -Bxa5f1 -Bya2f1
# creating a colormap with customized boundaries
gmt makecpt -Crelief -T-6000/6000/200 -D -Z -H > elevation.cpt
# plotting the topography data, which can be remotely obtained from gmt database using␣
↪the @earth_relief_??? option
gmt grdimage @earth_relief_01m -Celevation.cpt -I+
# plotting the coastline data
gmt coast -W0.5p,black -Da
# inset a global figure with the specify boundary
gmt inset begin -Dn1/0+jBR+w2.5c
gmt coast -Rg -JG123/-10/90/? -B -Swhite -A5000 -Ggray -W0.2p,black -Da --MAP_FRAME_
↪PEN=0.2p
echo -e "\n114 -14\n131 -14\n131 -5\n114 -5\n114 -14" | gmt plot -W0.5p,red
gmt inset end
# plotting the trench data using given data file
gmt plot java_trench.txt -W1p,white
# inserting text
echo '116 -12 Java trench' | gmt text -F+f8p,white
# extracting stations locations from data files
awk 'NR>1 {print $5,$6}' GE_3_stations.txt | gmt plot -Si6p -W0.1p,black -Gcyan
awk -F"|" 'NR>3 {print $6,$5}' YS_30_stations.txt | gmt plot -Si6p -W0.1p,black -Gwhite
# creating colorbar legend
gmt colorbar -Dn1/0+o0.5c/0c+jBL+w5.5c -Bxa3000 -By+l"Topo (m)" -Celevation.cpt

gmt end show

rm elevation.cpt
```

To reproduce it by yourself, you may first download or save java_trench.txt, GE_3_stations.txt, YS_30_stations.txt, and then move these files into your working directory. Try to copy the above commands and run them on your own computer to see if you can generate the same figure without warning or error.

3.2.2 Step-by-Step explanation

1. gmt basemap -JM10c -R114/131/-14/-5 -BWeSn -Bxa5f1 -Bya2f1 to plot base maps and frames

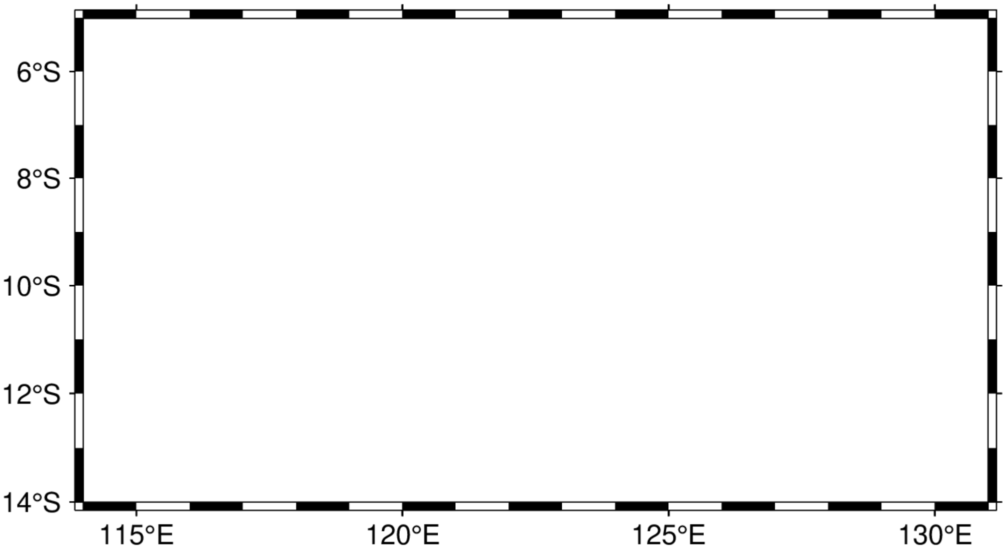
`-JM10c` specifies the map projection type to be Mercator projection. The width of map is 10c (10 centimeters).

`-R114/131/-14/-5` specifies the map range, minimum and maximum longitudes are 114 and 131, minimum and maximum latitudes are -14 -5.

`-BWeSn` specifies that the left and bottom ticklabels are visible, which the right and top ticklabels are invisible.

`-Bxa5f1` specifies that x axes have ticklabels with interval of 5 and ticks with interval of 1.

`-Bya2f1` specifies that y axes have ticklabels with interval of 2 and ticks with interval of 1.



2. gmt makecpt -Crelief -T-6000/6000/200 -D -Z -H > elevation.cpt to make color palette tables

`-Crelief` specifies the input cpt to be relief, click to see a full list of built-in cpt

`-T-6000/6000/200` defines the range of the new CPT by giving the lowest and highest z-values as -6000 and 6000, the increment is 200.

`-D` selects the back- and foreground colors to match the colors for lowest and highest z-values in the output CPT.

`-Z` forces a continuous CPT when building from a list of colors and a list of z-values [discrete].

`-H` is required for modern mode.

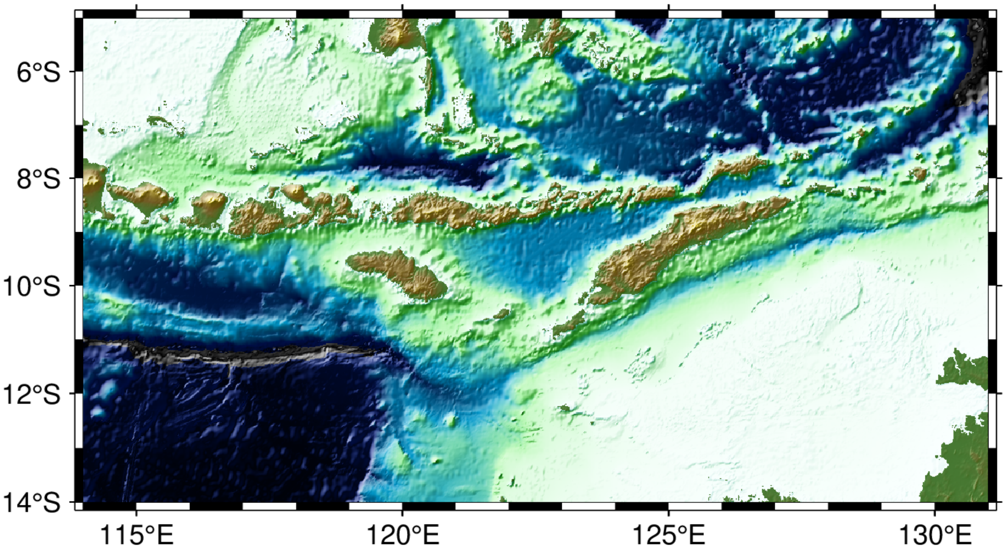
`> elevation.cpt` save the output CPT into a file named elevation.cpt

3. gmt grdimage @earth_relief_01m -Celevation.cpt -I+ to read a 2-D grid file and produce a colored map

`@earth_relief_01m` will download global relief grids (the resolution is 1 minute, it depends on the map range) from the GMT server. Click to view a full list of provided Global Relief Datasets

`-Celevation.cpt` specifies which CPT to use.

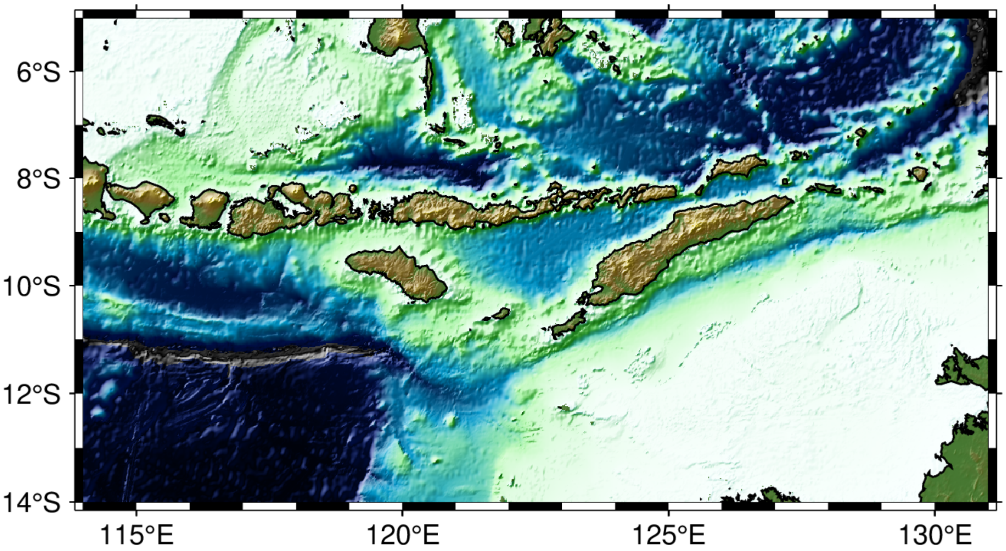
`-I+` selects the default arguments to apply intensity.



4. gmt coast -W0.5p,black -Da to add shorelines

`-W0.5p,black` specifies the line width to be 0.5p (0.5 point) and line color to be black.

`-Da` means that selecting the resolution of shorelines automaticly.



5. gmt inset begin -Dn1/0+jBR+w2.5c to initiate an inset plotting

`-Dn1/0+jBR+w2.5c` gives the location and size of the inset. Now the anchor point is relative to both main and inset map. In the main map, both x and y axes are normalzied with range to be 0-1. So here `n1/0` means the right most and lower most point, i.e., the bottom right point of the main map. In the inset map, the location of this anchor point is also at bottom right (BR). And the width of the inset map is 2.5c.

6. gmt coast -Rg -JG123/-10/90/? -B -Swhite -A5000 -Ggray -W0.2p,black -Da –MAP_FRAME_PEN=0.2p to plot continents and shorelines

`-Rg` specifies the global domain.

`-JG123/-10/90/?` specifies the projection type to be orthographic azimuthal projection and the center longitude and latitude to be 123 and -10. The horizon is 90 degree (<=90). The width is same as the inset map, thus use ?.

`-B` means no ticks and gridlines.

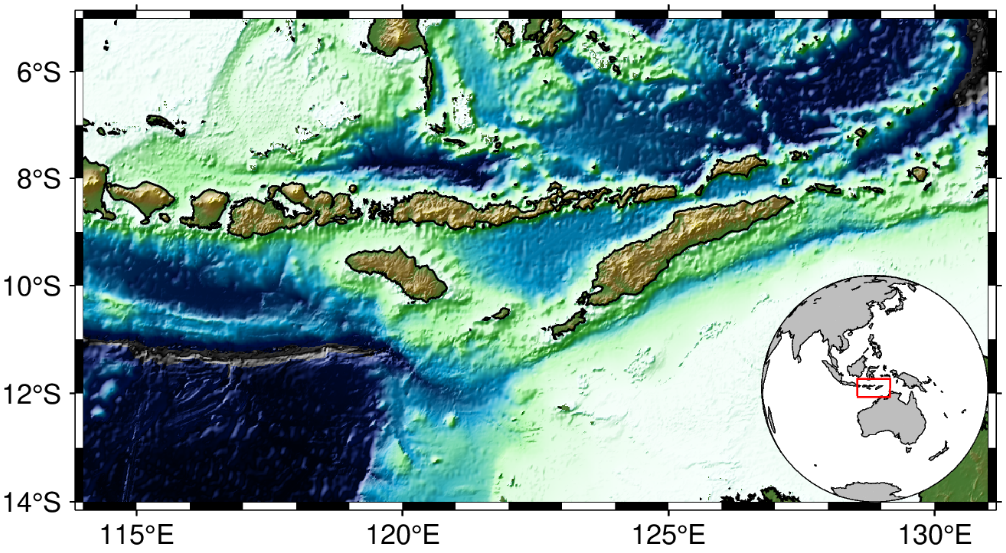
`-Swhite` specifies the color of wet areas to be white.

`-A5000` means that an area smaller than 5000 km^2 will not be plotted.

`-Ggray` specifies the color of dry areas to be gray.

`--MAP_FRAME_PEN=0.2p` sets the map's frame width to be 0.2p.

7. gmt inset end to terminate the inset plotting



8. gmt plot java_trench.txt -W1p,white to plot trench locations

`java_trench.txt` contains two columns of data, the first column is longitude and the second column is latitude. Each row means a sample point along the trench line.

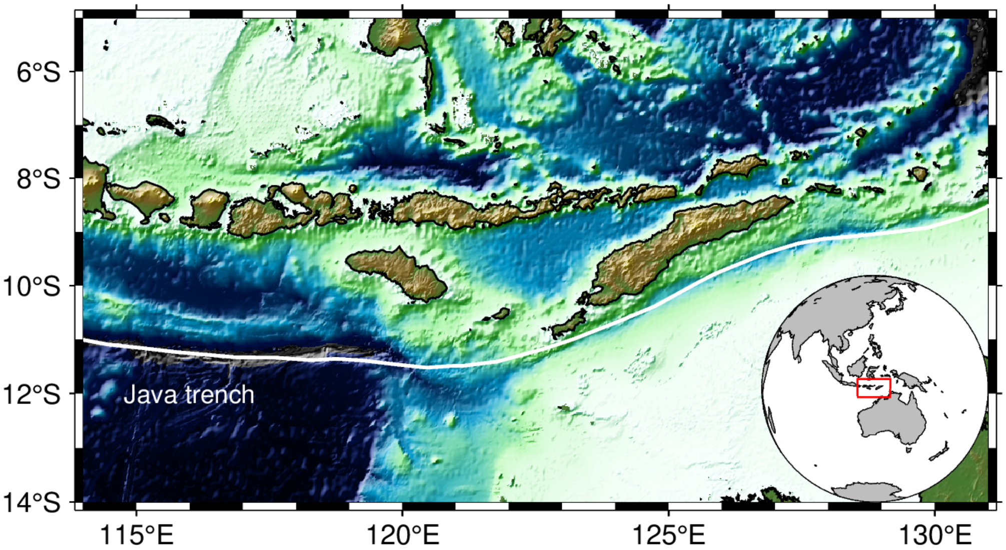
`-W1p,white` specifies the line width and color to be 1p and white.

9. echo '116 -12 Java trench' | gmt text -F+f8p,white to add text

`116 -12` specifies the location.

`Java trench` is the text being added.

`-F+f8p,white` specfies the fontsize and color of text.



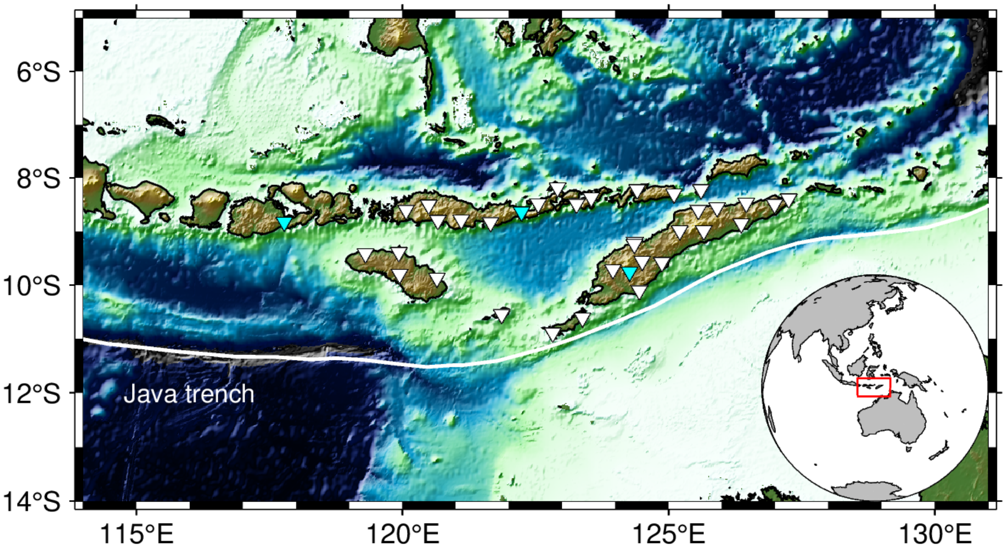
10. awk 'NR>1 {print $5,$6}' GE_3_stations.txt | gmt plot -Si6p -W0.1p,black -Gcyan to plot seismic stations

`Si6p` specifies the symbols to be inverted triangles (i) and their size is 6p.

`-Gcyan` specifies fill color of the symbols.

`-W0.1p,black` specifies the outline properties of symbols.

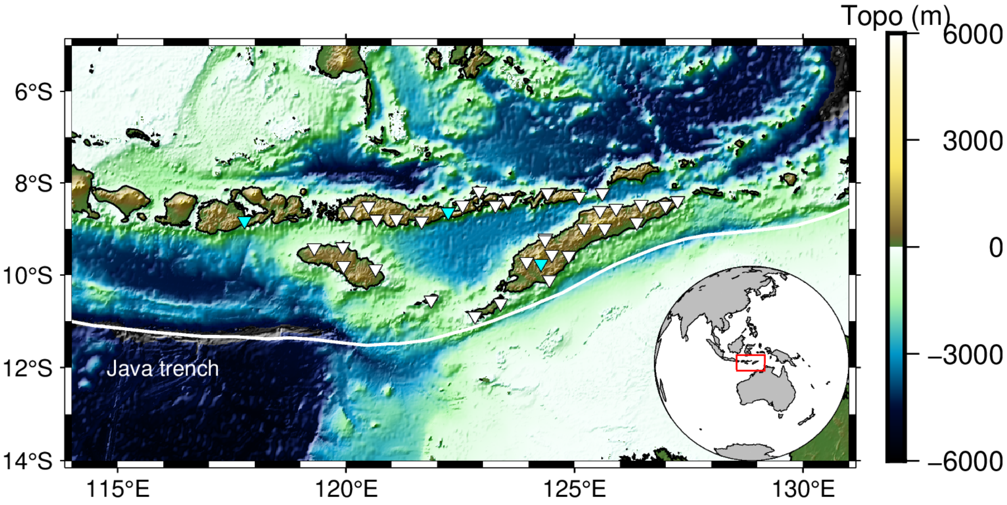
11. awk -F"|" 'NR>3 {print $6,$5}' YS_30_stations.txt | gmt plot -Si6p -W0.1p,black -Gwhite to plot more seismic stations



12. gmt colorbar -Dn1/0+o0.5c/0c+jBL+w5.5c -Bxa3000 -By+l"Topo (m)" -Celevation.cpt to add a color bar

`+o0.5c/0c` means the anchor point in the main figure is further moved for 0.5c along the x direction.

`-By+l"Topo (m)"` adds a label along the y axis.



CHAPTER

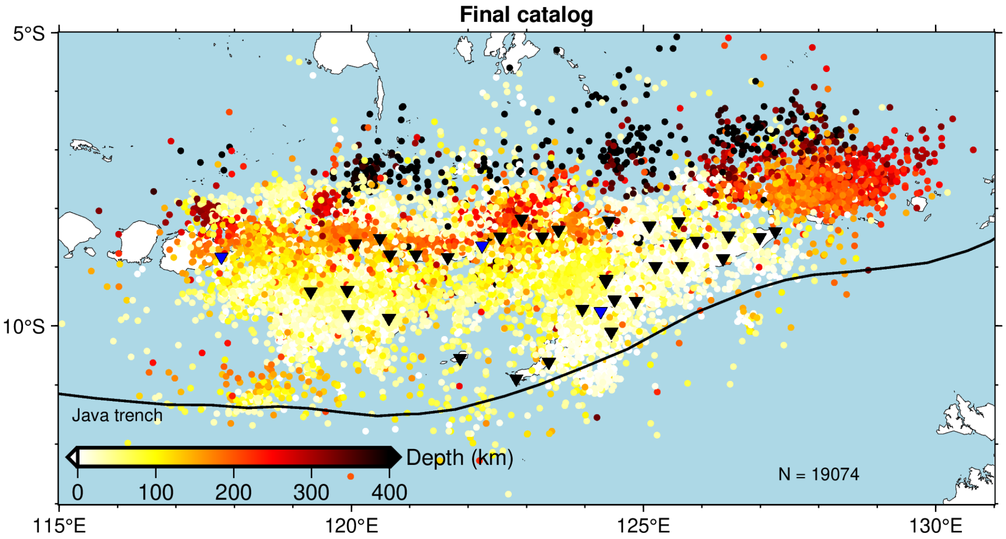
FOUR

GENERIC MAPPING TOOLS - SEISMIC DATA

4.1 Plotting earthquake catalog

4.1.1 Preview

The figure below is modified from Figure 2 in this paper.



It is generated using the following commands:

```
#!/bin/sh
gmt begin catalog png
# the frame
gmt basemap -JM15c -R115/131/-13/-5 -Bxa5f1 -Bya5f1 -BWeSn+t"Final catalog" --MAP_FRAME_
↪TYPE=plain --FONT_TITLE=10p --MAP_TITLE_OFFSET=-8p
# plotting coastline with specify land and sea colours
gmt coast -Gwhite -Slightblue -W0.1p,black -Da
# creating customized colormaps
gmt makecpt -Chot -T0/400/10 -D -Z -Ic -H > depth.cpt
awk '{print $9,$8,$10}' banda_arc_catalog.txt > catalog.xyz

# plotting the earthquake data
gmt plot catalog.xyz -Sc0.1c -Cdepth.cpt
# plotting the trench
```

(continues on next page)

(continued from previous page)

```
gmt plot java_trench.txt -W1p,black
# extracting station data and plot them
awk 'NR>1 {print $5,$6}' GE_3_stations.txt |gmt psxy -Si7p -W0.01p,black -Gblue
awk -F"|" 'NR>3 {print $6,$5}' YS_30_stations.txt |gmt psxy -Si7p -W0.01p,black -Gblack
# creating a colorbar
gmt colorbar -DjBL+h+o0.3c/0.6c+jBL+w5c/0.3c+e -By+l"Depth (km)" -Bxa100 -Cdepth.cpt
# plotting text
echo "128 -12.5 N = 19074" | gmt text -F+f8p,black
echo "116 -11.5 Java trench" | gmt text -F+f8p,black
gmt end
```

To reproduce it by yourself, you may first download or save `banda_arc_catalog.txt`, `java_trench.txt`, `GE_3_stations.txt`, `YS_30_stations.txt`, and then move these files into your working directory. Try to copy the above commands and run them on your own computer to see if you can generate the same figure without warning or error.

4.1.2 Step-by-Step explanation

1. gmt basemap -JM15c -R115/131/-13/-5 -Bxa5f1 -Bya5f1 -BWeSn+t”Final catalog” –MAP_FRAME_TYPE=plain –FONT_TITLE=10p –MAP_TITLE_OFFSET=-8p to plot base maps and frames

`-JM15c` specifies the map projection type to be Mercator projection. The width of map is 15c (15 centimeters).

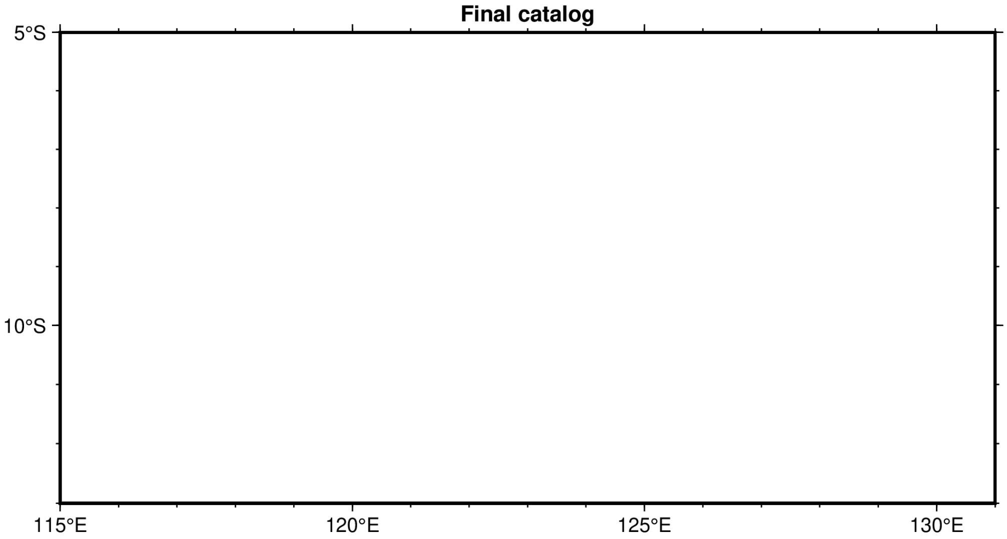
`-R115/131/-13/-5` specifies the map range, minimum and maximum longitudes are 115 and 131, minimum and maximum latitudes are -13 -5.

`-Bxa5f1` specifies that x axes have ticklabels with interval of 5 and ticks with interval of 1.

`-Bya5f1` specifies that y axes have ticklabels with interval of 5 and ticks with interval of 1.

`-BWeSn+t"Final catalog"` specifies that the left and bottom ticklabels are visible, which the right and top ticklabels are invisible, where the `+t"Final catalog"` indicates plotting the title “Final catalog”

`--MAP_FRAME_TYPE=plain --FONT_TITLE=10p --MAP_TITLE_OFFSET=-8p` are the gmt settings. `--MAP_FRAME_TYPE=plain` specifies the frame type as plain(i.e., simple line); `--FONT_TITLE=10p` specifies the font of title to be 10p; `--MAP_TITLE_OFFSET=-8p` specifies the distance between the title with the frame to be -8p.



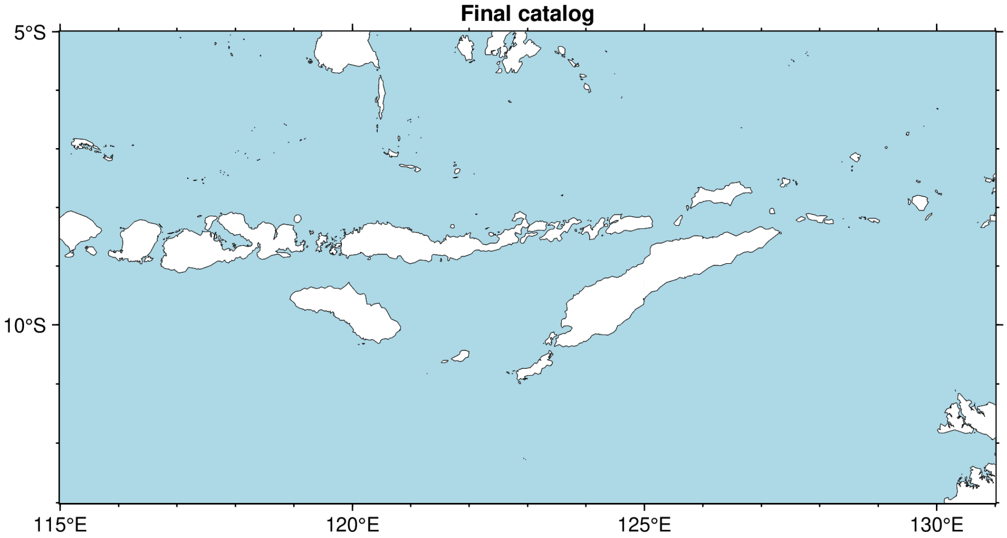
2. gmt coast -Gwhite -Slightblue -W0.1p,black -Da to plot coastline and specify land and sea colours

`-Gwhite` specifies fill the dry/land area with white.

`-Slightblue` specifies fill the wet/sea/lake area with lightblue.

`--W0.1p,black` specifies the line with a witdh of 0.1p and line color of black.

`-Da` specifies automatically selects the appropriate data precision based on the size of the current drawing area



3. gmt makecpt -Chot -T0/400/10 -D -Z -Ic -H > depth.cpt to make color palette tables

`-Chot` specifies the input cpt used is hot

`-D` selects the back- and foreground colors for the colorbar

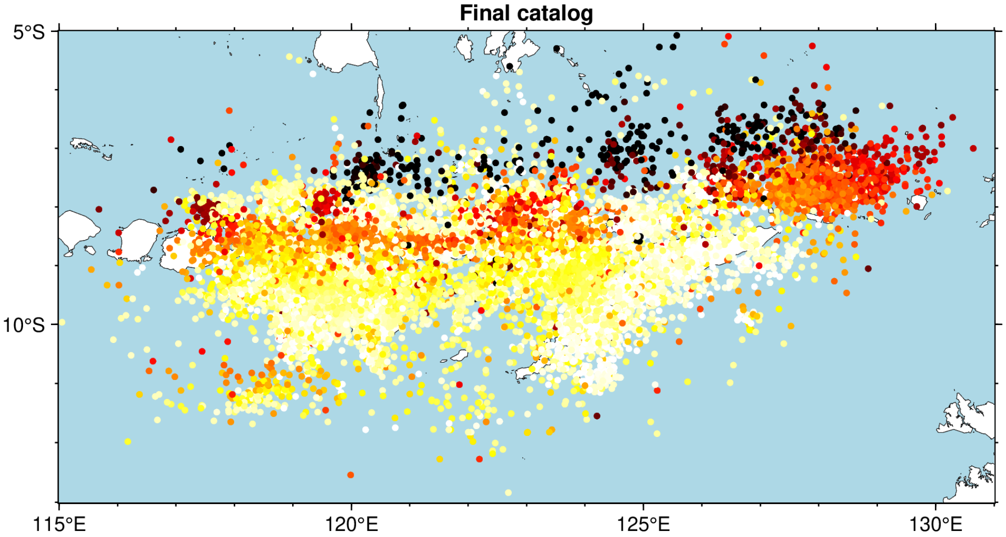
`-Z` creates a continuous cpt file

`-Ic` reverse the CPT

4. awk '{print $9,$8,$10}' banda_arc_catalog.txt > catalog.xyz

5. gmt plot catalog.xyz -Sc0.1c -Cdepth.cpt to plot earthquake data

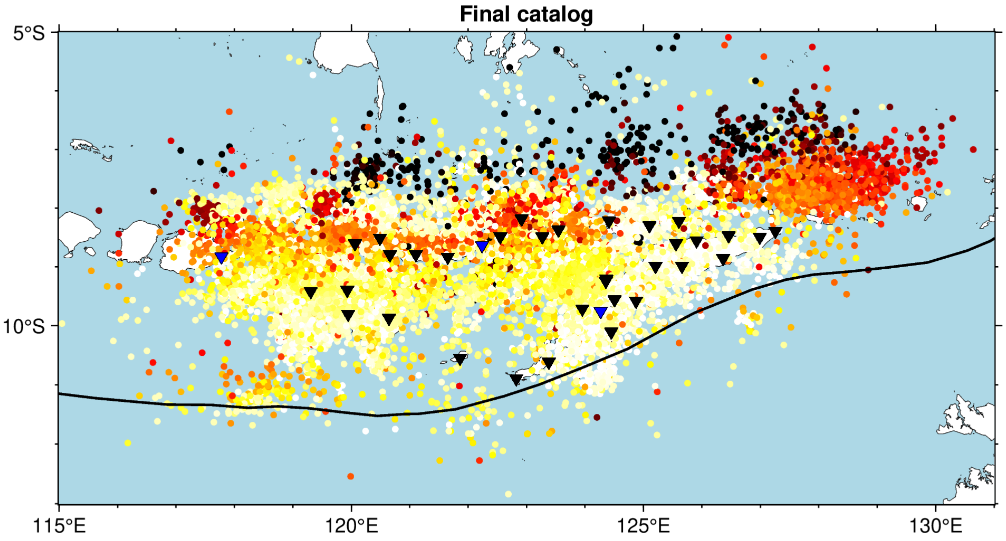
`catalog.xyz` contains three columns of data. longitude, latitude, and depth. The value of depth column will be used for coloring points based on CPT file.



6. gmt plot java_trench.txt -W1p,black to plot the trench line.

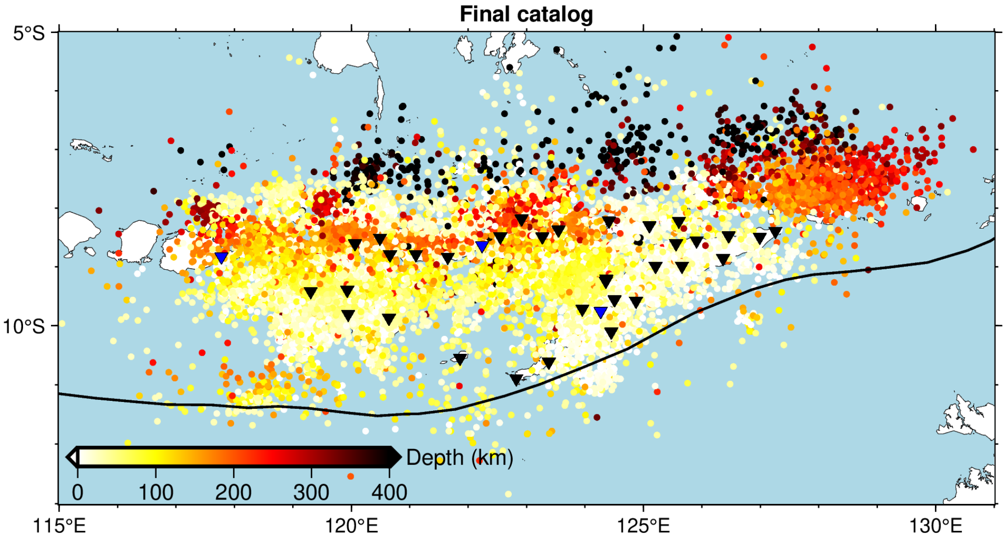
7. awk 'NR>1 {print $5,$6}' GE_3_stations.txt |gmt psxy -Si7p -W0.01p,black -Gblue to extract station data and plot them

8. awk -F"|" 'NR>3 {print $6,$5}' YS_30_stations.txt |gmt psxy -Si7p -W0.01p,black -Gblack to extract station data and plot them



9. gmt colorbar -DjBL+h+o0.3c/0.6c+jBL+w5c/0.3c+e -By+l"Depth (km)" -Bxa100 -Cdepth.cpt to plot a colorbar

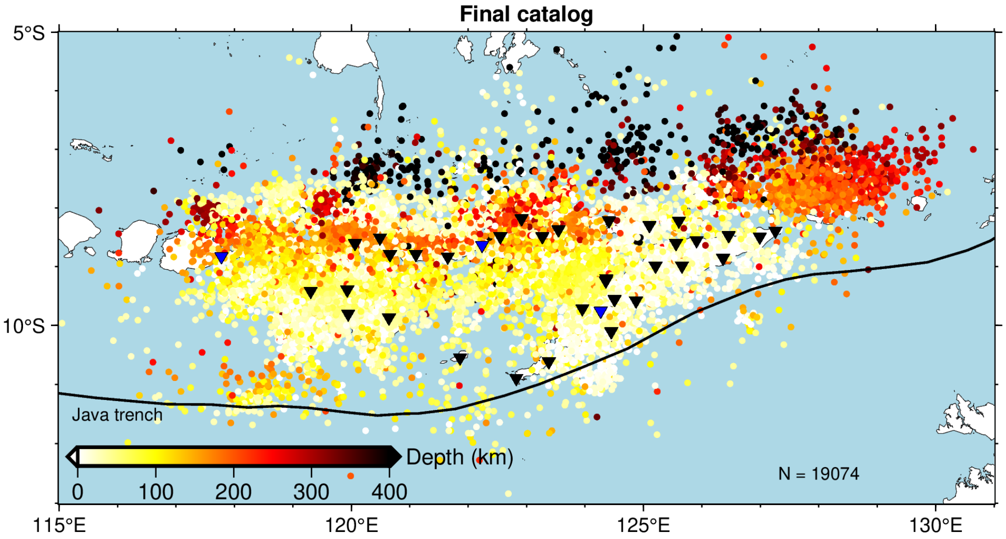
`-DjBL+h+o0.3c/0.6c+jBL+w5c/0.3c+e` specifices the paramter of colorbar. `-DjBL` means plot color at the Bottom Left; `+h` means draw horizontal color scale; `+o0.3c/0.6c` means plot move the colorbar 0.3 cm in X direction and 0.6 cm in Y Direction; `+w5c/0.3c` means plot a colorbar with a length of 5 cm and a width of 0.3 cm; `+e` means add a triangle to the foreground and background colors in the colorbar.



10. echo "128 -12.5 N = 19074" | gmt text -F+f8p,black to plot text

11. echo "116 -11.5 Java trench" | gmt text -F+f8p,black to plot text

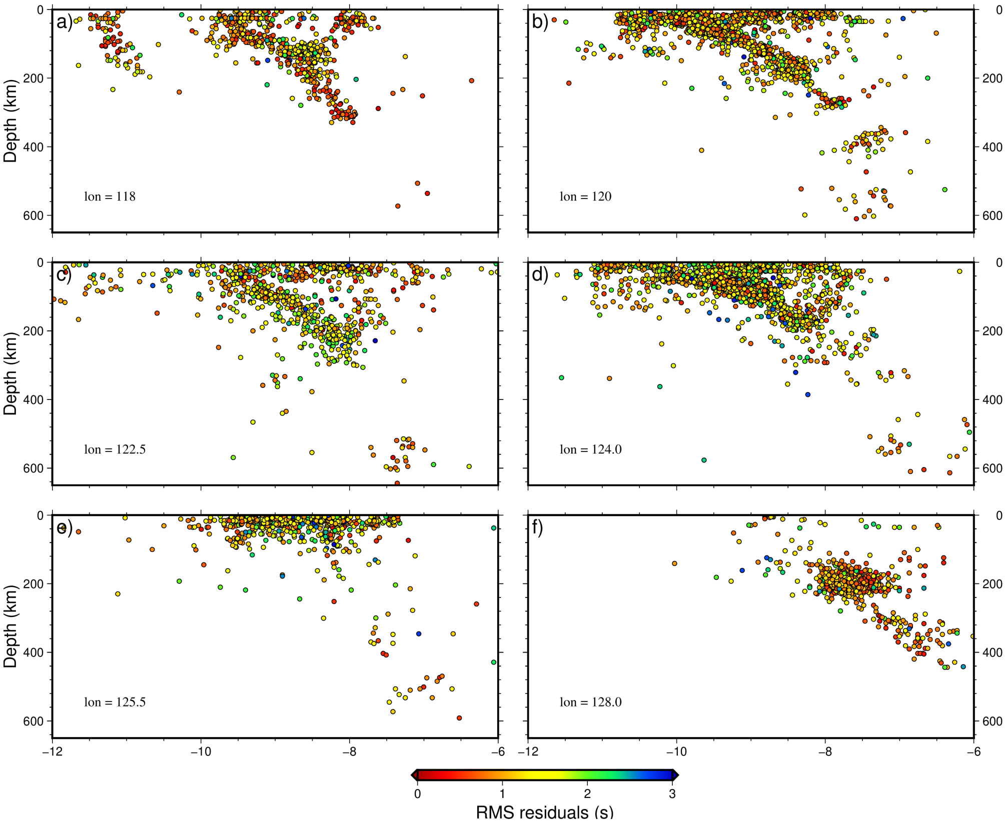
`-F+f8p,black` specifices the font size of 8p and color of black



4.2 Plotting cross sections

4.2.1 Preview

The figure below is modified from Figure 3 in this paper.



It is generated using the following commands:

```
#!/bin/sh
# extract data ignoring header, in order : lon, lat, depth, residual
awk 'NR>1 {print $9,$8,$10,$7}' banda_arc_catalog.txt > extracted.txt

gmt begin section png
gmt makecpt -Cseis -T0/3/0.1 -D -Z -H > res.cpt

gmt subplot begin 3x2 -Fs14c/7c -A
gmt subplot set 0 # transect along lon = 118
# project the data within 0.5 degree onto plane
# output file in order: latitude , depth , residual
gmt project extracted.txt -C118/-12 -E118/-6 -Lw -W-0.5/0.5 -Fyz > projected_input.txt
gmt plot projected_input.txt -JX14c/-7c -R-12/-6/0/650 -BWesn -Bya200f40+l"Depth (km)" -
↪Bxa2f0.5 -Sc4p -W0.5p -Cres.cpt # ploting
echo "lon = 118" | gmt text -F+cBL+f12p,4,black -Dj1c/1c # adding text

gmt subplot set 1 # transect along lon = 120
# project the data within 0.5 degree onto plane
# output file in order: latitude , depth , residual
gmt project extracted.txt -C120/-12 -E120/-6 -Lw -W-0.5/0.5 -Fyz > projected_input.txt
gmt plot projected_input.txt -JX14c/-7c -R-12/-6/0/650 -BwEsn -Bya200f40 -Bxa2f0.5 -Sc4p␣
↪-W0.5p -Cres.cpt # ploting
echo "lon = 120" | gmt text -F+cBL+f12p,4,black -Dj1c/1c # adding text

gmt subplot set 2 # transect along lon = 122.5
# project the data within 0.5 degree onto plane
# output file in order: latitude , depth , residual
gmt project extracted.txt -C122.5/-12 -E122.5/-6 -Lw -W-0.5/0.5 -Fyz > projected_input.
↪txt
gmt plot projected_input.txt -JX14c/-7c -R-12/-6/0/650 -BWesn -Bya200f40+l"Depth (km)" -
↪Bxa2f0.5 -Sc4p -W0.5p -Cres.cpt # ploting
echo "lon = 122.5" | gmt text -F+cBL+f12p,4,black -Dj1c/1c # adding text

gmt subplot set 3 # transect along lon = 124.0
# project the data within 0.5 degree onto plane
# output file in order: latitude , depth , residual
gmt project extracted.txt -C124.0/-12 -E124.0/-6 -Lw -W-0.5/0.5 -Fyz > projected_input.
↪txt
gmt plot projected_input.txt -JX14c/-7c -R-12/-6/0/650 -BwEsn -Bya200f40 -Bxa2f0.5 -Sc4p␣
↪-W0.5p -Cres.cpt # ploting
echo "lon = 124.0" | gmt text -F+cBL+f12p,4,black -Dj1c/1c # adding text

gmt subplot set 4 # transect along lon = 125.5
# project the data within 0.5 degree onto plane
# output file in order: latitude , depth , residual
gmt project extracted.txt -C125.5/-12 -E125.5/-6 -Lw -W-0.5/0.5 -Fyz > projected_input.
↪txt
gmt plot projected_input.txt -JX14c/-7c -R-12/-6/0/650 -BWeSn -Bya200f40+l"Depth (km)" -
↪Bxa2f0.5 -Sc4p -W0.5p -Cres.cpt # ploting
echo "lon = 125.5" | gmt text -F+cBL+f12p,4,black -Dj1c/1c # adding text

gmt subplot set 5 # transect along lon = 128.0
# project the data within 0.5 degree onto plane
```

(continues on next page)

(continued from previous page)

```
# output file in order: latitude , depth , residual
gmt project extracted.txt -C128.0/-12 -E128.0/-6 -Lw -W-0.5/0.5 -Fyz > projected_input.
↪txt
gmt plot projected_input.txt -JX14c/-7c -R-12/-6/0/650 -BwESn -Bya200f40 -Bxa2f0.5 -Sc4p␣
↪-W0.5p -Cres.cpt # ploting
echo "lon = 128.0" | gmt text -F+cBL+f12p,4,black -Dj1c/1c # adding text

gmt subplot end

gmt colorbar -DJBC+e+w8c+o1c -Cres.cpt -Bxa1+L"RMS residuals (s)"
gmt end
```

To reproduce it by yourself, you may first download or save `banda_arc_catalog.txt` and then move this files into your working directory. Try to copy the above commands and run them on your own computer to see if you can generate the same figure without warning or error.

You may go to the official tutorial website of GMT v6.3 **for more exploring**

4.3 Excercises

4.3.1 Reproduce figure 1 in the paper

1. include the base map and other samples (including the station, plate boundary, subduct direction arrows, and so on)
2. plot the station name, filled the station samples by yellow, and the station list in Unix command by red.
3. add a scale to the figure.
4. plot the map view cross-section, mark two ends of it with "A" and "A'". The cross-section should cross through the "MMRI" station, and with a length of 300 km, strike 30 degrees west of north

4.3.2 Plot a cross-section plot based on the catalog generated in the Unix command tutorial

1. project the earthquake within 30 km to the cross-section.
2. Scale the circles by earthquake magnitude and filled the circle according to their depth.
3. marked "A" and "A'" in the figure.

4.3.3 Plot the magnitude variation figure. Refer to Figure 4b in the paper

1. used stars to represent 10 maximum magnitude earthquakes. and label the magnitude of the largest one.
2. filled the circle according to their depth.

CHAPTER

FIVE

SEISMIC DATA REQUEST VIA OBSPY

5.1 Brief introduction

5.1.1 What is ObsPy?

ObsPy is an open-source project dedicated to provide a **Python** framework for **processing seismological data**. It provides parsers for common file formats, clients to access data centers and seismological signal processing routines which allow the manipulation of seismological time series (copied from ObsPy Github page).

ObsPy

A Python Framework for Seismology

We assume that you already have some experience of using Python. If not, you are suggested to read this small, incomplete introduction to the Python programming language.

5.1.2 How to install ObsPy

Note:

It’s strongly recommended to install ObsPy via conda.
We here assume you have already installed conda on your computer after finishing the previous GMT tutorial.
If not, please first install it (suggest installation of miniconda).

Open your terminal and run the following commands.

```
$ conda create --name obspy
$ conda activate obspy
$ conda install obspy=1.2
```

Warning: Exclude $ sign and start without whitespace!

5.1.3 Contents of this tutorial

We will introduce how to request, read, visualize, and further process seismic data using a few basic `functions` in the ObsPy `module`. it includes:

1. UTC DateTime
2. Basic Seismic Data Processing
3. Theoreotical Travel Time Calculation
4. Cross-section Plot with TauP arrivals

Developed by LAU Tsz Lam Zoe under the instructions of Junhao SONG, Han CHEN, and Suli Yao.

5.2 1 UTC Date Time

Now let's introduce the UTC DateTime format.

The UTC DateTime is a Coordinated Universal Time. Usually we could see that HKT (UTC+8) or BJT (UTC+8), which means that Hong Kong or Beijing time is 8 hours earlier than UTC time. We usually use UTC Datetime to present the origin time of an earthquake. Seismic time-series data like digital seismograms also use UTC Datetime to present the time of each sample.

5.2.1 1.1 DateTime Initialization

First in the terminal, type `python` and then type `enter`:

```
## Method1
>>>from obspy import UTCDateTime   ## import the module
>>>year = 2022
>>>month = 1
>>>day = 7
>>>hour = 17
>>>minute = 45
>>>second = 30.0
>>>UTCDateTime(year, month, day, hour, minute, second)  ## make the UTCDateTime object␣
↪according to the argument
UTCDateTime(2022, 1, 7, 17, 45, 30)

##Method 2
>>>UTCDateTime("2012-09-07T12:15:00")
UTCDateTime(2012, 9, 7, 12, 15)
```

Note:

There are many ways to produce the UTCDateTime object.
Method 1 & 2 are examples. You can explore others here.

5.2.2 1.2 DateTime Attribute Access

Now we can assign the UTCDateTime object to a variable "time".

```
>>>time = UTCDateTime("2012-09-07T12:15:00")
>>>print(time)
2012-09-07T12:15:00.000000Z
>>>print(type(time))
<class 'obspy.core.utcdatetime.UTCDateTime'>
```

Then, since it's a python class object, we can extract different time information by using UTCDateTime built-in functions/atttributes.

```
>>>print(time.year)  ## only output the year of "time"
2012
>>>print(time.julday)  ## output the Julian day of "time"
251
>>>print(time.timestamp) ## output the UNIX timestamp format of "time".
1347020100.0
>>>print(UTCDateTime("1970-01-01").timestamp)
0.0
```

Note:

Julian Day is a continuous count of days since the beginning of the year.
Simple example: What is the July of 1st Feb, 2022?
Ans: 32

The UNIX timestamp format means the number of seconds since the Epoch. The reference time is: "1970-01-01"

5.2.3 1.3 Handling time differences

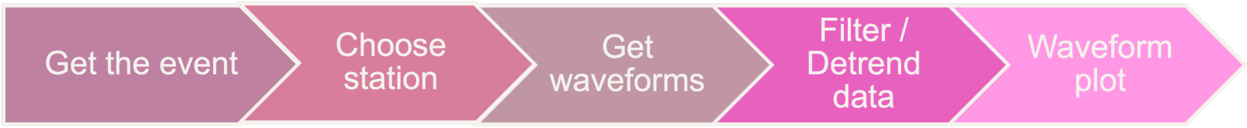
Calculate the time difference or add seconds into original "time"

```
>>>print(time - UTCDateTime("2012-09-07"))
44100.0
>>>time2 = time + 3600
>>>print(time2)
2012-09-07T13:15:00.000000Z
```

Clearly, we can see that "time2" is 1 hour (3600 seconds) later than "time".

5.3 2 Seismic data acquisition and visualization

5.3.1 Flow chart



5.3.2 2.1 Choose an event

You can select one event in the event list.

Note:

Here is the header information of the event list
indx year mon day time sec_relative_to_day res lat lon dep mag
9393 2015 08 11 16:22:15.200000 58935.2000 1.403 -8.624 123.202 171.9 3.9

Input the origin time, coordinates and magnitude of the selected event.

```
from obspy import UTCDateTime
origin_time = UTCDateTime("2015-08-11T16:22:15.200000")

# Coordinates and the magnitude of the event
eq_lon = 123.202
eq_lat = -8.624
eq_dep = 171.9
eq_mag = 3.9
```

5.3.3 2.2 Choose a station

Choose one station from the station list. Make sure the selected station is operating during the event.

Note:

Here is the header information of the station list.

Network | Station | Location | Channel | Latitude | Longitude | Elevation | Depth | Azimuth | Dip | SensorDescription | Scale | ScaleFreq | ScaleUnits | SampleRate | StartTime | EndTime

YS|BAOP||BHZ|-8.4882|123.2696|67.0|0.0|0.0|-90.0|Nanometrics Trillium 120 Sec Response/Taurus Stand|1.19642E9|0.3|m/s|50.0|2014-10-31T00:00:00|2016-12-31T23:59:59

5.3.4 2.3 Get waveforms

Import the web service providers and input station information.

```
from obspy.clients.fdsn import Client

# IRIS is one of those providers.
client = Client('IRIS')    ##to initialize a client object(as IRIS here)

# Input station informations
# network
net = 'YS'
# station
sta = 'BAOP'
# location
loc = ''
# channel
cha = 'BHZ'

# starttime
stt = origin_time
# endtime
edt = origin_time + 120

# Get the waveforms from client
st = client.get_waveforms(net, sta, loc, cha, stt, edt)  ## to get the waveform by the␣
↪corresponding argument from clients.
print(st)
```

Note:

FDSN web services for data access to different web service providers.

IRIS is one of the web service providers which is commonly used.

5.3.5 2.4 Meta data

We can print the meta data inside the stream.

```
print(st[0].stats)
```

```
                 network: YS
                 station: BAOP
                location:
                 channel: BHZ
               starttime: 2015-08-11T16:22:15.200000Z
                 endtime: 2015-08-11T16:24:15.200000Z
           sampling_rate: 50.0
                   delta: 0.02
                    npts: 6001
                   calib: 1.0
_fdsnws_dataselect_url: http://service.iris.edu/fdsnws/dataselect/1/query
                 _format: MSEED
                   mseed: AttribDict({'dataquality': 'M', 'number_of_records': 26,
↪'encoding': 'STEIM1', 'byteorder': '>', 'record_length': 512, 'filesize': 13312})
```

```
#You can print the corresponding attributes by calling them individually.
print(st[0].stats.sampling_rate)
```

```
50.0
```

`.stats` contains all header information of a Trace object.

Tip: There are some default attributes.

`sampling rate`	Sampling rate in hertz	`network`	Network code
`station`	Station code	`channel`	Channel code
`starttime`	UTCDateTime of the first data sample	`endtime`	UTCDateTime of the last data sample
`gcarc`	Epicentral distance	`baz`	Back azimuths

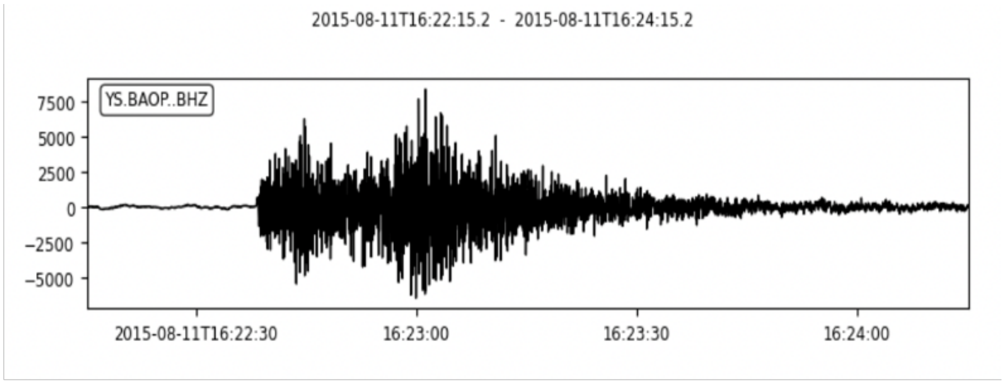
For `gcarc` and `bac` , they are available in sac file. You can print them by:

```
print(st[0].stats.sac.gcarc)

# If the header value is empty, you can assign value into the header.
st[0].stats.sac.gcarc = 10000
```

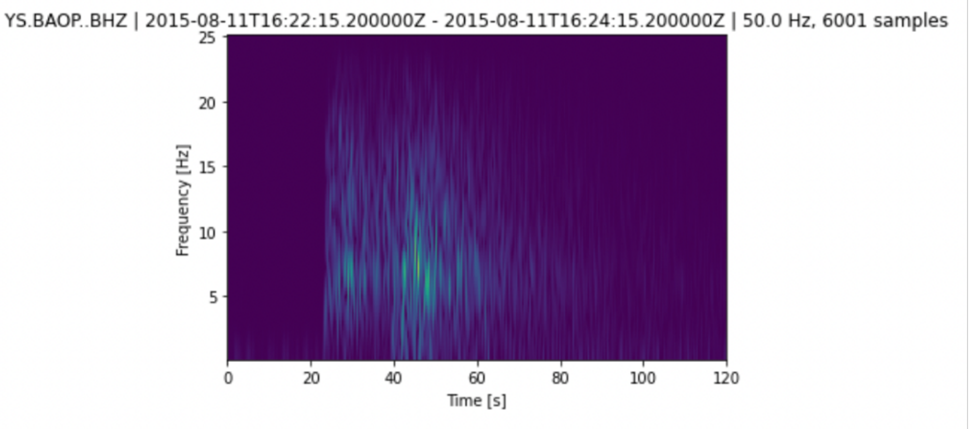
5.3.6 2.5 Plot the waveforms

Here we plot the waveforms without any preprocessing procedure.

```
st.plot();
```



```
st.spectrogram();
```



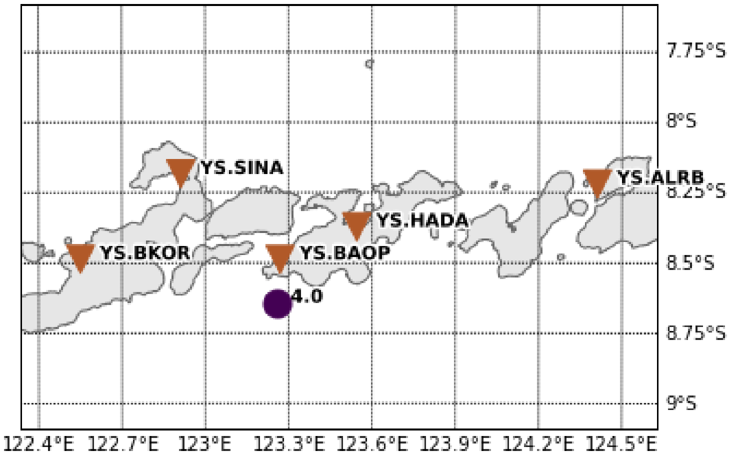
Note: Spectrogram is a frequency content of a seismogram. You can check the energy level of the waves over time.

5.3.7 2.6 Waveform Cross-section Plot

Plot a record section.

5.3.8 2.6.1 Get the waveform data with more than 1 station

For our example, station 'BAOP', 'HADA', 'SINA' 'BKOR' and 'ALRB' are located near the epicentre of the earthquake. It is expected that these 5 stations can record the event well.



