

A Python framework for microseismic monitoring network design

Denis Anikiev

Version 1.1.0

October 12, 2026

CONTENTS

1 Getting Started **2**
- 1.1 Installation . . . 2
  - 1.1.1 Requirements . . . 2
  - 1.1.2 Dependencies . . . 2
  - 1.1.3 Install using conda and pip . . . 2
  - 1.1.4 Install using script . . . 3
  - 1.1.5 Uninstall . . . 3
- 1.2 Package Design . . . 3
- 1.3 Input Files . . . 4
  - 1.3.1 Velocity model . . . 4
  - 1.3.2 Monitoring network geometry . . . 4
- 1.4 Quickstart . . . 6

2 Methodology **8**
- 2.1 Spectral conventions and source-model assumption . . . 8
- 2.2 Intrinsic attenuation . . . 9
- 2.3 Peak frequency . . . 9
- 2.4 Detection threshold . . . 10
- 2.5 Network detectability . . . 10
- 2.6 Free surface boundary condition . . . 10
- 2.7 Receiver projection and DAS directionality . . . 11
- 2.8 Radiation pattern . . . 11
- 2.9 Magnitude sensitivity . . . 12
  - 2.9.1 Homogeneous medium . . . 12
- 2.10 Derivation of the seismic moment in a homogeneous medium . . . 13
- 2.11 Derivation of effective radiation pattern magnitudes . . . 13
  - 2.11.1 P-wave average . . . 14
  - 2.11.2 S-wave average . . . 14
  - 2.11.3 SV- and SH-wave average . . . 15

3 Examples **17**
- 3.1 01.1 Magnitude sensitivity for homogeneous model: test case 1 . . . 17
  - 3.1.1 Load all necessary packages . . . 17
  - 3.1.2 Input . . . 18
  - 3.1.3 Prepare to compute magnitude sensitivity . . . 22
  - 3.1.4 Compute and plot magnitude sensitivity for single stations . . . 23
  - 3.1.5 Compute magnitude sensitivity for several stations using the whole grid . . . 44
  - 3.1.6 Get sensitivity slices by cutting from the grid . . . 44
  - 3.1.7 Plot magnitude sensitivity for several stations . . . 45
- 3.2 02.1 Magnitude sensitivity for homogeneous model: test case 2 . . . 54

3.2.1 Load all necessary packages . . . 55
3.2.2 Input . . . 55
3.2.3 Prepare to compute magnitude sensitivity . . . 59
3.2.4 Compute magnitude sensitivity . . . 60
3.2.5 Plot magnitude sensitivity . . . 62
3.3 03.1 Magnitude sensitivity for homogeneous model: test case 3 . . . 71
3.3.1 Load all necessary packages . . . 72
3.3.2 Input . . . 72
3.3.3 Prepare to compute magnitude sensitivity . . . 76
3.3.4 Retrieve minimum displacement amplitudes . . . 77
3.3.5 Compute magnitude sensitivity without directionality correction . . . 77
3.3.6 Plot magnitude sensitivity without directionality correction . . . 80
3.3.7 Compute magnitude sensitivity with directionality correction . . . 91
3.3.8 Plot magnitude sensitivity with directionality correction . . . 94
3.4 04.1 Magnitude sensitivity for homogeneous model: test case 4 . . . 105
3.4.1 Load all necessary packages . . . 106
3.4.2 Input . . . 106
3.4.3 Prepare to compute magnitude sensitivity . . . 113
3.4.4 Compute and plot magnitude sensitivity for independent geometries . . . 114
3.4.5 Compute and plot magnitude sensitivity for combined geometry . . . 143
3.5 05.1 Magnitude sensitivity for homogeneous model: test case 5 . . . 153
3.5.1 Load all necessary packages . . . 154
3.5.2 Input . . . 154
3.5.3 Prepare to compute magnitude sensitivity . . . 158
3.5.4 Compute magnitude sensitivity . . . 159
3.5.5 Plot magnitude sensitivity . . . 160
3.6 06.1 Magnitude sensitivity for homogeneous model: test case 6 . . . 172
3.6.1 Load all necessary packages . . . 173
3.6.2 Input . . . 173
3.6.3 Prepare to compute magnitude sensitivity . . . 177
3.6.4 Compute magnitude sensitivity . . . 178
3.6.5 Plot magnitude sensitivity . . . 179
3.7 07.1 Magnitude sensitivity for homogeneous model: 1-km DAS array benchmark . . . 191
3.7.1 Load all necessary packages . . . 192
3.7.2 Input . . . 192
3.7.3 Prepare to compute magnitude sensitivity . . . 194
3.7.4 Compute magnitude sensitivity . . . 195
3.7.5 Plot magnitude sensitivity . . . 196
3.8 08.1 Magnitude sensitivity for homogeneous model: test case 8 . . . 200
3.8.1 Load all necessary packages . . . 200
3.8.2 Input . . . 200
3.8.3 Prepare to compute magnitude sensitivity . . . 204
3.8.4 Compute magnitude sensitivity . . . 205
3.8.5 Plot magnitude sensitivity . . . 206
3.9 09.1 Magnitude sensitivity for homogeneous model: test case 9 . . . 218
3.9.1 Load all necessary packages . . . 219
3.9.2 Input . . . 219
3.9.3 Prepare to compute magnitude sensitivity . . . 225
3.9.4 Compute magnitude sensitivity . . . 226
3.9.5 Plot magnitude sensitivity . . . 227
3.10 10.1 Magnitude sensitivity for homogeneous model: test case 10 . . . 239
3.10.1 Load all necessary packages . . . 240
3.10.2 Input . . . 240
3.10.3 Prepare to compute magnitude sensitivity . . . 246

