

微動アレイ観測データ 処理スクリプト PyArrayの紹介

京都大学 防災研究所 後藤浩之

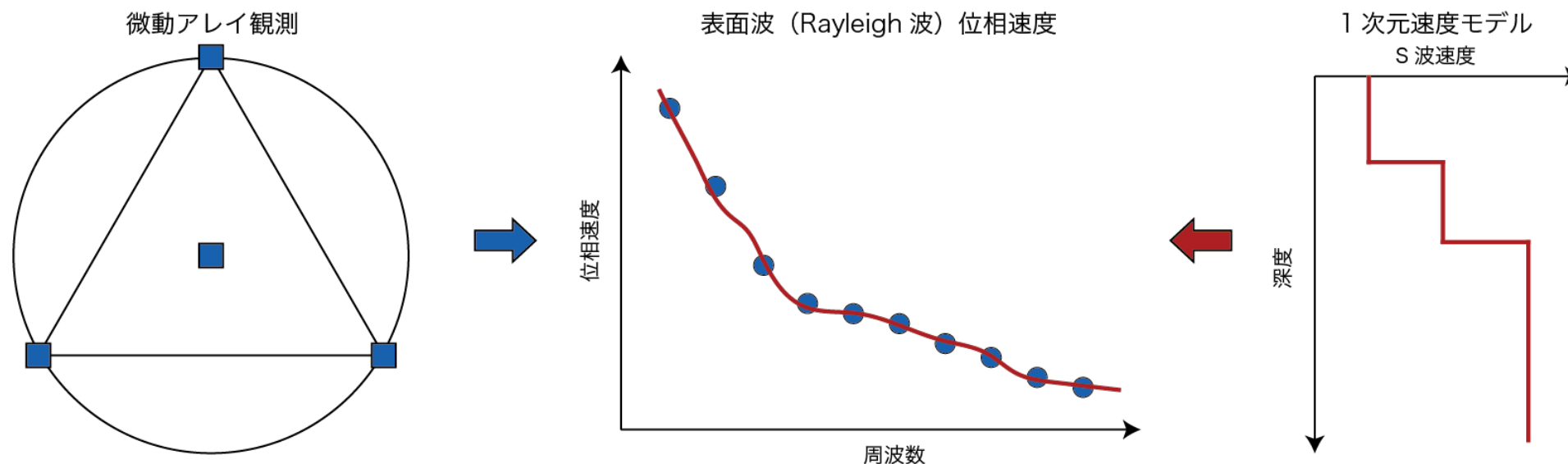


特徴

- 微動アレイ観測のデータ処理を行うためのパッケージ群を含むスクリプト
- [Python](#) (ver.3) パッケージ（一部Fortran）で書かれているので、pythonが動作する環境であればOS（Windows / Mac OS / Linux）は何でも良い

出来ること

1. 微動アレイ観測データからRayleigh波の位相速度（分散曲線）を推定する
2. 1次元速度モデルからRayleigh波の位相速度を計算する（フィッティング）



準備

0. 実行環境の準備

Python (ver.3)といくつかの関連パッケージのインストールが必要です.

- Python
 - NumPy
 - Matplotlib
 - SciPy

Pythonのインストールは [Python環境構築ガイド](#) を参考にしてください.