```
# Set up a list for bulk request
bulk = [('YS', 'BAOP', '', 'BHZ', origin_time, origin_time+120),
        ('YS', 'HADA', '', 'BHZ', origin_time, origin_time+120),
        ('YS', 'SINA', '', 'BHZ', origin_time, origin_time+120),
        ('YS', 'BKOR', '', 'BHZ', origin_time, origin_time+120),
        ('YS', 'ALRB', '', 'BHZ', origin_time, origin_time+120)]

st_bulk = client.get_waveforms_bulk(bulk)
print(st)
```

`get_waveforms_bulk` send a bulk request for waveforms to the server

5.3.9 2.6.2 Calculate the great circle distance from stations to earthquake

```
# Input the coordinates of stations
ALRB_loc = [-8.2194, 124.4115]
BAOP_loc = [-8.4882, 123.2696]
BKOR_loc = [-8.4868, 122.5509]
HADA_loc = [-8.3722, 123.5454]
SINA_loc = [-8.1838, 122.9124]

from obspy.geodetics import gps2dist_azimuth

# Loop, get the station coordinates and calculate the distance
for tr in st_bulk:
    sta = tr.stats.station
    if sta == 'ALRB':
        sta_lat = ALRB_loc[0]
        sta_lon = ALRB_loc[1]
    if sta == 'BAOP':
        sta_lat = BAOP_loc[0]
        sta_lon = BAOP_loc[1]
    if sta =='BKOR':
        sta_lat = BKOR_loc[0]
        sta_lon = BKOR_loc[1]
    if sta =='HADA':
```

(continues on next page)

(continued from previous page)

```
        sta_lat = HADA_loc[0]
        sta_lon = HADA_loc[1]
    if sta =='SINA':
        sta_lat = SINA_loc[0]
        sta_lon = SINA_loc[1]

    tr.stats.distance = gps2dist_azimuth(sta_lat, sta_lon,eq_lat, eq_lon)[0]

# To check the result, you can print the distance with stations.
for tr in st_bulk:
   print(tr.stats.station, tr.stats.distance)
```

`gps2dist_azimuth` calculate the distance between two geographic points and forward and backward azimuths between these points

Note:

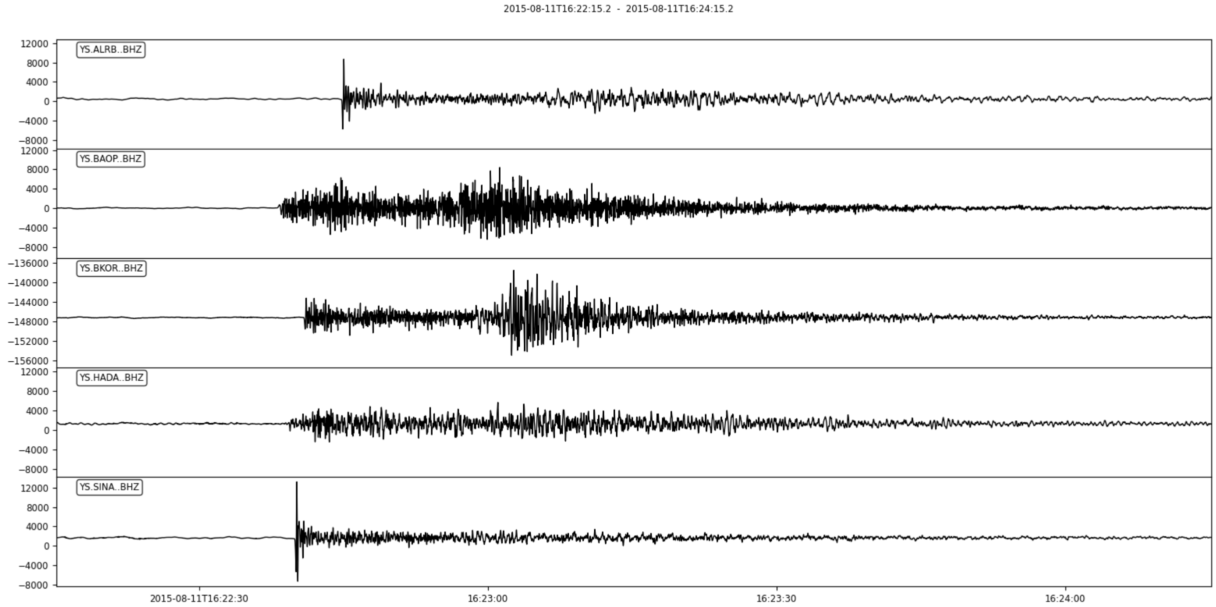
As the mseed file does not contain the location of station, we have to get the information from the station list.

We also have to save the calculated distance into the metadata, as the value is empty initially.

5.3.10 2.6.3 Plot the waveform cross-section plot

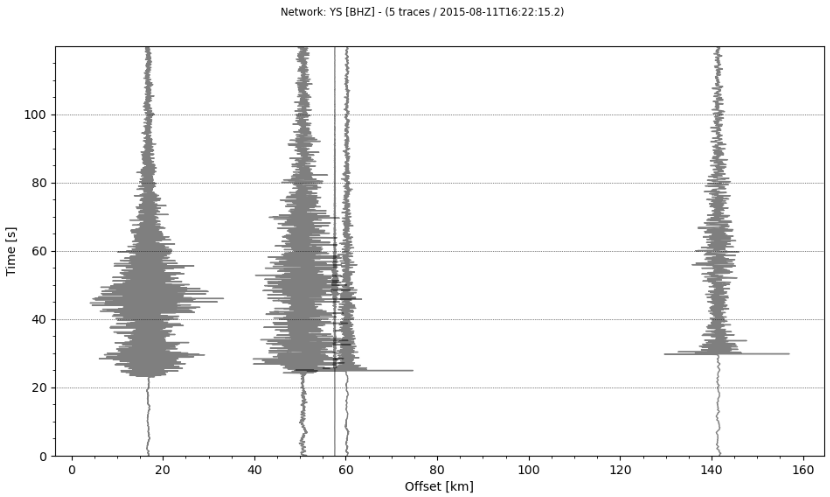
`stream.plot` allows us to plot the streams at the same figure. For example, we can plot all stream with the same X axis:

```
st_bulk.plot(size=(1600,800))  ## size=(1600,800) sets the figure size
```



We can also plot the streams in a cross-section view:

```
st_bulk.plot(type='section')  ## type='section' indicates a record section can be plotted
```



Note: We could add more features to the plot, for example, the phase arrivals, and the station names. It is a good way for us to recognize different seismic phases, We can also get the apparent velocity of P - and S - waves from the plot. We will introduce these in the next section.

CHAPTER

SIX

SEISMIC DATA PROCESS VIA OBSPY

6.1 1 Basic Seismic Data Processing

6.1.1 1.1 Detrend / Filter Data

`detrend()` is provided to remove a trend from the trace. There are many methods listed for detrend function (simple, linear, constant . . .), please refer to `obspy.core.trace.Trace.detrend`

To better visualize and demonstrate the effect of detrending, we will provide some examples with significant trends, and show the effect of detrend function.

1.1.1 Remove Mean

You can download the waveform file here. `PA01.bhy` We first read the downloaded waveform file.

```
from obspy import read
rmean_raw = read('PA01.bhy')  ## Read waveform files into an ObsPy Stream object.

# Plot the waveform without any processing and copy the stream.
rmean_raw.plot()
rmean_processed = rmean_raw.copy()  ## Copy the raw stream to the new one to be processed
```

Tip:

This operation is performed in place on the actual data arrays. If you want to see the difference before and after the processing of data. It is better to copy the stream as the processing will overwrite the original waveforms.

Now let's detrend the waveforms and plot it again.

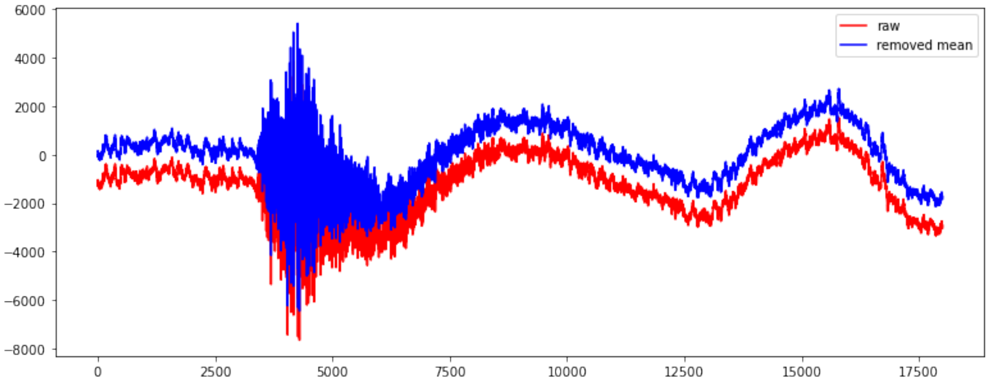
```
rmean_processed.detrend("demean")
rmean_processed.plot()

# You can compare the raw and processed waveforms by overlapping them.
import matplotlib.pyplot as plt

plt.figure(figsize=(13,5))
for tr in rmean:
    data1 = tr.data
plt.plot(data1,color='red',label='raw')

for tr in rmean_test:
```

(continues on next page)

(continued from previous page)

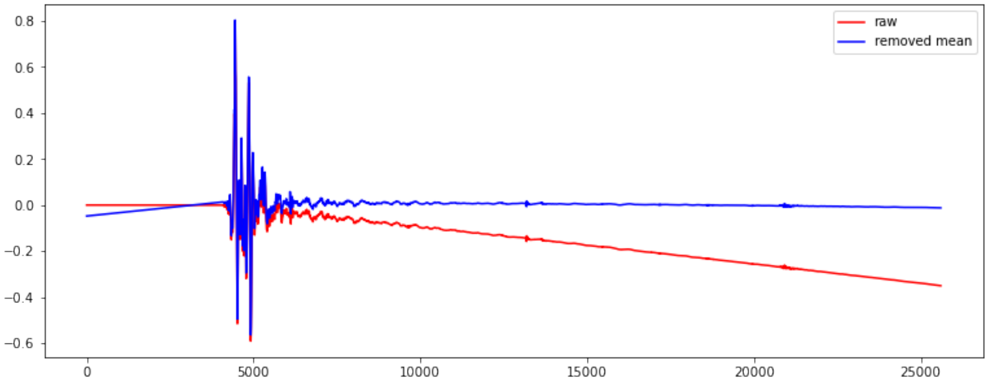
```
    data2 = tr.data
plt.plot(data2,color='blue',label='removed mean')

plt.legend()
```

Here is the result.



1.1.2 Remove Linear trend

You can download the waveform file here. `LLT.E.Vel.BF.SAC` The procedures are the same with remove mean, but use `.detrend("linear")` to remove linear trend. Follows is the result.



6.1.2 1.2 Filter Data

Filtering data removes the noise so that we can identify the phases easier in the filtered waveforms. You can apply different filters to extract the signal of interest from the raw data. `filter` function provides different filters. For example, `bandpass`, `highpass` and `lowpass`.

To further understand the effects of different filter, you can download the waveform file here `PA03.bhz` for practice.

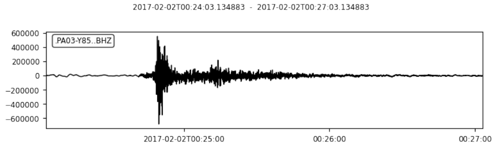
```
from obspy import read

# Read the waveform file
raw_data = read('./PA03.bhz')


# Plot the raw waveform
```

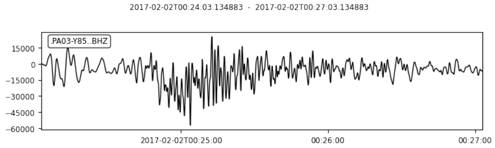
(continues on next page)

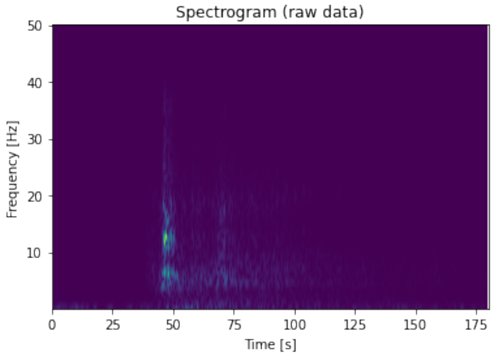
(continued from previous page)

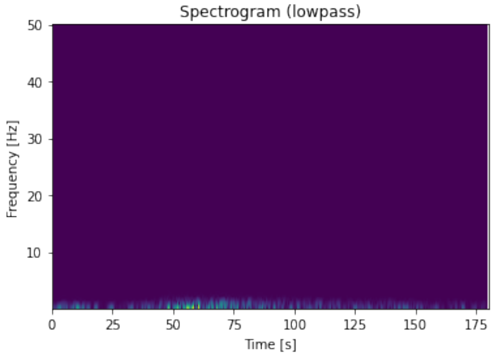
```
raw_data.plot()
raw_data.spectrogram()

# Copy the waveform for further processing
processed_low = raw_data.copy()

# Filter the waveform with lowpass filter
processed_low.filter("lowpass",freq=1)

# Plot the waveform and spectrogram to see the difference
processed_low.plot(starttime=start_time,endtime=start_time+80)
processed_low.spectrogram(title='lowpass')
```

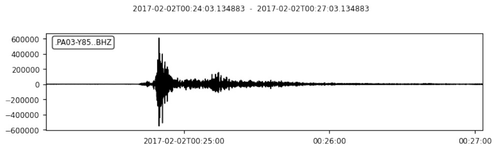
You can try with different filters using the above code. Here is the comparison using different filters

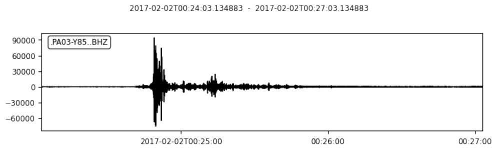


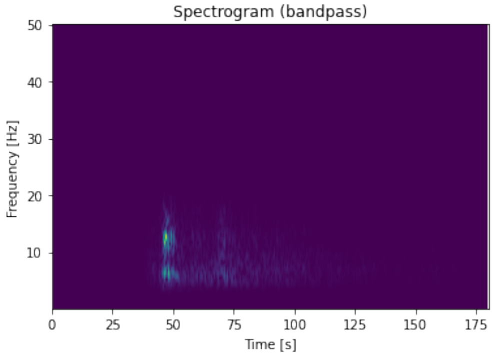


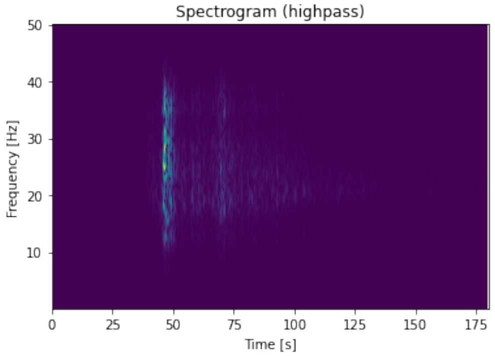












6.1.3 1.3 Waveform rotation

For better reconigizing particlar seismic waves, such as Love waves and Rayleigh waves, we will rotate a seismogram from the North - East coordinate to Radial- Transverse coordinate.

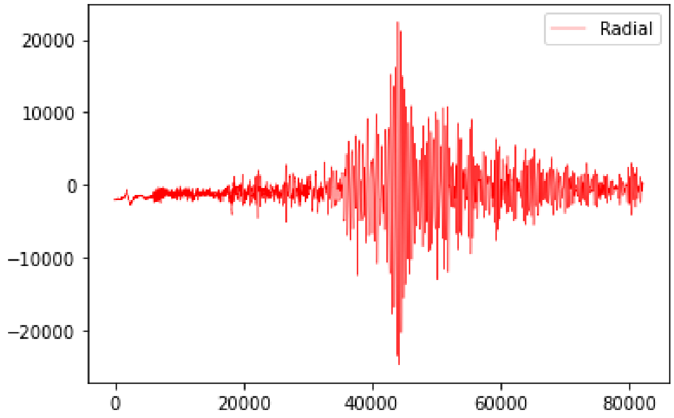
Here we provide a example. You can download the waveform file here. BINY.N and BINY.E

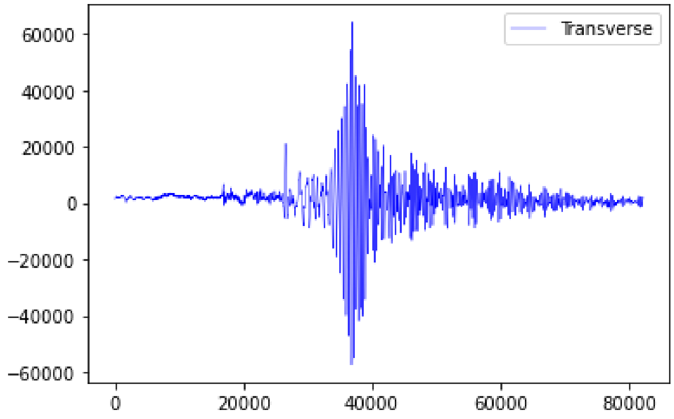
```
## import modules that needed in processing
from obspy import read
from obspy.signal import rotate
import matplotlib.pyplot as plt

# Read the North-East components
rotation_N = read('BINY.N')

# Read the South-West components
rotation_E = read('BINY.E')

```

(continues on next page)

(continued from previous page)

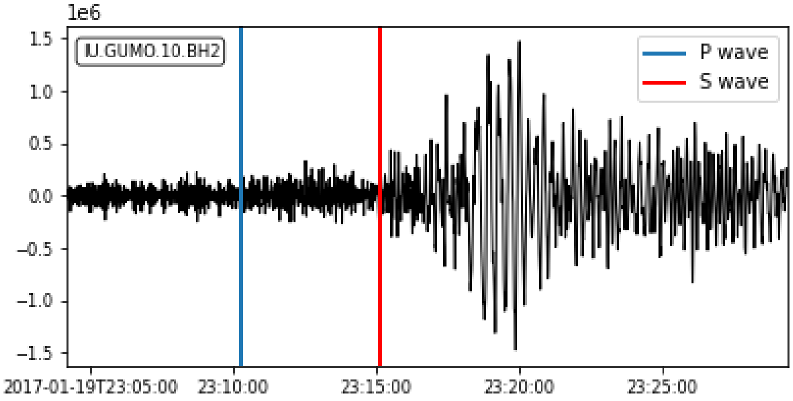
```
# print out the meta data
print(rotation_N[0].stats)

# The data of North -East components
north = rotation_N[0].data

east = rotation_E[0].data

# Get the back azimuth.
Baz = rotation_N[0].stats.sac.baz


# Rotate the data into radial and transverse components.
Radial, Transverse = rotate.rotate_ne_rt(north,east,Baz)  ## rotate waveforms from North - East components to radial and transverse component

# Plot the result
plt.plot(Radial)
plt.plot(Transverse)
```





6.1.4 1.4 Seismic Phases in seismogram

Many seismic phases can be presented in the seismogram. For beginners, we can focus on P - and S - waves. Here is the demonstration of picking P - and S - waves in a teleseismic earthquake.



We will introduce the picking method in next section!

6.2 2 TauP

TauP is a toolkit to calculate the seismic travel time calculator. It handles many types of velocity models and calculate times for virtually any seismic phase with a phase parser.

Note: Seismic velocity model is the velocity profile of P and S waves along depth. IASP91 model is commonly used.

TauP can provide us a reference for identifying different phases. We can also compare it with the real arrivals, the difference between actual and theoretical arrival may interpret as a site effect.

6.2.1 2.1 Source Configuration

Input the information of the source (earthquake)

```
eq_lat = -8.624
eq_lon = 123.202
eq_dep = 171.9
```

6.2.2 2.2 Receiver Configuration

Input the information of the receiver (station)

```
sta_lat = -8.4882
sta_lon = 123.2696
```

6.2.3 2.3 Travel Time Calculation

There are 2 methods to calculate the travel time.

6.2.4 Method 1

```
from obspy.taup import TauPyModel
# Import the velocity model
model = TauPyModel(model="iasp91")

from obspy.geodetics import locations2degrees

for tr in st:
    # calculate the distance in degree between the source and receiver
    deg_distance = locations2degrees(sta_lat, sta_lon,eq_lat, eq_lon)
    print(deg_distance)


    # Get the arrivals using the model configured
    arrivals = model.get_travel_times(source_depth_in_km=eq_dep, distance_in_degree=deg_
↪distance, )
```

`location2degrees` calculate the great circle distance between 2 points on a spherical earth

`model.get_travel_times` get the travel times of the phases

6.2.5 Method 2

```
p_arrival,s_arrival = model.get_travel_times_geo(source_depth_in_km=eq_dep,
                                        source_latitude_in_deg=eq_lat,
                                        source_longitude_in_deg=eq_lon,
                                        receiver_latitude_in_deg=float(sta_lat),
                                        receiver_longitude_in_deg=float(sta_lon),
                                        phase_list=["p","s"])
```

`model.get_travel_times_geo` get the travel times of the phases given geographical data

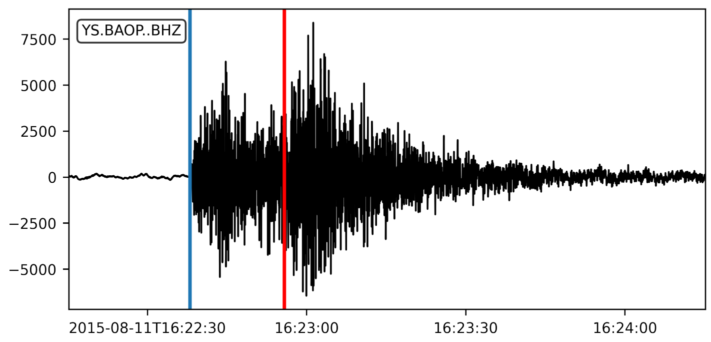
Then you can get the travel time of P - and S waves.

```
print(p_arrival, "\n", s_arrival)

#Output P - and S waves arrival time(s)
print(p_arrival.time, s_arrival.time)
```

6.2.6 2.4 Visualise the result

Plot the theoretical travel time onto the waveform.

```
# Import matplotlib module
import matplotlib.pyplot as plt
from matplotlib.dates import date2num

# Make figure
fig = plt.figure()
st.plot(fig=fig)

# Axis of the plot
ax = fig.axes[0]
# Add vertical line across the axes
ax.axvline(date2num((origin_time+p_arrival.time).datetime),lw=2)
ax.axvline(date2num((origin_time+s_arrival.time).datetime),lw=2,color='r')
plt.show()
fig.savefig('taup_single_waveform.png',dpi=500)
```



6.3 3 Section Plot

6.3.1 3.1 Waveform cross-section plot

We have introduce how to make a waveform cross-section plot in 2.6. For this section, we would like to add the calculated TauP arrivals onto to waveform cross-section plot.

```
bulk = [('YS', 'BAOP', '', 'BHZ', origin_time, origin_time+120),
        ('YS', 'HADA', '', 'BHZ', origin_time, origin_time+120),
        ('YS', 'SINA', '', 'BHZ', origin_time, origin_time+120),
        ('YS', 'BKOR', '', 'BHZ', origin_time, origin_time+120),
        ('YS', 'ALRB', '', 'BHZ', origin_time, origin_time+120)]
```

(continues on next page)

(continued from previous page)

```
st = client.get_waveforms_bulk(bulk)
print(st)

# Input the coordinates of stations
ALRB_loc = [-8.2194, 124.4115]
BAOP_loc = [-8.4882, 123.2696]
BKOR_loc = [-8.4868, 122.5509]
HADA_loc = [-8.3722, 123.5454]
SINA_loc = [-8.1838, 122.9124]

# Loop, get the station coordinates and calculate the distance
for tr in st:
    sta = tr.stats.station
    if sta == 'ALRB':
        sta_lat = ALRB_loc[0]
        sta_lon = ALRB_loc[1]
    if sta == 'BAOP':
        sta_lat = BAOP_loc[0]
        sta_lon = BAOP_loc[1]
    if sta =='BKOR':
        sta_lat = BKOR_loc[0]
        sta_lon = BKOR_loc[1]
    if sta =='HADA':
        sta_lat = HADA_loc[0]
        sta_lon = HADA_loc[1]
    if sta =='SINA':
        sta_lat = SINA_loc[0]
        sta_lon = SINA_loc[1]

    tr.stats.distance = gps2dist_azimuth(sta_lat, sta_lon,eq_lat, eq_lon)[0]

# To check the result, you can print the distance with stations.
for tr in st:
   print(tr.stats.station, tr.stats.distance)
```

6.3.2 3.2 TauP travel time

```
from obspy import taup

# velocity model configuration
model = taup.TauPyModel(model="iasp91")

p_time = []
s_time = []
sta = []
for tr in st_bulk:
    # Get the station location for the input
    station_coordinate = str(tr.stats.station)+"_loc"

    p_arrival ,s_arrival = model.get_travel_times_geo(source_depth_in_km=eq_dep,
```

(continues on next page)

(continued from previous page)

```
                                               source_latitude_in_deg=eq_lat,
                                               source_longitude_in_deg=eq_lon,
                                               receiver_latitude_in_
↪deg=float(eval(station_coordinate)[0]),
                                               receiver_longitude_in_
↪deg=float(eval(station_coordinate)[1]),
                                               phase_list=["p","s"])
    print(p_arrival, s_arrival)
    # Append lists by stations, p & s arrivals
    sta.append(tr.stats.station)
    p_time.append(p_arrival.time)
    s_time.append(s_arrival.time)
```

The goal for us is to get the P - and S wave arrival of each station and save them into lists so that we can handle the result later.

6.3.3 3.3 Output the TauP result as text file for further processing

As we are handling the data with more than 1 station, it is better for us to save the TauP result in a txt file.

```
# Make a table using pandas and save it to the text file
import pandas as pd
# List to pandas.dataframe
Station = pd.DataFrame(sta)
P_arrival = pd.DataFrame(p_time)
S_arrival = pd.DataFrame(s_time)

# Combine the column together and make a table
tauP_result = pd.concat([Station, P_arrival, S_arrival], axis=1)

# Output the table as a text file
tauP_result.to_csv('taup_result.txt',sep=' ', index=False, header=False)
print(tauP_result[0])
```

Note:

Pandas is a python library which is used to analyse data.

`pd.DataFrame` Data structure

`pd.concat` concatenate pandas objects along a particular axis with optional set logic along the other axes

`pd.to_csv` write object to a comma-separated values (csv) file

6.3.4 3.4 Trim and filter data

```
# Trim the waveform data - shorter time range
st.trim(origin_time, origin_time + p_time[0]+150)
# Filter the waveform
st.detrend('linear')
st.filter('bandpass', freqmin=2, freqmax=15)
```

`trim()` cut all traces with given start time and end time

6.3.5 3.5 Add more components on your plot

Then you can add more components in the plot. For example, station name, calculated P - and S wave arrival time.

```
# Add more components onto the section plot
import numpy as np

ax = fig.axes[0]
# Add title
ax.set_title('Waveform cross-section plot')

# Add station names next to the waveforms
for tr in st:
    ax.text((tr.stats.distance / 1e3)+1, 1, tr.stats.station, rotation=270,va="top", ha=
↪"center", zorder=10)

# Load the tauP output text file
ps_pick = np.loadtxt('taup_result.txt', dtype=str)
# Mark the P & S arrival onto the waveform plot

for tr in st:
    # Find the P & S arrivals by stations
    sta = tr.stats.station
    print(ps_pick[ps_pick[:,0] == sta,1 ])

    # Y-axis
    p_pick = float(ps_pick[ps_pick[:,0] == sta,1 ])
    s_pick = float(ps_pick[ps_pick[:,0] == sta,2 ])
    # Offset (x-axis) in km
    offset = tr.stats.distance/1e3
    # Make the scatter plot
    ax.scatter(offset,p_pick, c ='b', marker = '_',s=150)
    ax.scatter(offset,s_pick, c ='r', marker = '_',s=150)
```

`axes.set_title` set a title for the axes

`axes.text` add text to the axes

`np.loadtxt` load the data from the text file

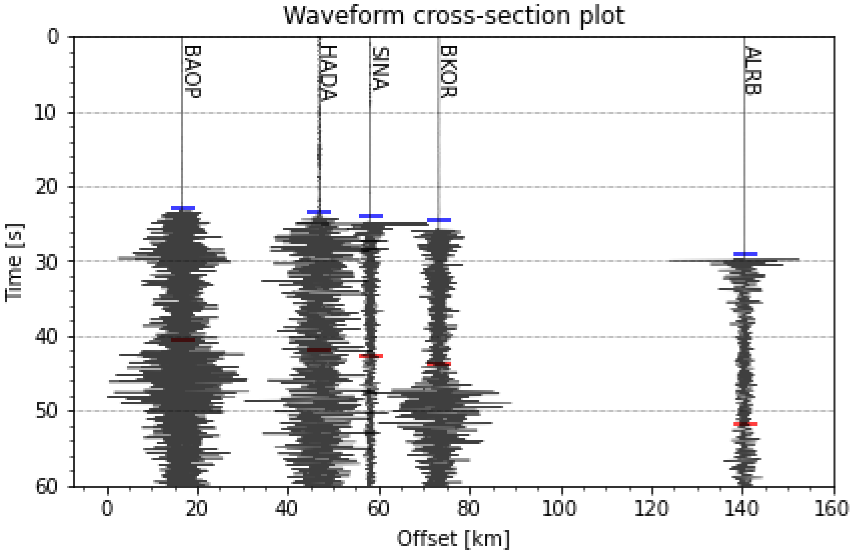
`axes.scatter` a scatter plot of y vs. X with varying marker size and/or colour

```
# plot again

```

(continues on next page)

(continued from previous page)

```
st.plot(type='section', recordstart=0, recordlength=60, time_down=True, linewidth=.5,␣
↪grid_linewidth=.5, show=False, fig=fig)

# Save the figure
# dpi = how many pixels the figure comprises
fig.savefig('section_plot.png',dpi=500)
```



The section plot is just a recap of the previous section. Let's have a try!!

6.4 4 Exercises

Here is the event information of a magnitude 7.7 earthquake occurred in 2018.

Note:

Origin time: 2017/07/17 23:34:13.870 (UTC)
Location (lat/lon/dep): 54.4715| 168.8148| 10.99
Magnitude: mww,7.7,us
Region: KOMANDORSKIYE OSTROVA REGION

1. The above is an earthquake with magnitude 7.3, try to find a station that was operating during the event and download the waveform data. (10 marks)
2. Visualize the waveforms and the frequency content of the phases (10 marks)
3. Make filter to highlight the phases of the seismic trace. and Plot the waveform again with clear P- and S-waves arrivals (20 marks)
4. Try to identify the Love and Rayleigh waves and estimate their arrivals. (30 marks)
5. Plot a cross-section with title(5) , station names(5) , P - and S - wave arrival(10). And estimate the apparent velocity of P and S wave(10). (30 marks)

CHAPTER

SEVEN

EARTHQUAKE DETECTION STA/LTA

Developer: Hui LIU, Earth Science System Program, CUHK

Testers: Zhangyu SUN & Ng Kai Yin, Earth Science System Program, CUHK

7.1 Brief Introduction

7.1.1 Why do we use STA/LTA?

By introducing digital seismic data acquisition, long-term continuous recording and archiving of seismic signals has become a demanding technical problem. A seismic network or even a single seismic station operating continuously at high sampling frequency produces an enormous amount of data, which is often difficult to store (and analyze) locally or even at the recording center of a network. This situation has forced seismologists to invent triggered seismic data acquisition. In a triggered mode, a seismic station or a seismic network still processes all incoming seismic signals in real time (or in near-real-time) but incoming data is not stored continuously and permanently. Processing software - a trigger algorithm - serves for the detection of typical seismic signals (earthquakes, controlled source seismic signals, underground nuclear explosion signals, etc.) in the constantly present seismic noise signal. Once an assumed seismic event is detected, recording and storing of all incoming signals starts. It stops after the trigger algorithm 'declares' the end of the seismic signal.

The short-time-average/long-time-average **STA/LTA** trigger is usually used in weak-motion applications that try to record as many seismic events as possible. These are the applications where the **STA/LTA** algorithm is most useful. It is nearly a standard trigger algorithm in portable seismic recorders, as well as in many real time processing software packages of the weak-motion seismic networks. However, it may also be useful in many strong motion applications, except when interest is limited to the strongest earthquakes.

7.1.2 How does STA/LTA works?

The **STA/LTA** algorithm processes filter seismic signals in two moving time windows – a short-time average window (STA) and a long-time average window (LTA). The STA measures the 'instant' amplitude of the seismic signal and watches for earthquakes. The LTA takes care of the current average seismic noise amplitude.

First, the absolute amplitude of each data sample of an incoming signal is calculated. Next, the average of absolute amplitudes in both windows is calculated. In a further step, a ratio of both values — STA/LTA ratio—is calculated. This ratio is continuously compared to a user selected threshold value - STA/LTA trigger threshold level. If the ratio exceeds this threshold, a channel trigger is declared. A channel trigger does not necessarily mean that a multi-channel data logger or a network actually starts to record seismic signals. All seismic networks and most seismic recorders have a 'trigger voting' mechanism built in that defines how many and which channels have to be in a triggered state before the instrument or the network actually starts to record data. To simplify the explanation, we shall observe only one signal channel. We will assume that a channel trigger is equivalent to a network or a recorder trigger.

After the seismic signal gradually terminates, the channel detriggers. This happens when the current STA/LTA ratio falls below another user-selected parameter - STA/LTA detrigger threshold level. Obviously, the STA/LTA detrigger threshold level should be lower (or rarely equal) than the STA/LTA trigger threshold level.

In addition to the data acquired during the 'trigger active' time, seismic networks and seismic recorders add a certain amount of seismic data to the event file before triggering – pre-event-time (PEM) data. After the trigger active state terminates, they also add post-event-time (PET). data.

For better understanding, Figure 1 shows a typical local event and the trigger variables (simplified) during STA/LTA triggering. Graph a) shows an incoming continuous seismic signal (filtered); graph b) shows an averaged absolute signal in the STA and LTA windows, respectively, as they move in time toward the right side of the graph; and graph c) shows the ratio of both. In addition, the trigger active state (solid line rectangle), the post-event time (PET), and the pre-event time (PEM) (dotted line rectangles) are shown. In this example, the trigger threshold level parameter was set to 10 and the detrigger threshold level to 2 (two short horizontal dotted lines). One can see that the trigger became active when the STA/LTA ratio value exceeded 10. It was deactivated when the STA/LTA ratio value fell below 2. On graph d) the actually recorded data file is shown. It includes all event phases of significance and a portion of the seismic noise at the beginning. In reality, the STA/LTA triggers are usually slightly more complicated, however, the details are not essential for the understanding and proper setting of trigger parameters.

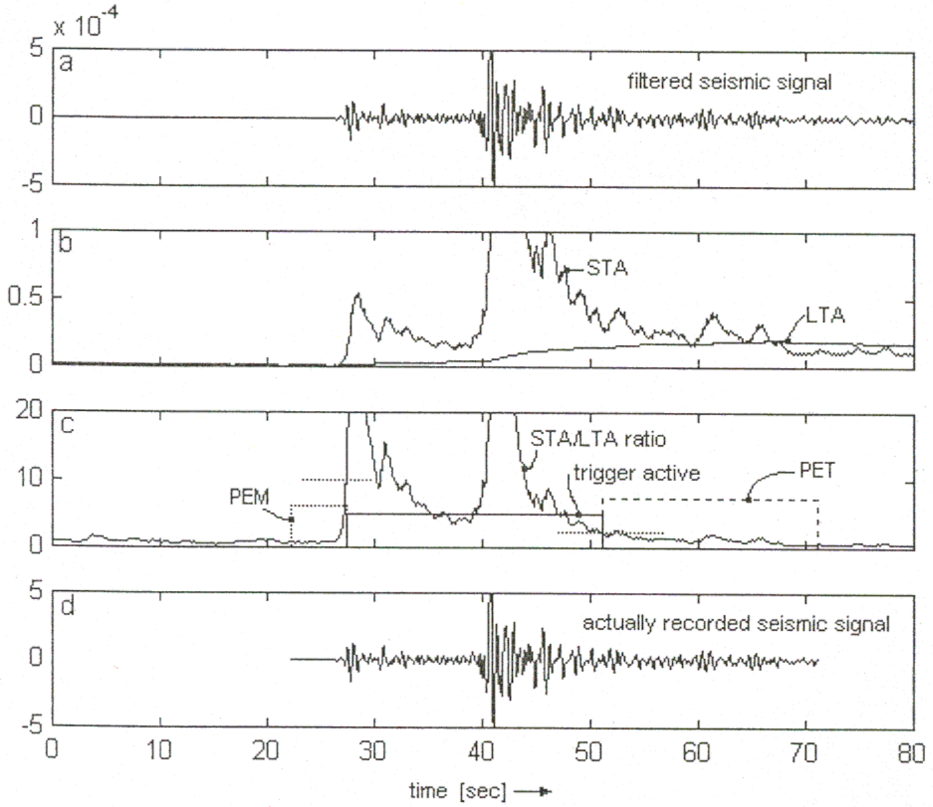


Figure 1 Function and variables of STA/LTA trigger calculations (see text for explanations).

7.1.3 Aim of this Module

The STA/LTA trigger parameter settings are always a tradeoff among several seismological and instrumental considerations. Successful capturing of seismic events depends on proper settings of the trigger parameters. To help with this task, this module explains the STA/LTA trigger functioning and gives general instructions on selecting its parameters. In addition, several different STA/LTA methods will also be introduced in this module.

7.1.4 Activate ObsPy environment

Note:

We here assume you have already installed ObsPy package in the opspy environment on your computer after finishing the previous **Python ObsPy Tutorial**.

Open your terminal and run the following commands.

```
$ conda activate obspy
```

7.1.5 Import necessary packages

```
from obspy.signal.trigger import plot_trigger
from obspy.signal.trigger import classic_sta_lta
```

7.2 1 Choose an Event and Read the Waveform Data

7.2.1 1.1 Choose an event

```
from obspy import UTCDateTime
origin_time = UTCDateTime("2015-08-11T16:22:15.200000")

# Coordinates and the magnitude of the event
eq_lon = 123.202
eq_lat = -8.624
eq_dep = 171.9
eq_mag = 3.9
```

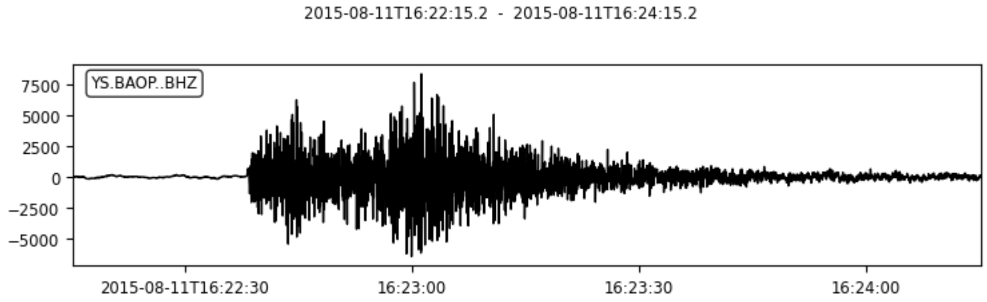
7.2.2 1.2 Choose a station and get the waveform

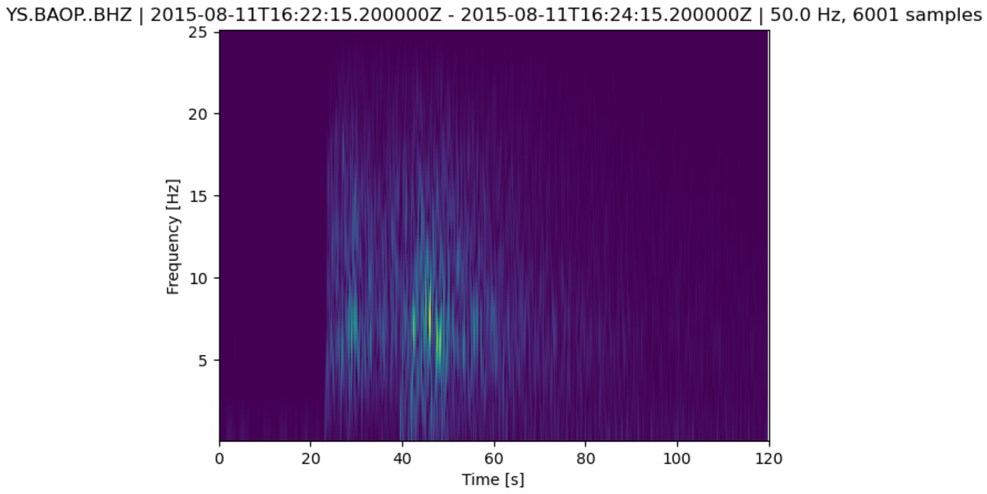
```
from obspy.clients.fdsn import Client

# IRIS is one of those providers.
client = Client('IRIS')

# Input station informations
# network
```

(continues on next page)

(continued from previous page)

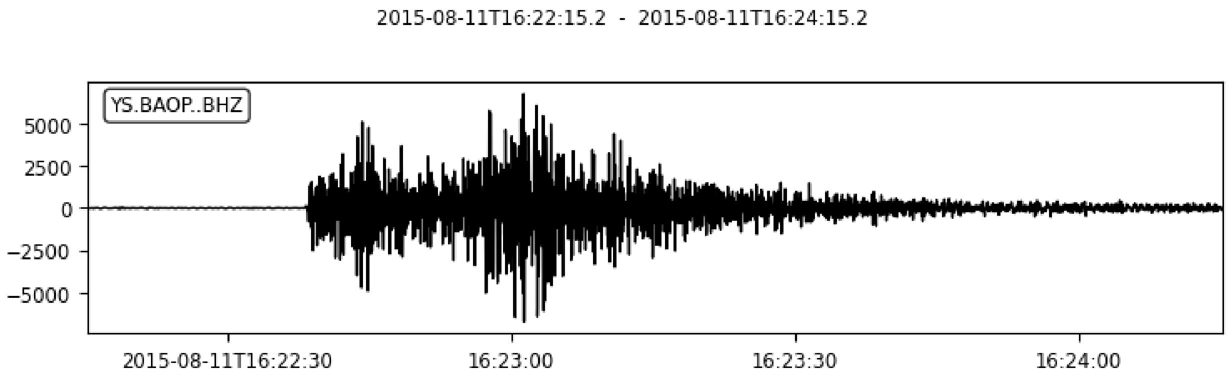
```
net = 'YS'
# station
sta = 'BAOP'
# location
loc = ''
# channel
cha = 'BHZ'

# starttime
stt = origin_time
# endtime
edt = origin_time + 120

# Get the waveforms from client
st= client.get_waveforms(net, sta, loc, cha, stt, edt)
st.plot()
st.spectrogram()
```





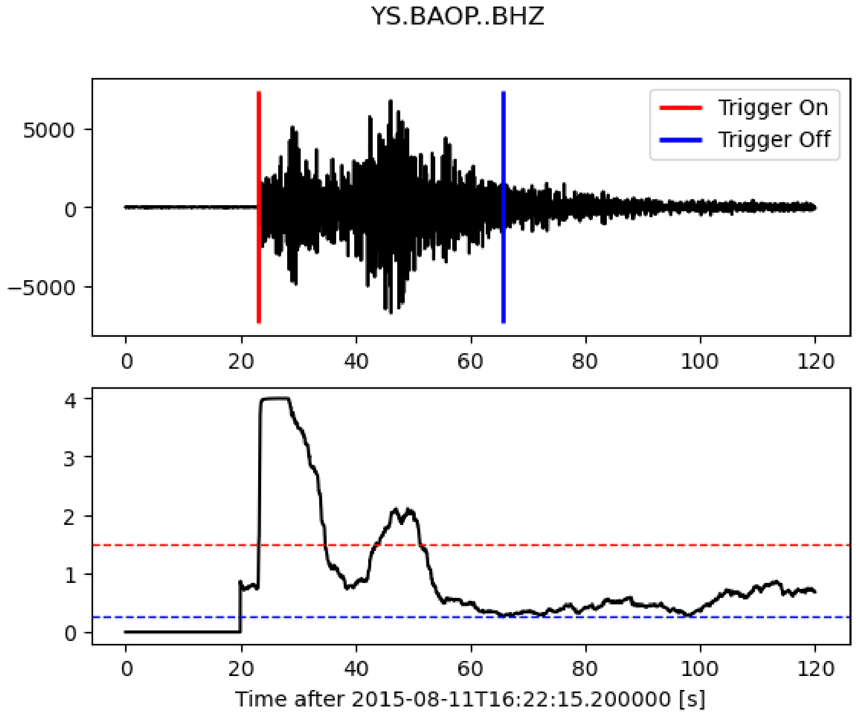
7.2.3 1.3 Filter Data

```
# copy the raw data
st2 = st.copy()
# apply the bandpass between 1.0HZ and 10.0HZ, in order to filter the noise.
st2.filter("bandpass",freqmin=1.0, freqmax=10.0)
st2.plot()
```



7.2.4 1.4 Triggering Example

```
df = st2[0].stats.sampling_rate
# set the STA=5 seconds, LTA=20 seconds
cft = classic_sta_lta(st2[0].data, int(5 * df), int(20 * df))
# set the trigger threshold=1.5, detrigger threshold=0.27
plot_trigger(st2[0], cft, 1.5, 0.27)
```



7.3 2 How to adjust STA/LTA trigger parameters

To set the basic STA/LTA trigger algorithm parameters one has to select the following:

1. `STA window duration`
2. `LTA window duration`
3. `STA/LTA trigger threshold level`
4. `STA/LTA detrigger threshold level`

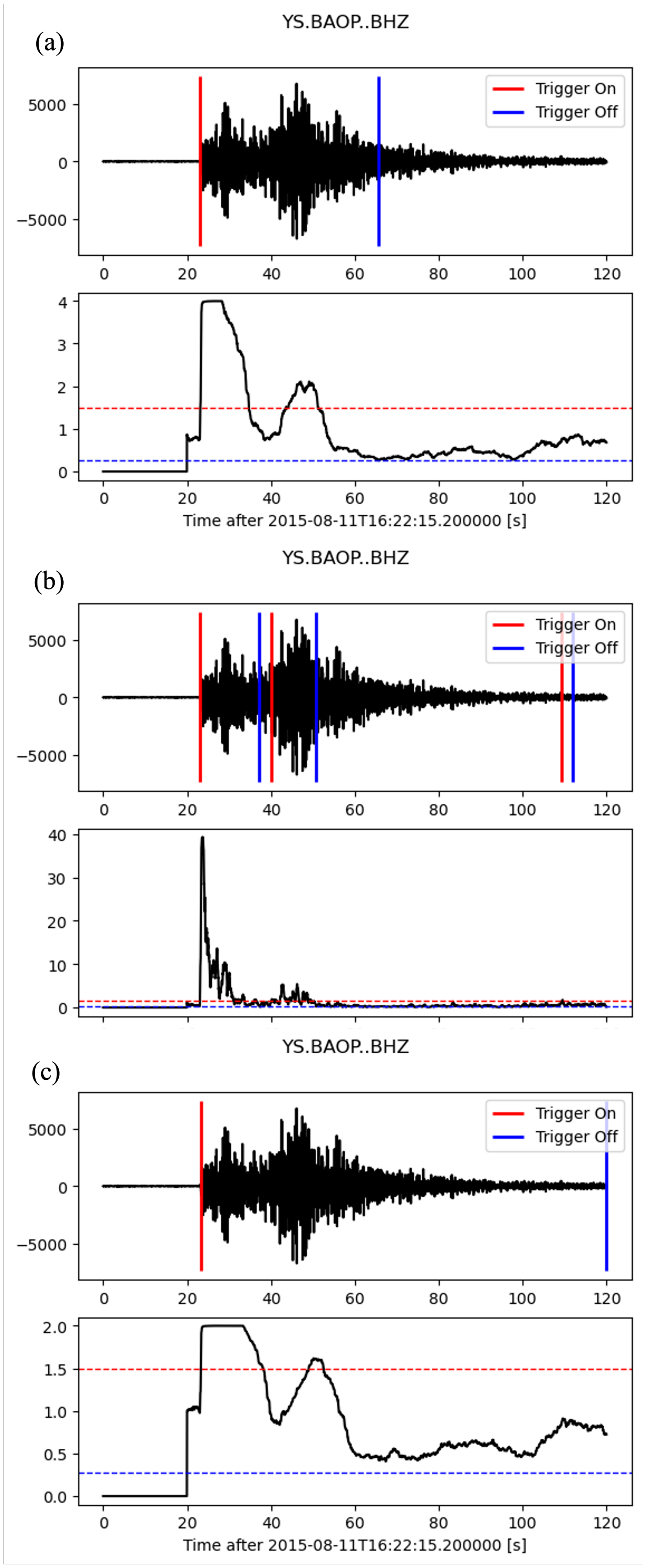
7.3.1 2.1 Selection of short-time average window (STA) duration

Short-time average window measures the 'instant' value of a seismic signal or its envelope. Generally, STA duration must be longer than a few periods of a typically expected seismic signal. If the STA is too short, the averaging of the seismic signal will not function properly. The STA is no longer a measure of the average signal (signal envelope) but becomes influenced by individual periods of the seismic signal. On the other hand, STA duration must be shorter than the shortest events we expect to capture.

The STA can be considered as a signal filter. The shorter the duration selected, the higher the trigger's sensitivity to short lasting local earthquakes compared to long lasting and lower frequency distant earthquakes. The longer the STA duration selected, the less sensitive it is for short local earthquakes. Therefore, by changing the STA duration one can prioritize capturing of distant or local events.

For regional events, a typical value of STA duration is between 1 and 2 sec. For local earthquakes shorter values around 0.5 to 0.3 s are commonly used in practice.

```
df = st2[0].stats.sampling_rate
# only set different short-time average window (STA) durations
# sta=5 seconds
cft = classic_sta_lta(st2[0].data, int(5 * df), int(20 * df))
plot_trigger(st2[0], cft, 1.5, 0.27)
# sta=0.5 seconds, represents a smaller value
cft = classic_sta_lta(st2[0].data, int(0.5 * df), int(20 * df))
plot_trigger(st2[0], cft, 1.5, 0.27)
# sta=10 seconds, represents a larger value
cft = classic_sta_lta(st2[0].data, int(10 * df), int(20 * df))
plot_trigger(st2[0], cft, 1.5, 0.27)
```

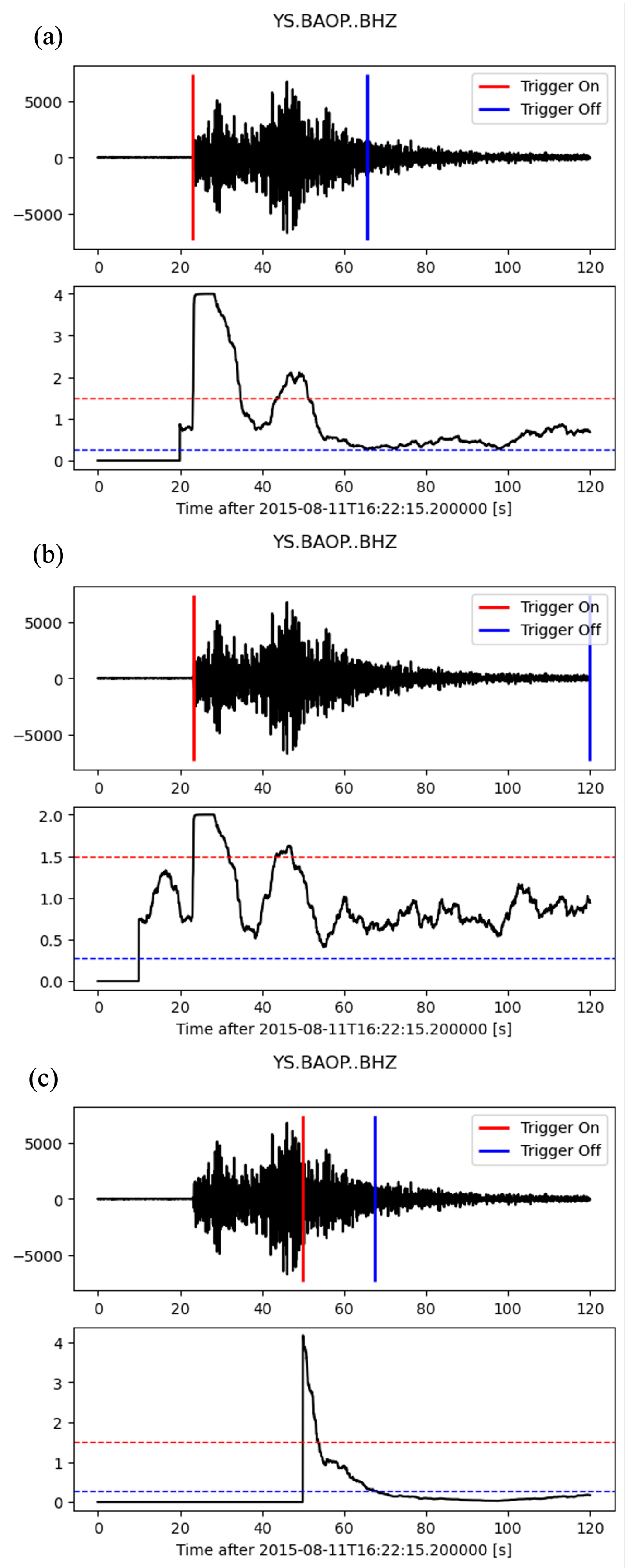

YS.BAOP..BHZ
(a)
Trigger On
Trigger Off
5000
0
−5000
0
20
40
60
80
100
120
4
3
2
1
0
Time after 2015-08-11T16:22:15.200000 [s]
YS.BAOP..BHZ
(b)
Trigger On
Trigger Off
40
30
20
10
YS.BAOP..BHZ
(c)
Trigger On
Trigger Off
2.0
1.5
1.0
0.5
0.0
Time after 2015-08-11T16:22:15.200000 [s]

7.3.2 2.2 Selection of long-time average window (LTA) duration

The LTA window measures average amplitude seismic noise. It should last longer than a few 'periods' of typically irregular seismic noise fluctuations. By changing the LTA window duration, one can make the recording more or less sensitive to regional events in the 'Pn'-wave range from about 200 to 1500 km epicentral distance. These events typically have the low-amplitude emergent Pn- waves as the first onset. A short LTA duration allows the LTA value more or less to adjust to the slowly increasing amplitude of emergent seismic waves. Thus the STA/LTA ratio remains low in spite of increasing STA (nominator and denominator of the ratio increase). This effectively diminishes trigger sensitivity to such events. In the opposite case, using a long LTA window duration, trigger sensitivity to the emergent earthquakes is increased because the LTA value is not so rapidly influenced by the emergent seismic signal, allowing Sg/Lg waves to trigger the recording.

The LTA duration of 60 seconds is a common initial value. A shorter LTA duration is needed to exclude emergent regional events from triggering, if desired, or if quickly changing manmade noise is typical for the site. A longer LTA can be used for distant regional events with very long S-P times and potentially emergent P waves.

```
# only set different long-time average window (STA) durations
# lta=20 seconds
cft = classic_sta_lta(st2[0].data, int(5 * df), int(20 * df))
plot_trigger(st2[0], cft, 1.5, 0.27)
# lta=10 seconds, represents a smaller value
cft = classic_sta_lta(st2[0].data, int(5 * df), int(10 * df))
plot_trigger(st2[0], cft, 1.5, 0.27)
# lta=50 seconds, represents a larger value
cft = classic_sta_lta(st2[0].data, int(5 * df), int(50 * df))
plot_trigger(st2[0], cft, 1.5, 0.27)
```

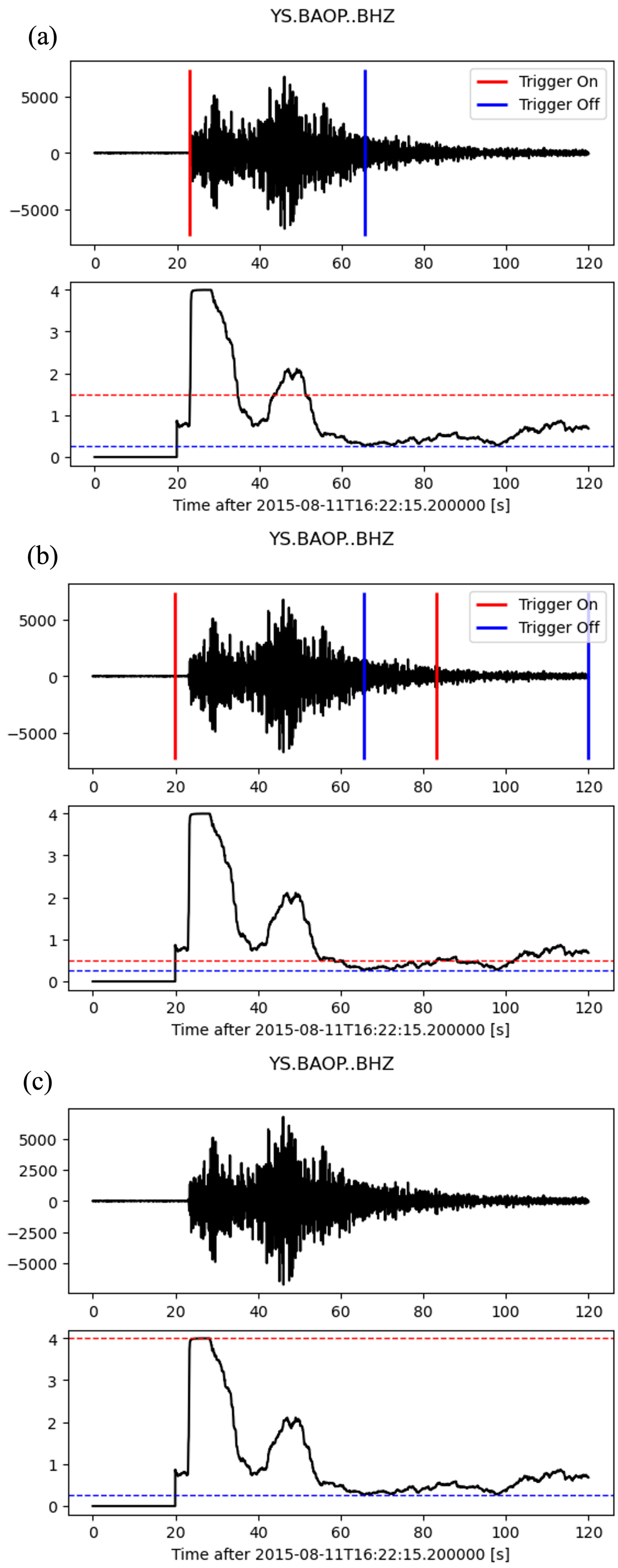

(a)
YS.BAOP..BHZ
Trigger On
Trigger Off
5000
0
−5000
0
20
40
60
80
100
120
4
3
2
1
0
Time after 2015-08-11T16:22:15.200000 [s]
(b)
YS.BAOP..BHZ
Trigger On
Trigger Off
5000
0
−5000
2.0
1.5
1.0
0.5
0.0
Time after 2015-08-11T16:22:15.200000 [s]
(c)
YS.BAOP..BHZ
Trigger On
Trigger Off
5000
0
−5000
4
3
2
1
0
Time after 2015-08-11T16:22:15.200000 [s]

7.3.3 2.3 Selection of STA/LTA trigger threshold level

The STA/LTA trigger threshold level to the greatest extent determines which events will be recorded and which will not. The higher value one sets, the more earthquakes will not be recorded, but the fewer false-triggers will result. The lower the STA/LTA trigger threshold level is selected, the more sensitive the seismic station will be and the more events will be recorded. However, more frequent false triggers also will occupy data memory and burden the analyst. An optimal STA/LTA trigger threshold level depends on seismic noise conditions at the site and on one's tolerance to falsely triggered records. Not only the amplitude but also the type of seismic noise influence the setting of the optimal STA/LTA trigger threshold level. A statistically stationary seismic noise (with less irregular fluctuations) allows a lower STA/LTA trigger threshold level; completely irregular behavior of seismic noise demands higher values.

An initial setting for the STA/LTA trigger threshold level of 4 is common for an average quiet seismic site. Much lower values can be used only at the very best station sites with no man- made seismic noise. Higher values about 8 and above are required at less favorable sites with significant man-made seismic noise. In strong-motion applications, higher values are more common due to the usually noisier seismic environment and generally smaller interest in weak events.