3.10.4 Compute magnitude sensitivity . . . 247
3.10.5 Plot magnitude sensitivity . . . 248

4 API Reference **261**
4.1 Modelling . . . 261
4.1.1 Core . . . 261
4.1.2 Homogeneous . . . 263
4.1.3 Geometry generators . . . 268
4.1.4 Input / output . . . 269
4.1.5 Classes . . . 274
4.1.6 Modelling utilities . . . 275
4.2 Visualization . . . 290
4.2.1 Geometry . . . 290
4.2.2 Sensitivity . . . 291
4.2.3 Visualization utilities . . . 293

5 Citing **295**
5.1 Earlier release . . . 296

6 Contributing **297**
6.1 Reporting a problem . . . 297
6.2 Development setup . . . 297
6.3 Branches and releases . . . 297
6.4 Tests . . . 297
6.5 Documentation . . . 298
6.6 Changelog . . . 298

7 Changelog **299**
7.1 v1.1.0 - 2026-10-12 . . . 299
7.1.1 Added . . . 299
7.1.2 Changed . . . 301
7.1.3 Removed . . . 302
7.1.4 Fixed . . . 302
7.2 v1.0.1 - 2026-09-10 . . . 302
7.2.1 Fixed . . . 302
7.2.2 Changed . . . 303
7.2.3 Removed . . . 303
7.2.4 Added . . . 303
7.3 v1.0.0 - 2025-04-01 . . . 303
7.3.1 Added . . . 303

8 Credits **305**
8.1 Author . . . 305
8.2 Acknowledgments . . . 305
8.3 License . . . 305

Bibliography **306**

Index **307**

PyNetDesign is an open-source Python framework for testing and designing microseismic monitoring networks in homogeneous velocity models.

PyNetDesign determines the **magnitude sensitivity** of a monitoring network, that is the minimum moment magnitude that a given set of receivers can theoretically detect. It works with conventional station networks and with distributed acoustic sensing (DAS) cables, including both at the same time, and accounts for the directional sensitivity that makes a DAS cable behave differently from a three-component station. Results are presented as horizontal or vertical slices through a 3D grid of potential source locations.

The modelling API also provides synthetic geometry generators for borehole, surface and dark-fibre layouts, so that trial networks can be built from code rather than from hand-written tables.

