《OBSPY 中文教程》

V 1.0

2020 / 04 / 12

说明：

Obspy 是一个针对地震领域开发的 python 库。其目的是使地震学软件包和工作流程的发展更为便利，为地震学进入更大的科学 python 生态系统建桥铺路。Obspy 对地震学界内通常使用的几乎所有文件格式提供读写支持，它取代了大量的文件格式转换工具；在这样广泛的输入/输出支持的基础上，在处理程序时使用地震学家之间交流的专业术语。Obspy 整合世界范围内了获取地震数据中心所发布数据的方法。它集成了大量地震学界所用的专有库，并用易用的接口统一了所有功能的调用。Python 高效便捷易懂的特点，基于其开发的 obspy 也非常容易理解使用，是一个很好的帮助科研的软件工具。

这里将 obspy 官网的（http://docs.obspy.org/）的教程翻译为中文文档（英文教程链接：http://docs.obspy.org/tutorial/index.html）。受个人学识所限，翻译教程中的错误和不当还请谅解和指正。

如有问题请联系：penghuiw@outlook.com

本教程不尝试全面，涵盖每一个功能。相反，它介绍了许多 ObsPy 最值得注意的功能，并会给你一些关于库使用的建议。

目录

1. UTCDateTime（世界标准时间数据）

ObsPy 中的所有绝对时间值均由 UTCDateTime 类一致处理。因为精度问题它基于高精度 POSIX 时间戳，而不是 Python datetime 类。

1) 初始化:

```
>>> from obspy.core import UTCDateTime
>>> UTCDateTime("2012-09-07T12:15:00")
UTCDateTime(2012, 9, 7, 12, 15)
>>> UTCDateTime(2012, 9, 7, 12, 15, 0)
UTCDateTime(2012, 9, 7, 12, 15)
>>> UTCDateTime(1347020100.0)
UTCDateTime(2012, 9, 7, 12, 15)
```

多数情况下不需要担心时区问题，但是支持以下格式：

```
>>> UTCDateTime("2012-09-07T12:15:00+02:00")
UTCDateTime(2012, 9, 7, 10, 15)
```

2) 属性访问

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

3) 处理时间差

```
>>> time = UTCDateTime("2012-09-07T12:15:00")
>>> print(time + 3600)
2012-09-07T13:15:00.000000Z
>>> time2 = UTCDateTime(2012, 1, 1)
>>> print(time - time2)
21644100.0
```

2. Reading Seismograms（读取地震数据）

obspy 可以将各种格式的地震数据（例如 SAC，MiniSEED，GSE2，SEISAN，Q 等）使用 read（）函数的对象导入到 Stream 对象中。
Stream 类似一个包含多个 Trace 对象（例如无间隙连续时间序列和相关的对象头/元信息）的列表。

每个 Trace 对象都有一个名为 data 的属性，对应实际时间序列的 NumPy ndarray，以及 stats 属性。(Stats 对象是一个包含所有元信息的字典。)Stats 中的 starttime 和 endtime 也都是 UTCDateTime 对象。

以下示例演示如何将单个 GSE2 格式的地震图文件读入 ObsPy Stream 中。 在给定的地震记录中只有一个 Trace：

```
>>> from obspy import read
>>> st =
read('http://examples.obspy.org/RJOB_061005_072159.ehz.new')
>>> print(st)
1 Trace(s) in Stream:
.RJOB..Z | 2005-10-06T07:21:59.849998Z - 2005-10-
06T07:24:59.844998Z | 200.0 Hz, 36000 samples
>>> len(st)
1
>>> tr = st[0] # assign first and only trace to new variable
>>> print(tr)
.RJOB..Z | 2005-10-06T07:21:59.849998Z - 2005-10-
06T07:24:59.844998Z | 200.0 Hz, 36000 samples
```

1) 访问元信息

在每个 Trace 中可通过 stats 关键词对地震元数据（描述实际波形数据的数据）进行访问：

```
>>> print(tr.stats)
network:
station: RJOB
location:
channel: Z
starttime: 2005-10-06T07:21:59.849998Z
endtime: 2005-10-06T07:24:59.844998Z
sampling_rate: 200.0
delta: 0.005
npts: 36000
calib: 0.0948999971151
_format: GSE2
gse2: AttribDict({'instype': ' ', 'datatype': 'CM6', 'hang': -
```

```
1.0, 'auxid'
>>> tr.stats.station
'RJOB'
>>> tr.stats.gse2.datatype
'CM6'
```

2) 访问波形数据

在每个 Trace 中可以通过 data 关键词对实际波形数据进行检索：

```
>>> tr.data
array([-38, 12, -4, ..., -14, -3, -9])
>>> tr.data[0:3]
array([-38, 12, -4])
>>> len(tr)
36000
```

3) 数据浏览

Stream 对象提供了一个用于快速预览波形的 plot()方法(需要 obspy.imaging 模块)：

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

3. Waveform Plotting Tutorial（波形绘制）

该教程中我们使用两个不同的 obspy Stream 对象：一个是只包含一个 Trace 的 singlechannel，另一个是包含三分量波形数据的 threechannel

```
>>> from obspy.core import read
>>> singlechannel =
read('https://examples.obspy.org/COP.BHZ.DK.2009.050')
>>> print(singlechannel)
1 Trace(s) in Stream:
DK.COP..BHZ | 2009-02-19T00:00:00.025100Z - 2009-02-
19T23:59:59.975100Z | 20.0 Hz, 1728000 samples
>>> threechannels =
read('https://examples.obspy.org/COP.BHE.DK.2009.050')
>>> threechannels +=
read('https://examples.obspy.org/COP.BHN.DK.2009.050')
>>> threechannels +=
read('https://examples.obspy.org/COP.BHZ.DK.2009.050')
>>> print(threechannels)
3 Trace(s) in Stream:
```

```
DK.COP..BHE | 2009-02-19T00:00:00.035100Z - 2009-02-
19T23:59:59.985100Z | 20.0 Hz, 1728000 samples
DK.COP..BHN | 2009-02-19T00:00:00.025100Z - 2009-02-
19T23:59:59.975100Z | 20.0 Hz, 1728000 samples
DK.COP..BHZ | 2009-02-19T00:00:00.025100Z - 2009-02-
19T23:59:59.975100Z | 20.0 Hz, 1728000 samples
```

1) 基本绘制

使用 Stream 对象中的 plot（）方法显示图示。该图像的默认大小为 800*250 像素。可使用 size 属性调整其大小。

```
>>> singlechannel.plot()
```

2) 自定义绘制

下例展示了如何调整图像颜色、坐标刻度值、格式、角度以及显示出的开始和结束时间。所有参数的设置可参考 plot（）方法的介绍文档。

```
>>> dt =singlechannel[0].stats.starttime
>>>.plot(color='red', number_of_ticks=7,
...          tick_rotation=5, tick_format='%I:%M %p',
...          starttime=dt + 60*60, endtime=dt + 60*60 + 120)
```

3) 保存图像

图像可以通过 outfile 参数保存到文件系统中。文件格式通过文件后缀自动识别，例如：png、pdf、ps、eps 和 svg。

```
>>> singlechannel.plot(outfile='singlechannel.png')
```

4) 绘制多通道图形

如果 Stream 对象中包含多个 Trace，每个 Trace 都可以在同一个窗口中复合显示。每个 Trace 的开始和结束时间保持一致，且 y 轴的范围相等。每附加一条图像都会使结果增高 250 像素。下例中通过使用 size 属性设置整幅图像大小。

```
>>> threechannels.plot(size=(800, 600))
```

5) 创建 One-Day 图像

一天的 Trace 图像可以通过设置 type 参数“dayplot”进行显示：

```
>>> singlechannel.plot(type='dayplot')
```

事件信息同样可以包含在图示中（实验性功能，版本可能会改变）。

```
>>> from obspy import read
>>> st =
read("https://examples.obspy.org/GR.BFO..LHZ.2012.108")
>>> st.filter("lowpass", freq=0.1, corners=2)
>>> st.plot(type="dayplot", interval=60,
right_vertical_labels=False,
... vertical_scaling_range=5e3, one_tick_per_line=True,
... color=['k', 'r', 'b', 'g'], show_y_UTC_label=False,
... events={'min_magnitude': 6.5})
```

6) 绘制一段记录

绘制 Stream 中的某一段记录可以通过设置 type 参数为“section”：

```
>>> stream.plot(type='section')
```

Obspy 头 trace.stats.distance(offset) 单位必须为米，或者地理位置 trace.stats.coordinates.latitude & trace.stats.coordinates.longitude 必须有定义（如果该部分与参数 ev_coord 一起以圆距离绘制（dist_degree = True））。更多信息参考 plot（）。

7) 绘图和颜色选项

调整一些参数可以改变波形图的外观。所有可调选项参考 plot（）方法。

8) 使用 Matplotlib 自定义绘图

就像下面这个简约的例子所示，使用 matplotlib 可以进行自定义绘制：

```
import matplotlib.pyplot as plt
from obspy import read
st = read()
tr = st[0]
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(tr.times("matplotlib"), tr.data, "b-")
ax.xaxis_date()
fig.autofmt_xdate()
plt.show()
```

4. Retrieving Data from Data Centers（从数据中心检索数据）

PS:此部分提供了使用 obspy 下载数据的推荐方式，但是由于数据中心和 web 服务在不断更新变化，所有有些建议可能变无效。

Note：最常见的应用为下载波形和整个或部分事件元信息。多数情况下你都可以用到 obspy.clients.fdsn 模块。它支持最大数量的数据集并使用最新的数据格式。虽然可以选择不同的模块，但是请确保你至少有一个可用模块。

1) FDSN 网络服务

基础 FDSN 网络服务

Table 1

可用数据类型：	格式：
Waveforms	MiniSEED and optionally others
Station Information	StationXML and Text
Event Information	QuakeML and Text

Note：并不是所有数据都有这三种数据格式，很多只提供一或两种。

如果想访问数据或者事件元信息，你很有可能会使用到 obspy.clients.fdsn 模块。它可以从任何一个部署了 FDSN 网络的数据中心访问请求数据服务。示例数据中心包含 IRIS/ORFEUS/INGV/ETH/GFZ/RESIF/...。一个突出的优势是其返回的数据使用最新的并可用于未来的数据格式。

FDSN Routing 网络服务

如果你不清楚某个数据中心有哪些数据，你可以使用 Obspy 支持的以下两种 routing 服务中的一种：.

1.The IRIS Federator.

2. The EIDAWS Routing Service.

更多使用细节可以参考 obspy.clients.fdsn 模块的说明。

FDSN 下载器：

如果你需要从一些数据中心下载大量的数据，obspy 提供了 Mass downloader。例如，您可以根据地理区域制定查询，ObsPy 将下载波形和相应的站点元信息，并生成完整的有基本质量控制的数据集以供研究使用。

2) ArcLink

Table 2

可用数据类型：	格式：
Waveforms	MiniSEED ， SEED
Station Information	Dataless SEED， SEED

ArcLink 是一种分布式数据请求协议，可用于访问 MiniSEED 和 SEED 格式的波形数据并关联元信息到无数据的 SEED 文件。你可以使用 obspy.clients.arclink 模型从 EIDA 访问数据，同时也可以使用 obspy.clients.fdsn 模块进行访问。

3) IRIS 网络服务

IRIS 网络服务包含各种数据类型和格式。IRIS（作为 FDSN 网络服务的补充）提供各种专用 Web 服务。Obspy 在 obspy.clients.iris 模块中提供了一些接口，如果您需要 SAC 极点和零点或 RESP 格式的响应信息，请使用此选项。 如果您只关心仪器响应，请使用 obspy.clients.fdsn 模块来请求包含相同信息的 StationXML 数据。

Table 3

IRIS 网络服务	等同的 obspy 模块
obspy.clients.iris.client.Client.traveltime()	obspy.taup
obspy.clients.iris.client.Client.distaz()	obspy.geodetics
obspy.clients.iris.client.Client.flinnengdahl()	obspy.geodetics.flinengdahl.FlinnEngdahl

4) Earthworm Wave 服务

Table 4

可用数据类型：	格式：
Waveforms	Custom Format

使用 obspy.clients.earthworm 模块访问 Earthmorm 系统的数据。

5) NERIES 网络服务

建议使用 obspy.clients.fdsn 模块进行访问

6) NEIC

Table 5

可用数据类型：	格式：
Waveforms	MiniSEED

连续波形数据（CWB）是一个针对由 NEIC“Edge”系统处理的地震波形数据的存储库。使用 obspy.clients.neic 模块向其请求数据。

7) Syngine 服务

Table 6

可用数据类型：	格式：
Waveforms	MiniSEED and zipped SAC files

模块：obspy.clients.syngine

5. Filtering Seismograms（滤波）

以下脚本展示了如何对一个地震记录进行滤波。示例使用一个零相移低通（1Hz）滤波器处理波形。

可用的滤波器有：带通，带阻，低通，高通

```
import numpy as np
import matplotlib.pyplot as plt
import obspy
# Read the seismogram
st =
obspy.read("https://examples.obspy.org/RJOB_061005_072159.ehz.
new")
# There is only one trace in the Stream object, let's work on
that trace...
tr = st[0]
# Filtering with a lowpass on a copy of the original Trace
tr_filt = tr.copy()
tr_filt.filter('lowpass', freq=1.0, corners=2, zerophase=True)
# Now let's plot the raw and filtered data...
t = np.arange(0, tr.stats.npts / tr.stats.sampling_rate,
tr.stats.delta)
plt.subplot(211)
plt.plot(t, tr.data, 'k')
plt.ylabel('Raw Data')
plt.subplot(212)
plt.plot(t, tr_filt.data, 'k')
plt.ylabel('Lowpassed Data')
plt.xlabel('Time [s]')
plt.suptitle(tr.stats.starttime)
plt.show()
```

6. Downsampling Seismograms（下采样）

下面的脚本展示了如何对地震记录进行下采样。目前支持简单的整数下采样。如果未明确禁用，则在下采样之前应先进行低通滤波以防产生混叠。为了比较，也绘制了过滤过的未下采样数据。应用的处理步骤记录在每个 Trace 的 trace.stats.processing 中。请注意处理开始的相位，因为默认情况下应用的滤波器不是零相位类型。这可以通过手动应用零相滤波器并在下下采样期间停用自动滤波来避免（no_filter = True）。

```
import numpy as np
import matplotlib.pyplot as plt
import obspy
```

```
# Read the seismogram
st =
obspy.read("https://examples.obspy.org/RJOB_061005_072159.ehz.
new")
# There is only one trace in the Stream object, let's work on
that trace...
tr = st[0]
# Decimate the 200 Hz data by a factor of 4 to 50 Hz. Note
that this
# automatically includes a lowpass filtering with corner
frequency 20 Hz.
# We work on a copy of the original data just to demonstrate
the effects of
# downsampling.
tr_new = tr.copy()
tr_new.decimate(factor=4, strict_length=False)
# For comparison also only filter the original data (same
filter options as in
# automatically applied filtering during downsampling, corner
frequency
# 0.4 * new sampling rate)
tr_filt = tr.copy()
tr_filt.filter('lowpass', freq=0.4 * tr.stats.sampling_rate /
4.0)
# Now let's plot the raw and filtered data...
t = np.arange(0, tr.stats.npts / tr.stats.sampling_rate,
tr.stats.delta)
t_new = np.arange(0, tr_new.stats.npts /
tr_new.stats.sampling_rate,tr_new.stats.delta)
plt.plot(t, tr.data, 'k', label='Raw', alpha=0.3)
plt.plot(t, tr_filt.data, 'b', label='Lowpassed', alpha=0.7)
plt.plot(t_new, tr_new.data, 'r',
label='Lowpassed/Downsampled', alpha=0.7)
plt.xlabel('Time [s]')
plt.xlim(82, 83.5)
plt.suptitle(tr.stats.starttime)
plt.legend()
plt.show()
```

7. Merging Seismograms（合并）

下面的例子显示了如何合并然后绘制三个重叠的地震图，最长的一个被认为是正确的。请参阅 merge（）方法的文档。

```
import numpy as np
import matplotlib.pyplot as plt
import obspy
# Read in all files starting with dis.
st = obspy.read("https://examples.obspy.org/dis.G.SCZ.__.BHE")
st +=
obspy.read("https://examples.obspy.org/dis.G.SCZ.__.BHE.1")
st +=
obspy.read("https://examples.obspy.org/dis.G.SCZ.__.BHE.2")
# sort
st.sort(['starttime'])
# start time in plot equals 0
dt = st[0].stats.starttime.timestamp
# Go through the stream object, determine time range in julian
seconds
# and plot the data with a shared x axis
ax = plt.subplot(4, 1, 1) # dummy for tying axis
for i in range(3):
    plt.subplot(4, 1, i + 1, sharex=ax)
    t = np.linspace(st[i].stats.starttime.timestamp -
dt,st[i].stats.endtime.timestamp - dt,st[i].stats.npts)
plt.plot(t, st[i].data)
# Merge the data together and show plot in a similar way
st.merge(method=1)
plt.subplot(4, 1, 4, sharex=ax)
t = np.linspace(st[0].stats.starttime.timestamp - dt,
st[0].stats.endtime.timestamp - dt,
st[0].stats.npts)
plt.plot(t, st[0].data, 'r')
plt.show()
```

8. Beamforming - FK Analysis （FK 分析）

下列代码展示了如何使用 obspy 进行 FK 分析。这些数据来自 ••••

我们应用下列设置执行 array_processing()：

- 设置慢度网络拐点-3.0：3.0s/km，步长 sl_s=0.03
- 窗口长 1s，步长 0.05
- 数据经过 1~8Hz 的带通滤波，不使用预白化
- semb_thres 和 vel_thres 设置为无穷小的数字，不得更改。
- 时间戳 timestamp 设置为“mlabday”他可以被我们的绘图程序直接读取
- stime 和 etime 定为 UTCDate Time 格式

输出被存储在 out 中。

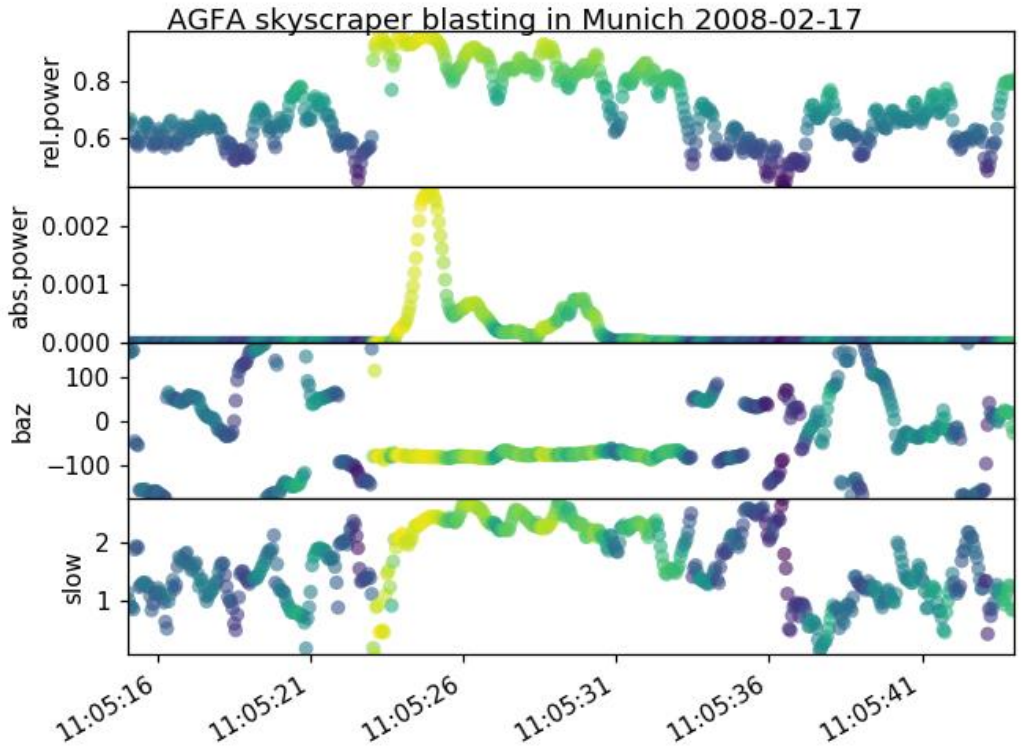
下半部分展示了如何绘制输出。我们使用 array_processing()输出的 out，它是包含了

时间戳、相对功率、绝对功率、反量和慢度的 numpy ndarray。色条代表相对功率。

```
import matplotlib.pyplot as plt
import matplotlib.dates as mdates

import obspy
from obspy.core.util import AttribDict
from obspy.imaging.cm import obspy_sequential
from obspy.signal.invsim import corn_freq_2_paz
from obspy.signal.array_analysis import array_processing


# Load data
st = obspy.read("https://examples.obspy.org/agfa.mseed")

# Set PAZ and coordinates for all 5 channels
st[0].stats.paz = AttribDict({
    'poles': [(-0.03736 - 0.03617j), (-0.03736 + 0.03617j)],
    'zeros': [0j, 0j],
    'sensitivity': 205479446.68601453,
    'gain': 1.0})
st[0].stats.coordinates = AttribDict({
    'latitude': 48.108589,
    'elevation': 0.450000,
    'longitude': 11.582967})

st[1].stats.paz = AttribDict({
    'poles': [(-0.03736 - 0.03617j), (-0.03736 + 0.03617j)],
    'zeros': [0j, 0j],
    'sensitivity': 205479446.68601453,
    'gain': 1.0})
st[1].stats.coordinates = AttribDict({
    'latitude': 48.108192,
    'elevation': 0.450000,
    'longitude': 11.583120})

st[2].stats.paz = AttribDict({
    'poles': [(-0.03736 - 0.03617j), (-0.03736 + 0.03617j)],
    'zeros': [0j, 0j],
    'sensitivity': 250000000.0,
    'gain': 1.0})
st[2].stats.coordinates = AttribDict({
    'latitude': 48.108692,
    'elevation': 0.450000,
    'longitude': 11.583414})
```

```
st[3].stats.paz = AttribDict({
    'poles': [(-4.39823 + 4.48709j), (-4.39823 - 4.48709j)],
    'zeros': [0j, 0j],
    'sensitivity': 222222228.10910088,
    'gain': 1.0})
st[3].stats.coordinates = AttribDict({
    'latitude': 48.108456,
    'elevation': 0.450000,
    'longitude': 11.583049})

st[4].stats.paz = AttribDict({
    'poles': [(-4.39823 + 4.48709j), (-4.39823 - 4.48709j), (-
2.105 + 0j)],
    'zeros': [0j, 0j, 0j],
    'sensitivity': 222222228.10910088,
    'gain': 1.0})
st[4].stats.coordinates = AttribDict({
    'latitude': 48.108730,
    'elevation': 0.450000,
    'longitude': 11.583157})


# Instrument correction to 1Hz corner frequency
paz1hz = corn_freq_2_paz(1.0, damp=0.707)
st.simulate(paz_remove='self', paz_simulate=paz1hz)

# Execute array_processing
stime = obspy.UTCDateTime("20080217110515")
etime = obspy.UTCDateTime("20080217110545")
kwargs = dict(
    # slowness grid: X min, X max, Y min, Y max, Slow Step
    sll_x=-3.0, slm_x=3.0, sll_y=-3.0, slm_y=3.0, sl_s=0.03,
    # sliding window properties
    win_len=1.0, win_frac=0.05,
    # frequency properties
    frqlow=1.0, frqhigh=8.0, prewhiten=0,
    # restrict output
    semb_thres=-1e9, vel_thres=-1e9, timestamp='mlabday',
    stime=stime, etime=etime
)
out = array_processing(st, **kwargs)

# Plot
```

```
labels = ['rel.power', 'abs.power', 'baz', 'slow']

xlocator = mdates.AutoDateLocator()
fig = plt.figure()
for i, lab in enumerate(labels):
    ax = fig.add_subplot(4, 1, i + 1)
    ax.scatter(out[:, 0], out[:, i + 1], c=out[:, 1],alpha=0.6,
               edgecolors='none', cmap=obspy_sequential)
    ax.set_ylabel(lab)
    ax.set_xlim(out[0, 0], out[-1, 0])
    ax.set_ylim(out[:, i + 1].min(), out[:, i + 1].max())
    ax.xaxis.set_major_locator(xlocator)

ax.xaxis.set_major_formatter(mdates.AutoDateFormatter(xlocator
))

fig.suptitle('AGFA skyscraper blasting in Munich %s' % (
    stime.strftime('%Y-%m-%d'), ))
fig.autofmt_xdate()
fig.subplots_adjust(left=0.15, top=0.95, right=0.95,
bottom=0.2, hspace=0)
plt.show()
```



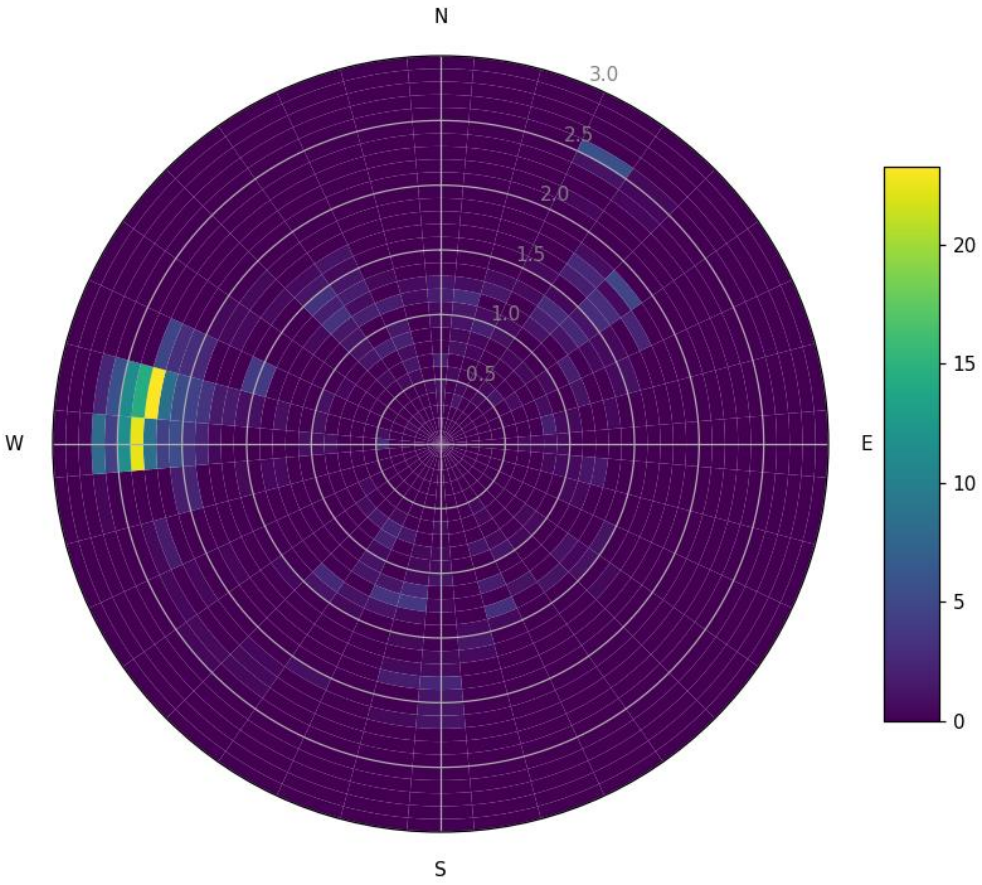
也可以使用极坐标图表示，其将网格箱中的相对功率相加，每个网格箱都由背调角和要分析的部分信号的慢度定义。反向量从北向开始顺时针计数，慢度限制可以手动设定。

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colorbar import ColorbarBase
from matplotlib.colors import Normalize

import obspy
from obspy.core.util import AttribDict
from obspy.imaging.cm import obspy_sequential
from obspy.signal.invsim import corn_freq_2_paz
from obspy.signal.array_analysis import array_processing


