UNIVERSIDAD DE CONCEPCIÓN
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE GEOFÍSICA

Tópico II:
Guía de uso software ObsPy

Diego González Vidal

Índice general

Capítulo 1

Introducción

El presente informe muestra un modo práctico de como usar el software ObsPy como también entregar las herramientas necesarias para comenzar el autoaprendizaje en el análisis de series de tiempo.

ObsPy es un software de código abierto basado en Python para el procesamiento de datos sismológicos. Posee rutinas para el análisis de series de tiempo de registros sismológicos para los formatos más comunes de estos – .mseed, .sac, .seed, entre otros.

El modo de instalación del software está basado en Debian/Ubuntu de GNU/LINUX pero para mayor información puede visitar la siguiente página para mayor información **https://github.com/obspy/obspy/wiki**.

A modo de nomenclatura, consideraremos el símbolo "$" para mostrar que la línea es un comando en la consola de linux. El símbolo " >>> " lo utilizaremos para denotar que estamos trabajando en Python.

El primer paso para comenzar a utilizar ObsPy corresponde a su instalación que lo veremos en el siguiente capítulo

Capítulo 2

Instalación software ObsPy

Para la instalación de ObsPy debemos agregar sus paquetes a tus repositorios de ubuntu. Para ello debemos saber nuestro código de distribución de ubuntu. Si no estamos seguros de cual es tipeamos en consola lo siguiente:

```
$ lsb_release -cs
```

En mi caso me entregó "precise". El siguiente paso es modificar los paquetes de los repositorios.

```
$ sudo gedit /etc/apt/sources.list
```

Agregamos al final la siguiente línea, donde CODENAME es el código que obtuvimos anteriormente:

```
deb http://deb.obspy.org CODENAME main
```

ObsPy está actualmente soportado oficialmente por debian/ubuntu. Ahora viene la instalación de ObsPy.

```
$ wget --quiet -O - https://raw.github.com/obspy/obspy/
master/misc/debian/public.key | sudo apt-key add -
$ sudo apt-get update
$ sudo apt-get install python-obspy
```

Ahora probamos si ObsPy está correctamente instalado:

```
$ obspy-runtests
$ python
>>> from obspy.core import read
>>> a = read()
>>> a.plot()
```

Capítulo 3

Introducción a Python

Comenzemos iniciando Python y probemos su entorno:

```
$ python
>>> a = 'Hola a todos!'
>>> print a
Hola a todos!
```

Para cerrar Python simplemente tipeamos *Ctrl* + *d*.

3.1. Expresiones regulares

3.1.1. Definir un string

Python puede manipular expresiones regulares o strings. Para definir un string debemos utilizar un apóstrofe y escribir dentro lo que deseamos. Si escribimos un string entre comillas Python interpretará esto y va a imprimir exáctamente lo que hemos escrito. Probemos lo siguiente:

```
>>> 'Hola a todo el mundo!'
'Hola a todo el mundo!'
>>> 'Isn\'t it?'
"Isn't it?"
>>> '"Isn\'t" it?'
'"Isn\'t" it?'
```

3.1.2. Definir un string de varias líneas

También podemos definir un string como una variable cualquiera y de varias líneas. Python reconoce como un salto de línea al escribir \n.

```
>>> A = 'Hola, esta es una prueba sin salto'
>>> print A
Hola, esta es una prueba sin salto
>>> B = 'Hola, esta es una \nprueba con salto'
>>> print B
Hola, esta es una
prueba con salto
```

Notar que Python reconoce los espacios luego de los saltos, esto quiere decir que si tecleamos "\n prueba " tendremos un un espacio antes al comienzo de la siguiente línea. El comando "print" despliega en pantalla la variable que hemos definido anteriormente.

3.1.3. Operadores entre varios strings

Podemos tambien operar sobre expresiones regulares. Con el sísmbolo + juntaremos dos strings distintos y con * repetiremos las veces que queramos el string correspondiente:

```
>>> 'Hola' 'Mundo'
'HolaMundo'
>>> A = 'Hola'
>>> B = 'Mundo'
>>> A+B
'HolaMundo'
>>> A+5*B
'HolaMundoMundoMundoMundoMundo'
```

3.1.4. Seleccionar componentes de un string

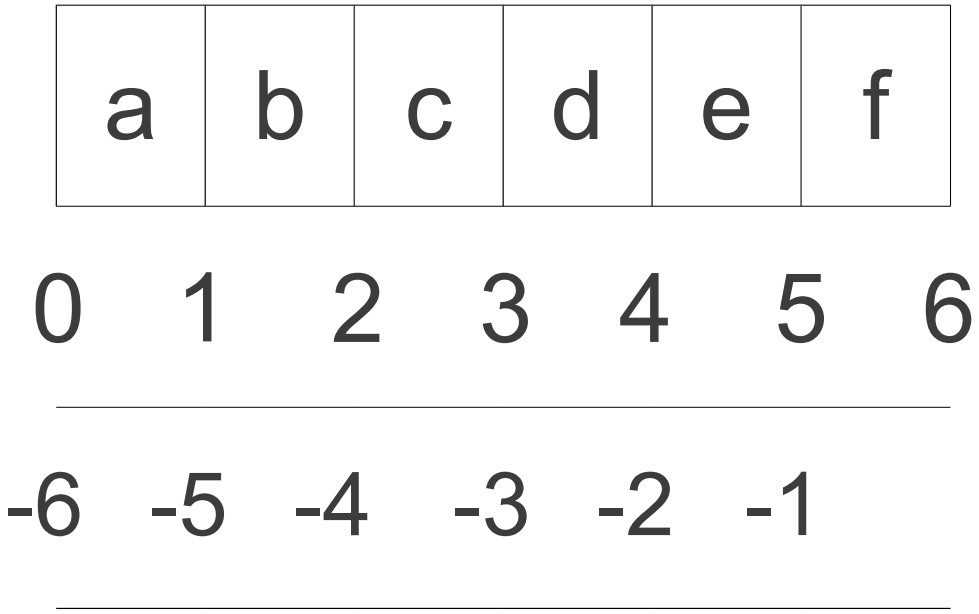
Otra herramienta que poseemos es seleccionar cada componente, individual o en grupo, de un string. Veamos esto:

```
>>> prueba = 'abcdef'
>>> len(prueba)
6
>>> prueba[0]
'a'
>>> prueba[6]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
>>> prueba[-2]
'e'
>>>prueba[0:2]
'ab'
>>>prueba[3:5]
'de'
>>>prueba[5:6]
'f'
```

El comando "len()" nos muestra el largo del string. Notar que al escoger componentes dentro del string, Python define como la componente 0 al primer término del string, además, podemos seleccionar expresiones como -1, -2, etc.

Regla general: al Python parte el conteo en 0. Para seleccionar caracteres dentro de un rango del string debemos escribir entre llaves el rango que deseamos [a:b], donde "a" corresponde al carácter inicial (considerando que el primer término corresponde al 0) y "b" la componente+1 del string que deseamos.

Si tenemos un string "A" de largo n, tendremos que $A = A[:i] + A[i:]$, donde $i \in n$



3.2. Listas de strings

Para almacenar una serie de datos en una variable Python define las listas para ello. Uno puede especificar una lista entre llaves:

```
>>> a = ['hola', 'dgeo', 15, 28]
['hola', 'dgeo', 15, 28]
>>> a[1]
'dgeo'
>>> a[:2] + ['si',4*5]
['hola', 'dgeo', 'si', 20]
```

3.2.1. Operar sobre un elemento individual

Podemos operar sobre un elemento de la lista individual de la siguiente formatos:

```
>>> a[2] = a[2]+30
>>> print a
['hola', 'dgeo', 45, 28]
```

3.2.2. Reemplazar términos de una lista

Podemos reemplazar términos dentro de una lista:

```
>>> a[0:2] = [20, 15]
>>> print a
[20, 15, 45, 28]
```

3.2.3. Insertar elementos dentro de una lista

```
>>> a[1:1] = ['hola']
>>> print a
[20, 'hola', 15, 45, 28]
```

3.2.4. Insertar una lista dentro de otra lista

```
>>> p = [1,2]
>>> len(p)
2
>>> q = ['a',p,'b']
['a', [1, 2], 'b']
>>> len(q)
3
```

3.3. Expresiones matemáticas

3.3.1. Operadores básicos

Python funciona como una calculadora y reconoce automáticamente los símbolos +,-,*,/ y funcionan igual que en otros lenguajes como C o Pascal.

```
>>> 2+2
4
>>>(50-5*6)/4
5
```

También podemos definir muchas variables simultáneamente:

```
>>> x = y = z = 120
>>> print x,y,z
120 120 120
```

3.3.2. Variables reales, enteras y complejas

En Python al definir variables matemáticas debemos diferenciar entre enteros, reales o imaginarios. Si definimos una variable podemos usar la sentencia type() para ver el tipo de variable que estamos utilizando:

```
>>> a = 2
>>> type(a)
<type 'int'>
>>> b = 2.
>>> type(b)
```

```
<type 'float'>
>>> c = 1+2j
>>> type(c)
<type 'complex'>
```

Aquí podemos ver que 'int' es una variable entera, 'float' una variable real y 'complex' una variable compleja.

Si realizamos una división entre variables enteras el resultado que se guardará será solo la parte entera. Por esa razón, es recomendable tener mucho cuidado al momento de definir nuestras variables para poder obtener los resultados que deseamos.

3.3.3. Variables complejas

Si definimos un número complejo "c" lo escribiremos con la siguiente sintaxis:

```
>>> c = a + bj
```

Donde a corresponde a la parte real y b a la parte imaginaria. Notar que la parte imaginaria se representa con j.

También podemos extraer la parte real (real) e imaginaria (imag) como se muestra a continuación:

```
>>> c = 3+4j
>>> c.real
3
>>> c.imag
4
```

Podemos calcular el valor absoluto o la norma de este número complejo con la sentencia abs()

```
>>> abs(c)
5
```

3.3.4. Operador suma especial de Python

En Python podemos definir una variable, como por ejemplo $a = 5$. Si queremos redefinir esta variable sumándole un número podemos hacerlo de la siguiente manera $a = a + 3 = 8$ o, con una función especial de Python. Veámoslo ahora:

```
>>> a = 5
>>> a+= 3
>>> a
8
```

3.3.5. Paquete MATH para funciones matemáticas

En Python no está incluidas automáticamente las funciones matemáticas trigonométricas, exponenciales, logarítmicas o constantes numéricas como π. Para ello debemos importar el paquete math escribiendo en python lo siguiente

```
>>> import math
```

Para llamar a cualquier función deberemos anteponer la sentencia "math.", si queremos por ejemplo, calcular $\sin(\frac{\pi}{2})$ en Python deberemos escribir entonces:

```
>>> math.sin(math.pi/2)
1.0
```

Para saber el nombre de las funciones y lo que realizan podemos visitar la página:

`http://docs.python.org/2/library/math.html?highlight=math#math`

3.4. Scripts en Python

En Python además de trabajar directamente en consola podemos escribir scripts en un editor de texto, como por ejemplo Gedit y luego ejecutarlos en consola. Los script deberán tener extensión .py, si nuestro script se llama test.py y queremos que nos muestre el valor en pantalla de $\sin(\frac{\pi}{2})$, tendríamos que escribir un script del siguiente modo:

```
import math
a = math.sin(math.pi/2)
print a
```

El modo de ejecución en consola sería el siguiente:

```
$ python test.py
1.0
```

3.5. Loops y condicionales

3.5.1. Sentencia IF

La sentencia IF es una de las más conocidas. Si bien, realiza las mismas operaciones que en otros lenguajes, variará simplemente la sintaxis.