```
cft = classic_sta_lta(st2[0].data, int(5 * df), int(20 * df))
# only set different STA/LTA trigger threshold levels
# STA/LTA trigger threshold = 1.5
plot_trigger(st2[0], cft, 1.5, 0.27)
# STA/LTA trigger threshold = 0.5, represents a small trigger threshold value
plot_trigger(st2[0], cft, 0.5, 0.27)
# STA/LTA trigger threshold = 4, represents a large trigger threshold value
plot_trigger(st2[0], cft, 4, 0.27)
```

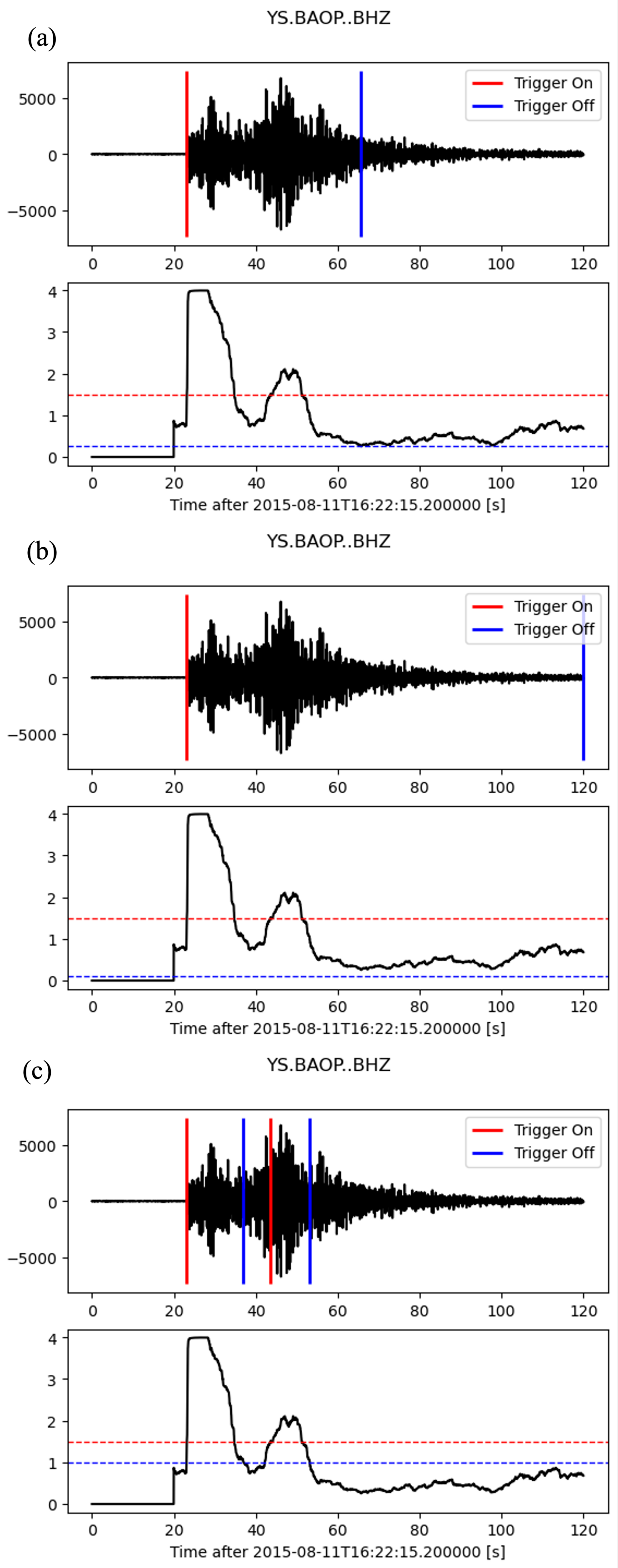

YS.BAOP..BHZ
(a)
Trigger On
Trigger Off
5000
0
−5000
0
20
40
60
80
100
120
4
3
2
1
0
Time after 2015-08-11T16:22:15.200000 [s]
YS.BAOP..BHZ
(b)
Trigger On
Trigger Off
Time after 2015-08-11T16:22:15.200000 [s]
YS.BAOP..BHZ
(c)
5000
2500
0
−2500
−5000
Time after 2015-08-11T16:22:15.200000 [s]

7.3.4 2.4 Selection of STA/LTA detrigger threshold level

The STA/LTA detrigger threshold level determines the termination of data recording. To include as much of the coda waves as possible, a low value is required. If one uses coda duration for magnitude determinations, such setting is obvious. However, a too low STA/LTA detrigger threshold level is occasionally dangerous. It may cause very long or even endless records, for example, if a sudden increase in seismic noise does not allow the STA/LTA ratio to fall below the STA/LTA detrigger threshold level. On the other hand, if one is not interested in coda waves, a higher value of STA/LTA detrigger threshold level enables significant savings in data memory and/or data transmission time. Note that coda waves of distant earthquakes can be very long.

A typical initial value of the STA/LTA detrigger threshold level is 2 to 3 for seismically quiet sites and weak motion applications. For noisier sites, higher values must be set. For strong- motion applications, where coda waves are not of the highest importance, higher values are frequently used.

```
cft = classic_sta_lta(st2[0].data, int(5 * df), int(20 * df))
# only set different STA/LTA detrigger threshold levels
# STA/LTA trigger threshold = 1.5
plot_trigger(st2[0], cft, 1.5, 0.27)
# STA/LTA detrigger threshold = 0.1, represents a small trigger threshold value
plot_trigger(st2[0], cft, 1.5, 0.1)
# STA/LTA detrigger threshold = 1, represents a large trigger threshold value
plot_trigger(st2[0], cft, 1.5, 1)
```

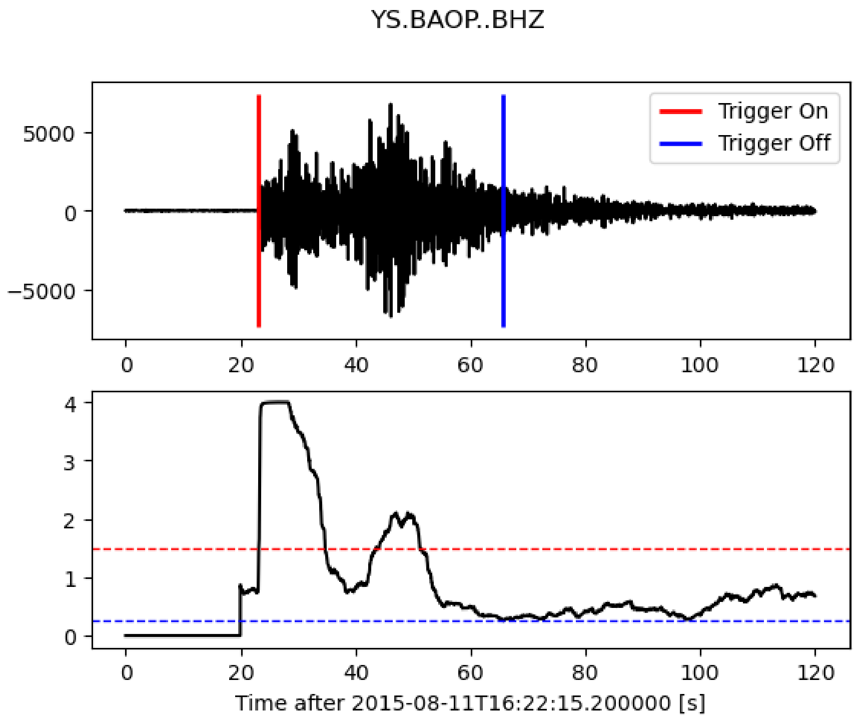

(a)
YS.BAOP..BHZ
Trigger On
Trigger Off
5000
0
−5000
0
20
40
60
80
100
120
4
3
2
1
0
Time after 2015-08-11T16:22:15.200000 [s]
(b)
YS.BAOP..BHZ
Trigger On
Trigger Off
Time after 2015-08-11T16:22:15.200000 [s]
(c)
YS.BAOP..BHZ
Trigger On
Trigger Off
Time after 2015-08-11T16:22:15.200000 [s]

7.4 3 Available STA/LTA Methods

`recursive_sta_lta` (a, nsta, nlta)	Recursive STA/LTA.
`carl_sta_trig` (a, nsta, nlta, ratio, quiet)	Computes the carlSTAtrig characteristic function.
`classic_sta_lta` (a, nsta, nlta)	Computes the standard STA/LTA from a given input array a.
`delayed_sta_lta` (a, nsta, nlta)	Delayed STA/LTA.
`z_detect` (a, nsta)	Z-detector.
`pk_baer` (reltrc, samp_int, tdownmax, ...[, ...])	Wrapper for P-picker routine by M.
`ar_pick` (a, b, c, samp_rate, f1, f2, lta_p, ...)	Pick P and S arrivals with an AR-AIC + STA/LTA algorithm.

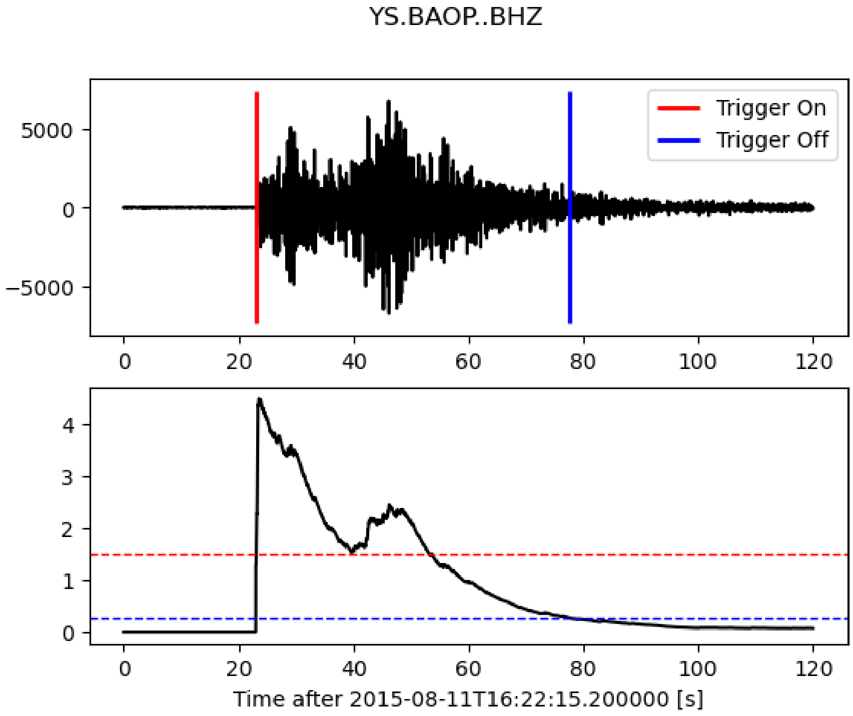
7.4.1 3.1 Classic Sta Lta

```
# we already import the classic_sta_lta at the begining.
df = st2[0].stats.sampling_rate
# set the STA=5 seconds, LTA=20 seconds
cft = classic_sta_lta(st2[0].data, int(5 * df), int(20 * df))
# set the trigger threshold=1.5, detrigger threshold=0.27
plot_trigger(st2[0], cft, 1.5, 0.27)
```



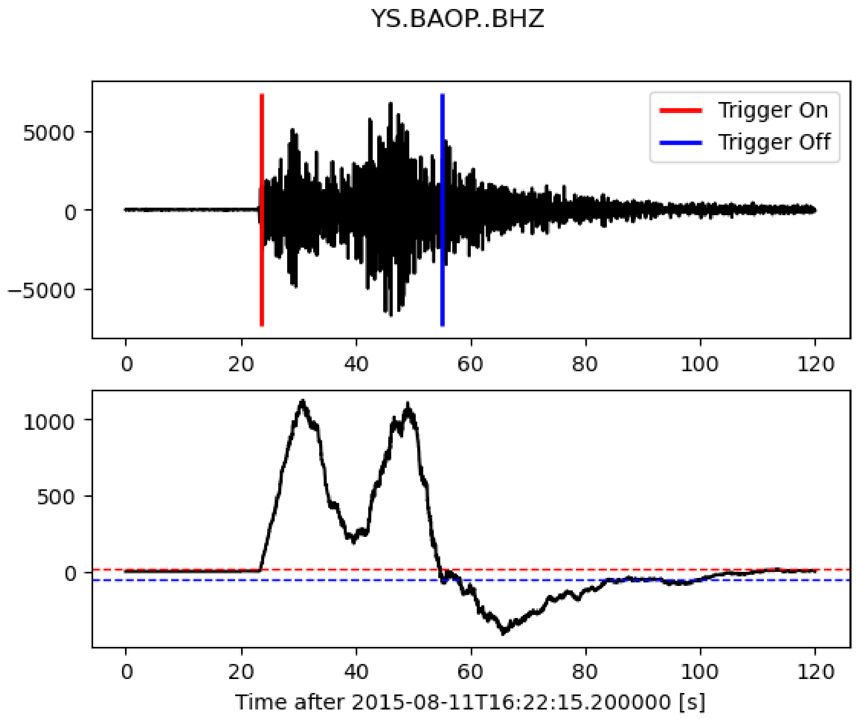
7.4.2 3.2 Recursive Sta Lta

```
from obspy.signal.trigger import recursive_sta_lta
# set the STA=5 seconds, LTA=23 seconds
cft = recursive_sta_lta(st2[0].data, int(5 * df), int(23 * df))
# set the trigger threshold=1.5, detrigger threshold=0.27
plot_trigger(st2[0], cft, 1.5, 0.27)
```



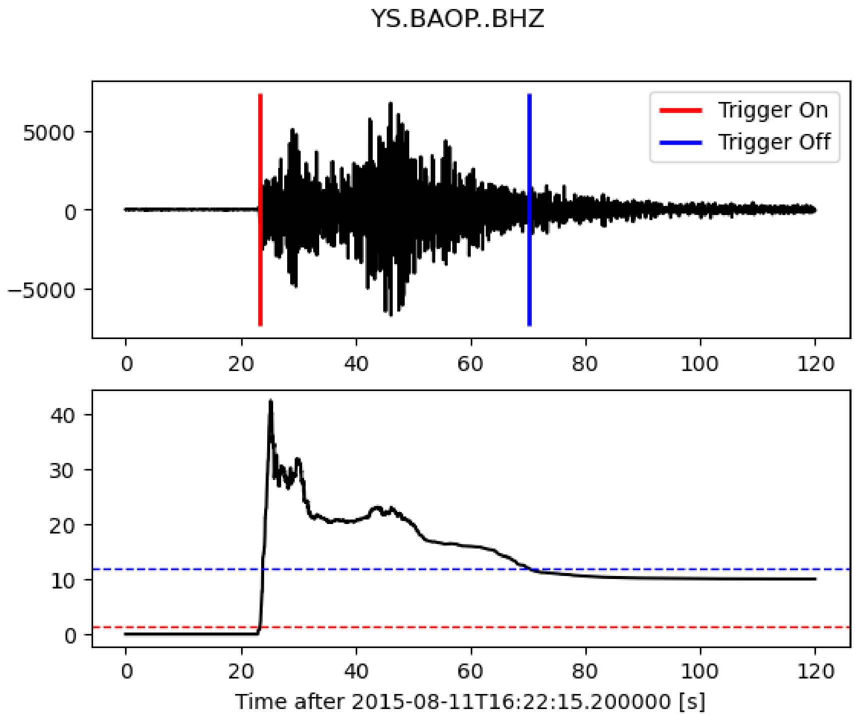
7.4.3 3.3 Carl-Sta_Trig

```
from obspy.signal.trigger import carl_sta_trig
# set the STA=5 seconds, LTA=20 seconds
cft = carl_sta_trig(st2[0].data, int(5 * df), int(20 * df), 0.8, 0.8)
# set the trigger threshold=20, detrigger threshold=-50.0
plot_trigger(st2[0], cft, 20.0, -50.0)
```



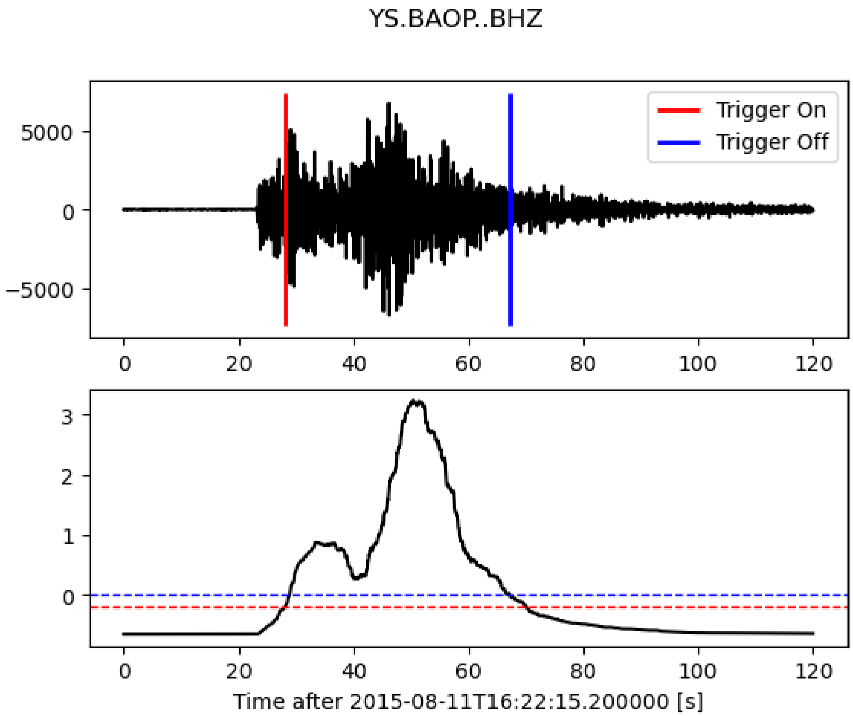
7.4.4 3.4 Delayed Sta Lta

```
from obspy.signal.trigger import delayed_sta_lta
# set the STA=5 seconds, LTA=20 seconds
cft = delayed_sta_lta(st2[0].data, int(2 * df), int(20 * df))
# set the trigger threshold=1.5, detrigger threshold=12
plot_trigger(st2[0], cft, 1.5,12)
```



7.4.5 3.5 Z-Detect

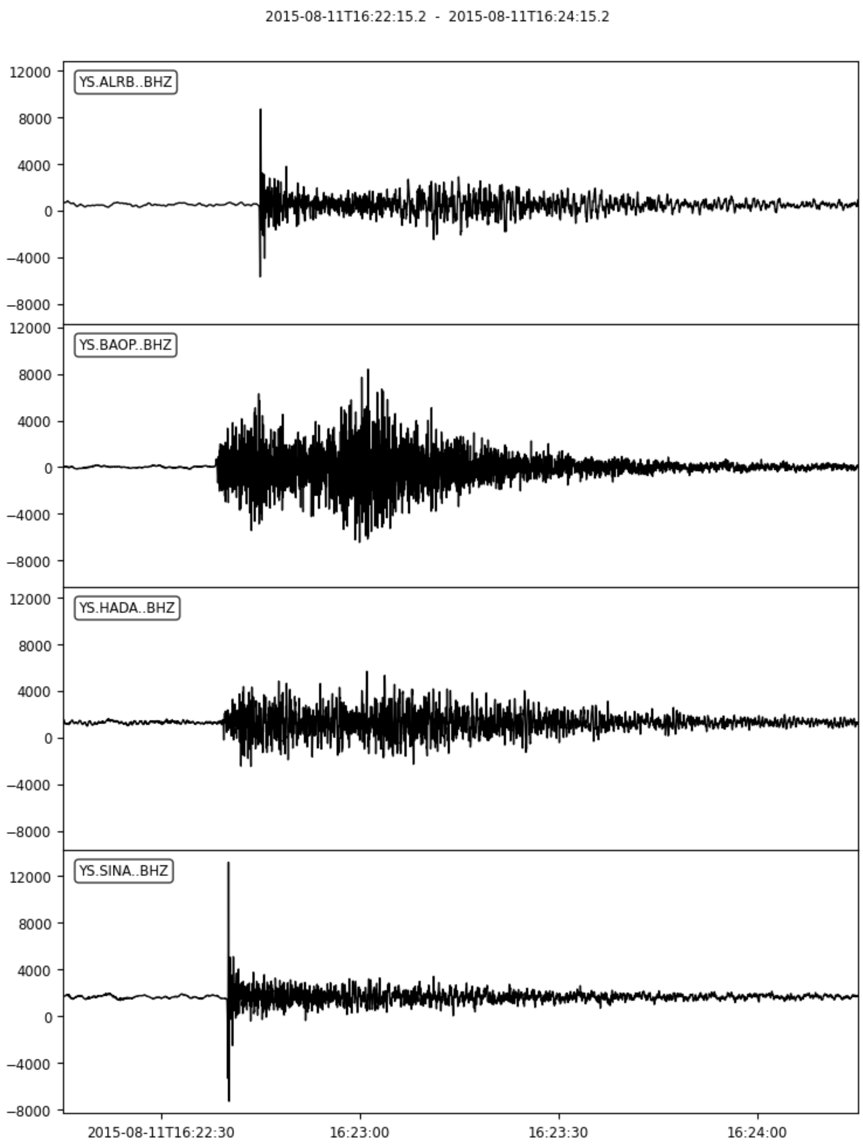
```
from obspy.signal.trigger import z_detect
# set the LTA=10 seconds
cft = z_detect(st2[0].data, int(10 * df))
# set the trigger threshold=-0.2, detrigger threshold=0
plot_trigger(st2[0], cft, -0.2, 0)
```



7.5 4 Detect events on multiple traces by using STA/LTA

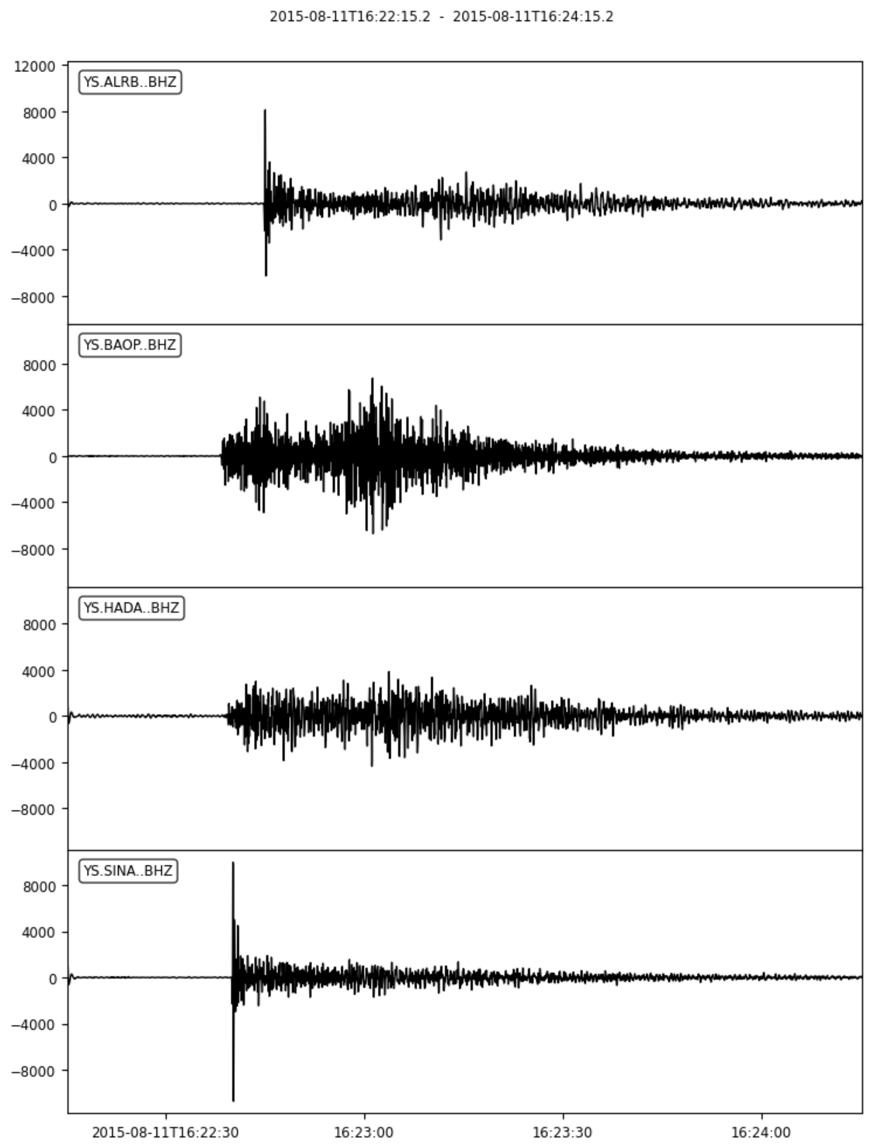
7.5.1 4.1 Get the waveform data with more than 1 station

```
# Set up a list for bulk request
bulk = [('YS', 'BAOP', '', 'BHZ', origin_time, origin_time+120),
        ('YS', 'HADA', '', 'BHZ', origin_time, origin_time+120),
        ('YS', 'SINA', '', 'BHZ', origin_time, origin_time+120),
        ('YS', 'ALRB', '', 'BHZ', origin_time, origin_time+120)]

st_bulk = client.get_waveforms_bulk(bulk)
st_bulk.plot()
```



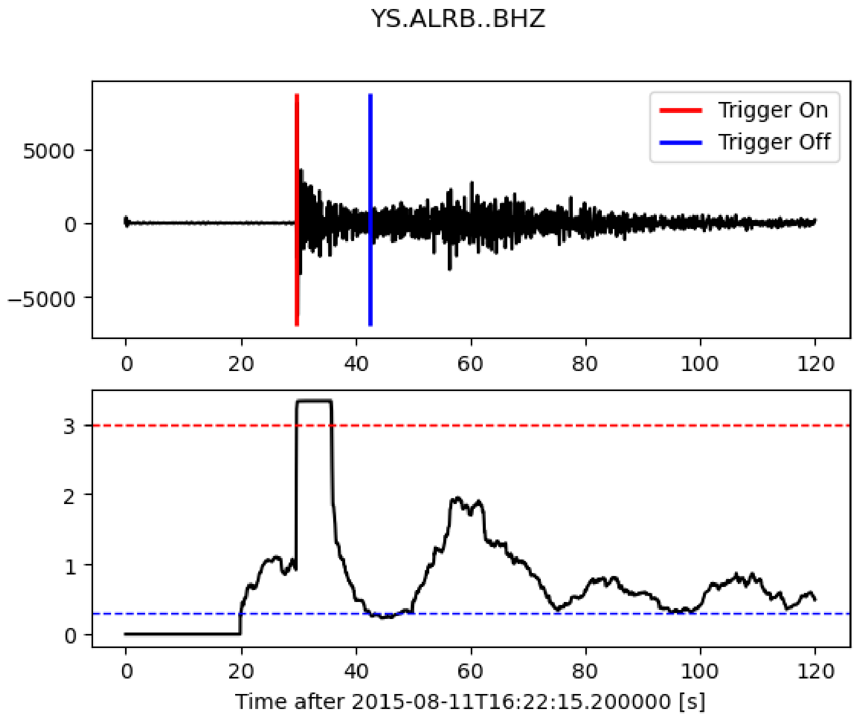
7.5.2 4.2 Filter the data

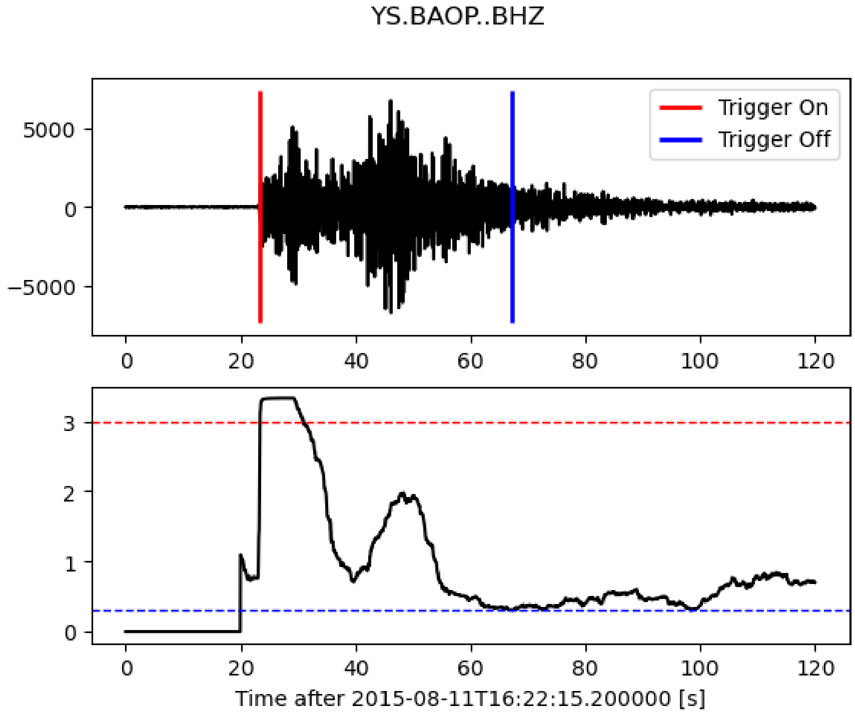
```
# make a copy of raw data
st2 = st_bulk.copy()
# apply the bandpass
st2.filter('bandpass',freqmin=1,freqmax=10)
st2.plot()
```

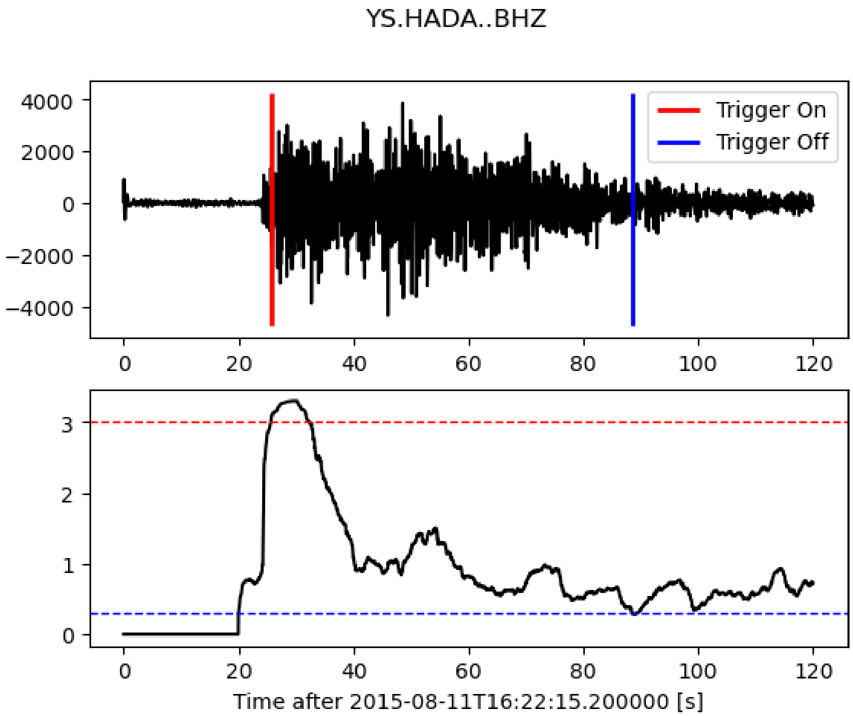


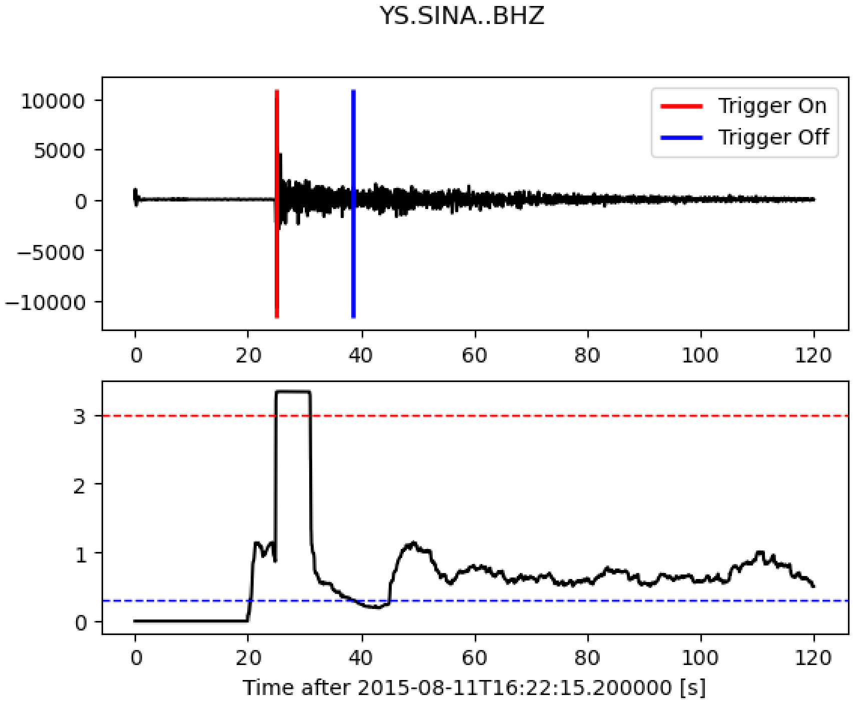
7.5.3 4.2 Using STA/LTA to detect events

```
# we already import the classic_sta_lta at the begining.
# set a loop to detect the events using STA/LTA, the classic_sta_lta method is used here.
# you can try other STA/LTA methods and find the differences
for tr in st2:
    df = tr.stats.sampling_rate
    cft = classic_sta_lta(tr.data, int(6 * df), int(20 * df))
    plot_trigger(tr, cft, 3, 0.3)
```


YS.ALRB..BHZ
Trigger On
Trigger Off
5000
0
−5000
0
20
40
60
80
100
120
3
2
1
0
Time after 2015-08-11T16:22:15.200000 [s]


YS.BAOP..BHZ
Trigger On
Trigger Off
5000
0
−5000
0
20
40
60
80
100
120
3
2
1
0
Time after 2015-08-11T16:22:15.200000 [s]


YS.HADA..BHZ
Trigger On
Trigger Off
4000
2000
0
−2000
−4000
0
20
40
60
80
100
120
3
2
1
0
Time after 2015-08-11T16:22:15.200000 [s]


YS.SINA..BHZ
Trigger On
Trigger Off
10000
5000
0
−5000
−10000
0
20
40
60
80
100
120
3
2
1
0
Time after 2015-08-11T16:22:15.200000 [s]

CHAPTER

EIGHT

EARTHQUAKE ABSOLUTE LOCATION

8.1 Introduction

The earthquake detection process finds the arrival times of P&S wave on different stations for each event. The next step is to get the earthquake location (**longitude(x),latitude(y),depth(z)**) and origin time (**t**). The central idea is to invert the data by the equations governing the relationship between arrival data and hypocenter location. However, due to heterogenities in real earth setting, a solution that fits all data rarely exists.

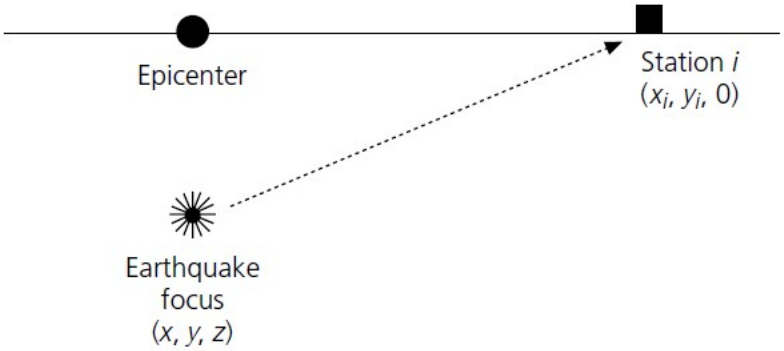
Since the eathquake location cannot fit well with all data. It is mostly determined by the location with minimum misfit. One widely used method is the least-square inversion. Another method used is the grid search method which earthquake location is searched iteratively by defined possible model ranges. Each grid is assumed to be the earthquake location and their misfit with the data is computed. Hence, the minimum misfit location is determine. One large limitation of grid search is the resolution is controlled by grid size. …

The most widely used methods are based on least-square algorithm. That is, the earthquake location is determined by the minimum misfit with observations. One least-square method is the grid search method. In this method, the possible space is splited into 3D grids, and then the misfit of each grid as the earthquake location is calculated one by one. The earthquake is then considered to be occurred inside the grid with minimum misfit. The grid size controls the earthquake location resolution, a larger grid size will lead to a larger uncertainty (the range where the true earthquake location might be). However, a smaller grid size will lead to heavy calculation load due to dramtically increased grid quantity. Therefore, there is a trade-off between resolution and calculation efficiency in this method.

Another earthquake location method is by inversion. The equation is linearized and solved iteratively by least-sqaure methods. This method does not require intense computation and converge fastly. It is therefore widely used. …

Another least-square earthquake location method is the iterative absolute earthquake location method, which can achieve high-resolution location with ideal calculation efficiency and is widely used. In this session, we introduce the details of this method. We start from simple one-layer model for practice and then run into real data processing using the popular `HYPOINVERSE` program, which is developed based on the iterative absolute earthquake location method.

8.1.1 Theory Background



The arrival time recorded by one station could be presented as:

$$T_i^k = O_k + \int_s^r udS$$

where T_i^k is the arrival time of event k on station i. In the right side of equation, there are two components: 1) The origin time of event k O_k; 2) The event travel time. It is the integral over the ray path. The s denotes the source location, r represents the receiver (station) location, u is the reciprocal of velocity named slowness.

Assume the event with true source parameters $\mathbf{m_{true}} = (x_{true}, y_{true}, z_{true}, o_{true})^T$ is recorded by n stations, we use $\mathbf{d_obs}$ to denote the arrival times of stations, then $\mathbf{m_{true}}$ should satisfy:

$$\mathbf{Fm_{true}} = \mathbf{d_obs}$$

The relationship between $\mathbf{m}$ and $\mathbf{d}$ is non-linear because integral is involved as shown in previous equation. For earthquake parameters that are close to $\mathbf{m_{true}}$, we present them as $\mathbf{m} = \mathbf{m_{true}} + \mathbf{\Delta m}$, the $\mathbf{\Delta m}$ will lead to variation in $\mathbf{d_{obs}} + \mathbf{\Delta d}$

According to Taylor Expansion theorem, we have:

$$\mathbf{F(m + \Delta m) = Fm + F'\Delta m} + ...,$$

Neglect the items after the first-order partial derivative, we then have $\mathbf{\Delta d} = \mathbf{F'\Delta m}$.

Given the initial location and origin time $\mathbf{m_0} = (x_0, y_0, z_0, t_0)$, we can calculate the corresponding arrival time at each station $\mathbf{d_cal}$ and the $\mathbf{\Delta d}$. Using the $\mathbf{\Delta d}$ we can estimate the $\mathbf{\Delta m}$. By updating the $\mathbf{\Delta m}$, an absolute location of hypocenter can be derived.

8.1.2 Contents of this tutorial

We will introduce how to derive and analyze absolute locations of hypocenters in pyhon which has been devided into listed parts:

1. Gird search method
2. Iteration method
3. Error analyze

Authors: ZI Jinping & SONG Zilin, Earth Science System Program, CUHK.

Testers: XIA Zhuoxuan & SUN Zhangyu, Earth Science System Program, CUHK.

8.2 Python Environment and model Setup

We'll first generate synthetic arrival times on stations from one earthquake location, which is called **Forward Modelling**. We'll then try to re-generate hypocentral location using these known station arrival times, which is called **Inversion**. In this tutorial, we use one-layer model to avoid complicated ray-tracing (`Snell's law`)

Note:

The purpose is that, by comparing the difference between **inverted location** and **true location**, we can: 1. see whether the location process runs properly; 2. conduct error analysis.

8.2.1 Python environment

```
import numpy as np
import matplotlib.pyplot as plt
import time
from matplotlib.patches import Ellipse
from mpl_toolkits.mplot3d import Axes3D
```

Note:

Define functions that will be used later.

```
def matrix_show(*args,**kwargs):
    """
    Show matrix values in grids shape
    Parameters:cmap="cool",gridsize=0.6,fmt='.2f',label_data=True
    """
    ws = []
    H = 0
    str_count = 0
    ndarr_count = 0
    new_args = []
    for arg in args:
        if isinstance(arg,str):
            new_args.append(arg)
            continue
        if isinstance(arg,list):
            arg = np.array(arg)
        if len(arg.shape)>2:
            raise Exception("Only accept 2D array")
        if len(arg.shape) == 1:
            n = arg.shape[0]
            tmp = np.zeros((n,1))
            tmp[:,0] = arg.ravel()
            arg = tmp
        h,w = arg.shape
        if h>H:
            H=h
```

(continues on next page)

(continued from previous page)

```
        ws.append(w)
        new_args.append(arg)
        ndarr_count += 1
    W = np.sum(ws)+len(ws)    # text+matrix+text+...+matrix+text
    if W<0:
        raise Exception("No matrix provided!")

    fmt = '.2f'
    grid_size = 0.6
    cmap = 'cool'
    label_data = True
    for arg in kwargs:
        if arg == "fmt":
            fmt = kwargs[arg]
        if arg == 'grid_size':
            grid_size = kwargs[arg]
        if arg == 'cmap':
            cmap = kwargs[arg]
        if arg == 'label_data':
            label_data = kwargs[arg]
    fig = plt.figure(figsize=(W*grid_size,H*grid_size))
    gs = fig.add_gridspec(nrows=H,ncols=W)

    wloop = 0
    matrix_id = 0
    for arg in new_args:
        if isinstance(arg,str):
            ax = fig.add_subplot(gs[0:H,wloop-1:wloop])
            ax.axis("off")
            ax.set_xlim(0,1)
            ax.set_ylim(0,H)
            ax.text(0.5,H/2,arg,horizontalalignment='center',verticalalignment='center')
        if isinstance(arg,np.ndarray):
            h,w = arg.shape
            hlow = int(np.round((H-h+0.01)/2))        # Find the height grid range
            hhigh = hlow+h
            wlow = wloop
            whigh = wlow+w
#            print("H: ",H,hlow,hhigh,"; W ",W,wlow,whigh)
            ax = fig.add_subplot(gs[hlow:hhigh,wlow:whigh])

            plt.pcolormesh(arg,cmap=cmap)
            for i in range(1,w):
                plt.axvline(i,color='k',linewidth=0.5)
            for j in range(1,h):
                plt.axhline(j,color='k',linewidth=0.5)
            if label_data:
                for i in range(h):
                    for j in range(w):
                        plt.text(j+0.5,i+0.5,format(arg[i,j],fmt),
                                 horizontalalignment='center',
                                 verticalalignment='center')
```

(continues on next page)

(continued from previous page)

```
            plt.xlim(0,w)
            plt.ylim([h,0])
            plt.xticks([])
            plt.yticks([])
            wloop+=w+1
            matrix_id+=1
    plt.show()
```

8.2.2 Model setup

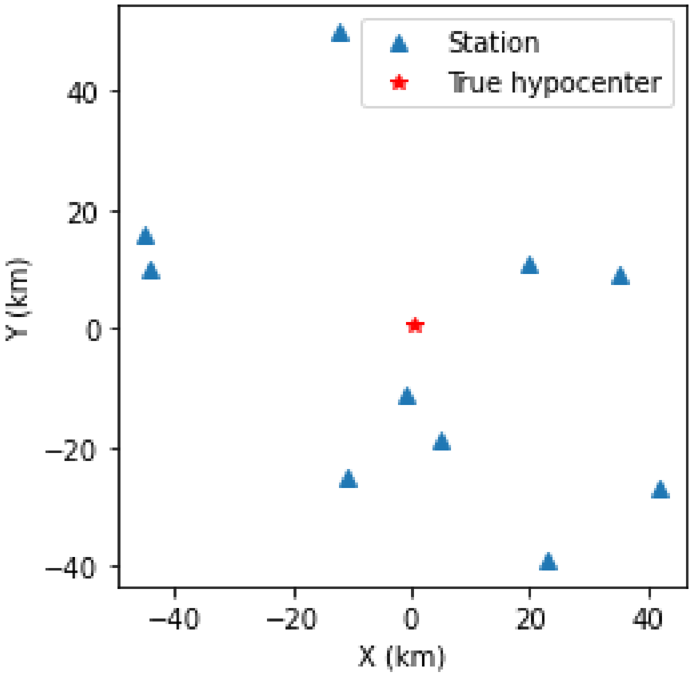
Define basic parameters:

1. Station locations (stats)
2. True hypocenter location (hyc_true)
3. Velocity (Vp)

```
stas_set1 = np.array([[-45,16,0],
                     [-44,10,0],
                     [-12,50,0],
                     [-11,-25,0],
                     [-1,-11,0],
                     [5,-19,0],
                     [20,11,0],
                     [23,-39,0],
                     [35,9,0],
                     [42,-27,0]])
stas = stas_set1
nsta = stas.shape[0]
```

```
hyc_true = np.array([0.5,0.5,9.45,0])     # The true hypocenter value(x,y,z,t)
Vp = 5
```

```
plt.plot(stas[:,0],stas[:,1],'^',label="Station")
plt.plot(hyc_true[0],hyc_true[1],'r*',label='True hypocenter')
plt.xlabel("X (km)")
plt.ylabel("Y (km)")
plt.gca().set_aspect("equal")
plt.legend();
```



Generate synthetic arrival times

```
dobs = np.zeros((nsta,1))
for i in range(dobs.shape[0]):
    dx = stas[i,0]-hyc_true[0]
    dy = stas[i,1]-hyc_true[1]
    dz = stas[i,2]-hyc_true[2]
    dobs[i,0] = np.sqrt(dx**2+dy**2+dz**2)/Vp+hyc_true[3]
nobs = dobs.shape[0]
```

8.3 The Grid-Search Method

The grid search method separates the possible earthquake location zone into 3-D grids, trying each grid as earthquake center and calculating the residual. The grid where earthquake is located should has the lowest residual.

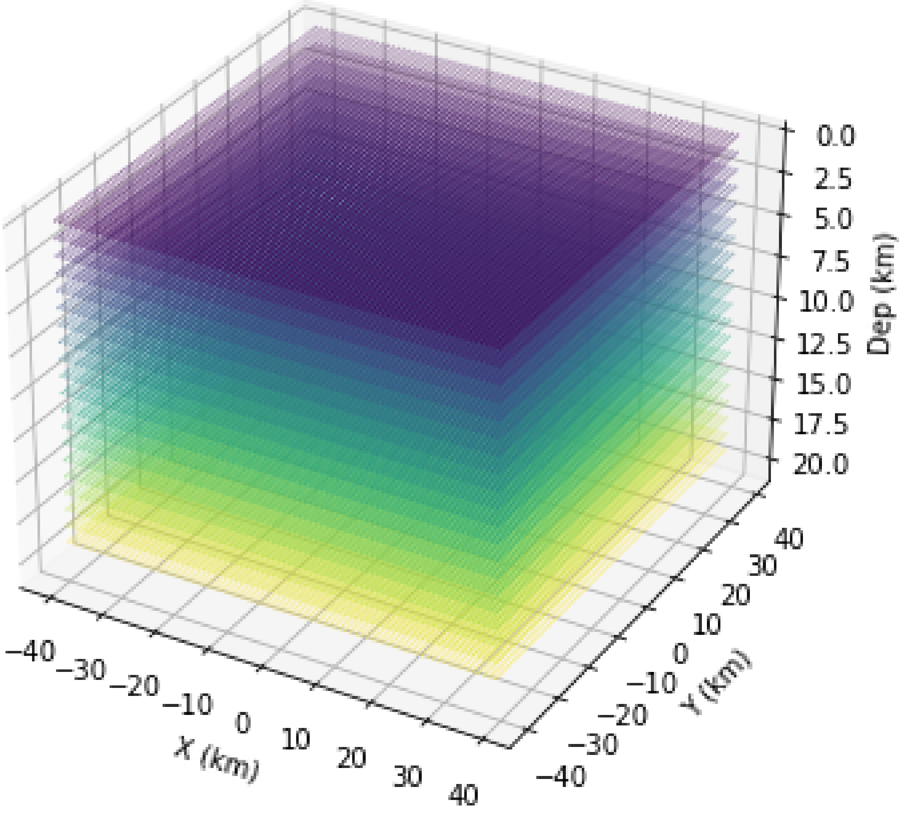
8.3.1 1. Set up grids

```
dx = 1
dy = 1
dz = 1
xs = np.arange(-40,41,dx)
ys = np.arange(-40,41,dy)
zs = np.arange(0,20,dz)
nx = len(xs)
ny = len(ys)
nz = len(zs)
print("Total number of nodes are: ",)  # For students, fill in the blank
```

(continues on next page)

(continued from previous page)

```
fig = plt.figure(figsize=(6,6))
ax = plt.axes(projection='3d')
nodes = []
for x in xs[:-1]:
    for y in ys[:-1]:
        for z in zs[:-1]:
            nodes.append([x,y,z])
nodes = np.array(nodes)
ax.scatter3D(nodes[:,0],nodes[:,1],nodes[:,2],c=nodes[:,2],s=0.1)
ax.set_xlabel("X (km)")
ax.set_ylabel("Y (km)")
ax.set_zlabel("Dep (km)")
ax.set_zlim([20,0])
plt.show()
```



8.3.2 2. Try each grid and calculate error

```
V = Vp
sq_errs = np.zeros((nx,ny,nz))
ta = time.time()                        # The time before calculation
for i in range(len(xs)):
    for j in range(len(ys)):
        for k in range(len(zs)):
            dcal = np.zeros((nsta,1))
            x = xs[i];y=ys[j];z=zs[k]
            for m in range(nsta):
                sta_x = stas[m,0]
                sta_y = stas[m,1]
                sta_z = stas[m,2]
                dist = np.sqrt((sta_x-x)**2+(sta_y-y)**2+(sta_z-z)**2)
                dcal[m,0] = dist/V
            sq_err = np.linalg.norm(dobs-dcal)**2
            sq_errs[i,j,k] = sq_err
tb = time.time()                        # The time after calculation
print("Time for location process: ",format(tb-ta,'.3f'),'s')
```

8.3.3 3. Find the minimum misfit grid

```
sq_err_min = np.min(sq_errs)             # Get the min value
sq_err_max = np.max(sq_errs)
kk = np.where(sq_errs==sq_err_min)       # Get the value indexs
idx = kk[0][0]
idy = kk[1][0]
idz = kk[2][0]
print(f"Minimum occurred in x={xs[idx]}, y={ys[idy]}, z={zs[idz]}")
```

```
Minimum occurred in x=1, y=1, z=9
```

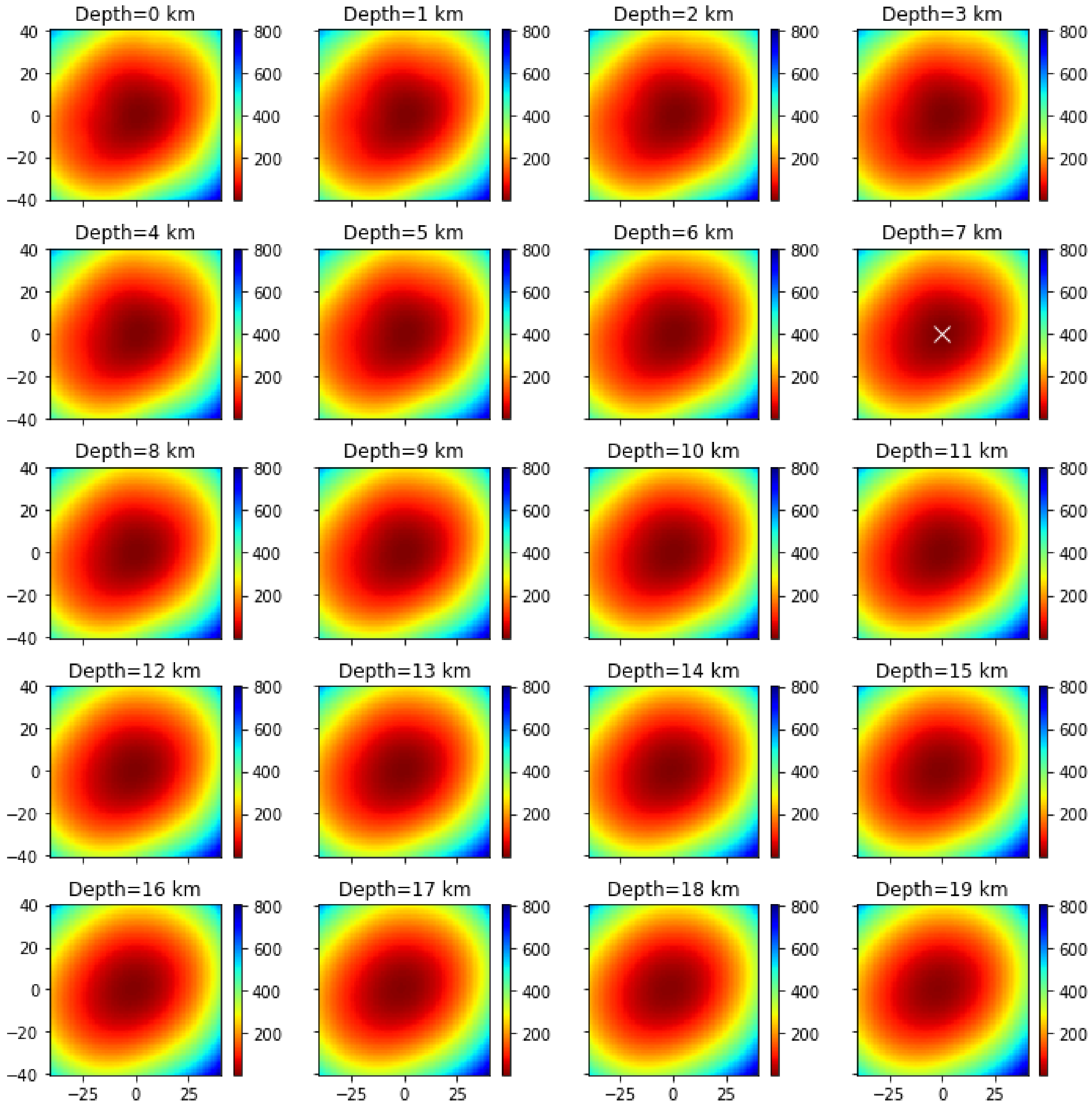
```
ncol = 4
if nz%ncol==0:
    nrow = int(nz/ncol)
else:
    nrow = int(nz/ncol)+1
xs_mesh,ys_mesh = np.meshgrid(xs,ys)
fig, axs = plt.subplots(nrow,ncol,figsize=(2.5*ncol,2*nrow),sharex=True,sharey=True)
axs = axs.ravel()
for i in range(nz):
    axs[i].pcolormesh(xs_mesh,ys_mesh,sq_errs[:,:,i],
              shading='auto',cmap='jet',vmin=sq_err_min,vmax=sq_err_max)
    plt.sca(axs[i])        # set current active axis
    plt.colorbar(pm)
    tmp_sq_err_min = np.min(sq_errs[:,:,i])
    _tmp_sq_err_min = format(tmp_sq_err_min,'6.3f')
    tmp_kk = np.where(sq_errs[:,:,i]==tmp_sq_err_min)
    idx = tmp_kk[0][0]
    idy = tmp_kk[1][0]
```

(continues on next page)

(continued from previous page)

```
    _Z = str(zs[i]).zfill(2)
    if tmp_sq_err_min == sq_err_min:
        print(f"Z={_Z},min_sq_error={_tmp_sq_err_min}, x={xs[idx]}, y={ys[idy]},global minimum")
        axs[i].plot(xs[idx],ys[idx],'wx',ms=10)
    else:
        print(f"Z={_Z},min_sq_error={_tmp_sq_err_min}, x={xs[idx]}, y={ys[idy]}")
    axs[i].set_aspect('equal')
    axs[i].set_title(f"Depth={zs[i]} km")

# adjust plot
plt.tight_layout()
```

```
Z=00,min_sq_error= 0.915, x=0, y=2
Z=01,min_sq_error= 0.898, x=0, y=2
Z=02,min_sq_error= 0.845, x=0, y=1
Z=03,min_sq_error= 0.743, x=0, y=1
Z=04,min_sq_error= 0.613, x=0, y=1
Z=05,min_sq_error= 0.469, x=0, y=1
Z=06,min_sq_error= 0.326, x=0, y=1
Z=07,min_sq_error= 0.203, x=0, y=1
Z=08,min_sq_error= 0.119, x=0, y=1
Z=09,min_sq_error= 0.072, x=1, y=1,global minimum
Z=10,min_sq_error= 0.073, x=0, y=0
Z=11,min_sq_error= 0.147, x=1, y=0
Z=12,min_sq_error= 0.323, x=1, y=0
Z=13,min_sq_error= 0.641, x=1, y=0
Z=14,min_sq_error= 1.124, x=1, y=0
Z=15,min_sq_error= 1.757, x=1, y=-1
Z=16,min_sq_error= 2.562, x=1, y=-1
Z=17,min_sq_error= 3.594, x=1, y=-1
Z=18,min_sq_error= 4.874, x=1, y=-1
Z=19,min_sq_error= 6.422, x=1, y=-1
Z=20,min_sq_error= 8.226, x=1, y=-2
```



8.3.4 Exercise

Modify V=4.9 and redo the grid search, what do you find?

8.4 Iterative Method

The arrival time recorded by one station could be presented as:

$$T_i^k = O_k + \int_s^r u ds$$

where T_i^k is the arrival time of event k on station i, *s* is source, *r* is receiver, *u* is slowness. In the right side of equation, there are two components:

1. The origin time :math: *O_k*;
2. The travel time. It is the integral over the ray path.

It could be presented as below:

$$\mathbf{F}\mathbf{m}_{\mathbf{true}} = \mathbf{d}_{\mathbf{obs}}$$

Note the equation above is non-linear. Using Taylor Expansion, we have:

$$\mathbf{F}(\mathbf{m} + \mathbf{\Delta m}) = \mathbf{F}\mathbf{m} + \frac{\partial \mathbf{F}}{\partial \mathbf{m}}\mathbf{\Delta m} + ...,$$

where $\mathbf{m} = (x, y, z, t)$. Ingoring high-order component:

$$\mathbf{\Delta d} = \frac{\partial \mathbf{F}}{\partial \mathbf{m}}\mathbf{\Delta m}$$

It means the misfit of data is related to the misfit of earthquake location, the relationship is presented as:

$$F_i^k = T_i^k = O_k + \int_s^r u ds$$

$$\frac{\partial \mathbf{F}}{\partial \mathbf{m}} = \frac{\partial T}{\partial x}\Delta x + \frac{\partial T}{\partial y}\Delta y + \frac{\partial T}{\partial z}\Delta z + \frac{\partial T}{\partial t}\Delta t$$

More in detail:

$$\begin{cases} \frac{\partial T}{\partial x} = dx/ds \cdot u \\ \frac{\partial T}{\partial y} = dy/ds \cdot u \\ \frac{\partial T}{\partial z} = dz/ds \cdot u \\ \frac{\partial T}{\partial t} = 1 \end{cases}$$

where $ds = \sqrt{(dx)^2 + (dy)^2 + (dz)^2}$

For one-layer model, $T_i^k = o_t + \sqrt{x^2 + y^2 + z^2}/v$, where x, y, z denotes distance between the source (earthquake location) and receiver(station), v is velocity. Partial derivatives of one-layer model are:

$$\frac{\partial T_i^k}{\partial x} = \frac{x}{\sqrt{x^2 + y^2 + z^2}v}$$

$$\frac{\partial T_i^k}{\partial y} = \frac{y}{\sqrt{x^2 + y^2 + z^2}v}$$

$$\frac{\partial T_i^k}{\partial y} = \frac{z}{\sqrt{x^2 + y^2 + z^2}v}$$

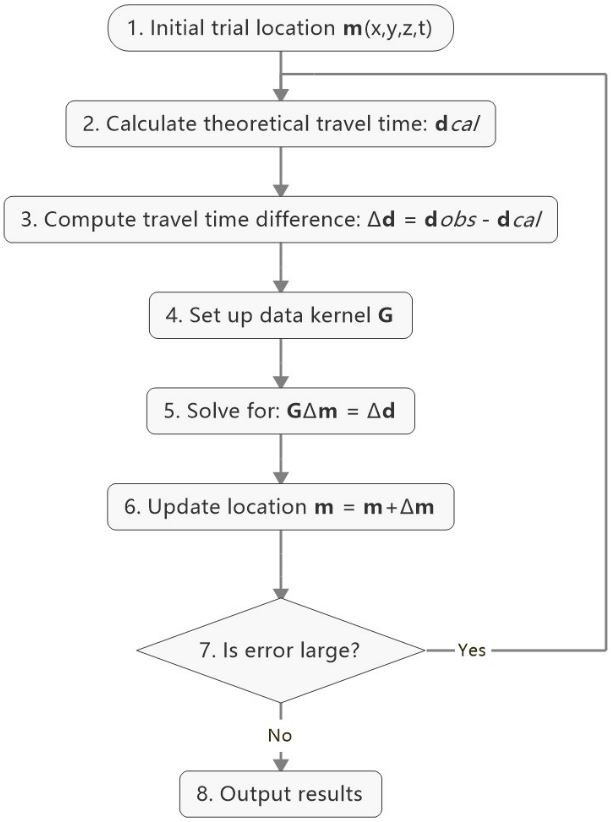
$$\frac{\partial T_i^k}{\partial o_t} = 1$$

$$\begin{bmatrix} \frac{\partial T_1}{\partial x} & \frac{\partial T_1}{\partial y} & \frac{\partial T_1}{\partial z} & 1 \\ \frac{\partial T_2}{\partial x} & \frac{\partial T_2}{\partial y} & \frac{\partial T_2}{\partial z} & 1 \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial T_i}{\partial x} & \frac{\partial T_i}{\partial y} & \frac{\partial T_i}{\partial z} & 1 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta t \end{bmatrix} = \begin{bmatrix} d_1^{obs} - d_1^{cal} \\ d_2^{obs} - d_2^{cal} \\ \vdots \\ d_i^{obs} - d_i^{cal} \end{bmatrix}$$

After solve this equation, we can update the earthquake location:

$$\mathbf{m} = \mathbf{m} + \mathbf{\Delta m}$$

This process generally will not finish in one iteration, more iterations are needed to update the locations until no apparent change of misfit.


1. Initial trial location **m**(x,y,z,t)
2. Calculate theoretical travel time: **d***cal*
3. Compute travel time difference: Δ**d** = **d***obs* - **d***cal*
4. Set up data kernel **G**
5. Solve for: **G**Δ**m** = Δ**d**
6. Update location **m** = **m**+Δ**m**
7. Is error large?
Yes
No
8. Output results

8.4.1 1. Give an initial source parameters

The station which records the earliest arrival is the cloest to the hypocenter, so it is reasonable to be set as initial location:

1. The same x and y with the closest station;
2. Initial depth at 5 km;
3. Initial origin time 1 sec before the earliest arrival;