# Load data
st = obspy.read("https://examples.obspy.org/agfa.mseed")

# Set PAZ and coordinates for all 5 channels
st[0].stats.paz = AttribDict({
    'poles': [(-0.03736 - 0.03617j), (-0.03736 + 0.03617j)],
    'zeros': [0j, 0j],
    'sensitivity': 205479446.68601453,
    'gain': 1.0})
st[0].stats.coordinates = AttribDict({
    'latitude': 48.108589,
    'elevation': 0.450000,
    'longitude': 11.582967})

st[1].stats.paz = AttribDict({
    'poles': [(-0.03736 - 0.03617j), (-0.03736 + 0.03617j)],
    'zeros': [0j, 0j],
    'sensitivity': 205479446.68601453,
    'gain': 1.0})
st[1].stats.coordinates = AttribDict({
    'latitude': 48.108192,
    'elevation': 0.450000,
    'longitude': 11.583120})

st[2].stats.paz = AttribDict({
    'poles': [(-0.03736 - 0.03617j), (-0.03736 + 0.03617j)],
    'zeros': [0j, 0j],
    'sensitivity': 250000000.0,
    'gain': 1.0})
st[2].stats.coordinates = AttribDict({
    'latitude': 48.108692,
    'elevation': 0.450000,
```

```
    'longitude': 11.583414})

st[3].stats.paz = AttribDict({
    'poles': [(-4.39823 + 4.48709j), (-4.39823 - 4.48709j)],
    'zeros': [0j, 0j],
    'sensitivity': 222222228.10910088,
    'gain': 1.0})
st[3].stats.coordinates = AttribDict({
    'latitude': 48.108456,
    'elevation': 0.450000,
    'longitude': 11.583049})

st[4].stats.paz = AttribDict({
    'poles': [(-4.39823 + 4.48709j), (-4.39823 - 4.48709j), (-
2.105 + 0j)],
    'zeros': [0j, 0j, 0j],
    'sensitivity': 222222228.10910088,
    'gain': 1.0})
st[4].stats.coordinates = AttribDict({
    'latitude': 48.108730,
    'elevation': 0.450000,
    'longitude': 11.583157})


# Instrument correction to 1Hz corner frequency
paz1hz = corn_freq_2_paz(1.0, damp=0.707)
st.simulate(paz_remove='self', paz_simulate=paz1hz)

# Execute array_processing
kwargs = dict(
    # slowness grid: X min, X max, Y min, Y max, Slow Step
    sll_x=-3.0, slm_x=3.0, sll_y=-3.0, slm_y=3.0, sl_s=0.03,
    # sliding window properties
    win_len=1.0, win_frac=0.05,
    # frequency properties
    frqlow=1.0, frqhigh=8.0, prewhiten=0,
    # restrict output
    semb_thres=-1e9, vel_thres=-1e9,
    stime=obspy.UTCDateTime("20080217110515"),
    etime=obspy.UTCDateTime("20080217110545")
)
out = array_processing(st, **kwargs)

# Plot
```

```
cmap = obspy_sequential

# make output human readable, adjust backazimuth to values
between 0 and 360
t, rel_power, abs_power, baz, slow = out.T
baz[baz < 0.0] += 360

# choose number of fractions in plot (desirably 360 degree/N
is an integer!)
N = 36
N2 = 30
abins = np.arange(N + 1) * 360. / N
sbins = np.linspace(0, 3, N2 + 1)

# sum rel power in bins given by abins and sbins
hist, baz_edges, sl_edges = \
    np.histogram2d(baz, slow, bins=[abins, sbins],
weights=rel_power)

# transform to radian
baz_edges = np.radians(baz_edges)

# add polar and colorbar axes
fig = plt.figure(figsize=(8, 8))
cax = fig.add_axes([0.85, 0.2, 0.05, 0.5])
ax = fig.add_axes([0.10, 0.1, 0.70, 0.7], polar=True)
ax.set_theta_direction(-1)
ax.set_theta_zero_location("N")

dh = abs(sl_edges[1] - sl_edges[0])
dw = abs(baz_edges[1] - baz_edges[0])

# circle through backazimuth
for i, row in enumerate(hist):
    bars = ax.bar(left=(i * dw) * np.ones(N2),
                  height=dh * np.ones(N2),
                  width=dw, bottom=dh * np.arange(N2),
                  color=cmap(row / hist.max()))

ax.set_xticks(np.linspace(0, 2 * np.pi, 4, endpoint=False))
ax.set_xticklabels(['N', 'E', 'S', 'W'])

# set slowness limits
```

```
ax.set_ylim(0, 3)
[i.set_color('grey') for i in ax.get_yticklabels()]
ColorbarBase(cax, cmap=cmap,
             norm=Normalize(vmin=hist.min(), vmax=hist.max()))

plt.show()
```

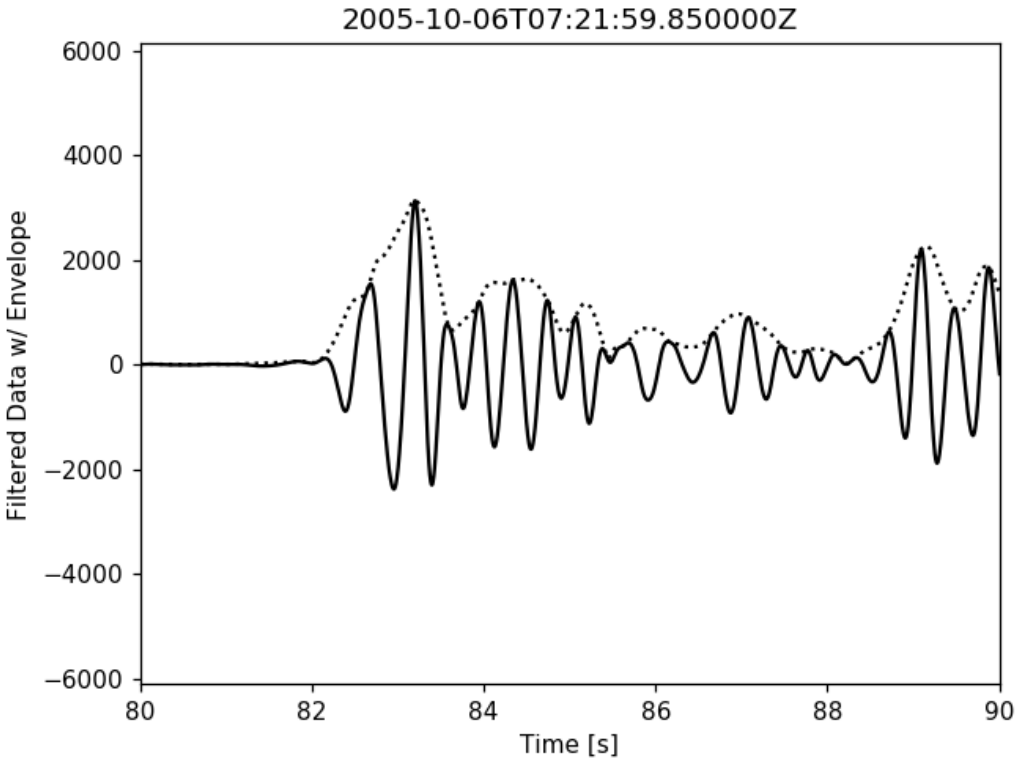


9. Seismogram Envelopes（信号包络）

以下脚本显示如何过滤地震数据并将其与其包络信号一起绘制图像。本示例使用零相移带通滤波器（1~3Hz）进行滤波。然后我们计算出包络并将其与 Trace 一起绘制。

```
import numpy as np
import matplotlib.pyplot as plt

import obspy
import obspy.signal
```

```
st =
obspy.read("https://examples.obspy.org/RJOB_061005_072159.ehz.
new")
data = st[0].data
npts = st[0].stats.npts
samprate = st[0].stats.sampling_rate

# Filtering the Stream object
st_filt = st.copy()
st_filt.filter('bandpass', freqmin=1, freqmax=3, corners=2,
zerophase=True)

# Envelope of filtered data
data_envelope = obspy.signal.filter.envelope(st_filt[0].data)

# The plotting, plain matplotlib
t = np.arange(0, npts / samprate, 1 / samprate)
plt.plot(t, st_filt[0].data, 'k')
plt.plot(t, data_envelope, 'k:')
plt.title(st[0].stats.starttime)
plt.ylabel('Filtered Data w/ Envelope')
plt.xlabel('Time [s]')
plt.xlim(80, 90)
plt.show()
```

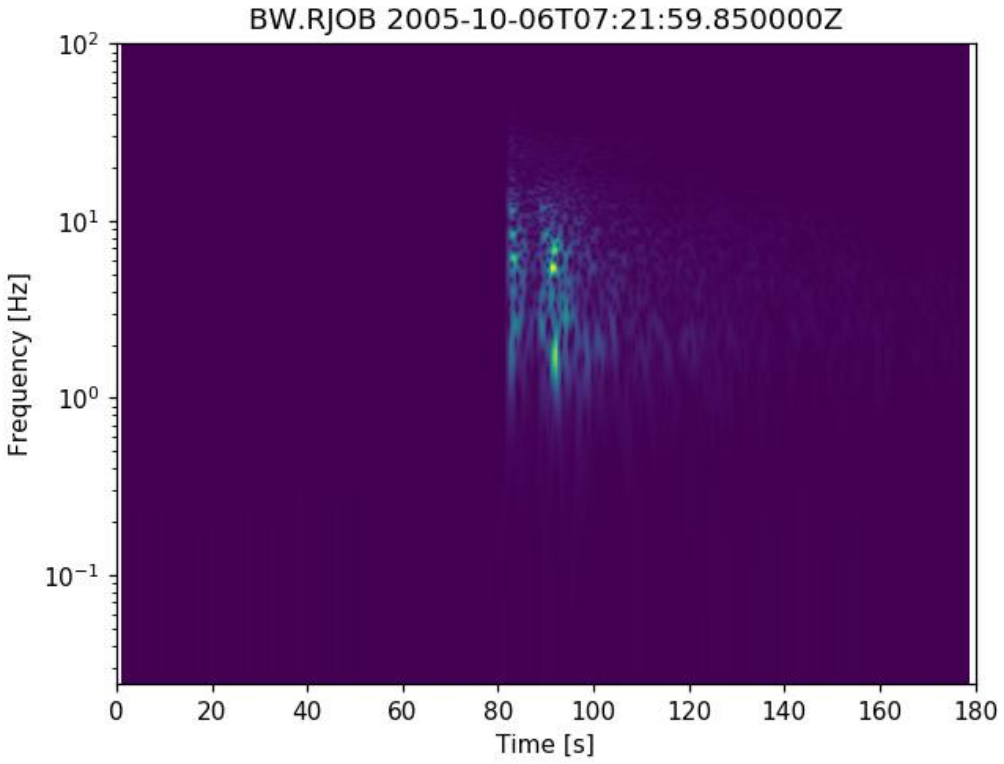


10. Plotting_Spectrograms（绘制频谱图）

下列代码展示了如何使用 obspy Stream 对象绘制频谱图。其中的很多选项都可以自定义，参考 spectrogram（）获取更多细节。比如，可以很轻松的通过调用 matplotlib.crm 中预定义的色彩图用来调整所需，这里提供几个教程：

- http://www.physics.ox.ac.uk/users/msshin/science/code/matplotlib_cm/
- http://matplotlib.org/examples/color/colormaps_reference.html
- https://wiki.scipy.org/Cookbook/Matplotlib/Show_colormaps

```
import obspy

st =
obspy.read("https://examples.obspy.org/RJOB_061005_072159
.ehz.new")
st.spectrogram(log=True, title='BW.RJOB ' +
str(st[0].stats.starttime))
```



11. Trigger/Picker Tutorial（触发器/拾取器教程）

教程中所用测试数据在这里 trigger_data.zip. 下载。

触发器的实现如[Withers1998]. 中所述。参考[Trnkoczy2012]. 求取 STA / LTA 类型触发器的正确触发参数信息。请注意使用 obspy 中提供的 Stream.trigger 和 Trace.trigger 作为便利方法。

1) 读取波形数据

使用 read（）函数读取数据文件到 Trace 对象。

```
from obspy.core import read
st = read("https://examples.obspy.org/ev0_6.a01.gse2")
st = st.select(component="Z")
tr = st[0]
```

数据类型可以被自动检测。这个教程中最重要的是掌握 Trace 中的各个属性：

Table 7

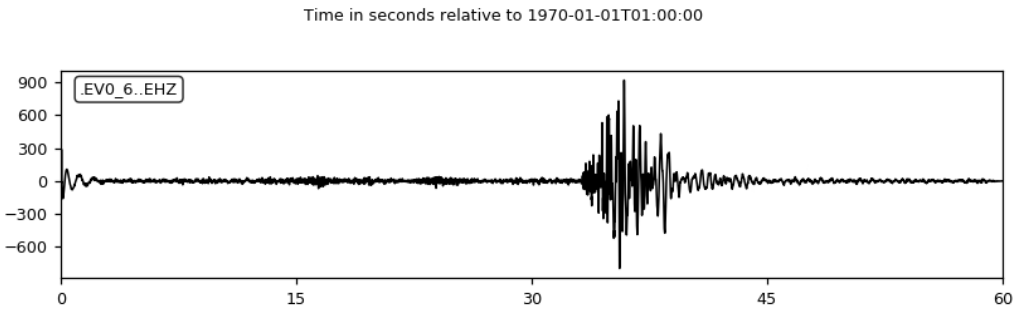
tr.data	Numpy.ndarray
tr.stats	包含头信息的类似字典的类
tr.stats.sampling_rate	采样率
tr.stats.npts	采样的数量

下面的例子展示了数据文件头信息，并绘制了数据图像：

```
>>>print(tr.stats)
         network:
         station: EV0_6
        location:
         channel: EHZ
       starttime: 1970-01-01T01:00:00.000000Z
         endtime: 1970-01-01T01:00:59.995000Z
   sampling_rate: 200.0
           delta: 0.005
            npts: 12000
           calib: 1.0
         _format: GSE2
            gse2: AttribDict({'instype': '      ', 'datatype':
'CM6', 'hang': 0.0, 'auxid': '    ', 'vang': -1.0, 'calper':
1.0})
```

使用 Trace 对象的 plot（）方法绘制：

```
>>>tr.plot(type="relative")
```



2) 可用方法

在加载完数据后，我们可以将波形数据传递给 obspy.signal.trigger 中定义的以下触发器例程：

Table 8

recursive_sta_lta(a, nsta, nlta)	递归 STA / LTA
carl_sta_trig(a, nsta, nlta, ratio, quiet)	计算 carlSTAtrig 特征函数
classic_sta_lta(a, nsta, nlta)	计算标准 STA / LTA
delayed_sta_lta(a, nsta, nlta)	延迟的 STA / LTA
z_detect(a, nsta)	Z 检测器
pk_baer(reltrc, samp_int, tdownmax, ...)	P_picker 包装
ar_pick(a, b, c, samp_rate, f1, f2, lta_p, ...)	使用 AR-AIC + STA / LTA 算法检测 P 和 S 的到达

每个函数的帮助信息都有 HTML 格式信息或通常的 Python 方式查询：

```
>>>from obspy.signal.trigger import classic_sta_lta
>>>help(classic_sta_lta)
```

设置触发器主要经由以下两个步骤：1)计算特征函数,2)根据特征函数的值设置 picks

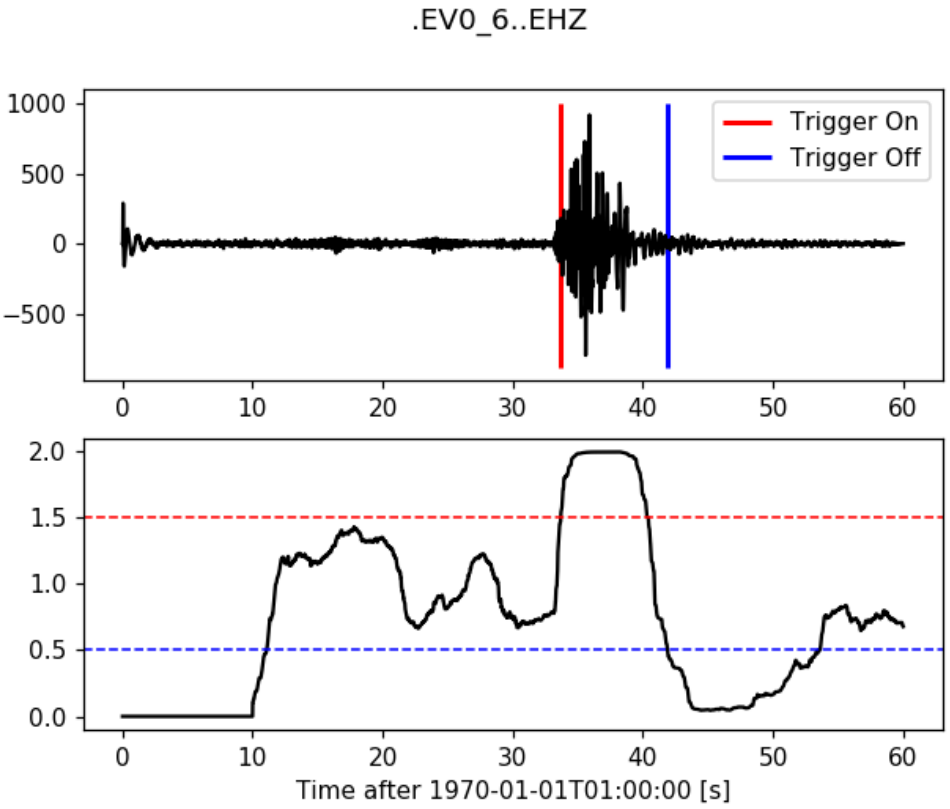
3) 触发器的例子

所有的例子中，读取数据加载模型的命令如下：

```
>>>from obspy.core import read
>>>from obspy.signal.trigger import plot_trigger
>>>trace= read("https://examples.obspy.org/ev0_6.a01.gse2")[0]
>>>df = trace.stats.sampling_rate
```

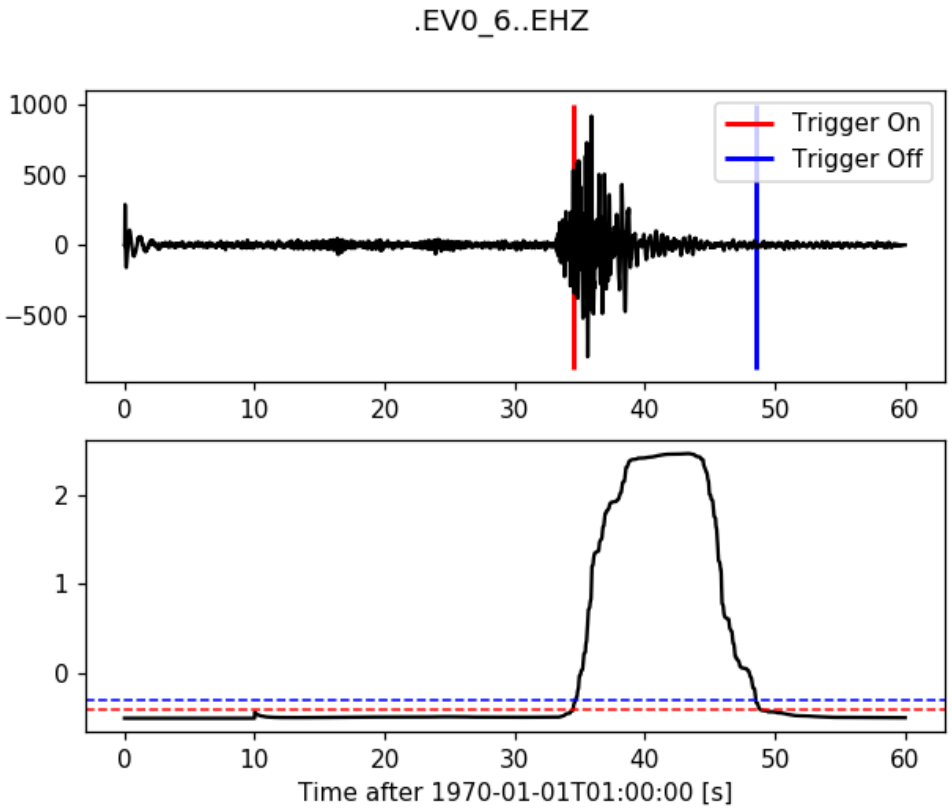
a) 典型 STA/LTA

```
>>>from obspy.signal.trigger import classic_sta_lta
>>>cft = classic_sta_lta(trace.data, int(5 * df), int(10 *
df))
>>>plot_trigger(trace, cft, 1.5, 0.5)
```



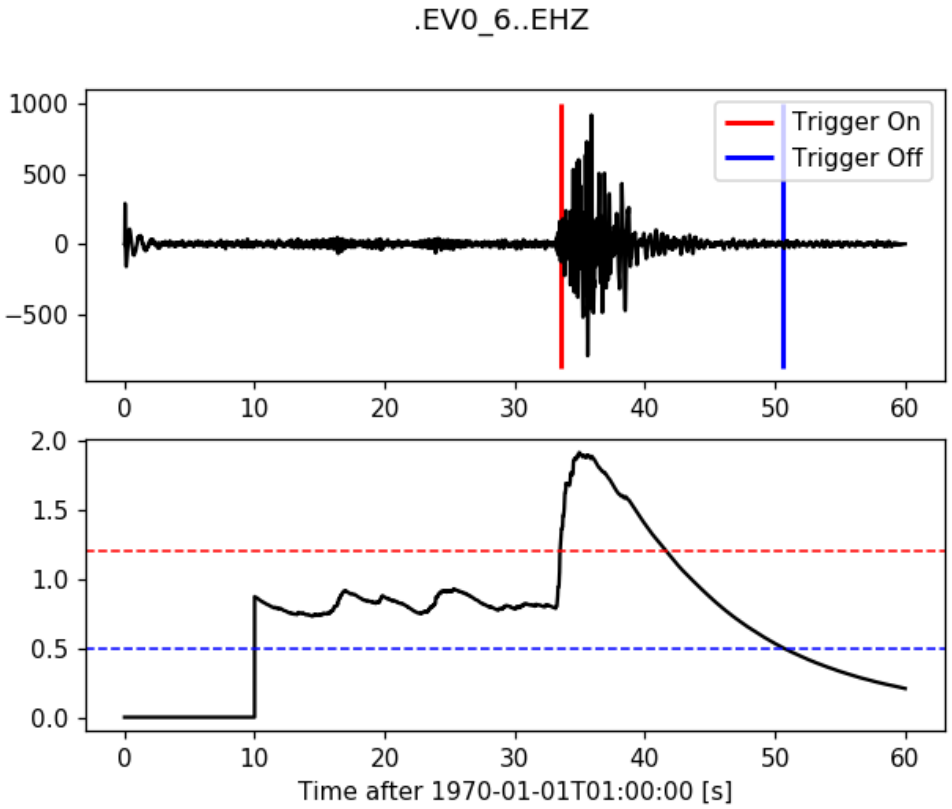
b) Z 探测

```
>>>from obspy.signal.trigger import z_detect
>>>cft = z_detect(trace.data, int(10 * df))
>>>plot_trigger(trace, cft, -0.4, -0.3)
```



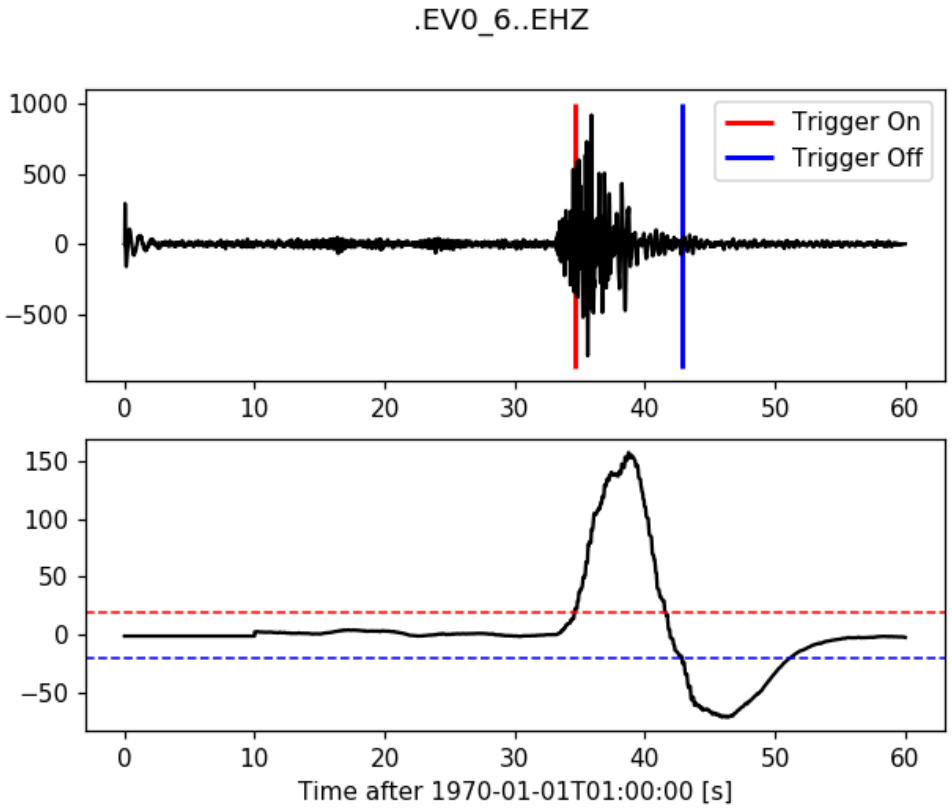
c) 递归 STA/LTA

```
>>>from obspy.signal.trigger import recursive_sta_lta
>>>cft = recursive_sta_lta(trace.data, int(5 * df), int(10 *
df))
>>>plot_trigger(trace, cft, 1.2, 0.5)
```



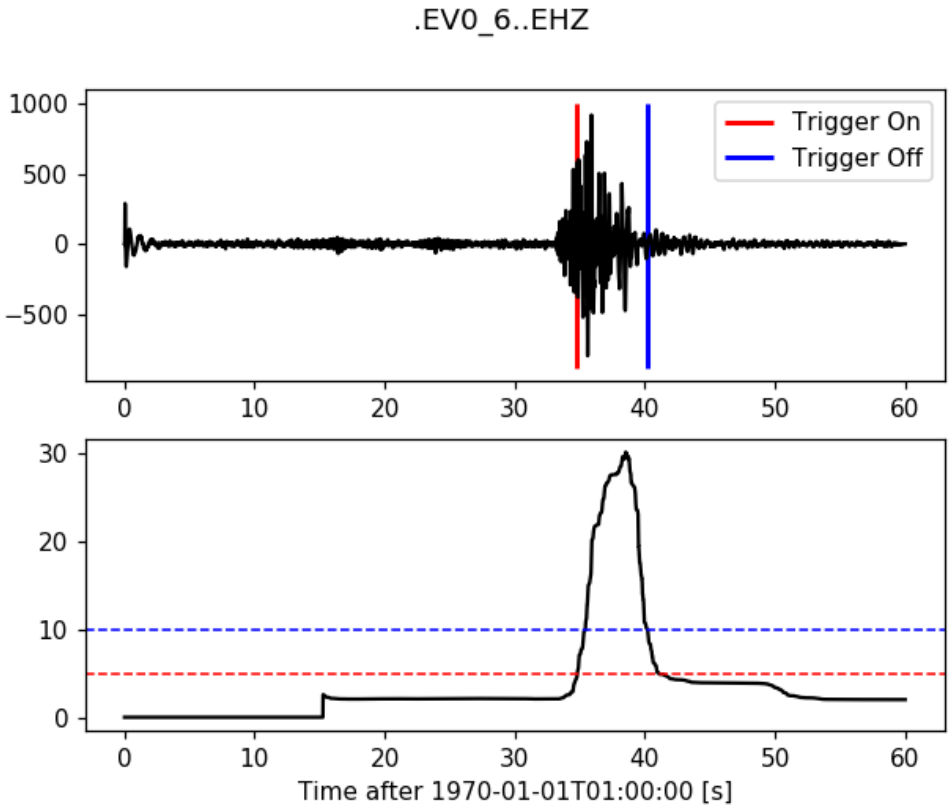
d) 卡尔-STA-触发