Utilizemos el siguiente ejemplo:

Escribamos un script que nos solicite ingresar un número entero. Con el condicional IF haremos que si el número ingresado es mayor a 3 imprima su cuadrado, si es igual a 3 que imprima cero y si es menor a 3 que imprima su raiz. Ese código se vería de la siguiente forma:

```
import math
print 'Ingrese un numero entero'
x = int(raw_input())
if x > 3:
     print "el cuadrado de x es", x**2
elif x == 3:
     print 0
else:
     print "la raiz de x es", math.sqrt(x)
```

3.5.2. Sentencia FOR

Similar a bash, en la sentencia for se define una variable que se utiliza solo en el loop para extraer información de alguna lista que definimos anteriormente.

Veamos un ejemplo en que tenemos una lista con números del 1 al 10 y queremos que el script nos imprima su cuadrado. Esto es:

```
a = [1,2,3,4,5,6,7,8,9,10]
for i in a:
    print i**2
```

3.5.3. Sentencia WHILE

La sentencia WHILE indicará que se cumplirá cierto proceso mientras se cumpla la condición que uno ingrese.

```
a = 1
while a<10:
     print a
     a += 1
```

3.6. Función RANGE()

La Función Range() tiene como función generar una progresión aritmética.

```
>>> range(4)
[0, 1, 2, 3]
```

El ejemplo anterior genera una lista de largo 4 partiendo por el cero.

```
>>> range(2,6)
[2, 3, 4, 5]
>>> range(2,6,3)
[2, 5]
```

El ejemplo anterior genera una lista que comienza en 2, termina en el número anterior a 6 y un espacio de 3 números.

Como regla general tendremos que en la sentencia $RANGE(A,B,C)$ generará una lista que comienza en A, llega hasta el número anterior a B. C indicará el espacio que abrá entre un número de la lista y el siguiente.

Combinar len( ) y range( )

Podemos combinar las funciones len() y range().

```
a = ['a', 'b', 'c', 'd', 'e','f']
for i in range(len(a)):
    print i, a[i]
```

Para profundizar mejor en las posibilidades que ofrece python recomendamos el siguiente link:

http://docs.python.org/2/py-modindex.html

Capítulo 4

Introducción a ObsPy

4.1. UTC date time

En ObsPy tenemos una gran herramienta que consiste en manejar los datos sincronizado con UTC date time.

```
>>> from obspy.core import UTCDateTime
>>> time = UTCDateTime("2012-09-07T12:15:00")
>>> print time
2012-09-07T12:15:00.000000Z
```

Si estamos conocemos solo la hora en cierto huso de horario, por ejemplo en -07:00, el programa nos puede entregar la hora en tiempo universal:

```
>>>from obspy.core import UTCDateTime
>>>time = UTCDateTime("2012-09-07T12:15:00-07:00")
>>>print time
2012-09-07T19:15:00.000000Z
```

4.1.1. Extracción de información de UTC date time

Si deseamos saber el año, el julday u otro parámetro a partir de una variable de UTC date time podemos extraerlo de la siguiente manera:

```
>>> time = UTCDateTime("2012-09-07T12:15:00")
>>> time.year
2012
>>> time.julday
251
>>> time.timestamp
1347020100.0
>>> time.weekday
4
```

4.1.2. Modificando UTC date time

Para definir una variable tiempo que contenga información de UTC date time se puede escribir de la siguiente manera igualmente:

```
>>> UTCDateTime(2012, 9, 7, 12, 15, 0)
```

Donde el primer térmico corresponde al año, seguido del mes, día, hora, minutos y segundos.
Podemos restar dos variable de tiempo, el cual nos entregará el resultado en segundos:

```
>>> time1 = UTCDateTime(2012, 9, 7, 12, 15, 0)
>>> time2 = UTCDateTime(2012, 9, 7, 13, 15, 0)
>>> time2-time1
3600.0
```

4.2. Información de sismogramas

Para la lectura de sismogramas tendremos que importar la funció para leer sismogramas:

```
>>> from obspy.core import read
```

En ObsPy generalmente llamamos st (stream) a la variable que contiene las tres componentes de un registro sismológico. Llamamos tr (traza) a solo una componente de la variable stream.
Consideremos los archivos *CCSP.HNE..a.sac*, *CCSP.HNN..a.sac* y *CCSP.HNZ..a.sac* para estudios:

```
>>> from obspy.core import read
>>> st = read('*sac')
>>> print st
3 Trace(s) in Stream:
.CCSP..HNE | 2010-02-27T06:34:23.000000Z - 2010-02-27T06:37:44.980000Z
| 100.0 Hz, 20199 samples
.CCSP..HNN | 2010-02-27T06:34:23.000000Z - 2010-02-27T06:37:44.980000Z
| 100.0 Hz, 20199 samples
.CCSP..HNZ | 2010-02-27T06:34:23.000000Z - 2010-02-27T06:37:44.980000Z
| 100.0 Hz, 20199 samples
```

Al imprimir el stream st vemos la información que contiene los archivos que leímos anteriormete. Vemos que los registros tienen una frecuencia de muestreo de 100 Hz, 20199 npts y además imprime la información UTC date time de los registros.

```
>>> tr = st[0]
>>> tr.stats.station
'CCSP'
>>> msg = "%s %s" %(tr.stats.station, str(tr.stats.starttime))
>>> print msg
CCSP 2010-02-27T06:34:23.000000
```

Para definir la varaible msg definimos con el símbolo % un string en el cual guardará, en este caso, el nombre de la estación y el tiempo de inicio del registro sísmico.

Capítulo 5

Lectura de sismogramas

Un stream es una variable que contiene múltiples trazas. Cada traza tiene atribuído información estadística (header).

```
>>> from obspy.core import read
>>> st = read('*sac')
>>> tr = st[0]
>>> print tr
.CCSP..HNE | 2010-02-27T06:34:23.000000Z - 2010-02-27T06:37:44.980000Z
| 100.0 Hz, 20199 samples
```

5.1. Acceso a información del header de cada traza (Meta data)

Podemos acceder a la Meta Data a través de la sentencia STATS.

```
>>> print tr.stats
         network:
         station: CCSP
        location:
         channel: HNE
       starttime: 2010-02-27T06:34:23.000000Z
         endtime: 2010-02-27T06:37:44.980000Z
   sampling_rate: 100.0
           delta: 0.01
            npts: 20199
           calib: 1.0
         _format: SAC
>>> print tr.stats.starttime
2010-02-27T06:34:23.000000Z
>>> print tr.stats.npts
20199
```

5.2. Acceso a información de la serie de tiempo

```
>>> tr.data
array([ 0.77003998,  0.77249002,  0.77494001, ...,  1.85169816,
        0.7978    ,  0.056248  ], dtype=float32)
>>> len(tr)
20199
```

5.3. Previsualización de registros

```
>>> st.plot()
```

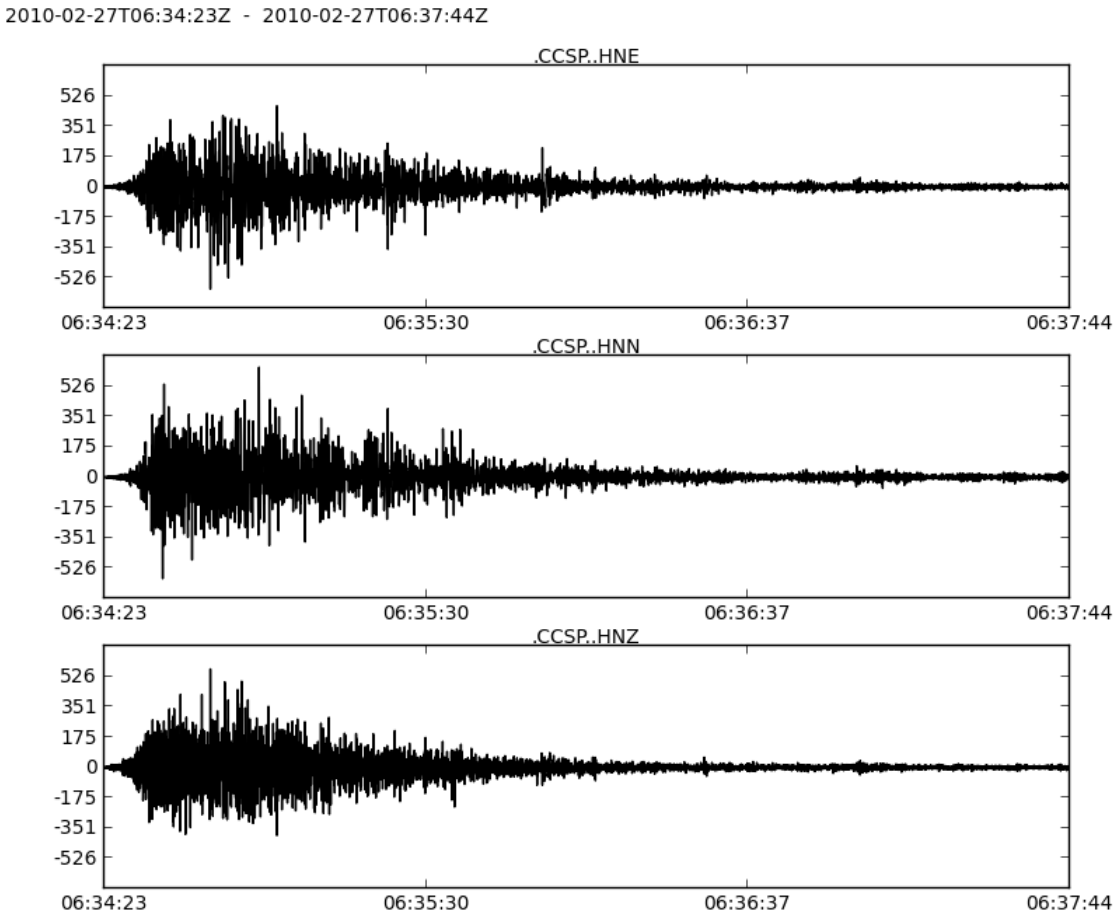


Figura 5.1: Multicanal CCSP, Terremoto Maule, Febrero 2010.

Capítulo 6

Ploteo de sismogramas

6.1. Plotear un canal simple

```
>>> from obspy.core import read
>>> st = read('*sac')
>>> tr = st[0]
>>> tr.plot(outfile='single.png')    # guardar el grafico
```

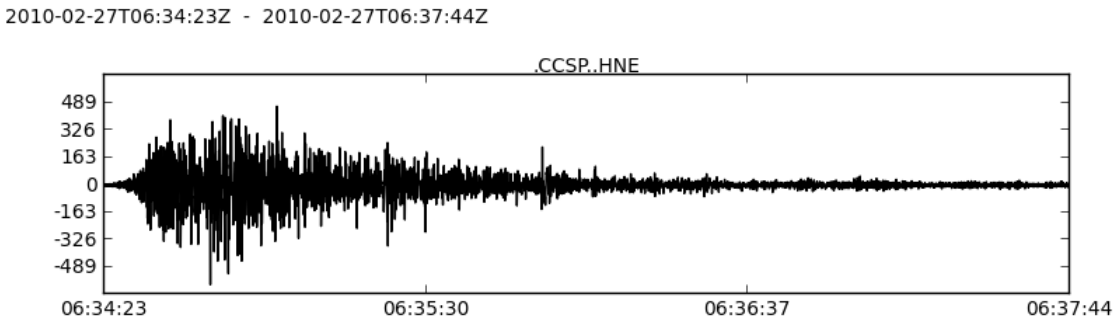


Figura 6.1: Componente Este CCSP, Terremoto Maule, Febrero 2010.