NumPy, Matplotlib, SciPyのインストールは [pip](#) を利用すると簡単です

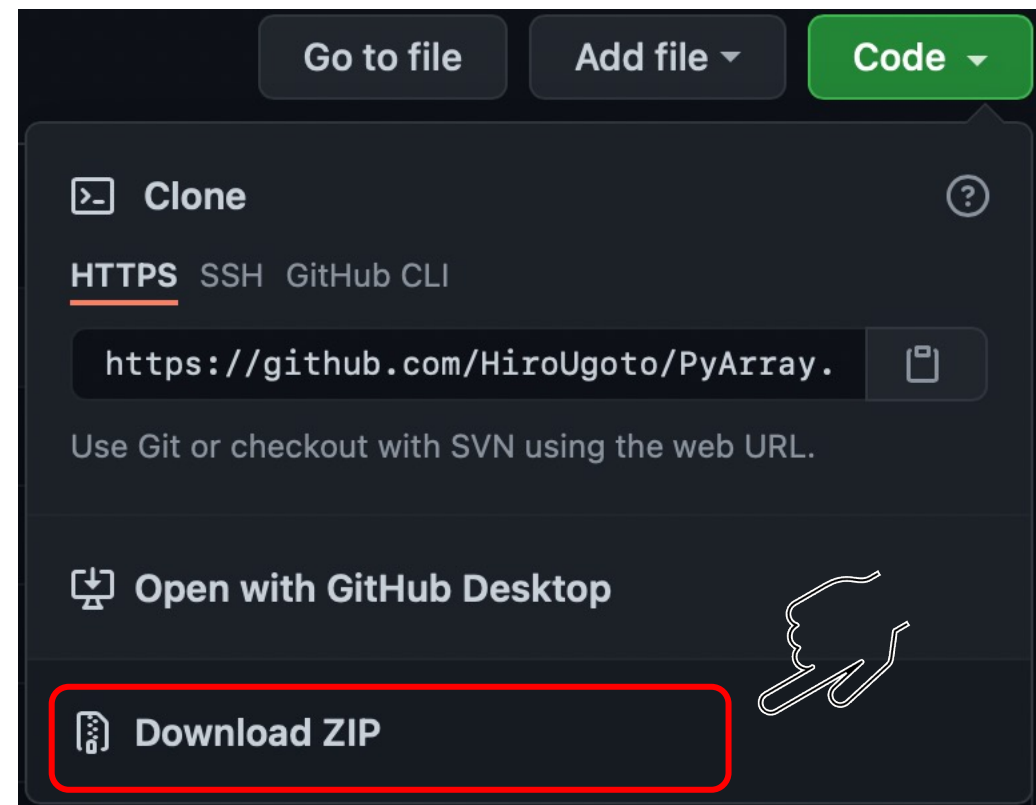
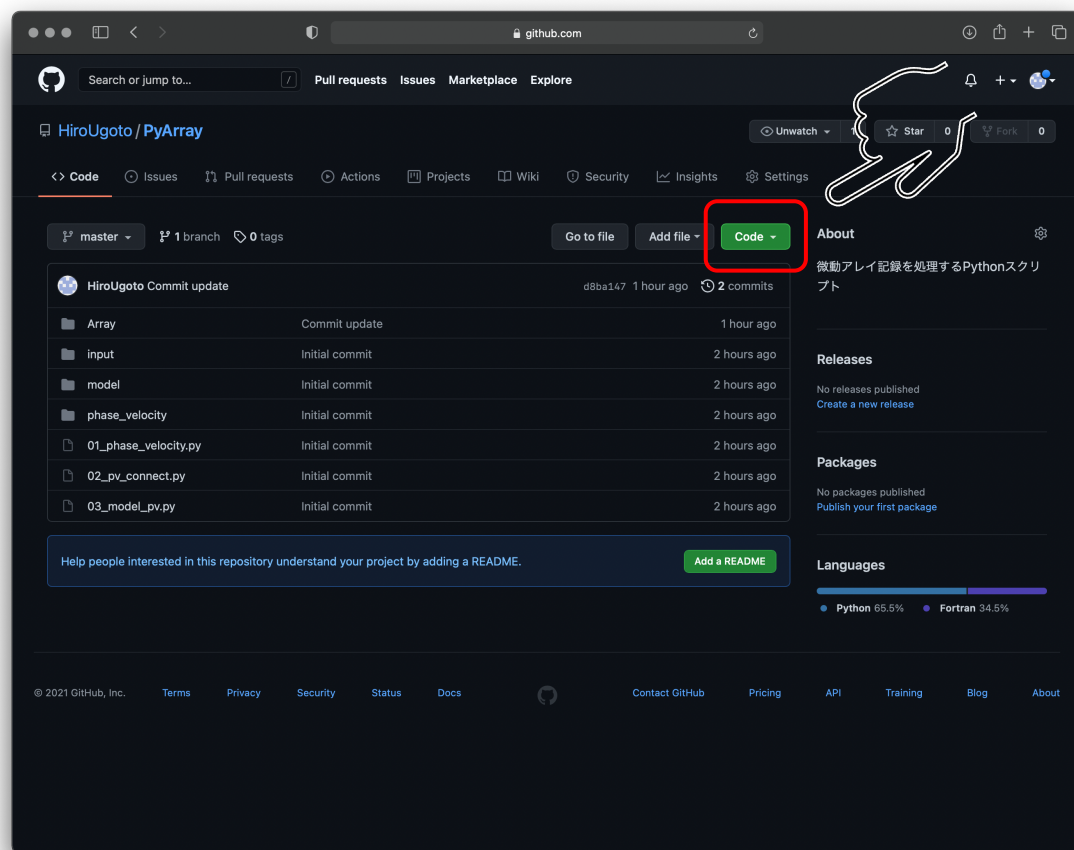
```
> python -m pip install numpy
> python -m pip install matplotlib
> python -m pip install scipy
```

準備

1. スクリプトのダウンロード

以下のURLからスクリプト（データ含む）一式をダウンロードしてください

<https://github.com/HiroUgoto/PyArray>

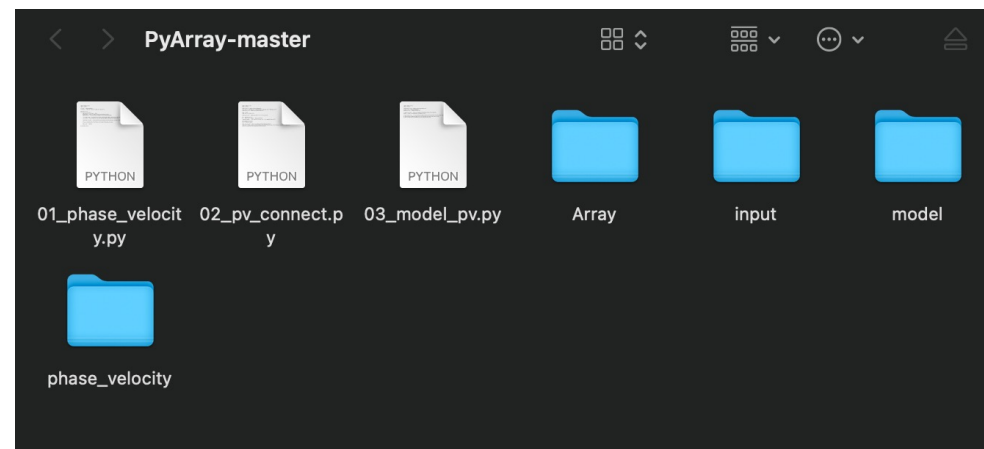


準備

2. ダウンロードファイルの展開

ダウンロードしたファイルを, pythonが使える (任意の) フォルダの中に展開してください.
以下のようなファイル構成となります. 詳細は後ほど説明します

- 01_phase_velocity.py
 - 02_pv_connect.py
 - 03_model_pv.py
 - Array/
 - input/
 - model/
 - phase_velocity/
- 解析用のスクリプト
- パッケージ群
 - 微動観測データと設定ファイル
 - 1次元速度モデル
 - 位相速度



解析手順

以下の手順で解析を進めます.