```
>>>from obspy.signal.trigger import carl_sta_trig
>>>cft = carl_sta_trig(trace.data, int(5 * df), int(10 * df),
0.8, 0.8)
>>>plot_trigger(trace, cft, 20.0, -20.0)
```



e) 延迟 STA/LTA

```
>>>from obspy.signal.trigger import delayed_sta_lta
>>>cft = delayed_sta_lta(trace.data, int(5 * df), int(10 *
df))
>>>plot_trigger(trace, cft, 5, 10)
```



4) 网络联合触发示例

这个例子中展示了一个由 4 个台站组成的网络联合触发器，每个触发器使用递归 STA/LTA。波形数据持续四分钟，包含四个事件。其中两个很容易识别（MI 1-2），剩下的两个（MI <= 0）需要调整好合适的触发器设定才能被检测到.

首先我们将所有波形数据整合于一个 Stream 对象中，使用的数据能从我们的网络服务器中获取：

```
>>>from obspy.core import Stream, read
>>>st = Stream()
>>>files = ["BW.UH1..SHZ.D.2010.147.cut.slist.gz",
         "BW.UH2..SHZ.D.2010.147.cut.slist.gz",
         "BW.UH3..SHZ.D.2010.147.cut.slist.gz",
         "BW.UH4..SHZ.D.2010.147.cut.slist.gz"]
>>>for filename in files:
st += read("https://examples.obspy.org/" + filename)
```

在经过带通滤波之后，对所有数据进行联合 tiggering。使用递归 STA/LTA，触发器参数设置：时间窗 0.5~10 秒，开阈值 3.5，关阈值 1，例子中每个台站的权重相同，联合总阈值为 3。对于更复杂的网络，可以为每个台站/信道分配不同的权重。为了保留源数据，我们使用源 Stream 的副本进行处理：

```
>>>st.filter('bandpass', freqmin=10, freqmax=20)  # optional
prefiltering
>>>from obspy.signal.trigger import coincidence_trigger
```

```
>>>st2 = st.copy()
>>>trig = coincidence_trigger("recstalta", 3.5, 1, st2, 3,
sta=0.5, lta=10)
```

使用 pprint 打印结果显示如下：

```
>>>from pprint import pprint
>>>pprint(trig)
[{'coincidence_sum': 4.0,
  'duration': 4.5299999713897705,
  'stations': ['UH3', 'UH2', 'UH1', 'UH4'],
  'time': UTCDateTime(2010, 5, 27, 16, 24, 33, 190000),
  'trace_ids': ['BW.UH3..SHZ', 'BW.UH2..SHZ', 'BW.UH1..SHZ',
                'BW.UH4..SHZ']},
 {'coincidence_sum': 3.0,
  'duration': 3.4400000572204459,
  'stations': ['UH2', 'UH3', 'UH1'],
  'time': UTCDateTime(2010, 5, 27, 16, 27, 1, 260000),
  'trace_ids': ['BW.UH2..SHZ', 'BW.UH3..SHZ', 'BW.UH1..SHZ']},
 {'coincidence_sum': 4.0,
  'duration': 4.7899999618530273,
  'stations': ['UH3', 'UH2', 'UH1', 'UH4'],
  'time': UTCDateTime(2010, 5, 27, 16, 27, 30, 490000),
  'trace_ids': ['BW.UH3..SHZ', 'BW.UH2..SHZ', 'BW.UH1..SHZ',
                'BW.UH4..SHZ']}]
```

使用当前这些设定触发器报告了三个事件，提供每个（可能的）事件的开始时间和持续时间。此外还提供台站名和 trace ID 的列表，其按台站触发时间排序，这可以帮助我们对事件发生的位置有一个初始的粗略的想法。我们可以设定“details = True”来访问更多信息。

```
>>>st2 = st.copy()
>>>trig = coincidence_trigger("recstalta", 3.5, 1, st2, 3,
sta=0.5, lta=10,details=True)
```

为了显示清晰，我们这里只展示了结果中的第一项：

```
>>>pprint(trig[0])
{'cft_peak_wmean': 19.561900329259956,
 'cft_peaks': [19.535644192544272,
               19.872432918501264,
               19.622171410201297,
               19.217352795792998],
 'cft_std_wmean': 5.4565629691954713,
 'cft_stds': [5.292458320417178,
              5.6565387957966404,
              5.7582248973698507,
              5.1190298631982163],
 'coincidence_sum': 4.0,
 'duration': 4.5299999713897705,
```

```
 'stations': ['UH3', 'UH2', 'UH1', 'UH4'],
 'time': UTCDateTime(2010, 5, 27, 16, 24, 33, 190000),
 'trace_ids': ['BW.UH3..SHZ', 'BW.UH2..SHZ', 'BW.UH1..SHZ',
'BW.UH4..SHZ']}
```

在此，提供了关于单个台站触发器的特征函数的峰值和标准偏差等一些附加信息。此外也计算有二者的加权平均值。这些值可以帮助我们区分某些有问题的网络触发器。

5) 含有相似检测的高级网络联合触发器示例

这个例子是普通网络联合触发器的扩展。已知事件的波形可以被用于检测单个台站触发器的波形的相似性。如果相似度超过了相应的相似性阈值，即使联合值不及设定的最小值，该事件触发也会被记录到结果列表。使用这种方法，可以在某个台站有非常相似的波形记录时完成探测即使其他台站由于某些原因未能探测到触发（例如台站临时停电或者噪声水平高等因素）。可以为任何台站提供随意数量的波形模板。由于必要的交叉相关计算时间会显著增加。在该例中我们使用两个三成分组合事件模板应用于公共网络触发器的垂直向。

```
>>>from obspy.core import Stream, read, UTCDateTime
>>>st = Stream()
>>>files = ["BW.UH1..SHZ.D.2010.147.cut.slist.gz",
         "BW.UH2..SHZ.D.2010.147.cut.slist.gz",
         "BW.UH3..SHZ.D.2010.147.cut.slist.gz",
         "BW.UH3..SHN.D.2010.147.cut.slist.gz",
         "BW.UH3..SHE.D.2010.147.cut.slist.gz",
         "BW.UH4..SHZ.D.2010.147.cut.slist.gz"]
>>>for filename in files:
    st += read("https://examples.obspy.org/" + filename)
>>>st.filter('bandpass', freqmin=10, freqmax=20)  # optional
prefiltering
```

这里我们为单个台站设置一个事件模板的字典。指定时间为 P 波的精确到时，事件持续时间（包括 S 波）约 2.5 秒。在 UH3 台站我们使用两个三分量事件模板数据，在 UH1 台站我们仅使用一个事件模板的垂直向数据。

```
>>>times = ["2010-05-27T16:24:33.095000", "2010-05-
27T16:27:30.370000"]
>>>event_templates = {"UH3": []}
>>>for t in times:
    t = UTCDateTime(t)
    st_ = st.select(station="UH3").slice(t, t + 2.5)
    event_templates["UH3"].append(st_)
>>>t = UTCDateTime("2010-05-27T16:27:30.574999")
>>>st_ = st.select(station="UH1").slice(t, t + 2.5)
>>>event_templates["UH1"] = [st_]
```

下一步提供事件模板波形和相似度阈值。注意重合总和设置是 4，我们手动设定只使用相同站点重合值为 1 的垂直分量。

```
>>>from obspy.signal.trigger import coincidence_trigger
>>>st2 = st.copy()
```

```
>>>trace_ids = {"BW.UH1..SHZ": 1,
              "BW.UH2..SHZ": 1,
              "BW.UH3..SHZ": 1,
              "BW.UH4..SHZ": 1}
>>>similarity_thresholds = {"UH1": 0.8, "UH3": 0.7}
>>>trig = coincidence_trigger("classicstalta", 5, 1, st2, 4,
sta=0.5,
                           lta=10, trace_ids=trace_ids,
                           event_templates=event_templates,

similarity_threshold=similarity_thresholds)
```

现在的结果包含两个事件触发，其并不超过设置的最小重合阈值但是和至少一个事件模板的相似度超过设定阈值。在检查事件触发器时，请注意 1.0 的值，我们为此示例提取了事件模板。

```
>>>from pprint import pprint
>>>pprint(trig)
[{'coincidence_sum': 4.0,
  'duration': 4.1100001335144043,
  'similarity': {'UH1': 0.9414944738498271, 'UH3': 1.0},
  'stations': ['UH3', 'UH2', 'UH1', 'UH4'],
  'time': UTCDateTime(2010, 5, 27, 16, 24, 33, 210000),
  'trace_ids': ['BW.UH3..SHZ', 'BW.UH2..SHZ', 'BW.UH1..SHZ',
'BW.UH4..SHZ']},
 {'coincidence_sum': 3.0,
  'duration': 1.9900000095367432,
  'similarity': {'UH1': 0.65228204570577764, 'UH3':
0.72679293429214198},
  'stations': ['UH3', 'UH1', 'UH2'],
  'time': UTCDateTime(2010, 5, 27, 16, 25, 26, 710000),
  'trace_ids': ['BW.UH3..SHZ', 'BW.UH1..SHZ', 'BW.UH2..SHZ']},
 {'coincidence_sum': 3.0,
  'duration': 1.9200000762939453,
  'similarity': {'UH1': 0.89404458774338103, 'UH3':
0.74581409371425222},
  'stations': ['UH2', 'UH1', 'UH3'],
  'time': UTCDateTime(2010, 5, 27, 16, 27, 2, 260000),
  'trace_ids': ['BW.UH2..SHZ', 'BW.UH1..SHZ', 'BW.UH3..SHZ']},
 {'coincidence_sum': 4.0,
  'duration': 4.0299999713897705,
  'similarity': {'UH1': 1.0, 'UH3': 1.0},
  'stations': ['UH3', 'UH2', 'UH1', 'UH4'],
  'time': UTCDateTime(2010, 5, 27, 16, 27, 30, 510000),
  'trace_ids': ['BW.UH3..SHZ', 'BW.UH2..SHZ', 'BW.UH1..SHZ',
'BW.UH4..SHZ']}]
```

6) 拾取器示例

a) Baer Picker

对于 pk_bear(),输入单位秒，输出为样本。

```
>>>from obspy.core import read
>>>from obspy.signal.trigger import pk_baer
>>>trace =
read("https://examples.obspy.org/ev0_6.a01.gse2")[0]
>>>df = trace.stats.sampling_rate
>>>p_pick, phase_info = pk_baer(trace.data, df,
                             20, 60, 7.0, 12.0, 100, 100)
>>>print(p_pick)
6894
>>print(phase_info)
EPU3
>>>print(p_pick / df)
34.47
```

生成值 34.47 EPU3 含义是 P 波选择 34.47s 相位信息 EPU3。

b) AR Picker

对于 ar_pick（），输入和输出单位都是秒。

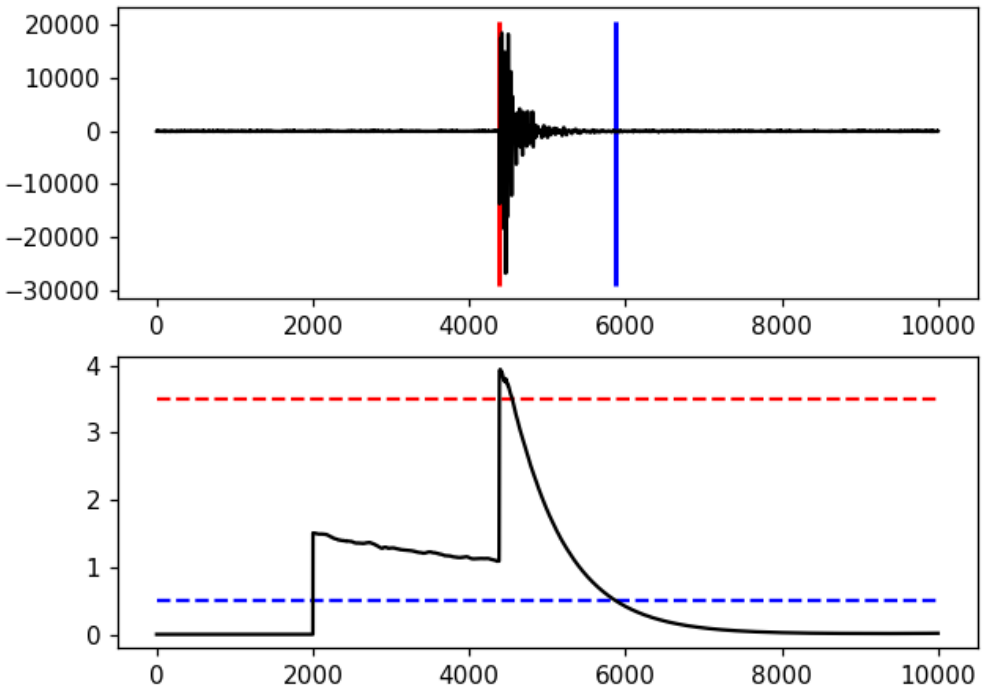
```
>>>from obspy.core import read
>>>from obspy.signal.trigger import ar_pick
>>>tr1 =
read('https://examples.obspy.org/loc_RJOB20050801145719850.z.g
se2')[0]
>>>tr2 =
read('https://examples.obspy.org/loc_RJOB20050801145719850.n.g
se2')[0]
>>>tr3 =
read('https://examples.obspy.org/loc_RJOB20050801145719850.e.g
se2')[0]
>>>df = tr1.stats.sampling_rate
>>>p_pick, s_pick = ar_pick(tr1.data, tr2.data, tr3.data, df,
                         1.0, 20.0, 1.0, 0.1, 4.0, 1.0, 2, 8,
0.1, 0.2)
>>>print(p_pick)
30.6350002289
>>>print(s_pick)
31.2800006866
```

输出的 30.6350002289 和 31.2800006866 意思是在 30.64s 的 P 波和在 31.28s 的 S 波被拾取出来。

7) 高级例程

这是一个更复杂的例子，数据通过 ArcLink 检索，结果被一步一步的绘制，代码如下：

```
import matplotlib.pyplot as plt

import obspy
from obspy.clients.arclink import Client
from obspy.signal.trigger import recursive_sta_lta,
trigger_onset


# Retrieve waveforms via ArcLink
client = Client(host="erde.geophysik.uni-muenchen.de",
port=18001,
                user="test@obspy.de")
t = obspy.UTCDateTime("2009-08-24 00:19:45")
st = client.get_waveforms('BW', 'RTSH', '', 'EHZ', t, t + 50)

# For convenience
tr = st[0]  # only one trace in mseed volume
df = tr.stats.sampling_rate

# Characteristic function and trigger onsets
cft = recursive_sta_lta(tr.data, int(2.5 * df), int(10. * df))
on_of = trigger_onset(cft, 3.5, 0.5)

# Plotting the results
ax = plt.subplot(211)
plt.plot(tr.data, 'k')
ymin, ymax = ax.get_ylim()
plt.vlines(on_of[:, 0], ymin, ymax, color='r', linewidth=2)
plt.vlines(on_of[:, 1], ymin, ymax, color='b', linewidth=2)
plt.subplot(212, sharex=ax)
plt.plot(cft, 'k')
plt.hlines([3.5, 0.5], 0, len(cft), color=['r', 'b'],
linestyle='--')
plt.axis('tight')
plt.show()
```



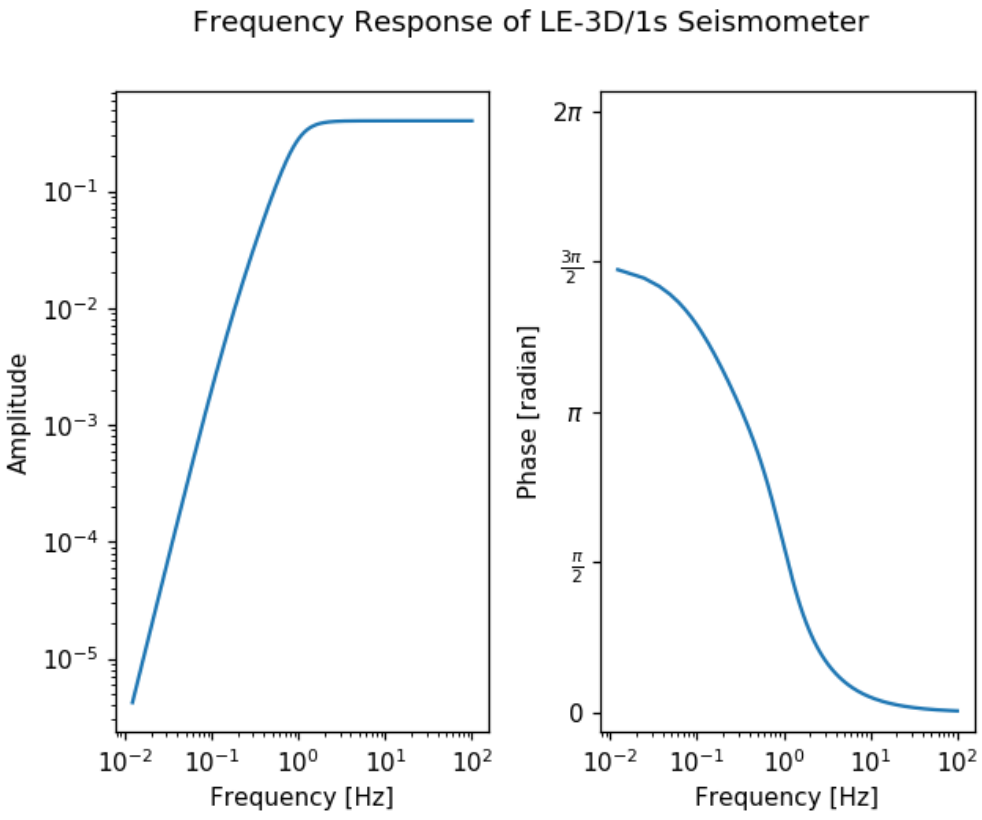
12. Poles and Zeros, Frequency Response（零极点和频率响应）

注意:使用read_inventory()读取元信息到Inventory对象(和相关的Network、Station、Channel、Response子对象）中。参考 Inventory.plot_response() or Response.plot()使用便捷的方法绘制波德图。

下面的代码展示了如何计算并显示 LE-3D/1S 地震仪的频率响应，其采样间隔 0.005s，傅里叶快速变换 16384 个点。我们希望相位角从 0 到 2π，而不是从-π到π的输出。

```
import numpy as np
import matplotlib.pyplot as plt

from obspy.signal.invsim import paz_to_freq_resp


poles = [-4.440 + 4.440j, -4.440 - 4.440j, -1.083 + 0.0j]
zeros = [0.0 + 0.0j, 0.0 + 0.0j, 0.0 + 0.0j]
scale_fac = 0.4

h, f = paz_to_freq_resp(poles, zeros, scale_fac, 0.005, 16384,
freq=True)

plt.figure()
```

```
plt.subplot(121)
plt.loglog(f, abs(h))
plt.xlabel('Frequency [Hz]')
plt.ylabel('Amplitude')

plt.subplot(122)
phase = 2 * np.pi + np.unwrap(np.angle(h))
plt.semilogx(f, phase)
plt.xlabel('Frequency [Hz]')
plt.ylabel('Phase [radian]')
# ticks and tick labels at multiples of pi
plt.yticks(
    [0, np.pi / 2, np.pi, 3 * np.pi / 2, 2 * np.pi],
    ['$0$', r'$\frac{\pi}{2}$', r'$\pi$', r'$\frac{3\pi}{2}$',
r'$2\pi$'])
plt.ylim(-0.2, 2 * np.pi + 0.2)
# title, centered above both subplots
plt.suptitle('Frequency Response of LE-3D/1s Seismometer')
# make more room in between subplots for the ylabel of right
plot
plt.subplots_adjust(wspace=0.3)
plt.show()
```



13. Seismometer Correction/Simulation（地震仪校准和仿真）

1) 计算滤波阶段的响应

a) 使用 StationXML 文件或者常用的 Inventory 对象

当使用 FDSN client 时，响应可以被直接附加在波形上，后续的删除可以使用 Stream.remove_response()：

```
from obspy import UTCDateTime
from obspy.clients.fdsn import Client

t1 = UTCDateTime("2010-09-3T16:30:00.000")
t2 = UTCDateTime("2010-09-3T17:00:00.000")
fdsn_client = Client('IRIS')
# Fetch waveform from IRIS FDSN web service into a ObsPy
stream object
# and automatically attach correct response
st = fdsn_client.get_waveforms(network='NZ', station='BFZ',
location='10',
                               channel='HHZ', starttime=t1,
endtime=t2,
                               attach_response=True)
# define a filter band to prevent amplifying noise during the
deconvolution
pre_filt = (0.005, 0.006, 30.0, 35.0)
st.remove_response(output='DISP', pre_filt=pre_filt)
```

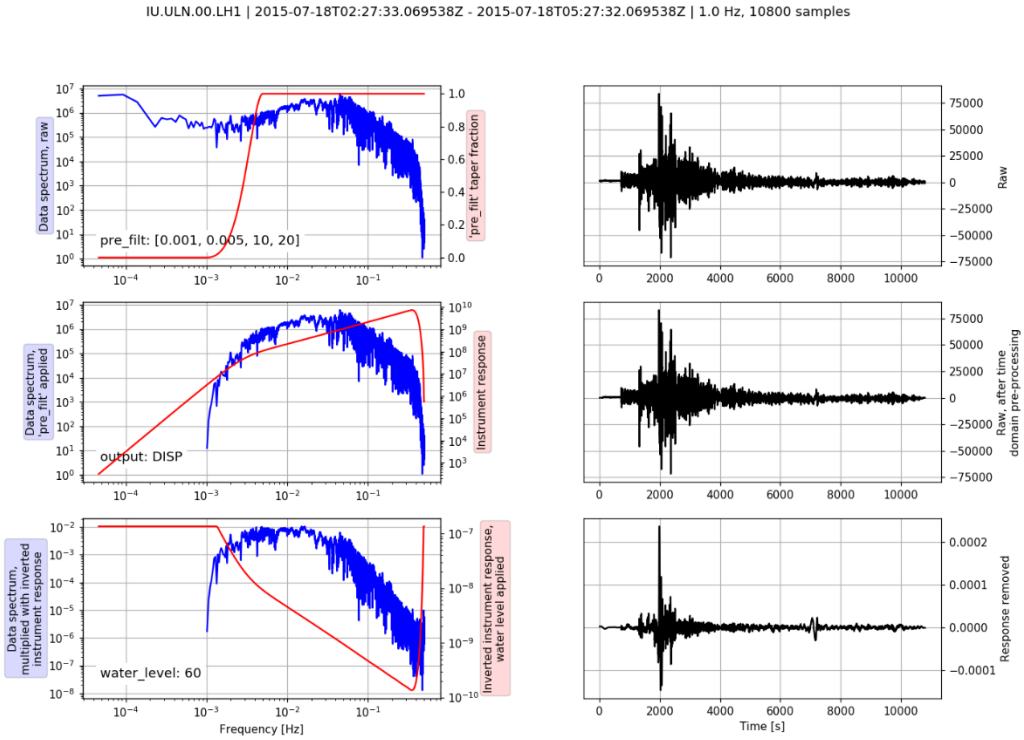
该方法同样适用于 Inventory 对象：

```
from obspy import read, read_inventory

# simply use the included example waveform
st = read()
# the corresponding response is included in ObsPy as a
StationXML file
inv = read_inventory()
# the routine automatically picks the correct response for
each trace
# define a filter band to prevent amplifying noise during the
deconvolution
pre_filt = (0.005, 0.006, 30.0, 35.0)
```

```
st.remove_response(inventory=inv, output='DISP',
pre_filt=pre_filt)
```

使用绘图选项，可以在频域中显示响应消除期间的各个步骤，用以检查 pre_filt 和 water_level 选项，以稳定倒置仪器响应谱的反卷积。

```
from obspy import read, read_inventory

st = read("/path/to/IU_ULN_00_LH1_2015-07-18T02.mseed")
tr = st[0]
inv = read_inventory("/path/to/IU_ULN_00_LH1.xml")
pre_filt = [0.001, 0.005, 10, 20]
tr.remove_response(inventory=inv, pre_filt=pre_filt,
output="DISP",
                   water_level=60, plot=True)
```



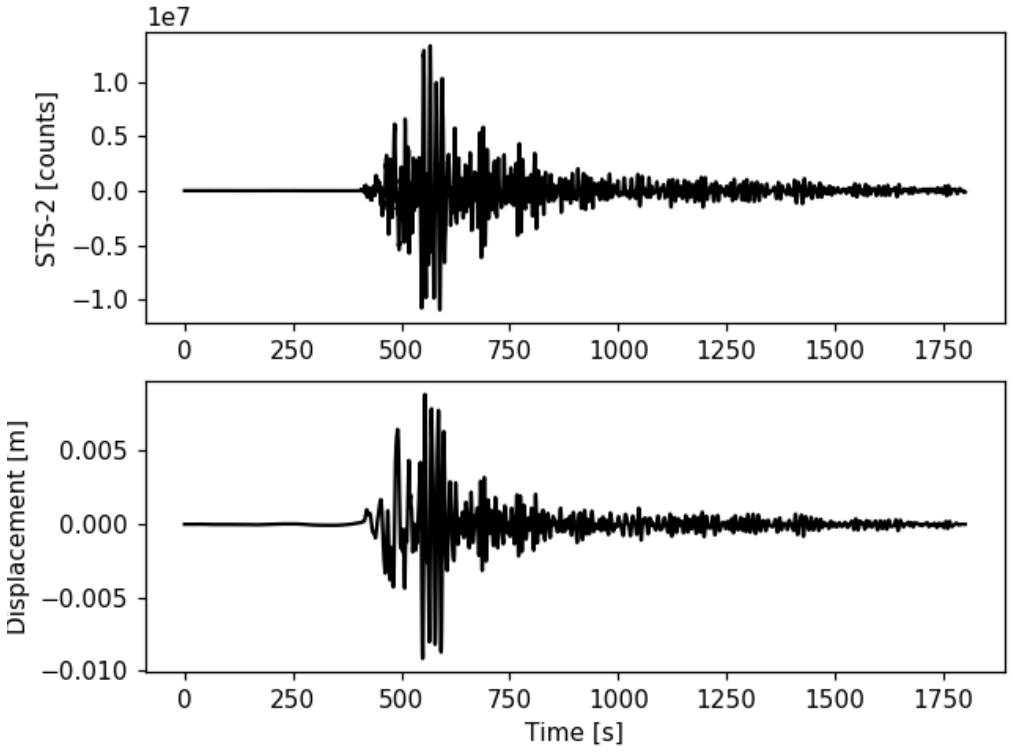
b) 使用 RESP 文件

使用 RESP 文件计算仪器响应信息。

```
import matplotlib.pyplot as plt

import obspy
from obspy.core.util import NamedTemporaryFile
from obspy.clients.fdsn import Client as FDSN_Client
from obspy.clients.iris import Client as OldIris_Client

```

```
# MW 7.1 Darfield earthquake, New Zealand
t1 = obspy.UTCDateTime("2010-09-3T16:30:00.000")
t2 = obspy.UTCDateTime("2010-09-3T17:00:00.000")

# Fetch waveform from IRIS FDSN web service into a ObsPy
stream object
fdsn_client = FDSN_Client("IRIS")
st = fdsn_client.get_waveforms('NZ', 'BFZ', '10', 'HHZ', t1,
t2)