6.1.1. Personalizar ploteos

En ObsPy podemos personalizar nuestros gráficos cambiando los colores e incluir uno u otra información.

```
>>> from obspy.core import read
>>> st = read('*sac')
>>> tr = st[0]
>>> dt = tr.stats.starttime
>>> tr.plot(color='blue', number_of_ticks=7,
...          tick_rotation=10, tick_format='%I:%M:%S %p',
...          starttime=dt+0.01, endtime=dt+2)
```

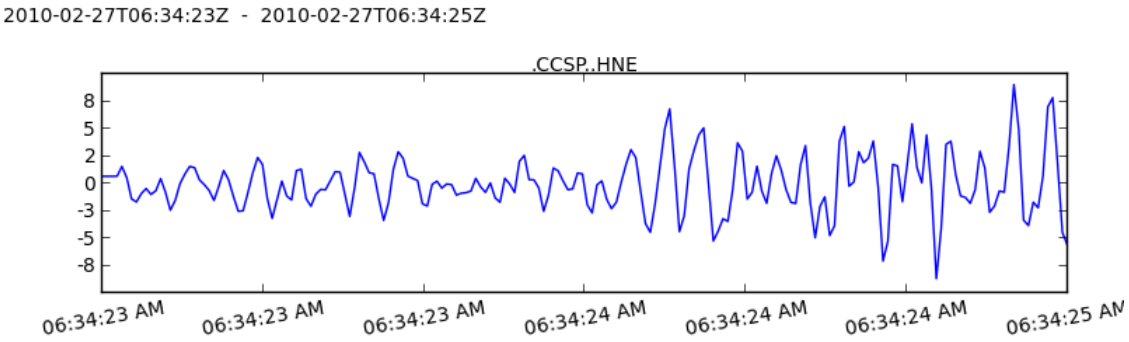


Figura 6.2: Componente Este CCSP, Terremoto Maule, Febrero 2010

Dentro de los parámetros para la función .plot(), podemos seleccionar el color, el número de marcas en el eje del tiempo, rotación en grados en sentido antihorario, formato (donde %I es un string para la hora, %M un string para los minutos, %S un string para los segundos) y además podemos plotear una ventana de tiempo dado por starttime y endtime.

6.2. Plotear múltiples canales

También podemos plotear un stream con una serie de trazas en su interior. Podemos modificar su tamaño con el parámetro size() donde se ingresan los pixeles deseados:

```
>>> from obspy.core import read
>>> st = read('*sac')
>>> st.plot(size=(800, 600))
```

6.3. Ploteo de un día de datos

Para esta sección consideremos el archivo $PB03_2007_278.mseed$ que corresponde al registo del día juliano 278 del año 2007, de la estación PB03 de la red IPOC en el norte de Chile.

```
>>> from obspy.core import read
>>> tr = read('PB03_2007_278.mseed')
>>> tr.plot(type='dayplot')
```

Para profundizar en las entradas de .plot() puedes visitar el siguiente link:

```
http://docs.obspy.org/packages/autogen/
obspy.core.stream.Stream.plot.html?highlight=plot#obspy.core.stream.Stream.plot
```

6.4. Taper, remover promedio y tendencia lineal

Para esta sección consideremos el archivo XJ.MURT.20040125115445..HHZ.mseed

Leemos el archivo y lo ploteamos

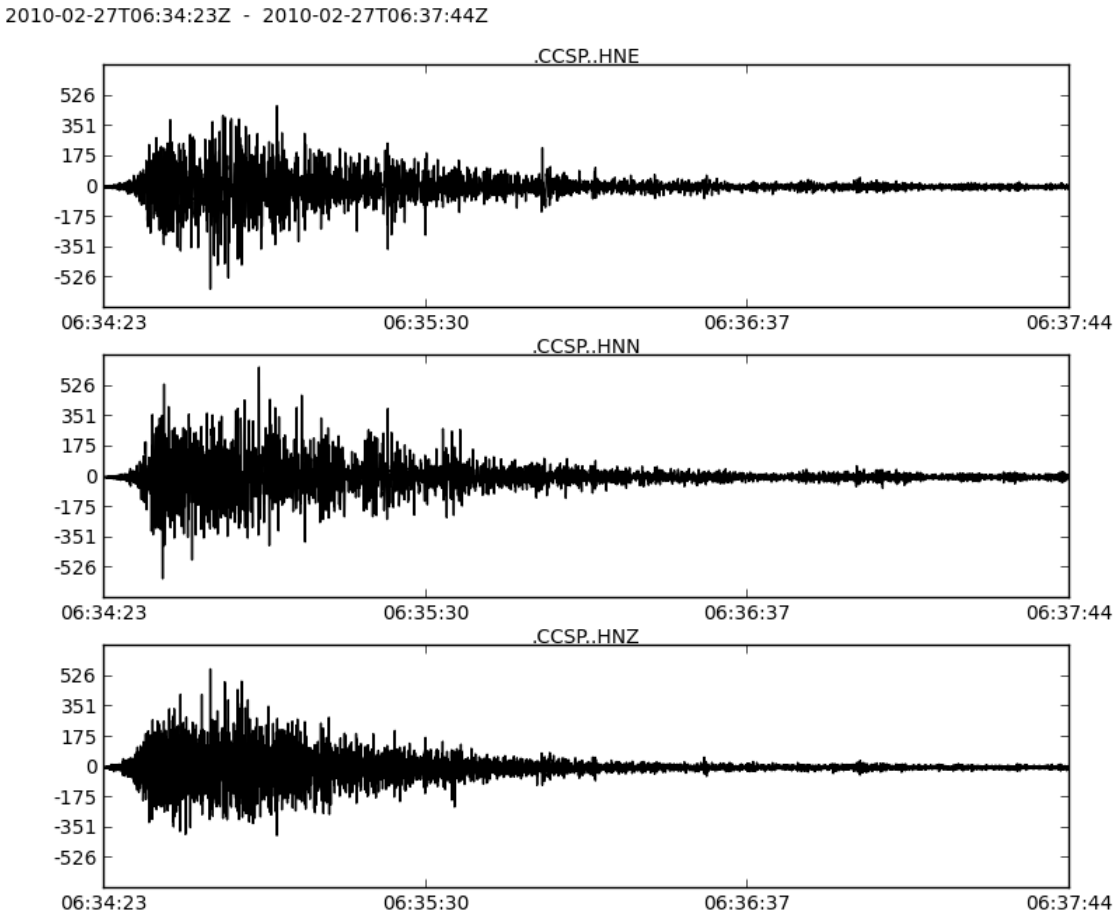


Figura 6.3: Terremoto Maule, Febrero 2010. Estación CCSP

```
>>> st = read('XJ.MURT.20040125115445..HHZ.mseed')
>>> st[0].plot()
```

6.4.1. DETREND

Para remover la tendencia lineal tenemos que utilizar la sentencia .DETREND()

Dentro de las opciones podemos seleccionar "constant" que está por defecto. Lo que hace esta opción es extraer la tendencia lineal considerando la recta que se genera entre el primer y último valor de la traza. La segunda opción que tenemos es "linear" que utiliza una aproximación por mínimos cuadrados para extraer la tendencia lineal de la traza.

```
>>> st[0].detrend('linear')
```

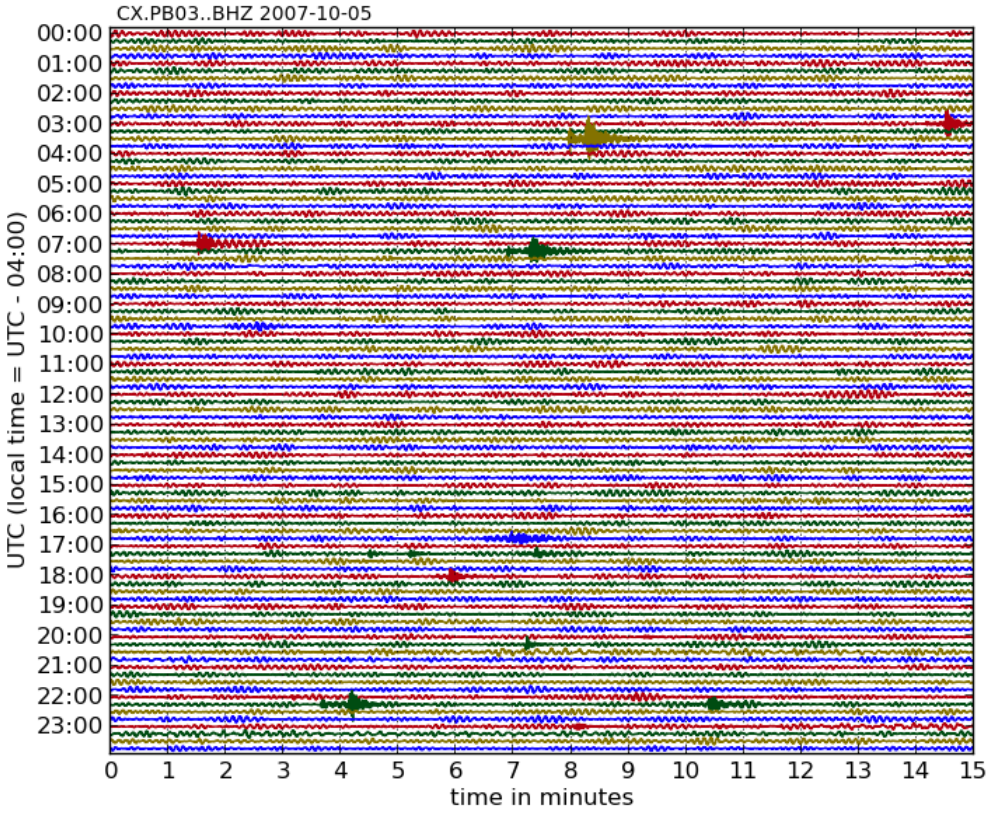


Figura 6.4: Ploteo de un día de datos de 278-2007. Estación PB03, red IPOC

6.4.2. DEMEAN

Al momento de querer remover el promedio de la serie simplemente debemos utilizar la tercera opción que poseemos con la sentencia .DETREND

La opción ”demean“ removerá el promedio de la traza seleccionada.

```
>>> st[0].detrend('demean')
```

6.4.3. TAPER

Hacer un taper nos ayuda a evitar el fenómeno de Gibbs (discontinuidades en la señal). Básicamente lo que se hace es llevar a cero los extremos de la serie de tiempo para evitar estas discontinuidades implícitas. Un ejemplo es una función de Hanning.

En OBSPY esto se hace simplemente con la función .TAPER()

Esta función tiene como entrada el tipo de taper a usar, entre las cuales se puede seleccionar ”cosine”, “bartlett”, “gaussian”, “triang”, entre otros. Por defecto se considera “cosine”.