```
idx = np.argmin(dobs)          # The index of station
dmin = np.min(dobs)            # The minimum arrival time

hyc_init = np.zeros(4);        # Init array
```

(continues on next page)

(continued from previous page)

```
hyc_init[:2] = stas[idx,:2]; # Set the same x,y with station
hyc_init[2] = 5;             # Set initial depth 5 km
hyc_init[3] = dmin-1;        # Set initial event time 1s earlier than arrival
hyc_loop = hyc_init.copy()
```

8.4.2 2. Calculate the arrival times based on input location

```
dcal = np.zeros((nsta,1))
for i in range(dobs.shape[0]):
    dx = stas[i,0]-hyc_loop[0]
    dy = stas[i,1]-hyc_loop[1]
    dz = stas[i,2]-hyc_loop[2]
    dcal[i,0] = np.sqrt(dx**2+dy**2+dz**2)/Vp+hyc_loop[3]
```

8.4.3 3. Measure the misfit between the d_{obs} and the d_{cal}

```
delta_d = dobs - dcal
e2 = 0
for i in range(delta_d.shape[0]):
    e2 += delta_d[i,0]**2
print(f"The square error: ",format(e2,'5.6f'))
```

```
The square error:  49.466691
```

8.4.4 4. Calculate Partial Derivatives

```
G = np.zeros((nsta,4))
for i in range(nsta):
    for j in range(3):
        denomiter = np.sqrt((hyc_loop[0]-stas[i,0])**2+(hyc_loop[1]-stas[i,1])**2+(hyc_
↪loop[2]-stas[i,2])**2)
        G[i,j]=(hyc_loop[j]-stas[i,j])/denomiter/Vp
G[:,3]=1
```

8.4.5 5. Estimation of Δm, generalized inversion problem

Define $\Delta m = (\Delta x, \Delta y, \Delta z, \Delta t)$, the relationship between Δm and Δd is:

$$G\Delta m = \Delta d$$

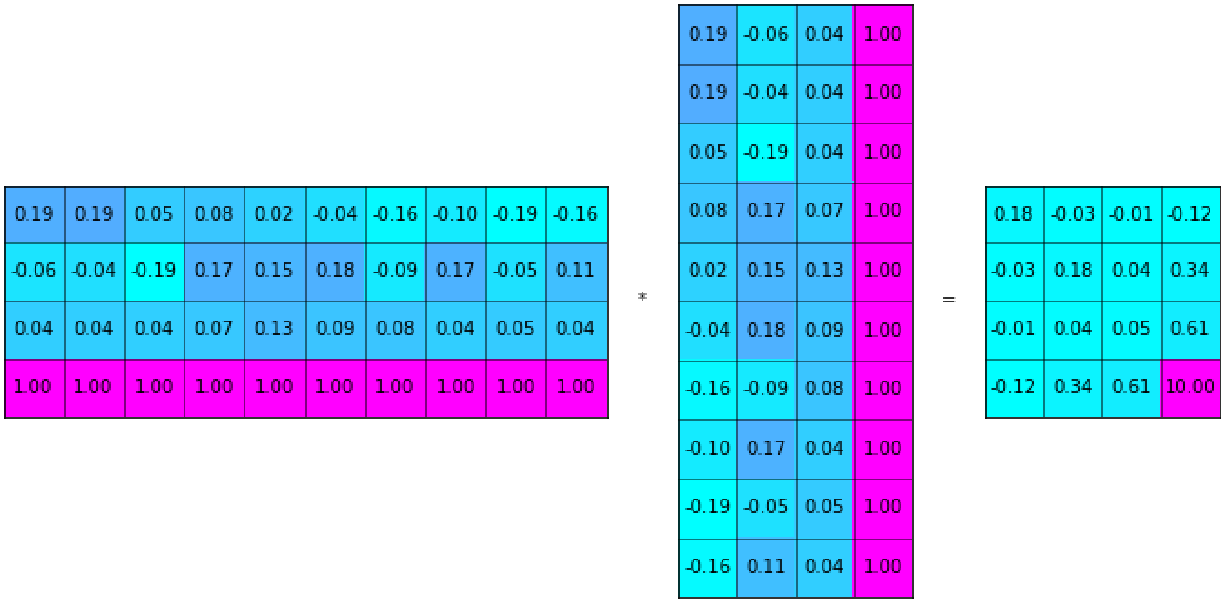
G is not a square matrix, G^TG is a square matrix, we then have:

$$G^TG\Delta m = G^T\Delta d$$

If the inverse of G^TG exists (the determinnant != 0, in here we have 10 observations to solve for 4 parameters), then:

$$\Delta m = (G^TG)^{-1}G^T\Delta d$$

```
GTG = np.matmul(G.T,G)
matrix_show(G.T,"*",G,"=",GTG)
```



$$\begin{bmatrix} 0.19 & 0.19 & 0.05 & 0.08 & 0.02 & -0.04 & -0.16 & -0.10 & -0.19 & -0.16 \\ -0.06 & -0.04 & -0.19 & 0.17 & 0.15 & 0.18 & -0.09 & 0.17 & -0.05 & 0.11 \\ 0.04 & 0.04 & 0.04 & 0.07 & 0.13 & 0.09 & 0.08 & 0.04 & 0.05 & 0.04 \\ 1.00 & 1.00 & 1.00 & 1.00 & 1.00 & 1.00 & 1.00 & 1.00 & 1.00 & 1.00 \end{bmatrix} * \begin{bmatrix} 0.19 & -0.06 & 0.04 & 1.00 \\ 0.19 & -0.04 & 0.04 & 1.00 \\ 0.05 & -0.19 & 0.04 & 1.00 \\ 0.08 & 0.17 & 0.07 & 1.00 \\ 0.02 & 0.15 & 0.13 & 1.00 \\ -0.04 & 0.18 & 0.09 & 1.00 \\ -0.16 & -0.09 & 0.08 & 1.00 \\ -0.10 & 0.17 & 0.04 & 1.00 \\ -0.19 & -0.05 & 0.05 & 1.00 \\ -0.16 & 0.11 & 0.04 & 1.00 \end{bmatrix} = \begin{bmatrix} 0.18 & -0.03 & -0.01 & -0.12 \\ -0.03 & 0.18 & 0.04 & 0.34 \\ -0.01 & 0.04 & 0.05 & 0.61 \\ -0.12 & 0.34 & 0.61 & 10.00 \end{bmatrix}$$

```
GTG_inv = np.linalg.inv(GTG)
GTG_inv_GT = np.matmul(GTG_inv,G.T)
delta_m = np.matmul(GTG_inv_GT,delta_d)
print("delta m: ",delta_m.ravel())
```

```
delta m:  [ 1.27106047 10.82922813  9.25013738 -1.91360853]
```

8.4.6 6. Update hypocenter

```
hyc_loop = np.add(hyc_loop,delta_m.ravel())
print("After this run, results (x,y,z,t) are:",hyc_loop)
print("True location parameters(x,y,z,t) are:",hyc_true)
```

```
After this run, results (x,y,z,t) are: [ 0.27106047 -0.17077187 14.25013738  0.07839748]
True location parameters(x,y,z,t) are: [0.5  0.5  9.45 0.  ]
```

8.4.7 7. Start new iteration

Move back to step two

8.4.8 8. Integrated Solution

Summarize previous steps into a loop function

```
k = 0
niter = 10
hyc_loop = hyc_init.copy()

dcal = np.zeros((10,1))
for i in range(dobs.shape[0]):
    dx = stas[i,0]-hyc_loop[0]
    dy = stas[i,1]-hyc_loop[1]
    dz = stas[i,2]-hyc_loop[2]
    dcal[i,0] = np.sqrt(dx**2+dy**2+dz**2)/Vp+hyc_loop[3]
delta_d = dobs - dcal

while k < niter:
    # >>>>> Build G matrix >>>>>>
    G = np.zeros((10,4))
    G[:,3]=1
    for i in range(10):
        for j in range(3):
            denomiter = np.sqrt((hyc_loop[0]-stas[i,0])**2+(hyc_loop[1]-stas[i,
↪1])**2+(hyc_loop[2]-stas[i,2])**2)
            G[i,j]=(hyc_loop[j]-stas[i,j])/denomiter/Vp

    # >>>>> Invert the m value >>>>
    GTG = np.matmul(G.T,G)
    GTG_inv = np.linalg.inv(GTG)
    GTG_inv_GT = np.matmul(GTG_inv,G.T)
    delta_m = np.matmul(GTG_inv_GT,delta_d)

    # >>>>> Update the hypocenter loop >>>>>
    hyc_loop = np.add(hyc_loop,delta_m.ravel())
    k = k+1
    dcal = np.zeros((10,1))
    for i in range(dobs.shape[0]):
        dx = stas[i,0]-hyc_loop[0]
        dy = stas[i,1]-hyc_loop[1]
        dz = stas[i,2]-hyc_loop[2]
        dcal[i,0] = np.sqrt(dx**2+dy**2+dz**2)/Vp+hyc_loop[3]
    delta_d = dobs - dcal
    e2 = 0
    for i in range(delta_d.shape[0]):
        e2 += delta_d[i,0]**2
    print(f"Iteration {k} square error: ",format(e2,'10.8f'))

    # >>>>> add codes to end the loop if error is small >>>>>

hyc_estimate = hyc_loop
print(hyc_estimate)
```

```
Iteration 1 square error:   1.85
```

(continues on next page)

(continued from previous page)

```
Iteration 2 square error:   0.03
Iteration 3 square error:   0.00
Iteration 4 square error:   0.00
[5.00000001e-01 5.00000005e-01 9.45000023e+00 7.74200567e-09]
```

8.4.9 Exercise (10 min)

1. Calculate the time used for the iterative location. Compare it with the grid search method.
2. It is a common practice that if the square error lower than a threshold, finish the iteration in advance. Add one criterion in above codes: if square error lows than 0.0000001, break the iteration.
3. It is common to set up an indicator parameter "istop" to show the stop reason of iteration, if iteration stops due to run over all the iterations, then istop = 0; if the iteration stops due to error threshold achieved, then istop = 1.
4. Try to change parameters, e.g. Vp, hyc_true, what's the maximum iterations needed to converge?

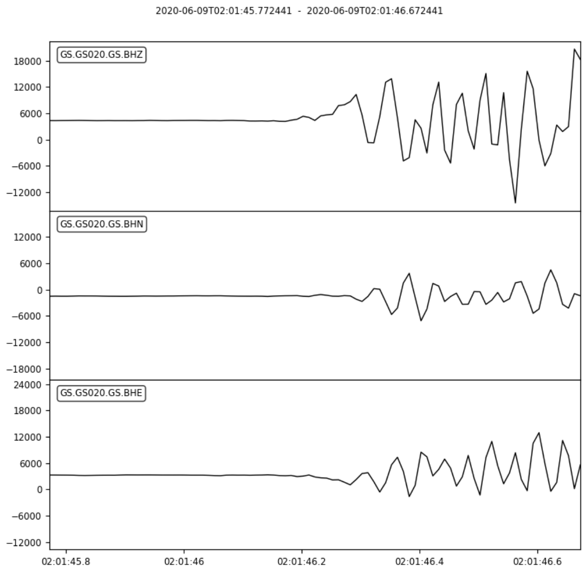
8.5 More Practical Case

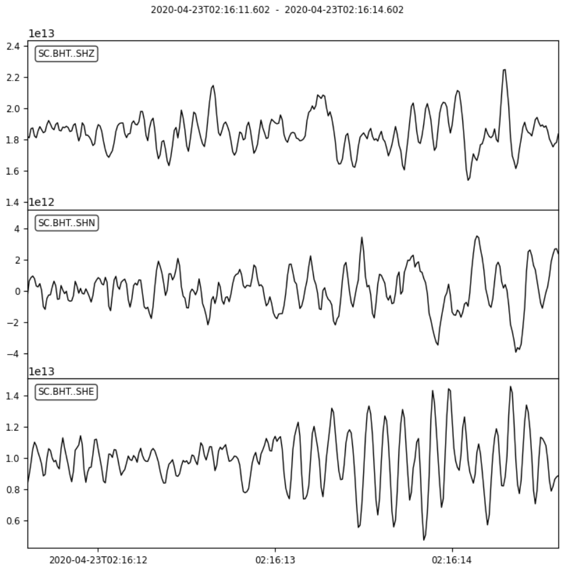
In the iterative earthquake case, we first generate the arrival times and then invert for the earthquake location, we find that it is very efficient, fast, and accurate to do so. The error decreases to nearly 0 in around 3 iterations. However, in real cases, it is rare to have error decreased to nearly 0 due to series of factors:

1. Phase picking error;
2. Time - error of stations;
3. Others.

8.5.1 Phase-Picking Error

Could you find the P arrival in below waveforms?





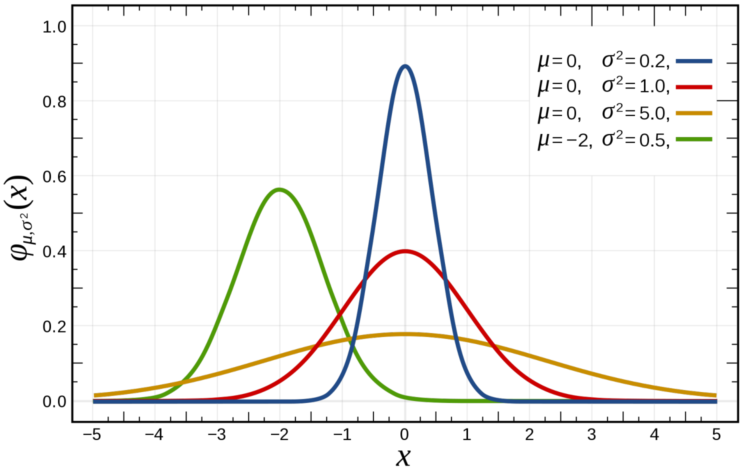
Note:

The most advanced machine learning phase-pick method has a standard error of ~0.08s in picking P phases.

It is reasonable to assume the picking errors follow the *Gaussian Distribution*, the probability we pick the phase arrival close to the true arrival is high and the probability that picked phase is far offset the true arrival is weak.

$$\sigma^2 = \frac{1}{K}\sum_{i=1}^{K}(d_i - \bar{d})^2$$

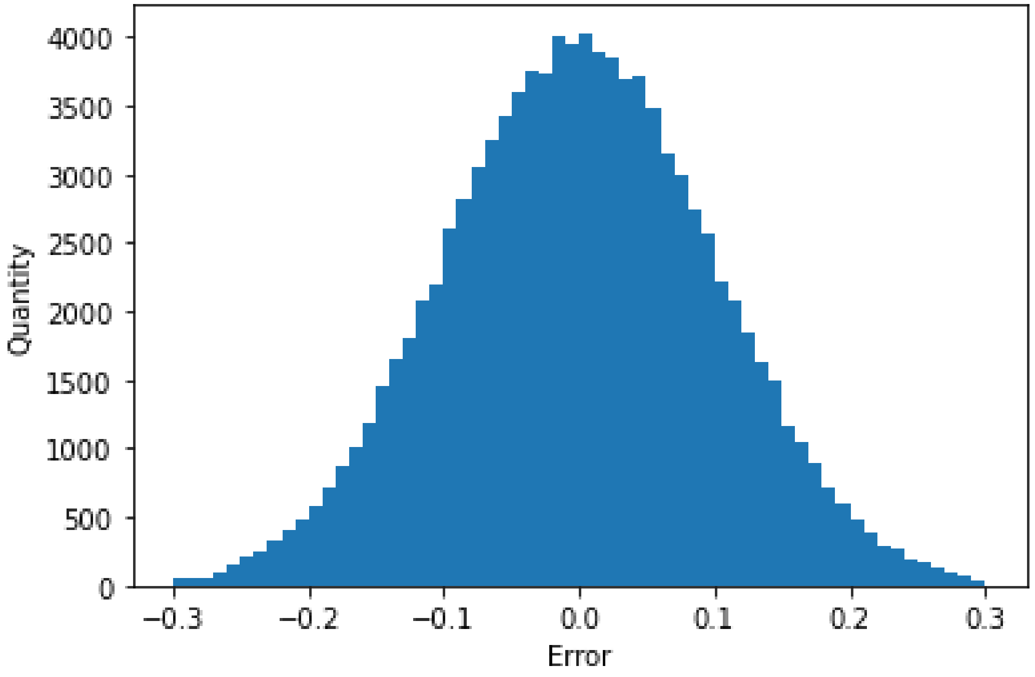
$$f(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}}exp(-\frac{(x-\mu)^2}{2\sigma^2})$$



Credit: Wikipedia

1. Generate random normal distribution error in python

```
mu = 0
sigma = 0.1
errors = np.random.normal(mu,sigma,size=(100000,1))
bins = np.arange(mu-3*sigma,mu+3*sigma,0.01)
plt.hist(errors,bins=bins);
plt.xlabel("Error")
plt.ylabel("Quantity")
```



2. Generate repeatable random normal distribution noise

```
print("Below ten sets of random data:")
for i in range(10):
    errors = np.random.normal(mu,sigma,size=(5,1))
    print(errors.ravel())

print("Below ten sets of repeatable random data:")
for i in range(10):
    seed = 5
    np.random.seed(seed)
    errors = np.random.normal(mu,sigma,size=(5,1))
    print(errors.ravel())
```

```
Below ten sets of random data:
[ 0.14576948 -0.03545659  0.01865004  0.06909433  0.10035061]
[-0.11188185 -0.00634874  0.12890032 -0.1214119  -0.07929655]
[-0.08868027  0.07272929 -0.04400131  0.03902781 -0.05310638]
[-0.19492339  0.05280531  0.01207171 -0.02196256  0.03234145]
[ 0.03812467  0.19008607  0.0689304  -0.06495476 -0.03542378]
[-0.16057787  0.00484336 -0.00963628 -0.09241747 -0.10234195]
[-0.0997116   0.07139755 -0.03709032  0.07398414 -0.04919343]
```

(continues on next page)

(continued from previous page)

```
[ 0.04309643  0.01775167  0.11226868 -0.03265422  0.29264822]
[-0.10930484  0.03639013  0.08391139  0.0606412  -0.07792868]
[-0.00797514 -0.08165227  0.04543699  0.0669631  -0.16680696]
Below ten sets of repeatable random data:
[ 0.04412275 -0.03308702  0.24307712 -0.02520921  0.01096098]
[ 0.04412275 -0.03308702  0.24307712 -0.02520921  0.01096098]
[ 0.04412275 -0.03308702  0.24307712 -0.02520921  0.01096098]
[ 0.04412275 -0.03308702  0.24307712 -0.02520921  0.01096098]
[ 0.04412275 -0.03308702  0.24307712 -0.02520921  0.01096098]
[ 0.04412275 -0.03308702  0.24307712 -0.02520921  0.01096098]
[ 0.04412275 -0.03308702  0.24307712 -0.02520921  0.01096098]
[ 0.04412275 -0.03308702  0.24307712 -0.02520921  0.01096098]
[ 0.04412275 -0.03308702  0.24307712 -0.02520921  0.01096098]
[ 0.04412275 -0.03308702  0.24307712 -0.02520921  0.01096098]
```

3. Update the observed data by adding noise

```
mu = 0         # mean of error
sigma = 0.1    # standard deviation of error
np.random.seed(100)
errors = np.random.normal(mu,sigma,size=(nsta,1))
dobs_noise = dobs+errors
```

4. Re-run the inversion

```
Vp = 5
k = 0
niter = 10
hyc_loop = hyc_init.copy()
while k < niter:
    dcal = np.zeros((10,1))
    for i in range(dobs_noise.shape[0]):
        dx = stas[i,0]-hyc_loop[0]
        dy = stas[i,1]-hyc_loop[1]
        dz = stas[i,2]-hyc_loop[2]
        dcal[i,0] = np.sqrt(dx**2+dy**2+dz**2)/Vp+hyc_loop[3]
    delta_d = dobs_noise - dcal
    e2 = 0
    for i in range(delta_d.shape[0]):
        e2 += delta_d[i,0]**2
    print(f"Iteration {k} square error: ",format(e2,'5.2f'))

    # >>>>> Build G matrix >>>>>>
    G = np.zeros((10,4))
    G[:,3]=1
    for i in range(10):
        for j in range(3):
            denomiter = np.sqrt((hyc_loop[0]-stas[i,0])**2+(hyc_loop[1]-stas[i,
↪1])**2+(hyc_loop[2]-stas[i,2])**2)
            G[i,j]=(hyc_loop[j]-stas[i,j])/denomiter/Vp

    # >>>>> Invert the m value >>>>
```

(continues on next page)

(continued from previous page)

```
    GTG = np.matmul(G.T,G)
    GTG_inv = np.linalg.inv(GTG)
    GTG_inv_GT = np.matmul(GTG_inv,G.T)
    delta_m = np.matmul(GTG_inv_GT,delta_d)
    if np.array_equal(delta_m.ravel(),[0,0,0,0]):
        print("Here")

    # >>>>> Update the hypocenter loop >>>>>
    hyc_loop = np.add(hyc_loop,delta_m.ravel())
    k = k+1

    # >>>>> End the loop if error is small >>>>>
    if e2<0.000001:
        break
hyc_estimate = hyc_loop
print(hyc_estimate)
```

```
Iteration 0 square error:  49.76
Iteration 1 square error:   1.84
Iteration 2 square error:   0.09
Iteration 3 square error:   0.07
Iteration 4 square error:   0.07
Iteration 5 square error:   0.07
Iteration 6 square error:   0.07
Iteration 7 square error:   0.07
Iteration 8 square error:   0.07
Iteration 9 square error:   0.07
[ 0.66712215  0.30531256  9.67044461 -0.03328683]
```

Exercise (2 min)

1. What do you find from the inversion? compare the results with the previous run.
2. Change the sigma value and check the variation of the inversion results.

8.5.2 Error analysis

The error in observed data will definitely lead to uncertainties in the estimation of earthquake location parameters. Their relationship could be described as:

$$\sigma_m^2 = \sigma_d^2 (G^T G)^{-1}$$

For two parameters, the definition of covariance is:

$$\sigma_{xy}{}^2 = \frac{1}{K} \sum_{k=1}^{K} (x^k - \bar{x})(y^k - \bar{y})$$

Note:

Wanna know how this relationship derived? Page 435 of **An Introduction to Seismology, Earthquakes, and Earth Structure (2003)**

```
sigma_d = np.std(delta_d)
sigma_d2 = sigma_d**2
sigma_m2 = sigma_d2 * GTG_inv
```

```
def present_loc_results(hyc,sig_square=None,std_fmt='.2f'):
    """
    Print earthquake location results
    """
    _x = format(np.round(hyc[0],4),format("6.2f"))
    _y = format(np.round(hyc[1],4),format("6.2f"))
    _z = format(np.round(hyc[2],4),format("6.2f"))
    _t = format(np.round(hyc[3],4),format("6.2f"))
    if not isinstance(sig_square,np.ndarray):
        print("x = ",_x," km")
        print("x = ",_y," km")
        print("z = ",_z," km")
        print("t = ",_t," s")
    else:
        stdx = sig_square[0,0]**0.5
        _stdx = format(np.round(stdx,4),std_fmt)
        stdy = sig_square[1,1]**0.5
        _stdy = format(np.round(stdy,4),std_fmt)
        stdz = sig_square[2,2]**0.5
        _stdz = format(np.round(stdz,4),std_fmt)
        stdt = sig_square[3,3]**0.5
        _stdt = format(np.round(stdt,4),std_fmt)
        print("x = ",_x,"±",_stdx," km")
        print("y = ",_y,"±",_stdy," km")
        print("z = ",_z,"±",_stdz," km")
        print("t = ",_t,"±",_stdt," s")
```

```
present_loc_results(hyc_estimate,sigma_m2)
```

```
x =    0.67 ± 0.20  km
y =    0.31 ± 0.22  km
z =    9.67 ± 0.99  km
t =   -0.03 ± 0.06  s
```

Question (2 min)

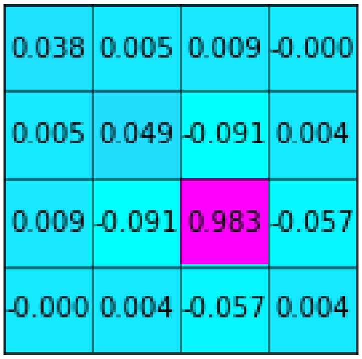
Test different parameters and see how standard error ($sigma$) changes, which parameter has the largest standard error? which parameter has the minimum standard error? Why? .. note:: | Check $(G^TG)^{-1}$, (G^TG) and G values

Covariance Matrix

$$\sigma_m^2 = \sigma_d^2(G^TG)^{-1} = \begin{bmatrix} \sigma_{xx}^2 & \sigma_{xy}^2 & \sigma_{xz}^2 & \sigma_{xt}^2 \\ \sigma_{yx}^2 & \sigma_{yy}^2 & \sigma_{yz}^2 & \sigma_{yt}^2 \\ \sigma_{zx}^2 & \sigma_{zy}^2 & \sigma_{zz}^2 & \sigma_{zt}^2 \\ \sigma_{tx}^2 & \sigma_{ty}^2 & \sigma_{tz}^2 & \sigma_{tt}^2 \end{bmatrix}$$

From the covariance matrix, we can estiamte the uncertainty(σ) of x,y,z,t using σ_x^2,σ_y^2,σ_z^2,σ_t^2

```
matrix_show(sigma_m2,fmt='.3f')
```

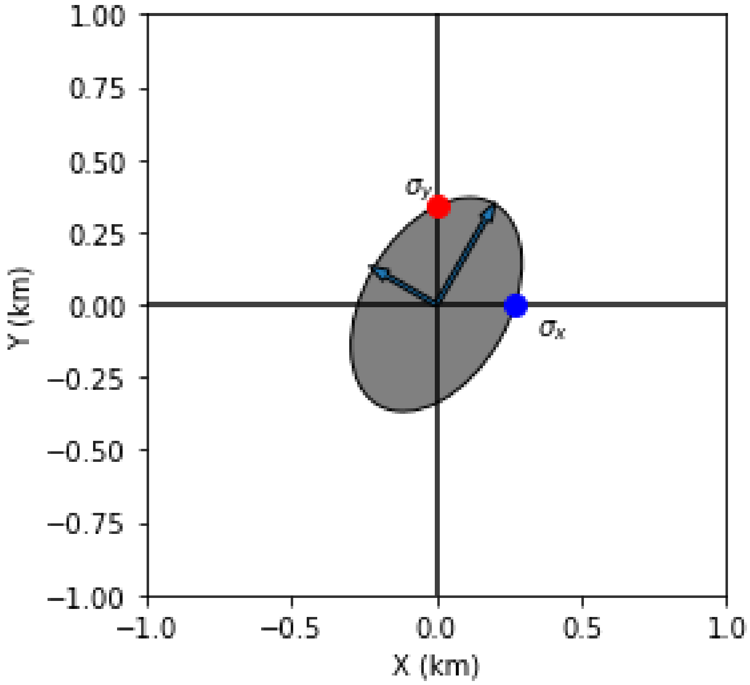


0.038	0.005	0.009	-0.000
0.005	0.049	-0.091	0.004
0.009	-0.091	0.983	-0.057
-0.000	0.004	-0.057	0.004

Principle axes

Note that off-diagonal elements of σ_m^2 is not zero. Using **xy plane** as an example, it is shape could be presented by the figure below generated. The principle axes are not along the same direction with **xy** axis.

```
angle = 30
width = 0.5
height = 0.8
ellipse = Ellipse(xy=[0,0],width=0.5,height=0.8,angle=-angle)
ellipse.set_facecolor('grey')
ellipse.set_edgecolor('black')
fig, ax = plt.subplots(subplot_kw={'aspect': 'equal'})
ax.add_artist(ellipse)

plt.xlabel("X (km)")
plt.ylabel("Y (km)")
plt.xlim([-1,1])
plt.ylim([-1,1])
plt.plot([-1,1],[0,0],'k')
plt.plot([0,0],[-1,1],'k')
plt.arrow(0,0,height/2*np.sin(np.deg2rad(angle))*0.85,height/2*np.cos(np.deg2rad(angle))*0.85,width=0.015,zorder=10)
plt.arrow(0,0,-width/2*np.cos(np.deg2rad(angle))*0.80,width/2*np.sin(np.deg2rad(angle))*0.80,width=0.015,zorder=10)
plt.plot(0.27,0,'o',color='blue',ms=8)
plt.plot(0,0.34,'o',color='red',ms=8)
plt.text(0.34,-0.1,'$\sigma_x$')
plt.text(-0.12,0.38,'$\sigma_y$')
plt.show()
```



Singular Value Decomposition (SVD) could be used to find the principle axes and principle values.

$$M = USV^T$$

S is the ordered eigenvalues array. V is the corresponding eigenvectors. Below demonstrate the decomposition of errors in xy-plane.

```
sigma_xy2 = sigma_m2[:2,:2]
u,s,vt = np.linalg.svd(sigma_xy2)
print("Maximum eigenvalue: ",format(s[0],'.5f')," corresponding eigenvector: ",vt[0,:])
print("Minimum eigenvalue: ",format(s[-1],'.5f')," corresponding eigenvector: ",vt[-1,:])
print("The maximum/minimum eigenvalue ratio: ",format(s[0]/s[1],'.2f'))
```

```
Maximum eigenvalue:  0.05139  corresponding eigenvector:  [0.35995123 0.93297112]
Minimum eigenvalue:  0.03604  corresponding eigenvector:  [ 0.93297112 -0.35995123]
The maximum/minimum eigenvalue ratio:  1.43
```

Plot the error ellipse and stations

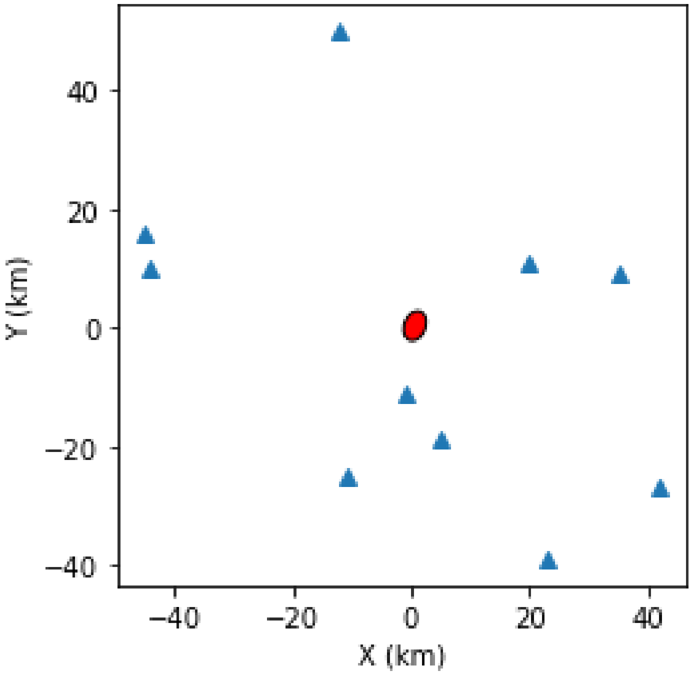
Note: the sigma values are small to be shown, here amplify the size by parameter **size_ratio**

```
angle = np.arctan(vt[0,0]/vt[0,1])/np.pi*180
size_ratio = 100
ellipse = Ellipse(xy=[hyc_estimate[0],hyc_estimate[1]],width=s[1]*size_ratio,
↪height=s[0]*size_ratio,angle=-angle)
ellipse.set_facecolor('red')
ellipse.set_edgecolor('black')
fig, ax = plt.subplots(subplot_kw={'aspect': 'equal'})
ax.add_artist(ellipse)

plt.plot(stas[:,0],stas[:,1],'^',label="Station")
```

(continues on next page)

(continued from previous page)

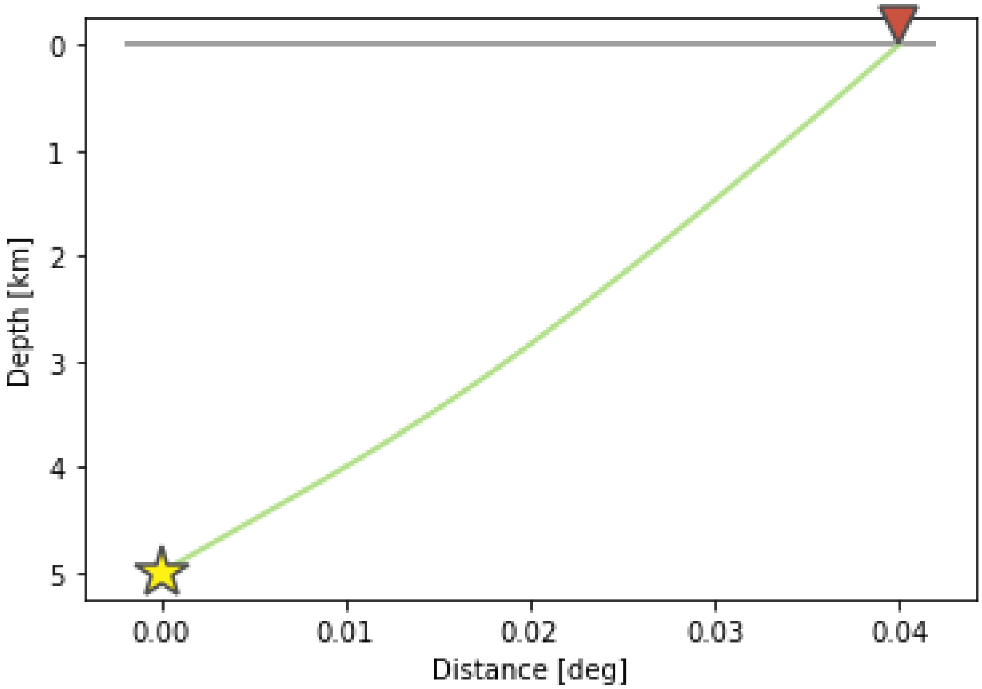
```
plt.xlabel("X (km)")
plt.ylabel("Y (km)")
plt.show()
```



8.6 Summary

8.6.1 One layer model

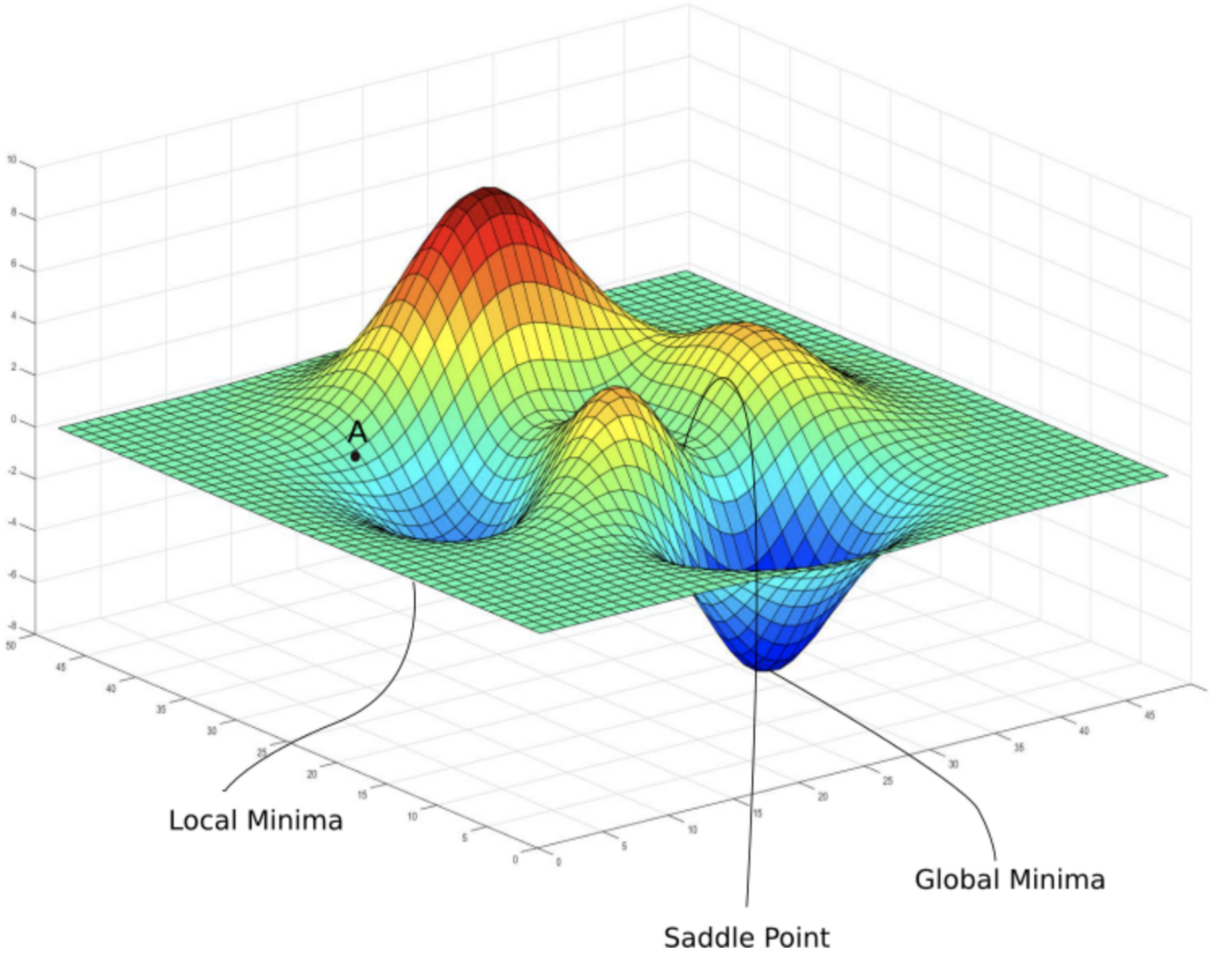
In the tutorial, we introduced the grid-search method and iterative location method using the one-layer velocity model. The advantage of one-layer is that the ray from the source to one station is a stright line, it is thus convenient to calculate the corresponding partial derivatives. In the real earth, however, the velocity varies due to material, pressure and other fators, the ray path is therefore a curved line, making things more complicated.



However, the key process in finding the earthquake locations remains the same.

8.6.2 The grid search method and the iteraive method

In this tutorial, using the **iterative method**, we can converge the minimum error location in limited iterations with the random initial location we set. However, in practical cases, due to the complexity of station coverage, velocity structure, and other factors, a random initiation might lead to local minimum rather than global minimum.



(Courtesy of https://medium.com/analytics-vidhya/journey-of-gradient-descent-from-local-to-global-c851eba3d367) The general solution is to **conduct rough grid-search first**, which could **avoid local minimum** effectively. Then run the iterative method from the grid search minimum.

8.6.3 Convenient functions

```
def iter_loc(hyc_loop,stas,dobs,V,niter=10,show=True):
    """
    Do iterative earthquake location
    Parameters:
    | hyc_loop: hypoceter for iteration
    |     stas: array contains stations location
    |     dobs: observed travel time
    Return:
    | hyc_loop: earthquake location after iteration
    | sigma_m2: square sigma matrix
    |  sigma_d: root mean square residual
    """
    nobs = dobs.shape[0]
    k = 0
    while k < niter:
```

(continues on next page)

(continued from previous page)

```
        dcal = np.zeros((nobs,1))
        for i in range(dobs.shape[0]):
            dx = stas[i,0]-hyc_loop[0]
            dy = stas[i,1]-hyc_loop[1]
            dz = stas[i,2]-hyc_loop[2]
            dcal[i,0] = np.sqrt(dx**2+dy**2+dz**2)/V+hyc_loop[3]
        delta_d = dobs - dcal
        e2 = 0
        for i in range(nobs):
            e2 += delta_d[i,0]**2
        if show:
            print(f"Iteration {k} square error: ",format(e2,'5.2f'))

        # >>>>> Build G matrix >>>>>>
        G = np.zeros((nobs,4))
        G[:,3]=1
        for i in range(dobs.shape[0]):
            for j in range(3):
                denomiter = np.sqrt((hyc_loop[0]-stas[i,0])**2+(hyc_loop[1]-stas[i,
↪1])**2+(hyc_loop[2]-stas[i,2])**2)
                G[i,j]=(hyc_loop[j]-stas[i,j])/denomiter/V

        # >>>>> Invert the m value >>>>
        GTG = np.matmul(G.T,G)
        GTG_inv = np.linalg.inv(GTG)
        GTG_inv_GT = np.matmul(GTG_inv,G.T)
        delta_m = np.matmul(GTG_inv_GT,delta_d)

        # >>>>> Update the hypocenter loop >>>>>
        hyc_loop = np.add(hyc_loop,delta_m.ravel())
        k = k+1

        # >>>>> End the loop if error is small >>>>>
        if e2<0.0000001:
            break
    sigma_d = np.std(delta_d)
    sigma_d2 = sigma_d**2
    sigma_m2 = sigma_d2 * GTG_inv
    return hyc_loop, sigma_m2, sigma_d

hyc_abs, sigma_m2, e2 = iter_loc(hyc_init,stas,dobs,Vp)
```

```
Iteration 0 square error:  49.47
Iteration 1 square error:   1.85
Iteration 2 square error:   0.03
Iteration 3 square error:   0.00
Iteration 4 square error:   0.00
```

```
def present_loc_results(hyc,sig_square=None,std_fmt='.2f'):
    """
    Print earthquake location results
```

(continues on next page)

(continued from previous page)

```
    Parameters:
    |         hyc: hypocenter
    |sigma_square: squared sigma matrix
    |     std_fmt: format control of the output uncertainty
    """
    _x = format(np.round(hyc[0],4),format("6.2f"))
    _y = format(np.round(hyc[1],4),format("6.2f"))
    _z = format(np.round(hyc[2],4),format("6.2f"))
    _t = format(np.round(hyc[3],4),format("6.2f"))
    if not isinstance(sig_square,np.ndarray):
        print("x = ",_x," km")
        print("x = ",_y," km")
        print("z = ",_z," km")
        print("t = ",_t," s")
    else:
        stdx = sig_square[0,0]**0.5
        _stdx = format(np.round(stdx,4),std_fmt)
        stdy = sig_square[1,1]**0.5
        _stdy = format(np.round(stdy,4),std_fmt)
        stdz = sig_square[2,2]**0.5
        _stdz = format(np.round(stdz,4),std_fmt)
        stdt = sig_square[3,3]**0.5
        _stdt = format(np.round(stdt,4),std_fmt)
        print("x = ",_x,"±",_stdx," km")
        print("y = ",_y,"±",_stdy," km")
        print("z = ",_z,"±",_stdz," km")
        print("t = ",_t,"±",_stdt," s")

    present_loc_results(hyc_abs,sigma_m2,std_fmt='.4f')
```

```
x =    0.50 ± 0.0000  km
y =    0.50 ± 0.0000  km
z =    9.45 ± 0.0000  km
t =   -0.00 ± 0.0000  s
```

8.6.4 Play around new station dataset

```
stas_set2 = np.array([[-45,36,0],
                 [-44,30,0],
                 [-12,50,0],
                 [8,-40,0],
                 [-1,-11,0],
                 [20,-19,0],
                 [20,0,0],
                 [23,-39,0],
                 [35,-5,0],
                 [42,-27,0]])
stas = stas_set2
```

8.7 Homework

1. Using the second station dataset (stats_set2), run the inversion with noise parameters (seed=100, mu=0,sigma=0.1), plot the error ellipse and stations, could you conclude relationship between the error ellipse and the stations coverage? Show your codes and results(30 Points)
2. In order to enhance the Z constraint, you can change the location of one station in station dataset1, what's your plan and why? Show your codes and results (20 points)
3. In previous example, we calculate the $\sigma_d^2 = \frac{1}{nobs}\sum_{i=1}^{nobs}(d_i - \bar{d})^2$, note it is the sum of square error divided by $nobs$ (number of observations). There are scientists proposed that the calculation should be $\sigma_d^2 = \frac{1}{nobs-k}\sum_{i=1}^{nobs}(d_i - \bar{d})^2$, where $nobs - k$ is called **the number of degrees of freedom**, k is the number of parameters determined by the data, in earthquake location process, $k = 4$ for four paramters (x,y,z,t) are inverted. Try to run the inversion 100 times with random noise $\sigma_{true} = 0.1s$, calculate the data standard error using two methods, conclude which one is more consistent with the input noise level. Show your codes and results (30 points)
4. What's your comments and suggestions to this tutorial (10 points)

8.8 Tutorial source code

Download `here`

8.9 HYPOINVERSE Tutorial

Previous python tutorial gives intuitive familarities of the earthquake relocation process, here we further prepared a tutorial of widely used earthquake absolute location tool, the HYPOINVERSE.

8.9.1 Introduction of HYPOINVERSE

Hypoinverse is a computer program that processes files of seismic station data for an earthquake (like p wave arrival times and seismogram amplitudes and durations) into earthquake locations and magnitudes (Klein, 2002). It is a single event location method.

The Hypoinverse program requires the input of station locations, seismic velocity model, and the phase data. By assuming a trial origin time and hypocentral location for the earthquake, it improves them by iteratively minimizing the least square error of the travel time computed from the input information.

8.9.2 Environment and example

For MacOS user, Xcode is needed to be installed, run `xcode-select --install` and wait for finishment.

`HYPOINVERSE example`

CHAPTER

NINE

FOCAL MECHANISM

Waveforms recorded at a seismic station, W(t), compose of three components:

W(t) = S(t) * G(t) * I(t); (1)

where S(t) represents the source, G(t) stands for the Green's functions, and I(t) is the instrument response. Therefore, retrieving source parameters of one earthquake S(t) requires deconvolving instrument response I(t) and Green's functions G(t) from the actual data W(t). We generally remove instrument responses after obtaining the waveform data. Therefore, we have the data in displacement or in velocity, u(t) = S(t) * G(t). The corresponding synthetic displacement s(t) for a douple-couple source can be expressed as

s(t) = M0 Ai(-,,)Gi(t); (2)

where Gi are the Green's functions, Ai are the radiation coeffcients, and is the station azimuth, M0 is scalar moment,,, and are strike, dip, and rake, respectively. Then we perform a grid search in all possible solutions of strike, dip, and rake to obtain the best fit by finding the minimal residual between the data and synthetics. This procedure consists of the following three steps, (1) computing Green's functions, (2) preparing data seismograms, and (3) deriving focal mechanism solutions. This tutorial will mostly cover step (3).

Here we compute focal mechanism solutions using the Cut and Paste" method (CAP) [Zhu and Helmberger, 1996]. This method decomposes seismograms and uses amplitude information in different time windows (e.g., Pnl/surface wave) to increase the stability and resolution of focal mechanism solution. Please refer to [Zhu and Helmberger, 1996] for details of the CAP method. The earthquake example used in the gCAP program is the 2008 Illinois Mw 5.2 earthquake. It is a left-lateral strike-slip event based on focal mechanism solutions and aftershock locations [Yang et al., 2009].

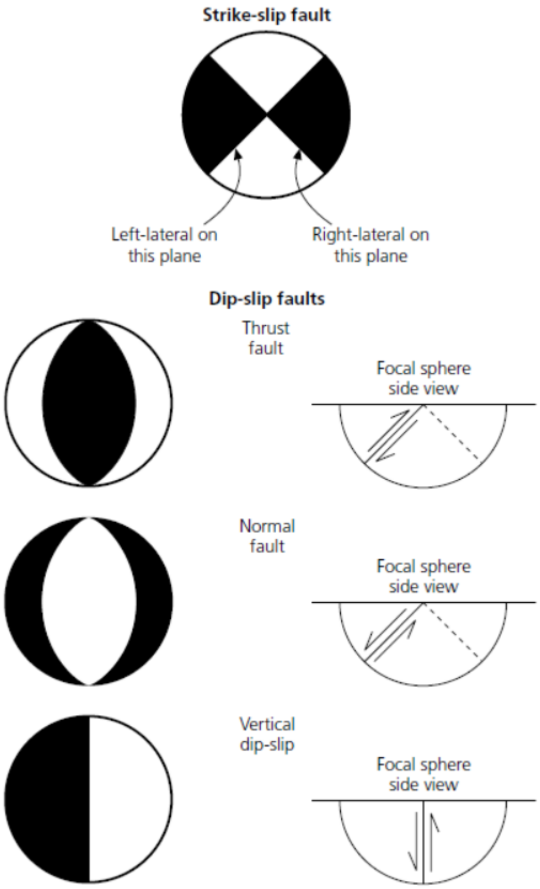
——— Prof. Hongfeng Yang

Developed by Han CHEN.

9.1 1 Introduction

9.1.1 1.1 What is focal mechanism

To describe the geometry of a fault, Scientists assume that the fault is a planar surface across which slip occurred during an earthquake. The focal mechanism describes the fault geometry of the seismogenic fault (e.g., strike(), dip()) and the slip in the fault (e.g., rake()). The beach ball were the lower-hemisphere stereographic projection of the fault geometry.



9.1.2 1.2 package required

- `fk` package
- Seismic Analysis Code (`SAC`) package
- `gCAP3D` package

9.2 2 Installation

9.2.1 2.1 fk

fk is a program written by Prof. Lupei ZHU for calculating the synthetic Green's function based on a horizontally layered velocity model. The detailed installation of fk could be find on here.

1. Files download

```
## Download
$ wget http://www.eas.slu.edu/People/LZhu/downloads/fk3.3.tar
## Unzip
$ tar -xvf fk3.3.tar
## make dir "~/Src" and move the fold to "~/Src"
```

(continues on next page)

(continued from previous page)

```
$ mkdir -p ~/Src/
$ mv fk ~/Src/
```

2. File modification and compilation

There are some errors in the source code of fk. Please `download` the file and put it into the `~/Src/fk` for modifying the Make file.

```
$ cd ~/Src/fk/
## modify the Make file
$ patch < fk3.3-v20190618.patch
## compilation
$ make clean
$ make
## Modify the environment variable and add the fk path to PATH
$ echo 'export PATH=${HOME}/Src/fk/:${PATH}'>> ~/.bashrc
$ source ~/.bashrc
```

3. Run fk

Type `fk.pl` in the terminal, there will be help information if installation is succeeded

```
$ fk.pl
Usage: fk.pl -Mmodel/depth[/f_or_k] [-D] [-Hf1/f2] [-Nnt/dt/smth/dk/taper] [-Ppmin/pmax[/kmax]] [-Rrdep] [-SsrcType] [-Uupdn] [-Xcmd] distances ...
...
```

9.2.2 2.2 Seismic Analysis Code (SAC)

SAC is one of the most widely used data analysis software packages in the field of natural seismology. The installation of SAC could be found at the website.

Apply SAC source file

You will need to submit an application online to obtain the SAC package from IRIShttp://ds.iris.edu/ds/nodes/dmc/forms/sac/

2.2.1 Installation for linux

1. Install dependent libraries

> for Ubuntu/Debian:

```
$ sudo apt update
$ sudo apt install libc6 libsm6 libice6 libxpm4 libx11-6
$ sudo apt install zlib1g libncurses5
```

> for CentOS/Fedora/RHEL:

```
$ sudo yum install glibc libSM libICE libXpm libX11
$ sudo yum install zlib ncurses-compat-libs
```

2. Install binary packages

```
## Unizp
$ tar -xvf sac-102.0-linux_x86_64.tar.gz
## Move to installation position
$ sudo mv sac ~/opt
```

3. Configuration variable

Add the environment variables and SAC global variables in `~/.bashrc`

```
$ echo 'export SACHOME=~/opt/sac'>> ~/.bashrc
$ echo 'export SACAUX=${SACHOME}/aux'>> ~/.bashrc
$ echo 'export PATH=${SACHOME}/bin:${PATH}'>> ~/.bashrc
$ echo 'export SAC_DISPLAY_COPYRIGHT=1'>> ~/.bashrc
$ echo 'export SAC_PPK_LARGE_CROSSHAIRS=1'>> ~/.bashrc
$ echo 'export SAC_USE_DATABASE=0'>> ~/.bashrc
$ source ~/.bashrc
```

4. Run sac

Type `sac` in the terminal, there will be version information if installation is succeeded

```
$ sac
 SEISMIC ANALYSIS CODE [11/11/2    3 (Version 1    .6a)]
 Copyright 1995 Regents of the University of California

SAC>
```

2.2.2 Installation for MacOS

1. Preparation

Install command line tools and X11 graphical interface related tools (XQuartzunder) MacOS

```
$ xcode-select --install
$ brew install --cask xquartz
```

2. Install binary packages

```
## Unizp
$ tar -xvf sac-102.0-mac.tar.gz
## Move to installation position
$ sudo mv sac ~/opt
```

3. Configuration variable

Add the environment variables and SAC global variables in `~/.zshrc`

```
$ echo 'export SACHOME=~/opt/sac'>> ~/.zshrc
$ echo 'export SACAUX=${SACHOME}/aux'>> ~/.zshrc
$ echo 'export PATH=${SACHOME}/bin:${PATH}'>> ~/.zshrc
```

(continues on next page)

(continued from previous page)

```
$ echo 'export SAC_DISPLAY_COPYRIGHT=1'>> ~/.zshrc
$ echo 'export SAC_PPK_LARGE_CROSSHAIRS=1'>> ~/.zshrc
$ echo 'export SAC_USE_DATABASE=0'>> ~/.zshrc
$ source ~/.zshrc
```

4. Run sac

Type `sac` in the terminal, there will be version information if installation is succeeded

```
$ sac
 SEISMIC ANALYSIS CODE [11/11/2    3 (Version 1    .6a)]
 Copyright 1995 Regents of the University of California

SAC>
```

9.2.3 2.3 gCAP3D

gCAP is a method for inversion of focal mechanism solutions developed by Prof. Lupei Zhu. The source code of gCAP could be found in His Home page.

The code is now open source. The installation of gCAP could be found on seisman website.

Here I integrated the file in the website and gave the modified package file for you `download`.

1. Download the file and unzip

```
## Download
## Unizp
$ tar -xvf gCAP3D.1.0.Cuseistut.tar
```

2. Download supplementary file and unzip

Refer to the seisman website above to download gcap_utils.tar.gz, unzip and put all files into the gCAP3D directory.

3. Install binary packages

```
## Move to installation position
$ sudo mv gCAP3D ~/Src
## change dir to gCAP3D
$ cd ~/Src/gCAP3D
## compilation
$ make clean
$ make
```

4. Configuration variable

Add the environment variables and SAC global variables in `~/.bashrc`

```
$ echo 'export PATH=~/Src/gcap:${PATH}'>> ~/.bashrc
$ source ~/.bashrc
```

5. Modify directory

Open `cap3D.pl`, change the directory on Line 17 from '~/Src/gCAP3D/cap_plt.pl' to '{your home directory}/Src/gCAP3D/cap_plt.pl'

6. Run gCAP3D

Type `cap3D.pl` in the terminal, there will be version information if installation is succeeded

```
$ gCAP3D.pl
  ===== CAP seismic source tensor inversion using seismic waveforms ====
  Ref: Zhu and Helmberger, 1996, BSSA 86, 1645-1641.
  Zhu and Ben-Zion, 2    3, GJI, submitted.
  ...
```

9.3 3 Data processing

Tip: The steps of focal mechanism inversion using gCAP3D

1. Calculating the Green's function
2. Remove the instrument response (PZ or RESP file) to get the real ground motion (requiring response files; find the details in sac manual).
3. Rotate the NEZ components to RTZ direction (ps, remember to remove the original NEZ data in the event folder). An introduction to these two coordinate systems can be found at this web
4. Transform the data into the unit of cm/s by multiplying `1e-7` (default unit in SAC data is `nm`), which is the default unit in CAP.
5. Set the onset time of the event as the reference time (the zero point in the time series). Find the details in sac manual.
6. Pick the p arrival time and write the time in t1 marker, (for the noisy data, band pass filter can be applied to pick the p arrival more accurately.) Remember that the bandpass filtered data should NOT be saved in the folder. All the sac file in the folder will be used in waveform inversion.
7. Remember to remove the waveform data from the folder whose P arrival is too dim to be seen.
8. Resample the data to that the same as the Green's function
9. Set up the "info.eve" file under each event folder. It is a five-column file, of which the format is: "o-marker evla evlo evdp mag".
10. To run CAP:

    (1) Input command: make ${event_name}/weight.dat; and then write the p arrival time – t1 into the eighth column of "weight.dat";

    (2) Input command: make ${event_name}/mt.best to start the waveform inversion process.
11. check the uncertainty

Note: Note that the unit of displacement data obtained by transferring the PZ file is `m`; the default unit of SAC is `nm`, so it must be multiplied by `1e9` to convert to the default unit of SAC.

9.3.1 3.1 Green's function calculation

The first step of runing gCAP is building a Green's function library by using fk package. An velocity model file (GD.vel) is need for the calculation. `here` is the velocity model used in this tutorial. The default setting of `gCAP` puts the Green's function library under `~/data/models/Glib`

Tip: There are two types of velocity model that could be used in fk, with the third column is the Vs or the vp/vs ratio. `-Kmodel/depth/k`, should be used while the 3rd column is vp/vs ratio (vp). The -S parameter controls the source type. -S: 0=explosion; 1=single force; 2=double couple (2).