# Download and save instrument response file into a temporary
file
with NamedTemporaryFile() as tf:
    respf = tf.name
    old_iris_client = OldIris_Client()
    # fetch RESP information from "old" IRIS web service, see
obspy.fdsn
    # for accessing the new IRIS FDSN web services
    old_iris_client.resp('NZ', 'BFZ', '10', 'HHZ', t1, t2,
filename=respf)

    # make a copy to keep our original data
    st_orig = st.copy()

    # define a filter band to prevent amplifying noise during
the deconvolution
    pre_filt = (0.005, 0.006, 30.0, 35.0)

    # this can be the date of your raw data or any date for
which the
    # SEED RESP-file is valid
    date = t1

    seedresp = {'filename': respf,  # RESP filename
                # when using Trace/Stream.simulate() the "date"
parameter can
                # also be omitted, and the starttime of the
trace is then used.
                'date': date,
                # Units to return response in ('DIS', 'VEL' or
ACC)
                'units': 'DIS'
                }
```

```
    # Remove instrument response using the information from the
given RESP file
    st.simulate(paz_remove=None, pre_filt=pre_filt,
seedresp=seedresp)

# plot original and simulated data
tr = st[0]
tr_orig = st_orig[0]
time = tr.times()

plt.subplot(211)
plt.plot(time, tr_orig.data, 'k')
plt.ylabel('STS-2 [counts]')
plt.subplot(212)
plt.plot(time, tr.data, 'k')
plt.ylabel('Displacement [m]')
plt.xlabel('Time [s]')
plt.show()
```



c) 使用缺/全数据 SEED 文件（或 XMLSEED 文件）

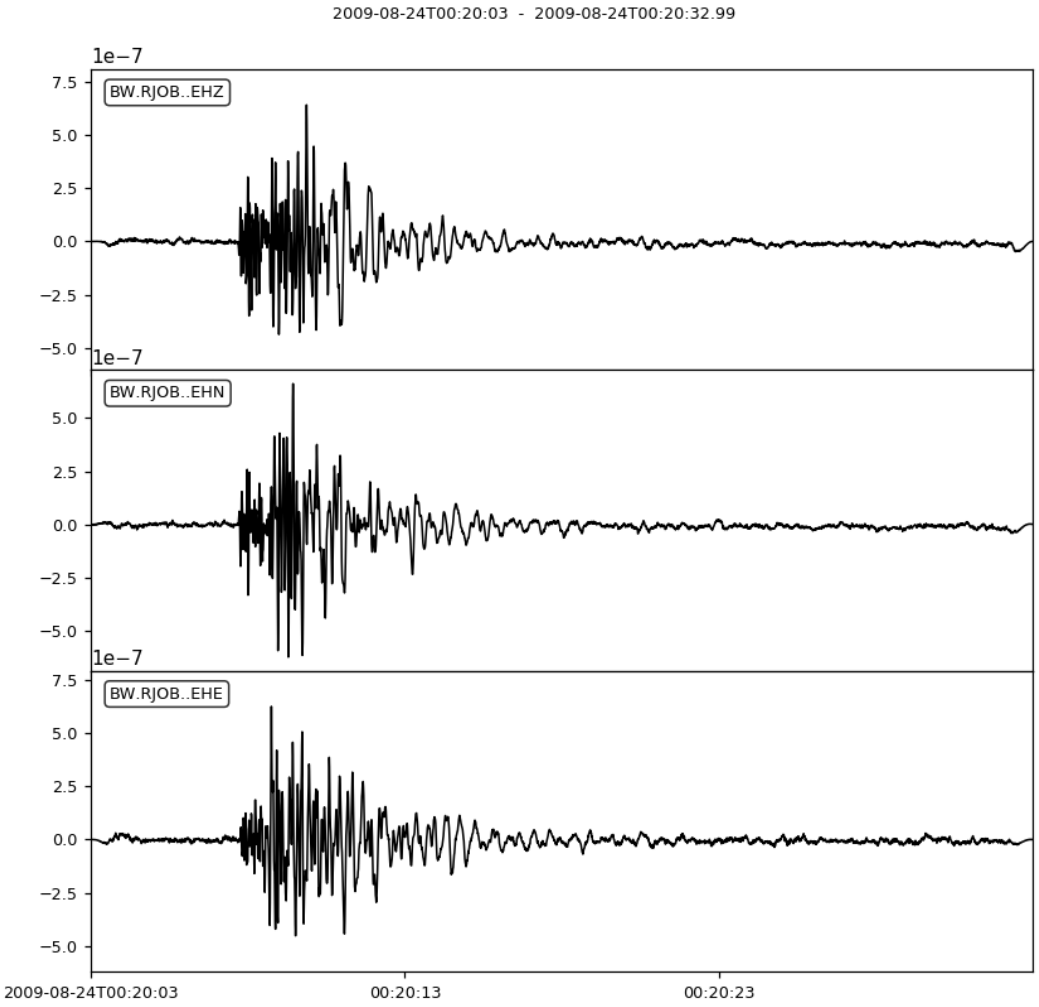
使用无数据 SEED 文件创建的 Parser 对象也可以被使用。然后对于每条轨迹提取 RESP 响应数据。当使用 Stream 或 Trace 中的 simulate（）便捷方法时，“data”参数可以省略。

```
import obspy
from obspy.io.xseed import Parser


st =
obspy.read("https://examples.obspy.org/BW.BGLD..EH.D.2010.037"
)
parser =
Parser("https://examples.obspy.org/dataless.seed.BW_BGLD")
st.simulate(seedresp={'filename': parser, 'units': "DIS"})
```

2) 使用 PAZ 字典

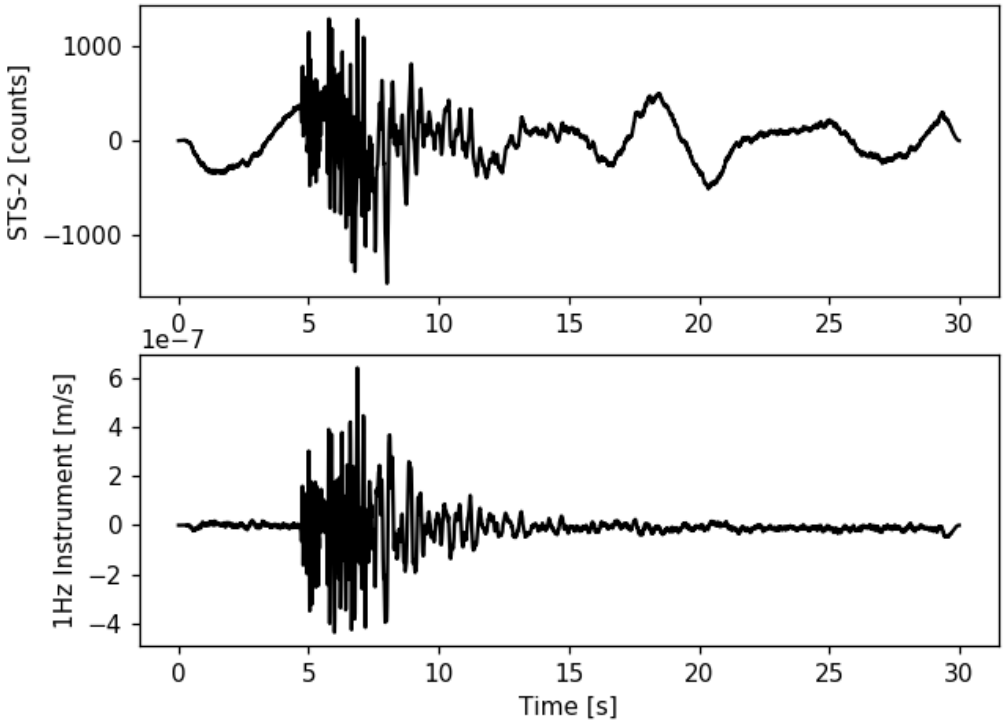
下面的脚本展示了如何使用给定的零极点模拟 1Hz 的 STS-2 地震仪。零极点、增益（归一化因子 A0）、灵敏度（总灵敏度）被指定为字典的关键字。

```
import obspy
from obspy.signal.invsim import corn_freq_2_paz


paz_sts2 = {
    'poles': [-0.037004 + 0.037016j, -0.037004 - 0.037016j, -
251.33 + 0j,
              - 131.04 - 467.29j, -131.04 + 467.29j],
    'zeros': [0j, 0j],
    'gain': 60077000.0,
    'sensitivity': 2516778400.0}
paz_1hz = corn_freq_2_paz(1.0, damp=0.707)  # 1Hz instrument
paz_1hz['sensitivity'] = 1.0

st = obspy.read()
# make a copy to keep our original data
st_orig = st.copy()

# Simulate instrument given poles, zeros and gain of
# the original and desired instrument
st.simulate(paz_remove=paz_sts2, paz_simulate=paz_1hz)

# plot original and simulated data
st_orig.plot()
st.plot()
```



更多自定义的绘图设置可以使用 matplotlib 手动设定：

```
import numpy as np
import matplotlib.pyplot as plt

tr = st[0]
tr_orig = st_orig[0]

t = np.arange(tr.stats.npts) / tr.stats.sampling_rate

plt.subplot(211)
plt.plot(t, tr_orig.data, 'k')
plt.ylabel('STS-2 [counts]')
plt.subplot(212)
plt.plot(t, tr.data, 'k')
plt.ylabel('1Hz Instrument [m/s]')
plt.xlabel('Time [s]')
plt.show()
```



14. Clone an Existing Dataless SEED File

（复制现有的无数据 SEED 文件）

以下代码展示了如何复制一个现有的无数据 SEED 文件（dataless.seed.BW_RNON）然后使用他作为一个模板为新的台站建立 DatalessSEED 文件。

首先，我们要做好必要的输入接口然后读取现有的 DatalessSEED 文件：

```
>>>from obspy import UTCDateTime
>>>from obspy.io.xseed import Parser
>>>p =
Parser("https://examples.obspy.org/dataless.seed.BW_RNON")
>>>blk = p.blockettes
```

现在我们可以调整仅在 DatalessSEED 文件开头中出现的信息：

```
>>>blk[50][0].network_code = 'BW'
>>>blk[50][0].station_call_letters = 'RMOA'
>>>blk[50][0].site_name = "Moar Alm, Bavaria, BW-Net"
>>>blk[50][0].latitude = 47.761658
>>>blk[50][0].longitude = 12.864466
>>>blk[50][0].elevation = 815.0
>>>blk[50][0].start_effective_date = UTCDateTime("2006-07-
18T00:00:00.000000Z")
```

```
>>>blk[50][0].end_effective_date = ""
>>>blk[33][1].abbreviation_description = "Lennartz LE-3D/1
seismometer"
```

之后我们还要调整所有三个通道分量的信息：

```
>>>mult = len(blk[58])/3
>>>for i, cha in enumerate(['Z', 'N', 'E']):
    blk[52][i].channel_identifier = 'EH%s' % cha
    blk[52][i].location_identifier = ''
    blk[52][i].latitude = blk[50][0].latitude
    blk[52][i].longitude = blk[50][0].longitude
    blk[52][i].elevation = blk[50][0].elevation
    blk[52][i].start_date = blk[50][0].start_effective_date
    blk[52][i].end_date = blk[50][0].end_effective_date
    blk[53][i].number_of_complex_poles = 3
    blk[53][i].real_pole = [-4.444, -4.444, -1.083]
    blk[53][i].imaginary_pole = [+4.444, -4.444, +0.0]
    blk[53][i].real_pole_error = [0, 0, 0]
    blk[53][i].imaginary_pole_error = [0, 0, 0]
    blk[53][i].number_of_complex_zeros = 3
    blk[53][i].real_zero = [0.0, 0.0, 0.0]
    blk[53][i].imaginary_zero = [0.0, 0.0, 0.0]
    blk[53][i].real_zero_error = [0, 0, 0]
    blk[53][i].imaginary_zero_error = [0, 0, 0]
    blk[53][i].A0_normalization_factor = 1.0
    blk[53][i].normalization_frequency = 3.0
    # stage sequence number 1, seismometer gain
    blk[58][i*mult].sensitivity_gain = 400.0
    # stage sequence number 2, digitizer gain
    blk[58][i*mult+1].sensitivity_gain = 1677850.0
    # stage sequence number 0, overall sensitivity
    blk[58][(i+1)*mult-1].sensitivity_gain = 671140000.0
```

注意：这个例子中没有设置 FIR 系数。

最后，我们可以将调整后的 DatalessSEED 卷写入一个新文件：

```
>>>p.write_seed("dataless.seed.BW_RMOA")
```

15.Export Seismograms to MATLAB（导出数据到 MATLAB）

下面的例子展示了如何使用 python 读取波形文件并保存每条 Trace 的结果 Stream 对象到一个 MATLAB 的 MAT 格式文件中。这些数据可以通过 MATLAB 使用 load 函数读取：

```
from scipy.io import savemat
import obspy

```

```
st =
obspy.read("https://examples.obspy.org/BW.BGLD..EH.D.2010.037"
)
for i, tr in enumerate(st):
    mdict = {k: str(v) for k, v in tr.stats.iteritems()}
    mdict['data'] = tr.data
    savemat("data-%d.mat" % i, mdict)
```

16. Export Seismograms to ASCII（导出数据为 ASCII 文件）

1) Built-in 内置格式

你可以使用 Obspy 的 write（）函数直接输出 Stream 对象的波形数据为任何 ASCII 格式。

```
>>>from obspy.core import read
>>>stream =
read('https://examples.obspy.org/RJOB20090824.ehz')
>>>stream.write('outfile.ascii', format='SLIST')
```

下面是现在支持的 ASCII 格式：

- SLIST——使用下列简单列表代表的 ASCII 时间序列格式：

```
TIMESERIES BW_RJOB__EHZ_D, 6001 samples, 200 sps, 2009-08-
24T00:20:03.000000, SLIST, INTEGER,
288 300 292 285 265 287
279 250 278 278 268 258
```

- TSPAIR——时间—样本对 ASCII 格式

```
TIMESERIES BW_RJOB__EHZ_D, 6001 samples, 200 sps, 2009-08-
24T00:20:03.000000, TSPAIR, INTEGER,
2009-08-24T00:20:03.000000  288
2009-08-24T00:20:03.005000  300
2009-08-24T00:20:03.010000  292
2009-08-24T00:20:03.015000  285
2009-08-24T00:20:03.020000  265
2009-08-24T00:20:03.025000  287
```

- SH_ASC——Seismic Handler 支持的 ASCII 格式

```
DELTA: 5.000000e-03
LENGTH: 6001
START: 24-AUG-2009_00:20:03.000
COMP: Z
CHAN1: E
CHAN2: H
STATION: RJOB
CALIB: 1.000000e+00
```

```
2.880000e+02 3.000000e+02 2.920000e+02 2.850000e+02
2.650000e+02 2.870000e+02 2.790000e+02 2.500000e+02
```

2) 自定义格式

下面的脚本示例如何使用自定义头转换每条 Trace 或者地震记录文件到 ASCII 文件。波形数据会乘上给定的 calibration 校准因子，然后使用 Numpy 的 savetxt（）函数保存。

```
"""
USAGE: export_seismograms_to_ascii.py in_file out_file
calibration
"""
from __future__ import print_function

import sys

import numpy as np
import obspy


try:
    in_file = sys.argv[1]
    out_file = sys.argv[2]
    calibration = float(sys.argv[3])
except:
    print(__doc__)
    raise

st = obspy.read(in_file)
for i, tr in enumerate(st):
    f = open("%s_%d" % (out_file, i), "w")
    f.write("# STATION %s\n" % (tr.stats.station))
    f.write("# CHANNEL %s\n" % (tr.stats.channel))
    f.write("# START_TIME %s\n" % (str(tr.stats.starttime)))
    f.write("# SAMP_FREQ %f\n" % (tr.stats.sampling_rate))
    f.write("# NDAT %d\n" % (tr.stats.npts))
    np.savetxt(f, tr.data * calibration, fmt="%f")
    f.close()
```

17.Anything to MiniSEED（转换任意文件格式为 MiniSEED）

下列程序展示如何转换任何数据到 MiniSEED 格式。例子中将几行气象站的输出写入 MiniSEED 文件。准确的元信息（starttime、sampling_rate、station 名称等）也被编码。

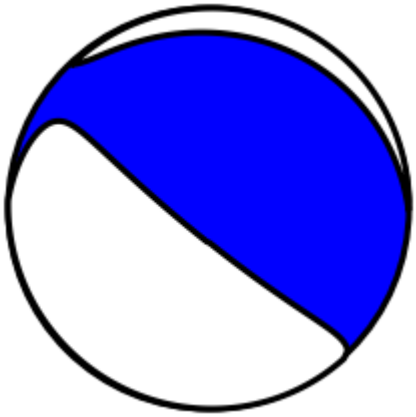
（注意：必须是 MiniSEED 标准允许的编码）。如果你想发送日志信息或者通过 SeedLink 协议输出气象站或其他任何东西，那么转换任意 ASCII 为 MiniSEED 极有帮助。

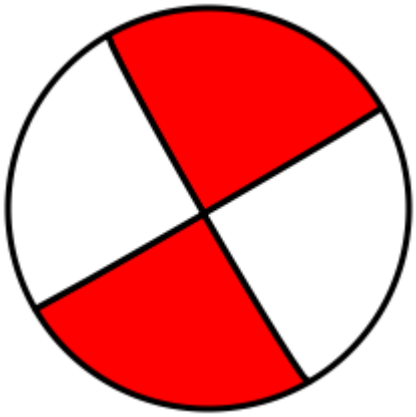
```
from __future__ import print_function

import numpy as np
from obspy import UTCDateTime, read, Trace, Stream


weather = """
00.0000 0.0 ??? 4.7 97.7 1015.0 0.0 010308 000000
00.0002 0.0 ??? 4.7 97.7 1015.0 0.0 010308 000001
00.0005 0.0 ??? 4.7 97.7 1015.0 0.0 010308 000002
00.0008 0.0 ??? 4.7 97.7 1015.4 0.0 010308 000003
00.0011 0.0 ??? 4.7 97.7 1015.0 0.0 010308 000004
00.0013 0.0 ??? 4.7 97.7 1015.0 0.0 010308 000005
00.0016 0.0 ??? 4.7 97.7 1015.0 0.0 010308 000006
00.0019 0.0 ??? 4.7 97.7 1015.0 0.0 010308 000007
"""

# Convert to NumPy character array
data = np.fromstring(weather, dtype='|S1')

# Fill header attributes
stats = {'network': 'BW', 'station': 'RJOB', 'location': '',
         'channel': 'WLZ', 'npts': len(data), 'sampling_rate':
0.1,
         'mseed': {'dataquality': 'D'}}
# set current time
stats['starttime'] = UTCDateTime()
st = Stream([Trace(data=data, header=stats)])
# write as ASCII file (encoding=0)
st.write("weather.mseed", format='MSEED', encoding=0,
reclen=256)

# Show that it worked, convert NumPy character array back to
string
st1 = read("weather.mseed")
print(st1[0].data.tostring())
```

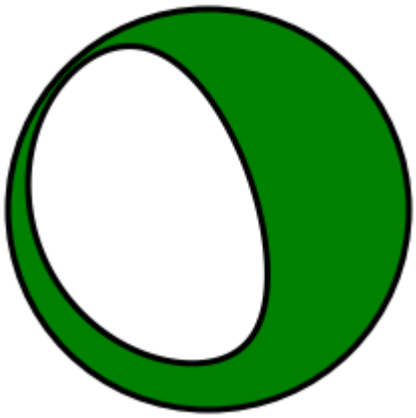
18.Beachball Plot（绘制沙滩球图）

下面展示如何创建震源机制的图形表示：

```
from obspy.imaging.beachball import beachball


mt = [0.91, -0.89, -0.02, 1.78, -1.55, 0.47]
beachball(mt, size=200, linewidth=2, facecolor='b')

mt2 = [150, 87, 1]
beachball(mt2, size=200, linewidth=2, facecolor='r')

mt3 = [-2.39, 1.04, 1.35, 0.57, -2.94, -0.94]
beachball(mt3, size=200, linewidth=2, facecolor='g')
```







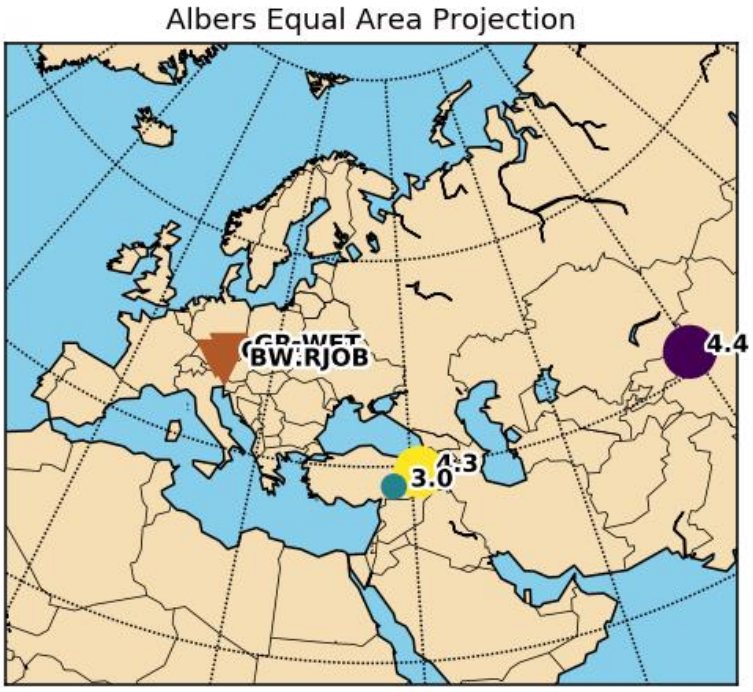
19.Basemap Plots

1) 设置自定义投影的 Basemap plot

简单的 Basemap 绘制可以通过 Inventory 或 Catalog 对象的内置方法执行：Inventory.plot（），Catalog.plot()。

```
from mpl_toolkits.basemap import Basemap
import numpy as np
import matplotlib.pyplot as plt

from obspy import read_inventory, read_events

# Set up a custom basemap, example is taken from basemap
users' manual
fig, ax = plt.subplots()

# setup albers equal area conic basemap
# lat_1 is first standard parallel.
# lat_2 is second standard parallel.
# lon_0, lat_0 is central point.
m = Basemap(width=8000000, height=7000000,
            resolution='c', projection='aea',
            lat_1=40., lat_2=60, lon_0=35, lat_0=50, ax=ax)
m.drawcoastlines()
m.drawcountries()
m.fillcontinents(color='wheat', lake_color='skyblue')
# draw parallels and meridians.
m.drawparallels(np.arange(-80., 81., 20.))
m.drawmeridians(np.arange(-180., 181., 20.))
m.drawmapboundary(fill_color='skyblue')
ax.set_title("Albers Equal Area Projection")

# we need to attach the basemap object to the figure, so that
obspy knows about
# it and reuses it
fig.bmap = m

# now let's plot some data on the custom basemap:
inv = read_inventory()
inv.plot(fig=fig, show=False)
cat = read_events()
cat.plot(fig=fig, show=False, title="", colorbar=False)
```

```
plt.show()
```



2) 确定地点的带有 Beachball 的 Basemap plot

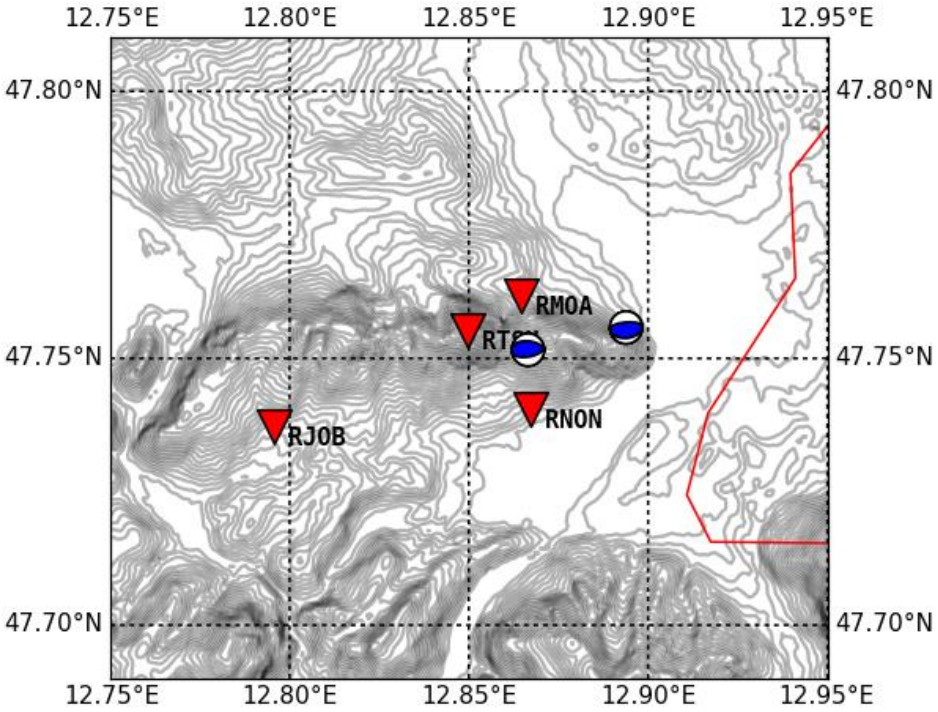
下面的例子展示如何在 basemap 图中绘制一些台站的 beachball 图。该例需要安装 basemap 包。演示的 SRTM 文件可以从 here 下载。我们的 SRTM 前几行是这样的：

```
ncols         400
nrows         200
xllcorner     12°40'E
yllcorner     47°40'N
xurcorner     13°00'E
yurcorner     47°50'N
cellsize      0.00083333333333333
NODATA_value  -9999
682 681 685 690 691 689 678 670 675 680 681 679 675 671 674
680 679 679 675 671 668 664 659 660 656 655 662 666 660 659
659 658 ....
```

```
import gzip
```

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap

from obspy.imaging.beachball import beach


# read in topo data (on a regular lat/lon grid)
# (SRTM data from: http://srtm.csi.cgiar.org/)
with gzip.open("srtm_1240-1300E_4740-4750N.asc.gz") as fp:
    srtm = np.loadtxt(fp, skiprows=8)

# origin of data grid as stated in SRTM data file header
# create arrays with all lon/lat values from min to max and
lats = np.linspace(47.8333, 47.6666, srtm.shape[0])
lons = np.linspace(12.6666, 13.0000, srtm.shape[1])

# create Basemap instance with Mercator projection
# we want a slightly smaller region than covered by our SRTM
data
m = Basemap(projection='merc', lon_0=13, lat_0=48,
resolution="h",
            llcrnrlon=12.75, llcrnrlat=47.69, urcrnrlon=12.95,
urcrnrlat=47.81)

# create grids and compute map projection coordinates for
lon/lat grid
x, y = m(*np.meshgrid(lons, lats))

# Make contour plot
cs = m.contour(x, y, srtm, 40, colors="k", lw=0.5, alpha=0.3)
m.drawcountries(color="red", linewidth=1)

# Draw a lon/lat grid (20 lines for an interval of one degree)
m.drawparallels(np.linspace(47, 48, 21), labels=[1, 1, 0, 0],
fmt="%.2f",
                dashes=[2, 2])
m.drawmeridians(np.linspace(12, 13, 21), labels=[0, 0, 1, 1],
fmt="%.2f",
                dashes=[2, 2])

# Plot station positions and names into the map
# again we have to compute the projection of our lon/lat
values
```

```
lats = [47.761659, 47.7405, 47.755100, 47.737167]
lons = [12.864466, 12.8671, 12.849660, 12.795714]
names = [" RMOA", " RNON", " RTSH", " RJOB"]
x, y = m(lons, lats)
m.scatter(x, y, 200, color="r", marker="v", edgecolor="k",
zorder=3)
for i in range(len(names)):
    plt.text(x[i], y[i], names[i], va="top",
family="monospace", weight="bold")

# Add beachballs for two events
lats = [47.751602, 47.75577]
lons = [12.866492, 12.893850]
x, y = m(lons, lats)
# Two focal mechanisms for beachball routine, specified as
[strike, dip, rake]
focmecs = [[80, 50, 80], [85, 30, 90]]
ax = plt.gca()
for i in range(len(focmecs)):
    b = beach(focmecs[i], xy=(x[i], y[i]), width=1000,
linewidth=1)
    b.set_zorder(10)
    ax.add_collection(b)

plt.show()
```