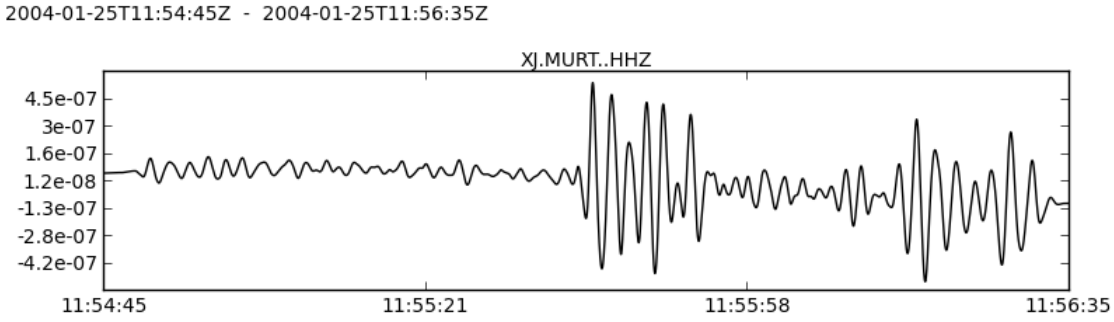


Figura 6.5: Registro evento sísmico puro. Fecha 2004-01-25 y frecuencia de muestreo 100 Hz.

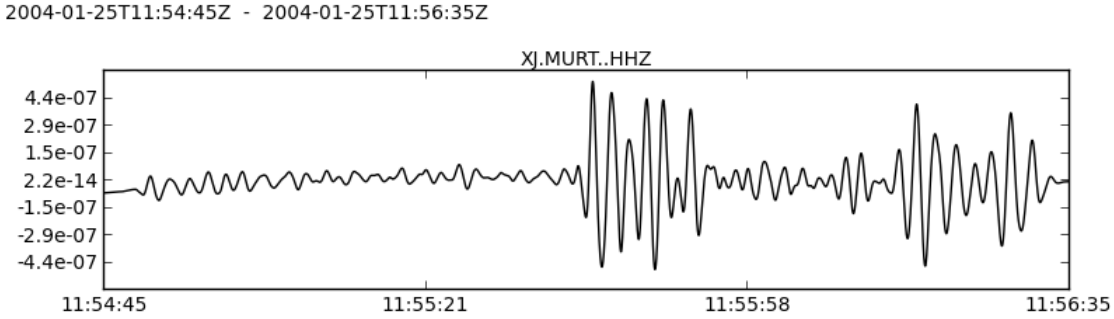


Figura 6.6: Registro evento sísmico con tendencia lineal removida a partir de una aproximación de mínimos cuadrados. Fecha 2004-01-25 y frecuencia de muestreo 100 Hz.

Para profundizar en los parámetros disponibles y sus métodos ver:

```
http://docs.obspy.org/packages/autogen/
obspy.core.trace.Trace.taper.html#obspy.core.trace.Trace.taper
```

Tipeamos esto en consola:

```
>>> st[0].taper()
```

Capítulo 7

Función Merge, Decimate y Trim

Muchas veces, tenemos que las series de tiempo que poseemos vienen cortadas entre sí y hasta se cruzan en algunos tiempos. Para poder unir estas series podemos utilizar la función MERGE que viene incluida en OBSPY. Además, para interpolar las series de tiempo o, dicho de otra forma, reducir la frecuencia de muestreo podemos utilizar la función DECIMATE para lograr nuestro objetivo. Tambien la función TRIM se utilizará para cortar la señal en una banda de tiempo que queramos.

7.1. Unir registros de una estación en una sola traza.

Como explicamos anteriormente, para unir en una traza diferentes series de tiempo podemos utilizar la función MERGE.

A continuación leemos la componente HHZ de la estación TRA del Volcán Villarica del 1 de Marzo de 2013. Ver Figura 7.1.

```
>>> from obspy.core import read
>>> st = read('VV.TRA.HHZ.2013.03.010[12345].sac')
>>> st.plot(automerge=False)
```

Entonces para unir todas las trazas seleccionadas lo que tipeamos es:

```
>>> st.merge(method=1, fill_value='latest')
```

El method=0 elimina los archivos que se cruzan, en cambio method=1 une los términos cruzados mediante una interpolación (en nuestro caso es el default = 0).

http://docs.obspy.org/packages/autogen/obspy.core.trace.Trace.__add__.html#handling-overlaps

El la opción "latest" del parámetro "fill_value" significa que utilizará el último valor de la traza al momento de unir las series de tiempo. El resultado se puede observar en la Figura 7.2.

7.2. Interpolar series de tiempo

La función .DECIMATE() interpola la traza seleccionada. Básicamente lo que hace es reducir la frecuencia de muestreo de una serie de tiempo en un factor determinado.

Por ejemplo, nuestra serie de tiempo tiene una frecuencia de muestreo de 100 Hz y queremos reducirla a 25 Hz, simplemente debemos aplicar un factor 4. Veamos el siguiente código:

```
>>> st = read('2004361005853_XJ_TRANB.HHZ')
>>> print st[0].stats.sampling_rate
100.0
>>> st[0].decimate(factor=2)
>>> print st[0].stats.sampling_rate
25.0
```

Una forma de calcular el factor que debemos ingresar para reducir el sampling rate al valor que deseamos se puede ver de la siguiente forma:

$$factor = \frac{sampling\_rate}{frecuencia\_deseada} \tag{7.1}$$

7.3. Cortar en una ventana de tiempo una traza

http://docs.obspy.org/packages/autogen/obspy.core.trace.Trace.trim.html

La función .TRIM() nos permite cortar una traza ingresando el starttime y endtime deseado. Para ello utilicemos el archivo QUE_2013_070..HHZ.mseed como ejemplo (Figura 7.3):

Cortemos la señal entre las 15.00 y 23.30 horas. Para ello definiremos el tiempo de inicio de la señal y le sumaremos la cantidad de horas necesarias. Notar que una hora posee 3600 segundos.

```
>>> st = read('QUE_2013_070..HHZ.mseed')
>>> t = st[0].stats.starttime
>>> t1 = t+3600*15
>>> t2 = t + 3600*23.5
>>> st[0].trim(t1, t2)
>>> st[0].plot()
```

Como resultado podemos ver en Figura 7.4

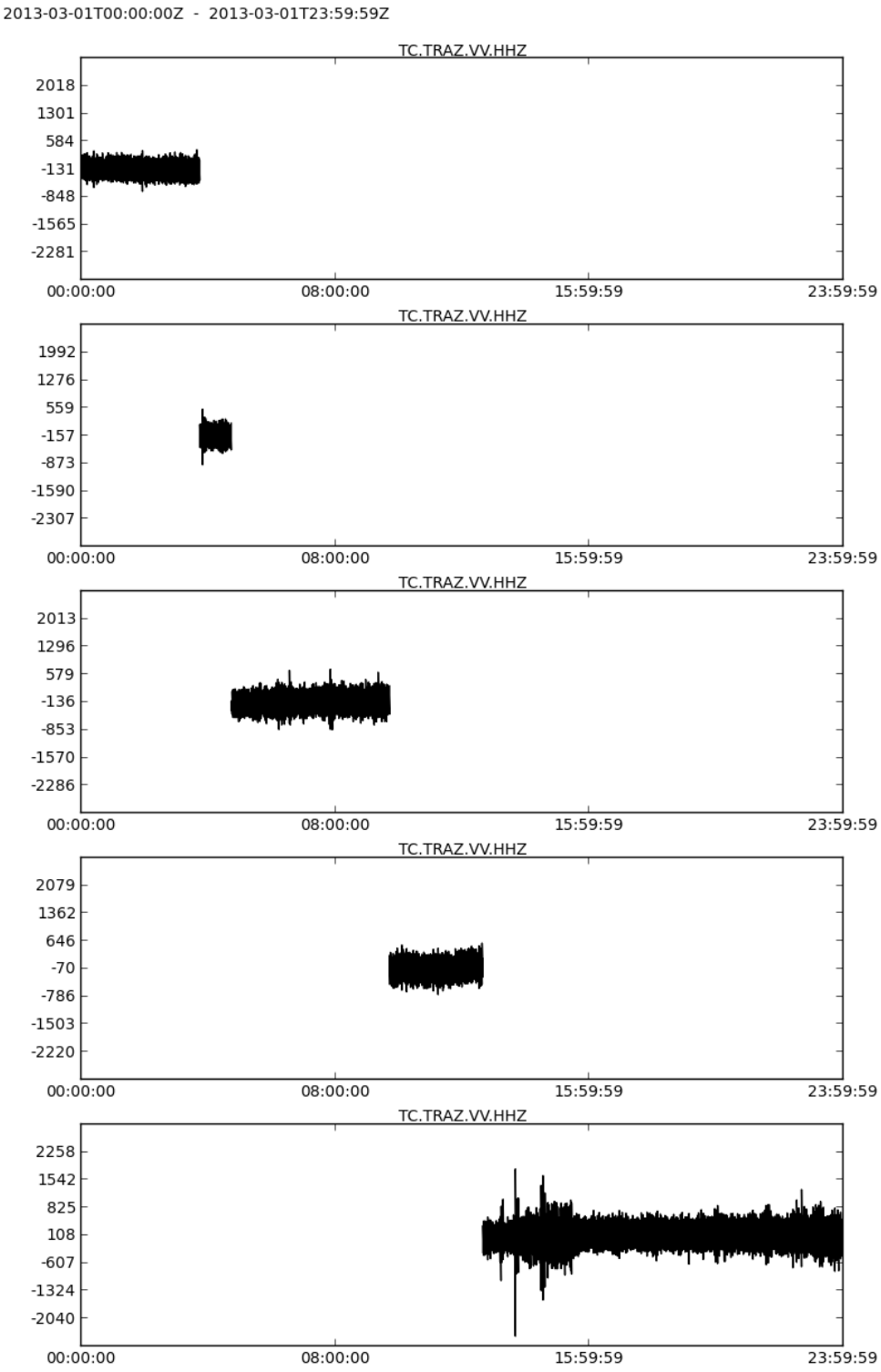


Figura 7.1: Estación TRA Volcán Villarrica 1 de Marzo de 2013 en múltiples trazas.

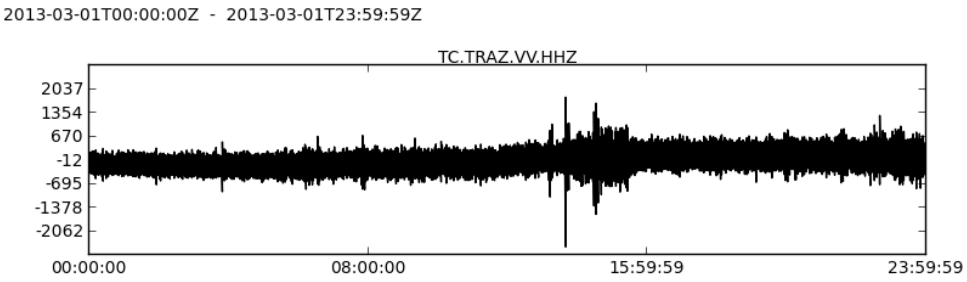


Figura 7.2: Estación TRA Volcán Villarrica 1 de Marzo de 2013 en una sola traza.