Step1 アレイサイズ毎に観測データからRayleigh波位相速度を推定する
01_pase_velocity.py

Step2 複数のアレイサイズの位相速度をまとめて1つのRayleigh波位相速度にする
02_pv_connect.py

Step3 1次元速度モデルから位相速度を計算してフィッティングさせる
03_model_pv.py

動作確認しましょう (Step1)

以下のコマンドでスクリプトを動かします

```
> python 01_phase_velocity.py
```

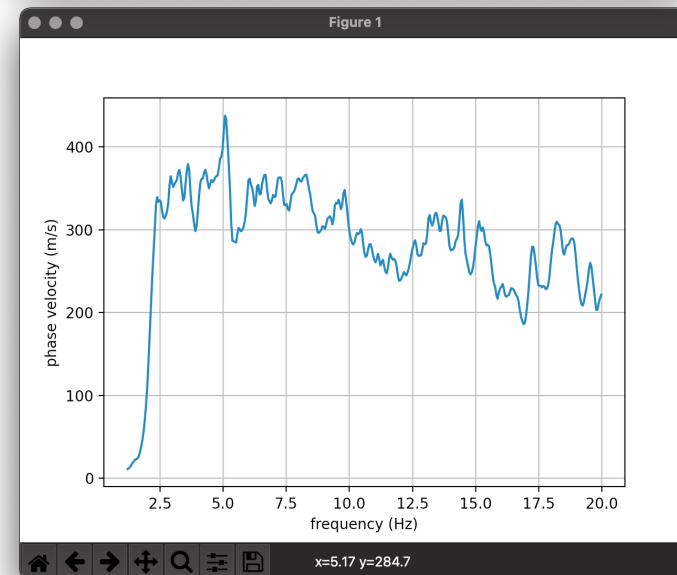
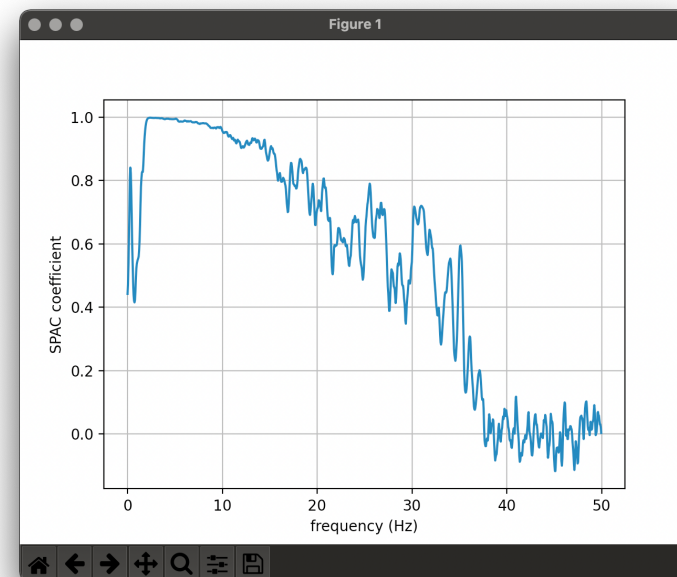
(注意)

環境によっては python ではなく python3 コマンドですので,
python を python3 に置き換えてください.

右のようなグラフが8回出力されれば成功です

(表れたグラフのウィンドウを閉じると次のグラフが表れます)

→ 設定ファイルは後ほど説明します



動作確認しましょう (Step2)

以下のコマンドでスクリプトを動かします

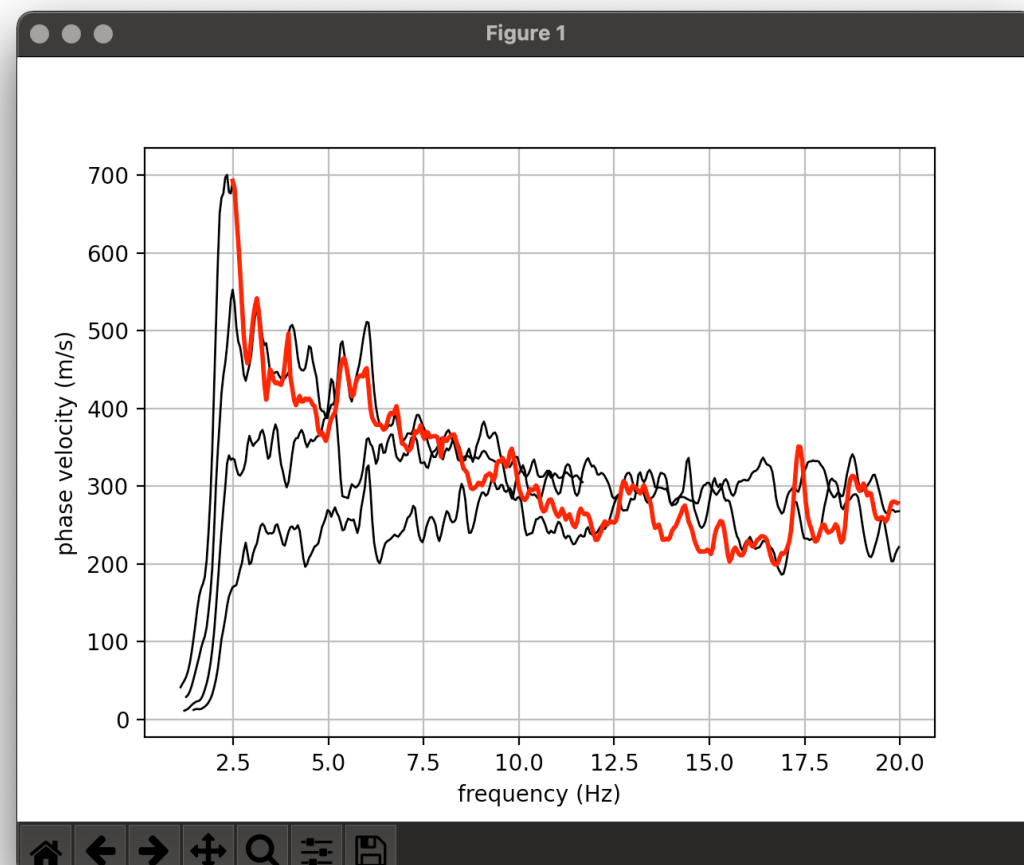
```
> python 02_pv_connect.py
```

(注意)

環境によっては python ではなく python3 コマンドですので, python を python3 に置き換えてください.