几个 Note：

- GDAL 包允许直接读取 GeoTiff 到 numpy ndarray。

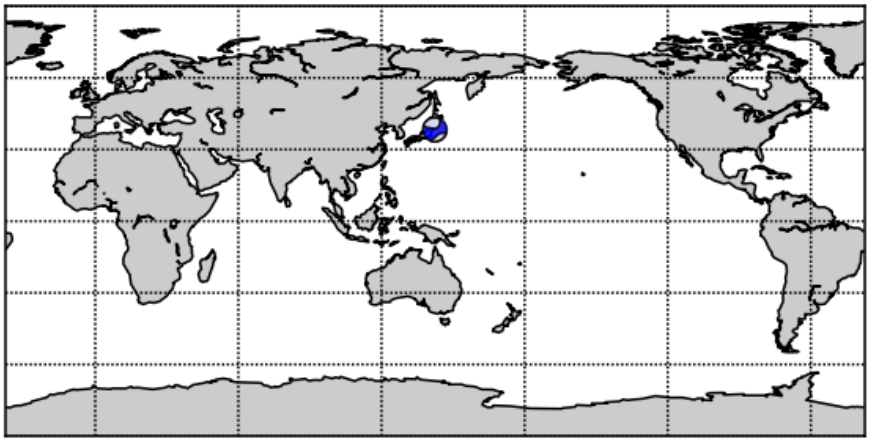
```
>>> geo = gdal.Open("file.geotiff")
>>> x = geo.ReadAsArray()
```

- 可从 ASTER 得到 Geo Tiff 高度数据。
- 可添加阴影/照明。查看 plotmap_shaded.py 例程获取更多信息。

3) 带有 beachball 的全球 Basemap

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap

from obspy.imaging.beachball import beach


m = Basemap(projection='cyl', lon_0=142.36929, lat_0=38.3215,
            resolution='c')

m.drawcoastlines()
m.fillcontinents()
m.drawparallels(np.arange(-90., 120., 30.))
m.drawmeridians(np.arange(0., 420., 60.))
```

```
m.drawmapboundary()

x, y = m(142.36929, 38.3215)
focmecs = [0.136, -0.591, 0.455, -0.396, 0.046, -0.615]

ax = plt.gca()
b = beach(focmecs, xy=(x, y), width=10, linewidth=1,
alpha=0.85)
b.set_zorder(10)
ax.add_collection(b)
plt.show()
```



20.Interfacing R from Python（从 python 对接到 R）

rpy2 包允许 python 对接到 R。下例展示如何转换 numpy.ndarray 数据为 R 矩阵，并对其执行 R 的 summary 命令。

```
>>>from obspy.core import read
>>>import rpy2.robjects as RO
>>>import rpy2.robjects.numpy2ri
>>>r = RO.r
>>>st = read("test/BW.BGLD..EHE.D.2008.001")
>>>M = RO.RMatrix(st[0].data)
```

```
>>>print(r.summary(M))
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-1056.0  -409.0  -393.0  -393.7  -378.0   233.0
```

21.Coordinate Conversions（坐标转换）

使用 pyproj 可以方便的进行坐标转换。在查找完源和目标坐标系的 EPSG 代码后，只需几行代码即可完成坐标转换。下例将两个德国台站的坐标信息转换为区域使用的 Gauß-Krüger 坐标。

```
>>> import pyproj
>>> lat = [49.6919, 48.1629]
>>> lon = [11.2217, 11.2752]
>>> proj_wgs84 = pyproj.Proj(init="epsg:4326")
>>> proj_gk4 = pyproj.Proj(init="epsg:31468")
>>> x, y = pyproj.transform(proj_wgs84, proj_gk4, lon, lat)
>>> print(x)
[4443947.179347951, 4446185.667319892]
>>> print(y)
[5506428.401023342, 5336354.054996853]
```

另一种常见用法是将纬度和经度的位置信息转换到 Universal Transverse Mercator coordinate system (UTM)坐标系。这对于小区域中的大密集阵列特别有用。 使用 utm 包可以轻松完成这种转换。

```
>>> import utm
>>> utm.from_latlon(51.2, 7.5)
(395201.3103811303, 5673135.241182375, 32, 'U')
>>> utm.to_latlon(340000, 5710000, 32, 'U')
(51.51852098408468, 6.693872395145327)
```

22.Hierarchical Clustering（分级聚类）

使用 SciPy 包中提供的方法可以执行分级聚类。它允许从相似性矩阵构建聚类并制作树状图。以下示例显示了如何对已计算的相似性矩阵执行此操作。相似性数据是根据具有诱发地震活动的区域中的事件计算的（使用 obspy.signal 中的相关例程），并且可以从我们的示例网络服务器获取：

首先，导入必要的模块并通过我们的网页载入数据：

```
>>> import io, urllib
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> from scipy.cluster import hierarchy
```

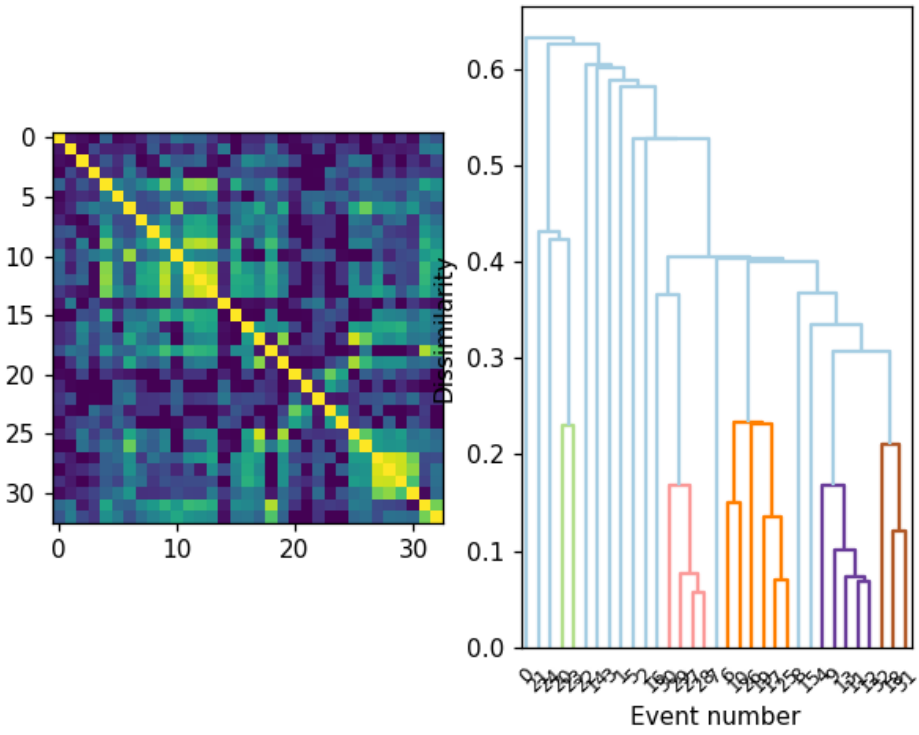
```
>>> from scipy.spatial import distance

>>> url = "https://examples.obspy.org/dissimilarities.npz"
>>> with io.BytesIO(urllib.urlopen(url).read()) as fh,
np.load(fh) as data:
... dissimilarity = data['dissimilarity']
```

现在我们绘制相异矩阵：

```
>>> plt.subplot(121)
>>> plt.imshow(1 - dissimilarity, interpolation="nearest")
```

然后我们使用 SciPy 构建并绘制树形图到图像右侧的子图中：

```
>>> dissimilarity = distance.squareform(dissimilarity)

>>> threshold = 0.3

>>> linkage = hierarchy.linkage(dissimilarity,
method="single")

>>> clusters = hierarchy.fcluster(linkage, threshold,
criterion="distance")

>>> plt.subplot(122)

>>> hierarchy.dendrogram(linkage, color_threshold=0.3)

>>> plt.xlabel("Event number")

>>> plt.ylabel("Dissimilarity")

>>> plt.show()
```



23. Visualizing Probabilistic Power Spectral Densities

（可视化概率功率谱密度）

下列代码展示使用 obspy.signal 中定义的 PPSD 类，该例程对于解释现场质量控制检查的噪声测量非常有用。更多信息参考[McNamara2004]。

```
>>> from obspy import read
>>> from obspy.io.xseed import Parser
>>> from obspy.signal import PPSD
```

读取数据并选择一条所需的台站/信道轨迹：

```
>>> st =
read("https://examples.obspy.org/BW.KW1..EHZ.D.2011.037")
>>> tr = st.select(id="BW.KW1..EHZ")[0]
```

元数据可以是：StationXML 文件或 FDSN 网络服务提供的 Inventory；无数据 SEED 文件的 Parser；本地 RESP 文件的文件名；传统的零极点字典。

初始化一个新的 PPSD 语句。ppsd 对象将确保随后只有适当的数据进入概率 psd 统计。

```
>>> parser =
Parser("https://examples.obspy.org/dataless.seed.BW_KW1")
>>> ppsd = PPSD(tr.stats, metadata=parser)
```

然后添加数据（Trace 或 Stream 对象）做 PPSD 估计。

```
>>> ppsd.add(st)
True
```

我们可以检查 ppsd 估计中表示的时间范围。它包含计算 psds 的一小时长切片的开始时间的排序列表（此处仅打印显示前两个）。

```
>>> print(ppsd.times[:2])
[UTCDateTime(2011, 2, 6, 0, 0, 0, 935000), UTCDateTime(2011,
2, 6, 0, 30, 0, 935000)]
>>> print("number of psd segments:", len(ppsd.times))
number of psd segments: 47
```

再次添加同样的 stream 无效（返回值 False），ppsd 对象确保没有重复的数据段进入 ppsd 估计。

```
>>> ppsd.add(st)
False
>>> print("number of psd segments:", len(ppsd.times))
number of psd segments: 47
```

可逐步添加来自其他文件/来源的更多信息。

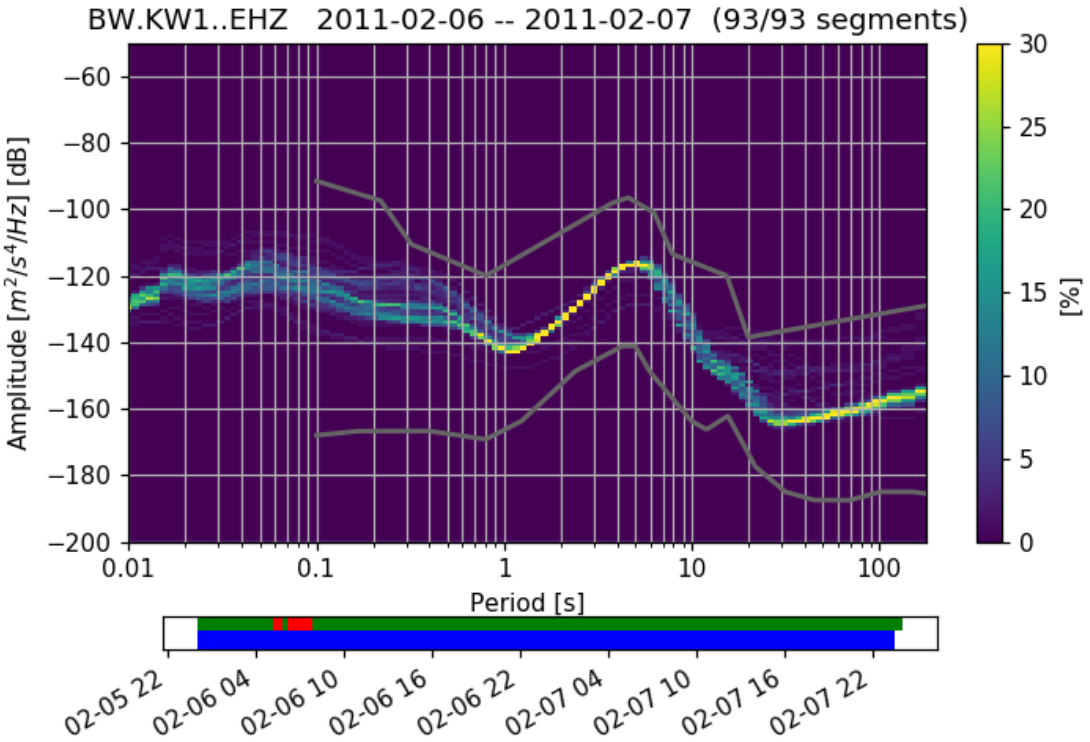
```
>>> st =
read("https://examples.obspy.org/BW.KW1..EHZ.D.2011.038")
>>> ppsd.add(st)
True
```

ppsd 的图形表示可以显示在 matplotlib 窗口中。

```
>>> ppsd.plot()
```

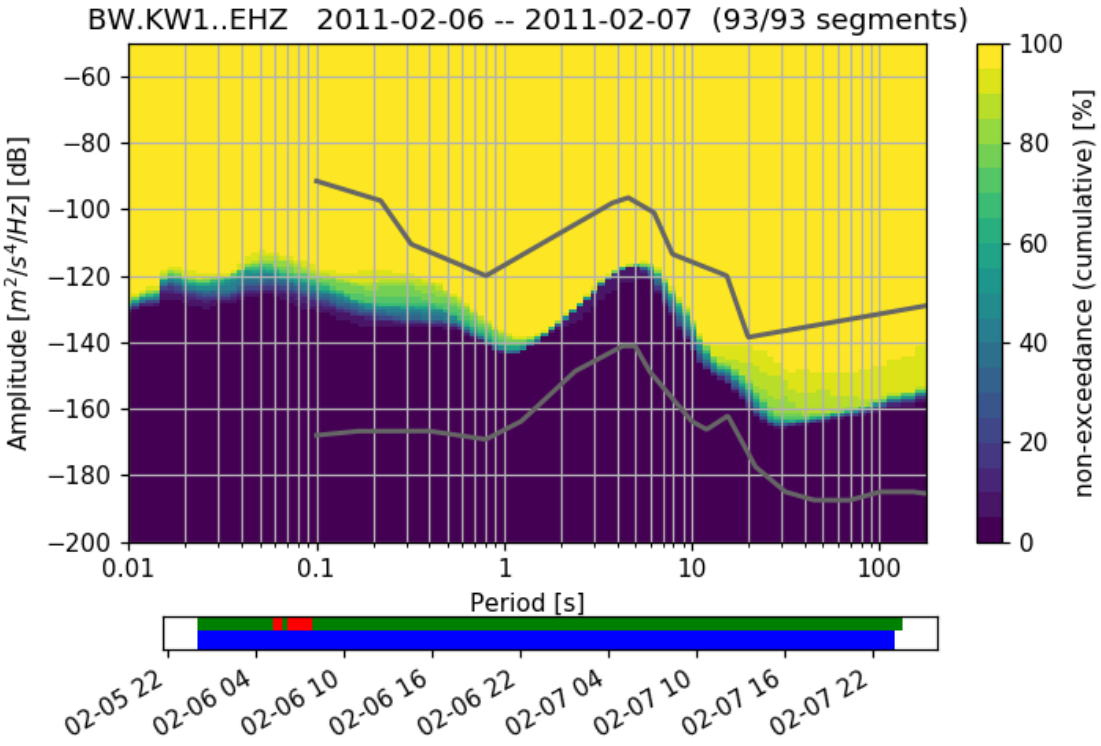
或者保存为图片文件。

```
>>> ppsd.plot("/tmp/ppsd.png")
>>> ppsd.plot("/tmp/ppsd.pdf")
```



对于每个频率仓，还可以显示累积的直方图：

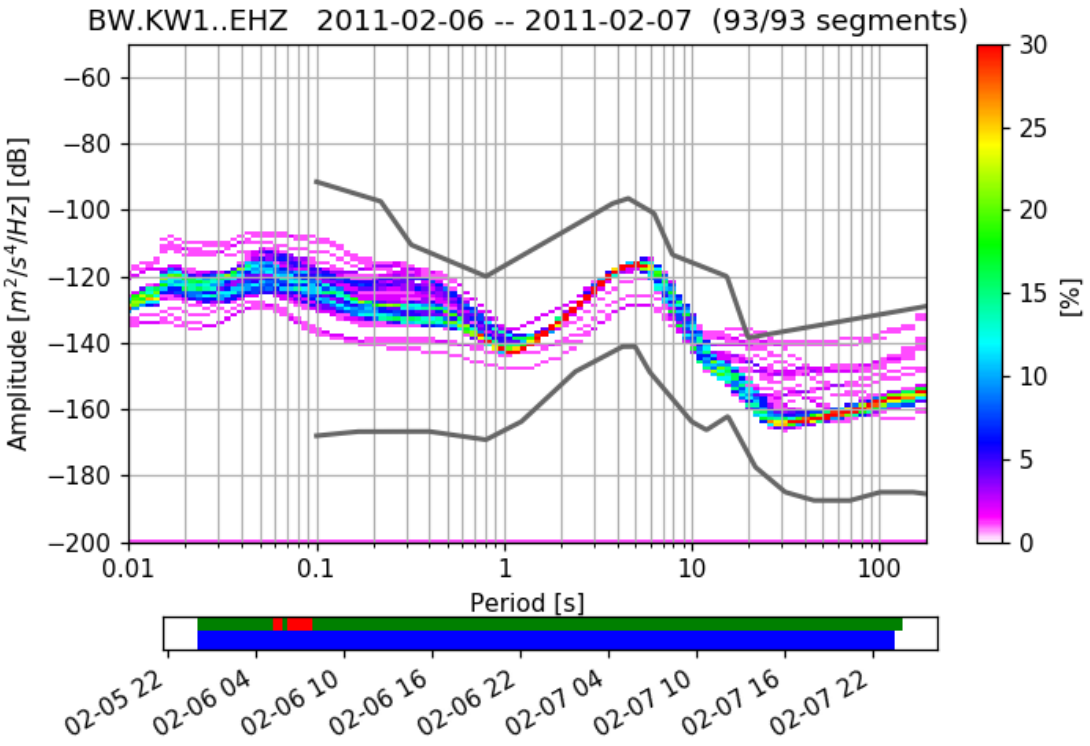
```
>>> ppsd.plot(cumulative=True)
```



要使用 PQLX /[McNamara2004]使用的色彩映射，可以从 obspy.imaging.cm 导入并使用该

色彩映射：

```
>>> from obspy.imaging.cm import pqlx
>>> ppsd.plot(cmap=pqlx)
```

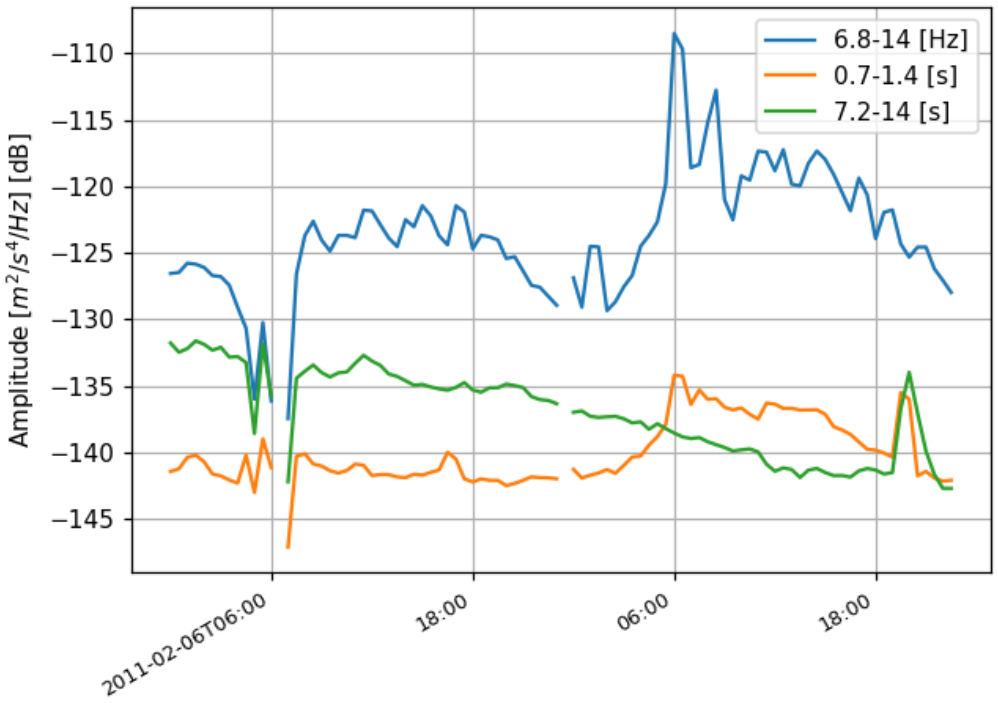


实际 PPSD（[McNamara2004]）下方的是可视化 PPSD 的数据基础。第一行显示输入 PPSD 的数据，绿条表示可用数据，红条表示添加到 PPSD 的流中的间隙。蓝色的底行显示进入直方图的单个 psd 测量值。默认处理方法用零填充间隙，然后这些数据段显示为单个偏离 psd 行。

Note：从无数据 SEED 或 StationXML 文件提供元数据比指定静态零极点信息更安全（参见 PPSD）。

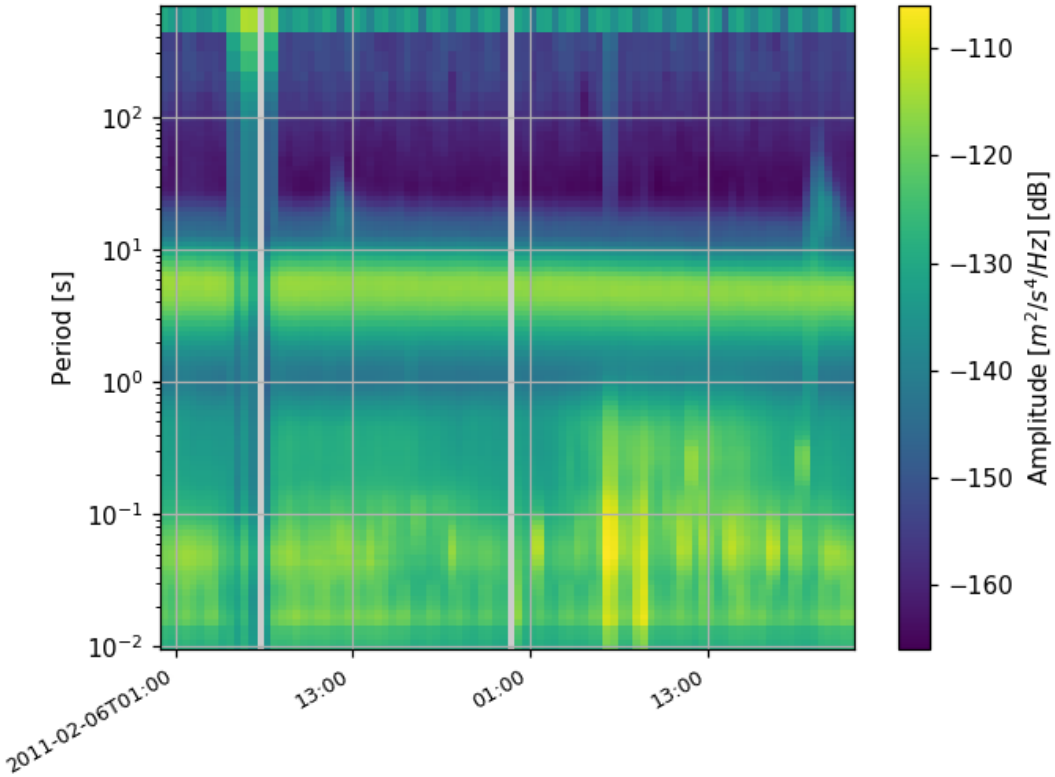
psd 值的时间序列也可以通过访问 psd_values 从 PPSD 中提取并使用 plot_temporal()绘制。

```
>>> ppsd.plot_temporal([0.1, 1, 10])
```



使用 plot_spectrogram()绘制频谱图：

```
>>> ppsd.plot_spectrogram()
```

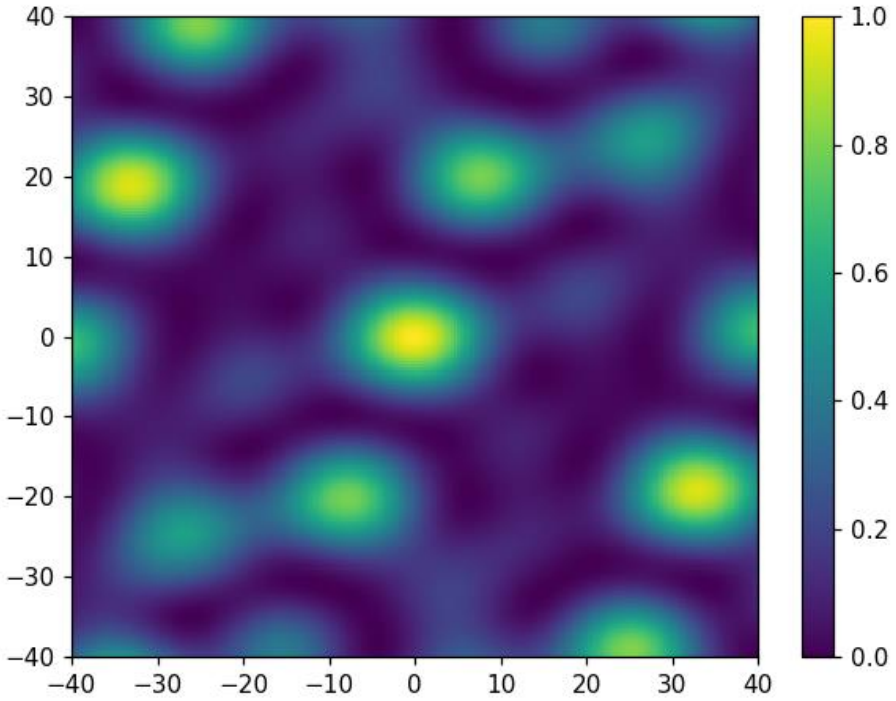


24.Array Response Function（数组响应函数）

下面的代码块展示了如何使用 ObsPy 的 obspy.signal.array_analysis.array_transff_wavenumber()函数绘制波束形成的数组传递函数（波数的函数）。

```
import numpy as np
import matplotlib.pyplot as plt

from obspy.imaging.cm import obspy_sequential
from obspy.signal.array_analysis import
array_transff_wavenumber


# generate array coordinates
coords = np.array([[10., 60., 0.], [200., 50., 0.], [-120.,
170., 0.],
                   [-100., -150., 0.], [30., -220., 0.]])

# coordinates in km
coords /= 1000.

# set limits for wavenumber differences to analyze
klim = 40.
kxmin = -klim
kxmax = klim
kymin = -klim
kymax = klim
kstep = klim / 100.