```
## mkdir the Green's function library directory
$ mkdir ~/data
$ mkdir ~/data/models
$ mkdir ~/data/models/Glib
## move to Green's function library directory
$ cd ~/data/models/Glib
## build a directory for velocity model GD
$ mkdir GD
$ cd GD
## cp GD.vel to GD directoty, please replace the '/path/to/fk/GD.vel' to the real path␣
↪in your system
$ cp /path/to/fk/GD.vel .
## calculate the Green's function library using fk
$ fk.pl -MGD.vel/15/k -N512/0.2 -S2 05 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85␣
↪90 95 100 105 110 115 120 125 130 135 140 145 150 155 160 165 170 175 180 185 190 195␣
↪200 205 210 215 220 225 230 235 240 245 250 255 260 265 270 275 280 285 290 295 300␣
↪305 310 315 320 325 330 335 340 345 350 355 360 365 370 375 380 385 390 395 400 405␣
↪410 415
$ fk.pl -MGD.vel/15/k -N512/0.2 -S0 05 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85␣
↪90 95 100 105 110 115 120 125 130 135 140 145 150 155 160 165 170 175 180 185 190 195␣
↪200 205 210 215 220 225 230 235 240 245 250 255 260 265 270 275 280 285 290 295 300␣
↪305 310 315 320 325 330 335 340 345 350 355 360 365 370 375 380 385 390 395 400 405␣
↪410 415
```

9.3.2 3.2 Sac file preparation

The sac file preparation could be done by using `SAC` software or other softwares, such as the `Obspy`. Here we briefly introduce the steps while using `SAC`. The details of commands used in this chapater could be found in the sac manual.

1. Remove the instrument response

> Use PZ file:

```
## Merge PZ files of all stations into the same file
$ cat SAC_PZs_* >> SAC.PZs
## remove the instrument response by command: transfer
$ sac
SAC> r *.SAC
## When going to the instrument response, try to choose a wider frequency band
SAC> trans from pol s SAC.PZs to none freq 0.004 0.007 40 45
SAC> mul 1.0e9
```

(continues on next page)

(continued from previous page)

```
SAC> w over
SAC> q
```

> Use RESP file:

```
## Merge RESP files of all stations into the same file
$ cat RESP.*.*.*.* >> RESP.ALL
## remove the instrument response by command: transfer
$ sac
SAC> r *.SAC
## When going to the instrument response, try to choose a wider frequency band
SAC> trans from evalresp fname RESP.ALL to none freq 0.004 0.007 40 45
SAC> w over
SAC> q
```

2. Rotate the NEZ components to RTZ direction

```
$ sac
SAC> r ./example.n ./example.e
SAC> lh cmpinc cmpaz
    FILE: ./example.n - 1
    --------------
    cmpinc = 9.000000e+
    cmpaz = 0.000000e+00
    FILE: ./example.e - 2
    --------------
    cmpinc = 9.000000e+
    cmpaz = 9.000000e+
## Rotate to Great Circle Path
SAC> rotate to gcp
SAC> lh cmpinc cmpaz
    FILE: ./example.n - 1
    --------------
    cmpinc = 9.000000e+
    cmpaz = 2.440466e+
    FILE: ./example.e - 2
    --------------
    cmpinc = 9.000000e+
    cmpaz = 1.144047e+02
## Save as R component and T component
SAC> w example.r example.t
SAC> q
```

3. Resampling the sac file

```
SAC> r example.SAC
SAC> lh delta npts

    delta = 1.000000e-02
    npts = 1000
## change the sampling rate to 0.005
SAC> interp delta 0.005
SAC> lh
```

(continues on next page)

(continued from previous page)

```

    delta = 5.000000e-03
    npts = 1999
SAC> q
```

4. Mark the P and S arrivels

The arrivels could be labeled by using SAC by using ppk command as well. please refer to the sac manual).

9.3.3 3.3 Focal mechanism inversion

After the sac files prepared, we could start to run the gCAP inversion. here we provided the SAC files of the 01/04/2020 Offshore Pearl River Delta earthquake [CHEN et.al., 2021] for you `download`. Please download it the unzip it and them change your current directory to `01-04-2020-earthquake`.

```
$ cd ./01-04-2020-earthquake
```

1. Make weight.dat file

```
#### please copy the following command to a bash script and then run the bash file.

#!/bin/bash

dir=20200104225537
cd $dir
pwd
rm temp
for sac in *z
    do
    sta=$(echo $sac | awk -F "." '{print $1}')
    dist1=$(saclst dist f $sac | awk '{print $2}')
    dist_c=$(echo $dist1 | awk '{printf ("%d\n",$1/5+0.5)}')
    dist=$(echo ""$dist_c"*5" |bc)
    P=$(saclst a f $sac | awk '{print $2}')
    #echo $dist1 $dist
    printf "%-10s %03d %-1s %-1s %-1s %-1s %-1s %3.1f %-1s\n"  $sta $dist "1" "1" "1" "1
↪" "1" $P "0" >> temp
done
cat temp | sort -n -k2 > weight.dat
```

The generated file as follows

```
$ cat weight.dat
$ HKPS       035 1 1 1 1 1 7.2 0
  ZHH        035 1 1 1 1 1 7.2 0
  HKOC       040 1 1 1 1 1 8.3 0
  SZN        060 1 1 1 1 1 10.3 0
  ZHS        065 1 1 1 1 1 11.8 0
  TIS        100 1 1 1 1 1 18.0 0
  XNH        100 1 1 1 1 1 17.1 0
  SCD        115 1 1 1 1 1 19.4 0
```

(continues on next page)

(continued from previous page)

```
  HUZ       140 1 1 1 1 1 22.7 0
  HUD       170 1 1 1 1 1 26.8 0
  ZHQ       180 1 1 1 1 1 27.9 0
  HYJ       195 1 1 1 1 1 30.1 0
  LTK       195 1 1 1 1 1 29.6 0
  XFJ       200 1 1 1 1 1 29.8 0
  YGJ       200 1 1 1 1 1 30.0 0
  XIG       205 1 1 1 1 1 31.1 0
  ZIJ       225 1 1 1 1 1 32.9 0
  YND       230 1 1 1 1 1 33.5 0
  LIP       260 1 1 1 1 1 37.1 0
  HUJ       270 1 1 1 1 1 38.9 0
  SHG       310 1 1 1 1 1 42.9 0
```

Tip: The weight file controls the components of each station that will be used in the inversion. if the number is set as 0, the component will be excluded in the inversion.

Station Azimuth Pz Pr Sz Sr Sh P-arrival time

2. Run the inversion

After generate the weight.dat file, please change your working direcroty to 01-04-2020-earthquake

```
$ cd ../
```

```
#### please copy the following command to a bash script and then run the bash file.

#!/bib/bash

## set the velocity model
model=hn
## set the target event
event=20200104225537
## set the event magnitude
mag=3.5

## loop the inversion for variou focal depth
for h in 08 09 10 11 12 13 14 15 100
do
    cap3D.pl -H0.01 -P180000 -S2/5/0 -T20/50 -D1/1/0.5 -C0.05/0.2/0.03/0.1 -W1 -J0/0.01/0/0.01 -X10 -M$model"_"$h  $event;
done

## obtained the inversion result and plot figure
grep -h Event $event/$model_*.out > junk_$model.out
./depth.pl junk_$model.out $event > junk_$model.ps
```

Note: The key paramters of cap3D.pl are as following:

-H the delta (1/samping rate) of the date

-S max. time shifts in sec for Pnl and surface waves (1/) and tie between SH shift and SV shift:

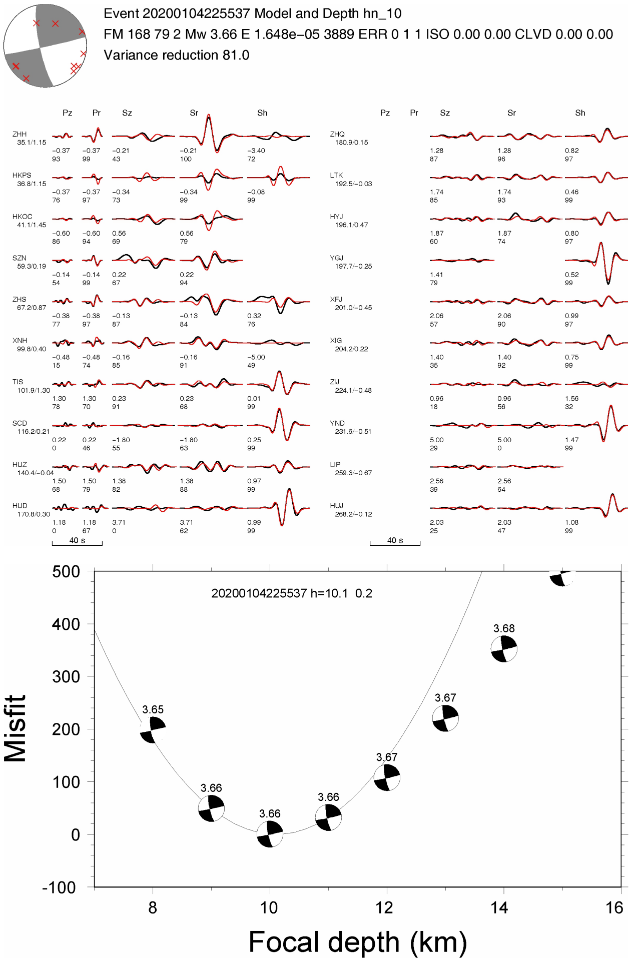
tie=0 shift SV and SH independently, tie=0.5 force the same shift for SH and SV (0.5).

-T max. time window lengths for Pnl and surface waves (35/70).

-C filters for Pnl and surface waves, specified by the corner frequencies of the band-pass filter. (0.02/0.2/0.02/0.1).

-W use displacement for inversion; 1=> data in velocity; 2=> data in disp (0).

The inversion should be conducted multiple times with various parameters to achieve robust inversion result.



9.3.4 3.3 Uncertainty analysis

The uncertainty of the CAP inversion result can be estimated using a bootstrapping inversion method (Tichelaar and Ruff, 1989), in which stations were randomly selected from the station pool (21 stations in total), allowing multiple sampling.

```
#### please copy the following command to a bash script and then run the bash file.

#!/bib/bash

model=hn
event=20200104225537
mag=3.5
```

(continues on next page)

(continued from previous page)

```

##making bootstrapping inversiong weiht.dat file by randomly choose stations from original weight.dat file. Extra weight were added for repeated stations.

ori=./weight1.dat

for i in `seq 1 1000`
do
echo "---------------$i-----------------"
cd 20200104225537
out=weight_test.dat
rm $out
rm temp
for j in `seq 1 21`
do
    shuf -n1 $ori >> temp
done
    cat temp |sort -n -k2 | uniq > temp2
    ## give weight for each station according to the times it was sampled
    for sta in `awk '{print $1}' temp2`
    do
        count=$(grep $sta temp | wc -l)
        grep $sta temp |awk 'NR==1 {print $1,$2,'$count'*$3,'$count'*$4,'$count'*$5,'$count'*$6,'$count'*$7,$8,$9}' >> $out

    done

    echo $out
    cd ..


    for h in 08 09 10 11 12 13 14 15;
    do
        cap3D.pl -H0.01 -P180000  -S2/5/0 -T20/50 -D1/1/0.5 -C0.05/0.2/0.03/0.1 -W1 -X10 -M$model"_"$h -Zweight_test.dat -R0/360/0/90/-90/90 $event;

    done


    grep -h Event $event/"$model"_*.out > junk_"$model"_"$i".out
    ./depth.pl junk_"$model"_"$i".out $event > junk_"$model"_"$i".ps

done
```

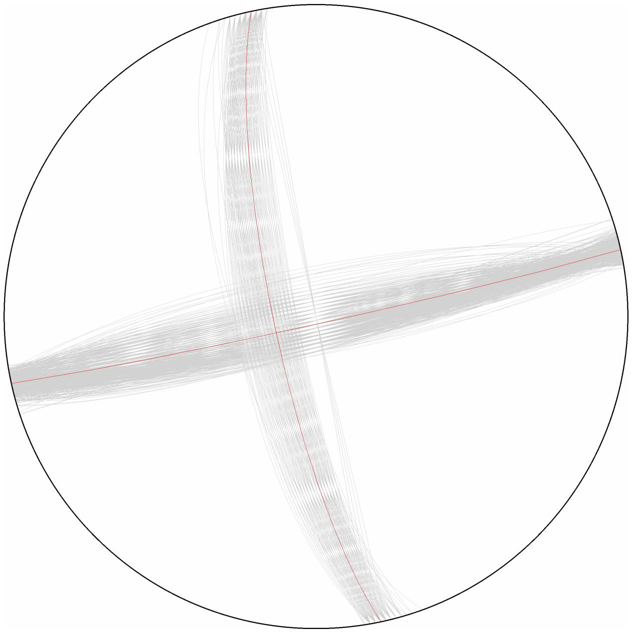
> plot bootstrapping results:

```
#### please copy the following command to a bash script and then run the bash file.

#!/bin/bash

out_ps=focal.ps
file=hn_10.lst
```

(continues on next page)

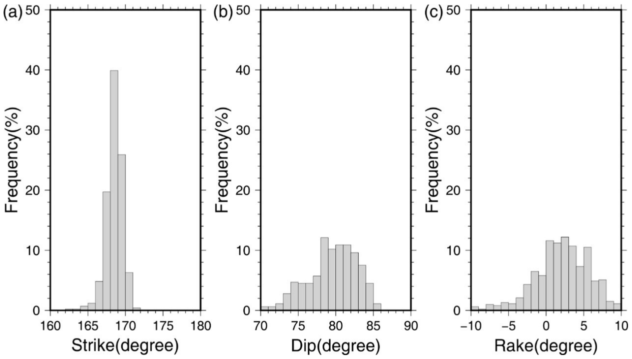
(continued from previous page)

```

    cat *.out > $file
gmt psbasemap -JM18c -R0/18/0/18 -Bxa2f1 -Bya2f1 -K > $out_ps
rm temp ./SDR.txt

## plot the inversion results of each bootstrapping inversion
while read line
do
    echo $line
    strike=$(echo $line |awk '{print $6}')
    dip=$(echo $line |awk '{print $7}')
    if [ $dip -eq 90 ]
    then
        dip=89.99
    else
        dip=$dip
    fi
    rake=$(echo $line |awk '{print $8}')
    echo $strike $dip $rake >> ./SDR.txt
    echo "9 9 10 $strike $dip $rake 5 0 0" > temp
    gmt psmeca temp -J -R -Sa17c -Fa/0.1c/s -Fered -Fgblue -T0/0.01p,gray -L0.01p,black -K -O >> $out_ps

done < $file


## plot the initial inversion result
echo "9 9 10 168 79 2 5 0 0" > temp
gmt psmeca temp -J -R -Sa17c -Fa/0.1c/s -Fered -Fgblue -T0/0.01p,red -L0.01p,black -K -O >> $out_ps
```



> plot the histogram

```
#### please copy the following command to a bash script and then run the bash file.

    #!bin/bash

    file=hn_10.lst
    ps=hist_srtike_dip_rake.ps
    cat $file |awk '{print $6}' |gmt pshistogram -R160/180/0/50 -JX5c/10c -Z1 -Bxa5f1+l"Strike(degree)" -Bya10f2+l"Frequency(%)" -BWSne -L0.1p -Ggray -W1+b -K > $ps
    cat $file |awk '{print $7}' |gmt pshistogram -R70/90/0/50 -JX5c/10c -Z1 -Bxa5f1+l"Dip(degree)" -Bya10f2+l"Frequency(%)" -BWSne -L0.1p -Ggray -W1+b -K -O -X7c>> $ps
    cat $file |awk '{print $8}' |gmt pshistogram -R-10/10/0/50 -JX5c/10c -Z1 -Bxa5f1+l"Rake(degree)" -Bya10f2+l"Frequency(%)" -BWSne -L0.1p -Ggray -W1+b -K -O -X7c>> $ps

    # plot sub-basemap

    gmt set FONT_ANNOT_PRIMARY +16p

    R=0/29.7/0/21
    J=x1/1
    B=a1g1
    #gmt set MAP_FRAME_TYPE=inside MAP_GRID_PEN_PRIMARY=0p,red,.
    #gmt psbasemap -R$R -J$J -B$B -BWSEN -K -O -Xf0c -Yf0c >> $ps

    echo "1.3 12.5 (a)" |gmt pstext -R$R -J$J -Xf0c -Yf0c -K -O >> $ps
    echo "8.3 12.5 (b)" |gmt pstext -R$R -J$J -Xf0c -Yf0c -K -O >> $ps
    echo "15.3 12.5 (c)" |gmt pstext -R$R -J$J -Xf0c -Yf0c -K -O >> $ps

    gmt ps2raster $ps -A -P -Tf

    rm gmt.conf gmt.history
```



> calculate the uncertaities:

```
cat SDR.txt |awk '{print $1}' | awk '{x[NR]=$0; s+=$0; n++} END{a=s/n; for (i in x){ss += (x[i]-a)^2} sd = sqrt(ss/n); print "SD of strike = "sd}'

cat SDR.txt |awk '{print $2}' | awk '{x[NR]=$0; s+=$0; n++} END{a=s/n; for (i in x){ss += (x[i]-a)^2} sd = sqrt(ss/n); print "SD of dip = "sd}'

cat SDR.txt |awk '{print $3}' | awk '{x[NR]=$0; s+=$0; n++} END{a=s/n; for (i in x){ss += (x[i]-a)^2} sd = sqrt(ss/n); print "SD of rake = "sd}'
```

9.4 References

Zhu, L., & Ben-Zion, Y. (2013). Parametrization of general seismic potency and moment tensors for source inversion of seismic waveform data. Geophysical Journal International, 194(2), 839-843.

Yang, H., Zhu, L., & Chu, R. (2009). Fault-plane determination of the 18 April 2008 Mount Carmel, Illinois, earthquake by detecting and relocating aftershocks. Bulletin of the Seismological Society of America, 99(6), 3413-3420.

Chen, H., He, X., Yang, H., & Zhang, J. (2021). Fault-plane determination of the 4 January 2020 offshore pearl river delta earthquake and its implication for seismic hazard assessment. Seismological Society of America, 92(3), 1913-1925.

CHAPTER

TEN

EARTHQUAKE TRANSFORMER TUTORIAL

10.1 Brief introduction

10.1.1 Deep learning model for automatic detection

The neural network was first applied in seismology to pick arrival phases since 2006. The algorithm of using machine learning in seismology is to build a model and train it with large sets of well - labeled data. After training, the model can predict from a given sets of input which means it can detect the P - and S - waves automatically. We can also obtain the probability of the results so to evaluate the performance.

There are numerous architecture of deep learning model. In seismology, Convolutional Neural Network (CNN) is commonly used for classifying the features.

By setting up a robust automatic system, we can enhance the accuracy and efficiency of the earthquake detection.

10.1.2 What is Earthquake Transformer (EQT)

Earthquake Transformer (EQT) is one of the deep learning model for earthquake detection and phase picking.

You can refer the architecture of EQT here.

By using EQT, we can detect the earthquake phases (P & S) automatically. The model has been trained on global seismic data so we can simply apply it in different sites.

10.1.3 How to install EQT

Note:

It’s strongly recommended to install ObsPy via conda.
We here assume you have already installed conda on your computer after finishing the previous GMT tutorial.
If not, please first install it (suggest installation of miniconda).

Open your terminal and run the following commands.

```
$ conda create --name eat python=3.7
$ conda activate eat
$ conda install -c smousavi05 eqtransformer
```

Warning: Exclude $ sign and start without whitespace!

In case if you cannot install EQT using conda, you can try:

```
$ pip install git+https://github.com/smousavi05/EQTransformer
```

10.1.4 Contents of this tutorial

In this tutorial, we will cover:

1. Make the station list and Download the data
2. Detect earthquakes
3. Visualise the result
4. Phase association

Developed by LAU Tsz Lam Zoe.

10.2 1 Make the station list and Download the data

10.2.1 1.1 Constrain the area and the time of interest

It is better for us to constrain a small area of interest, otherwise the downloading time will be extended.

We will use the example from the ObsPy tutorial. But, we will download the continuous data using EQT this time.

```
# Area of interest

# latitude
lat_min = -8
lat_max = -9

# longitude
lon_min = 122.5
lon_max = 124.5

# Time of interest
# You can try with 1 day data first
start_time = "2015-08-11 00:00:00.00"
end_time = "2015-08-12 00:00:00.00"
```

10.2.2 1.2 Make Station list

Station list contains the information of the stations and it will help to find the waveforms by stations during the detection.

```
import os
json_basepath = os.path.join(os.getcwd(),"json/station_list.json")

from EQTransformer.utils.downloader import makeStationList
makeStationList(json_path=json_basepath, client_list=["IRIS"], min_lat=lat_min, max_
↪lat=lat_max, min_lon=lon_min, max_lon=lon_max, start_time=start_time, end_time=end_
↪time, channel_list=["HH[ZNE]", "HH[Z21]", "BH[ZNE]"])
```

`os.path.join` combine one or more path names into a single path.

`makeStationList` make the station list that is based on your interest and applicable for the earthquake transformer.

Here is the example of the station list: `station_list.json`.

It is a dictionary containing the **station name**, **network**, **channels** and **coordinates**.

Note: You can also customise the station list by creating your own dictionary according to the format.

10.2.3 1.3 Download Data

We can download the waveform data directly from the EQT. The file names of the downloaded data must applicable to EQT.

```
from EQTransformer.utils.downloader import downloadMseeds

# We have already constrained the area and time of interest

downloadMseeds(client_list=["IRIS"], stations_json=json_basepath, output_dir="mseed_data
↪", min_lat=lat_min, max_lat=lat_max, min_lon=-lon_min, max_lon=lon_max, start_
↪time=start_time, end_time=end_time, chunk_size=1, channel_list=[], n_processor=2)
```

`downloadMseeds` download the waveform data according to your station list and your interest.

Make sure you have already made the station list before using this function. The data downloaded will be in mseed format and saved into a directory called **mseed_data**.

Note: If you have your own data, then you can skip this procedure.

10.3 2 Detect Earthquake

The idea of detecting earthquake is to input the waveform data and station list, then the model will classify the P - and S - wave together with their corresponding arrival time and save it according to the stations.

There are many alternative ways to undergo the detection. We here introduce one of the method which is using the preprocessed data (hdf5 files).

The hdf5 file can handle large datasets by slicing data and shorten the time of reading large chunk of individual waveforms.

```
from EQTransformer.utils.hdf5_maker import preprocessor

preprocessor(preproc_dir="preproc", mseed_dir='mseed_data', stations_json='json/station_
↪list.json', overlap=0.3, n_processor=2)
```

`preprocessor` prepare the data by slicing it into 1-min long Numpy array in hdf5 file and generate a CSV that contains the list of trace it the hdf5 file.

Note: Make sure you have preprocess the data using `preprocessor` before `predictor`.

After `preprocessor` , there will be a new directory name **mseed_data_processed_hdf5**. It will be used in the next step.

```
from EQTransformer.core.predictor import predictor

predictor(input_dir= 'mseed_data_processed_hdfs', input_model='EqT_model.h5', output_dir=
↪'detections', detection_threshold=0.3, P_threshold=0.1, S_threshold=0.1, number_of_
↪plots=100, plot_mode='time')
```

`predictor` predicts the earthquakes by naming the directories of input, output and the model.

Note: You can set up the detection according to different settings of your area.

The threshold of the detection should be adjusted with the ratio of false detection depending on the data noise level. For example, if the noise level is high in the studied region, then the threshold value has to be high in order to reduce the false detection.

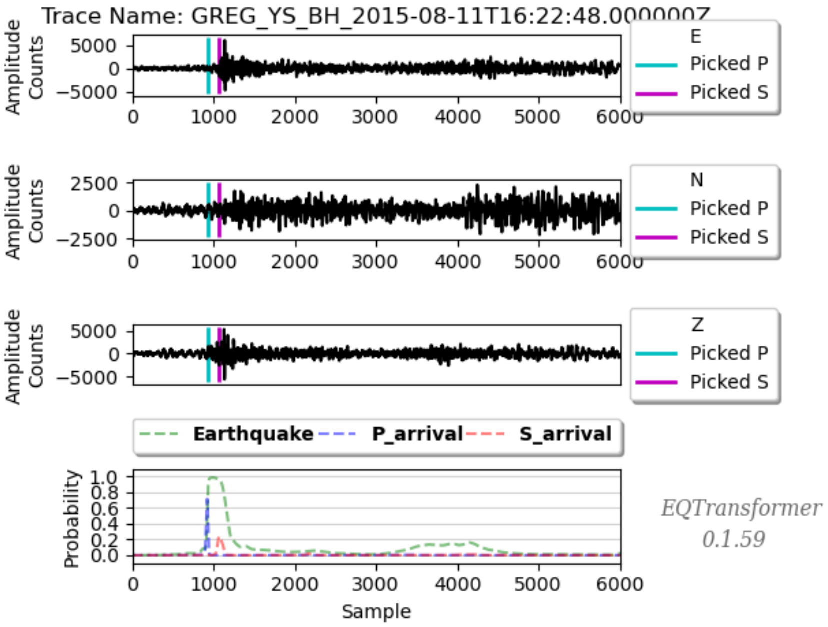
Here is the output files after the detection:

`X_report.txt` the report contains information on input and final results (including total number of events that are detected) `X_report.txt`.

`X_prediction_results.csv` the file contains detection and picking results `X_prediction_results.csv`.

`figures` a directory saves all the plots of the detected events.

Here is one of the output figures of the detected event:



In this figure, we can see the probability and the arrival times of P - and S - waves.

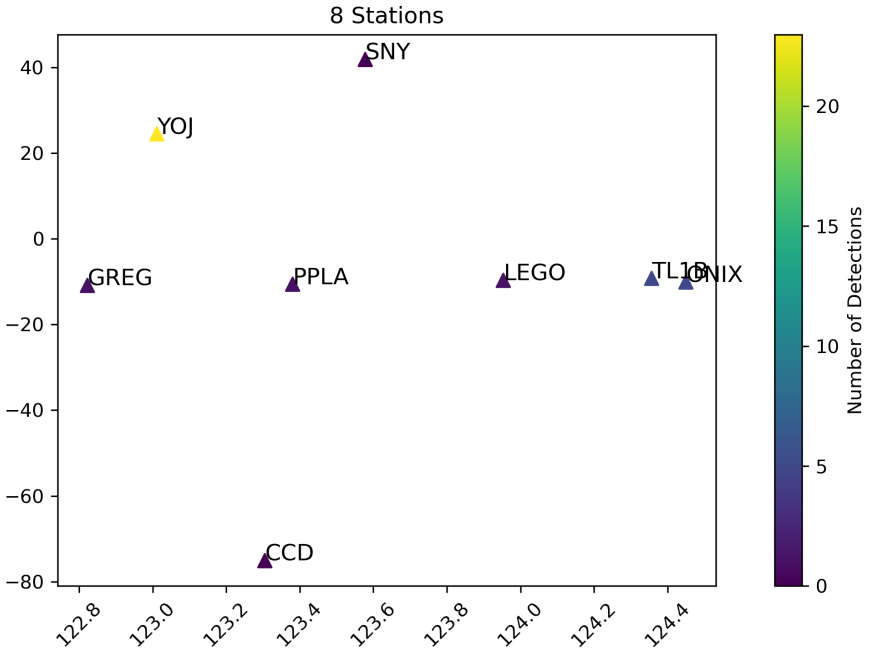
You can refer the same event in the ObsPy tutorial!

10.4 3 Visualise the results

Histogram

```
from EQTransformer.utils.plot import plot_detections

plot_detections(input_dir="detections", input_json="station_list.json", plot_type=
↪'station_map', marker_size=50)
```

plot_detections generate map plot to visualise the number of detections over stations.



Note: The plot here simply show the distribution of the stations and the number of detections accordingly. You can use other method to plot in order the show a detailed map plot (i.e. GMT).

10.5 4 Phase association

Phase association is used to review the detected seismic phases and group them together with the same origin of earthquake.

After the phase picking process, you can undergo a simple association procedure. This can help you to proceed to the next step (relocation) easily.

```
from EQTransformer.utils.associator import run_associator

run_associator(input_dir='detections', start_time="2015-08-11 00:00:00.00", end_time=
↪"2015-08-12 00:00:00.00",  moving_window=15, pair_n=3)
```

`run_associator` perform simple and fast association for further relocation.

Output files:

1. `Y2000.phs` results in HypoInverse format -> HypoInverse `Y2000.phs`

2. `associations.xml` ObsPy QuakeML format -> conventional location algorithms (i.e. NonLinLoc) `associations.xml`

3. `traceName_dic.json` dictionary saves the trace names for source waveforms of the detected events -> later access `traceName_dic.json`

Note: Relocation is needed for accurate location.

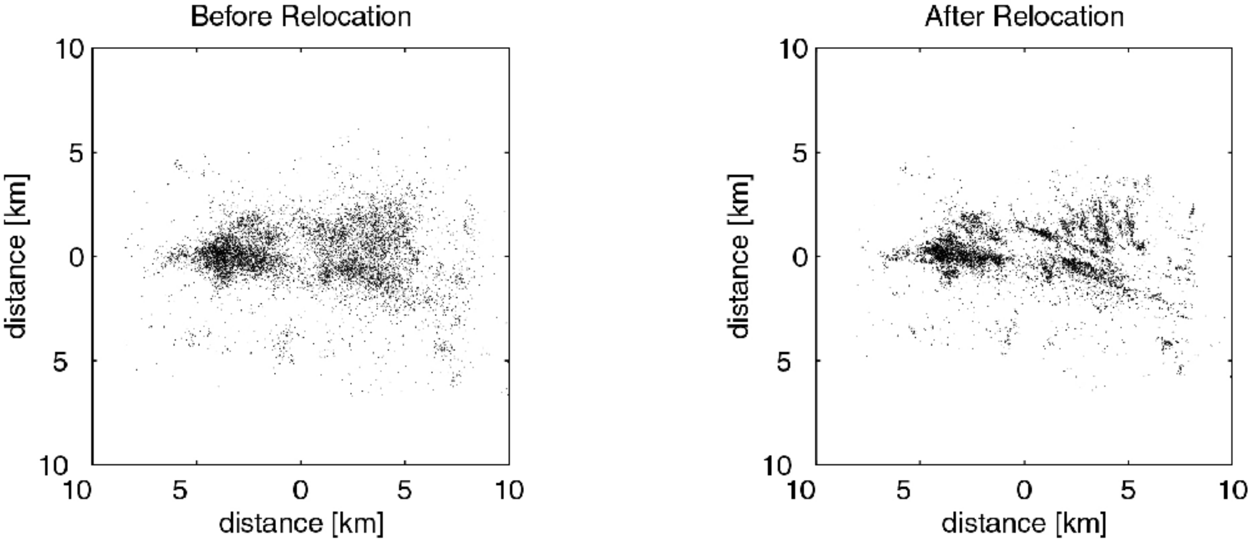
CHAPTER

ELEVEN

DOUBLE-DIFFERENCE EARTHQUAKE RELOCATION

11.1 Introduction

Double-difference earthquake relocation algorithm was developed to improve the location accuracy in the presence of measurement uncertainties when we locate earthquakes in the previous tutorial. It is based on the iterative least-square method, using the time difference between observed and predicted phase arrivals for event pairs recorded by a common station, some uncertainties can be canceled to derive high-accuracy hypocenter locations over large distance. Figure below shows a comparison of ~10,000 earthquake locations (left panel) and double-difference locations (right panel) during the 1997 seismic crisis in the Long Valley caldera (Waldhauser, 2001).



In this tutorial, we'll go through the powerful earthquake double-difference location method. To better convey the key conception of the method, we simplify the model to avoid it runs too complicated to be understood. We'll use:

1. one-layer homogeneous velocity model to get rid of complicated ray-tracing.
2. 2-D X-Z plane rather than 3-D X-Y-Z to decrease complexity
3. P arrivals only

After this tutorial, besides the understanding of key concepts in Double-Difference location, you will also find that model expands from 2-D to 3-D, from one-layer to multi-layers, from P to P&S arrivals, the key processing remains the same.

Note:

You are strongly encouraged to play around codes in this tutorial and introduce to others, you will find your understanding will be enhanced dramatically during the process.

11.1.1 Contents of this tutorial

1. Initiation process
2. Double difference method
3. Iterative double-difference method

Authors: ZI Jinping, SONG Zilin, Earth Science Sytem Program, CUHK.

Testers, XIA Zhuoxuan, Sun Zhangyu, Earth Science Sytem Program, CUHK.

11.2 Initiation

11.2.1 Preparation for Environment

```
import numpy as np
import time
import matplotlib.pyplot as plt
from matplotlib.patches import Polygon
from scipy.sparse.linalg import lsqr
from scipy.sparse import csc_matrix
```

Note:

It is important to define functions below first.

```
def iter_loc(hyc_loop,stas,d,V,niter=10):
    """
    Iterative aboslute earthquake location using least-square method,
    refer to absolute earthquake location python tutorial.
    Parameters:
    | hyc_loop: hypocenter for iteration
    |     stas: array containing station information
    |        d: the observed arrival time
    |        V: the velocity
    |    niter: maximum number of iteration
    """
    k = 0
    while k <= niter:
        dcal = np.zeros((d.shape[0],1))
        for i in range(d.shape[0]):
            dx = stas[i,0]-hyc_loop[0]
            dz = stas[i,1]-hyc_loop[1]
            dcal[i,0] = np.sqrt(dx**2+dz**2)/V+hyc_loop[2]
        delta_d = d - dcal
        e2 = 0
```

(continues on next page)

(continued from previous page)

```
        for i in range(delta_d.shape[0]):
            e2 += delta_d[i,0]**2
        print(f"Iteration {format(k,'2d')} square error: ",format(e2,'13.8f'))

        # >>>>> Build G matrix >>>>>>
        G = np.zeros((d.shape[0],3))
        G[:,2]=1
        for i in range(d.shape[0]):
            for j in range(2):
                denomiter = np.sqrt((hyc_loop[0]-stas[i,0])**2+(hyc_loop[1]-stas[i,
↪1])**2)
                G[i,j]=(hyc_loop[j]-stas[i,j])/denomiter/V

        # >>>>> Invert the m value >>>>
        GTG = np.matmul(G.T,G)
        GTG_inv = np.linalg.inv(GTG)
        GTG_inv_GT = np.matmul(GTG_inv,G.T)
        delta_m = np.matmul(GTG_inv_GT,delta_d)

        # >>>>> Update the hypocenter loop >>>>>
        hyc_loop = np.add(hyc_loop,delta_m.ravel())
        k = k+1

        # >>>>> End the loop if error is small >>>>>
        if e2<0.00000001:
            break

    sigma_d = np.std(delta_d)
    var = sigma_d**2*(d.shape[0])/(d.shape[0]-4)
    sigma_m2 = var * GTG_inv
    return hyc_loop, sigma_m2

def present_loc_results(hyc,sig_square=None,std_fmt='.2f'):
    """
    Print earthquake location results, refer to absolute earthquake location
    for reference
    Parameters:
    |         hyc: hypocenter
    |sig_square: squared convariance
    """
    _x = format(np.round(hyc[0],4),format("6.2f"))
    _z = format(np.round(hyc[1],4),format("6.2f"))
    _t = format(np.round(hyc[2],4),format("6.2f"))
    if not isinstance(sig_square,np.ndarray):
        print("x = ",_x," km")
        print("z = ",_z," km")
        print("t = ",_t," s")
    else:
        stdx = sig_square[0,0]**0.5
        _stdx = format(np.round(stdx,4),std_fmt)
        stdz = sig_square[1,1]**0.5
        _stdz = format(np.round(stdz,4),std_fmt)
```

(continues on next page)

(continued from previous page)

```
        stdt = sig_square[2,2]**0.5
        _stdt = format(np.round(stdt,4),std_fmt)
        print("x = ",_x,"±",_stdx," km")
        print("z = ",_z,"±",_stdz," km")
        print("t = ",_t,"±",_stdt," s")

def matrix_show(*args,**kwargs):
    """
    Show matrix values in grids shape
    Parameters:cmap="cool",gridsize=0.6,fmt='.2f',label_data=True
    """
    ws = []
    H = 0
    str_count = 0
    ndarr_count = 0
    new_args = []
    for arg in args:
        if isinstance(arg,str):
            new_args.append(arg)
            continue
        if isinstance(arg,list):
            arg = np.array(arg)
        if len(arg.shape)>2:
            raise Exception("Only accept 2D array")
        if len(arg.shape) == 1:
            n = arg.shape[0]
            tmp = np.zeros((n,1))
            tmp[:,0] = arg.ravel()
            arg = tmp
        h,w = arg.shape
        if h>H:
            H=h
        ws.append(w)
        new_args.append(arg)
        ndarr_count += 1
    W = np.sum(ws)+len(ws)    # text+matrix+text+...+matrix+text
    if W<0:
        raise Exception("No matrix provided!")

    fmt = '.2f'
    grid_size = 0.6
    cmap = 'cool'
    label_data = True
    for arg in kwargs:
        if arg == "fmt":
            fmt = kwargs[arg]
        if arg == 'grid_size':
            grid_size = kwargs[arg]
        if arg == 'cmap':
            cmap = kwargs[arg]
        if arg == 'label_data':
            label_data = kwargs[arg]
```

(continues on next page)

(continued from previous page)

```
    fig = plt.figure(figsize=(W*grid_size,H*grid_size))
    gs = fig.add_gridspec(nrows=H,ncols=W)

    wloop = 0
    matrix_id = 0
    for arg in new_args:
        if isinstance(arg,str):
            ax = fig.add_subplot(gs[0:H,wloop-1:wloop])
            ax.axis("off")
            ax.set_xlim(0,1)
            ax.set_ylim(0,H)
            ax.text(0.5,H/2,arg,horizontalalignment='center',verticalalignment='center')
        if isinstance(arg,np.ndarray):
            h,w = arg.shape
            hlow = int(np.round((H-h+0.01)/2))        # Find the height grid range
            hhigh = hlow+h
            wlow = wloop
            whigh = wlow+w
#            print("H: ",H,hlow,hhigh,"; W ",W,wlow,whigh)
            ax = fig.add_subplot(gs[hlow:hhigh,wlow:whigh])

            plt.pcolormesh(arg,cmap=cmap)
            for i in range(1,w):
                plt.axvline(i,color='k',linewidth=0.5)
            for j in range(1,h):
                plt.axhline(j,color='k',linewidth=0.5)
            if label_data:
                for i in range(h):
                    for j in range(w):
                        plt.text(j+0.5,i+0.5,format(arg[i,j],fmt),
                                 horizontalalignment='center',
                                 verticalalignment='center')
            plt.xlim(0,w)
            plt.ylim([h,0])
            plt.xticks([])
            plt.yticks([])
            wloop+=w+1
            matrix_id+=1
    plt.show()
```

11.2.2 Basic parameters

Set up station array, earthquake true location, wave-velocity and generate synthetic arrival time.

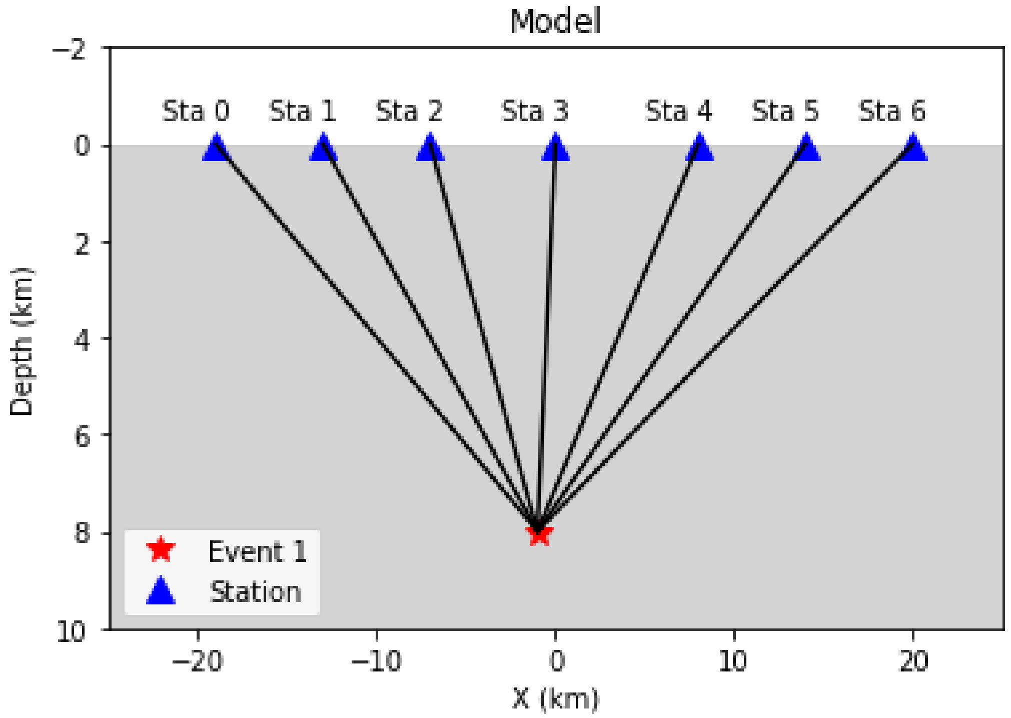
```
stas =np.array([[-20,0],[-14,0],[-8,0],[0,0],[8,0],[14,0],[20,0]]) # Station
stas =np.array([[-19,0],[-13,0],[-7,0],[0,0],[8,0],[14,0],[20,0]]) # Station
hyc1_true = np.array([-1,8,0])
Vtrue = 5
nsta = stas.shape[0]
dobs1 = np.zeros((nsta,1))
for i in range(dobs1.shape[0]):
```

(continues on next page)

(continued from previous page)

```
    dx = stas[i,0]-hyc1_true[0]
    dz = stas[i,1]-hyc1_true[1]
    dobs1[i,0] = np.sqrt(dx**2+dz**2)/Vtrue+hyc1_true[2]
```

```
# Plot event, stations, and rays
fig,ax= plt.subplots(1,1)
plt.plot(hyc1_true[0],hyc1_true[1],'r*',ms=10,label='Event 1')
plt.plot(stas[:,0],stas[:,1],'b^',ms=10,label="Station")
for sta in stas:
    plt.plot([hyc1_true[0],sta[0]],[hyc1_true[1],sta[1]],'k-')

# Add grey background
nodes = [[-25,10],[25,10],[25,0],[-25,0]]
p = Polygon(nodes,facecolor='lightgrey')
for i in range(stas.shape[0]):
    sta = stas[i]
    plt.text(sta[0]-3,sta[1]-0.5,'Sta '+str(i))
plt.gca().add_patch(p)

# Set up plot elements
plt.xlim([-25,25])
plt.ylim([10,-2])
plt.xlabel("X (km)")
plt.ylabel("Depth (km)")
plt.title("Model")
plt.legend();
```



11.3 Absolute earthquake location

11.3.1 Initial location

The station which records the earliest waveform is closest to the hypocenter, so it is reasonable to start iteration:

1. The same x and y with the closest station;
2. Initial depth at 5 km;
3. Initial origin time 1 sec before the earliest arrival;

```
idx = np.argmin(dobs1)       # The index of station
dmin = np.min(dobs1)         # The minimum arrival time

hyc1_init = np.zeros(3);      # Init array
hyc1_init[0] = stas[idx,0];   # Set the same x,y with station
hyc1_init[1] = 5;                 # Set initial depth 5 km
hyc1_init[2] = dmin-1;        # Set initial event time 1s earlier than arrival
print("Initial trial parameters ","x: ",hyc1_init[0],"km; ","z: ",hyc1_init[1],"km; ",
↪"t: ", format(hyc1_init[2],'.4f')+" s")
hyc1_loop = hyc1_init.copy()
```

```
Initial trial parameters  x:  0.0 km;  z:  5.0 km;  t:  0.6125 s
```

We can also define a function to get the initial location

```
def get_init_loc(dobs,stas,depth=5,gap_time=1):
    """
    Get initial earthquake location
    """
    dmin = np.min(dobs)         # The minimum arrival time
    idx = np.argmin(dobs)       # The index of observation

    hyc_init = np.zeros(3);      # Init array
    hyc_init[0] = stas[idx,0];   # Set the same x,y with station
    hyc_init[1] = depth;             # Set initial depth 5 km
    hyc_init[2] = dmin-gap_time;        # Set initial event time 1s earlier than arrival
    print("Initial trial parameters ","x: ",hyc_init[0],"km; ","z: ",hyc_init[1],"km; ",
↪"t: ", format(hyc_init[2],'.4f')+" s")
    return hyc_init
```

```
hyc1_init = get_init_loc(dobs1,stas)
```

```
Initial trial parameters  x:  0.0 km;  z:  5.0 km;  t:  0.6125 s
```

Note:

For the knowledge of iterative location, please refer to the tutorial Earthquake Absolute Location.

```
hyc1_abs, sigma_m2 = iter_loc(hyc1_loop,stas,dobs1,Vtrue)
present_loc_results(hyc1_abs,sigma_m2,std_fmt='.4f')
```

```
Iteration  0 square error:     0.83833287
Iteration  1 square error:     0.01411773
Iteration  2 square error:     0.00000020
Iteration  3 square error:     0.00000000
x =   -1.00 ± 0.0000  km
z =    8.00 ± 0.0000  km
t =    0.00 ± 0.0000  s
```

11.3.2 Velocity Error

In the calculation above, we use the true velocity (**Vtrue**) to conduct the inversion. However, in reality, the velocity we measure is more or less different from the true velocity, thus leading to some bias.

Note:

Try to use other velocity values to conduct the inversion and check the results, what features do you find?

```
Vp = 4.8
hyc1_abs, sigma_m2 = iter_loc(hyc1_init,stas,dobs1,Vp)
present_loc_results(hyc1_abs,sigma_m2,std_fmt='.4f')
print("True location (hyc1_true) ","x: ",hyc1_true[0],"km; ","z: ",hyc1_true[1],"km; ",
↪"t: ", format(hyc1_true[2],'.4f')+" s")
```

Note:

For the knowledge of iteration location, please refer to the absolute earthquake location tutorial

```
Iteration  0 square error:     1.44386729
Iteration  1 square error:     0.03284725
Iteration  2 square error:     0.00078154
Iteration  3 square error:     0.00077835
Iteration  4 square error:     0.00077835
Iteration  5 square error:     0.00077835
Iteration  6 square error:     0.00077835
Iteration  7 square error:     0.00077835
Iteration  8 square error:     0.00077835
Iteration  9 square error:     0.00077835
Iteration 10 square error:     0.00077835
x =   -0.98 ± 0.0398  km
z =    8.90 ± 0.1464  km
t =   -0.24 ± 0.0204  s
True location (hyc1_true)  x:  -1 km;  z:  8 km;  t:  0.0000 s
```

11.3.3 Station Delay

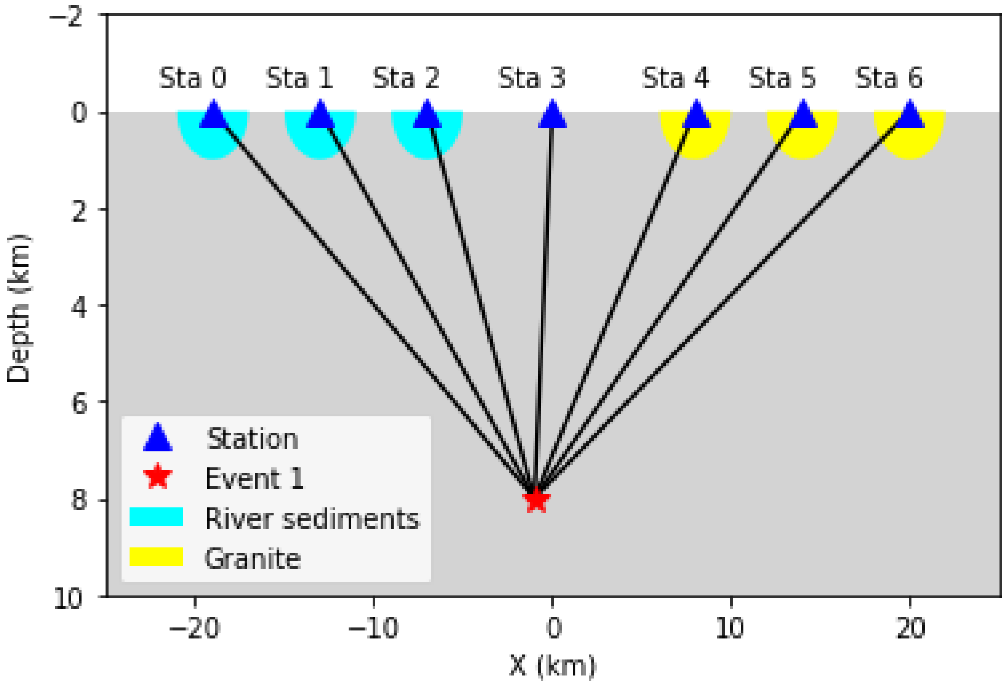
In near surface, the material velocity where stations are located might vary and lead to influence on the travel time, we call it **Station delay**. The **River sediments** are generally composed by not fully consolidated materials, their velocities are therefore low. A lower velocity will lead to a longer travel time, thus the actual arrival time will be later than estimated, here we call it **Positive delay**.

The **Granite** is igneous rock, its density is high with fast velocity. A higher velocity will lead to a shorter travel time, thus the actual arrival time will be earlier than estimated, we call it **Negative delay**.

In this tutorial, we set value of 0.05s for positive delay and -0.05s for negative delay.