右のようなグラフが出力されれば成功です

→ 使い方は後ほど説明します



動作確認しましょう (Step3)

以下のコマンドでスクリプトを動かします

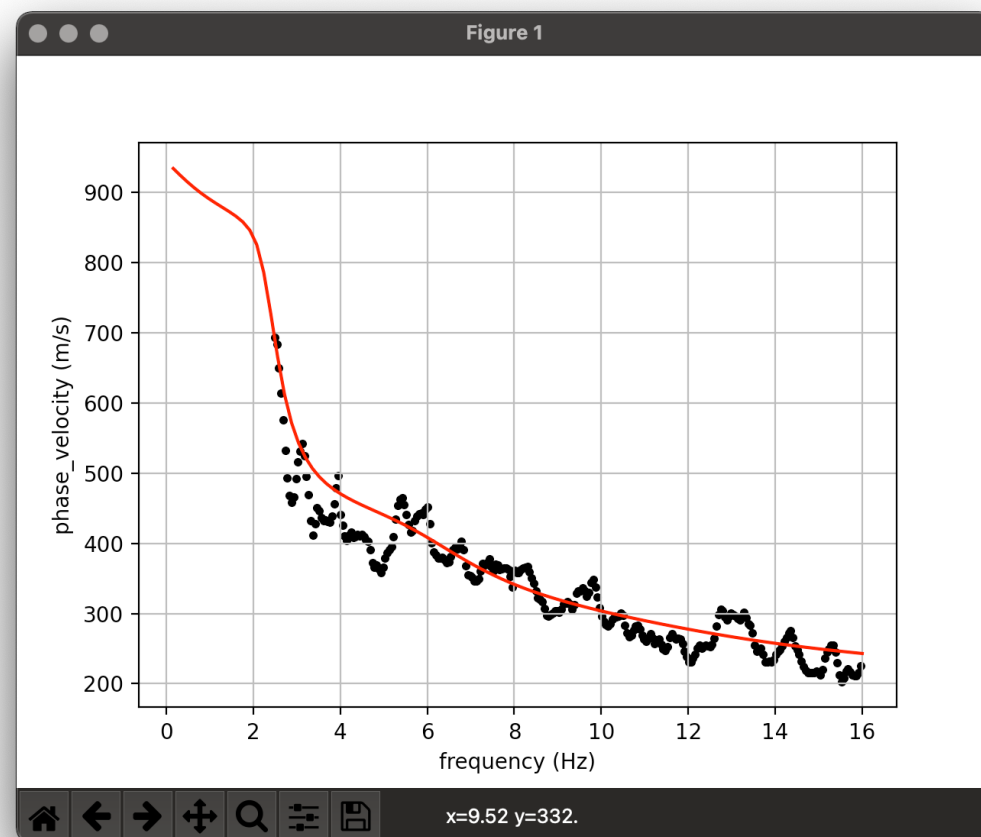
```
> python 03_model_pv.py
```

(注意)

環境によっては python ではなく python3 コマンドですので,
python を python3 に置き換えてください.

右のようなグラフが出力されれば成功です

→ 使い方は後ほど説明します



使い方 (Step1)

Step1

アレイサイズ毎に観測データからRayleigh波位相速度を推定する

使うスクリプト

01_pase_velocity.py

設定ファイル／データファイルのサンプル

input/KYT013

input/OSK002

設定ファイルの準備 (Step1)

- 微動データに関する情報や観測条件を設定します
サンプルファイル (input/KYT013/01.ctf)

```
# data directory
./input/KYT013/data/1m/
# output directory
./phase_velocity/KYT013/1m/
# sampling frequency[Hz]
100
# segment length[s], maximum number of segment
20.48 40
# band width[Hz]
0.3
# Number of sensors
4
# r[m], N?E[deg], file_name, center?[center:1 or arc:0]
0 0 A01.acc 1
1 0 A02.acc 0
1 120 A03.acc 0
1 240 A06.acc 0
```

- 1 微動データの格納フォルダ
スクリプトとの相対位置で記述する
- 2 位相速度の出力先のフォルダ
スクリプトとの相対位置で記述する
用意されていない場合には自動で作成される
- 3 微動データのサンプリング周波数 (Hz)
- 4 微動データ进行处理する際の
セグメント長さ (秒) とセグメント数
- 5 平滑化のバンド幅 (Hz)

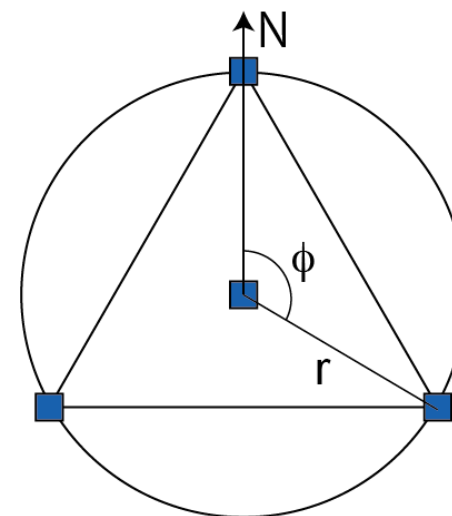
設定ファイルの準備 (Step1)