# compute transfer function as a function of wavenumber
difference
transff = array_transff_wavenumber(coords, klim, kstep,
coordsys='xy')

# plot
plt.pcolor(np.arange(kxmin, kxmax + kstep * 1.1, kstep) -
kstep / 2.,
           np.arange(kymin, kymax + kstep * 1.1, kstep) - kstep
/ 2.,
           transff.T, cmap=obspy_sequential)

plt.colorbar()
plt.clim(vmin=0., vmax=1.)
plt.xlim(kxmin, kxmax)
```

```
plt.ylim(kymin, kymax)
plt.show()
```



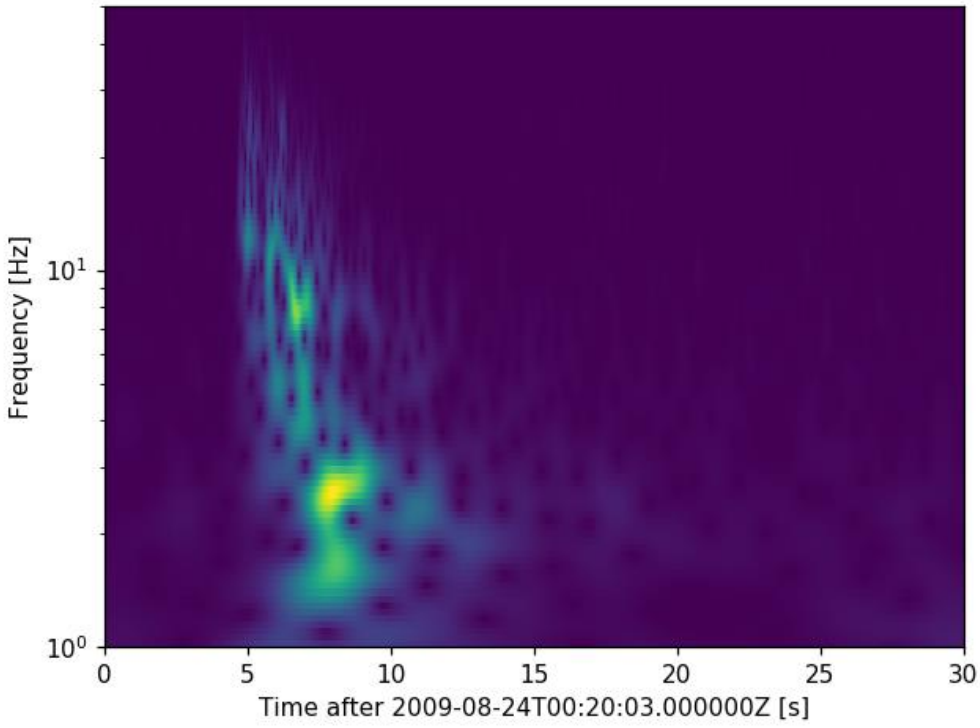
25.Continuous Wavelet Transform（连续小波变换）

1) 使用 obspy

下面的小例子是使用 obspy 内置例程（基于[Kristekova2006].）的连续小波变换。

```
import numpy as np
import matplotlib.pyplot as plt

import obspy
from obspy.imaging.cm import obspy_sequential
from obspy.signal.tf_misfit import cwt


st = obspy.read()
tr = st[0]
npts = tr.stats.npts
dt = tr.stats.delta
t = np.linspace(0, dt * npts, npts)
f_min = 1
```

```
f_max = 50

scalogram = cwt(tr.data, dt, 8, f_min, f_max)

fig = plt.figure()
ax = fig.add_subplot(111)

x, y = np.meshgrid(
    t,
    np.logspace(np.log10(f_min), np.log10(f_max),
scalogram.shape[0]))

ax.pcolormesh(x, y, np.abs(scalogram), cmap=obspy_sequential)
ax.set_xlabel("Time after %s [s]" % tr.stats.starttime)
ax.set_ylabel("Frequency [Hz]")
ax.set_yscale('log')
ax.set_ylim(f_min, f_max)
plt.show()
```

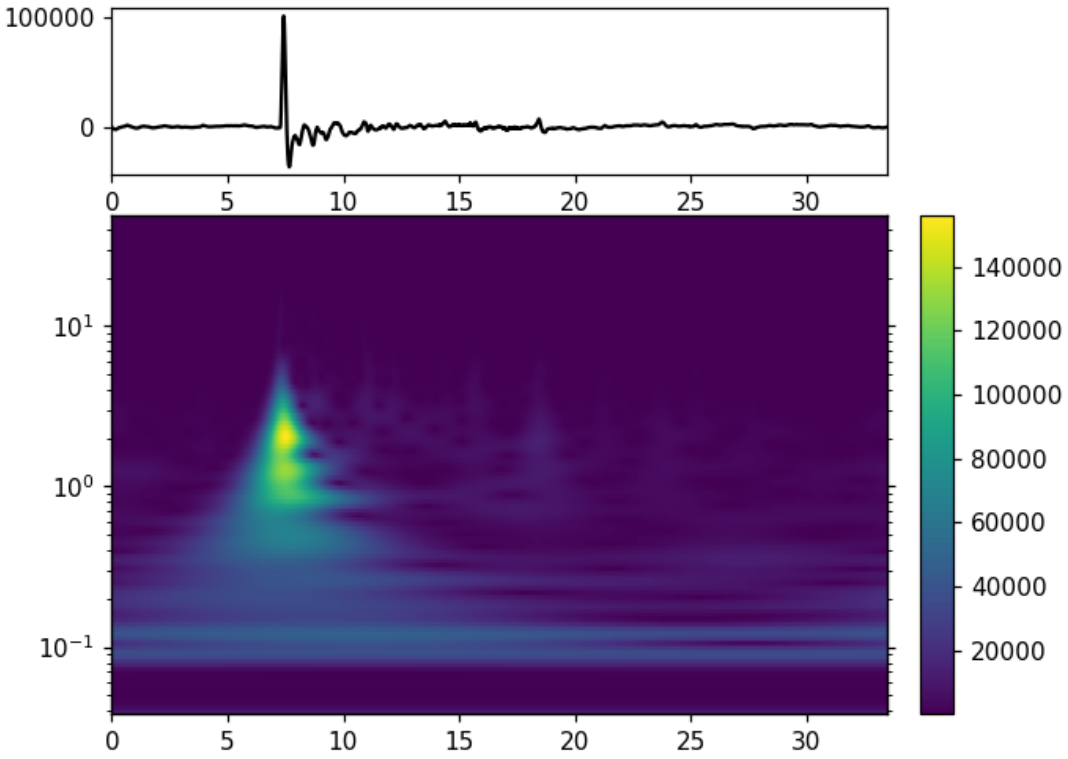


2) 使用 MLPY

下面的小脚本使用 mlpy 包对 2008 年 Yasur 记录到的次声数据进行连续小波变换。关于小波的更多信息参考 Wikipedia（其中的 omega0 因子这里记作 sigma）。

```
import numpy as np
import matplotlib.pyplot as plt

import mlpy

import obspy
from obspy.imaging.cm import obspy_sequential


tr =
obspy.read("https://examples.obspy.org/a02i.2008.240.mseed")[0
]

omega0 = 8
wavelet_fct = "morlet"
scales = mlpy.wavelet.autoscales(N=len(tr.data),
dt=tr.stats.delta, dj=0.05,
                                 wf=wavelet_fct, p=omega0)
spec = mlpy.wavelet.cwt(tr.data, dt=tr.stats.delta,
scales=scales,
                        wf=wavelet_fct, p=omega0)
# approximate scales through frequencies
freq = (omega0 + np.sqrt(2.0 + omega0 ** 2)) / (4 * np.pi *
scales[1:])

fig = plt.figure()
ax1 = fig.add_axes([0.1, 0.75, 0.7, 0.2])
ax2 = fig.add_axes([0.1, 0.1, 0.7, 0.60], sharex=ax1)
ax3 = fig.add_axes([0.83, 0.1, 0.03, 0.6])

t = np.arange(tr.stats.npts) / tr.stats.sampling_rate
ax1.plot(t, tr.data, 'k')

img = ax2.imshow(np.abs(spec), extent=[t[0], t[-1], freq[-1],
freq[0]],
                 aspect='auto', interpolation='nearest',
cmap=obspy_sequential)
# Hackish way to overlay a logarithmic scale over a linearly
scaled image.
twin_ax = ax2.twinx()
twin_ax.set_yscale('log')
twin_ax.set_xlim(t[0], t[-1])
twin_ax.set_ylim(freq[-1], freq[0])
ax2.tick_params(which='both', labelleft=False, left=False)
```

```
twin_ax.tick_params(which='both', labelleft=True, left=True,
labelright=False)

fig.colorbar(img, cax=ax3)

plt.show()
```



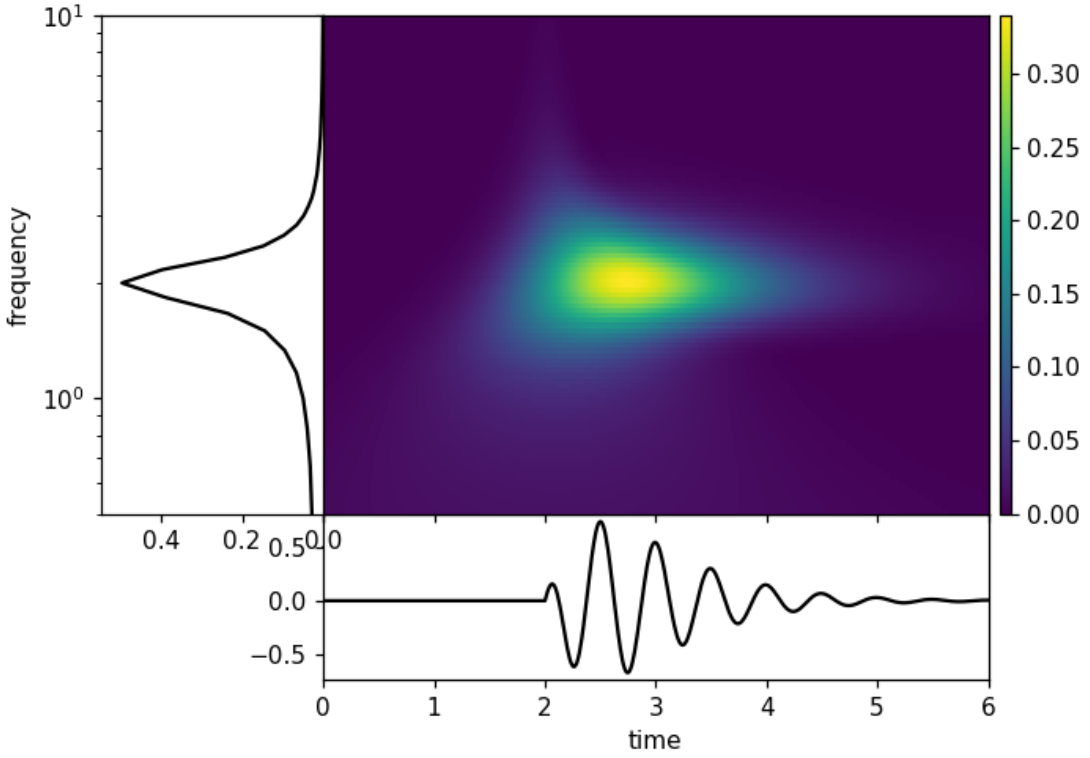
26.Time_Frequency_Misfit（时频失配）

tf_misfit 模块提供了几种基于[Kristekova2006] 和 [Kristekova2009]的时频错位函数。这里几个例子展示了如何使用其包含的绘制工具。

1) 绘制时频表示图

```
import numpy as np

from obspy.signal.tf_misfit import plot_tfr


# general constants
tmax = 6.
dt = 0.01
```

```
npts = int(tmax / dt + 1)
t = np.linspace(0., tmax, npts)

fmin = .5
fmax = 10

# constants for the signal
A1 = 4.
t1 = 2.
f1 = 2.
phi1 = 0.

# generate the signal
H1 = (np.sign(t - t1) + 1) / 2
st1 = A1 * (t - t1) * np.exp(-2 * (t - t1))
st1 *= np.cos(2. * np.pi * f1 * (t - t1) + phi1 * np.pi) * H1

plot_tfr(st1, dt=dt, fmin=fmin, fmax=fmax)
```

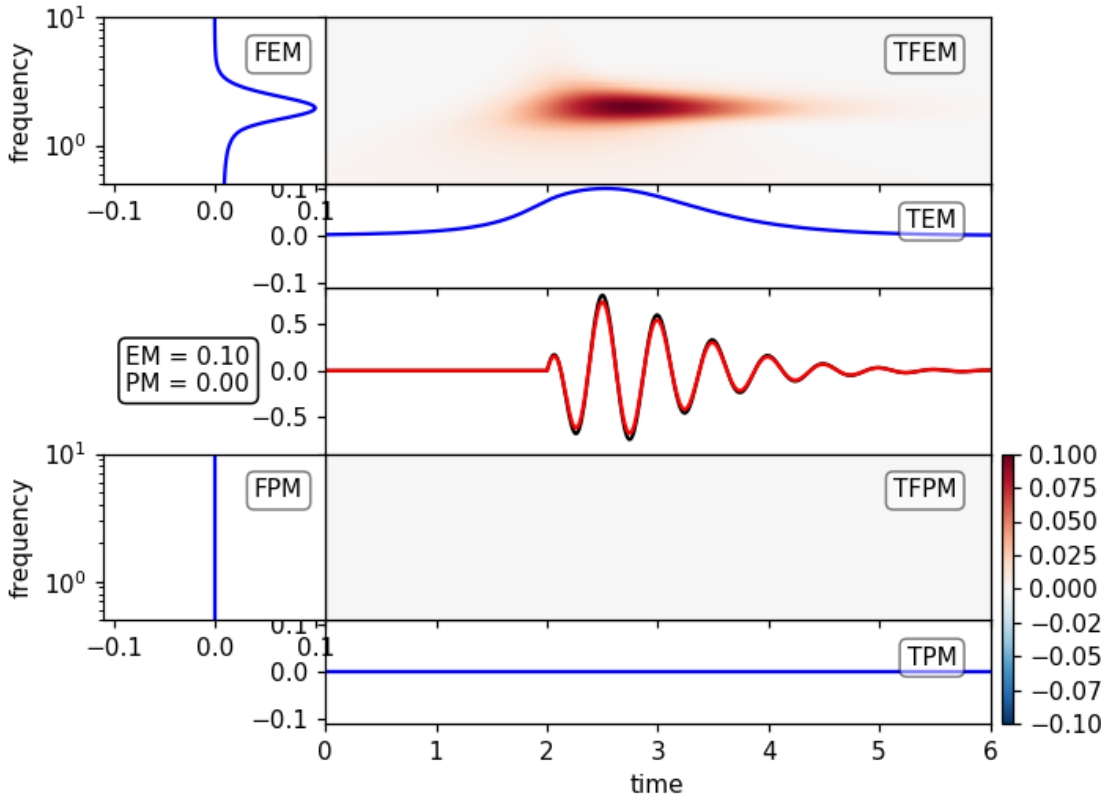


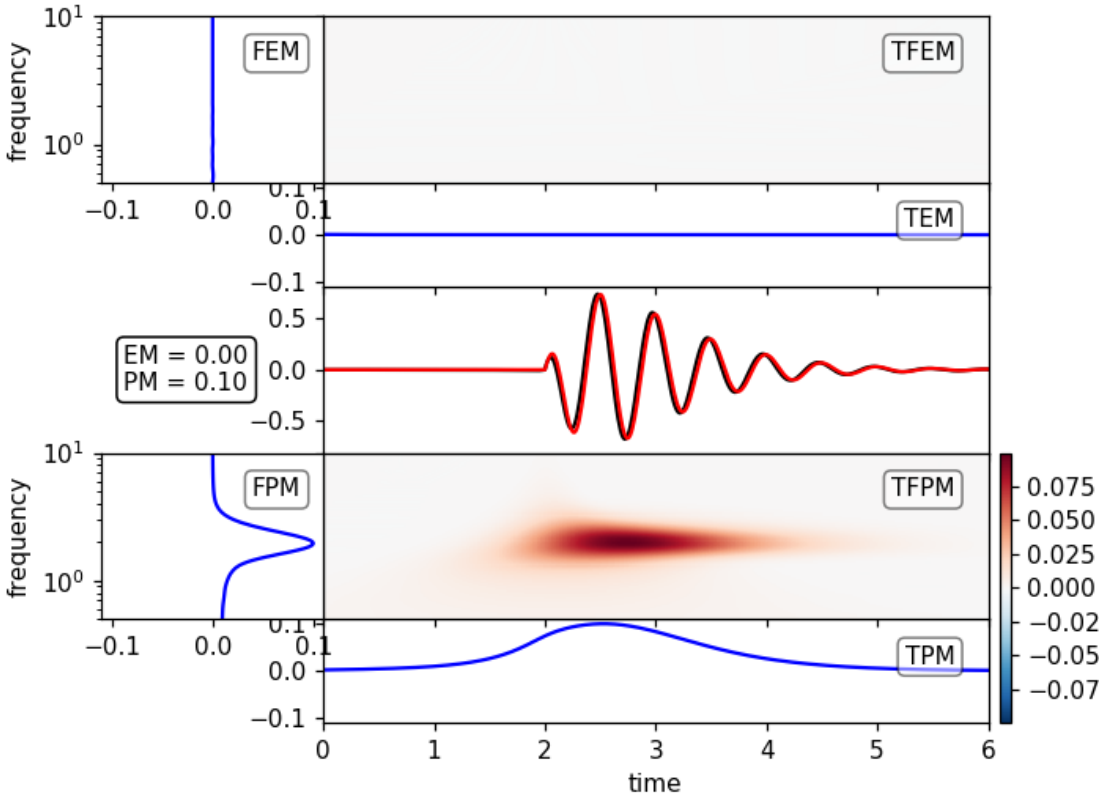
2) 绘制时频失配

时频失配适用于信号的小差异，接着上面的例子：

```
from scipy.signal import hilbert
from obspy.signal.tf_misfit import plot_tf_misfits
```

```
# amplitude and phase error
phase_shift = 0.1
amp_fac = 1.1

# reference signal
st2 = st1.copy()

# generate analytical signal (hilbert transform) and add phase
shift
st1p = hilbert(st1)
st1p = np.real(np.abs(st1p) * \
       np.exp((np.angle(st1p) + phase_shift * np.pi) * 1j))

# signal with amplitude error
st1a = st1 * amp_fac

plot_tf_misfits(st1a, st2, dt=dt, fmin=fmin, fmax=fmax,
show=False)
plot_tf_misfits(st1p, st2, dt=dt, fmin=fmin, fmax=fmax,
show=False)

plt.show()
```



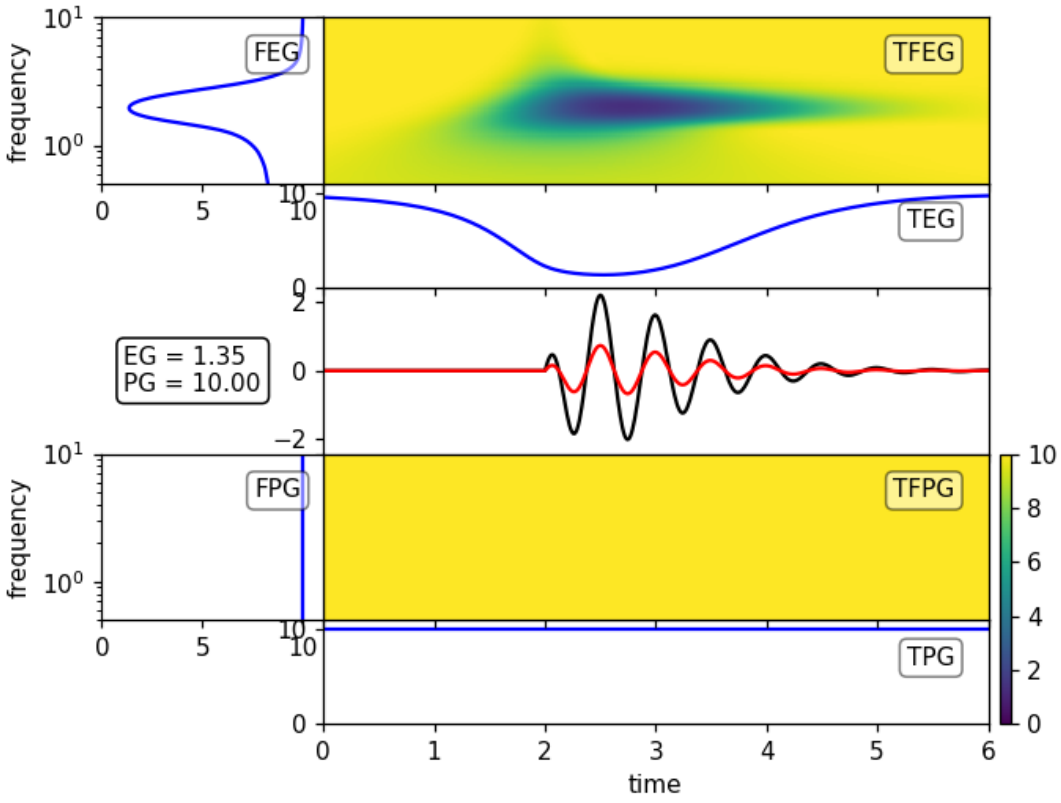


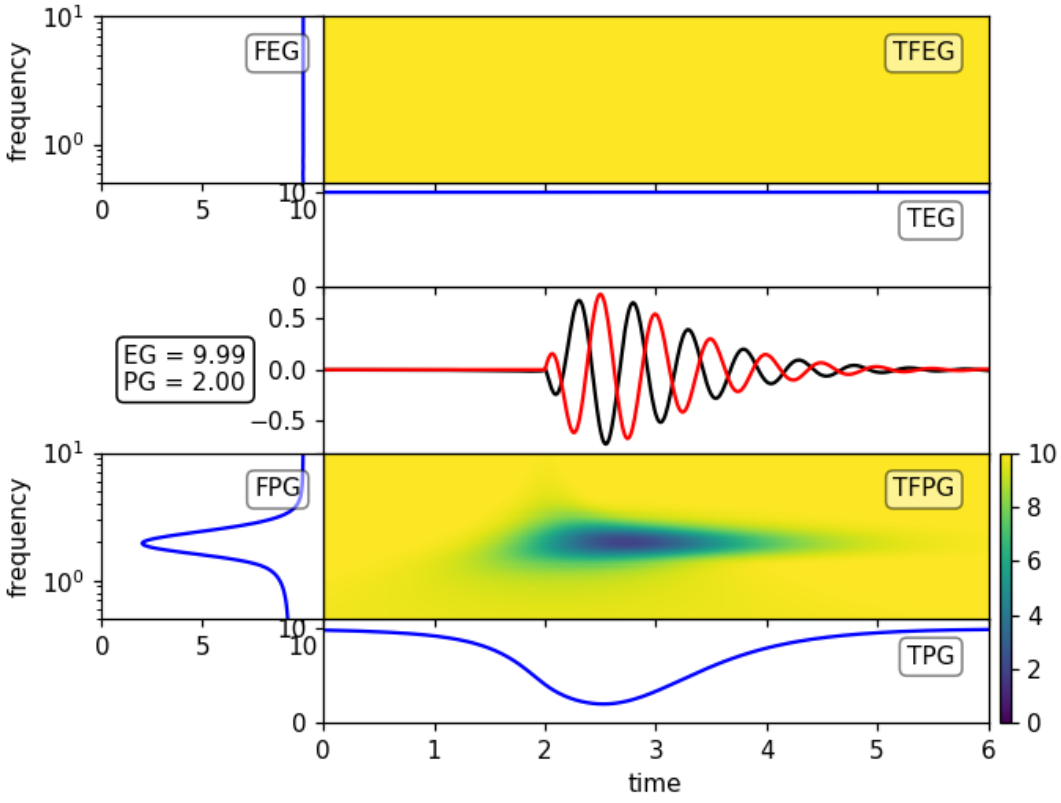
3) 绘制时频适配

时频失配适合于信号间的较大差异，接上面的例子：

```
from obspy.signal.tf_misfit import plot_tf_gofs

# amplitude and phase error
phase_shift = 0.8
amp_fac = 3.

# generate analytical signal (hilbert transform) and add phase
shift
st1p = hilbert(st1)
st1p = np.real(np.abs(st1p) * \
       np.exp((np.angle(st1p) + phase_shift * np.pi) * 1j))

# signal with amplitude error
st1a = st1 * amp_fac

plot_tf_gofs(st1a, st2, dt=dt, fmin=fmin, fmax=fmax,
show=False)
plot_tf_gofs(st1p, st2, dt=dt, fmin=fmin, fmax=fmax,
show=False)
```

```
plt.show()
```

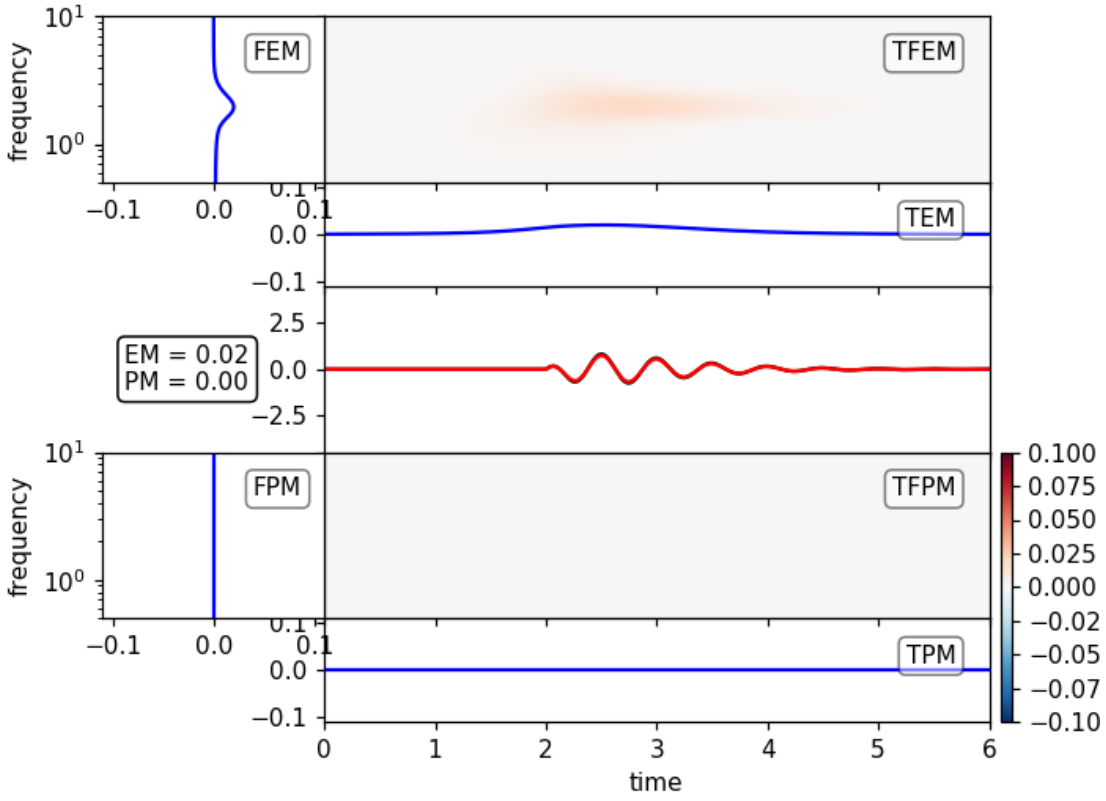

FEG
TFEG
TEG
EG = 1.35
PG = 10.00
FPG
TFPG
TPG
frequency
time

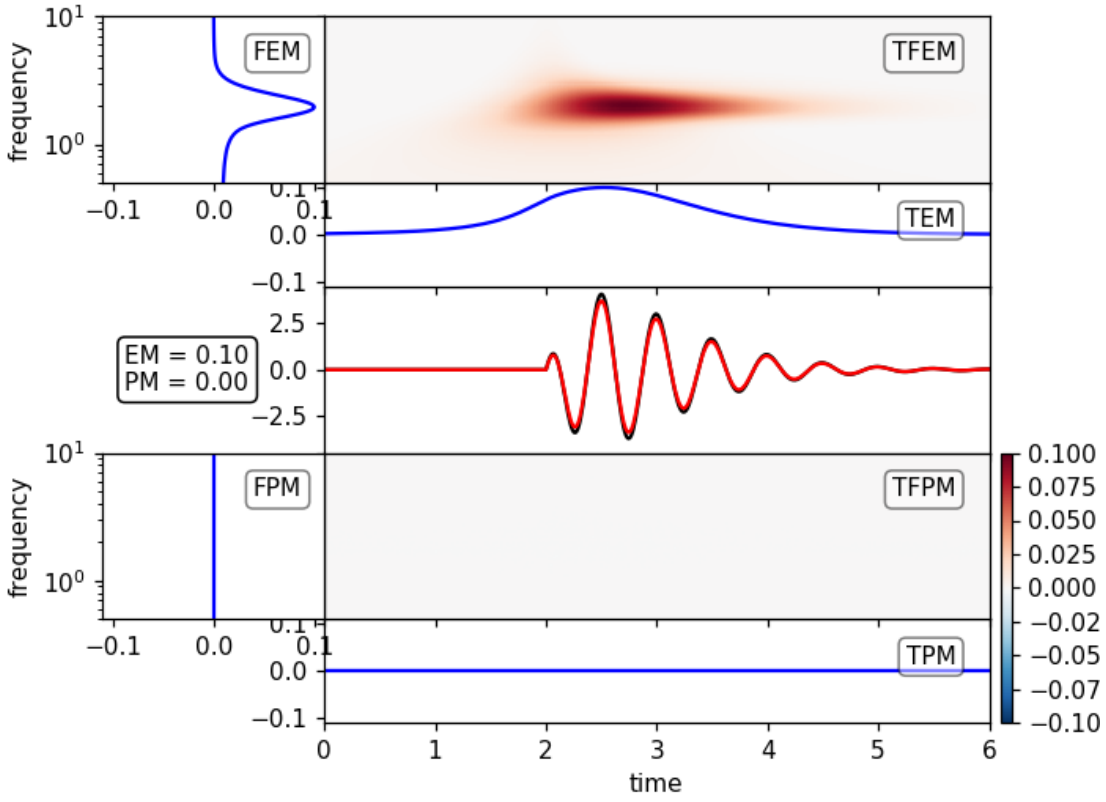

FEG
TFEG
TEG
EG = 9.99
PG = 2.00
FPG
TFPG
TPG
frequency
time

4) 多组件数据

对于多组件数据和不匹配的全局归一化，坐标轴应相应地缩放。接上面的例子：