```
semix = np.linspace(-1,1,101)
semiy = np.sqrt(1-semix**2)
semixy = np.zeros((101,2))
semixy[:,0] = semix
semixy[:,1] = semiy*0.5
```

```
for sta in stas:
    plt.plot([hyc1_true[0],sta[0]],[hyc1_true[1],sta[1]],'k')
station, = plt.plot(stas[:,0],stas[:,1],'b^',ms=10,label="Station")
event, = plt.plot(hyc1_true[0],hyc1_true[1],'r*',ms=10,label='Event 1')
nodes = [[-25,10],[25,10],[25,0],[-25,0]]
p = Polygon(nodes,facecolor='lightgrey')
plt.gca().add_patch(p)
for sta in stas[:3]:
    p_pos = Polygon(sta+semixy*2,facecolor='cyan')
    plt.gca().add_patch(p_pos)
for sta in stas[4:]:
    p_neg = Polygon(sta+semixy*2,facecolor='yellow')
    plt.gca().add_patch(p_neg)
for i in range(stas.shape[0]):
    sta = stas[i]
    plt.text(sta[0]-3,sta[1]-0.5,'Sta '+str(i))

plt.xlabel("X (km)")
plt.ylabel("Depth (km)")
plt.xlim([-25,25])
plt.ylim([10,-2])
plt.legend([station,event,p_pos,p_neg],["Station","Event 1","River sediments","Granite
↪"]);
```



```
stas_delay = np.zeros((nsta,1))
stas_delay[:,0]= [0.05,0.05,0.05,0,-0.05,-0.05,-0.05]
```

11.3.4 Conduct inversion with delayed data

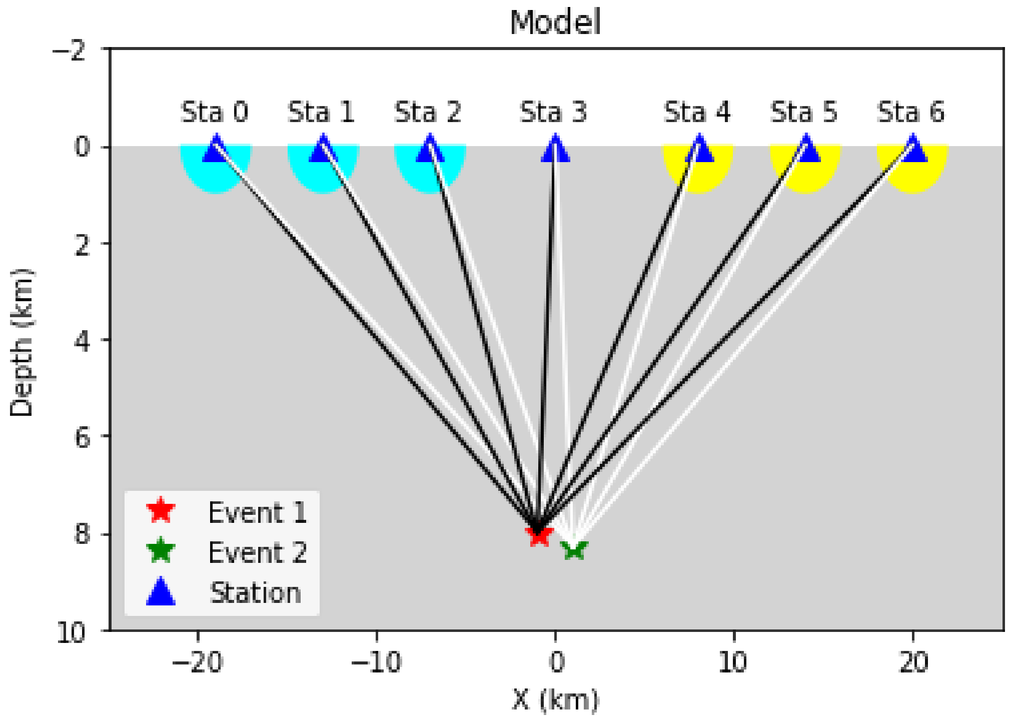
```
dobs1_delay = dobs1 + stas_delay
hyc1_abs_delay, sigma_m2 = iter_loc(hyc1_init,stas,dobs1_delay,Vp)
present_loc_results(hyc1_abs_delay,sigma_m2)
print("True location (hyc1_true) ","x: ",hyc1_true[0],"km; ","z: ",hyc1_true[1],"km; ",
↪"t: ", format(hyc1_true[2],'.4f')+" s")
```

```
Iteration  0 square error:     1.36803100
Iteration  1 square error:     0.03083627
Iteration  2 square error:     0.00074298
Iteration  3 square error:     0.00073813
Iteration  4 square error:     0.00073813
Iteration  5 square error:     0.00073813
Iteration  6 square error:     0.00073813
Iteration  7 square error:     0.00073813
Iteration  8 square error:     0.00073813
Iteration  9 square error:     0.00073813
Iteration 10 square error:     0.00073813
x =   -0.69 ± 0.04  km
z =    8.96 ± 0.14  km
t =   -0.24 ± 0.02  s
True location (hyc1_true)  x:  -1 km;  z:  8 km;  t:  0.0000 s
```

11.3.5 The second event

Now we consider a second event occurred close to the first event

```
hyc2_true = [1,8.3,1]
# Plot event, stations, and rays
fig,ax= plt.subplots(1,1)

# Add grey background
nodes = [[-25,10],[25,10],[25,0],[-25,0]]
p = Polygon(nodes,facecolor='lightgrey')
plt.gca().add_patch(p)

# Plot events
plt.plot(hyc1_true[0],hyc1_true[1],'r*',ms=10,label='Event 1')
plt.plot(hyc2_true[0],hyc2_true[1],'g*',ms=10, label="Event 2")


# Plot stations and rays
plt.plot(stas[:,0],stas[:,1],'b^',ms=10,label="Station")
for i in range(stas.shape[0]):
    sta = stas[i]
    plt.text(sta[0]-2,sta[1]-0.5,'Sta '+str(i))
    plt.plot([hyc1_true[0],sta[0]],[hyc1_true[1],sta[1]],'k-')
    plt.plot([hyc2_true[0],sta[0]],[hyc2_true[1],sta[1]],'w-')
    if i<3:
        p_pos = Polygon(sta+semixy*2,facecolor='cyan')
        plt.gca().add_patch(p_pos)
    if i>3:
        p_neg = Polygon(sta+semixy*2,facecolor='yellow')
        plt.gca().add_patch(p_neg)

# Set up plot elements
plt.xlim([-25,25])
plt.ylim([10,-2])
plt.xlabel("X (km)")
plt.ylabel("Depth (km)")
plt.title("Model")
plt.legend();
```



```
dobs2 = np.zeros((nsta,1))
for i in range(dobs2.shape[0]):
    dx = stas[i,0]-hyc2_true[0]
    dz = stas[i,1]-hyc2_true[1]
    dobs2[i,0] = np.sqrt(dx**2+dz**2)/Vtrue+hyc2_true[2]
```

```
hyc2_init = get_init_loc(dobs2,stas)
```

```
Initial trial parameters  x:  0.0 km;  z:  5.0 km;  t:  1.6720 s
```

```
dobs2_delay = dobs2 + stas_delay
hyc2_abs, sigma_m2 = iter_loc(hyc2_init,stas,dobs2_delay,Vtrue)
present_loc_results(hyc2_abs,sigma_m2)
print("True location (hyc2_true) ","x: ",hyc2_true[0],"km; ","z: ",hyc2_true[1],"km; ",
↪"t: ", format(hyc2_true[2],'.4f')+" s")
```

```
Iteration  0 square error:     1.12489413
Iteration  1 square error:     0.01976384
Iteration  2 square error:     0.00025005
Iteration  3 square error:     0.00024981
Iteration  4 square error:     0.00024981
Iteration  5 square error:     0.00024981
Iteration  6 square error:     0.00024981
Iteration  7 square error:     0.00024981
Iteration  8 square error:     0.00024981
Iteration  9 square error:     0.00024981
Iteration 10 square error:     0.00024981
x =    1.30 ± 0.02   km
```

(continues on next page)

(continued from previous page)

```
z =    8.23 ± 0.08  km
t =    1.01 ± 0.01  s
True location (hyc2_true)  x:  1 km;  z:  8.3 km;  t:  1.0000 s
```

11.3.6 Add Picking Noise

Note:

please refer to Earthquake Absolute Location tutorial for more information of add picking noise

Add random noise to simulate the phase picking uncertainty

```
mu = 0
sigma = 0.1
np.random.seed(252)
errors = np.random.normal(mu,sigma,size=(nsta,1))
dobs1_delay_noise = dobs1_delay+errors
np.random.seed(101)
errors = np.random.normal(mu,sigma,size=(nsta,1))
dobs2_delay_noise = dobs2_delay+errors
```

11.4 Double Difference Method

The travel-time residual of event i at station k:

$r_k^i = (T_k^i)^{obs} - (T_k^i)^{cal}$ comes from:

1. Earthquake location mistfit;
2. Earthquake origin time misfit;
3. Along ray-path velocity variation;
4. Station delay.

could be presented via below equation:

$$r_k^i = \sum_{l=1}^{2} \frac{\partial T_k^i}{\partial x_l^i} \Delta x_l^i + \Delta \tau^i + \int_{s_i}^{r_k} \Delta u ds + S_k$$

T: travel time

τ: event origin time

s, r: source and receiver location

$u = \frac{1}{V}$: slowness

S_k: station delay

Event :math:`j`, station :math:`k`

The travel-time residual of event j at station k:

$$r_k^j = (T_k^j)^{obs} - (T_k^j)^{cal} = \sum_{l=1}^{2} \frac{\partial T_k^j}{\partial x_l^j} \Delta x_l^j + \Delta \tau^j + \int_{s_j}^{r_k} \Delta u ds + S_k$$

11.4.1 Make difference

$$r_k^i - r_k^j = \sum_{l=1}^{2} \frac{\partial T_k^i}{\partial x_l^i} \Delta x_l^i + \Delta \tau^i + \int_{s_i}^{r_k} \Delta u ds - \sum_{l=1}^{2} \frac{\partial T_k^j}{\partial x_l^j} \Delta x_l^j - \Delta \tau^j - \int_{s_j}^{r_k} \Delta u ds$$

Noted that station delay S is removed.

$$r_k^i - r_k^j = \{(T_k^i)^{obs} - (T_k^i)^{cal}\} - \{(T_k^j)^{obs} - (T_k^j)^{cal}\}$$

Reorganizing the equation leads to

$$r_k^i - r_k^j = (T_k^i - T_k^j)^{obs} - (T_k^i - T_k^j)^{cal}$$

This is the so-called **double-difference**.

If **two events are close** to each other, then they have similar ray paths, that is:

$$\int_{s_i}^{r_k} \Delta u ds \approx \int_{s_j}^{r_k} \Delta u ds$$

The velocity anomaly along the ray path is the same for two events. Then we get

$$r_k^i - r_k^j = \sum_{l=1}^{2} \frac{\partial T_k^i}{\partial x_l^i} \Delta x_l^i + \Delta \tau^i - \sum_{l=1}^{2} \frac{\partial T_k^j}{\partial x_l^j} \Delta x_l^j - \Delta \tau^j$$

The travel time residual $r_k^i = (T_k^i)^{obs} - (T_k^i)^{cal}$, the travel time residual $r_k^j = (T_k^j)^{obs} - (T_k^j)^{cal}$, their difference is related to:

1. Earthquake location misfit
2. Origin time misfit

and the error sources: 1. Station delay 2. Velocity variation along ray-path are remove or mitigated by double-difference

An inversion equation could be set up:

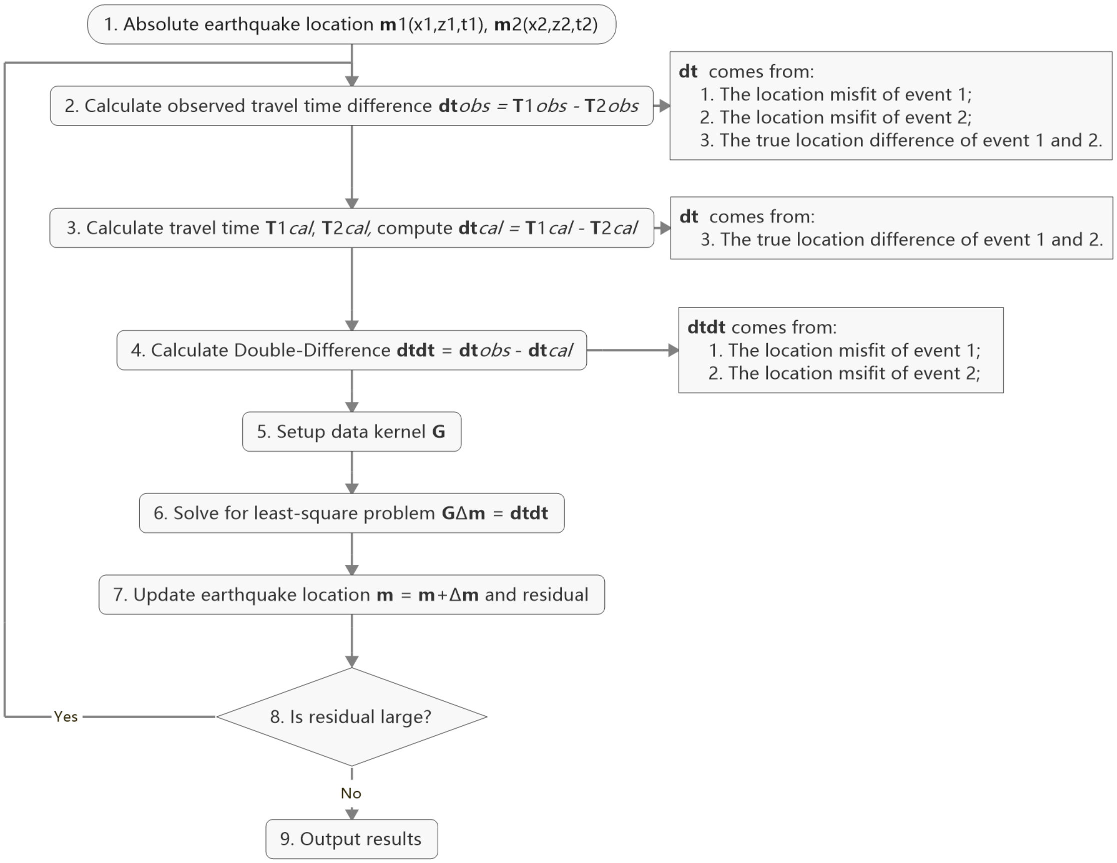
$$G \Delta m = \Delta d$$

Detailed expression is, note the negative signs in the last 3 columns of data kernel $\mathbf{G}$:

$$\begin{bmatrix} \frac{\partial T_1^1}{\partial x} & \frac{\partial T_1^1}{\partial z} & 1 & -\frac{\partial T_1^2}{\partial x} & -\frac{\partial T_1^2}{\partial z} & -1 \\ \frac{\partial T_2^1}{\partial x} & \frac{\partial T_2^1}{\partial z} & 1 & -\frac{\partial T_2^2}{\partial x} & -\frac{\partial T_2^2}{\partial z} & -1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial T_k^1}{\partial x} & \frac{\partial T_k^1}{\partial z} & 1 & -\frac{\partial T_k^2}{\partial x} & -\frac{\partial T_k^2}{\partial z} & -1 \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta z_1 \\ \Delta t_1 \\ \Delta x_2 \\ \Delta z_2 \\ \Delta t_2 \end{bmatrix} = \begin{bmatrix} r_1^1 - r_1^2 \\ r_2^1 - r_2^2 \\ \vdots \\ r_k^1 - r_k^2 \end{bmatrix}$$

Practical usage will be introduced later.

Workflow

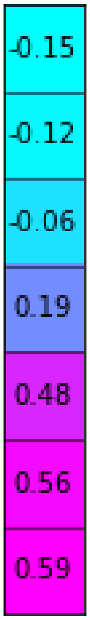

1. Absolute earthquake location m1(x1,z1,t1), m2(x2,z2,t2)
2. Calculate observed travel time difference dtobs = T1obs - T2obs
dt comes from:
1. The location misfit of event 1;
2. The location msifit of event 2;
3. The true location difference of event 1 and 2.
3. Calculate travel time T1cal, T2cal, compute dtcal = T1cal - T2cal
dt comes from:
3. The true location difference of event 1 and 2.
4. Calculate Double-Difference dtdt = dtobs - dtcal
dtdt comes from:
1. The location misfit of event 1;
2. The location msifit of event 2;
5. Setup data kernel G
6. Solve for least-square problem GΔm = dtdt
7. Update earthquake location m = m+Δm and residual
8. Is residual large?
Yes
No
9. Output results

```
hyc1_dd = hyc1_abs.copy()
hyc2_dd = hyc2_abs.copy()
```

11.5 1. Observed Travel Time Difference

```
obs_trav_t1 = dobs1_delay - hyc1_dd[2] # Travel time = arrival_time - origin_time
obs_trav_t2 = dobs2_delay - hyc2_dd[2]
obs_dt = obs_trav_t1 - obs_trav_t2
```

```
matrix_show(obs_dt)
```



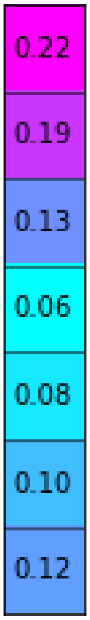
-0.15
-0.12
-0.06
0.19
0.48
0.56
0.59

11.6 2. Calculated Travel Time Difference

```
dcal1 = np.zeros((nsta,1))
for i in range(dobs1.shape[0]):
    dx = stas[i,0]-hyc1_dd[0]
    dz = stas[i,1]-hyc1_dd[1]
    dcal1[i,0] = np.sqrt(dx**2+dz**2)/Vtrue+hyc1_dd[2]
dcal2 = np.zeros((nsta,1))
for i in range(dobs1.shape[0]):
    dx = stas[i,0]-hyc2_dd[0]
    dz = stas[i,1]-hyc2_dd[1]
    dcal2[i,0] = np.sqrt(dx**2+dz**2)/Vtrue+hyc2_dd[2]
cal_trav_t1 = dcal1 - hyc1_dd[2] # Travel time = calculated_time - origin_time
cal_trav_t2 = dcal2 - hyc2_dd[2]
cal_dt = cal_trav_t1 - cal_trav_t2
```

11.7 3. Calculate Double-Difference

```
dtdt = obs_dt - cal_dt
matrix_show(dtdt)
```

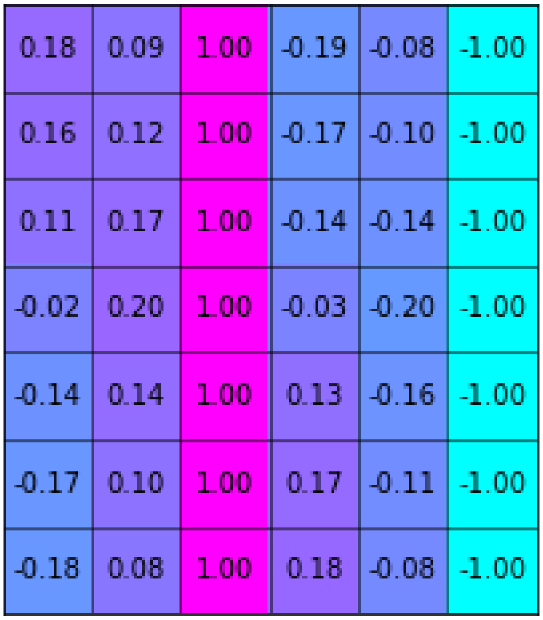


0.22
0.19
0.13
0.06
0.08
0.10
0.12

11.8 4. Build Up Data Kernel - G

```
ncol = 3 * 2            # Two event, each has 3 parameter (delta x, delta z, delta t)
G = np.zeros((nsta,ncol))
G[:,2]=1; G[:,5] = -1   # Partial derivative of origin column is 1
for i in range(nsta):
    for j in range(2):
        denomiter1 = np.sqrt((hyc1_dd[0]-stas[i,0])**2+(hyc1_dd[1]-stas[i,1])**2)
        G[i,j]=(hyc1_dd[j]-stas[i,j])/denomiter1/Vtrue
        denomiter2 = np.sqrt((hyc2_dd[0]-stas[i,0])**2+(hyc2_dd[1]-stas[i,1])**2)
        G[i,j+3]=-(hyc2_dd[j]-stas[i,j])/denomiter2/Vtrue
```

```
matrix_show(G)
```



0.18	0.09	1.00	-0.19	-0.08	-1.00
0.16	0.12	1.00	-0.17	-0.10	-1.00
0.11	0.17	1.00	-0.14	-0.14	-1.00
-0.02	0.20	1.00	-0.03	-0.20	-1.00
-0.14	0.14	1.00	0.13	-0.16	-1.00
-0.17	0.10	1.00	0.17	-0.11	-1.00
-0.18	0.08	1.00	0.18	-0.08	-1.00

11.9 5. Check GTG Inverse Exists

$$G\Delta m = \Delta d$$

G is not a square matrix, $G^T G$ is a squared matrix, we then have:

$$G^T G\Delta m = G^T \Delta d$$

If the inverse of $G^T G$ exists (the determinnant != 0, in here we have 10 observations to solve for 4 parameters), then:

$$\Delta m = (G^T G)^{-1} G^T \Delta d$$

```
GTG = np.matmul(G.T,G)
det = np.linalg.det(GTG)  # Calculate matrix determinant
if det == 0:
    print("Error! The determinant is ZERO!!!")
```

```
Error! The determinant is ZERO!!!
```

11.10 6. Add Damp to Matrix

Determinant equals zero means there is no unique solution to the inverse problem, that is, the constraints in data kernel G are not enough to get a result, more constraints is needed. The common method is to add damp to the data kernel.

### Damping the kernel Before damping:

$$[G]\,[m] = [d]$$

After damping:

$$\begin{bmatrix} G \\ \lambda I \end{bmatrix} \begin{bmatrix} m \end{bmatrix} = \begin{bmatrix} d \\ O \end{bmatrix}$$

I is an identity matrix, in this case, it should have columns with G, so its dimension is 6×6, here:

$$\lambda I = \begin{bmatrix} \lambda & 0 & 0 & 0 & 0 & 0 \\ 0 & \lambda & 0 & 0 & 0 & 0 \\ 0 & 0 & \lambda & 0 & 0 & 0 \\ 0 & 0 & 0 & \lambda & 0 & 0 \\ 0 & 0 & 0 & 0 & \lambda & 0 \\ 0 & 0 & 0 & 0 & 0 & \lambda \end{bmatrix}$$

### Mathematical Meaning Write new constraints in equation, that is:

$$\lambda \Delta x_1 = 0 \tag{11.1}$$

$$\lambda \Delta z_1 = 0 \tag{11.2}$$

$$\lambda \Delta t_1 = 0 \tag{11.3}$$

$$\lambda \Delta x_2 = 0 \tag{11.4}$$

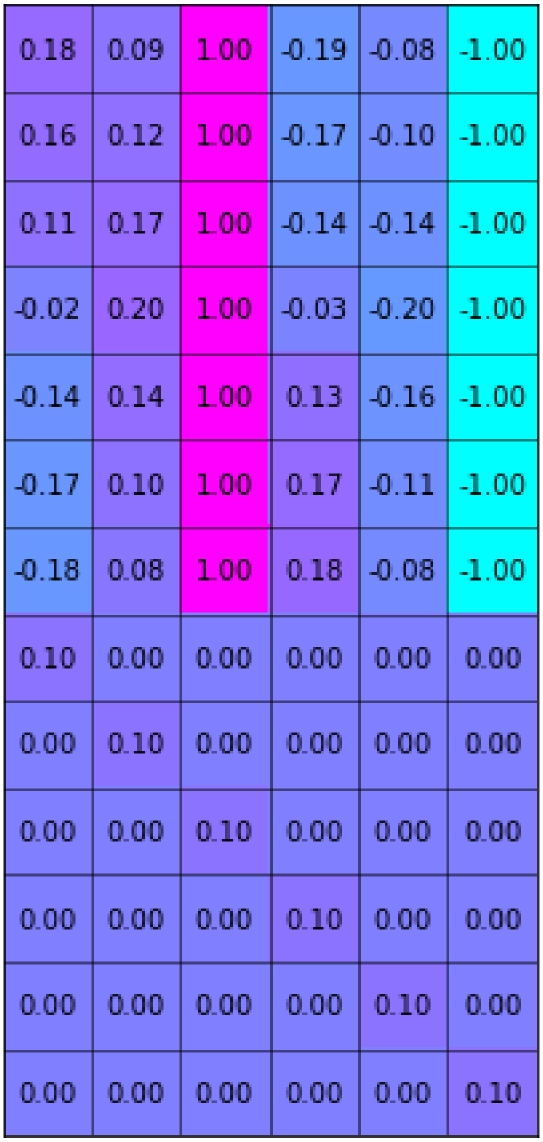
$$\lambda \Delta z_2 = 0 \tag{11.5}$$

$$\lambda \Delta t_2 = 0 \tag{11.6}$$

What does this mean? It means that the solution **SHOULD** be zero. As a least square problem solution is a tradeoff among equations. The application of damping factor will lead to the solution be small values. λ controls the weight(importance) of damping. A large damp will lead to the solution more close to zero.

```
G_dp = np.zeros((nsta+ncol,ncol))
G_dp[:nsta,:] = G
damp = 0.1
G_dp[nsta:,:] = np.diag([1,1,1,1,1,1])*damp
dtdt_damp = np.zeros((nsta+ncol,1))
dtdt_damp[:nsta,0] = dtdt.ravel()
```

```
matrix_show(G_dp)
```



0.18	0.09	1.00	-0.19	-0.08	-1.00
0.16	0.12	1.00	-0.17	-0.10	-1.00
0.11	0.17	1.00	-0.14	-0.14	-1.00
-0.02	0.20	1.00	-0.03	-0.20	-1.00
-0.14	0.14	1.00	0.13	-0.16	-1.00
-0.17	0.10	1.00	0.17	-0.11	-1.00
-0.18	0.08	1.00	0.18	-0.08	-1.00
0.10	0.00	0.00	0.00	0.00	0.00
0.00	0.10	0.00	0.00	0.00	0.00
0.00	0.00	0.10	0.00	0.00	0.00
0.00	0.00	0.00	0.10	0.00	0.00
0.00	0.00	0.00	0.00	0.10	0.00
0.00	0.00	0.00	0.00	0.00	0.10

11.11 7. Solve Damped Problem

Step 1:

$$\begin{bmatrix} G \\ \lambda I \end{bmatrix} [m] = \begin{bmatrix} d \\ O \end{bmatrix}$$

Step 2:

$$[G^T \lambda I] \begin{bmatrix} G \\ \lambda I \end{bmatrix} [m] = [G^T \lambda I] \begin{bmatrix} d \\ O \end{bmatrix}$$

Step 3:

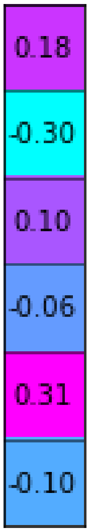
$$[G^T G + \lambda^2 I] \; [m] = [G^T d]$$

Step 4:

$$m = (G^T G + \lambda^2 I)^{-1} G^T d$$

```
G_dpTG_dp = np.matmul(G_dp.T,G_dp)
G_dpTG_dp_inv = np.linalg.inv(G_dpTG_dp)
G_dpTG_dp_inv_G_dpT = np.matmul(G_dpTG_dp_inv,G_dp.T)
m = np.matmul(G_dpTG_dp_inv_G_dpT,dtdt_damp)
```

```
matrix_show(m)
```



11.12 8. Update Location

The output results are the earthquake location misfit with reference to its absolute location. Therefore, the absolute earthquake location should be updated.

$$x_1 = x_1 + \Delta x_1$$

$$z_1 = z_1 + \Delta z_1$$

$$t_1 = t_1 + \Delta t_1$$

$$x_2 = x_2 + \Delta x_2$$

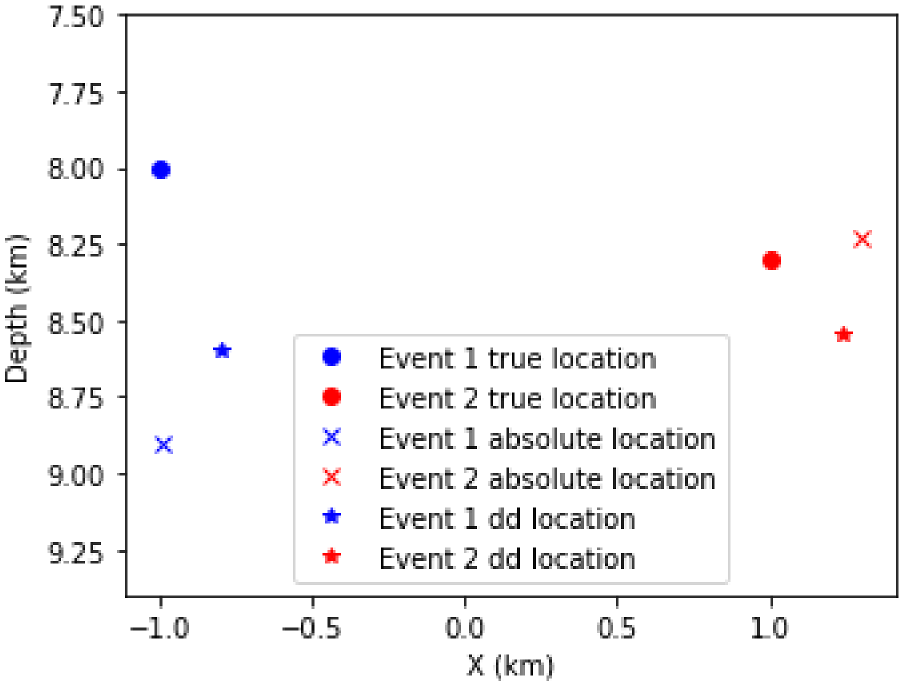
$$z_2 = z_2 + \Delta z_2$$

$$t_2 = t_2 + \Delta t_2$$

```
hyc1_dd = hyc1_dd+m.ravel()[:3]
hyc2_dd = hyc2_dd+m.ravel()[3:]
```

We can see the solutions already depart the initial location and move closer to the true location

```
xmin = min(hyc1_true[0],hyc1_abs[0],hyc1_dd[0])
xmax = max(hyc1_true[0],hyc1_abs[0],hyc1_dd[0])
ymin = min(hyc1_true[1],hyc1_abs[1],hyc1_dd[1])
ymax = max(hyc1_true[1],hyc1_abs[1],hyc1_dd[1])
plt.plot(hyc1_true[0],hyc1_true[1],"bo",label="Event 1 true location")
plt.plot(hyc2_true[0],hyc2_true[1],"ro",label="Event 2 true location")
plt.plot(hyc1_abs[0],hyc1_abs[1],'bx',label="Event 1 absolute location")
plt.plot(hyc2_abs[0],hyc2_abs[1],'rx',label="Event 2 absolute location")
plt.plot(hyc1_dd[0],hyc1_dd[1],'b*',label="Event 1 dd location")
plt.plot(hyc2_dd[0],hyc2_dd[1],'r*',label="Event 2 dd location")
plt.gca().set_aspect('equal')
plt.legend()
plt.ylim(ymax+0.5,ymin-0.5)
plt.ylabel("Depth (km)")
plt.xlabel("X (km)");
```



11.13 9. Error analysis

The error in observed data will of course lead to uncertainties in the estimation of earthquake location parameters. Their relationship could be described as:

$$\sigma_m^2 = \sigma^2(G^TG + \lambda^2 I)^{-1}$$

(Wanna know how this relationship derived? Page 435 of **An Introduction to Seismology, Earthquakes, and Earth Structure**)

```
mean_dtdt_damp = np.mean(dtdt_damp)
e2 = 0
```

(continues on next page)

(continued from previous page)

```
for i in range(dtdt.shape[0]):
    e2 += (dtdt_damp[i,0] - mean_dtdt_damp)**2
print(f"Square error: ",format(e2,'13.8f'))
var = e2/(dtdt_damp.shape[0]-6)
sigma_m2 = G_dpTG_dp_inv*var
```

```
Square error:      0.04330659
```

```
present_loc_results(hyc1_dd,sigma_m2[:3,:3])
present_loc_results(hyc2_dd,sigma_m2[3:,3:])
```

```
x =   -0.80 ± 0.56  km
z =    8.60 ± 0.64  km
t =   -0.14 ± 0.56  s
x =    1.24 ± 0.55  km
z =    8.54 ± 0.63  km
t =    0.90 ± 0.56  s
```

Exercise (5 min)

Try to modify the **damp** parameter and update the results, how it changes? What is the relationship between **damping factor**, **m**, and **Uncertainty**? Can you explain why?

11.14 10. Condition Number

We have realized that the damping factor controls the converge rate, a larger **damping factor** will lead to slow converge rate but small uncertainty; a smaller **damping factor** will lead to fast converge rate but large uncertainty. Then how to choose proper damping factor?

A good indicator is the conditon number. Conditon number quantifies the relationship between solution error and data error. In earthquake double difference location, the condition number should be in the range 40-100 (empirical).

```
cond = np.linalg.cond(G_dp)
print("Condtion number is: ",format(cond,'.2f'))
```

```
Condtion number is:  37.72
```

Exercise: Start Another Iteration

The error is still high, update the earthquake location and rerun the process to check the location variation.

11.15 Iterative Double-Difference Method

```
hyc1_loop = hyc1_abs
hyc2_loop = hyc2_abs
niter = 100
k = 0
event_number = 2
event_parameters = 3 #(x,y,z)
```

(continues on next page)

(continued from previous page)

```
#----------Iteration starts----------------------
while k <=niter:
    #----1. Update observed travel time difference-----------------
    obs_trav_t1 = dobs1_delay - hyc1_dd[2]              # Travel time = arrival_time -␣
↪origin_time
    obs_trav_t2 = dobs2_delay - hyc2_dd[2]
    obs_dt = obs_trav_t1 - obs_trav_t2
    #----2. Update calculated travel time difference------------------
    dcal1 = np.zeros((dobs1.shape[0],1))
    for i in range(dobs1.shape[0]):
        dx = stas[i,0]-hyc1_loop[0]
        dz = stas[i,1]-hyc1_loop[1]
        dcal1[i,0] = np.sqrt(dx**2+dz**2)/Vtrue+hyc1_loop[2]
    dcal2 = np.zeros((dobs2.shape[0],1))
    for i in range(dobs1.shape[0]):
        dx = stas[i,0]-hyc2_loop[0]
        dz = stas[i,1]-hyc2_loop[1]
        dcal2[i,0] = np.sqrt(dx**2+dz**2)/Vtrue+hyc2_loop[2]
    cal_trav_t1 = dcal1 - hyc1_dd[2]
    cal_trav_t2 = dcal2 - hyc2_dd[2]
    cal_dt = cal_trav_t1 - cal_trav_t2
    #----3. Calculate double difference-------------------------------
    dtdt = obs_dt - cal_dt
    #----4. Set up G kernel-------------------------------------------
    ncol = event_number * event_parameters
    G = np.zeros((nsta,ncol))
    G[:,2]=1; G[:,5] = -1   # Partial derivative of origin column is 1
    for i in range(nsta):
        for j in range(2):
            denomiter1 = np.sqrt((hyc1_loop[0]-stas[i,0])**2+(hyc1_loop[1]-stas[i,1])**2)
            G[i,j]=(hyc1_loop[j]-stas[i,j])/denomiter1/Vtrue
            denomiter2 = np.sqrt((hyc2_loop[0]-stas[i,0])**2+(hyc2_loop[1]-stas[i,1])**2)
            G[i,j+3]=-(hyc2_loop[j]-stas[i,j])/denomiter2/Vtrue
    #----5. Add damp--------------------------------------------------
    G_dp = np.zeros((nsta+ncol,ncol))
    G_dp[:nsta,:] = G
    damp = 0.1
    G_dp[nsta:,:] = np.diag([1,1,1,1,1,1])*damp
    dtdt_damp = np.zeros((nsta+ncol,1))
    dtdt_damp[:nsta,0] = dtdt.ravel()
    #----6. Solve for Solution-----------------------------------------
    G_dpTG_dp = np.matmul(G_dp.T,G_dp)
    G_dpTG_dp_inv = np.linalg.inv(G_dpTG_dp)
    G_dpTG_dp_inv_G_dpT = np.matmul(G_dpTG_dp_inv,G_dp.T)
    m = np.matmul(G_dpTG_dp_inv_G_dpT,dtdt_damp)
    #----7. Update location-------------------------------------------
    hyc1_loop = hyc1_loop+m.ravel()[:3]
    hyc2_loop = hyc2_loop+m.ravel()[3:]
    #----8. Error Calculation------------------------------------------
    mean_dtdt_damp = np.mean(dtdt_damp)
    e2 = 0
    for i in range(dtdt.shape[0]):
```

(continues on next page)

(continued from previous page)

```
        e2 += (dtdt_damp[i,0] - mean_dtdt_damp)**2
    print(f"Iteration {format(k,'4d')} square error: ",format(e2,'13.8f'))
    if e2<0.0000000001:
        print("Itertion stopped for too small error!")
        break
    k = k+1
#--------9. Variance analysis-------------------------------------------
var = e2/(dtdt_damp.shape[0]-event_number * event_parameters)
sigma_m2 = G_dpTG_dp_inv*var
hyc1_dd = hyc1_loop
hyc2_dd = hyc2_loop
```

```
Iteration    0 square error:     0.04330659
Iteration    1 square error:     0.00096939
Iteration    2 square error:     0.00013074
Iteration    3 square error:     0.00005316
Iteration    4 square error:     0.00004521
Iteration    5 square error:     0.00004369
Iteration    6 square error:     0.00004278
Iteration    7 square error:     0.00004195
Iteration    8 square error:     0.00004113
Iteration    9 square error:     0.00004032
Iteration   10 square error:     0.00003954
......
Iteration   90 square error:     0.00000830
Iteration   91 square error:     0.00000814
Iteration   92 square error:     0.00000799
Iteration   93 square error:     0.00000784
Iteration   94 square error:     0.00000769
Iteration   95 square error:     0.00000754
Iteration   96 square error:     0.00000740
Iteration   97 square error:     0.00000726
Iteration   98 square error:     0.00000713
Iteration   99 square error:     0.00000699
Iteration  100 square error:     0.00000686
```

```
present_loc_results(hyc1_dd,sigma_m2[:3,:3],std_fmt='.5f')
present_loc_results(hyc2_dd,sigma_m2[3:,3:],std_fmt='.5f')
```

```
x =   -0.87 ± 0.00700  km
z =    8.16 ± 0.00790  km
t =   -0.12 ± 0.00700  s
x =    1.14 ± 0.00700  km
z =    8.41 ± 0.00800  km
t =    0.88 ± 0.00700  s
```

11.15.1 LSQR Algorithm

Considering a double difference cluster with 1000 events, we estimate the time consumed for one iteration. Note the G^TG dimension is 4000×4000, it costs 16 seconds to calculate the inverse and singular value decomposition. What about 10 k events?

```
G = np.random.randn(4000,4000)
```

```
tmp1 = time.time()
G_inv = np.linalg.inv(G)
u,s,vt = np.linalg.svd(G_inv)
tmp2 = time.time()
print(tmp2-tmp1,' s')
if (tmp2-tmp1)>5:
    print("Wow, it cost a lot of time of do the calculation")
```

```
38.39115285873413  s
Wow, it cost a lot of time of do the calculation
```

Introduction to LSQR

Least-Square QR decompositon (LSQR, Paige, C.C and Saunders, M.A. (1982)) method is developed for least-square solution for large dataset, its performance in ill-conditioned problems is superior.

From problem $\mathbf{Am} = \mathbf{b}$, $\mathbf{A}$ maps the solution to the data space. $\mathbf{A^T}$ maps the data to the solution space. LSQR method eliminates residual iteratively with limited computation.

To ensure the stability of method, each A column is required to be scaled up to be unit value. That is:

$$\begin{aligned}\mathbf{Am} &= \begin{bmatrix} A_1 & A_2 & \cdots & A_k \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ \vdots \\ m_k \end{bmatrix} \\ &= A_1 m_1 + A_2 m_2 + \cdots + A_k m_k \\ &= \frac{A_1}{\|A_1\|}(\|A_1\| m_1) + \frac{A_2}{\|A_2\|}(\|A_2\| m_2) + \cdots + \frac{A_k}{\|A_k\|}(\|A_k\| m_k) \\ &= \mathbf{A'm'} = \mathbf{b}\end{aligned}$$

After get the solution, a conversion between $\mathbf{m'}$ and $\mathbf{m}$ is needed by $m_i = \frac{m'_i}{\|A_i\|}$

```
hyc1_loop = hyc1_abs
hyc2_loop = hyc2_abs
niter = 100
k = 0
event_number = 2
event_parameters = 3 #(x,y,z)
#----------Iteration starts----------------------
while k <=niter:
    #----1. Update observed travel time difference------------------
    obs_trav_t1 = dobs1_delay - hyc1_dd[2]             # Travel time = arrival_time -␣
↪origin_time
    obs_trav_t2 = dobs2_delay - hyc2_dd[2]
    obs_dt = obs_trav_t1 - obs_trav_t2
```

(continues on next page)

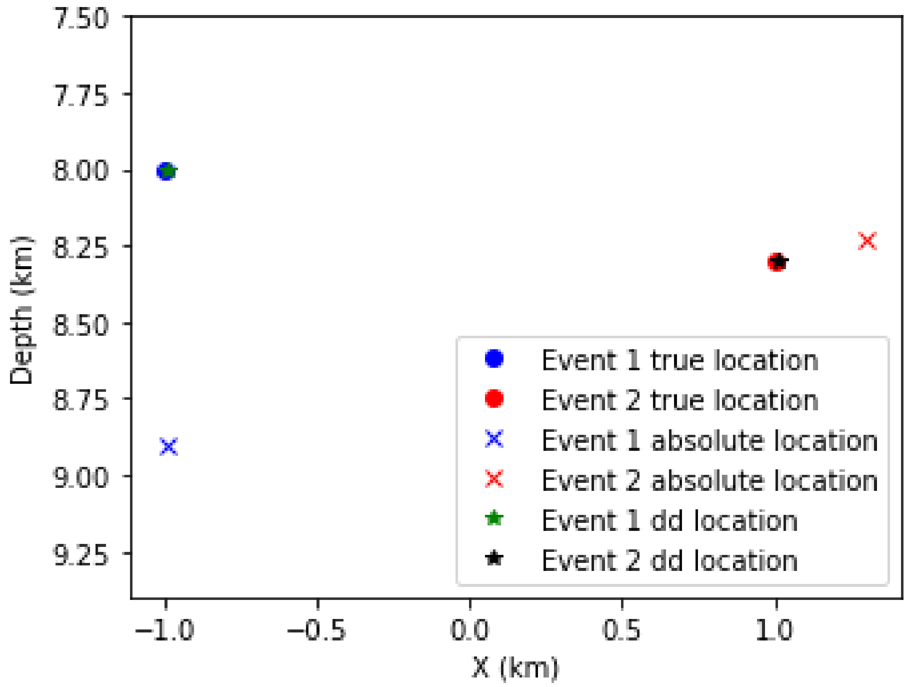
(continued from previous page)

```
    #----2. Update calculated travel time difference------------------
    dcal1 = np.zeros((dobs1.shape[0],1))
    for i in range(dobs1.shape[0]):
        dx = stas[i,0]-hyc1_loop[0]
        dz = stas[i,1]-hyc1_loop[1]
        dcal1[i,0] = np.sqrt(dx**2+dz**2)/Vtrue+hyc1_loop[2]
    dcal2 = np.zeros((dobs2.shape[0],1))
    for i in range(dobs1.shape[0]):
        dx = stas[i,0]-hyc2_loop[0]
        dz = stas[i,1]-hyc2_loop[1]
        dcal2[i,0] = np.sqrt(dx**2+dz**2)/Vtrue+hyc2_loop[2]
    cal_trav_t1 = dcal1 - hyc1_dd[2]
    cal_trav_t2 = dcal2 - hyc2_dd[2]
    cal_dt = cal_trav_t1 - cal_trav_t2
    #----3. Calculate double difference-------------------------------
    dtdt = obs_dt - cal_dt
    #----4. Set up G kernel-------------------------------------------
    ncol = event_number * event_parameters
    G = np.zeros((nsta,ncol))
    G[:,2]=1; G[:,5] = -1   # Partial derivative of origin column is 1
    for i in range(nsta):
        for j in range(2):
            denomiter1 = np.sqrt((hyc1_loop[0]-stas[i,0])**2+(hyc1_loop[1]-stas[i,1])**2)
            G[i,j]=(hyc1_loop[j]-stas[i,j])/denomiter1/Vtrue
            denomiter2 = np.sqrt((hyc2_loop[0]-stas[i,0])**2+(hyc2_loop[1]-stas[i,1])**2)
            G[i,j+3]=-(hyc2_loop[j]-stas[i,j])/denomiter2/Vtrue
    #---- Scale up G columns to unit length--------------------------
    Gnorms = np.zeros(ncol)
    for i in range(ncol):
        norm = np.linalg.norm(G[:,i])
        Gnorms[i] = norm
        G[:,i] = G[:,i]/norm
    #----6. LSQR and rescale solution---------------------------------
    damp = 0.1
    A = csc_matrix(G, dtype=float)
    m,istop,itn,r1norm,r2norm,anorm,acond,arnorm,xnorm,var=lsqr(A,dtdt,damp=damp,calc_
↪var=True)
    m = np.divide(m,Gnorms)
    var = np.divide(var,Gnorms**2)
    #----7. Update location-------------------------------------------
    hyc1_loop = hyc1_loop+m.ravel()[:3]
    hyc2_loop = hyc2_loop+m.ravel()[3:]
    #----8. Error Calculation-----------------------------------------
    print(f"Iteration {format(k,'4d')} residual: ",format(r1norm,'13.8f'))
    if r1norm<0.0000000001:
        print("Itertion stopped for too small error!")
        break
    k = k+1
#--------9. Variance analysis-------------------------------------------
sigma_m2 = np.diag(var)**2*r2norm**2
hyc1_dd = hyc1_loop
hyc2_dd = hyc2_loop
```

```
Iteration    0 residual:     0.01522528
Iteration    1 residual:     0.00683196
Iteration    2 residual:     0.00624882
Iteration    3 residual:     0.00581534
Iteration    4 residual:     0.00541587
Iteration    5 residual:     0.00504727
Iteration    6 residual:     0.00470742
Iteration    7 residual:     0.00439429
Iteration    8 residual:     0.00410597
Iteration    9 residual:     0.00384061
Iteration   10 residual:     0.00359645
......
Iteration   95 residual:     0.00013054
Iteration   96 residual:     0.00012620
Iteration   97 residual:     0.00012201
Iteration   98 residual:     0.00011796
Iteration   99 residual:     0.00011405
Iteration  100 residual:     0.00011026
```

We then find the residual is very small, and the earthquake location almost reached its true location

```
xmin = min(hyc1_true[0],hyc1_abs[0],hyc1_dd[0])
xmax = max(hyc1_true[0],hyc1_abs[0],hyc1_dd[0])
ymin = min(hyc1_true[1],hyc1_abs[1],hyc1_dd[1])
ymax = max(hyc1_true[1],hyc1_abs[1],hyc1_dd[1])
plt.plot(hyc1_true[0],hyc1_true[1],"bo",label="Event 1 true location")
plt.plot(hyc2_true[0],hyc2_true[1],"ro",label="Event 2 true location")
plt.plot(hyc1_abs[0],hyc1_abs[1],'bx',label="Event 1 absolute location")
plt.plot(hyc2_abs[0],hyc2_abs[1],'rx',label="Event 2 absolute location")
plt.plot(hyc1_dd[0],hyc1_dd[1],'*',color='green',label="Event 1 dd location")
plt.plot(hyc2_dd[0],hyc2_dd[1],'*',color='k',label="Event 2 dd location")
plt.gca().set_aspect('equal')
plt.legend()

plt.ylim(ymax+0.5,ymin-0.5)
plt.ylabel("Depth (km)")
plt.xlabel("X (km)");
```



```
present_loc_results(hyc1_dd,sigma_m2[:3,:3],std_fmt='.8f')
present_loc_results(hyc2_dd,sigma_m2[3:,3:],std_fmt='.8f')
```

```
x =   -0.99 ± 0.03210000  km
z =    8.00 ± 0.04610000  km
t =   -0.12 ± 0.00000000  s
x =    1.01 ± 0.03280000  km
z =    8.30 ± 0.04760000  km
t =    0.88 ± 0.00000000  s
```

11.16 Summary

In this tutorial, we first demonstrate the influence of velocity misfit and **Station delay**’s influence on earthquake location results.

We then introduce the double-difference method, which theoretically diminishes the station delay effect and limits the influence of velocity misfit. During processing, we: * Set up the data kernel **G** and calculate the double difference array **dtdt** * Add damping to the data kernel to make it stable (*determinant not be zero*) * Use **conditon number** to guide the selection of damping factor * Comparison shows **double-difference** location leads to location with **better performance**

Note:

we use one-layer velocity model for the convenience in finding the ray partial derivatives.

11.16.1 Homework

1. In the demo example, is event origin time fully recovered? Could you please explain the reason?(10 points)
2. Note we add negative symbol to partial derivatives of the event 2 in constructing the data kernel, do you know why? (10 points)
3. In calculating the variance(**var**), it is written `var = e2/(dtdt_damp.shape[0]-event_number * event_parameters)`, do you know why variance is different from square error here? (10 points)
4. Add one more event hyc_true3 = (0.2,8.1,1) (x,z,t) and prepare for inversion, set up suitable damping factor so that condition number is in the range 40-100.(80 points) - Show the absolute location result of the newly added event and its uncertainty. (20 points) - Show your data kernel G for Double-Difference inversion and its determinant. (20 points) - Show your Double-Difference inversion result and its uncertainty, how many iterations you used? (20 points) - Did your results get closer to true earthquake locations? Make a plot and show (20 points) #### Hint The dimension of m should be 9×1

$$m^T = \begin{bmatrix} \Delta x_1 & \Delta z_1 & \Delta t_1 & \Delta x_2 & \Delta z_2 & \Delta t_2 & \Delta x_3 & \Delta z_3 & \Delta t_3 \end{bmatrix}$$

For the double-difference value of event_1 and event_2 recorded in station k, its corresponding row of data kernel G should be:

$$\begin{bmatrix} \frac{\partial T_k^1}{\partial x} & \frac{\partial T_k^1}{\partial z} & \frac{\partial T_k^1}{\partial t} & -\frac{\partial T_k^2}{\partial x} & -\frac{\partial T_k^2}{\partial z} & -\frac{\partial T_k^2}{\partial t} & 0 & 0 & 0 \end{bmatrix}$$

For the double-difference value of event_2 and event_3 recorded in station k, its corresponding row of data kernel G should be:

$$\begin{bmatrix} 0 & 0 & 0 & \frac{\partial T_k^2}{\partial x} & \frac{\partial T_k^2}{\partial z} & \frac{\partial T_k^2}{\partial t} & -\frac{\partial T_k^3}{\partial x} & -\frac{\partial T_k^3}{\partial z} & -\frac{\partial T_k^3}{\partial t} \end{bmatrix}$$

11.16.2 Source code

Download tutorial code `here`

CHAPTER

TWELVE

SPECTRAL ANALYSIS

12.1 Introduction

12.1.1 What is earthquake stress drop?

Stress drop is an important earthquake source parameter, which is proportional to the ratio of fault slip to rupture extent (D/L). The higher stress drop means larger slip given the same rupture area. Since stress drop reflects the strength level on faults, the evolution of the stress drop may indicate some activities on faults such as fluid migration.

12.1.2 Stress drop in a circular fault

For a circular fault in a whole space, Eshelby(1957) obtained the relation between the average stress drop and the seismic moment:

$$\Delta\,\sigma = \frac{7\pi\mu\overline{D}}{16r} = \frac{7M_0}{16r^3}$$

M_0: moment; r: rupture radius; $\overline{D}$: average slip; μ: shear modules; $\Delta\,\sigma$: stress drop

12.1.3 What is earthquake spectrum?

We can calculate the spectrum for earthquakes from the seismograms after removing the effects from the path. The amplitude in the low frequency limit is determined by earthquake moment (M0). The spectrum starts to decay around corner frequency (fc), and keep a nearly constant decay rate in log-log domain in the high frequency.

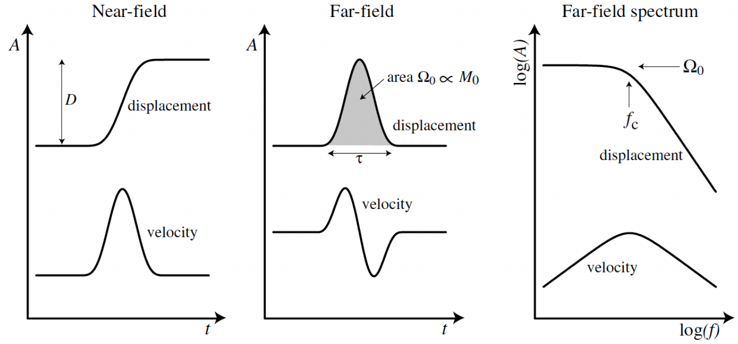


Figure 1: (Figure 9.13 in Introduction to Seismology) The relationships between near-field displacement and far-field displacement and velocity for time series (left two panels) and spectra (right panel).

12.1.4 Brune's model for far-field spectrum

Linking the spectra with the source parameters requires assumptions on the source. Brune (1970) assume a built up the relationship between the source spectra and the source parameters including the stress drop. He assumed that the stress pulse is tangential and rupture velocity is infinite on a circular rupture surface, then derived the near-filed displacement under the constraint of edge-effect. The slip and slip rate on the fault evolve with time.

$$u(x=0,t) = (\sigma/\mu)\beta\tau(1-e^{-t/\tau})$$

$$\dot{u}(x=0,t) = (\sigma/\mu)\beta e^{-t/\tau}$$

σ: effective stress ; τ: $\frac{r}{\beta}$; β: shear wave velocity; r: radius of circular rupture surface; μ: rigidity

The functions of slip and slip rate on faults in Brune's model

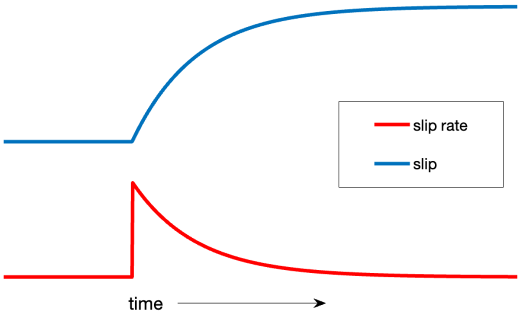


Figure 2: the shape of slip and slip rate function in Brune's model.

The conservation of energy density at high frequency and long-period limit(static field) were applied to constrain the displacement in Brune's model. Brune's model has been widely used in spectral analysis in the following format to link the spectra with the seismic moment and stress drop:

$$\Omega(\omega) = \frac{\Omega_0}{1+(\frac{f}{f_c})^2} \quad (1) \qquad \Omega_0 = \Re_{\theta\phi} M_0 (4\pi\rho R\beta^3)^{-1} \quad (2) \qquad \Delta\sigma = M_0(\frac{f_c}{0.42\beta})^3. \quad (3)$$

f_c : corner frequency; $\Re_{\theta\phi}$: radiation pattern; r: the radius of the rupture
R: the ray path length; β: P wave speed

12.1.5 Path and free surface effects

To calculate the source spectra, the effects from the path and depth phases should be removed. In this tutorial, we will try to remove the attenuation term. For earthquakes happened in shallow depth with relative long duration, the depth phases may overlap with P wave and impact the spectral shape (Hanks, 1981). Therefore, we need to calculate and remove the effect from those depth phases.

All those processing process will be included in the tutorial package `spectral_analysis.zip`. Following is an example of spectral analysis of 2017-11-13 M7.3 Iran earthquake.

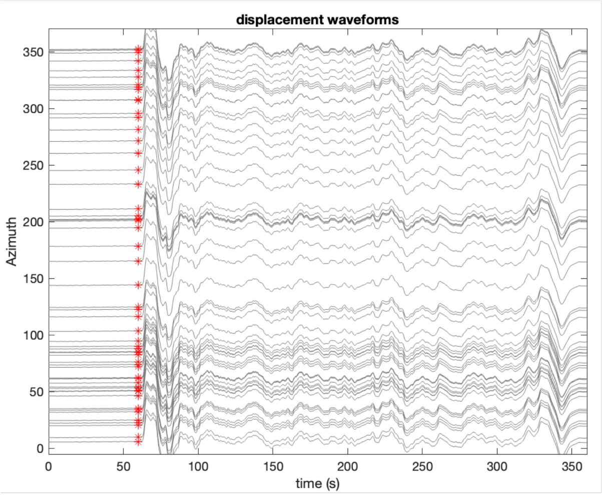
12.2 Example: the 2017-11-12 Iran M7.3 earthquake

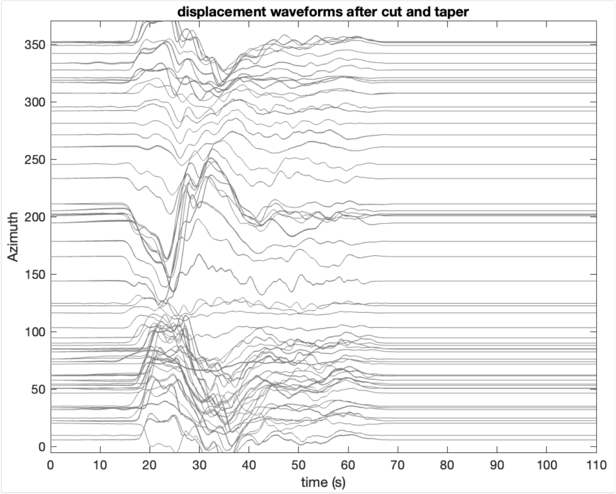
12.2.1 Download the waveforms and pre=processing

We use waveforms with epicentral distances of 30-90 degree. We provide some waveforms in the tutorial packages. You can also download the waveforms from IRIS. Considering the duration of the earthquake, we select a time window of 40s, starting from the P arrival for the spectra calculation. We taper the waveforms in 15s windows at the two ends of the selected time windows.

Note:

Pre-processing including taper, rmean, and so on are needed before start spectral analysis. Waveforms need to be transformed to displacement when removing the instrumental response. Please refer to the ObsPy tutorial. The waveforms provided in the package have been preprocessed. The codes for cutting and tapering are contained in the main script - spectral_analysis.m





Note:

We can see the waveform shape varies with azimuth due to the rupture directivity effect. It is suggested to use waveforms covering all the azimuth.

12.2.2 Calculate spectra

Then we calculate the spectra for each waveform traces by conducting fast Fourier transform. The function used here is `sacfft.m`

```
>>> function [amplitude, phase, frequency]=sacfft(displacement_seri, time_seri);
% Inputs: the displacement (m) and time series of the waveforms.
% Outputs in the left: the amplitude, phase term, and the frequency of the spectra
>>> amplitude=log10(amplitude);
% following processing are all conducted in log domain
```

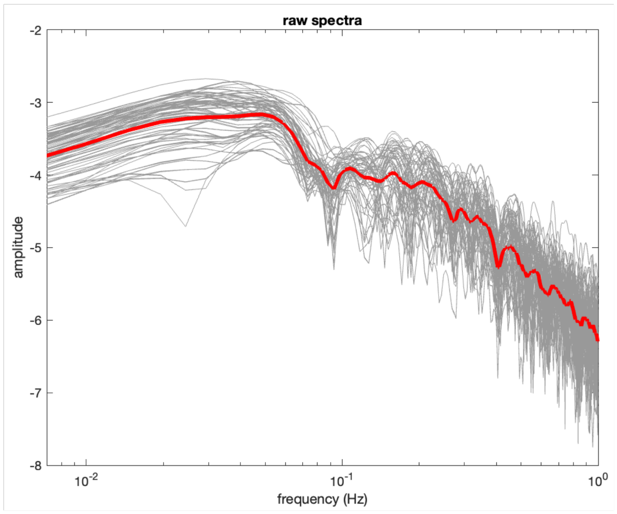
12.2.3 Correction for spreading term and radiation pattern

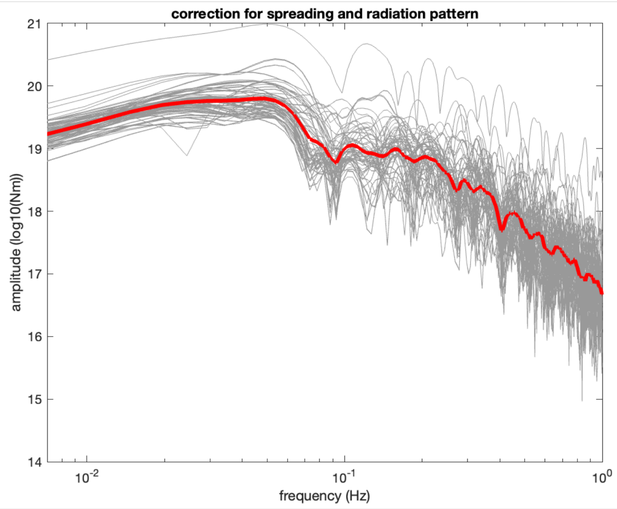
It has been illustrated in equation (2) that the amplitude of the spectra decay with ray path length with 1/R and is proportional to the radiation term. Here, we correct for the two terms using the functions `direct_P.m`

```
>>> function [radiationp,Ray_len,Amp_P]=direct_P(dist,depth,Azi,M,path)
% inputs in the right: epicentral distance (dist), source_depth(km,depth), azimuth of␣
↪station (Azi), moment tensor (M),path to the folder TT_M (path)
% outputs in the left: p wave radiation term (radiationp), ray path length (ray_len),␣
↪synthetic amplitude of P
```

The normalized moment tensor has been given as `M.dat` in the package. It can be downloaded from the GCMT website. The epicentral distance and azimuth info are extracted from the sac files. The `TT_M` folder contains info of the earth model and table for ray parameters. After calculate the synthetic amplitude given a normalized moment tensor, we correct the spectral by substract it from the raw spectra in the log domain. The the amplitude of the spectral should represent the seismic moment (Nm).

```
>>> amplitude=amplitude-log10(Amp_P);
% correction for the spreading term and the radiation pattern
```





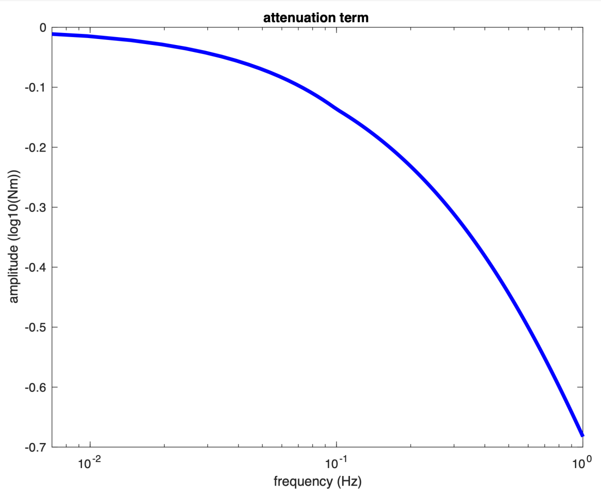
Note:

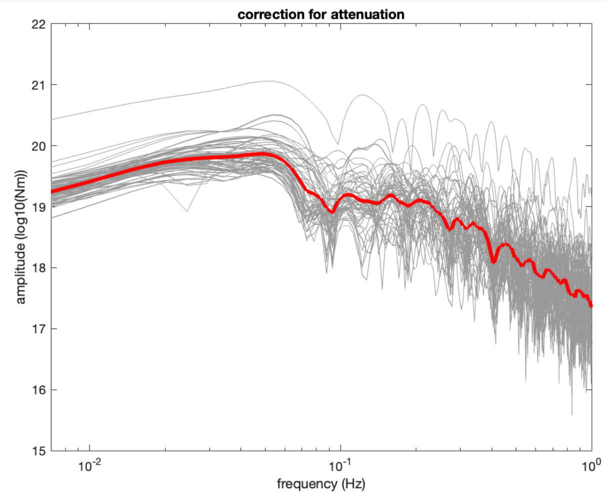
The amplitudes of spectra obviously become more concentrated after the correction, indicating that the scattering is mostly caused by the radiation pattern and the spreading term.

12.2.4 Correction for attenuation

Here, we adopt the global attenuation model to correct the spectra (Choy and Boatwright, 1985). The correction is based on the function `corre.m`.