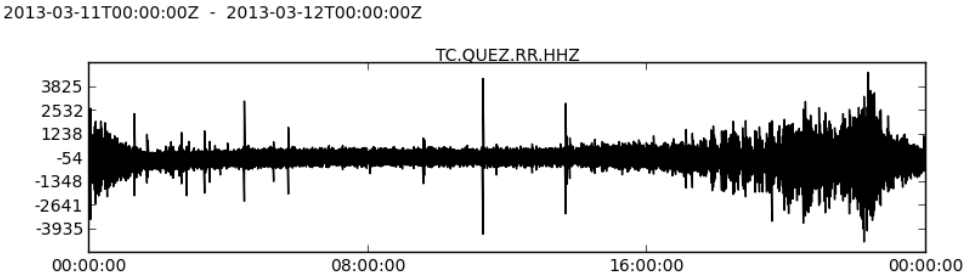


Figura 7.3: Estación QUE del Volcán Lascar.

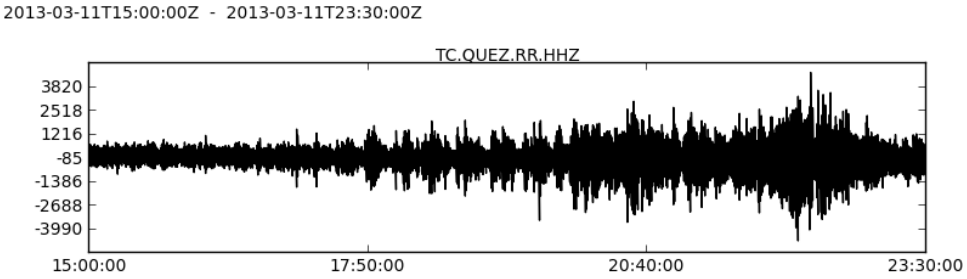


Figura 7.4: Estación QUE del Volcán Lascar. Corte entre las 15.00 y 23.30 horas.

Capítulo 8

Filtros

En OBSPY los filtros son de tipo Butterworth con la función .FILTER(). En este capítulo utilizaremos el archivo
CCSP.HNZ..a.sac

Los resultados podemos ver en la Figura 8.1.

8.1. Pasa banda

Para filtrar en pasa banda utilizaremos la opción “bandpass”.

Los parámetros de entrada serán “freqmin“ y ”freqmax“, que son la esquina inferior y superior en frecuencia en Hz respectivamente. Por ejemplo, para filtrar entre 0.5 y 2 Hz tendremos que tipear:

```
>>> tr.filter('bandpass',freqmin=0.5,freqmax=2.)
```

8.2. Pasa alto

Para filtrar en pasa alto utilizaremos la opción ”highpass”.

El parámetro de entrada será “freq“ frecuencia en Hz. Por ejemplo, para filtrar dejando pasar las frecuencias sobre 12 Hz tipearemos:

```
>>> tr.filter('highpass',freq=12.0)
```

8.3. Pasa bajo

Para filtrar en pasa bajo utilizaremos la opción ”lowpass”.

El parámetro de entrada será “freq“ frecuencia en Hz. Por ejemplo, para filtrar dejando pasar las frecuencias bajo 0.5 Hz tipearemos:

```
>>> tr.filter('lowpass', freq=0.5)
```

8.4. Band – stop

Remueve la señal entre una frecuencia de corte mínima y una frecuencia de corte máxima

```
>>> tr.filter('bandstop',freqmin=0.5,freqmax=2.)
```

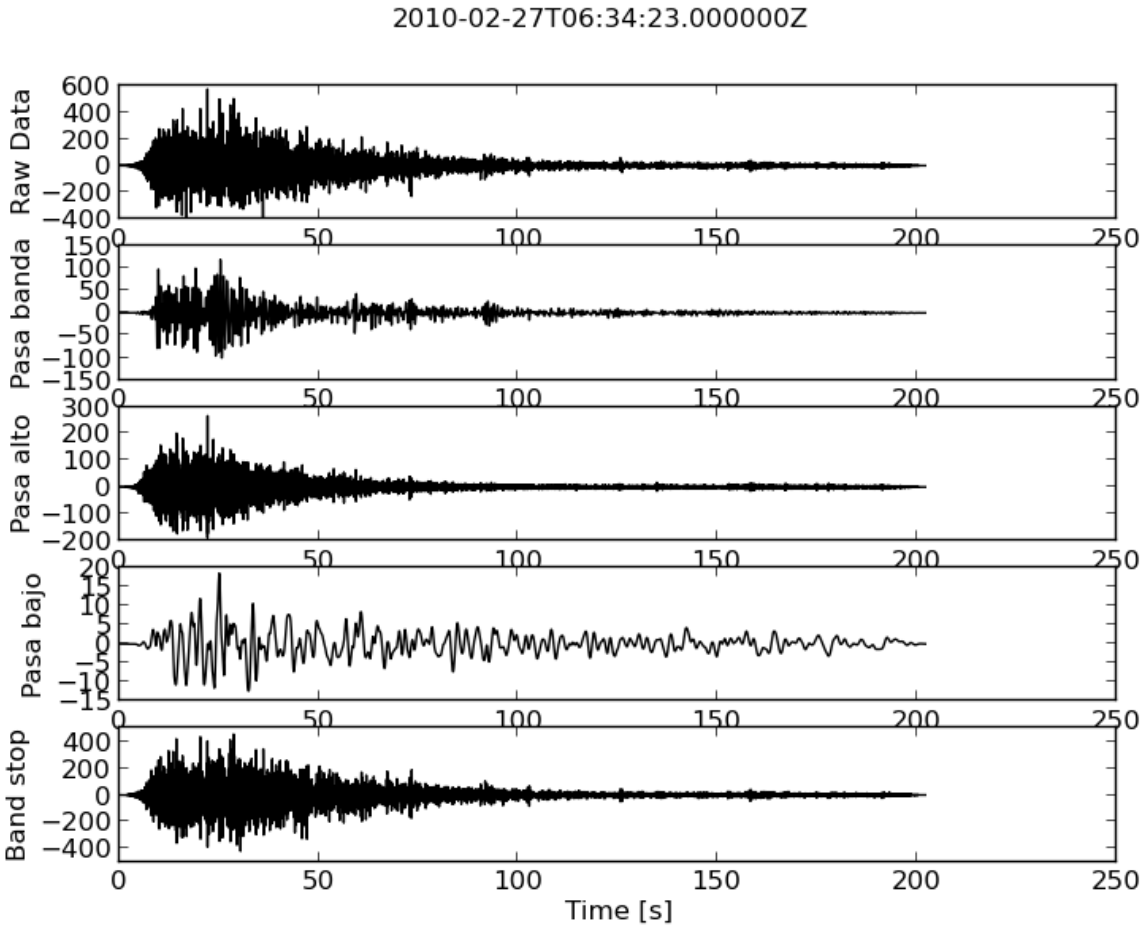


Figura 8.1: De arriba habia abajo. 1) Componente HHZ estación CCSP terremoto Febrero 2010. 2) Señal filtrada pasa banda entre 0.5 y 2 Hz. 3) Señal filtrada pasa alto con frecuencia de corte 12 Hz. 4) Señal filtrada pasa bajo con frecuencia de corte 0.5 Hz. 5) Señal filtrada band-stop entre los 0.5 y 2 Hz.

Capítulo 9

Espectrogramas

9.1. Espectrograma simple

Espectrogramas de series de tiempo se plotean simplemente con la función .SPECTROGRAM().

Como ejemplo usaremos la serie de tiempo 2004361005853_XJ_TRANB.HHZ, la cual le quitaremos la media, tendencia lineal, aplicaremos un taper y además interpolaremos la serie de tiempo bajando su frecuencia de muestreo a 50 Hz.

```
>>> tr.spectrogram(samp_rate=50, per_lap=0.90, wlen=70 , log=True)
```

(1) samp_rate indica la frecuencia de muestreo para la fft. (2) per_lap indica el porcentaje de superposición de la ventana móvil al momento de calcular los espectros correspondientes. Va de 0 a 1 y entre más alto el overlap toma más tiempo. (3) wlen indica el largo de la ventana para calcular la fft en segundos. Evitar que este parámetro sea demaciado pequeño. (4) log es un parámetro lógico, indica si la escala de frecuencia es logarítmica o no. (5) Podemos ingresar un título agregando el parámetro [title='XJ.TRANB..HHZ ' + str(st[0].stats.starttime)]. Ver Figura 9.1

9.2. Subplot y espectrograma

Para realizar un subplot que contenga el espectrograma y la serie de tiempo la cosa se torna un tanto más complejo. Para ello hay quer usar la función "matplotlib.mlab.specgram" la cual calcula un espectrograma utilizando por defecto una ventana de tipo hanning.

```
>>> from obspy.core import read
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> t = np.arange(0, tr.stats.npts / tr.stats.sampling_rate, tr.stats.delta)
>>> plt.subplot(211)
>>> plt.specgram(tr, NFFT=2**6, Fs=50, noverlap=2**5)
>>> plt.subplot(212)
>>> plt.plot(t, tr.data, 'k')
>>> plt.xlabel('Time [s]')
>>> plt.suptitle(tr.stats.starttime)
>>> plt.show()
```

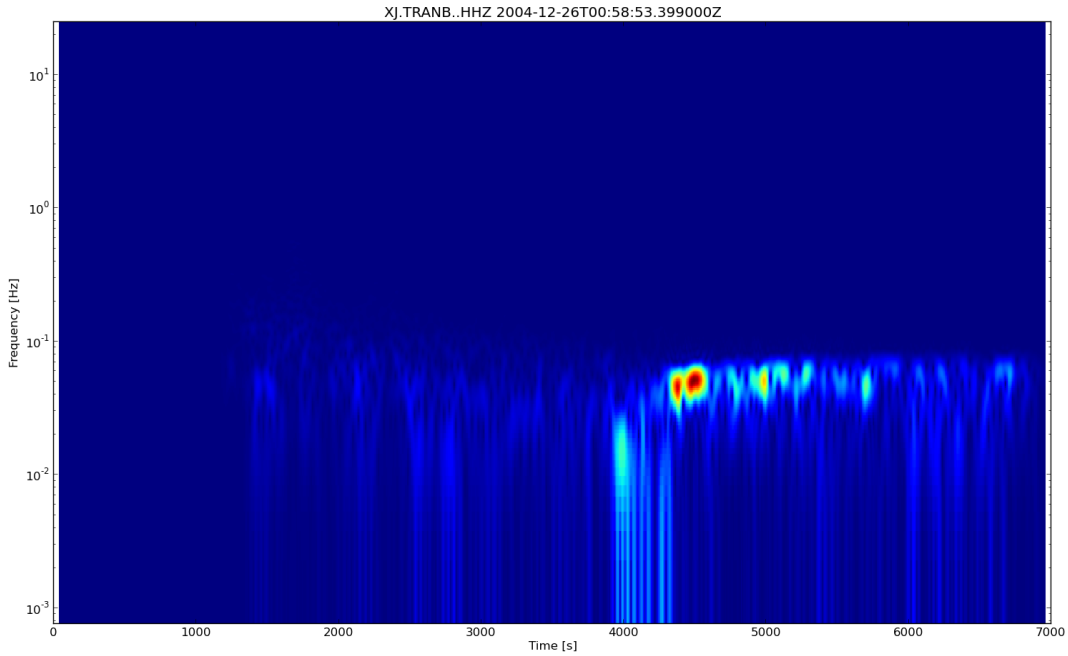


Figura 9.1: Espectrograma de 2004361005853_XJ_TRANB.HHZ.