```
# amplitude error
amp_fac = 1.1

# reference signals
st2_1 = st1.copy()
st2_2 = st1.copy() * 5.
st2 = np.c_[st2_1, st2_2].T

# signals with amplitude error
st1a = st2 * amp_fac

plot_tf_misfits(st1a, st2, dt=dt, fmin=fmin, fmax=fmax)
```



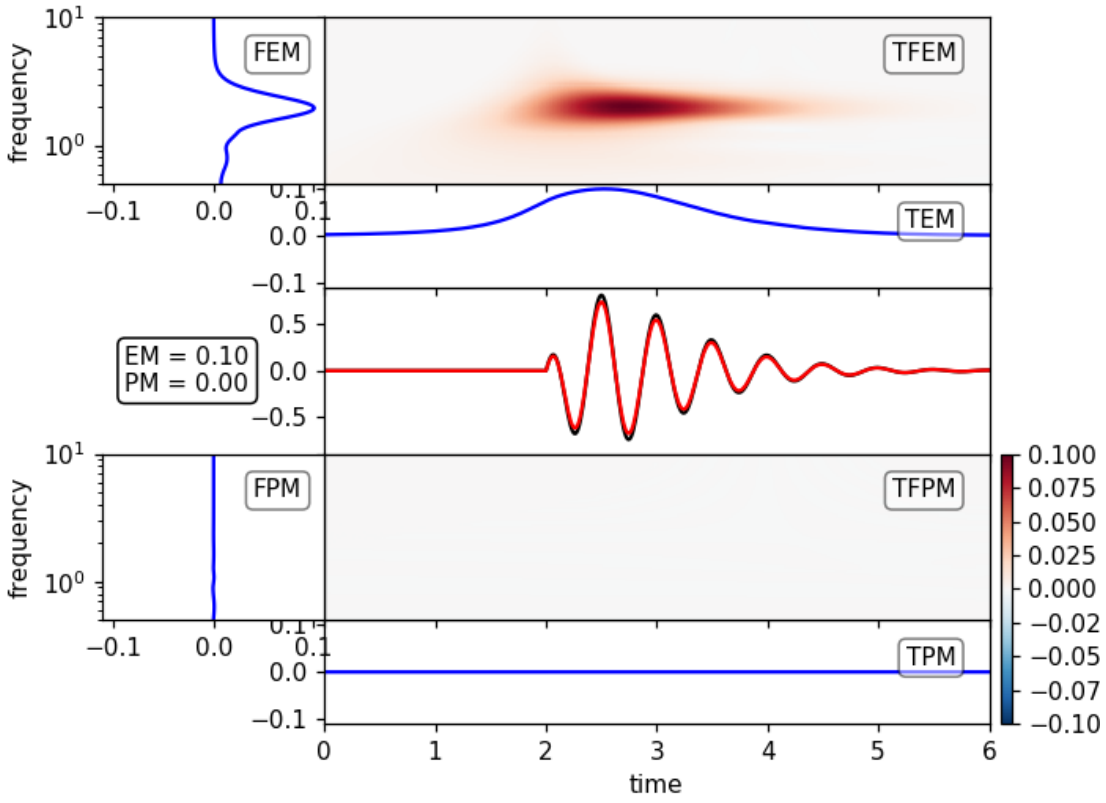


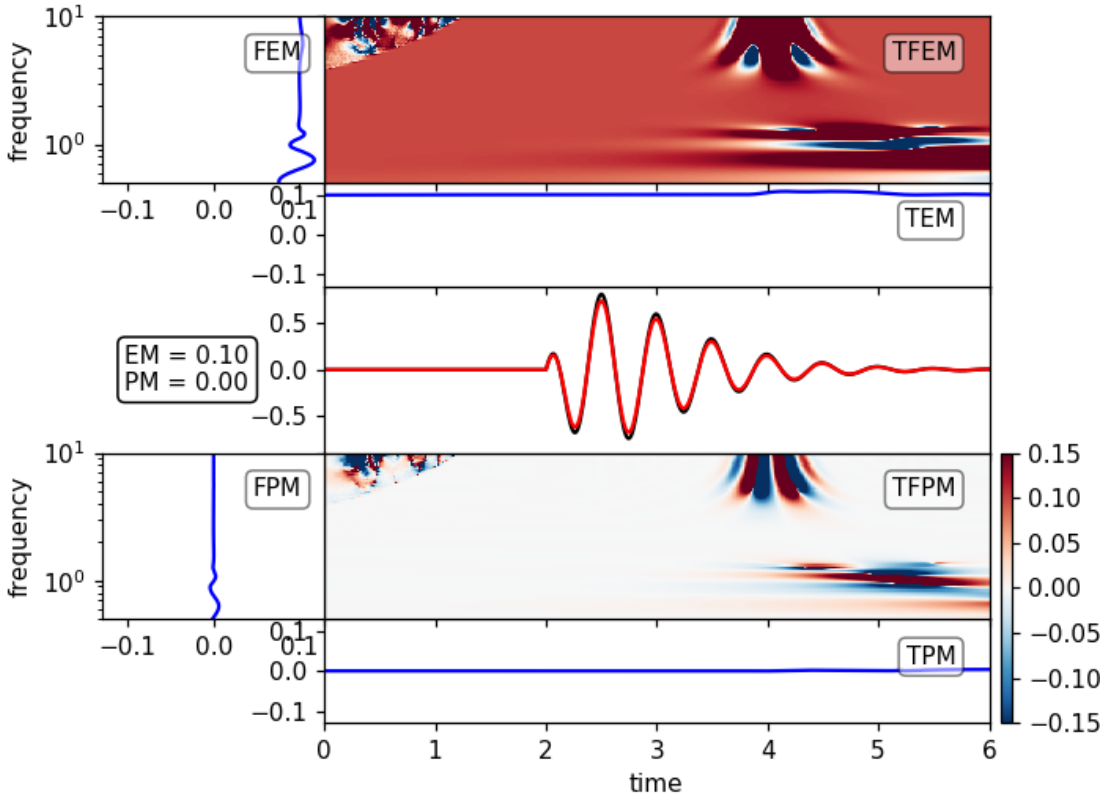
5) 局部归一化

局部归一化可以解决最大振幅波超出频率和时间的范围的问题，但是可能会在没有能量的区域中产生人为的影响。在该分析示例中，针对的是信号发生前的高频。为此需要手动设置限制：

```
# amplitude and phase error
amp_fac = 1.1

ste = 0.001 * A1 * np.exp(- (10 * (t - 2. * t1)) ** 2) \

# reference signal
st2 = st1.copy()

# signal with amplitude error + small additional pulse aftert
4 seconds
st1a = st1 * amp_fac + ste

plot_tf_misfits(st1a, st2, dt=dt, fmin=fmin, fmax=fmax,
show=False)
plot_tf_misfits(st1a, st2, dt=dt, fmin=fmin, fmax=fmax,
norm='local',
              clim=0.15, show=False)
```

```
plt.show()
```


FEM
TFEM
TEM
EM = 0.10
PM = 0.00
FPM
TFPM
TPM
frequency
time


FEM
TFEM
TEM
EM = 0.10
PM = 0.00
FPM
TFPM
TPM
frequency
time

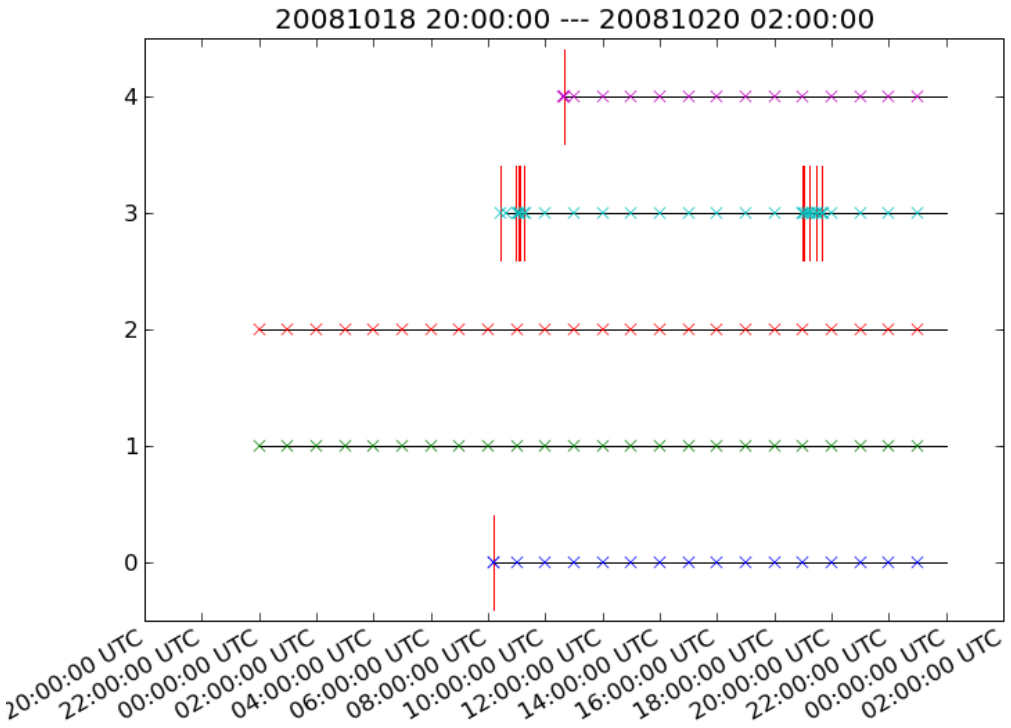
27.Visualize Data Availability of Local Waveform Archive

（可视化本地波形存档数据的可用性）

通常，您拥有大量数据并希望知道哪个站点在何时是可用的。对于这种假设的情况，obspy 提供了 obspy-scan 脚本（安装后即可用），它能从文件的数据头检测文件格式（MiniSEED, SAC, SACXY, GSE2, SH-ASC, SH-Q, SEISAN, 等），在间隙处绘制为垂直红线，可用数据的在开始时间绘制十字，数据本身绘为水平线。该脚本可以扫描超过 1000 个文件（已有被用于扫描 30000 个文件，耗时约 45 分钟。），自动绘制年/月范围。它会打开一个可放大的交互式绘图窗口。

从命令提示符执行类似下面的语句，使用通配符匹配文件：

```
$ obspy-scan /bay_mobil/mobil/20090622/1081019/*_1.*
```

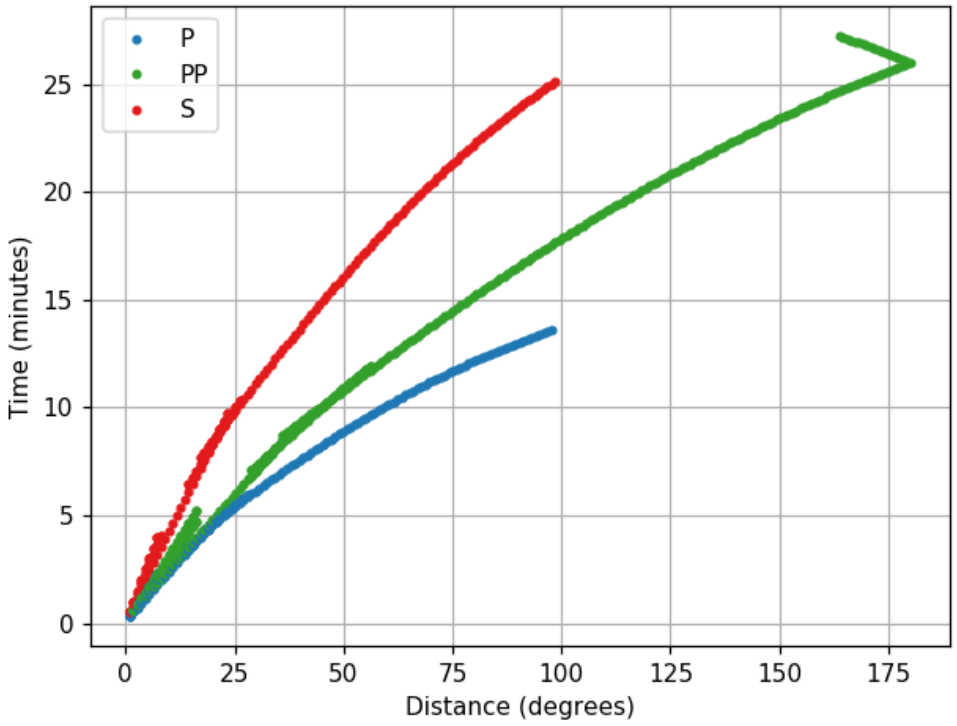


28. Travel Time and Ray Path Plotting（走时和射线路径绘制）

1) 走时绘制

下面的代码展示如何 plot_travel_times()函数绘制给定距离和相位的使用 iasp91 速度模型计算出的走时。

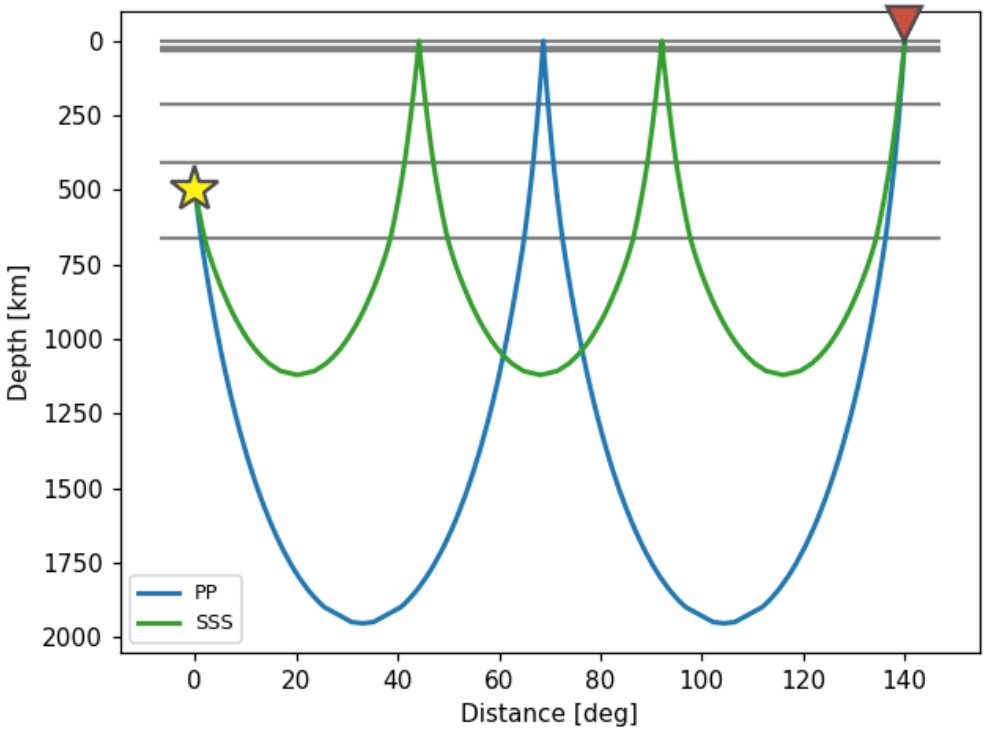
```
from obspy.taup import plot_travel_times
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
ax = plot_travel_times(source_depth=10, ax=ax, fig=fig,
                       phase_list=['P', 'PP', 'S'], npoints=200)
```



2) 笛卡尔射线路径

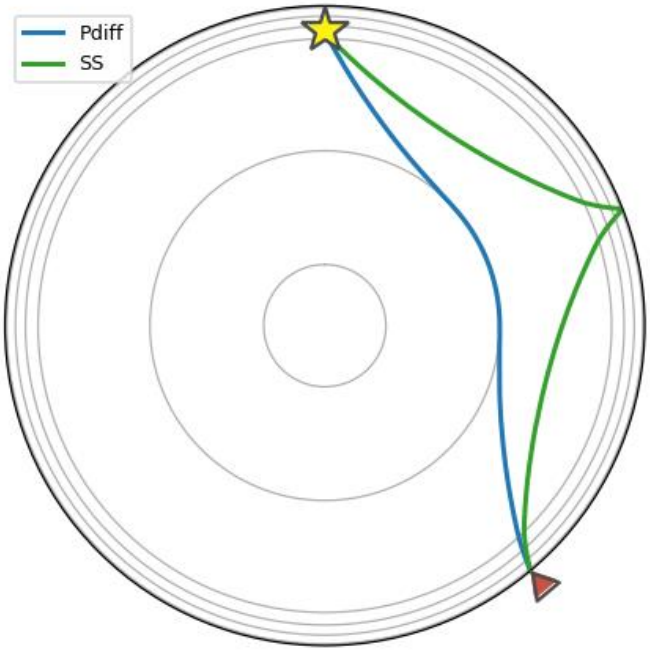
下面的几行代码展示了如何绘制给定距离和相位的射线路径。射线路径使用 iasp91 速度模型计算，并使用 obspy.taup.tau.Arrivals 类的 plot_rays()函数的绘制（在笛卡尔坐标系中）。

```
from obspy.taup import TauPyModel

model = TauPyModel(model='iasp91')
arrivals = model.get_ray_paths(500, 140, phase_list=['PP',
'SSS'])
arrivals.plot_rays(plot_type='cartesian', phase_list=['PP',
'SSS'],
                   plot_all=False, legend=True)
```



3) 球形射线路径

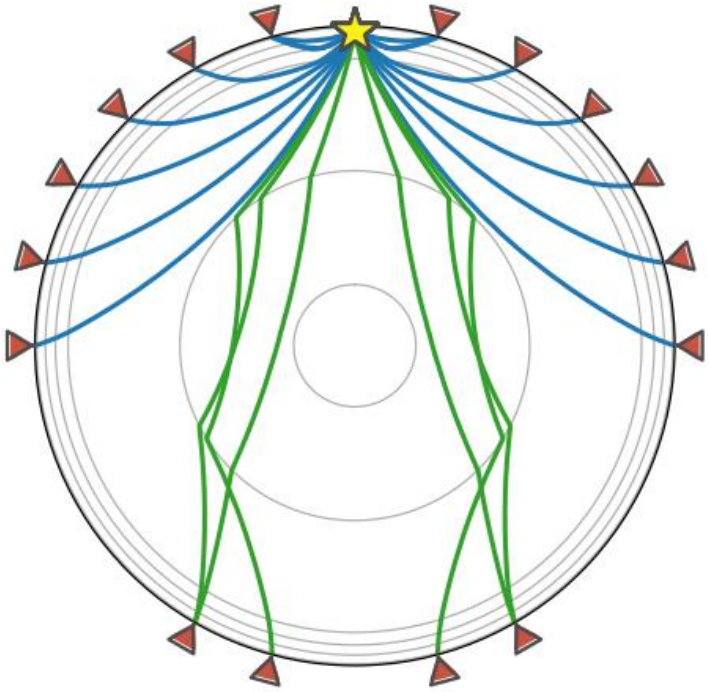
下面的几行代码展示了如何绘制给定距离和相位的射线路径。射线路径使用 iasp91 速度模型计算，并使用 obspy.taup.tau.Arrivals 类的 plot_rays()函数的绘制（在球形图中）。



4) 多距离射线路径

下面的几行代码展示了如何绘制有多个震中距和相位的射线路径。射线路径使用 iasp91 速度模型计算，并使用 obspy.taup.tau.Arrivals 类的 plot_ray_paths()函数的绘制（在球形图中）。

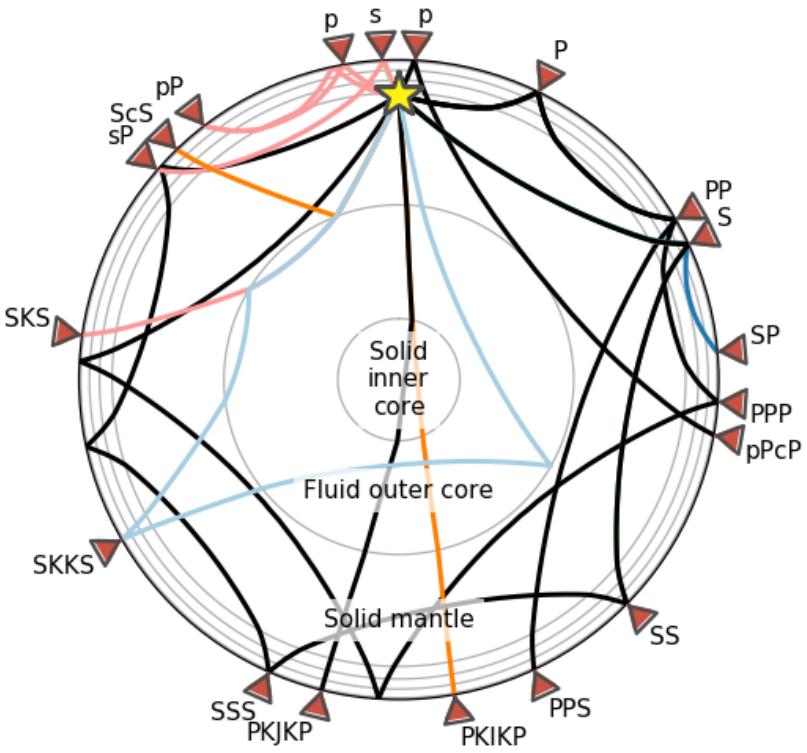
```
from obspy.taup.tau import plot_ray_paths
import matplotlib.pyplot as plt

fig, ax = plt.subplots(subplot_kw=dict(polar=True))
ax = plot_ray_paths(source_depth=100, ax=ax, fig=fig,
phase_list=['P', 'PKP'],
                    npoints=25)
```



对于单个震中距离的射线路径示例，请尝试上一节中的 plot_rays（）方法。以下是一个更高级的示例，其中包含自定义的相位和距离列表：

```
import numpy as np
import matplotlib.pyplot as plt

from obspy.taup import TauPyModel


PHASES = [
    # Phase, distance
    ('P', 26),
    ('PP', 60),
    ('PPP', 94),
    ('PPS', 155),
    ('p', 3),
    ('pPcP', 100),
    ('PKIKP', 170),
    ('PKJKP', 194),
    ('S', 65),
    ('SP', 85),
    ('SS', 134.5),
    ('SSS', 204),
    ('p', -10),
    ('pP', -37.5),
```

```
    ('s', -3),
    ('sP', -49),
    ('ScS', -44),
    ('SKS', -82),
    ('SKKS', -120),
]

model = TauPyModel(model='iasp91')

fig, ax = plt.subplots(subplot_kw=dict(polar=True))

# Plot all pre-determined phases
for phase, distance in PHASES:
    arrivals = model.get_ray_paths(700, distance,
phase_list=[phase])
    ax = arrivals.plot_rays(plot_type='spherical',
                            legend=False, label_arrivals=True,
                            plot_all=True,
                            show=False, ax=ax)