- 微動データに関する情報や観測条件を設定します
サンプルファイル (input/KYT013/01.ctf)

```
# data directory
./input/KYT013/data/1m/ 1
# output directory
./phase_velocity/KYT013/1m/ 2
# sampling frequency[Hz]
100 3
# segment length[s], maximum number of segment
20.48 40 4
# band width[Hz]
0.3 5
# Number of sensors
4 6
# r[m], N?E[deg], file_name, center?[center:1 or arc:0]
0 0 A01.acc 1
1 0 A02.acc 0
1 120 A03.acc 0 7
1 240 A06.acc 0
```

6 アレイの地震計の数

7 地震計の位置とデータに関する情報
1列目：中心からの距離 (m)
2列目：北からの方位角 (degree)
3列目：データファイル名
1 + このファイル名 で処理されます
4列目：アレイ中心の場合 1, 円周上 0



データファイルの書式 (Step1)

- 微動データファイルの書式

サンプルファイル (input/KYT013/data/1m/A01.acc)

```
3.887310881342740374e-04  
3.255279481342740434e-04  
1.043169581342740372e-04  
-2.749018818657259270e-04  
-7.173238618657258852e-04  
-1.317753691865725801e-03  
-1.949785091865725850e-03  
-2.645019631865725730e-03  
-3.751074581865725815e-03  
-5.015137381865725913e-03  
...
```

上下動成分の振幅値が1列に並んでいるようにする

1つの設定ファイルで扱うデータは、全て時刻が同期されているとして扱われるため、全ての時刻を揃えておくこと

加速度／速度や単位はなんでも良いが、全てのデータで揃えておくこと

スクリプトの使い方 (Step1)

- スクリプトファイルの赤字の箇所を書き換えます

01_phase_velocity.py

```
import numpy as np
import Array
```

設定ファイルのあるフォルダ

```
ctl_dir = "input/KYT013/"
ctl_list = ["01.ctl", "02.ctl", "05.ctl", "10.ctl"]
```

設定ファイルのリスト（複数設定できる）
この場合、1m, 2m, 5m, 10mアレイのデータを
まとめて順に解析します

```
fmin_list = []
for file in ctl_list:
    control_file = ctl_dir + file
    param,data = Array.io.read_control_file(control_file)
    segment_data = Array.analysis.segment_selection(param,data)

    freq,spac_coeff = Array.analysis.spac_coeff(param,segment_data,plot_flag=True)
    _,_,_,fmin = Array.analysis.cca_coeff(param,segment_data,plot_flag=False)

    freq_spac,vel_spac = Array.analysis.spac_phase_velocity(param,freq,spac_coeff,fmin=0.5*fmin,plot_flag=True)

    Array.io.output_data_file("spac.vel",param,freq_spac,vel_spac)

    fmin_list += [fmin]

print(fmin_list)
```

出力結果 (Step1)

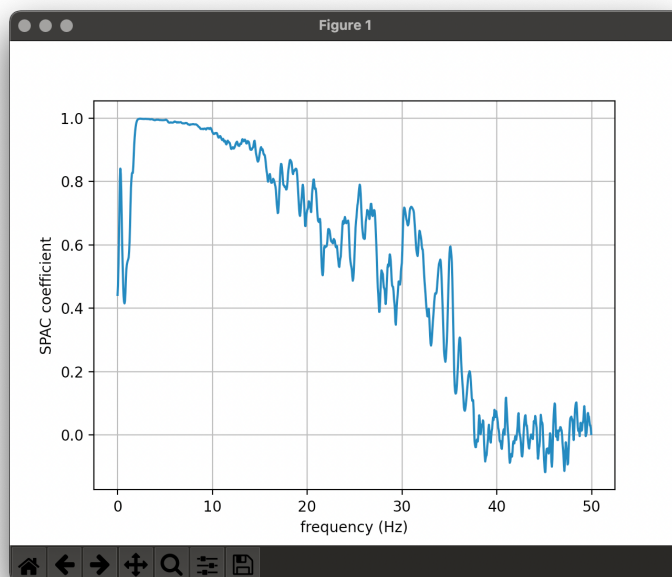
- 設定ファイル毎に, コンソール表示と2つのグラフが表示されます

コンソール表示 (抜粋)

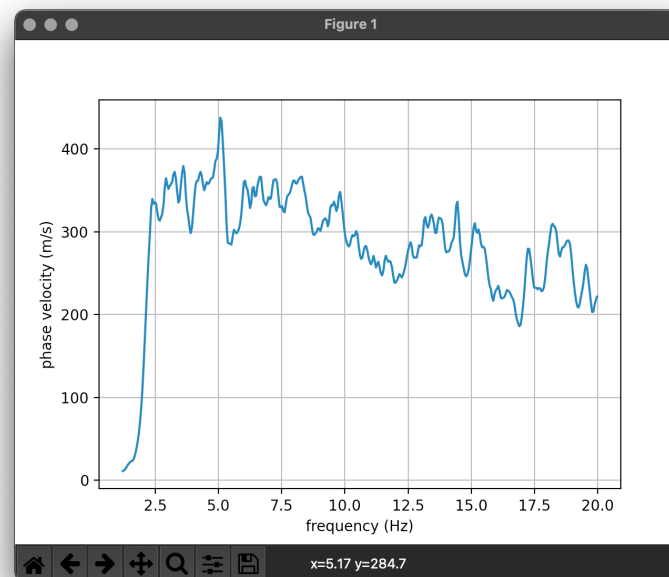
```
-----  
Resolve frequency range  
+ frequency gives minimum NSratio [Hz]      : 2.392578125  
+ frequency not exceeding NSratio 1.0 [Hz]: 0.0  
-----
```