The core algorithms are vectorized with NumPy, input and output rely on Pandas, and visualization is based on Matplotlib with perceptually uniform colour maps from cmcrameri.

Current PyNetDesign version: 1.1.0

CHAPTER

ONE

GETTING STARTED

This chapter explains how to install PyNetDesign, how to describe a monitoring network and a velocity model, and how to run a first sensitivity calculation.

1.1 Installation

1.1.1 Requirements

PyNetDesign works with Python 3.9 to 3.12. Cloning the repository requires Git, and the recommended way to create the environment is the Conda package manager, for example through Miniforge.

1.1.2 Dependencies

The runtime dependencies are declared in `pyproject.toml` and installed by `pip`:

Package	Purpose
numpy	vectorized numerical core
pandas	geometry and velocity model tables, and their metadata
matplotlib	plotting
cmcrameri	perceptually uniform scientific colour maps

Two optional groups are available: `dev` adds `pytest` and an interactive Jupyter stack, and `docs` adds the Sphinx toolchain used to build this documentation.

The author is grateful to the developers of these packages.

1.1.3 Install using conda and pip

`environment.yml` pins only the interpreter; every package dependency comes from `pyproject.toml`. Clone the repository:

```
git clone https://github.com/danikiev/PyNetDesign.git
cd PyNetDesign
```

create and activate the environment:

```
conda env create -f environment.yml -n pnd
conda activate pnd
```

and install the package:

```
pip install -e .
```

To also install the test and documentation tools:

```
pip install -e ".[dev,docs]"
```

Tip

Using `mamba` instead of `conda` speeds up environment creation considerably.

1.1.4 Install using script

The helper scripts perform the steps above, creating the `pnd` environment and asking whether to add the development tools:

```
./install.sh            # Linux, macOS
install.bat             # Windows, from a Miniforge prompt
```

Pass `--dev` or `--no-dev` to `install.sh` to answer in advance.

1.1.5 Uninstall

Deactivate and remove the environment:

```
conda deactivate
conda remove -n pnd --all
```

`./uninstall.sh` on Linux and macOS, or `uninstall.bat` on Windows, finds every environment whose name starts with `pnd` and asks about each one.

1.2 Package Design

PyNetDesign is organised into two subpackages:

- `pynetdesign.modelling` contains the algorithms:
  - `core` provides the high-level drivers,
  - `homo` implements the homogeneous-medium engine,
  - `geometry` generates synthetic receiver layouts,
  - `io` reads and writes input files,
  - `utils` holds the wave-mode, amplitude and grid helpers.
- `pynetdesign.visualization` plots receiver geometry and sensitivity results.

Both subpackages are re-exported at the top level, so either style works:

```
import pynetdesign.modelling as pndmod
import pynetdesign.visualization as pndvis
```

1.3 Input Files

PyNetDesign reads two whitespace-delimited ASCII files: a velocity model and a monitoring network geometry.

1.3.1 Velocity model

The velocity model file has a fixed header and, for a homogeneous medium, exactly one row of values:

```
Depth(m) Vp(m/s) Vp/Vs Qp Qs density(kg/m^3) Ep(anisotropy) Dt(anisotropy) Gm(anisotropy)
0        2370    2     100 100 2500           0              0              0
```

The columns are the layer top depth, the P-wave velocity, the P-to-S velocity ratio, the P- and S-wave quality factors, the density, and three Thomsen anisotropy parameters.

Note

The Thomsen parameters are read and stored but not used in the calculation: the medium is treated as isotropic. Supplying more than one row raises an error, because PyNetDesign models a homogeneous medium only.

Read it with *read_velocity*:

```
velocity_df = pndmod.io.read_velocity('velocity_model.txt')
```

The S-wave velocity is derived as `Vp / VpVsRatio` and added as a `Vs` column.