```
>>> function tp = corre(f)
% f: frequency; tp: correction factor for attenuation as a function of frequency
>>> amplitude=amplitude-log10(corr(f));
% correct for attenuation
```

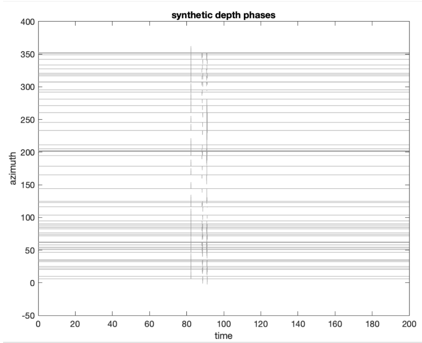


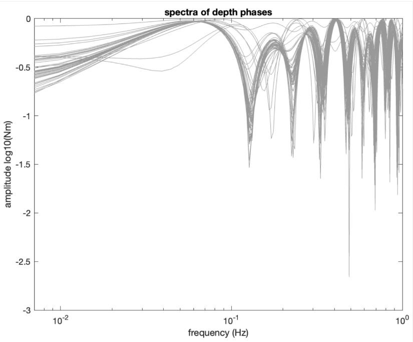


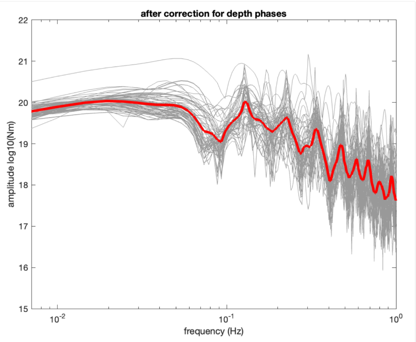
12.2.5 Correction for the depth phases

The depth of the event is 19 km in the SAC files. Given the depth here, the relative arrival times of sP and pP phases are within 15 seconds and contribute to the spectra. Here, we calculate the synthetic spectra for the depth phases effect using the function depth_phase_spectrum.m

```
>>> function [depth_time,response_depth,anp_dep,ph_dep,ffn_dep]=depth_phase_
↪spectrum(filename,window_length,datalength,M,path)
% Inputs: the name of the sac file (filename); the length of the selected windows in␣
↪second (window_length);
% Inputs: the length of the time series used in Fourier transform; moment tensor(M);path␣
↪to the folder TT_M
% Outputs: time series(depth_time);synthetic waveforms normalized by P wave amplitude;
% amplitude of spectra for depth phase effect (anp_dep); phase term (ph_dep); frequency␣
↪(ffn_dep)
>>> amplitude=amplitude-log10(anp_dep);
% correct for depth phase term effect
```







Note:

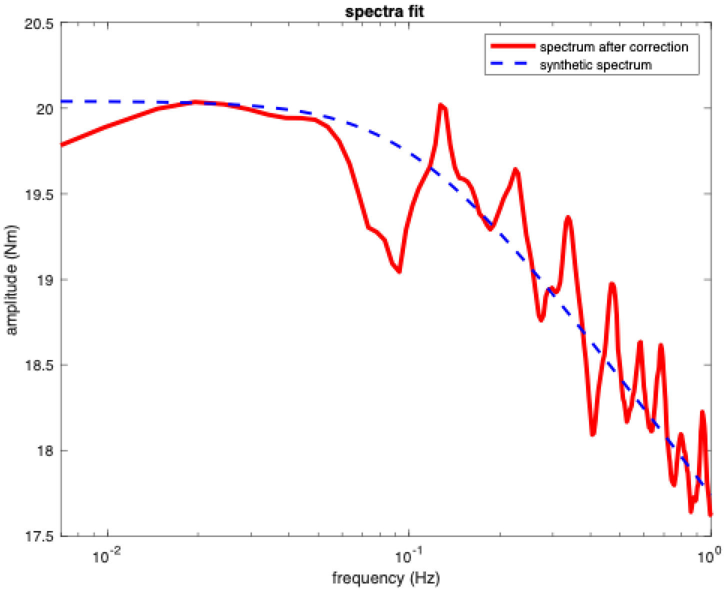
The depth phase will largely affect the waveform shape at low frequency. It is obvious that the spectra become more flat after correcting for the depth phases.

12.2.6 Estimating the moment and stress drop

The average magnitude of spectra in the low frequency limit is ~ 4*10^19 Nm. The moment magnitude is 1.1*10^20 Nm. The moment magnitude is 7.3. The corner frequency is ~0.1 Hz. Assuming a P wave speed of 3900 km/s and a density of 3000kg/m^3, the stress drop is estimated to be 25 MPa, residing in the normal range for continental earthquakes.

Note:

The fitting of the spectrum may bare considerable uncertainties according to your fitting algorithm. The uncertainties will be further amplified in the estimate for the stress drop. For example, if the corner frequency change by 20%, from 0.1 to 0.08 Hz, the estimated stress drop will decrease by half from 25 to 13 MPa.



CHAPTER

THIRTEEN

STATIC STRESS CHANGE - PYLITH

13.1 Brief introduction

Earthquake modeling provides a broad range of possible earthquake scenarios using physics-based models, which is critical for seismic hazard assessment. By incorporating observations and laboratory tests into models, we have better understanding of earthquake behaviors by recreating past earthquakes, and explore the range of potential future earthquakes, thus bridging our knowns and unknowns.

13.1.1 What is PyLith?

PyLith is a finite-element code for quasi-static and dynamic simulations of crustal deformation, primarily earthquakes and volcanoes. Quasi-static problems, also known as implicit problems, solve for the governing equations while neglecting the inertial terms. There are different quasi-static problems related to earthquakes in PyLith, including coseismic stress changes and fault slip, strain accumulation associated with interseismic deformation, and postseismic relaxation of the crust.

13.1.2 Aim of this module

In this teaching module, we will focus on solving for the stress change corresponding to a fault slip distribution.

Do you know why an earthquake would happen? When the stresses on a fault plane overcomes the friction, a sudden slip would occur on the fault, releasing energy in waves that travel through the earth's crust. This is known as an earthquake. Therefore, it is evident that there are stress changes on the fault after an earthquake.

In other words, given a slip distribution caused by an earthquake, there should be a corresponding stress change. This tutorial considers this as a quasi-static problem and aims to solve this problem through numerical simulation. In PyLith, the problem will be solved when the residual of the governing equation approaches zero.

In this tutorial, we will demonstrate a simple 3D case study. We will prescribe the heterogeneous slip distribution on the fault and the fault geometry using the mesh shown in the figure below.

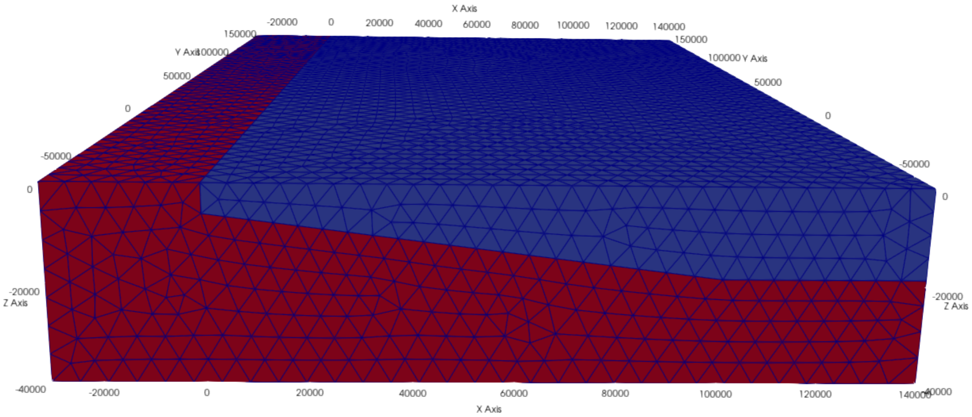


figure 1. 3D mesh given in this tutorial with dimensions in kilometers

Then we will run the static simulation to obtain the stress change, so you will learn:

1. the input parameters,
2. initiating a quasi-static simulation,
3. visualizing the output.

13.1.3 How to install PyLith?

1. Download the PyLith package suitable for your computer here, and then move it into your working directory

2. Open a terminal in Linux or Mac OS X (Darwin) platform and run these commands sequentially. For Windows 10 Users, we recommend using the Windows Subsystem for Linux (WSL). The installation of WSL could be found here or searched online. Please note that PyLith is using Python2.7. For the new MacOS updates where Python2.7 is removed, you would need to set the environment.

Warning: Exclude $ and start without whitespace!

```
$ tar xvf pylith-2.2.2-linux-x86_64.tar.gz
$ cd pylith-2.2.2-linux-x86_64/
$ source setup.sh
```

3. Check if it has been successfully installed

```
$ pylith
 >> {default}::
 -- pyre.inventory(error)
```

(continues on next page)

(continued from previous page)

```
 -- meshimporter.meshioascii.filename <- ''
 -- Filename for ASCII input mesh not specified.  To test PyLith, run an example as␣
↪discussed in the manual.
 >> {default}::
 -- pyre.inventory(error)
 -- timedependent.homogeneous.elasticisotropic3d.label <- ''
 -- Descriptive label for material not specified.
 >> {default}::
 -- pyre.inventory(error)
 -- timedependent.homogeneous.elasticisotropic3d.simpledb.label <- ''
 -- Descriptive label for spatial database not specified.
 >> {default}::
 -- pyre.inventory(error)
 -- timedependent.homogeneous.elasticisotropic3d.simpledb.simpleioascii.filename <- ''
 -- Filename for spatial database not specified.
pylithapp: configuration error(s)
```

13.2 Simple 3D case study - Nepal

1. Download the tutorial package `pylith_static.tar.gz` and move it to your working directory
2. Run this command to decompress the package

```
$ tar xvf pylith_static.tar.gz
```

3. `ls` different folders in the directory to familiarize yourself with the different files required for a static simulation

For running the static simulation, you will need configuration files (.cfg) that specifies the problem, mesh (.exo) where the simulation takes place, spatial database files (.spatialdb) that describes the variables in space. Here we have two spatial database files - one for the material property, and one for the fault slip distribution. Since the mesh generating softwares are commercial, we will only illustrate the structure and some important parameters of the configuration and spatial database files in the following.

13.2.1 Basic structure of a configuration (.cfg) file

Now, let's take a took at the **Nepal_kinematic_model.cfg** under the **pylith_static** directory. For a simulation in PyLith, you would need a **configuration file (.cfg)** which specifies the basic parameters of the simulation.

1. Problem

`[pylithapp.timedependent.formulation.time_step]` Settings that control the time of the problem. Note that static, quasi-static, and dynamic simulations are also time dependent problems in PyLith. `total_time` specifies the total time. We adjust the total simulation time to 0 second because we are running a static simulation. dt specifies the time step size.

[pylithapp.mesh_generator.reader] Settings that control mesh importing. filename specifies the filename of the mesh.

[pylithapp.timedependent.materials] Settings that control the material type. db_properties.iohandler.filename specifies the name of the spatial database containing the physical properties for the material. db_properties.query_type specifies the type of search query to perform. This parameter can be set to 'linear' or 'nearest'.

2. Boundary condition

[pylithapp.timedependent] Settings that control the problem, including the spatial dimension of the mesh. bc specifies the boundary conditions for different sides of the mesh. bc_dof specfies which degrees of freedom are being constrained for the boundary conditions. Note that the Degree of freedoms are: x=0, y=1, and z=2. For instance, bc = [2] refers to fixed displacement in z direction, and bc = [0, 1] means fixed displacement in x and y-directions. label specfies the name of the nodeset in ExodusII file from CUBIT/Trelis that defines the boundary. db_initial.label specfies the label for the spatial database which is required for informative error messages.

3. Faults

interfaces specifies an array containing the fault interfaces. Here we have one fault interface so we provide an array containing a single interface.

[pylithapp.timedependent.interfaces] fault specifies the type of fault interface condition. For this example we want to prescribe the fault slip, so the interface type is set to **FaultCohesiveKin**.

[pylithapp.timedependent.interfaces.fault]

label specifies the name of the nodeset in CUBIT/Trelis that defines the fault interface.

edge specifies the name of the nodeset in CUBIT/Trelis marking the buried edges of the fault.

quadrature.cell specifies the discretization components for fault cells. FIATSimplex deals with simplex finite-element cells, including point, line, triangle, and tetrahedron. We are having triangular cells on our fault interface mesh so FIATSimplex scheme is chosen here.

quadrature.cell.dimension specifies the dimension of fault cells. The fault cells are 2D in our case.

[pylithapp.timedependent.interface.fault.eq_scrs.rupture.slip_function] Settings for prescribing the coseismic slip distribution on the fault, including the final slip and slip initiation time.

slip.iohandler.filename specifies the name of the spatial database containing the coseismic slip distribution.

slip.query_type specifies the type of search query to perform. Here we define the slip to be a spatial database with linear interpolation.

slip_time.data specifies the slip time within an earthquake rupture relative to the origin time.

4. Output

Settings related to output of the solution over the domain, subdomain (ground surface), and synthetic stations.

`output` specifies the outputs of simulation. Note that the default output is for the entire domain.

`output.station` specifies the type of output for the stations.

`[pylithapp.timedependent.formulation.output.domain]` Settings for domain output.

`writer.filename` specifies the filename of domain output.

`[pylithapp.timedependent.formulation.output.station]` Settings for the station outputs.

`reader.filename` specifies the file with coordinates of stations.

`writer.filename` specifies the filename of station output.

`coordsys` specifies coordinate system used in station file.

Similar output parameters for the fault and the materials.

Note:

The above only documents the some parameters that we may change specified to our static simulation. Note that under this directory, there is also another configuration file **pylithapp.cfg**. **pylithapp.cfg** is not a self-contained simulation configuration file but it specifies the general parameters common to the simulations under this directory. This file is necessary for running our simulation.
For more functions and information, please browse through the `PyLith manual`.

13.2.2 Basic structure of a spatial database (.spatialdb) file

After knowing what the configuration files do, let's learn about the spatial database files under the spatialdb directory.

1. Slip_distribution.spatialdb

This spatial database file specifies the distribution of slip on the fault surface. The slip is dependent on the x and y-directions but independent of the depth.

`SPATIAL.ascii 1` the magic header for spatial database files in ASCII format.

`SimpleDB` specifies spatial database files contain a header describing the set of points and then the data with each line listing the coordinates of a point followed by the values of the fields for that point.

`num-values` number of values in the database

`value-names` specifies the names and the order of the values as they appear in the data. Note that the names of the values must correspond to the names PyLith requests in querying the database. Here we are having the slip distributions in three different directions - left-lateral (along-strike), reverse (along-dip), and fault-opening (fault-normal).

`value-units` specifies the units of the values in Python syntax (e.g. m, kg/m**3).

`num-locs` specifies the number of locations where values are given.

`data-dim` specifies the locations of data points form a line.

`space-dim` specifies the spatial dimension in which data resides.

`cs-data` specifies the coordinate system associated with the coordinates of the locations where data is given. We are using a Cartesian coordinate system here.

`to-meters` specifies the coordinates in km.

space-dim specifies the spatial dimension of the coordinate system. Note that this value must match the one associated with the database.

2. mat_concrust_1D.spatialdb

This spatial database file specifies the material properties. Here we prescribe depth dependent material properties - density, compressional wave velocity vp, and shear wave velocity vs. The parameter setting is basically the same with the spatial database file about slip distribution as illustrated above except value-names, value-units, and data-dim are changed.

13.2.3 Running the static simulation

Now that we have a general picture of the files required for our simulation, let's run the simulation.

```
$ cd your_working_directory/pylith_static/
$ pylith Nepal_kinematic_model.cfg
```

Congrats on running your first simulation!

13.3 Visualizing results

After finishing the simulation, you should be able to see different output files under the **output** directory. In this tutorial, you will learn how to process the fault output in .h5 format using Python. Hierarchical Data Format (HDF) is a set of file formats (HDF4, HDF5) designed to store and organize large amounts of scientific data. We assume that you already have some experience of using Python.

13.3.1 How to install h5py

```
$ conda create --name pylith
$ conda activate pylith
$ conda install h5py
```

Please also make sure your have installed matplotlib in your environment. Now, let's run the following commands.

1. Import the necessary packages

```
import h5py
import matplotlib.pyplot as plt
```

2. Read the .h5 files

```
filename = 'your_directory/output/Nepal_kinematic_model-fault.h5'
f = h5py.File(filename,'r')
fields = list(f)
```

If you check `fields`, you would realize that there are four fields in this file - geometry, time, topology, and vertex_fields. In this tutorial, we will plot results using data from the vertices under `geometry`, and the slip and traction change under `vertex_fields`. For your reference, the figure below illustrates the general layout of a PyLith HDF5 file (extracted from the `PyLith manual`).

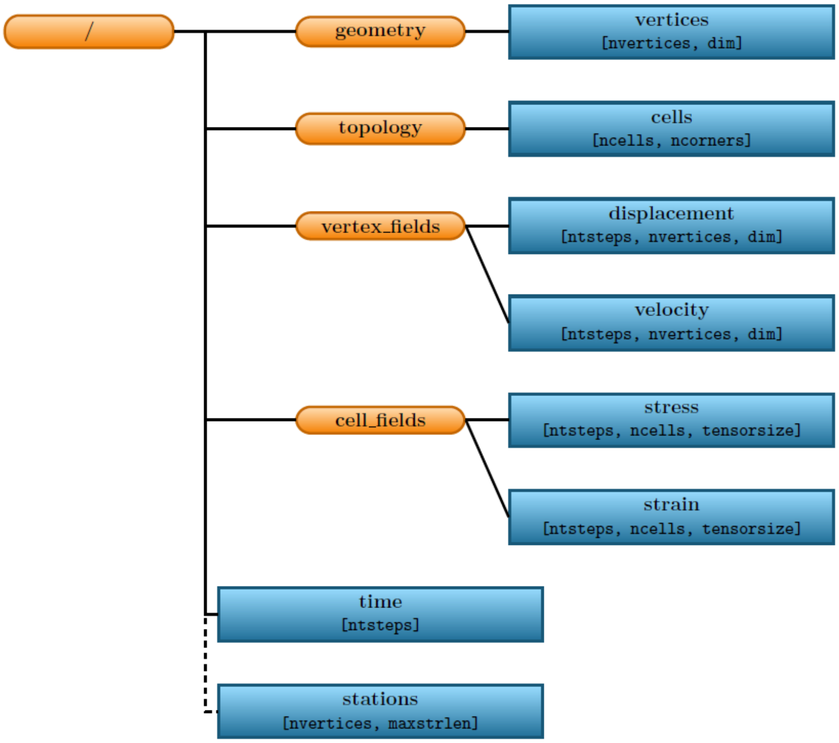


figure 2. General layout of a PyLith HDF5 file. The orange rectangles with rounded corners identify the groups and the blue rectangles with sharp corners identify the datasets. The dimensions of the data sets are shown in parentheses. Most HDF5 files will contain either vertex_fields or cell_fields but not both. (Adopted from PyLith Manual)

3. Extract the data

```
slip = f.get('/vertex_fields/slip')
traction_change = f.get('/vertex_fields/traction_change')
geometry = f.get('/geometry')
geometry_vertices = f.get('/geometry/vertices')

x_vertex = geometry_vertices[:,0]/1000   # along dip distance in km
y_vertex = geometry_vertices[:,1]/1000   # along strike distance in km
dip_slip = slip[:,:,1]              # updip slip in m
lateral_slip = slip[:,:,0]              # left lateral slip in m
dip_traction = traction_change[:,:,1]/10**6    # updip traction in MPa
lateral_traction = traction_change[:,:,0]/10**6    # left lateral traction in MPa
```

4. Plot the slip distribution and traction change

```
fig = plt.figure(facecolor='white', figsize=(14, 14))
size = 13

ax1 = fig.add_subplot(221, aspect='equal')
scat = ax1.scatter(x_vertex,y_vertex, s = size,c=dip_slip,cmap = 'jet', vmin = 0, vmax =␣
↪5.5)
plt.xticks(fontsize=11)
plt.yticks(fontsize=11)
plt.title('Updip slip',size = 14)
ax1.set_xlabel('along-dip (km)', size = 12)
ax1.set_ylabel('along-strike (km)', size = 12)
cbar = fig.colorbar(scat,ax=ax1)
cbar.ax.tick_params(labelsize=10)
ax1.spines['right'].set_visible(False)
ax1.spines['top'].set_visible(False)

ax2 = fig.add_subplot(222, aspect='equal')
scat = ax2.scatter(x_vertex,y_vertex, s = size,c=dip_traction,cmap = 'seismic', vmin = -
↪20, vmax = 20)
plt.xticks(fontsize=11)
plt.yticks(fontsize=11)
plt.title('Updip traction change',size = 14)
ax2.set_xlabel('along-dip (km)', size = 12)
ax2.set_ylabel('along-strike (km)', size = 12)
cbar = fig.colorbar(scat,ax=ax2)
cbar.ax.tick_params(labelsize=10)
ax2.spines['right'].set_visible(False)
ax2.spines['top'].set_visible(False)

ax3 = fig.add_subplot(223, aspect='equal')
reversed_map = plt.cm.get_cmap('jet').reversed()
scat = ax3.scatter(x_vertex,y_vertex, s = size,c=lateral_slip,cmap = reversed_map, vmin␣
↪= -5.5, vmax = 0)
plt.xticks(fontsize=11)
plt.yticks(fontsize=11)
plt.title('Left lateral slip',size = 14)
ax3.set_xlabel('along-dip (km)', size = 12)
ax3.set_ylabel('along-strike (km)', size = 12)
cbar = fig.colorbar(scat,ax=ax3)
cbar.ax.tick_params(labelsize=10)
ax3.spines['right'].set_visible(False)
ax3.spines['top'].set_visible(False)
#plt.ylim([-10, 210])

ax4 = fig.add_subplot(224, aspect='equal')
scat = ax4.scatter(x_vertex,y_vertex, s = size,c=lateral_traction,cmap = 'seismic', vmin␣
↪= -20, vmax = 20)
plt.xticks(fontsize=11)
plt.yticks(fontsize=11)
plt.title('Left lateral traction change',size = 14)
```

(continues on next page)

(continued from previous page)

```
ax4.set_xlabel('along-dip (km)', size = 12)
ax4.set_ylabel('along-strike (km)', size = 12)
cbar = fig.colorbar(scat,ax=ax4)
cbar.ax.tick_params(labelsize=10)
ax4.spines['right'].set_visible(False)
ax4.spines['top'].set_visible(False)

plt.tight_layout()
plt.show()
```

You should be able to generate the following plot.

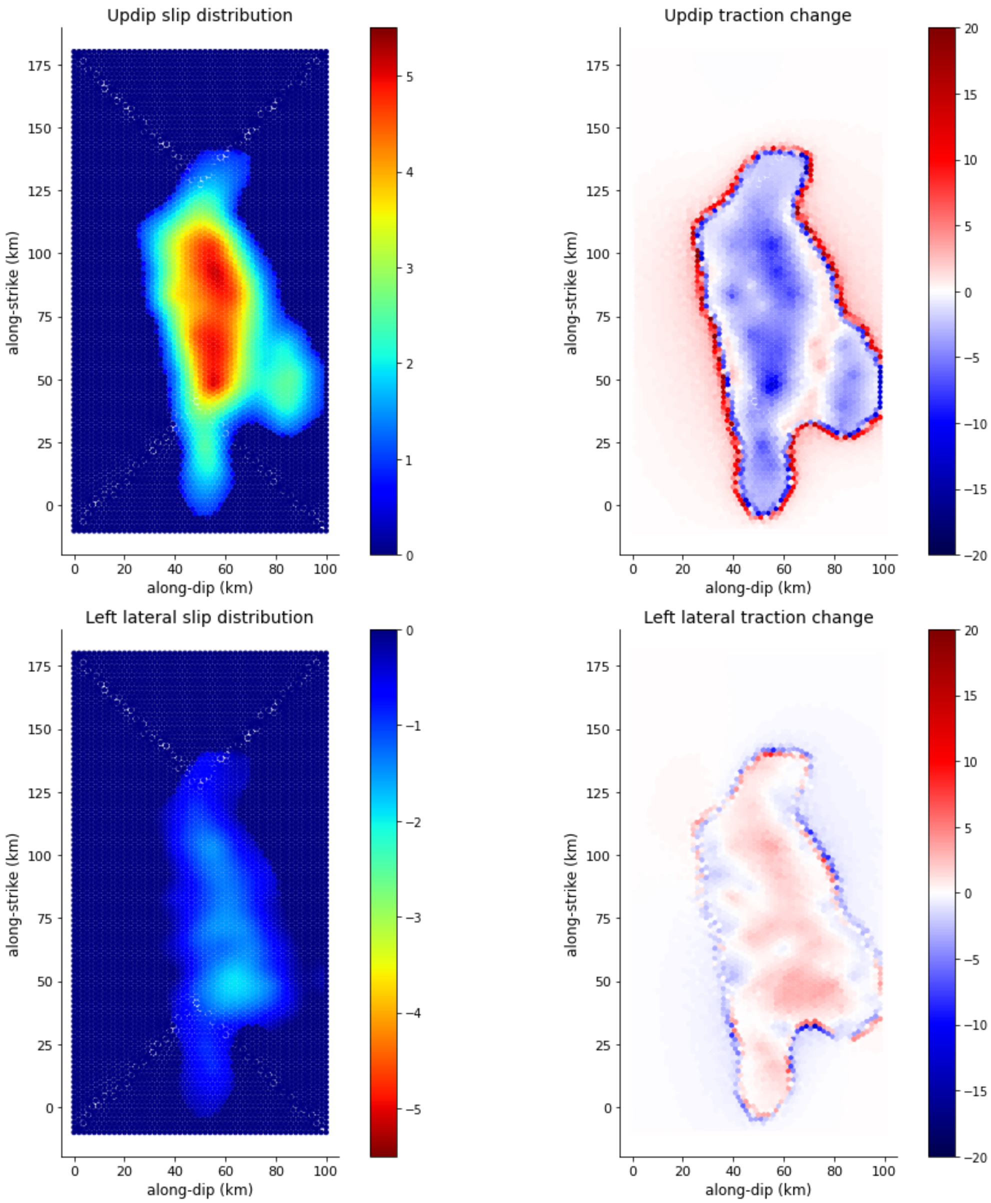


figure 3. A plot of the prescribed slip distributions and their corresponding stress changes

CHAPTER

FOURTEEN

DYNAMIC RUPTURE SIMULATION

14.1 Brief introduction

Dynamic modeling of earthquakes provides important insights in the physics of earthquake rupture processes. For example, earthquake initiation, propagation, arrest, afterslip, locking, and interseismic stress accumulation can be solved. In particular, dynamic earthquake rupture codes compute the physical earthquake behaviors over short time scales, producing possible earthquake size, fault slip amount, ground motion, and crustal deformation, which are crucial for risk assessment.

Currently, dynamic rupture simulations are actively used in seismology research for reconstructing well-recorded earthquakes as well as investigating the complex factors controlling earthquakes, such as fault geometry, fault frictional properties, and initial stresses.

14.1.1 Aim of this module

There are different dynamic earthquake rupture codes for a range of code types, including finite element, discontinuous Galerkin finite element, and spectral element methods. In this module, we will focus on **PyLith** which is a finite-element code.

In this tutorial, we are going to conduct a dynamic rupture simulation using a simple 3D example. As illustrated in the below figure, the inputs of PyLith requires the fault geometry and rock properties, initial stresses, and fault friction.

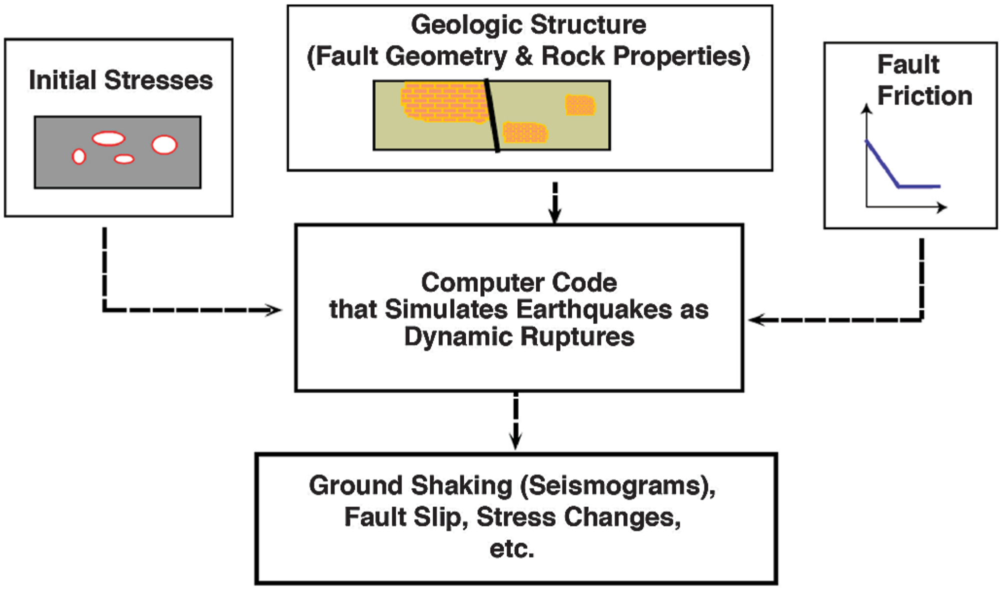

Initial Stresses
Geologic Structure
(Fault Geometry & Rock Properties)
Fault
Friction
Computer Code
that Simulates Earthquakes as
Dynamic Ruptures
Ground Shaking (Seismograms),
Fault Slip, Stress Changes,
etc.

figure 1. Key ingredients of a dynamic (spontaneous) earthquake rupture simulation (Harris et al. 2018)

The fault geometry is described by the given mesh. In this example, we will assume the materials to be elastic. For the rock properties, we will use a spatial database to specify a simple depth-dependent material properties, including density, compressional wave velocity (vp), and shear wave velocity (vs).

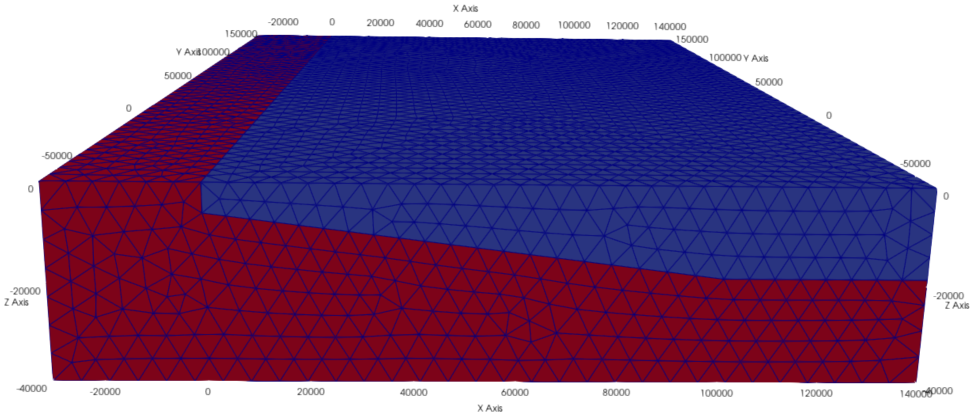


figure 2. 3D mesh given in this tutorial with dimensions in kilometers

There are different fault frictional laws. Here we are going to introduce the linear slip-weakening law, which is described by yield strength/stress, initial stress, sliding-friction/dynamic stress, and critical slip distance dc. Yield strength/stress is the maximum stress level that can be applied before the fault slips. Sliding friction is the background stress level. The difference between the initial stress and the yield strength is known as the strength excess. A fault has to overcome this strength excess to allow ruptures.

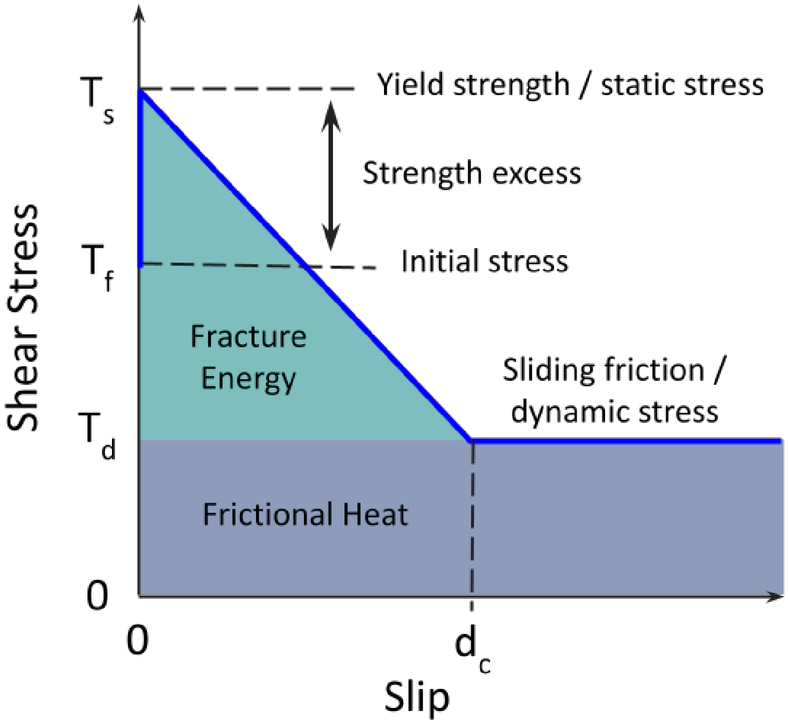


figure 3. Schematic diagram of linear slip-weakening model

We can also take a look at the equation for the traction on the fault plane using linear slip-weakening law. When the fault slip (d) is within the critical slip distance (dc), the stress level (Tf) decreases linearly from the static (Ts) to dynamic level (Td). When the fault slip exceeds the critical slip distance, the stress level is the same as the sliding friction.

$$T_f = \begin{cases} T_s - (T_s - T_d)\frac{d}{d_c} & \text{if } d \le d_c \\ T_d & \text{if } d > d_c \end{cases}$$

In **PyLith**, we would use friction coefficients to specify the frictional properties of the fault. Yield strength is the product of static coefficient and fault normal traction while sliding friction is the product of dynamic coefficient and fault normal traction. In this example, we are using homogeneous initial stresses and dynamic stress on fault. To initiate the rupture, we lower the yield stress below the initial stress level within a circular zone. You can also visualize the yield strength, initial stress, and dynamic stress as the following plot.

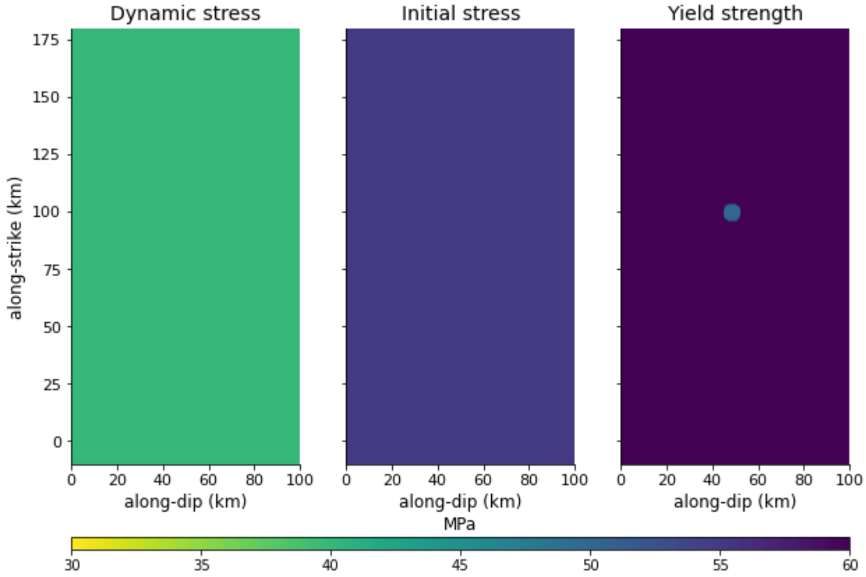


figure 4. Plot of the stresses on the fault in our example

Now, you already have a general picture of our problem in this tutorial. In the following, you will learn:

1. the input parameters
2. initiating a rupture simulation
3. visualizing the output

Note:

Here we assume that you have already installed and initialized PyLith. If not, you may refer to our module on static simulations .

14.2 Simple 3D case study - Nepal

1. Download the tutorial package `pylith_dynamic.tar.gz` and move it to your working directory
2. Run this command to decompress the package

```
$ tar xvf pylith_dynamic.tar.gz
```

3. `ls` different folders in the directory to familiarize yourself with the different files required for a dynamic simulation

For running the dynamic simulation, you will need configuration files (.cfg) that specifies the problem, mesh (.exo) where the simulation takes place, spatial database files (.spatialdb) that describes the variables in space. Here we have three spatial database files - material property, frictional property, and traction. Since the mesh generating softwares are commercial, only the configuration and spatial database files will be illustrated in the following.

14.2.1 Basic structure of a configuration (.cfg) file

Now, let's take a took at the **dynamic_rupture_model.cfg** under the **pylith_dynamic** directory. For a simulation in PyLith, you would need a **configuration file (.cfg)** which specifies the basic parameters of the simulation. Many of the commands are the same with those used in a static simulation so the following will only introduce some parameters specified for dynamic simulations.

1. Problem

simulations. `normalizer` used to nondimensionalize the equations. `shear_wave_speed` specifies the shear wave speed used to nondimensionalize length and pressure (default is 3.0 km/s). `mass_density` specifies the mass density to nondimensionalize density and pressure (default is 3.0e+3 kg/m3). `wave_period` specifies the period of seismic waves used to nondimensionalize time (default is 1.0 s).

2. Boundary condition

`pylith.bc.AbsorbingDampers` specifies absorbing boundary condition. Note that this function does not perfectly absorb all incident waves. Please refer to the `manual` for more details.

3. Faults

`pylith.faults.FaultCohesiveDyn` specifies dynamic fault interface. `pylith.friction.SlipWeakening` specifies linear slip-weakening friction fault constitutive model. There are also other friction models, e.g. rate and state friction, and linear time-weakening friction model. `friction.db_properties.idhandler.filename` specifies the filename for the friction parameters. `traction_perturbation` specifies initial tractions on fault surface. `db_initial.data` specifies the left lateral shear traction, updip shear traction, and normal traction (tension is positive, compression is negative) to be 0 MPa, 55 MPa, -100 MPa.

Note:

The above only documents the some parameters that we may change specified to our static simulation. Note that under this directory, there is also another configuration file **pylithapp.cfg**. **pylithapp.cfg** is not a self-contained simulation configuration file but it specifies the general parameters common to the simulations under this directory. This file is necessary for running our simulation.

For more functions and information, please browse through the `PyLith manual`.

14.2.2 Basic structure of a spatial database (.spatialdb) file

After knowing what the configuration files do, let's learn about the spatial database files under the spatialdb directory. The structure and parameter settings of the spatial database files used here are basically the same with those introduced in the static simulation tutorial but their contents are different, thus `value-names`, `value-units`, and `data-dim` are changed.

1. friction.spatialdb

This spatial database file specifies the parameters required in the frictional law. As we are using the linear slip-weakening friction model, the static coefficient, dynamic coefficient, critical slip distance, and cohesive stress are specified in this file.

2. mat_concrust_1D.spatialdb

This spatial database file specifies the material properties. Here we prescribe depth dependent material properties - density, compressional wave velocity vp, and shear wave velocity vs.

14.2.3 Running the dynamic simulation

Now that we have a general picture of the files required for our simulation, let's run the simulation.

```
$ cd your_working_directory/pylith_dynamic/
$ pylith dynamic_rupture_model.cfg
```

Now the current timesteps are displayed in the terminal, and you can wait for the simulation to finish.

14.3 Visualizing results

After finishing the simulation, you should be able to see different output files under the **output** directory. In this tutorial, you will learn how to process the fault output in .h5 format using Python. Hierarchical Data Format (HDF) is a set of file formats (HDF4, HDF5) designed to store and organize large amounts of scientific data. We assume that you already have some experience of using Python and installed h5py. If not, please refer to our static simulation tutorial .

14.3.1 How to install h5py

```
$ conda create --name pylith
$ conda activate pylith
$ conda install h5py
```

Please also make sure your have installed matplotlib in your environment. Now, let's run the following commands.

1. Import the necessary packages

```
import h5py
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable
import matplotlib.animation as manimation
```

2. Read the .h5 files

```
filename = 'C:/Users/ypbow/OneDrive/Desktop/mphil/teaching_modules/pylith_dynamic/output/
↪v5/Nepal_dynamic_model-fault.h5'
f = h5py.File(filename,'r')
fields = list(f)
```

If you check `fields`, you would realize that there are four fields in this file - geometry, time, topology, and vertex_fields. In this tutorial, we will plot results using data from the vertices under `geometry`, time in `time` as well as the slip, slip rate, and traction under `vertex_fields`. For your reference, the figure below illustrates the general layout of a PyLith HDF5 file (extracted from the `PyLith manual`).

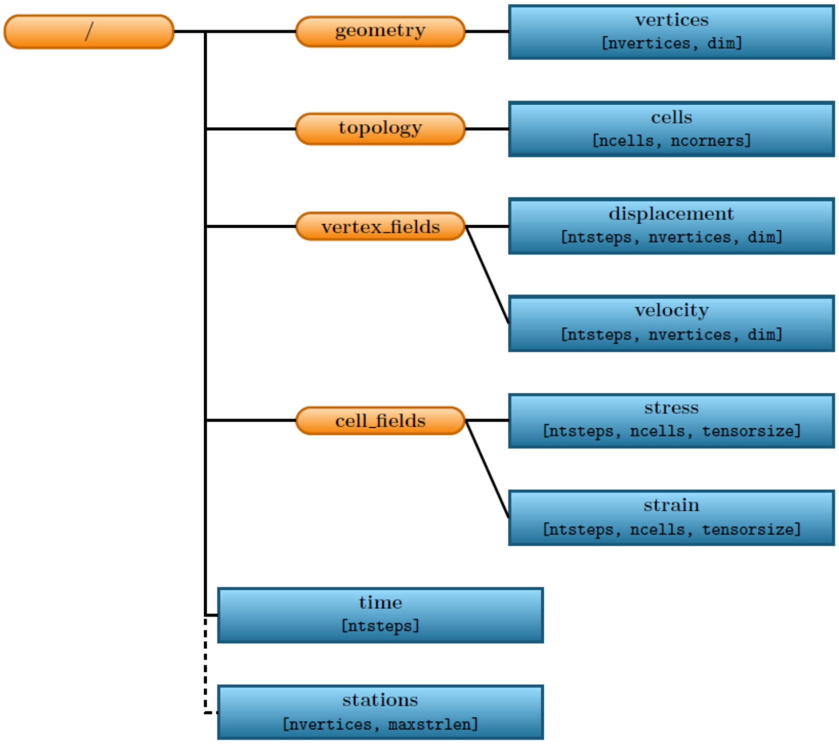


figure 5. General layout of a PyLith HDF5 file. The orange rectangles with rounded corners identify the groups and the blue rectangles with sharp corners identify the datasets. The dimensions of the data sets are shown in parentheses. Most HDF5 files will contain either vertex_fields or cell_fields but not both. (Adopted from PyLith Manual)

3. Extract the data

Since our simulation prescribed zero left-lateral shear traction, we will only focus on the updip shear traction and slip patterns in our plot.

```
time = f.get('/time')
slip = f.get('/vertex_fields/slip')
slip_rate = f.get('/vertex_fields/slip_rate')
traction = f.get('/vertex_fields/traction')
geometry_vertices = f.get('/geometry/vertices')

x_vertex = geometry_vertices[:,0]/1000   # along dip distance in km
y_vertex = geometry_vertices[:,1]/1000   # along strike distance in km
dip_slip = slip[:,:,1]              # updip slip in m
dip_slip_rate = slip_rate[:,:,1]  # updip slip rate in m/s
dip_traction = traction[:,:,1]/10**6    # updip traction in MPa
```

4. Make an animation for the updip slip distribution, slip rate, and traction during the rupture

```
# Define the meta data for the movie
FFMpegWriter = manimation.writers['ffmpeg']
metadata = dict(title='updip_fault_movie', artist='Matplotlib')
writer = FFMpegWriter(fps=5, metadata=metadata)

# Plot figure
fig = plt.figure(facecolor='white', figsize=(14, 14))
size = 15

with writer.saving(fig, "updip_fault.mp4", 100):    # or .gif
    for t in range(len(time)):
        ax1 = fig.add_subplot(131, aspect='equal')
        scat = ax1.scatter(x_vertex,y_vertex, s = size,c=dip_slip[t,:],cmap = 'jet',vmin␣
↪= 0,vmax = 65)
        plt.xticks(fontsize=11)
        plt.yticks(fontsize=11)
        plt.title('Updip slip',size = 14)
        ax1.set_xlabel('along-dip (km)', size = 12)
        ax1.set_ylabel('along-strike (km)', size = 12)
        plt.xlim([0, 100])
        plt.ylim([-10,180])
        divider = make_axes_locatable(ax1)
        cax = divider.append_axes('right', size='5%', pad=0.05)
        cbar1 = fig.colorbar(scat,cax=cax, orientation='vertical')
        cbar1.ax.set_title('m')
        cbar1.ax.tick_params(labelsize=10)
        ax1.spines['right'].set_visible(False)
        ax1.spines['top'].set_visible(False)

        ax2 = fig.add_subplot(132, aspect='equal')
        reversed_map = plt.cm.get_cmap('inferno').reversed()
        scat = ax2.scatter(x_vertex,y_vertex, s = size,c=dip_slip_rate[t,:],cmap =␣
↪reversed_map,vmin = 0,vmax = 4)
        plt.xticks(fontsize=11)
        plt.yticks(fontsize=11)
        plt.title('Updip slip rate',size = 14)
        ax2.set_xlabel('along-dip (km)', size = 12)
        ax2.set_ylabel('along-strike (km)', size = 12)
        plt.xlim([0, 100])
        plt.ylim([-10,180])
        divider = make_axes_locatable(ax2)
        cax = divider.append_axes('right', size='5%', pad=0.05)
        cbar2 = fig.colorbar(scat,cax=cax, orientation='vertical')
        cbar2.ax.set_title('m/s')
        cbar2.ax.tick_params(labelsize=10)
        ax2.spines['right'].set_visible(False)
        ax2.spines['top'].set_visible(False)

        ax3 = fig.add_subplot(133, aspect='equal')
        reversed_map2 = plt.cm.get_cmap('viridis').reversed()
        scat = ax3.scatter(x_vertex,y_vertex, s = size,c=dip_traction[t,:],cmap =␣
↪reversed_map2,vmin = 0,vmax = 60)
        plt.xticks(fontsize=11)
```

(continues on next page)

(continued from previous page)

```
        plt.yticks(fontsize=11)
        plt.title('Updip traction',size = 14)
        ax3.set_xlabel('along-dip (km)', size = 12)
        ax3.set_ylabel('along-strike (km)', size = 12)
        plt.xlim([0, 100])
        plt.ylim([-10,180])
        divider = make_axes_locatable(ax3)
        cax = divider.append_axes('right', size='5%', pad=0.05)
        cbar3 = fig.colorbar(scat,cax=cax, orientation='vertical')
        cbar3.ax.set_title('MPa')
        cbar3.ax.tick_params(labelsize=10)
        ax3.spines['right'].set_visible(False)
        ax3.spines['top'].set_visible(False)

        fig.suptitle('Time = '+str(t)+' s', y = 0.77,size = 17)
        plt.tight_layout()
        #plt.show()
        writer.grab_frame()
```

You should be able to generate the following animation.

figure 6. An animation of the updip slip distribution, slip rate, and traction during the rupture

CHAPTER

FIFTEEN

BODY WAVE TOMOGRAPHY

15.1 The significance of velocity model in seismology

Velocity is the essential physical property of underground objects. It can be used to detect the presence (location and size) of different geological objects underground and can be used to search for oil and metal deposits. Furthermore, velocity models can be used to investigate the evolution of the earth, as well as the location and origin of earthquakes. Body waves and surface waves are two types of seismic waves. For velocity inversion, we can use body waves (P- and S-waves), surface waves, or joint body waves and surface waves. There are many velocity inversion methods, such as one-dimensional model inversion, tomographic velocity inversion, and full waveform inversion. These methods have different computational efficiency and accuracy. The research method to use is determined by the accuracy requirements of the research objectives, the quality of seismic data, and the economic cost. Here we introduce a seismic traveltime tomographic inversion method. It uses first arrivals of body waves and plays an important role in detecting the internal structure of the solid earth.

15.2 Brief introduction of the tomography inversion method-TomoDD

TomoDD is a local-scale (<100 km) seismic tomography software based on ray travel time. It is a gird-based 3-D velocity inversion method that uses absolute and relative arrival times to invert both the velocity structure and the event location. The relative arrival time is the differential arrival time of the event pair. Relative arrival times can reduce systematic errors and thus generate more accurate velocity models than standard tomography methods that only use absolute arrival times. TomoDD is an optimization inversion method in which the LSQR algorithm (Paige et al., 1982) is used to minimize the residual between forward data and observed data, so as to obtain the final velocity model. The forward travel time calculation is computed by the pseudo-bending ray tracing algorithm (Koketsu and Sekine, 1998).

Note:

Professor Haijiang Zhang of the University of Science and Technology of China and Professor Clifford H. Thurber of the University of Wisconsin-Madison collaborated on the software. Thanks to Prof. Haijiang Zhang and his team members for their sharing of `TomoDD` codes and tutorials. `TomoDD` is modified from `hypoDD` (Waldhauser 2001), so it is recommended to learn `hypoDD` before you learn `TomoDD`.

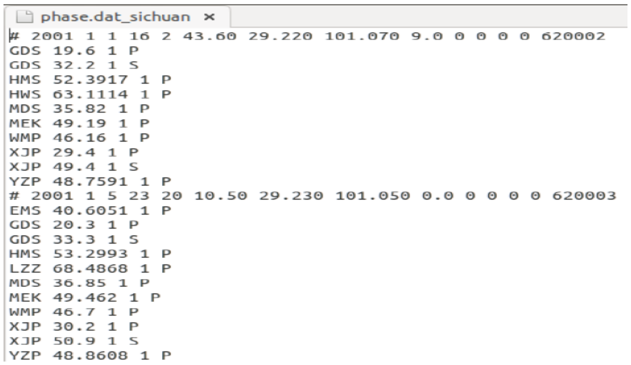
15.2.1 An introduction to the use of TomoDD software

The use of TomoDD software mainly consists of three steps: 1) Preparation of input data; 2) Modify parameter files and perform field data inversion; 3) Resolution test of inversion results. Here we give a practical data processing example to illustrate the use of the TomoDD software. Download the TomoDD example package here. After unzipping, we can see that the package contains four folders. The Doc folder stores the input/output parameters and file format description files of TomoDD, as well as two pieces of literatures related to TomoDD. The folder Example shows an actual velocity inversion case. The Script folder shows some MATLAB drawing scripts, and the text file named illustration shows how to use them. The folder Src contains TomoDD's source code as well as the code for calculating differential arrival times. The platform and software needed to run the program: ubuntu, gfortran, matlab.

1 preparation of input data

1.1 Prepare the seismic phase file (phase.dat) and station file (station.dat)

1.1.1 Seismic phase file: the format of this file is shown below (as shown in Example/ph2dt)



```
# 2001 1 1 16 2 43.60 29.220 101.070 9.0 0 0 0 0 620002
GDS 19.6 1 P
GDS 32.2 1 S
HMS 52.3917 1 P
HWS 63.1114 1 P
MDS 35.82 1 P
MEK 49.19 1 P
WMP 46.16 1 P
XJP 29.4 1 P
XJP 49.4 1 S
YZP 48.7591 1 P
# 2001 1 5 23 20 10.50 29.230 101.050 0.0 0 0 0 0 620003
EMS 40.6051 1 P
GDS 20.3 1 P
GDS 33.3 1 S
HMS 53.2993 1 P
LZZ 68.4868 1 P
MDS 36.85 1 P
MEK 49.462 1 P
WMP 46.7 1 P
XJP 30.2 1 P
XJP 50.9 1 S
YZP 48.8608 1 P
```

Figure 1. The format of the seismic phase file. Lines that begin with # represent event information, while lines that do not begin with # represent the station that received the event.

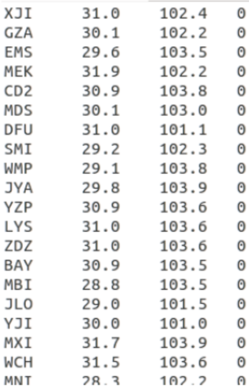
The meaning of the event and the station line is:

```
# Year   Month   day   hour   minute   second   latitude longitude   depth (km)   EH   EV   RMS   Event ID
Station name   travel time   data quality   seismic phase
......
```

Note:

The data quality value is generally between 0 and 3. If it is greater than 3, the data will be removed.

1.1.2 Seismic phase file: the format of this file is shown below (as shown in `Example/ph2dt`)



```
XJI     31.0    102.4   0
GZA     30.1    102.2   0
EMS     29.6    103.5   0
MEK     31.9    102.2   0
CD2     30.9    103.8   0
MDS     30.1    103.0   0
DFU     31.0    101.1   0
SMI     29.2    102.3   0
WMP     29.1    103.8   0
JYA     29.8    103.9   0
YZP     30.9    103.6   0
LYS     31.0    103.6   0
ZDZ     31.0    103.6   0
BAY     30.9    103.5   0
MBI     28.8    103.5   0
JLO     29.0    101.5   0
YJI     30.0    101.0   0
MXI     31.7    103.9   0
WCH     31.5    103.6   0
MNT     28.3    102.2   0
```

Figure 2. The format of the seismic phase file

The meaning of each line is:

```
Station name    Latitude    longitude elevation (m)
......
```

1.2 Calculating absolute arrival times (`absolute.dat`), differential arrival times of event pair (`dt.ct`) and event information (`event.dat`).

1.2.1 Compile program (input: `ph2dt.inc`; output: `ph2dtN3`)

a. Change the path to `Src/ph2dt/include` and change the three parameters in `ph2dt.inc`: MEV, MSTA, MOBS.

b. Change the path to `Src/ph2dt` and use the following two command: `make clean`; `make`. Then we can obtain the executable file named `ph2dtN3`. `make clean` means deleting the files generated by the previous compilation, `make` means compiling the code and generating executable files.

Note:

A detailed illustration for this subprogram is provided in Waldhauser, F. (2001). HypoDD A program to compute double difference Hypocenter Locations, pages 4-6 and 15-16.

1.2.2 Execute the generated compiler to calculate differential times (input: `ph2dt.inp`; output: `absolute.dat`, `dt.ct`, `event.dat`)

Copy the executable file `ph2dtN3` to the folder `Example/ph2dt` and edit the parameter file `ph2dt.inp` in that folder. Then, enter `./ph2dtN3 ph2dt.inp` in the command line, and click enter to complete.

1.2.3 Convert the resulting `event.dat` file to the data format required by `TomoDD`, as shown below

```
awk -f addEve0.awk event.dat > event.dat
```

At this point, we have all the data we need for `TomoDD` software, which are `absolute.dat`, `dt.ct`, `event.dat`, `station.dat`.

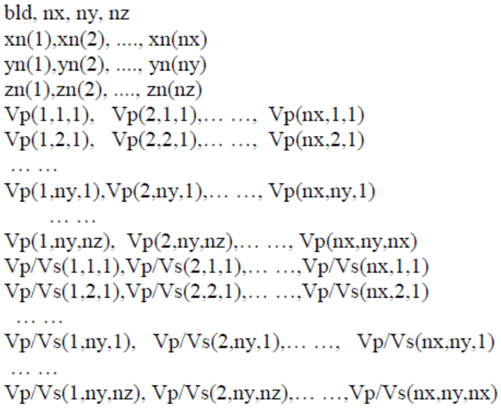
2 Inversion example

2.1 Preparation of the initial velocity model

1 The initial velocity model may be given a transversely homogeneous gradient model increasing by depth.

2 In addition, the initial velocity model can also be obtained by using the `VELEST` (Kissling et al., 1994) program.

The format of the initial velocity model is as follows



```
bld, nx, ny, nz
xn(1),xn(2), ....., xn(nx)
yn(1),yn(2), ....., yn(ny)
zn(1),zn(2), ....., zn(nz)
Vp(1,1,1),  Vp(2,1,1),… …,  Vp(nx,1,1)
Vp(1,2,1),  Vp(2,2,1),… …,  Vp(nx,2,1)
 … …
Vp(1,ny,1),Vp(2,ny,1),… …, Vp(nx,ny,1)
        … …
Vp(1,ny,nz),  Vp(2,ny,nz),… …, Vp(nx,ny,nx)
Vp/Vs(1,1,1),Vp/Vs(2,1,1),… …,Vp/Vs(nx,1,1)
Vp/Vs(1,2,1),Vp/Vs(2,2,1),… …,Vp/Vs(nx,2,1)
  … …
Vp/Vs(1,ny,1),  Vp/Vs(2,ny,1),… …,  Vp/Vs(nx,ny,1)
 … …
Vp/Vs(1,ny,nz), Vp/Vs(2,ny,nz),… …,Vp/Vs(nx,ny,nx)
```

Figure 3. The format of the initial velocity model. `bld` is 1 or 0.1 or 0.01, which depends on the minimum precision of the grid points in the x, y, and z directions. `nx`, `ny`, and `nz` are the number of grid nodes in x, y and z directions. `xn(1)`, `xn(nx)`, `yn(1)`, `yn(ny)`, `zn(1)` and `zn(nz)` are boundary nodes that must be large enough to hold all the events and stations.

2.2 TomoDD source code compilation: `tomoDD-SE`

2.2.1 Allocate the memory required by the program by modifying the following parameter file

a. Under the folder `Src/tomoDD-SE/include`, modify the following parameters of `RaySPDR.inc`:

maxnx—maximum number of nodes in x direction

maxny— maximum number of nodes in y direction

maxnz— maximum number of nodes in z direction

mxpari— maximum number of parameters to invert for.

maxpar— maximum number of potential parameters that could be included in the inversion. For `tomoDD`, mxpari is equal to maxpar. Both of them should be at least equal to iuses*(maxnz-2)*(maxny-2)*(maxnz-2).

b. Under the folder `Src/tomoDD-SE/include`, modify the following parameters of `tomoFDD.inc`:

MAXEVE: maximum number of events used in the inversion.

MAXSTA: maximum number of stations used in the inversion.

MAXDATA: maximum number of phase data including both absolute and differential data.

MAXNODE: maximum number of inversion nodes for each ray to sample (~4*MAXNZ).

MAXND: it is used to control the maximum number of nonzero slowness partial derivatives (MAXND*MAXDATA) (<4*MAXNZ).

2.2.2 Compile program

In the folder `Src/tomoDD-SE`, run `make clean` and `make` in sequence to obtain the executable file `tomoDD-SE` for inversion.

Note:

For 32-bit computers: use `Makefile.32` & `Makefile.32_syn`; Terminal operation: `make clean -f Makefile.32; make -f Makefile.32` and `make clean -f Makefile.32_syn; make -f Makefile.32_syn`

For 64-bit computers: use `Makefile` & `Makefile_syn`; Terminal operation: `make clean; make` and `make clean -f Makefile_syn; make -f Makefile_syn`

`Makefile.32_syn` and `Makefile_syn` are used to generate the forward simulation executable, `Makefile.32` and `Makefile` are used to generate velocity inversion executables.

It requires downloading an older version of the gfortran compiler, such as `gfortran 4.7` or lower, updating the compiler name in the Makefile (line 5) to the newly downloaded version, and then recompiling.

2.3 Preparation of parameter file: `tomoDD-SE.inp`

This file is shown in `Example/RealData-Inversion`. Change the inversion parameters in the file according to the filed data, and the meaning of the parameters is clearly explained in the parameter file.

For parameter adjusting, we need to emphasize the following points:

a. The absolute data is given a large weight (WTDD) to invert the overall velocity structure; In order to better constrain the source region structure, more weight (WTCTP, WTCTS) is given to the differential arrival times data.

b. Since the convergence of velocity in joint inversion is faster than that of seismic location, it is suggested to add a location-only inversion (JOINT = 0) after each joint inversion.

c. How to select WRCT, WRCC, WDCT and WDCC in each group of iteration parameters?

   WRCT, WRCC: The program will weight each data according to its residual. That is, the larger the residual of a data, the smaller its weight, or even 0. WDCT, WDCC: The program will weight the data according to the distance between earthquakes. That is, the greater the distance between earthquakes, the smaller the weight will be. After the distance exceeds the distance, the weight will be 0. None of these parameters can be used (set to -9). Especially for the joint inversion of velocity and position, WDCT and WDCC may not be used. It is generally recommended to use WDCT and WDCC unless the data is ideal and the values are gradually reduced. If the data quality is very high, you are advised to set WDCT and WDCC to about 3.

d. The regularization parameters smooth and damp are selected using the L-curve method. For the L-curve method, please refer to Sections 5.1 and 5.2 of Parameter Estimation and Inverse Problems (Richard C. Aster et al. 2005). In `Example/RealData-Inversion-Lcurve`, the procedure for choosing smooth using L-Curve is shown. The selection process for damp is the same as for smooth.

2.4 Prepare the folder for input data: `Input_Files`

As shown in `Example\RealData-Inversion\Input_Files`, place the prepared files (`absolute.dat`, `dt.ct`, `event.dat`, `station.dat`) into it.

2.5 Prepare the folder for output data: `Output_Files`

The velocity model and source location information generated by inversion procedure are placed in folder `Example\RealData-Inversion\Output_Files`. In `Output_Files`, matlab programs in this folder can be used to draw the earthquake location and velocity model in three directions of the slice map.

2.6 Perform inversion

After the above five steps are completed, copy the executable files to the folder `Example/RealData-Inversion` and execute `./tomoDD SE tomoDD SE.inp` in the terminal to carry out velocity inversion. Note that `ak135.15.SKS`, `layer-16.dat` are files that are prepared for the `TomoDD` executable file. We don’t need to change it.

2.7 Display of inversion results

The figure below shows the velocity model of inversion, which is drawn by the matlab script under the folder `Example\RealData-Inversion\Output_Files`.

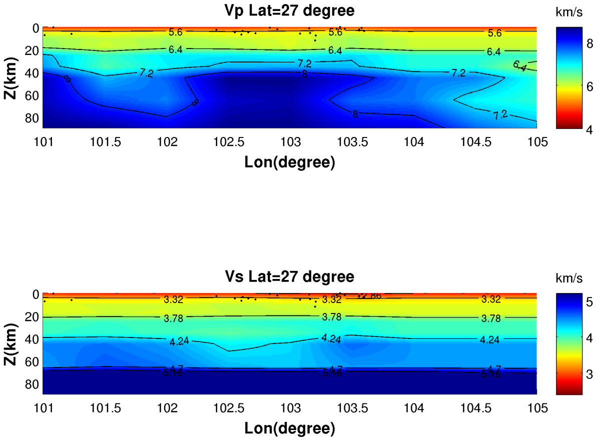
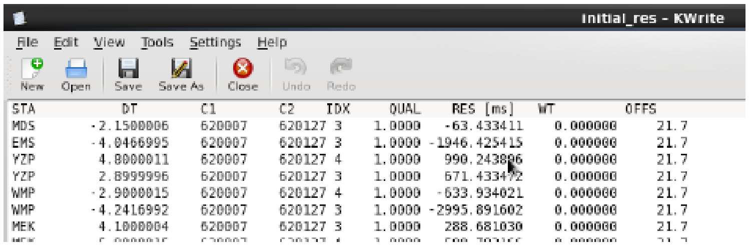


Figure 4. Inversion results

2.8 Data residuals

The initial residual file is `tomoDD-SE.res.ini`, as shown below. Its default output is in the directory where the program is run.