これより高い周波数 (実際にはより高い周波数)
の位相速度の結果しか使えません

周波数—SPAC係数



周波数—位相速度 (位相分散曲線)



出力結果 (Step1)

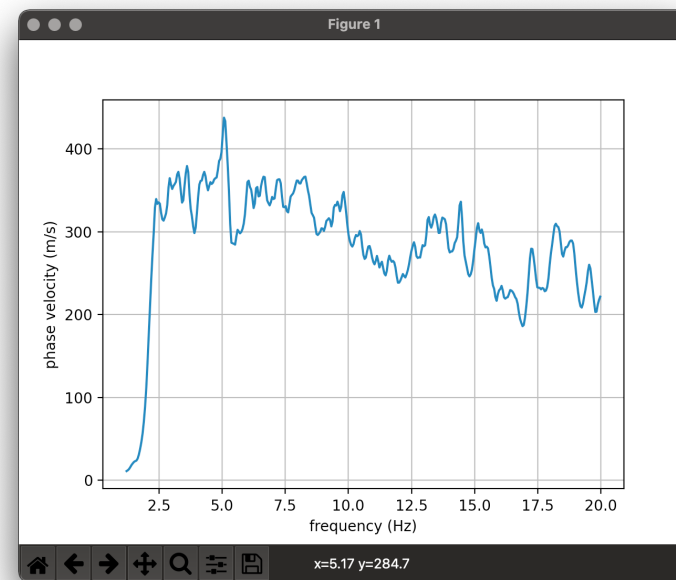
- 設定ファイル毎（アレイ半径毎）に，位相速度がファイルに出力されます
設定ファイルで指定したフォルダに spac.vel として保存されます

```
1.464843750000000000e+00 1.208421858823107975e+01  
1.513671875000000000e+00 1.306094376828892045e+01  
1.562500000000000000e+00 1.328628285218192673e+01  
1.611328125000000000e+00 1.292246085921884990e+01  
1.660156250000000000e+00 1.325660801220996277e+01  
1.708984375000000000e+00 1.460070727384464107e+01  
1.757812500000000000e+00 1.633083666874630069e+01  
1.806640625000000000e+00 1.843556900228933770e+01  
1.855468750000000000e+00 2.160030133953836895e+01  
1.904296875000000000e+00 2.589924619847419862e+01  
...
```

1列目：周波数 (Hz)
2列目：位相速度 (m/s)

周波数一位相速度
(位相分散曲線)

同じもの



使い方 (Step2)

Step2

複数のアレイサイズの位相速度をまとめて1つのRayleigh波位相速度にする

使うスクリプト

02_pv_connect.py

データファイルのサンプル (Step1の出力結果)

phase_velocity/KYT013/

スクリプトの使い方 (Step2)

- スクリプトファイルの赤字の箇所を書き換えます

02_pv_connect.py

位相速度データのあるフォルダ

位相速度データファイルのリスト（複数設定できる）
小さいレイ半径から順に並べること

```
import numpy as np
import Array

input_pv_dir = "phase_velocity/KYT013/"
file_list = ["1m/spac.vel", "2m/spac.vel", "5m/spac.vel", "10m/spac.vel"]
output_pv_file = "phase_velocity/KYT013.vel"

fmax = 20.0
fmin_list = [12, 8, 4, 2.5]

input_pv_files = [input_pv_dir+s for s in file_list]

ns = len(fmin_list)
fr = fmin_list + [fmax] + fmin_list[0:-1]
frequency_range = list(zip(*[fr[i:i+ns] for i in range(0, 2*ns, ns)]))

print(input_pv_files)
print(frequency_range)

freq_list, pv_list = Array.io.read_pv_files(input_pv_files)
freq, vel = Array.analysis.connect_phase_velocity(freq_list, pv_list, frequency_range, plot_flag=True)
Array.io.output_pv_file(output_pv_file, freq, vel)
```

結果を出力するフォルダ

スクリプトの使い方 (Step2)

- スクリプトファイルの赤字の箇所を書き換えます

02_pv_connect.py

```
import numpy as np
import Array

input_pv_dir = "phase_velocity/KYT013/"
file_list = ["1m/spac.vel", "2m/spac.vel", "5m/spac.vel", "10m/spac.vel"]
output_pv_file = "phase_velocity/KYT013.vel"

fmax = 20.0
fmin_list = [12, 8, 4, 2.5]

input_pv_files = [input_pv_dir+s for s in file_list]

ns = len(fmin_list)
fr = fmin_list + [fmax] + fmin_list[0:-1]
frequency_range = list(zip(*[fr[i:i+ns] for i in range(0, 2*ns, ns)]))

print(input_pv_files)
print(frequency_range)

freq_list, pv_list = Array.io.read_pv_files(input_pv_files)
freq, vel = Array.analysis.connect_phase_velocity(freq_list, pv_list, frequency_range, plot_flag=True)
Array.io.output_pv_file(output_pv_file, freq, vel)
```

最大周波数

接続周波数のリスト

スクリプトの使い方 (Step2)

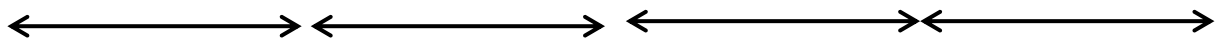
- スクリプトファイルの赤字の箇所を書き換えます
02_pv_connect.py

```
fmax = 20.0  
fmin_list = [12, 8, 4, 2.5]
```