1.3.2 Monitoring network geometry

The geometry file describes the receivers, one per row. A local-coordinate station network looks like this:

```
Name  Northing(m) Easting(m) Elevation(m) NoiseLevel(m/s)
ST1   0           0          0            5.0E-09
ST2   5000        0          0            5.0E-09
ST3   2500        4330       0            5.0E-09
```

and a DAS cable declares its gauge length in the header:

```
Northing(m) Easting(m) Elevation(m) NoiseLevel(1/s) GaugeLength=10(m)
0           0          0            1.95E-09
0           0          -1           1.95E-09
0           0          -2           1.95E-09
```

Required columns are a northing or latitude, an easting or longitude, an elevation and a noise level. Optional columns are:

Column	Meaning
`Name`	receiver label, used in plots
`Surface`	weight in [0, 1] marking receivers at the free surface; a fractional value blends between no amplification and full amplification
`Components`	`3C` or `Z`; a `Z` receiver records only the vertical component and is projected accordingly. Not allowed together with a gauge length

Geographic coordinates are given as `Latitude(WGS84)` and `Longitude(WGS84)` and are converted to a local Cartesian frame with *global2local*. Latitude and longitude must both be present, or neither.

Note

`Elevation` is positive upwards, while the internal `Z` coordinate is depth, positive downwards. A receiver written with `Elevation = -500` is stored as `Z = 500`. The writer reverses this, so a round trip through *save_geometry* is lossless.

Distance units

Coordinate columns and the gauge length accept a unit in parentheses and are converted to metres: `m`, `km`, `cm`, `mm`, `ft` and `in`. All coordinate columns of one file must use the same unit.

Amplitude units

The unit of `NoiseLevel` selects how the noise is converted to a displacement amplitude spectrum, as described in Section *Detection threshold*:

Unit in header	Interpreted as	Typical receiver
`m`	displacement	displacement sensor
`m/s`	particle velocity	geophone, seismometer
`m/s^2`	particle acceleration	accelerometer
`1/s`	strain rate	DAS
`strain`, `1`, `units`, `none`	strain	DAS

A gauge length is required for strain and strain-rate data, and its presence is what marks a geometry as DAS.

Read geometry files

```
geometry_df = pndmod.io.read_geometry('net_geometry.txt')
print(geometry_df.attrs)
```

The returned DataFrame has columns `Name`, `X`, `Y`, `Z` and `NoiseLevel`, plus any optional columns, and carries metadata in `attrs`: the file path, the original coordinate unit, the noise unit and type, the coordinate system, and the gauge length when present.

Generate and save geometry files

Trial layouts can be built from code with *generate_geometry*, which returns geometry text lines that *save_geometry* writes to disk:

```
lines = pndmod.geometry.generate_geometry(
    mode='borehole',         # 'borehole', 'surface' or 'darkfiber'
    cable_length=999.0,      # m of cable
    spacing=1.0,             # m between channels
    noise_level=1.95e-9,     # strain rate
    gauge_length=10.0,       # m
)
pndmod.io.save_geometry(lines, 'das_geometry.txt')
geometry_df = pndmod.io.read_geometry('das_geometry.txt')
```

Surface layouts accept a `surface_shape` of `line`, `zigzag`, `L`, `square` or `double_line`; boreholes accept an `azimuth` and `dip`, and can apply a depth-dependent noise profile. A dense cable can be thinned with *decimate_geometry*:

```
sparse_df = pndmod.io.decimate_geometry(geometry_df, 5)    # every 5th channel
```

Combine geometries

Several networks can be evaluated together, for instance a surface station array and a borehole DAS cable, with *combine_geometry*:

```
combined_df = pndmod.io.combine_geometry(stations_df, das_df)
```

Each contributing geometry keeps its own noise type and gauge length, and the per-receiver thresholds are concatenated before the network criterion is applied.