```
STA          DT          C1       C2      IDX      QUAL      RES [ms]    WT           OFFS
MDS       -2.1500006    620007   620127 3      1.0000     -63.433411    0.000000      21.7
EMS       -4.0466995    620007   620127 3      1.0000   -1946.425415    0.000000      21.7
YZP        4.8000011    620007   620127 4      1.0000     990.243896    0.000000      21.7
YZP        2.8999996    620007   620127 3      1.0000     671.433472    0.000000      21.7
WMP       -2.9000015    620007   620127 4      1.0000    -633.934021    0.000000      21.7
WMP       -4.2416992    620007   620127 3      1.0000   -2995.891602    0.000000      21.7
MEK        4.1000004    620007   620127 3      1.0000     288.681030    0.000000      21.7
```

Figure 5. The initial residual file. `IDX` represents data type (1 denotes P-wave cross-correlation data; 2 denotes S-wave cross-correlation data; 3 denotes P-wave directory data; 4 denotes S-wave directory data), `RES` represents residual (ms).

`tomoDD.res` represents the travel time residual of the data after final inversion. It can be found in the folder of `Example/RealData-Inversion/Output_Files` and is in the same format as the `tomoDD-SE.res.ini`. We can make use of the above two data files to plot the changes in residual errors of travel time before and after inversion. See, for example, the following figure:

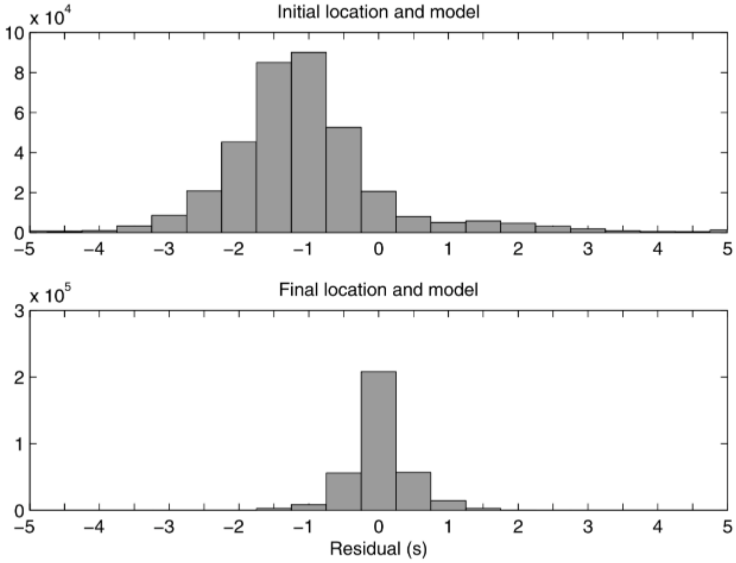


Figure 6. Residual misfit improvement.

Note:

All parameters and file formats used in this tutorial are defined in `Doc/tomoDD_manual` and can be queried if necessary

3 Resolution test of inversion results

We need to know the reliability of the obtained velocity model, as well as its horizontal and vertical resolution, in order to conduct further research on seismogenesis interpretation. We can use the ray distribution density diagram and the checkerboard test to study the resolution of the velocity model.

3.1 Ray distribution density

The script `extract_dws.awk` in the folder `Example\RealData-Inversion\Output_Files` can be used to extract the ray density (DWS_P, DWS_S) of each grid point from `tomoDD.vel`. By doing `awk -f extract_DWS.awk tomoDD.vel` at the terminal, we can obtain `DWS_P` and `DWS_S`. The data format of `DWS_P` and `DWS_S` is the same as `Vp_model.dat` and `Vs_model.dat`

3.2 Synthetic resolution test (checkerboard test)

The checkerboard test can be explained as follows: first, the checkerboard velocity model with different grid sizes is constructed, and then the theoretical observation record is calculated by using the constructed velocity model and the same recording geometry as in the field. Then, the inversion method is used for the simulated observation record. By comparing the inversion model with the real checkerboard model, the region with the better inversion of checkerboard velocity and size is the region with higher resolution. An example is shown in the folder `Example/checkerboard`. As follows the specific implementation process:

3.2.1 Forward modeling and inversion code compilation

Previously, we compiled the executable `tomoDD-SE` for inversion, so we only need to compile the additional executable `tomoDD-SE_syn` for forward modeling. Changeing the folder to `Src/tomoDD-SE`. By executing `make clean -f Makefile_syn`, `make -f Makefile_syn` in sequence on the terminal, we can obtain the executable `tomoDD-SE_syn`.

3.2.2 Generate synthetic data

Copy the generated forward and inverse executable files to folder `Example/checkerboard`. Use the checkerboard velocity model and the actual recording geometry to simulate the travel time of the first-arrival (the generated data is put in the folder `Syn`).

3.2.3 Inversion velocity model

Velocity inversion is carried out by using the calculated travel time and the initial model (the inversion results are placed in the folder `Vel`). In this example, `checkerboard.pl` integrates accurate checkerboard velocity model construction, travel time forward calculation, and velocity inversion. So simply execute the `checkerboard.pl` in the folder `Example/checkerboard` to implement checkerboard testing.

3.2.4 Display of inversion results

The real checkerboard model and the inverted checkerboard model are shown below.

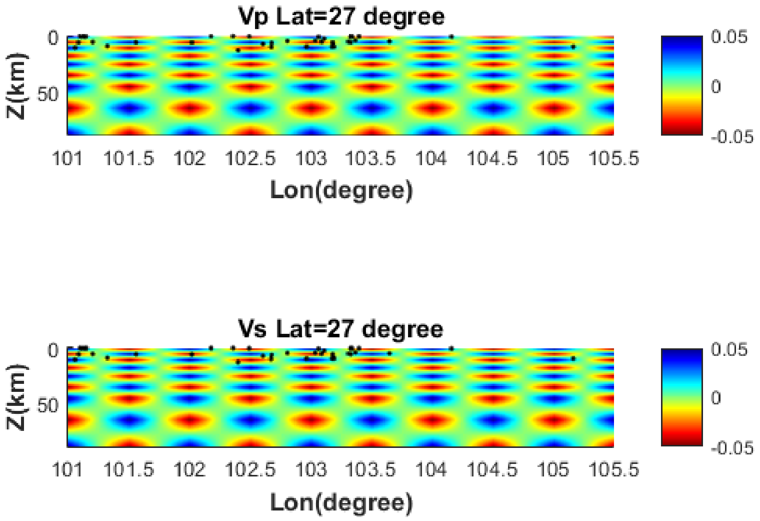


Figure 7. A real checkerboard model

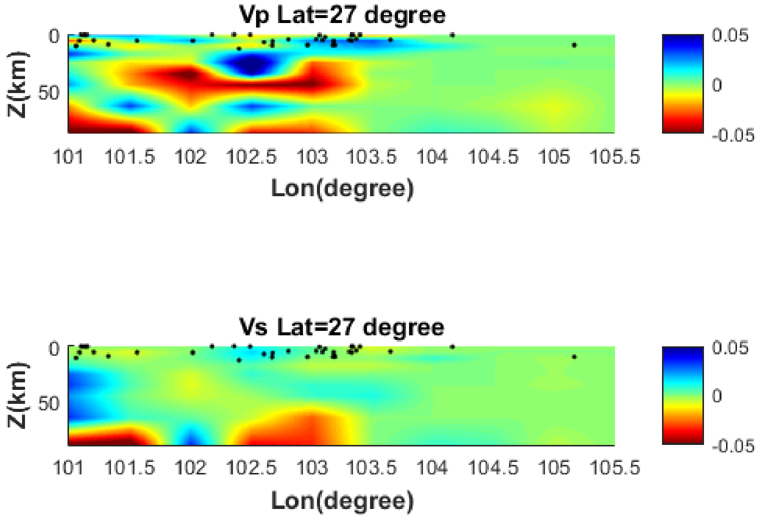


Figure 8. The inverted checkerboard model

By comparison with Figures. 7 and 8, the inversion accuracy of field data is also higher in the region where velocity value and checkerboard size are better recovered.

15.3 Reference:

Paige, C. C. and Saunders, M. A., 1982, LSQR: Sparse Linear Equations and Least Squares Problems. ACM Transactions on Mathematical Software, 8 (2), 195–209.

Kissling, E., Ellsworth, W. L., Eberhart-Phillips, D. and Kradolfer, U., 1994, Initial reference models in local earthquake tomography. Journal of Geophysical Research: Solid Earth, 99, 19635–19646.

Koketsu, K. & Sekine, S., 1998, Pseudo-bending method for three-dimensional seismic ray tracing in a spherical earth with discontinuities. Geophysical Journal International 132, 339–346.

Waldhauser, F., 2001, hypoDD–A Program to Compute Double-Difference Hypocenter Locations (hypoDDversion 1.0-03/2001).US Geol. Surv. Open-File Rept. 01,113.

Zhang, H. J. and Thurber, C. H., 2003, Double-Difference Tomography: The Method and Its Application to the Hayward Fault, California: Bulletin of the Seismological Society of America, 93 (5), 1875–1889.

Richard C. Aster, Brian Borchers, Clifford H. Thurber, 2005, Parameter Estimation and Inverse Problems.

Zhang, H. J. and Thurber, C. H., 2006, Development and Applications of Double-difference Seismic Tomography: Pure and Applied Geophysics, 163, 373–403.

CHAPTER

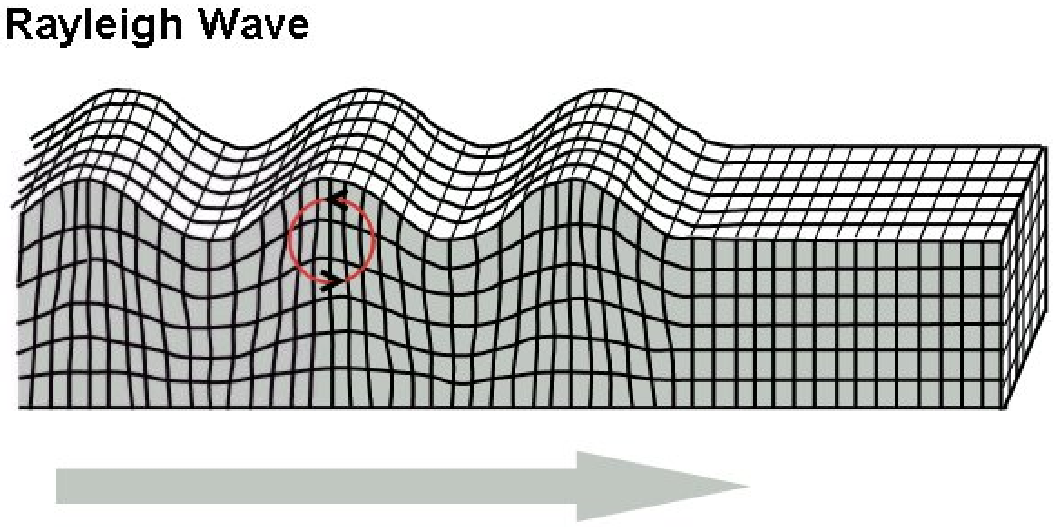
SIXTEEN

SURFACE WAVE TOMOGRAPHY

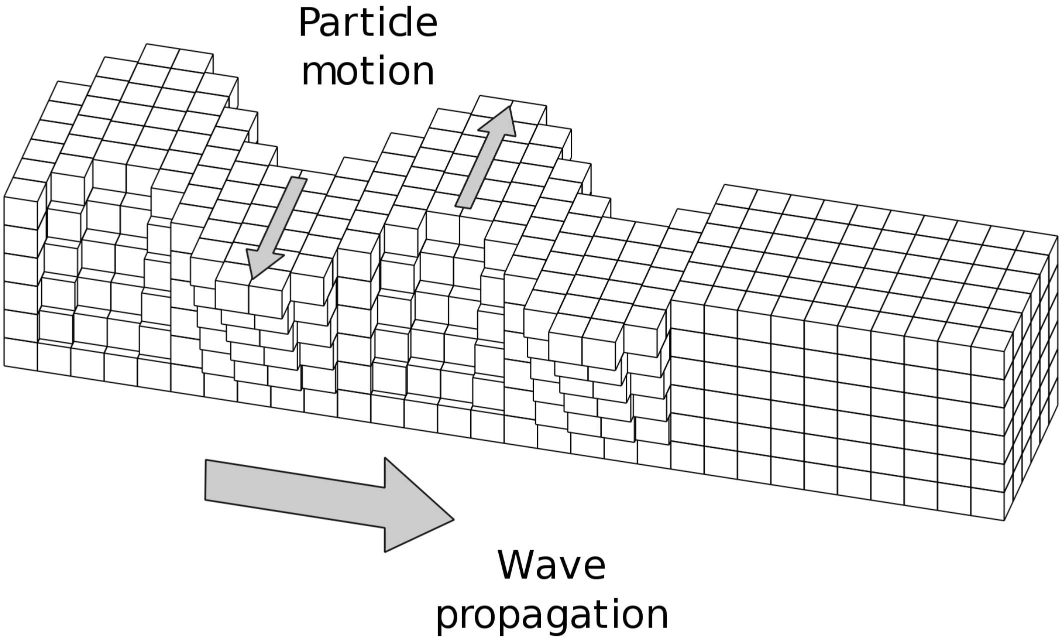
16.1 Introduction to surface waves

Unlike body waves that travel through the earth's interior, surface waves propagate along the earth's surface. They could be generated from earthquakes and travel more slowly that body waves, but correspond to lower-frequency and higher amplitude in the seismogram. There are two most common types of surface waves in Seismology - Rayleigh waves and Love waves, which are named after their discoverers Lord Rayleigh and August Love.

Rayleigh waves (or so called ground roll) move both longitudinally and horizontally and result in elliptical movement, similar to the rolling water waves but against the direction of propagation.



Love waves move horizontally and at the right angles to the direction of propagation, but cannot propagate in water like S-waves. The shear movement of the ground caused by Love waves is more damaging to the man-made structures.



16.2 Surface waves observations

Here we provide example waveforms showing the surface waves generated from `2019 M7.1 Ridgecrest earthquake` recorded by `IU.HRV station located at Adam Dziewonski Observatory (Oak Ridge), Massachusetts, USA`.

For later use, please first download the `waveform data` of miniSEED format into your local computer. Also make sure that ObsPy is also installed since it's used for reading and processing waveforms.

1. Import necessary modules or functions

```
# from obspy modules import functions
from obspy import read, UTCDateTime
from obspy.geodetics.base import gps2dist_azimuth
from obspy.core.stream import Stream
# import matplotlib module
import matplotlib.pyplot as plt
```

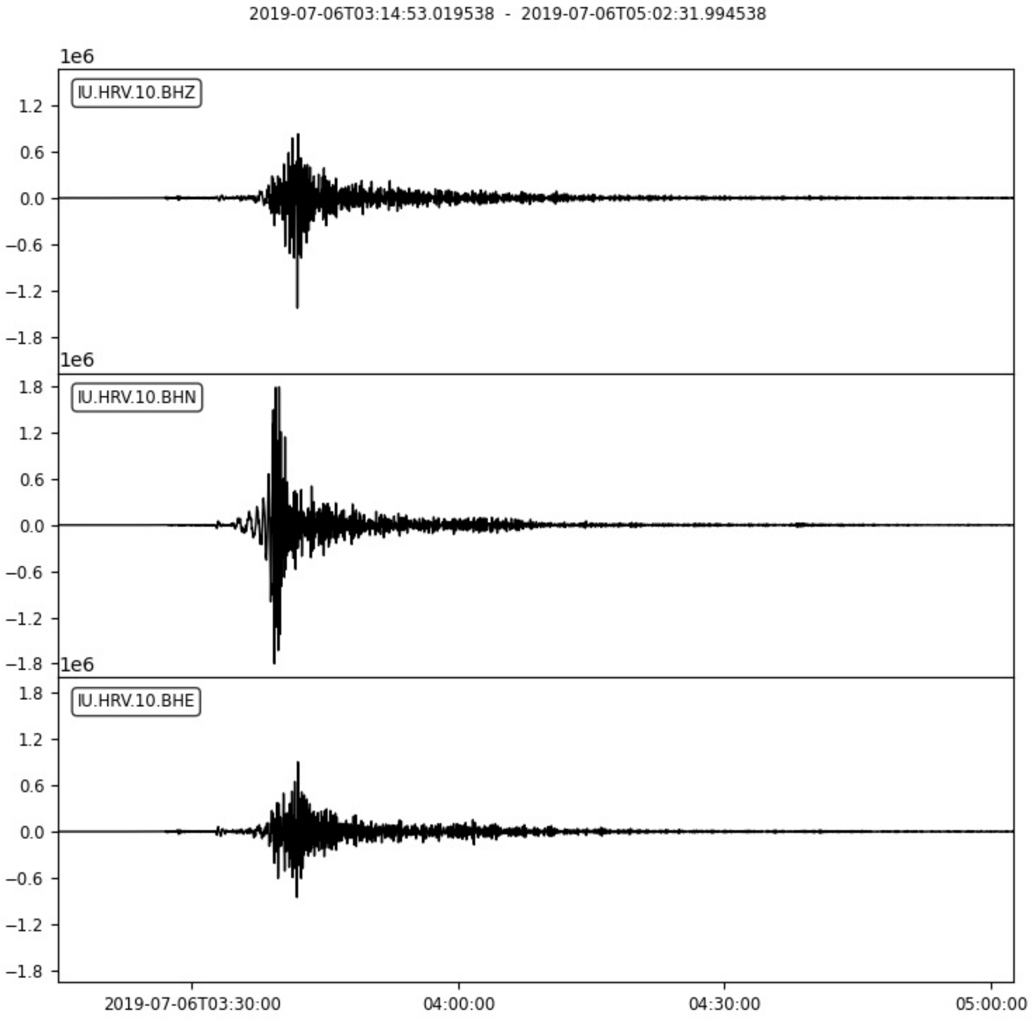
2. Read the waveform data and only keep waveforms of location `10`

```
# read the downloaded waveforms into obspy stream object
st = read('./2019-07-06-mw71-central-california-2.miniseed')
# select traces whose location is '10'
# use copy function to aviod processing original stream
st_10 = st.select(location='10').copy()
# from the IRIS webpage about this instrument
# channel 'BH1' azimuth and dip are 0 and 0
# channel 'BH2' azimuth and dip are 90 and 0
# thus, we rename 'BH1' into 'BHN'
```

(continues on next page)

(continued from previous page)

```
# and rename 'BH2' into 'BHE'
# this is useful for waveform rotation
for tr in st_10:
    if tr.stats.channel == 'BH1':
        tr.stats.channel = 'BHN'
    if tr.stats.channel == 'BH2':
        tr.stats.channel = 'BHE'
# quick check
st_10.plot()
```



3. Rotate waveforms from ZNE coordinate system into ZRT coordinate system

```
# event info
event_otime = UTCDateTime('2019-07-06 03:19:53')
event_longitude = -117.599333
event_latitude = 35.7695
event_depth = 8 # kilometer
event_magnitude = 7.1
```

(continues on next page)

(continued from previous page)

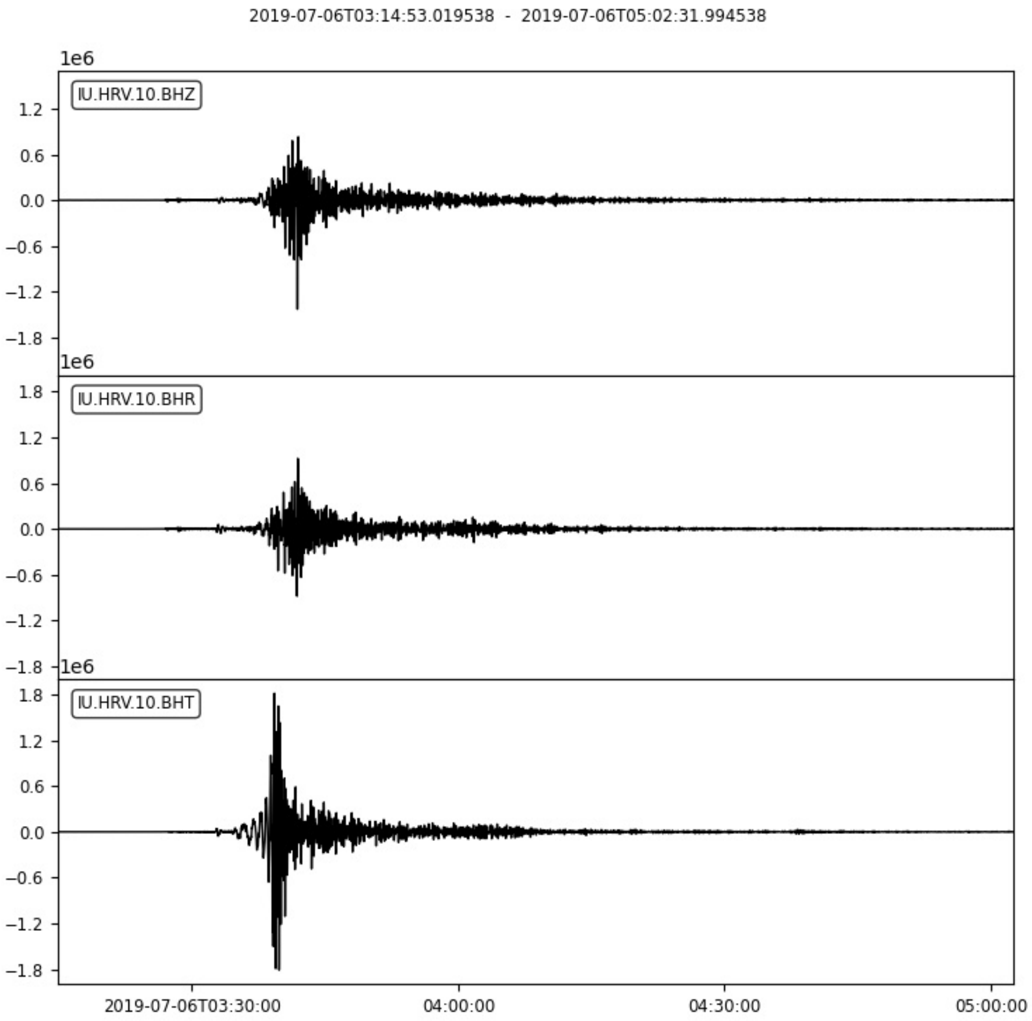
```
# station info
station_longitude = -71.5583
station_latitude = 42.5064
station_elevation = 200.0 # meter
station_depth = 0.0
```

Compute necessary geometric parameters.

```
# compute epicentral distance, azimuth, and back azimuth
# for this earthquake-station pair
dist_in_m, az, baz = gps2dist_azimuth(event_latitude, event_longitude, station_latitude,␣
↪station_longitude)
print('epicentral distance is: %.2f m'%dist_in_m)
print('azimuth is: %.2f'%az)
print('back azimuth is: %.2f'%baz)
```

```
epicentral distance is: 3999969.28 m

azimuth is: 64.89

back azimuth is: 274.97
```

```
# use the rotate function to rotate waveforms
# from ZNE coordinate system to ZRT coordinate system
st_10.rotate('NE->RT', back_azimuth=baz)
# quick check
# Note that Love wave in the transverse component
# Rayleigh wave in the radial and vertical components
st_10.plot()
```



Clearly, We can see that surface waves arrive after the body waves and have much larger amplitude.

16.3 Phase shift of Rayleigh waves

Because Rayleigh waves correspond to elliptical movement in a vertical plane pointed in the direction in which the waves are travelling, thus the ground motions in radial and vertical directions should have phase shift.

Here we can apply a narrow bandpass filter to the waveforms and then plot them together to check the abovementioned phase shift phenomenon.

```
# first select vertical and radial components for analysis
st_BHRZ = st_10.select(component='[RZ]').copy()
# remove mean, linear trend, and taper at both ends
st_BHRZ.detrend('demean').detrend('linear').taper(0.05)
# apply a narrow bandpass filter to the raw waveform
# in this example: 8 s-12 s
st_BHRZ.filter('bandpass', freqmin=1/12., freqmax=1/8., corners=2, zerophase=True)
# further zoom into the Rayleigh wave part
```

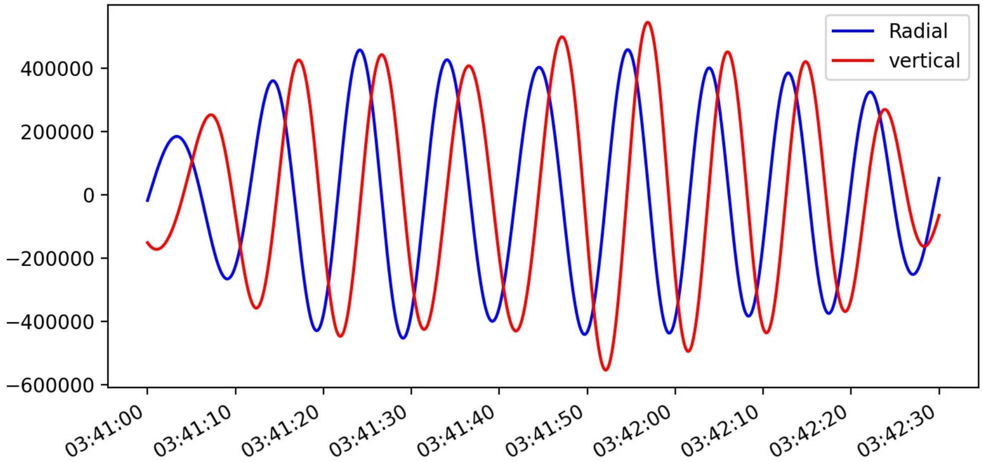
(continues on next page)

(continued from previous page)

```
st_ray = st_BHRZ.slice(starttime=UTCDateTime('2019-07-06 03:41:00'), endtime=UTCDateTime(
↪'2019-07-06 03:42:30'))
```

Note: **Exercise**: try to change starttime and endtime to see if phase delay exists in other time windows.

```
# plot both traces together
# can you observe the phase shift?
fig = plt.figure(figsize=(8, 4), dpi=200)
ax = fig.add_subplot(1, 1, 1)
ax.plot(st_ray[0].times("matplotlib"), st_ray[0].data, "b-", label='Radial')
ax.plot(st_ray[1].times("matplotlib"), st_ray[1].data, "r-", label='vertical')
ax.xaxis_date()
fig.autofmt_xdate()
plt.legend(loc='upper right')
plt.show()
```



16.4 Dispersion of surface waves

Since longer period surface waves would penetrate deeper into higher speed layer, they will travel faster. As a result, velocities at a number of periods are different, this is so called surface wave dispersion.

For the sake of simplicity, here we only show the dispersion analysis of Love wave by applying few bandpass filters and measure the corresponding group velocity dispersion curve.

1. select transverse component waveform and set several period bands

```
# now only select transverse component for further analysis
tr = st_10.select(component='T')[0]
# now choose a set of period bands
# and their units are in second
period_bands = [[10, 20],
```

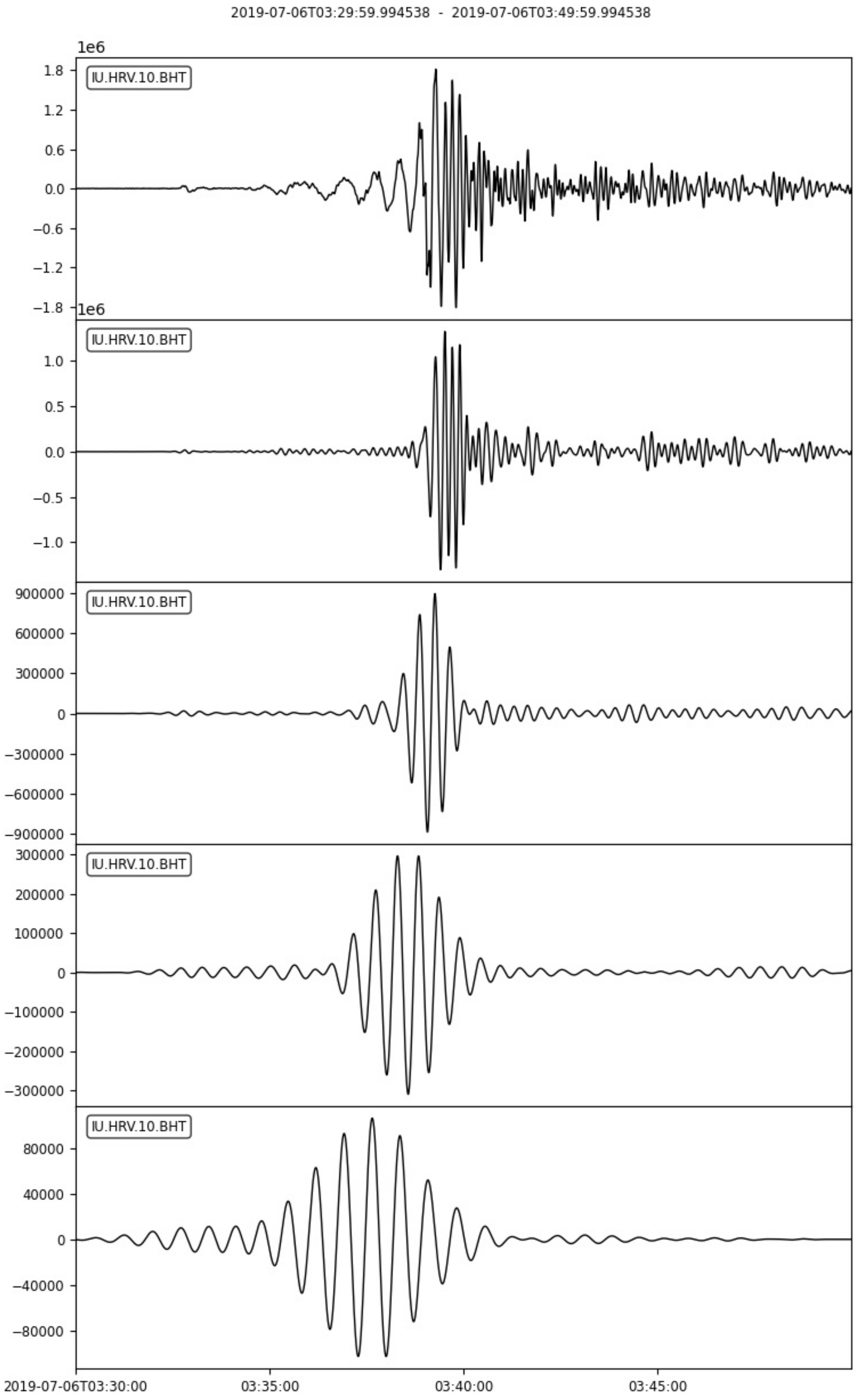
(continues on next page)

(continued from previous page)

```
                [20, 30],
                [30, 40],
                [40, 50]]
```

2. save raw waveform into a stram and append filtered waveforms

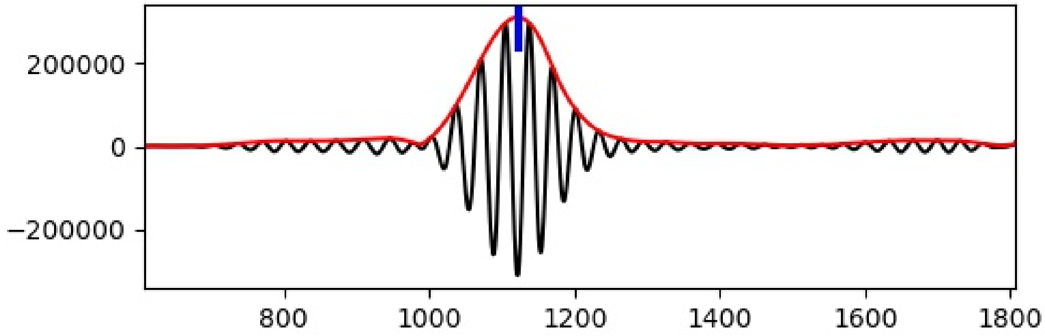
```
# first create an empty stream
st_BHT = Stream()
# the first trace is raw waveform without filtering
st_BHT += tr
# similar to the above loop except the last plot line is changed
for period_band in period_bands:
    freq_high, freq_low = 1./period_band[0], 1./period_band[1]
    tr_copy = tr.copy()
    tr_copy.detrend('demean').detrend('linear').taper(0.05)
    tr_copy.filter('bandpass', freqmin=freq_low, freqmax=freq_high, corners=2,␣
↪zerophase=True)
    st_BHT += tr_copy
# then let's zoom-in
st_BHT.trim(starttime=UTCDateTime('2019-07-06 03:30:00'), endtime=UTCDateTime('2019-07-
↪06 03:50:00'))
# quick plot with some newly arguments specified to make sure all traces share the same␣
↪x-axis
st_BHT.plot(equal_scale=False, automerge=False)
```


2019-07-06T03:29:59.994538 - 2019-07-06T03:49:59.994538
1e6
1.8
1.2
0.6
0.0
−0.6
−1.2
−1.8
IU.HRV.10.BHT
1e6
1.0
0.5
0.0
−0.5
−1.0
IU.HRV.10.BHT
900000
600000
300000
0
−300000
−600000
−900000
IU.HRV.10.BHT
300000
200000
100000
0
−100000
−200000
−300000
IU.HRV.10.BHT
80000
40000
0
−40000
−80000
IU.HRV.10.BHT
2019-07-06T03:30:00
03:35:00
03:40:00
03:45:00

3. measure group velocities at these periods

```
# 1. take 30 s to 40 s bandpass filtered waveform as an example
data3040 = st_BHT[3].data.copy()
# ensure event original time corresponds to 0 s
time3040 = st_BHT[3].times() + (st_BHT[3].stats.starttime - event_otime)
# 2. perform Hilbert transform to get envelope (unnecessary to know details)
envelope = np.abs(hilbert(data3040))
# 3. "measure the time where largest amplitude observed" - here for simplicity of␣
↪analysis
indexmax = np.argmax(envelope)
travtime = time3040[indexmax]
# 4. compute group velocity by dividing distance by travel time
velocity = dist_in_m * 0.001 / travtime
# 5. print results
print('The distance is %.4f km, and travel time is %.2f s'%(dist_in_m * 0.001,␣
↪travtime))
print('Thus, the corresponding group velocity is %.4f km/s'%velocity)
# simple plot using pyplot
plt.figure(figsize=(6, 2), dpi=100)
plt.plot(time3040, data3040, 'k-')
plt.plot(time3040, envelope, 'r-')
plt.axvline(travtime, 0.85, 1, color='b', lw=3)
plt.xlim(time3040[0], time3040[-1])
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.show();
```

```
The distance is 3999.9693 km, and travel time is 1121.84 s

Thus, the corresponding group velocity is 3.5655 km/s
```



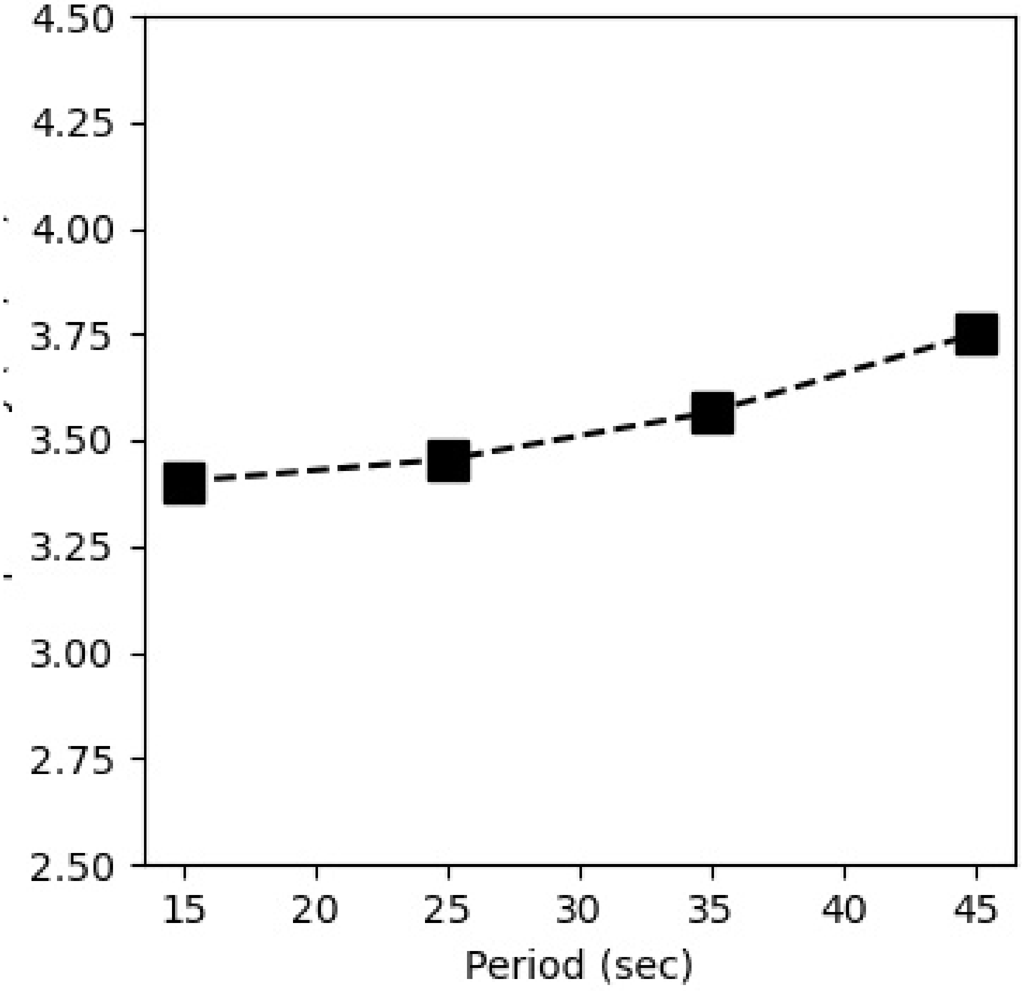
4. repeat the above process for other period bands

After changing to different period bands and re-run the step 3, we can get the following results:

period	traveltime	velocity
10-20s	1175.52s	3.4027km/s
20-30s	1157.97s	3.4543km/s
30-40s	1121.84s	3.5655km/s
40-50s	1066.54s	3.7504km/s

5. plot the results

```
# Now if we let each center period represent the period band
center_periods = [15, 25, 35, 45]
group_velocities = [3.4027, 3.4543, 3.5655, 3.7504]
# then show the period-velocity curve (i.e., dispersion curve)
plt.figure(figsize=(4, 4), dpi=100)
plt.plot(center_periods, group_velocities, 'ks--', markersize=10)
plt.xlabel('Period (sec)')
plt.ylabel('Group velocity (km/sec)')
plt.ylim(2.5, 4.5)
plt.show();
```



Above all, this tutorial only gives a simple introduction to the ideas of surface waves and their dispersion, through which you may have a better understanding of surface waves.

Note: **Exercise**: Search a teleseismic earthquake that can generate clear surface waves (http://ds.iris.edu/wilber3/find_event) and finish the following tasks

1. Make a map view plot to show the position of the earthquake and the seismic station you chose. (refer to the GMT tutorial).
2. Request the seismograms at the station by Obspy (refer to the ObsPy tutorial).
3. Perform phase delay and dispersion analysis on the requested waveforms.

16.5 Compute theoretical dispersion curve

16.6 Invert observed dispersion curve for 1D velocity

CHAPTER SEVENTEEN

AMBIENT NOISE CROSS CORRELATION

In this tutorial, we will introduce why and how to compute cross-correlation functions (CCFs) from seismic ambient noise data.

17.1 A Brief Introdcution of Ambient Noise Cross-correlation

17.1.1 1.1 What is Ambient Noise?

In seismology, the term **noise** refers to ground motion which is recorded in the absence of an identifiable source of seismic energy, such as an earthquake. In particular, seismic noise is the weak, low-frequency seismic signal originating from randomly distributed sources. The continuous interaction of low frequency (<1Hz) oceanic surface waves (swells) with the Earth's crust is the main cause of natural ambient noise and is observed globally (Aki and Richards).

It is often discarded and cannot be used for geophysical applications, but not always. Seismic noise can be utilized for earth imaging in the border sense, i.e., structural analysis of crust, global tomography, and monitoring velocity changes along faults and volcanic regions. For instance, the cross-correlation of seismic noise reflects the medium properties along the prorogation path.

Note: This process is known as Seismic noise interferometry (SNI).

17.1.2 1.2 Ambient noise cross correlation

Given two seismometers, u_1 and u_2, placed at some distance apart on the surface, the stations will record ground motion as a function of time: $u_1(t)$ and $u_2(t)$. Over long periods of time, the cross-correlation of ground motions of $u_1(t)$ and $u_2(t)$

$$_{12}() =_1 ()_2(+)$$

yields a band-limited approximation of the elastodynamic Green's function, or impulse response, between u_1 and u_2. We call $C_{12}(\tau)$ the time-domain cross-correlation function.

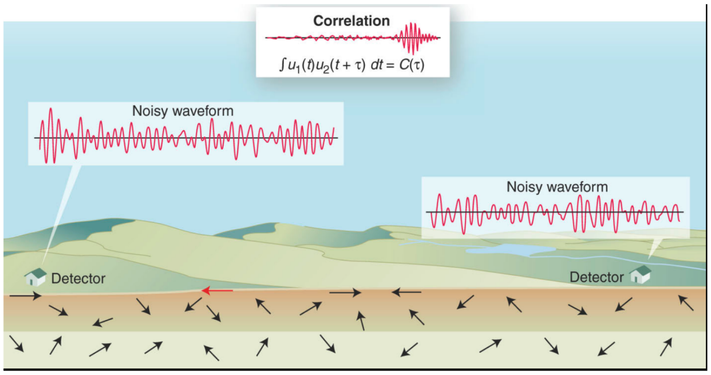


Figure 01: Typical setup for Ambient Noise Cross-Correlation

A useful way to understand the role of cross-correlation in SNI is that it highlights the traveltimes of seismic waves. A wavefield which has travelled between two stations will cause a similar signal to be recorded at each, shifted in time: the cross-correlation function (CCF) of the records will therefore contain a peak at a time lag which corresponds to the traveltime of the wavefield between the two stations (Fig.2).

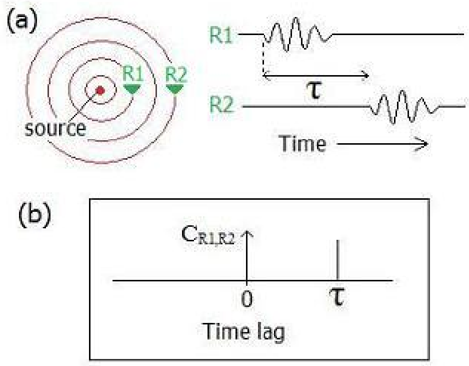


Figure 02: (a) a source of seismic energy (left) is recorded at receivers R1 and R2, causing identical signals separated by a time (right), corresponding to the travel time of the wavefield between the receivers. (b) the cross-correlation function of the two records,on which the traveltime is highlighted as a peak.!

Cross-correlation is performed for positive and negative time lags, so CCFs have both positive ("causal") and negative ("acausal") parts. Oceanic noise arrives from all directions, and the noise records which are cross-correlated during SNI therefore contain energy which has travelled in both directions along an interstation path.

GFs which emerge on noise cross-correlation functions (NCFs) will therefore contain energy in both the causal and acausal parts, symmetric about zero time lag: the causal part representing energy arriving at station B in response to an impulsive source at station A, and the acausal part energy arriving at station A in response to a source at station B (Fig.3). Because GFs are usually assumed to be the same in both directions along a path, the two sides of an NCF can be treated as representing exactly the same information, but reversed in time.

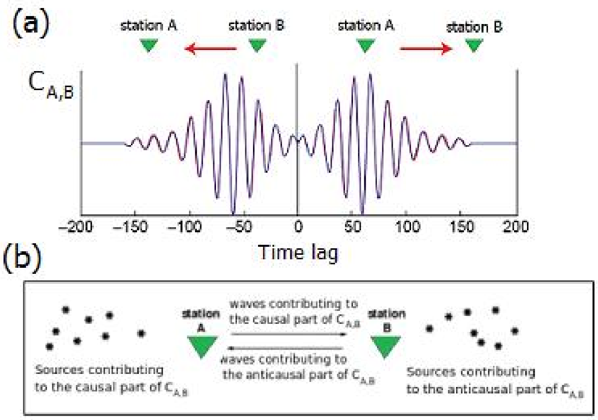

(a)
station A
station B
station A
station B
$C_{A,B}$
−200
−150
−100
−50
0
50
100
150
200
Time lag
(b)
station A
station B
waves contributing to the causal part of $C_{A,B}$
waves contributing to the anticausal part of $C_{A,B}$
Sources contributing to the causal part of $C_{A,B}$
Sources contributing to the anticausal part of $C_{A,B}$

Figure 3.Causal and anticausal parts of an NCF. (a) Idealised NCF for the cross-correlation order station A to station B. The arrows indicate the energy path which is represented by the positive and negative parts of the cross-correlation function. (b) Representation of the noise sources which (theoretically) contribute to the causal and anticausal part of the station A - station B NCF

17.1.3 1.3 Workflow:

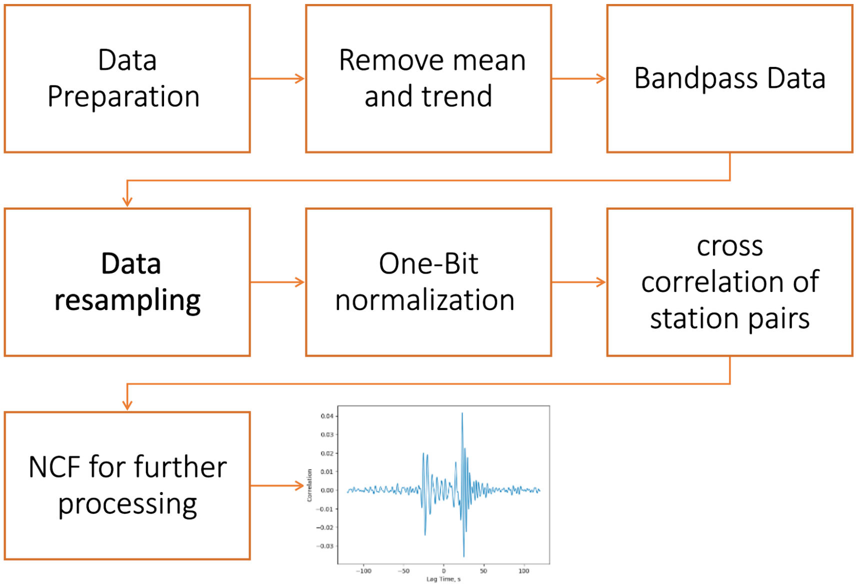


Figure 04: Schematic representation of the data processing scheme

17.1.4 1.4 Data Preparation and inital processing

The basic step in the data processing method is to prepare waveform data from each station separately. The goal of this phase is to accentuate broad-band ambient noise by removing earthquake signals and instrumental irregularities that tend to obscure it.

1. The mean and trend are removed from the data ("rmean" and "rtrend" respectively).
2. The bandpass filtering is applied on data
3. In order to further down-weight the contribution of high-energy arrivals which could obscure the lower amplitude ambient noise signal, a "one-bit" normalisation is applied after various authors (e.g. Stehly et. al. (2007))
4. Do cross correlation between station pairs

17.2 Tutorial

17.2.1 2.1 Import useful packages from Python library

```
import obspy
import numpy as np
from obspy import Stream
from obspy.geodetics import gps2dist_azimuth
import matplotlib.pyplot as plt
```

17.2.2 2.2 Stations information

```
sta1_lon = 104.462283; sta1_lat = 29.409383
sta2_lon = 104.483083; sta2_lat = 29.478317
dist_m,_,_ = gps2dist_azimuth(sta1_lat,sta1_lon,sta2_lat,sta2_lon)
dist_km = dist_m/1000
```

Note: Station information can also be directly extracted using obspy.

17.2.3 2.3 Bandpass filter

```
period_min = 2
period_max = 5
st.filter("bandpass",freqmin=1/period_max,freqmax=1/period_min,zerophase=True)
st_bak = st.copy()                  #keep a copy of orignal data
st.plot();
```

17.2.4 2.4 Plot the waveforms

```
st = Stream()
st1 = obspy.read("GS020_BHZ_20200512.mseed") # Load 1 Day Data from Station GS020
st.append(st1[0])
st2 = obspy.read("ML17_BHZ_20200512.mseed") # Load 1 Day Data from Station ML17
st.append(st2[0])
st.detrend("linear")
st.detrend("constant")
st.plot();
```

17.2.5 2.5 Down-sampling Data

Note: Ambient noise generally conducted in low frequency range, it will reduce computation cost if data is down-sampled.

```
print(f"Before down-sampling, station {st[0].stats.station} has data points {st[0].stats.
↪npts} with sampling rate {st[0].stats.sampling_rate}")
st[0].decimate(factor=5)
print(f"Before down-sampling, station {st[0].stats.station} has data points {st[0].stats.
↪npts} with sampling rate {st[0].stats.sampling_rate}")

print(f"Before down-sampling, station {st[1].stats.station} has data points {st[1].stats.
↪npts} with sampling rate {st[1].stats.sampling_rate}")
st[1].decimate(factor=5)
print(f"Before down-sampling, station {st[1].stats.station} has data points {st[1].stats.
↪npts} with sampling rate {st[1].stats.sampling_rate}")
fs_new = st[0].stats.sampling_rate
```

```
Before down-sampling, station GS020 has data points 8640001 with sampling rate 100.0

Before down-sampling, station GS020 has data points 1728001 with sampling rate 20.0

Before down-sampling, station ML17 has data points 8640001 with sampling rate 100.0

Before down-sampling, station ML17 has data points 1728001 with sampling rate 20.0
```

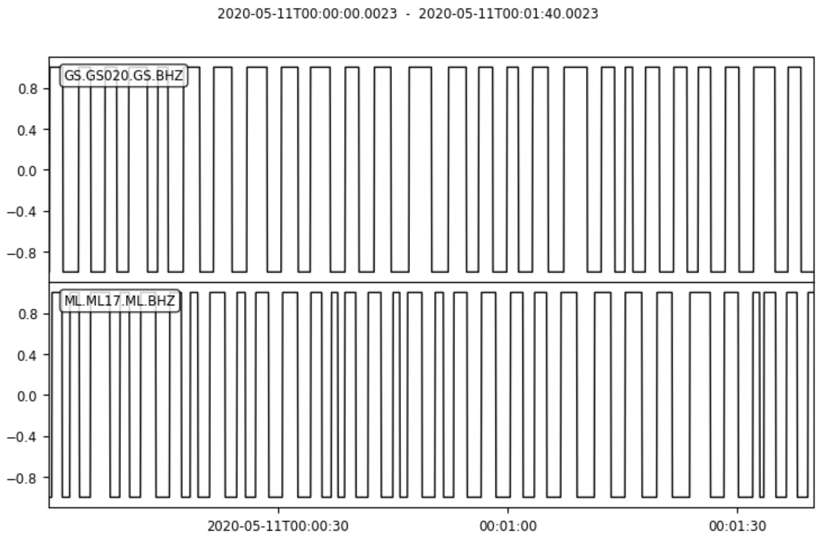
Note: data is resampled from 100 to 20 Hz.

17.2.6 2.6 One-bit Normalization

To remove the non-stationary events that inevitably lie in seismic records, the so-called **one-bit normalization** is commonly applied to the noise data. This processing consists of replacing each sample of a record by its sign.

Note: In “one-bit” normalisation, each data point is replaced with either a 1 or -1, depending on its sign, which remove the amplitude information from the data.

```
st = st_bak.copy()
st[0].data = np.sign(st[0].data)
st[1].data = np.sign(st[1].data)
starttime = st[0].stats.starttime
st.plot(starttime=starttime,endtime=starttime+100 );
```

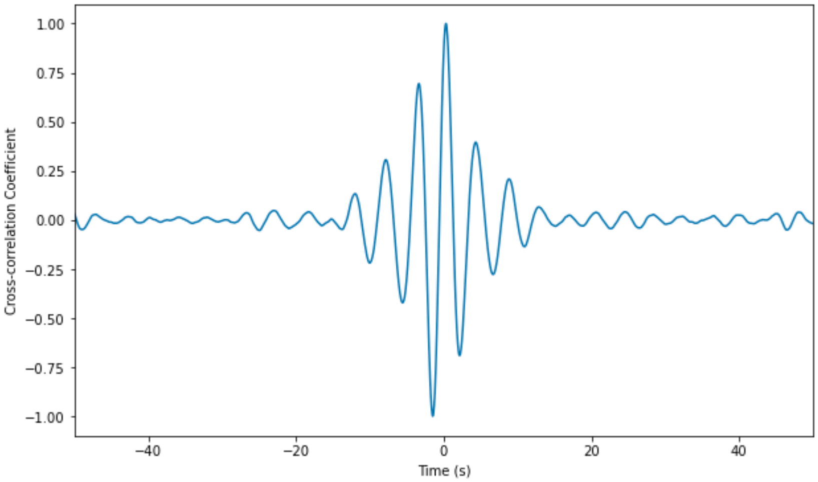


17.2.7 2.7 Conduct Cross-Correlation

```
max_lagtime = 50
max_shift_num = int(np.round(max_lagtime*st[0].stats.sampling_rate))
data1 = st[0].data
data2 = st[1].data
len1 = len(data1)
len2 = len(data2)
min_len = min(len1,len2)
cross_list = []
for shift_num in np.arange(-max_shift_num,max_shift_num+1,1):
    if shift_num<0:
        correlate_value = np.correlate(data1[:min_len+shift_num],data2[-shift_num:min_
↪len])
        cross_list.append(correlate_value.ravel())
    else:
        correlate_value = np.correlate(data2[:min_len-shift_num],data1[shift_num:min_
↪len])
        cross_list.append(correlate_value.ravel())
cross_list = np.array(cross_list)
cross_list = cross_list/np.max(cross_list)
```

17.2.8 2.8 Plot Results

```
time = np.linspace(-max_lagtime,max_lagtime,int(2*max_lagtime*fs_new+1))
plt.plot(time,cross_list)
plt.xlabel("Time (s)")
plt.ylabel("Cross-correlation Coefficient")
plt.xlim(-max_lagtime,max_lagtime)
```



17.2.9 2.5 References:

1. Bensen, G. D., M. H. Ritzwoller, M. P. Barmin, A. Lin Levshin, Feifan Lin, M. P. Moschetti, N. M. Shapiro, and Yanyan Yang. "Processing seismic ambient noise data to obtain reliable broad-band surface wave dispersion measurements." Geophysical journal international 169, no. 3 (2007): 1239-1260.

2. Tutorial 6 - Seismic noise interferometry http://volc_seis_commission.leeds.ac.uk/indexa63d.html

3. Stehly, L., Campillo, M., and Shapiro, N.M. Traveltime measurements from noise correlation: stability and detection of instrumental time shifts. Geophys. J. Int. (2007) 171 223-230.

4. SeisNoise.jl GPU Computing Tutorial https://nextjournal.com/thclements/seisnoisejl-gpu-computing-tutorial

CHAPTER EIGHTEEN

ACKNOWLEDGEMENT

The CUSeisTut website is a product of the project "Developing hands-on micro modules of seismological data processing", supported by the Teaching Development and Language Enhancement Grant (TDLEG) of CUHK.

List of Main Developers:

ABBAS Aqeel

CHAN Yuk Po, Bowie

CHEN Han

LAM Po Wang

LAU Tsz Lam, Zoe

LIU Hui

MU Xinru

SONG Junhao

SUN Chin Long

YAO Suli

YANG Yuyun

ZI Jiping