Debido a que esta librería está aún en desarrollo, por ahora no existe la posibilidad de seleccionar el tipo de escala de la frecuencia. Por defecto será una escala lineal.

(1) NFFT corresponde al número de puntos utilizados en la fft (fast fourier transform). Conviene utilizar potencias de 2 para aumentar la eficiencia del cálculo. (2) Fs corresponde a la frecuencia de muestreo. Calcula las frecuencias a utilizar en la fft. (3) noverlap corresponde al parámetro de superposición de los datos mientras se mueve la ventana para calcular cada fft. Debe ser un entero. (4) scale_by_freq. Ver Figura 9.2

Para mayor información puedes visitar: http://matplotlib.org/api/mlab_api.html#matplotlib.mlab.specgram

9.3. CWT

La función CWT (Continuous Wavelet Transformation) calculará las ondeletas en el dominio de la frecuencia utilizando una ventana morlet por defecto. Lamentablente, aún no hay otras opciones para seleccionar.

cwt(st, dt, w0, fmin, fmax). (1) st se refiere a los datos. (2) dt es tiempo de muestreo. (3) w0 indica un parámetro de la ondeleta que equilibra la resolución de la frecuencia con el dominio del tiempo. (4) fmin y fmax corresponden a los límites de las frecuencias en el eje y.

El método de cálculo de CWT está explicado en el paper [Kristekova2006]

http://www.bssaonline.org/content/96/5/1836

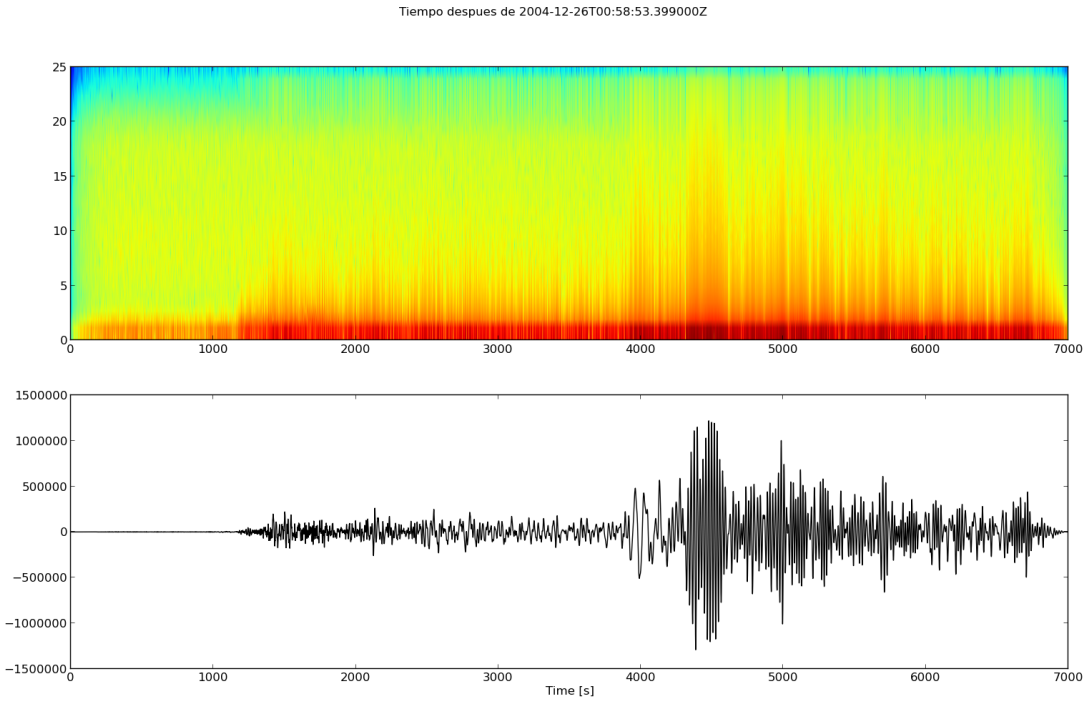


Figura 9.2: Subplot de espectrograma y serie de tiempo 2004361005853_XJ_TRANB.HHZ.

Como ejemplo, adjunto el siguiente código (ver Figura 9.3):

```
>>> import matplotlib.pyplot as plt
>>> from obspy.core import read
>>> import numpy as np
>>> import mlpy
>>> from obspy.signal.tf_misfit import cwt

>>> st = read('2004361005853_XJ_TRANB.HHZ')
>>> tr = st[0]

>>> tr.detrend('linear')
>>> tr.detrend('demean')
>>> tr.taper()
>>> tr.decimate(factor=2)

>>> npts = tr.stats.npts
>>> dt = tr.stats.delta
>>> t = np.arange(0, tr.stats.npts / tr.stats.sampling_rate, tr.stats.delta)
>>> f_min = 0.01
>>> f_max = 25.
```

```
>>> scalogram = cwt(tr.data, 1/50., 8, f_min, f_max)

>>> fig = plt.figure()
>>> ax1 = fig.add_axes([0.1, 0.35, 0.8, 0.6])
>>> ax2 = fig.add_axes([0.1, 0.1, 0.8, 0.2])
>>> ax1.imshow(np.abs(scalogram)[-1::-1], extent=[t[0], t[-1], f_min, f_max],
    aspect='auto', interpolation="nearest")
>>> ax1.set_ylabel("Frecuencia [Hz]")
>>> ax1.set_yscale('log')
>>> ax2.plot(t, tr.data, 'k')
>>> ax2.set_xlabel("Time after %s [s]" %tr.stats.starttime)
>>> plt.show()
```

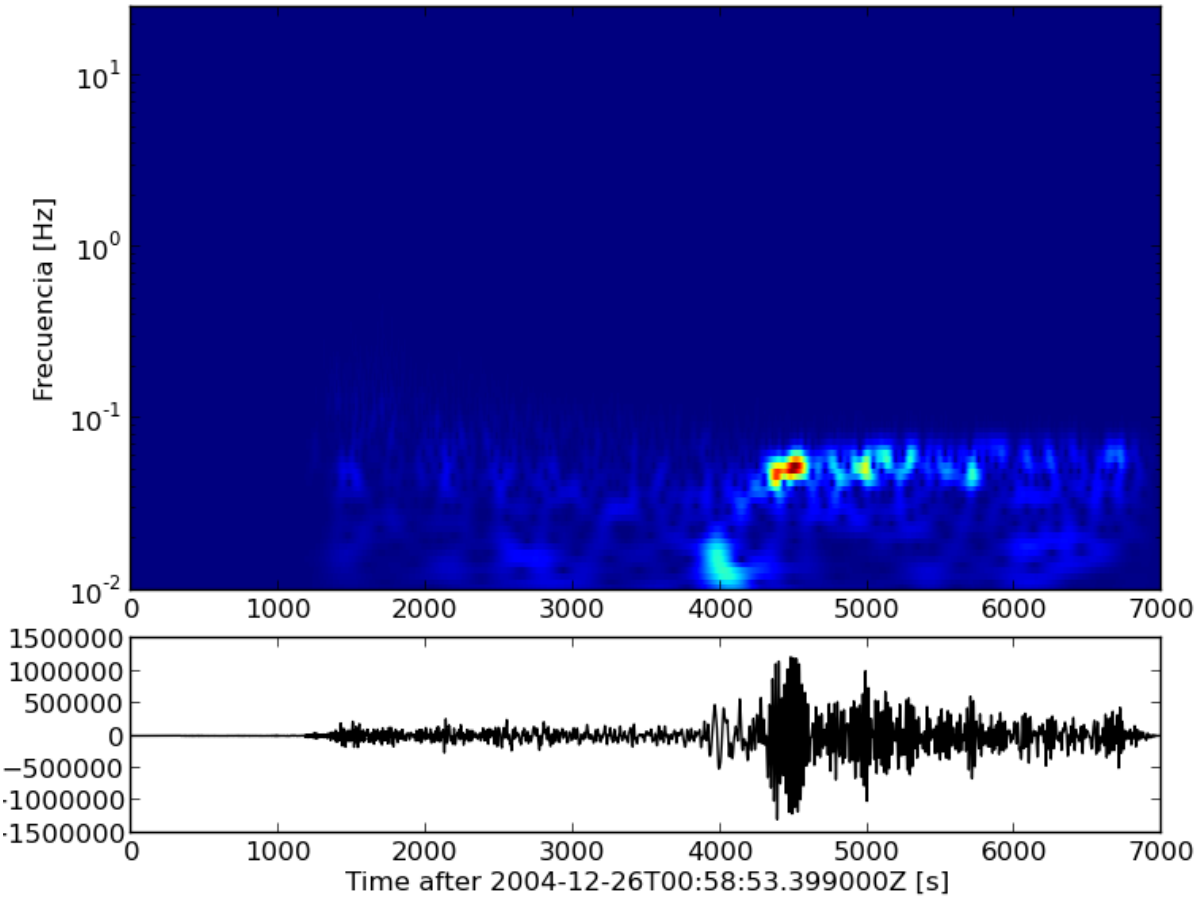


Figura 9.3: CWT de 2004361005853_XJ_TRANB.HHZ.

Capítulo 10

Respuesta del Instrumento

Para remover la respuesta del instrumento a cualquier registro sismológico solo necesitamos conocer los polos, ceros y la sensibilidad del sensor en V/m/s.

10.1. Visualización respuesta en frecuencia

En esta sección mostraremos como pueden visualizar la respuesta en frecuencia de cada sensor conociendo solo sus polos y ceros. Como opción podemos aplicar un factor de ganancia. En el siguiente script se definen los polos y ceros que tengamos del fabricante del sensor.