1.4 Quickstart

The following computes the minimum detectable moment magnitude on a vertical section through a borehole DAS array and plots it:

```
import numpy as np
import pynetdesign.modelling as pndmod
import pynetdesign.visualization as pndvis

# Input
geometry_df = pndmod.io.read_geometry('das_geometry.txt')
velocity_df = pndmod.io.read_velocity('velocity_model.txt')

# Grid of potential source locations
gx = np.arange(0.0, 2000.0 + 25.0, 25.0)
gy = np.array([0.0])
gz = np.arange(25.0, 1500.0 + 25.0, 25.0)
grid_points = pndmod.utils.generate_grid(x=gx, y=gy, z=gz)

# Detection criteria
params = pndmod.classes.DetectionParameters(
    min_SNR_p=2,          # SNR required on one channel
    min_SNR_s=2,
    min_stations_p=60,    # channels required for detection
    min_stations_s=60,
)

# Minimum detectable magnitude, requiring both P and S
mw_min = pndmod.core.get_mag_sensitivity(
    grid_coords=grid_points,
    geometry_df=geometry_df,
    velocity_df=velocity_df,
    params=params,
    wave_mode='PS',
)

# Vertical section at Y = 0
_ = pndvis.sensitivity.plot_sensitivity_slice(
```

(continues on next page)

(continued from previous page)

```
    sens_data=mw_min, xi=gx, zi=gz,
    geometry_df=geometry_df, clb_title='$M_w$')
```

`wave_mode` selects the phases: `'P'`, `'SV'`, `'SH'`, `'S'` for the composite S branch, or `'PS'` for joint P and S detection. See Section *Network detectability* for how the per-receiver thresholds are combined, and the *examples* for complete scripts.

CHAPTER

TWO

METHODOLOGY

This chapter presents the theory behind PyNetDesign. It describes how the minimum detectable moment magnitude of a microseismic event is evaluated for a network of receivers in a homogeneous, isotropic and attenuating medium, separately for each seismic phase and for each receiver type.

Note

PyNetDesign models a homogeneous medium. The ray path between a source and a receiver is therefore a straight line, there are no interfaces and no transmission losses, and geometrical spreading is simply proportional to distance.

2.1 Spectral conventions and source-model assumption

We work with frequency in Hz, f, and angular frequency $\omega = 2\pi f$.

Let $|U(f)|$ denote the amplitude spectrum magnitude of **particle displacement** (projected onto the considered phase polarization) recorded at the receiver. If the instrument measures particle velocity, the corresponding spectrum is

$$|V(f)| = 2\pi f\,|U(f)|, \tag{2.1}$$

which follows from $v(t) = \dot{u}(t)$.

For shear-dominated microseismic events we adopt the standard ω^{-2} source model [Aki and Richards, 2002]. The source displacement spectrum is

$$\Omega_D(f) = \begin{cases} C_0 M_0, & f < f_c, \\ C_0 M_0 \left(\dfrac{f_c}{f}\right)^2, & f \geq f_c, \end{cases}$$

where M_0 is seismic moment, f_c is corner frequency, and C_0 contains radiation-pattern and medium constants.

We define

$$\Omega_0 \equiv C_0 M_0,$$

which is the low-frequency (plateau) level of the far-field source displacement spectrum.

We assume a representative frequency f below the corner frequency ($f < f_c$, often $f = f_{\text{peak}}$, see Section *Peak frequency*), so the source spectrum is approximately flat, $\Omega_D(f) \approx \Omega_0$. In this regime, the source spectral function reduces to its low-frequency limit and can be approximated as unity.

2.2 Intrinsic attenuation

The attenuation operator t^* (known as global absorption factor [Červený, 2001]) is the path integral along the ray:

$$t^* = \int_R \frac{dr}{v(r)\,Q(r)}, \tag{2.2}$$

For a homogeneous medium, where v and Q are constant along the straight ray of length r, Equation (2.2) reduces to

$$t^* = \frac{r}{vQ}. \tag{2.3}$$

2.3 Peak frequency

The peak frequency of an observed signal is the frequency corresponding to the maximum amplitude in Fourier spectra of the particle velocity. As shown by Eisner *et al.* [2013], the peak frequency can be determined for geophone recordings (i.e., particle velocity) from the global absorption factor t^* only. The particle velocity is convenient mainly because most of the observations are carried out on instruments recording particle velocity (e.g., geophones or even broadband seismometers).