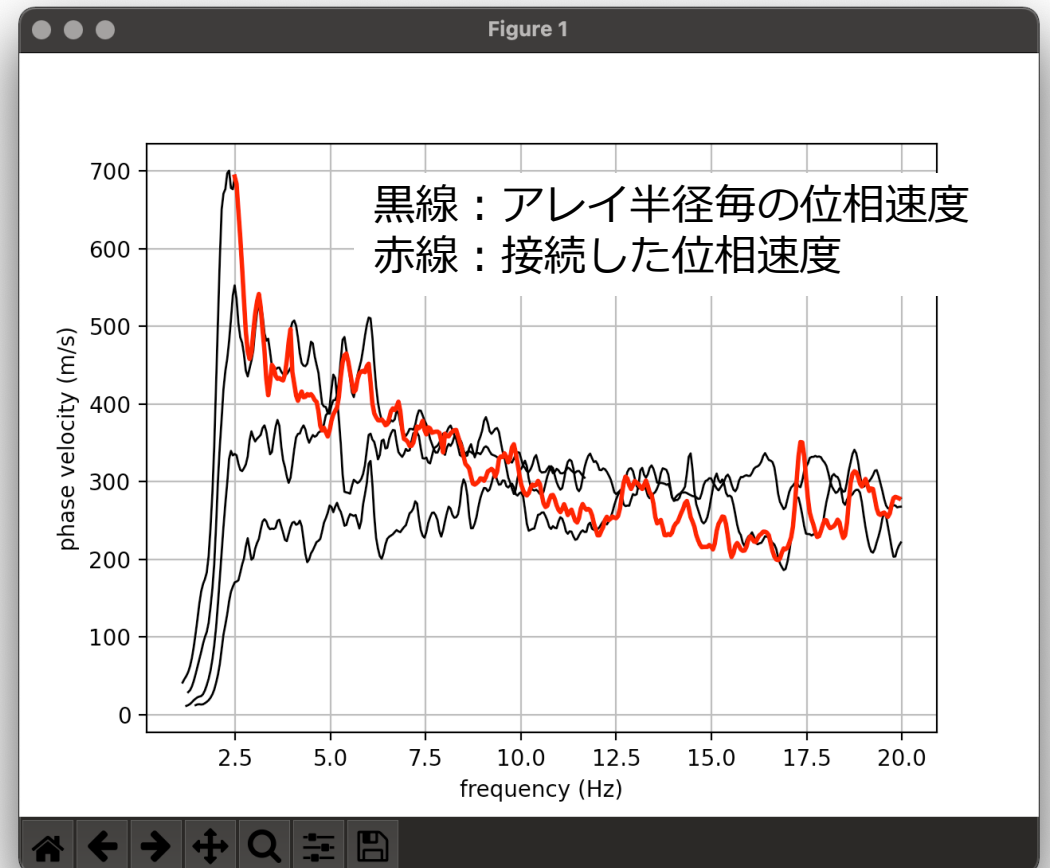
このような意味になります

10mアレイ	5mアレイ	2mアレイ	1mアレイ
2.5Hz - 4Hz	4Hz - 8Hz	8Hz - 12Hz	12Hz - 20Hz



高周波数側から記述することに注意

出力されるグラフを見ながら、
接続周波数を調整します



出力結果 (Step2)

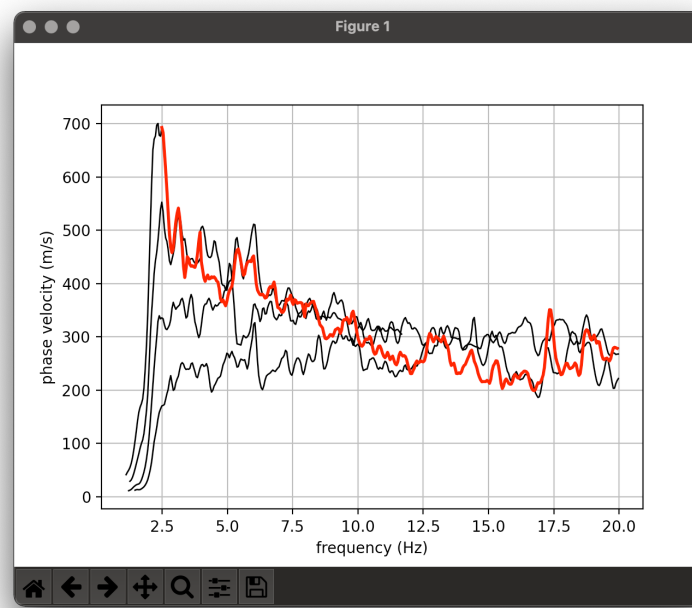
- 接続した位相速度がファイルに出力されます
サンプルファイル (phase_velocity/KYT013.vel)

```
2.490234375000000000e+00 6.929559528576182856e+02
2.539062500000000000e+00 6.834891616547772628e+02
2.587890625000000000e+00 6.494998423532302922e+02
2.636718750000000000e+00 6.140745412478711387e+02
2.685546875000000000e+00 5.758883544928710307e+02
2.734375000000000000e+00 5.327689685424421668e+02
2.783203125000000000e+00 4.932369854728897280e+02
2.832031250000000000e+00 4.679775667534746617e+02
2.880859375000000000e+00 4.580409435696012110e+02
2.929687500000000000e+00 4.664312177548768545e+02
...
```

1列目：周波数 (Hz)
2列目：位相速度 (m/s)

周波数一位相速度
(位相分散曲線)

赤線と同じもの



使い方 (Step3)

Step3

1次元速度モデルから位相速度を計算してフィッティングさせる

使うスクリプト

03_model_pv.py

モデルのサンプル

model/KYT013.dat

モデルファイルの準備 (Step3)