# Annotate regions
ax.text(0, 0, 'Solid\ninner\ncore',
        horizontalalignment='center',
verticalalignment='center',
        bbox=dict(facecolor='white', edgecolor='none',
alpha=0.7))
ocr = (model.model.radius_of_planet -
       (model.model.s_mod.v_mod.iocb_depth +
        model.model.s_mod.v_mod.cmb_depth) / 2)
ax.text(np.deg2rad(180), ocr, 'Fluid outer core',
        horizontalalignment='center',
        bbox=dict(facecolor='white', edgecolor='none',
alpha=0.7))
mr = model.model.radius_of_planet -
model.model.s_mod.v_mod.cmb_depth / 2
ax.text(np.deg2rad(180), mr, 'Solid mantle',
        horizontalalignment='center',
        bbox=dict(facecolor='white', edgecolor='none',
alpha=0.7))

plt.show()
```



29.Cross Correlation Pick Correction（交叉相关拾取校正）

该示例展示如何对齐两个地震的起始波形相位，以便纠正在常规分析中无法完全设置一致的原始拾取时间。按照[Deichmann1992]的方法，互相关函数的凹陷部分最大值附近可用抛物线拟合。

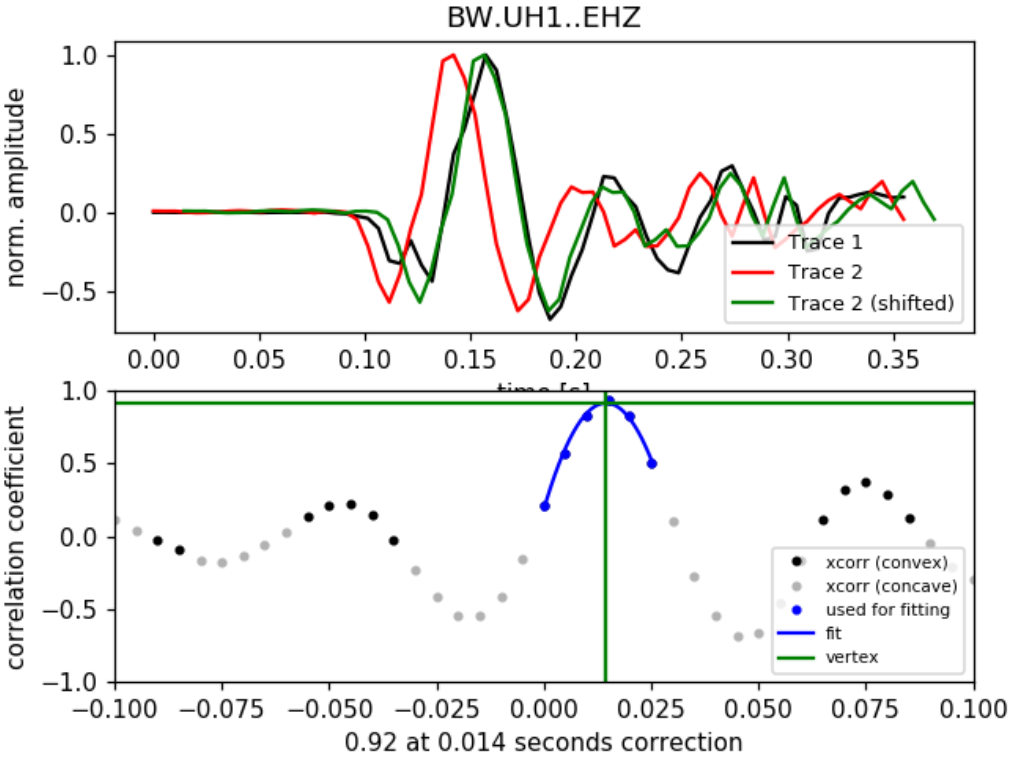
为调整参数并验证检查结果，可以选择展示图形或者将其存为图像文件。参见 xcorr_pick_correction()。

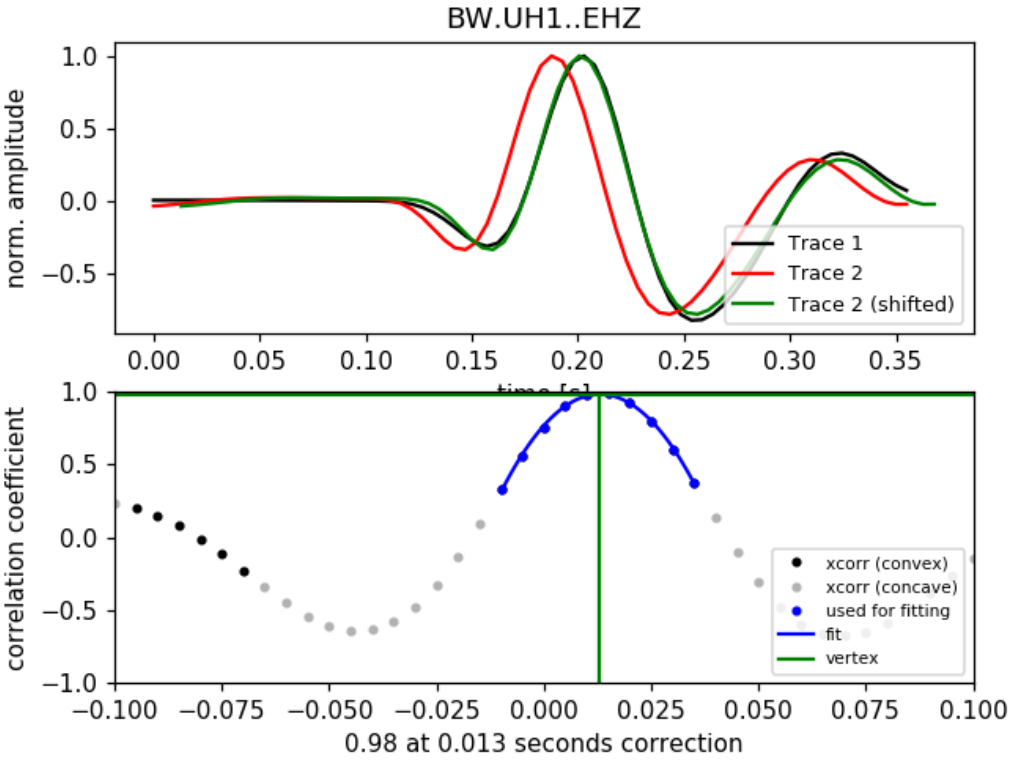
该示例将打印拾取序列 2 的时间校正和相应的相关系数，并打开原始和预处理数据相关性的绘图窗口：

```
No preprocessing:
  Time correction for pick 2: -0.014459
  Correlation coefficient: 0.92
Bandpass prefiltering:
  Time correction for pick 2: -0.013025
  Correlation coefficient: 0.98
```

```
from __future__ import print_function

import obspy
from obspy.signal.cross_correlation import
xcorr_pick_correction

```

```
# read example data of two small earthquakes
path =
"https://examples.obspy.org/BW.UH1..EHZ.D.2010.147.%s.slist.gz
"
st1 = obspy.read(path % ("a", ))
st2 = obspy.read(path % ("b", ))
# select the single traces to use in correlation.
# to avoid artifacts from preprocessing there should be some
data left and
# right of the short time window actually used in the
correlation.
tr1 = st1.select(component="Z")[0]
tr2 = st2.select(component="Z")[0]
# these are the original pick times set during routine
analysis
t1 = obspy.UTCDateTime("2010-05-27T16:24:33.315000Z")
t2 = obspy.UTCDateTime("2010-05-27T16:27:30.585000Z")

# estimate the time correction for pick 2 without any
preprocessing and open
# a plot window to visually validate the results
dt, coeff = xcorr_pick_correction(t1, tr1, t2, tr2, 0.05, 0.2,
0.1, plot=True)
print("No preprocessing:")
print("  Time correction for pick 2: %.6f" % dt)
print("  Correlation coefficient: %.2f" % coeff)
# estimate the time correction with bandpass prefiltering
dt, coeff = xcorr_pick_correction(t1, tr1, t2, tr2, 0.05, 0.2,
0.1, plot=True,
                                  filter="bandpass",
                                  filter_options={'freqmin': 1,
'freqmax': 10})
print("Bandpass prefiltering:")
print("  Time correction for pick 2: %.6f" % dt)
print("  Correlation coefficient: %.2f" % coeff)
```


BW.UH1..EHZ
norm. amplitude
Trace 1
Trace 2
Trace 2 (shifted)
correlation coefficient
xcorr (convex)
xcorr (concave)
used for fitting
fit
vertex
0.92 at 0.014 seconds correction


BW.UH1..EHZ
norm. amplitude
Trace 1
Trace 2
Trace 2 (shifted)
correlation coefficient
xcorr (convex)
xcorr (concave)
used for fitting
fit
vertex
0.98 at 0.013 seconds correction

30.Handling custom defined tags in QuakeML and the ObsPy Catalog/Event framework（在 QuakeML 和 ObsPy 的目录/事件框架中处理自定义标记）

除了 QuakeML 标准定义的“usual”信息之外，QuakeML 还允许使用自定义元素。它允许：

- a 自定义命名空间属性到 QuakeML 命名空间标记
- b 自定义命名空间子标签到 QuakeML 命名空间元素

ObsPy 可以在输入/输出 QuakeML 的过程中处理事件类型对象中的基本自定义标记和自定义属性。

以下基本示例说明如何使用自定义 xml 标记/属性输出有效的 QuakeML 文件：

```
from obspy import Catalog, UTCDateTime

extra = {'my_tag': {'value': True,
                    'namespace': 'http://some-
page.de/xmlns/1.0',
                    'attrib': {'{http://some-
page.de/xmlns/1.0}my_attrib1': '123.4',
                               '{http://some-
page.de/xmlns/1.0}my_attrib2': '567'}},
         'my_tag_2': {'value': u'True',
                      'namespace': 'http://some-
page.de/xmlns/1.0'},
         'my_tag_3': {'value': 1,
                      'namespace': 'http://some-
page.de/xmlns/1.0'},
         'my_tag_4': {'value': UTCDateTime('2013-01-
02T13:12:14.600000Z'),
                      'namespace': 'http://test.org/xmlns/0.1'},
         'my_attribute': {'value': 'my_attribute_value',
                          'type': 'attribute',
                          'namespace':
'http://test.org/xmlns/0.1'}}

cat = Catalog()
cat.extra = extra
cat.write('my_catalog.xml', format='QUAKEML',
          nsmap={'my_ns': 'http://test.org/xmlns/0.1'})
```

所有储存在自定义 QuakeML 中的信息必须存为带有自定义信息（如 Catalog， Event，Pick）的 extra 属性的 dict 或 AttribDict 对象。关键字用作 xml 标记的名称，xml 标记的内容

在一个字典中定义:'value'定义标记的内容(对象的字符串表示形式存储在文本 xml 输出中)。'namespace'必须为标记指定自定义命名空间。 'type'可用于指定额外信息是应存储为子元素('element', default)还是存储为属性('attribute')。 自定义子元素的属性可以以字典的形式提供为'attrib'。如果需要更好的可读性,输出 xml 中的命名空间缩写可以在输出期间指定为 QuakeML,方法是将命名空间缩写映射字典作为 nsmap 参数提供给 Catalog.write()。上例的 xml 输出如下所示:

```
<?xml version='1.0' encoding='utf-8'?>
<q:quakeml xmlns:q='http://quakeml.org/xmlns/quakeml/1.2'
          xmlns:ns0='http://some-page.de/xmlns/1.0'
          xmlns:my_ns='http://test.org/xmlns/0.1'
          xmlns='http://quakeml.org/xmlns/bed/1.2'>
  <eventParameters publicID='smi:local/b425518c-9445-40c7-8284-
d1f299ed2eac'
                   my_ns:my_attribute='my_attribute_value'>
    <ns0:my_tag ns0:my_attrib1='123.4'
ns0:my_attrib2='567'>true</ns0:my_tag>
    <my_ns:my_tag_4>2013-01-
02T13:12:14.600000Z</my_ns:my_tag_4>
    <ns0:my_tag_2>True</ns0:my_tag_2>
    <ns0:my_tag_3>1</ns0:my_tag_3>
  </eventParameters>
</q:quakeml>
```

再次使用 read_events()读取上述 xml 时,自定义标记被解析并附加到相应的事件类型对象(在此示例中为 Catalog 对象)为".extra"。 请注意,所有值都作为文本字符串读取:

```
from obspy import read_events

cat = read_events('my_catalog.xml')
print(cat.extra)
```

```
AttribDict({u'my_tag': {u'attrib': {'{http://some-
page.de/xmlns/1.0}my_attrib2': '567',
                                    '{http://some-
page.de/xmlns/1.0}my_attrib1': '123.4'},
                        u'namespace': u'http://some-
page.de/xmlns/1.0',
                        u'value': 'true'},
            u'my_tag_4': {u'namespace':
u'http://test.org/xmlns/0.1',
                          u'value': '2013-01-
02T13:12:14.600000Z'},
            u'my_attribute': {u'type': u'attribute',
                              u'namespace':
u'http://test.org/xmlns/0.1',
                              u'value': 'my_attribute_value'},
```

```
           u'my_tag_2': {u'namespace': u'http://some-
page.de/xmlns/1.0',
                         u'value': 'True'},
           u'my_tag_3': {u'namespace': u'http://some-
page.de/xmlns/1.0',
                         u'value': '1'}})
```

自定义标签可以嵌套：

```
from obspy import Catalog
from obspy.core import AttribDict

ns = 'http://some-page.de/xmlns/1.0'

my_tag = AttribDict()
my_tag.namespace = ns
my_tag.value = AttribDict()

my_tag.value.my_nested_tag1 = AttribDict()
my_tag.value.my_nested_tag1.namespace = ns
my_tag.value.my_nested_tag1.value = 1.23E+10

my_tag.value.my_nested_tag2 = AttribDict()
my_tag.value.my_nested_tag2.namespace = ns
my_tag.value.my_nested_tag2.value = True

cat = Catalog()
cat.extra = AttribDict()
cat.extra.my_tag = my_tag
cat.write('my_catalog.xml', 'QUAKEML')
```

这将产生类似于以下内容的 xml 输出：

```
<?xml version='1.0' encoding='utf-8'?>
<q:quakeml xmlns:q='http://quakeml.org/xmlns/quakeml/1.2'
           xmlns:ns0='http://some-page.de/xmlns/1.0'
           xmlns='http://quakeml.org/xmlns/bed/1.2'>
  <eventParameters publicID='smi:local/97d2b338-0701-41a4-9b6b-
5903048bc341'>
    <ns0:my_tag>
      <ns0:my_nested_tag1>12300000000.0</ns0:my_nested_tag1>
      <ns0:my_nested_tag2>true</ns0:my_nested_tag2>
    </ns0:my_tag>
  </eventParameters>
</q:quakeml>
```

输出的 XML 可以再次使用 read_events()读取，嵌套标签可使用下面的方法检索：

```
from obspy import read_events

```

```
cat = read_events('my_catalog.xml')
print(cat.extra.my_tag.value.my_nested_tag1.value)
print(cat.extra.my_tag.value.my_nested_tag2.value)
```

结果：

```
12300000000.0
true
```

可以使用 OrderedDict 为额外属性控制额外标签的顺序：

```
from collections import OrderedDict
from obspy.core.event import Catalog, Event

ns = 'http://some-page.de/xmlns/1.0'

my_tag1 = {'namespace': ns, 'value': 'some value 1'}
my_tag2 = {'namespace': ns, 'value': 'some value 2'}

event = Event()
cat = Catalog(events=[event])
event.extra = OrderedDict()
event.extra['myFirstExtraTag'] = my_tag2
event.extra['mySecondExtraTag'] = my_tag1
cat.write('my_catalog.xml', 'QUAKEML')
```

31.Handling custom defined tags in StationXML with the Obspy Inventory（使用 Obspy Inventory 处理 StationXML 中的自定义标签）

以下基本示例说明如何输出包含其他 xml 标记/属性的 StationXML 文件：

```
from obspy import Inventory, UTCDateTime
from obspy.core.inventory import Network
from obspy.core.util import AttribDict

extra = AttribDict({
          'my_tag': {
              'value': True,
              'namespace': 'http://some-page.de/xmlns/1.0',
              'attrib': {
                '{http://some-page.de/xmlns/1.0}my_attrib1':
'123.4',
                '{http://some-page.de/xmlns/1.0}my_attrib2':
```

```
'567'
                }
            },
            'my_tag_2': {
                'value': u'True',
                'namespace': 'http://some-page.de/xmlns/1.0'
            },
            'my_tag_3': {
                'value': 1,
                'namespace': 'http://some-page.de/xmlns/1.0'
            },
            'my_tag_4': {
                'value': UTCDateTime('2013-01-
02T13:12:14.600000Z'),
                'namespace': 'http://test.org/xmlns/0.1'
            },
            'my_attribute': {
                'value': 'my_attribute_value',
                'type': 'attribute',
                'namespace': 'http://test.org/xmlns/0.1'
            }
        })

inv = Inventory([Network('XX')], 'XX')
inv[0].extra = extra
inv.write('my_inventory.xml', format='STATIONXML',
          nsmap={'my_ns': 'http://test.org/xmlns/0.1',
                 'somepage_ns': 'http://some-
page.de/xmlns/1.0'})
```

要存储在定制的 StationXML 中的所有自定义信息必须以 dict 或 AttribDict 对象的形式存储，作为应携带附加自定义信息的对象的.extra 属性（例如，Network, Station, Channel）。关键字作 xml 标记的名称，xml 标记的内容在简单字典中定义：'value'定义标记的内容（对象的字符串表示形式存储在文本 xml 输出中）。 'namespace'必须为标记指定自定义命名空间。 'type'可用于指定额外信息是应存储为子元素（'element'，default）还是存储为属性（'attribute'）。 自定义子元素的属性可以以字典的形式提供为“attrib”。

如果需要更好可读性，输出 xml 中的命名空间缩写可以在输出期间指定为 StationXML，方法是将名称空间缩写映射字典作为 nsmap 参数提供给 Inventory.write()。xml 输出如下所示：

```
<?xml version='1.0' encoding='UTF-8'?>
<FDSNStationXML xmlns:my_ns="http://test.org/xmlns/0.1"
xmlns:somepage_ns="http://some-page.de/xmlns/1.0"
xmlns="http://www.fdsn.org/xml/station/1" schemaVersion="1.0">
  <Source>XX</Source>
  <Module>ObsPy 1.0.2</Module>
```

```
  <ModuleURI>https://www.obspy.org</ModuleURI>
  <Created>2016-10-17T18:32:28.696287+00:00</Created>
  <Network code="XX">
    <somepage_ns:my_tag somepage_ns:my_attrib1="123.4"
somepage_ns:my_attrib2="567">True</somepage_ns:my_tag>
    <my_ns:my_tag_4>2013-01-
02T13:12:14.600000Z</my_ns:my_tag_4>
    <my_ns:my_attribute>my_attribute_value</my_ns:my_attribute>
    <somepage_ns:my_tag_2>True</somepage_ns:my_tag_2>
    <somepage_ns:my_tag_3>1</somepage_ns:my_tag_3>
  </Network>
</FDSNStationXML>
```

再次读取上面的 XML 时，使用 read_inventory()，自定义标记被解析并作为为'.extra'附加到相应的网络类型对象（在此示例中为 Inventory 对象）。请注意，所有值都作为文本字符串读取：

```
from obspy import read_inventory

inv = read_inventory('my_inventory.xml')
print(inv[0].extra)
```

```
AttribDict({
    u'my_tag': AttribDict({
        'attrib': {
          '{http://some-page.de/xmlns/1.0}my_attrib2': '567',
          '{http://some-page.de/xmlns/1.0}my_attrib1': '123.4'
        },
        'namespace': 'http://some-page.de/xmlns/1.0',
        'value': 'True'
    }),
    u'my_tag_4': AttribDict({
        'namespace': 'http://test.org/xmlns/0.1',
        'value': '2013-01-02T13:12:14.600000Z'
    }),
    u'my_attribute': AttribDict({
        'namespace': 'http://test.org/xmlns/0.1',
        'value': 'my_attribute_value'
    }),
    u'my_tag_2': AttribDict({
        'namespace': 'http://some-page.de/xmlns/1.0',
        'value': 'True'
    }),
    u'my_tag_3': AttribDict({
        'namespace': 'http://some-page.de/xmlns/1.0',
        'value': '1'
```

```
    })
})
```

自定义标签可以嵌套：

```
from obspy import Inventory
from obspy.core.inventory import Network
from obspy.core.util import AttribDict

ns = 'http://some-page.de/xmlns/1.0'

my_tag = AttribDict()
my_tag.namespace = ns
my_tag.value = AttribDict()

my_tag.value.my_nested_tag1 = AttribDict()
my_tag.value.my_nested_tag1.namespace = ns
my_tag.value.my_nested_tag1.value = 1.23E+10

my_tag.value.my_nested_tag2 = AttribDict()
my_tag.value.my_nested_tag2.namespace = ns
my_tag.value.my_nested_tag2.value = True

inv = Inventory([Network('XX')], 'XX')
inv[0].extra = AttribDict()
inv[0].extra.my_tag = my_tag
inv.write('my_inventory.xml', format='STATIONXML',
          nsmap={'somepage_ns': 'http://some-
page.de/xmlns/1.0'})
```

这将产生类似于以下内容的 xml 输出：

```
<?xml version='1.0' encoding='UTF-8'?>
<FDSNStationXML xmlns:somepage_ns="http://some-
page.de/xmlns/1.0" xmlns="http://www.fdsn.org/xml/station/1"
schemaVersion="1.0">
  <Source>XX</Source>
  <Module>ObsPy 1.0.2</Module>
  <ModuleURI>https://www.obspy.org</ModuleURI>
  <Created>2016-10-17T18:45:14.302265+00:00</Created>
  <Network code="XX">
    <somepage_ns:my_tag>

<somepage_ns:my_nested_tag1>12300000000.0</somepage_ns:my_nest
ed_tag1>

<somepage_ns:my_nested_tag2>True</somepage_ns:my_nested_tag2>
```

```
    </somepage_ns:my_tag>
  </Network>
</FDSNStationXML>
```

输出 XML 可以使用 read_inventory()再次读取，嵌套标签使用下面的方法检索：

```
from obspy import read_inventory

inv = read_inventory('my_inventory.xml')
print(inv[0].extra.my_tag.value.my_nested_tag1.value)
print(inv[0].extra.my_tag.value.my_nested_tag2.value)
```

结果：

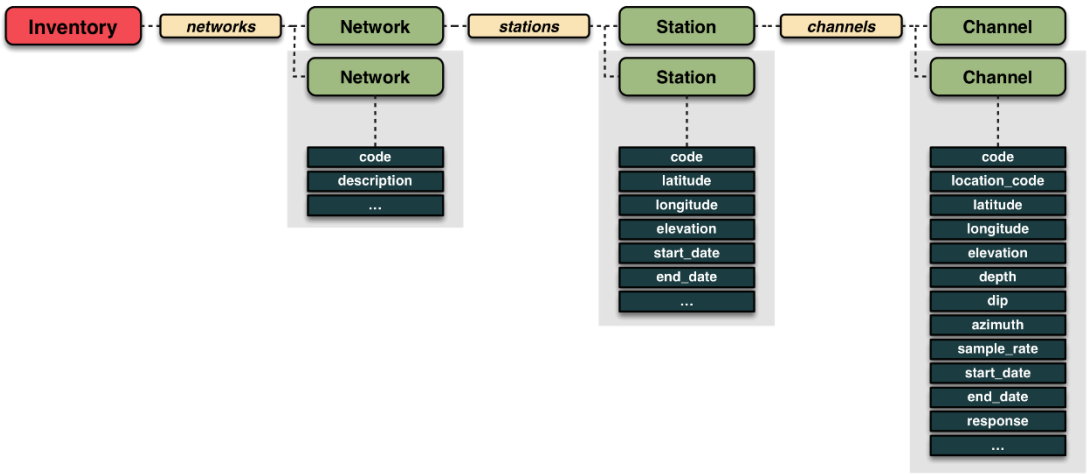
```
12300000000.0
True
```

32. Creating a StationXML file from Scratch

（从 Scratch 创建 StationXML 文件）

在地震学中有时需要创建自定义 StationXML 文件。本节演示如何使用 ObsPy 完成这样的任务请注意，这不一定比直接编辑 XML 文件更容易或更容易观察，但它确实提供了一种与 ObsPy 的其余部分更紧密结合的方法，并且可以保证最终结果有效。

这里假定您对 FDSN StationXML standard.有一定的了解。我们将创建一个相当简单的 StationXML 文件，并且许多参数是可选的。 ObsPy 将在写入时根据其模式验证生成的 StationXML 文件，以确保最终文件对 StationXML 模式有效。下图显示了 ObsPy 内部表示的基本结构。



每个大框都是一个对象，所有对象都必须分层链接以形成一个 Inventory 对象。Inventory 可以包含任意数量的网络对象，这些对象又可以包含任意数量的工作站对象，这些工具又可以包含任意数量的信道对象。对于每个信道，仪器响应可以存储为 response 属性。

使用 ObsPy 的 NRL 客户端，可以从 IRIS DMC Library of Nominal Responses（NRL）中查找仪器响应并将其附加到每个信道。

```
import obspy
from obspy.core.inventory import Inventory, Network, Station,
Channel, Site
from obspy.clients.nrl import NRL


# We'll first create all the various objects. These strongly
follow the
# hierarchy of StationXML files.
inv = Inventory(
    # We'll add networks later.
    networks=[],
    # The source should be the id whoever create the file.
    source="ObsPy-Tutorial")

net = Network(
    # This is the network code according to the SEED standard.
    code="XX",
    # A list of stations. We'll add one later.
    stations=[],
    description="A test stations.",
    # Start-and end dates are optional.
    start_date=obspy.UTCDateTime(2016, 1, 2))

sta = Station(
    # This is the station code according to the SEED standard.
    code="ABC",
    latitude=1.0,
    longitude=2.0,
    elevation=345.0,
    creation_date=obspy.UTCDateTime(2016, 1, 2),
    site=Site(name="First station"))

cha = Channel(
    # This is the channel code according to the SEED standard.
    code="HHZ",
    # This is the location code according to the SEED standard.
    location_code="",
    # Note that these coordinates can differ from the station
coordinates.
    latitude=1.0,
    longitude=2.0,
    elevation=345.0,
    depth=10.0,
```

```
    azimuth=0.0,
    dip=-90.0,
    sample_rate=200)

# By default this accesses the NRL online. Offline copies of
the NRL can
# also be used instead
nrl = NRL()
# The contents of the NRL can be explored interactively in a
Python prompt,
# see API documentation of NRL submodule:
# http://docs.obspy.org/packages/obspy.clients.nrl.html
# Here we assume that the end point of data logger and sensor
are already
# known:
response = nrl.get_response( # doctest: +SKIP
    sensor_keys=['Streckeisen', 'STS-1', '360 seconds'],
    datalogger_keys=['REF TEK', 'RT 130 & 130-SMA', '1',
'200'])


# Now tie it all together.
cha.response = response
sta.channels.append(cha)
net.stations.append(sta)
inv.networks.append(net)

# And finally write it to a StationXML file. We also force a
validation against
# the StationXML schema to ensure it produces a valid
StationXML file.
#
# Note that it is also possible to serialize to any of the
other inventory
# output formats ObsPy supports.
inv.write("station.xml", format="stationxml", validate=True)
```

33. Connecting to a SeedLink Server（连接到 SeedLink 服务器）

obspy.clients.seedlink 模块提供了 Python 实现的 SeedLink 客户端协议。obspy.clients.seedlink.easyseedlink 子模块包含 SeedLink 实现的高级接口，有助于创建 SeedLink

客户端。

1) create_client 函数

使用 create_client()函数创建一个新的 EasySeedLinkClient 类是连接到 SeedLink 服务器的最简单方法。它接受一个从 SeedLink 服务器接收的新数据的函数作为参数，例如：

```
def handle_data(trace):
    print('Received the following trace:')
    print(trace)
    print()
```

此函数随后可以与 SeedLink 服务器 URL 一同传递给 create_client()创建一个客户端：

```
client = create_client('geofon.gfz-potsdam.de',
on_data=handle_data)
```

客户端在创建时立即连接到服务器。

a) 发送 INFO 请求到服务器

客户端可以发送 INFO 请求到服务器：

```
# Send the INFO:ID request
client.get_info('ID')

# Returns:
# <?xml version="1.0"?>\n<seedlink software="SeedLink v3.2
(2014.071)" organization="GEOFON" started="2014/09/01
14:08:37.4192"/>\n
```

INFO 请求的响应是 XML 格式的。客户端提供了检索和解析服务器功能的快捷方式（经由一个 INFO:CAPABILITIES 请求。）

```
>>> client.capabilities
['dialup', 'multistation', 'window-extraction', 'info:id',
'info:capabilities', 'info:stations', 'info:streams']
```

首次访问属性时会提取和解析这些功能，并在此之后进行缓存。

b) 从服务器传输数据流

为了开始接受波形数据，需要先通过 select_stream()函数选择至少一个 stream。

```
client.select_stream('BW', 'MANZ', 'EHZ')
```

多个 stream 也可以被选择，也支持 SeedLink 通配符：

```
client.select_stream('BW', 'ROTZ', 'EH?')
```

选择 stream 后，客户端就准备好进入 streaming 模式了：

```
client.run()
```

这将从服务器开始流式传输数据。 在从服务器接收的每个完整跟踪时，使用跟踪对象调用上面定义的函数：

```
Received new data:
BW.MANZ..EHZ | 2014-09-04T19:47:25.625000Z - 2014-09-
04T19:47:26.770000Z | 200.0 Hz, 230 samples

Received new data:
BW.ROTZ..EHZ | 2014-09-04T19:47:22.685000Z - 2014-09-
04T19:47:24.740000Z | 200.0 Hz, 412 samples

Received new data:
BW.ROTZ..EHZ | 2014-09-04T19:47:24.745000Z - 2014-09-
04T19:47:26.800000Z | 200.0 Hz, 412 samples

Received new data:
BW.ROTZ..EHN | 2014-09-04T19:47:20.870000Z - 2014-09-
04T19:47:22.925000Z | 200.0 Hz, 412 samples

Received new data:
BW.ROTZ..EHN | 2014-09-04T19:47:22.930000Z - 2014-09-
04T19:47:24.985000Z | 200.0 Hz, 412 samples
```

2) 高级用法：子类化客户端

对于高级用法，继承 EasySeedLinkClient 类可以更好地控制。下列代码实现与上面相同的客户端：

```
class DemoClient(EasySeedLinkClient):
    """
    A custom SeedLink client
    """
    def on_data(self, trace):
        """
        Override the on_data callback
        """
        print('Received trace:')
        print(trace)
        print()
```