In Section *Spectral conventions and source-model assumption* we have shown that the Fourier spectrum of particle displacement is flat at low frequencies with nonzero permanent displacement at the source [Aki and Richards, 2002]. At higher frequencies, the source spectrum is limited by the rise and rupture times of an earthquake fault rupture. These limitations result in a sharp slope change of spectral power density of the particle displacement spectrum. This abrupt change in slope occurs at a corner frequency f_c. Hence, the peak of this positive function must occur for frequencies at or below the corner frequency [Eisner *et al.*, 2013].

The peak frequency f_{peak} is the frequency of the maximum amplitude in the particle velocity spectra. Propagation attenuation modifies the observed spectrum through the global absorption factor t^* (e.g., Červený [2001], Eisner *et al.* [2013]):

$$|V(f)| \propto f\, e^{-\pi f t^*}.$$

Maximization of this expression for $f < f_c$ yields

$$f_{\text{peak}} = \frac{1}{\pi t^*}, \tag{2.4}$$

where t^* is evaluated according to Equation (2.2).

Note

Equation (2.4) is only valid for $f < f_c$, i.e., when the source spectrum is approximately flat. If not, the maximum occurs at the boundary $f_{\text{peak}} = f_c$.

Similarly, the peak amplitude for particle acceleration will occur at a frequency twice bigger than those for particle velocity, i.e., at $f_{\text{peak}} = 2/(\pi t^*)$.

Taking into account Equation (2.3), in homogeneous medium the peak amplitude for particle velocity is

$$f_{\text{peak}} = \frac{vQ}{\pi r}, \tag{2.5}$$

where v is the body wave velocity (P- or S-wave), Q is the attenuation factor (P- or S-wave), and r is the distance from a source to a receiver.

2.4 Detection threshold

An event is considered detected on a receiver when the amplitude of the arriving phase exceeds the local noise level by a required factor. For receiver i the minimum detectable displacement amplitude spectrum is

$$|U_{\mathrm{min},i}(f)| = D_{\mathrm{SNR}}\, A_{N,i}\, S(f), \tag{2.6}$$

where $A_{N,i}$ is the measured noise level of that receiver, D_{SNR} is the signal-to-noise ratio required for detection, and $S(f)$ converts the recorded quantity to a displacement amplitude spectrum.

Following Equation (2.1), the conversion coefficient is

Recorded quantity	$S(f)$	Comment
displacement	1	already a displacement
particle velocity	$(2\pi f)^{-1}$	from $\lvert V(f)\rvert = 2\pi f\,\lvert U(f)\rvert$
particle acceleration	$(2\pi f)^{-2}$	two integrations
strain	L_g	strain averaged over the gauge length L_g
strain rate	$L_g/(2\pi f)$	as strain, plus one integration in time

For DAS, the recorded quantity is the strain or strain rate along the cable, averaged over the gauge length L_g. Multiplying by L_g converts it to an equivalent displacement, which assumes the strain to be constant over the gauge length.

The threshold is then corrected for the receiver response of Section *Receiver projection and DAS directionality*, giving the polarization-aligned threshold $\widetilde{U}^{\phi}_{\mathrm{min},i} = C_\phi\,|U_{\mathrm{min},i}(f)|$ for phase ϕ, and divided by the free-surface factor of Section *Free surface boundary condition* for receivers at the surface.

2.5 Network detectability

Inserting the threshold into Equation (2.7) and converting to a magnitude yields a per-receiver threshold magnitude $M_w^{\mathrm{min}}(\mathbf{x}, i)$ for every grid point $\mathbf{x}$ and every receiver i.

An event is only useful if it is seen on several receivers. Sorting the per-receiver values at a grid point in ascending order, the network threshold for a phase is the N-th smallest value, where N is the minimum number of receivers required for detection. For joint P- and S-wave detection, the stricter of the two phase thresholds applies,

$$M_w^{\mathrm{min}} = \max\!\left(M_w^{\mathrm{min},P},\; M_w^{\mathrm{min},S}\right).$$

For the composite `S` mode, the SV and SH branches are evaluated separately for each source-receiver pair and the more detectable branch, that is the smaller magnitude, is retained. A phase whose projection vanishes is excluded rather than treated as infinitely detectable.