La función "pazToFreqResp(poles,zeros,scale_fac,t_samp,nfft)" Entrega la respuesta del instrumento en frecuencia.(1-3) En nuestro caso ingresamos los polos y ceros de un sensor Guralp 40T y, factor de escala. (4) Luego el valor t_samp da el tiempo de muestreo en segundos. (5) nfft corresponde al número de puntos utilizados para el fast fourier transform. Ver Figura 10.1

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> from obspy.signal import pazToFreqResp

>>> # Guralp 40T HHZ
>>> poles= [-0.02365+0.02365j,-0.02365-0.02365j,-180+0j,-160+0j,-80+0j]
>>> zeros= [0+0j,0+0j]
>>> scale_fac = 1
>>> h, f = pazToFreqResp(poles, zeros, scale_fac, 0.005, 16384, freq=True)

>>> plt.figure()

>>> plt.subplot(211)
>>> plt.loglog(f, abs(h))
>>> plt.ylabel('Amplitud')

>>> plt.subplot(212)
>>> phase = np.unwrap(np.arctan2(-h.imag, h.real))
>>> plt.semilogx(f, phase)
>>> plt.xlabel('Frequencia [Hz]')
```

```
>>> plt.ylabel('Fase [radianes]')

>>> plt.suptitle('Guralp 40T')
>>> plt.subplots_adjust(wspace=0.3)
>>> plt.show()
```

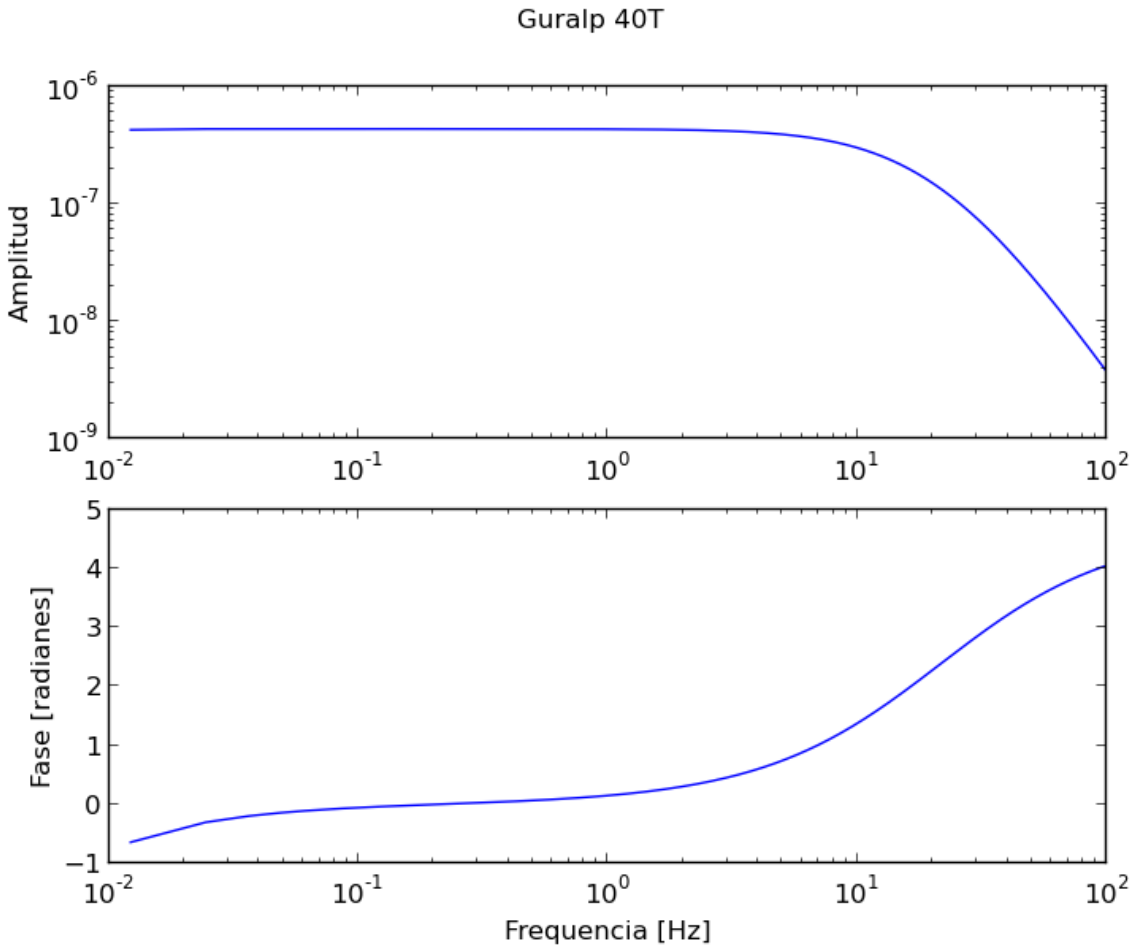


Figura 10.1: Respuesta en frecuencia de sensor Guralp 40T.

10.2. Remover Respuesta del Instrumento

En esta sección procederemos a remover la respuesta del instrumento al sismograma "XJ_TRANB.HHZ" considerando que fue registrado por un Guralp 40T.

Además de los polos y ceros que requerimos del fabricante, necesitaremos la sensibilidad del sensor, en nuestro caso será 2304 V/m/s.

Luego, el código queda como se explica a continuación. Ver Figura 10.2

```
>>> import numpy as np
>>> from obspy.core import read, UTCDateTime, Stream, Trace
>>> from obspy.signal import bandpass, highpass, cornFreq2Paz

>>> st = read('2004361005853_XJ_TRANB.HHZ')
>>> tr = st[0]

>>> paz_40T = {'gain':1,
            'poles': [-0.02365+0.02365j,-0.02365-0.02365j,-180+0j,-160+0j,-80+0j],
            'sensitivity': 2304.,
            'zeros': [0+0j,0+0j]}

>>> cornerfreq = 1./30
>>> damp=0.707
>>> sensitivity = 1.0

>>> paz_sim = cornFreq2Paz(cornerfreq, damp)
>>> paz_sim['sensitivity'] = sensitivity
>>> tr.simulate(paz_remove=paz_40T, paz_simulate=paz_sim,
    remove_sensitivity=True, simulate_sensitivity=True)
>>> tr.plot()
```

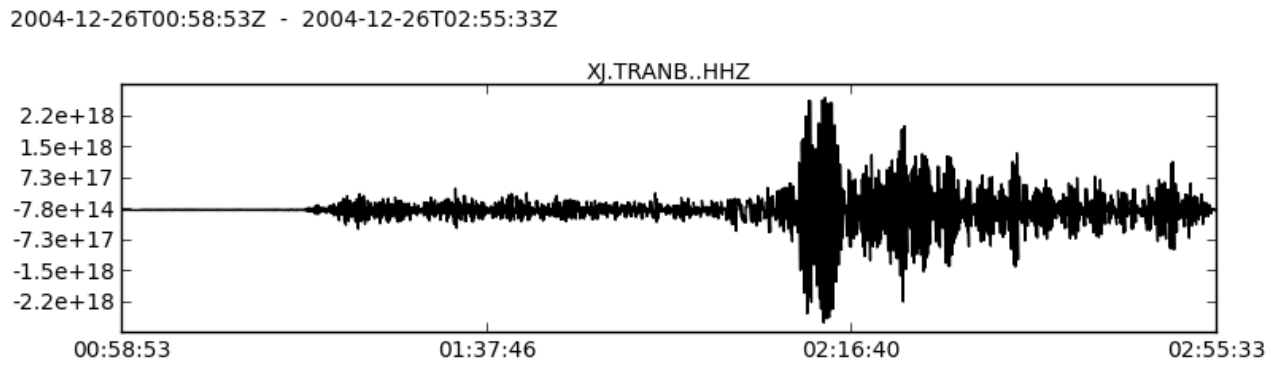


Figura 10.2: Corrección del sensor Guralp 40T a sismograma XJ.TRANB..HHZ.

Capítulo 11

Modelo de tiempos de viaje – TAUP

11.1. Tiempos de viaje

Para obtener los tiempos de viaje de fases sísmicas solo se debe utilizar la función ”getTravelTimes(delta, depth, model)“. El parámetro delta es la distancia entre el sensor y el epicentro en grados, depth es la profundidad en kilómetros del evento sísmico y el parámetro model corresponde al modelo de velocidades escogido. El modelo por defecto es ”iasp91”. Se puede seleccionar como opción “ak135”.

Esta función entregará como resultado las fases sísmicas en orden de llegada. Cada fase se asociará con 6 parámetros. El nombre de la fase sísmica, el tiempo de viaje en segundos, el ángulo de salida y 3 derivadas ($\frac{dT}{dh}$, $\frac{dT}{dD}$ y $\frac{d^2T}{dh^2}$).

Como ejemplo, calculemos los tiempos de viaje de todas las fases sísmicas que llegan a un sensor ubicado a 90° de un evento que ocurre a 25 km de profundidad considerando el modelo de velocidades “ias91”

```
>>> from obspy.taup.taup import getTravelTimes
>>> t = getTravelTimes(delta=90, depth=25.0, model='iasp91')
[{'d2T/dD2': -0.010935438,
  'dT/dD': 4.6607537,
  'dT/dh': -0.14797933,
  'phase_name': 'P',
  'take-off angle': 15.87395,
  'time': 777.26416},

         .
         .
         .                  ,

  {'d2T/dD2': -0.010966078,
  'dT/dD': -2.5771503,
  'dT/dh': -0.26564962,
  'phase_name': "S'S'ac",
  'take-off angle': 5.0057459,
  'time': 3182.7314}]
```

11.2. Ploteo tiempos de viaje

En Obspy también existe la posibilidad de graficar los tiempos de viaje de fases sísmicas con a partir de un modelo de velocidades. "travelTimePlot(min_degree, max_degree, npoints, phases, depth, model)" es la función que permite realizar esto. El parámetro min_degree y max_degree es la distancia en grados del eje x (0-360), npoints es el número de puntos del ploteo (1000), phases son las fases sísmicas que queremos plotear, depth es la profundidad del evento sísmico y model corresponde al modelo de velocidades.

En el siguiente ejemplo, plotiemos las fases sísmicas P y S para un evento que ocurre a 100 km de profundidad y modelo iasp91. Ver Figura 11.1

```
>>> from obspy.taup.taup import travelTimePlot
>>> travelTimePlot(min_degree=0, max_degree=180, phases=['P','S'],
    depth=35, model='iasp91')
```

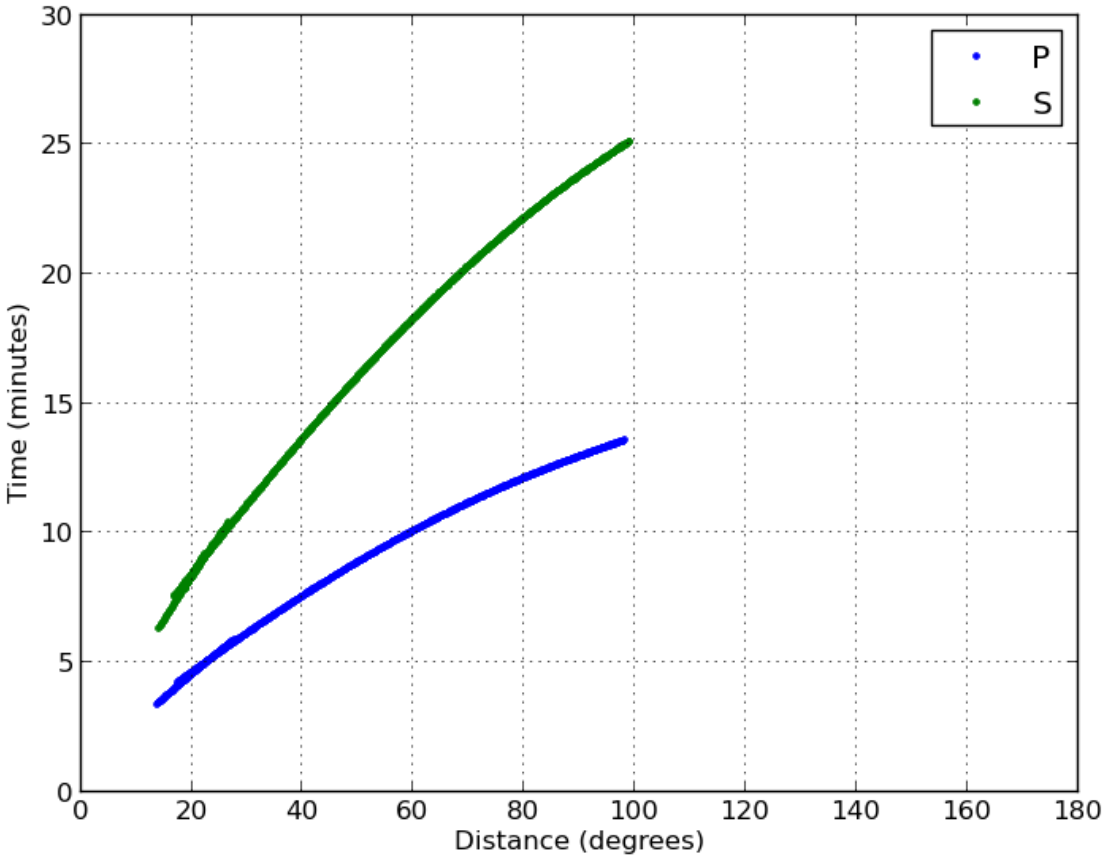


Figura 11.1: Curvas de tiempo de viaje de fases P y S utilizando modelo de velocidades IASP91.

11.3. Cliente de IRIS

11.3.1. Obtener serie de tiempo

La función para obtener datos directamente desde IRIS se hace simplemente ingresando lo siguiente:

Client.getWaveform(network, station, location, channel, starttime, endtime, quality='B')

(1) network es el parámetro que indica la red de la cual queremos bajar una o más series de tiempo. (2) station indica el código de la estación. (3) location se refiere al código de locación, generalmente 00 o 10. A veces es mejor utilizar "*" para ver todas las opciones disponibles. (4) En channel hay que ingresar los canales que deseamos bajar. Esto será relativo para cada estación y red sismológica. (5-6) se ingresan los tiempos de inicio y final que deseamos de nuestra serie de tiempo en UTC. (7) quality es un parámetro de calidad. Por defecto está "B" que indica bajar la mejor serie disponible.

Como ejemplo descargaremos los datos del terremoto de Japón M_w 8.3 del 24 de Mayo de 2013 a 609 km de profundidad. Ver Figura 11.2