- 1次元速度モデルを設定します

サンプルファイル (model/KYT013.dat)

```
# number of layers          _____ 1
5
# vs[m/s], vp[m/s], rho[kg/m3], depth[m]
140 240 1590 2
275 1550 1773 10
350 2260 2031 25          _____ 2
510 2260 2200 95
1000 2600 2400
```

1 モデルの層数 (基盤含む)

2 層毎の物性
1列目: S波速度 (m/s)
2列目: P波速度 (m/s)
3列目: 密度 (kg/m3)
4列目: 下方境界面の深さ (m)
基盤の場合は空欄

スクリプトの使い方 (Step3)

- スクリプトファイルの赤字の箇所を書き換えます

03_model_pv.py

```
import numpy as np
import Array
```

```
observed_pv_file = "phase_velocity/KYT013.vel"
model_file = "model/KYT013.dat"
model_pv_file = "model/KYT013.vel"
```

```
freq_obs, vel_obs = Array.io.read_pv_file(observed_pv_file, fmax=16)
model = Array.io.read_model_file(model_file)
```

```
freq_sim, vel_sim, _ = Array.analysis.model_phase_velocity_py(model, fmax=16, print_flag=True, plot_flag=False)
```

```
Array.analysis.compare_phase_velocity(freq_obs, vel_obs, freq_sim, vel_sim)
Array.io.output_pv_file(model_pv_file, freq_sim, vel_sim)
```

位相速度のデータ

モデルファイル

モデルから計算された位相速度

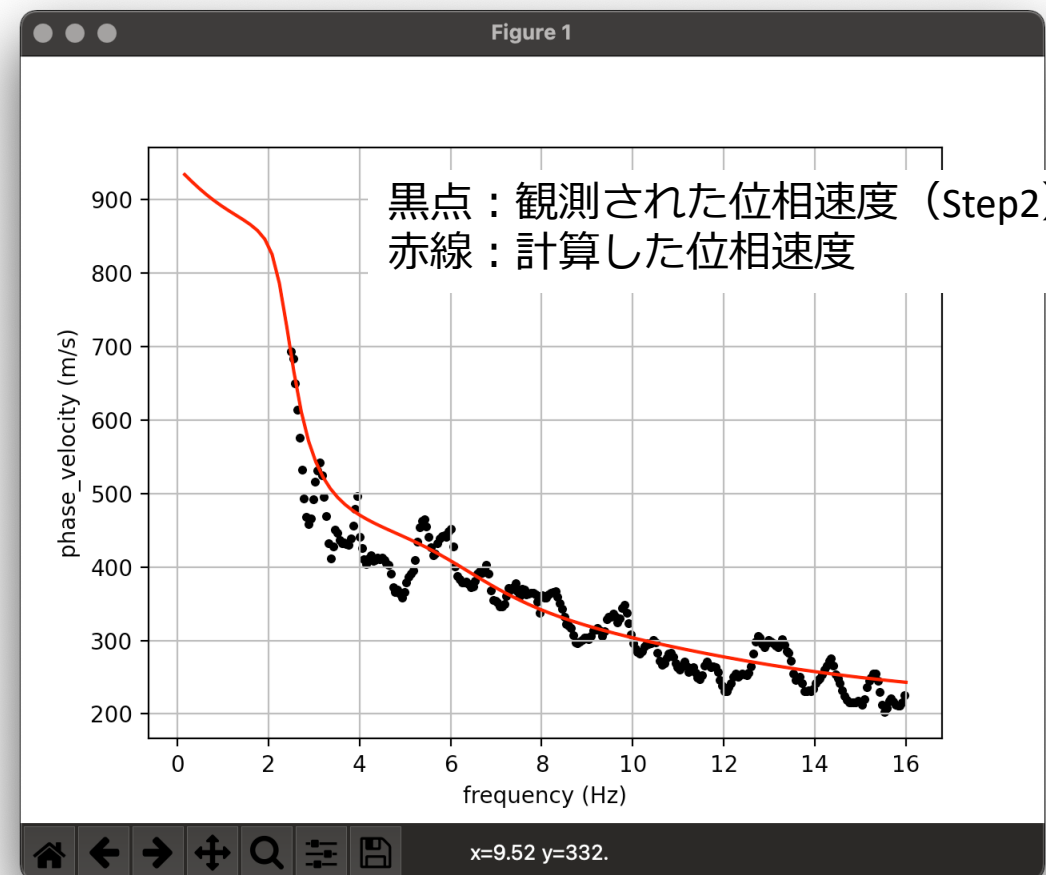
フィッティングさせる最大周波数

出力結果 (Step3)

- モデルから計算された位相速度が観測位相速度にあうように、モデルを修正します

赤線が黒点にあうように、
モデルを繰り返し修正する

```
# number of layers
5
# vs[m/s], vp[m/s], rho[kg/m3], depth[m]
140 240 1590 2
275 1550 1773 10
350 2260 2031 25
510 2260 2200 95
1000 2600 2400
```



出力結果 (Step3)

- モデルから計算された位相速度がファイルに出力されます
サンプルファイル (model/KYT013.vel)

```
1.600000000000000033e-01 9.343980000000000246e+02
3.200000000000000067e-01 9.247949999999999591e+02
4.799999999999999822e-01 9.151920000000000073e+02
6.400000000000000133e-01 9.0733500000000000364e+02
8.000000000000000444e-01 8.994779999999999518e+02
9.600000000000000755e-01 8.9249400000000000282e+02
1.119999999999999885e+00 8.855099999999999909e+02
1.280000000000000027e+00 8.7939900000000000009e+02
1.439999999999999947e+00 8.7328800000000000109e+02
1.599999999999999867e+00 8.663039999999999736e+02
...
```

1列目 : 周波数 (Hz)
2列目 : 位相速度 (m/s)

赤線と同じもの