```
>>> from obspy.iris import Client
>>> from obspy import UTCDateTime

>>> client = Client()
>>> t1 = UTCDateTime('2013-05-24T05:40:00')
>>> t2 = UTCDateTime('2013-05-24T06:20:00')
>>> st = client.getWaveform('II', 'ERM', '10', 'BH*', t1, t2)

>>> print st
>>> st.plot()
```

Para una mayor información, ingresar a Wilber 3 de IRIS (http://www.iris.edu/wilber3/find_event)

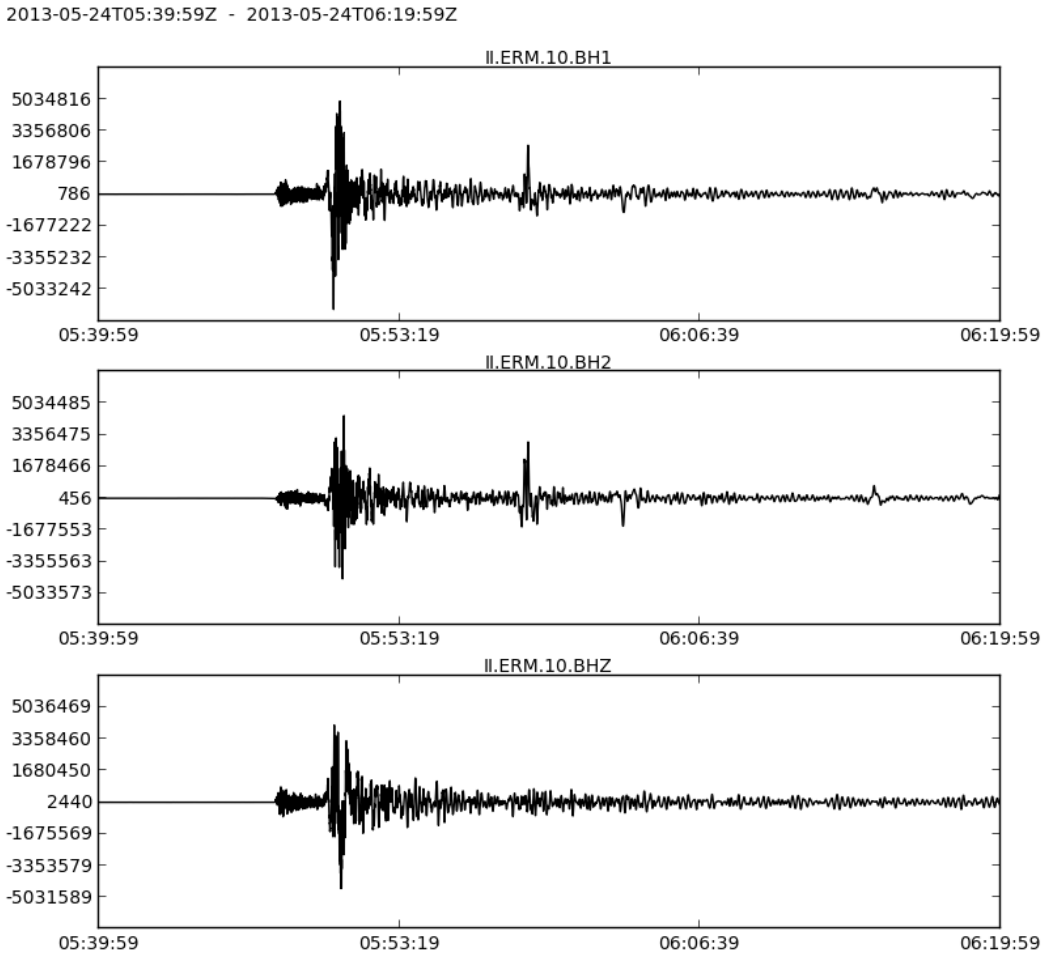


Figura 11.2: Serie de tiempo descargado desde cliente IRIS. Terremoto de Japón M_w 8.3 del 24 de Mayo de 2013 a 609 km de profundidad.

Capítulo 12

Mecanismos focales – Beachballs

En Obspy existe una poderosa herramienta para plotear los mecanismos focales. Solo se deben conocer los parámetros strike (medido en sentido horario desde el norte geográfico), dip (en sentido horario perpendicular al strike de 0 a 90) y rake (de -180 a 180, positivo hacia arriba) ó las 6 componentes independientes del tensor de estrés (M11,M22,M33,M12,M13,M23).

Veamos el siguiente ejemplo, considerando una falla inversa pura, con un manteo de 20° y strike 0°. Ver Figura 12.1

```
>>> from obspy.imaging.beachball import Beachball
>>> mt = [0, 20, 90]
>>> Beachball(mt, size=200, linewidth=2, facecolor='k')
```

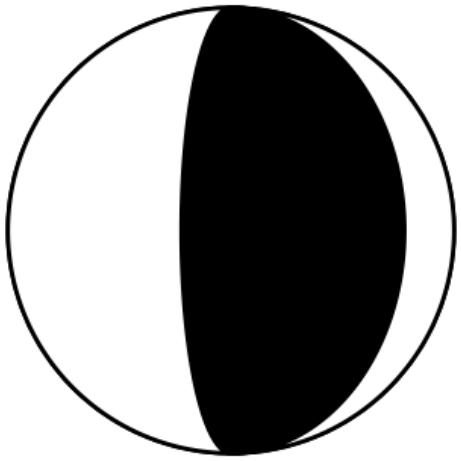


Figura 12.1: Mecanismo inverso puro.

Capítulo 13

Correlación cruzada

Para realizar una correlación cruzada, debemos utilizar la función "xcorr(tr1, tr2, shift_len, full_xcorr)". (1-2) tr1 y tr2 corresponden a las trazas de las señales sísmicas. (3) shift_len es el tamaño de la ventana de desplazamiento hacia la derecha e izquierda. (4) si full_xcorr es verdadero, entonces se entregará como salida la la posición del valor máximo de la correlación, su valor correspondiente y un vector con todos los valores de la función de correlación.

Como ejemplo, utilicemos las serie de ruido sísmico de la red IPOC en el norte de Chile del año 2007 y día juliano 273 "PB03.2007.278.mseed - PB04.2007.278.mseed". Utilizaremos una ventana de desplazamiento de 1000 segundos. Ver Figura 13.1

```
>>> from obspy.core import read, UTCDateTime
>>> from obspy.signal.cross_correlation import xcorr
>>> import matplotlib.pyplot as plt
>>> import numpy as np

>>> st1 = read('PB03_2007_278.mseed')
>>> st2 = read('PB04_2007_278.mseed')
>>> tr1 = st1[0]
>>> tr2 = st2[0]
>>> print 'Frecuencia muestreo serie 1 es: ', tr1.stats.sampling_rate
>>> print 'Frecuencia muestreo serie 2 es: ', tr2.stats.sampling_rate

>>> time_lag=1000  # tiempo de retardo

>>> shift_len = int(tr1.stats.sampling_rate * time_lag)
>>> index, value, fct = xcorr(tr1, tr2, shift_len, full_xcorr=True)
>>> t = np.arange(-time_lag,time_lag+time_lag/(tr1.stats.sampling_rate * time_lag),time

>>> plt.plot(t,fct)
>>> plt.xlim(-time_lag,time_lag)
>>> plt.xlabel('Time lag [s]')
>>> plt.ylabel('Coeficiente de correlacion')
>>> plt.suptitle('Correlacion cruzada entre dos registros sismicos')
>>> plt.show()
```

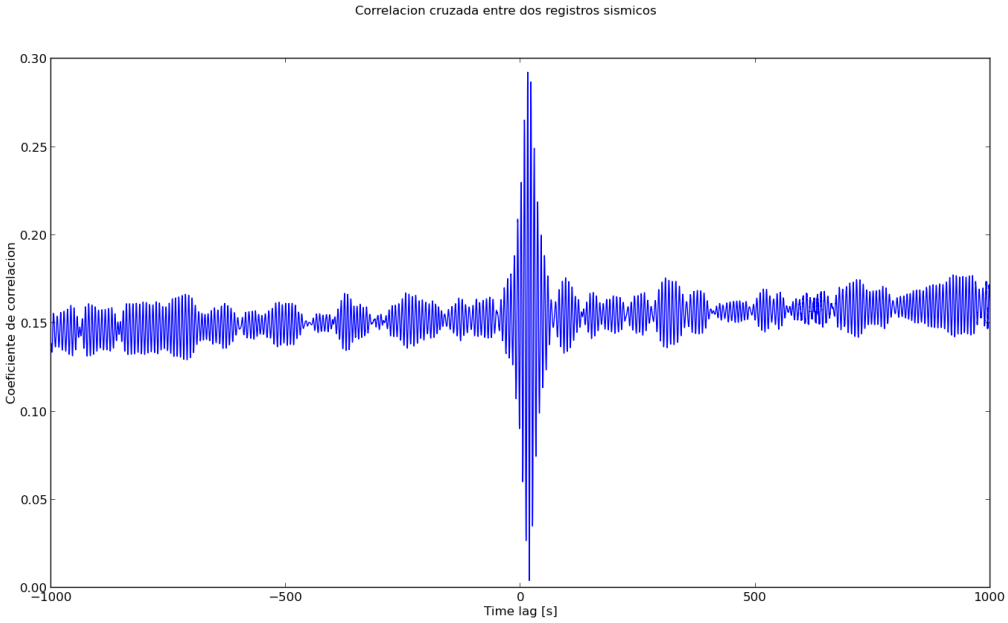


Figura 13.1: Correlación cruzada entre estación PB03 y PB04 de la red IPOC en año 2007.