2.6 Free surface boundary condition

At the free surface boundary, incoming body waves are reflected, resulting in a complex interference pattern that modifies the recorded amplitude. For the purposes of detectability, we approximate the recorded displacement amplitude to be twice that of the incident wave. This approximation corresponds exactly to the free-surface amplification factor for normal or near-normal incidence, as derived from the free-surface reflection equations (Aki and Richards [2002], Section 5.2.2).

2.7 Receiver projection and DAS directionality

PyNetDesign treats ordinary geophones as three-component receivers by default. For these receivers the recorded scalar amplitude is assumed to be aligned with the arriving phase polarization, so no projection loss is applied. If a station geometry row has `Components=Z`, the arriving displacement is projected onto the vertical axis. For DAS receivers, PyNetDesign applies the same phase-dependent projection logic onto the local fiber tangent $\boldsymbol{\tau}$.

In a homogeneous medium the ray is a straight line, so the phase polarization is computed from the straight receiver-to-grid direction. The amplitude correction for phase ϕ and receiver direction $\mathbf{d}$ is

$$C_\phi = \frac{1}{|\mathbf{d} \cdot \mathbf{e}_\phi|},$$

where $\mathbf{d}$ is $\hat{\mathbf{z}}$ for vertical-component geophones and $\boldsymbol{\tau}$ for DAS. $\mathbf{e}_P$ is the longitudinal P polarization, $\mathbf{e}_{SV}$ is the transverse polarization in the local vertical ray plane, and $\mathbf{e}_{SH}$ is perpendicular to that plane. The corrected minimum recorded amplitude is multiplied by C_ϕ. For composite `S` mode, SV and SH are corrected separately and the more detectable branch is retained.

If the projection is zero, that phase is not observable on the selected receiver component for that source-station pair. For example, SH is horizontal in the current isotropic convention and is therefore not recorded by a vertical-component geophone.

For nearly vertical rays, the explicit SV/SH azimuthal basis is convention-dependent because the horizontal ray azimuth is poorly defined. PyNetDesign uses a stable horizontal fallback basis in this case; the composite `S` mode is the practical robust choice when only S-wave detectability is required.

2.8 Radiation pattern

In the far field, body-wave amplitudes from a seismic moment tensor source depend on the take-off direction and on the wave polarization. Let $\mathbf{n}$ be the unit vector in the P-wave take-off direction at the source (pointing from source to receiver), and let $\mathbf{e}$ be the unit polarization vector of the considered phase at the source.

Following the quantitative formulation of Aki and Richards [2002] (Section 4.5), for a moment tensor $\mathbf{M}$, the dimensionless scalar radiation pattern factor for the phase with polarization $\mathbf{e}$ is

$$R(\mathbf{n}, \mathbf{e}) = \frac{\mathbf{e}^T \mathbf{M}\, \mathbf{n}}{M_0},$$

where M_0 is the scalar seismic moment (used only for normalization of the dimensionless factor).

For P waves, the polarization is longitudinal ($\mathbf{e} = \mathbf{n}$), leading to the familiar radiation pattern factor (Aki and Richards [2002], Eq. 4.89):

$$R_P = \frac{\mathbf{n}^T \mathbf{M}\, \mathbf{n}}{M_0}.$$

For S waves, there are two orthogonal transverse polarizations. Denoting by $\mathbf{s}_{SV}$ the unit SV polarization (in the ray plane) and by $\mathbf{s}_{SH}$ the unit SH polarization (perpendicular to the ray plane), the corresponding radiation factors are (Aki and Richards [2002], Eqs. 4.90 and 4.91):

$$R_{SV} = \frac{\mathbf{s}_{SV}^T \mathbf{M}\, \mathbf{n}}{M_0}, \qquad R_{SH} = \frac{\mathbf{s}_{SH}^T \mathbf{M}\, \mathbf{n}}{M_0}.$$

In this documentation and in the magnitude-sensitivity formulas, $|R|$ denotes the **magnitude** of the appropriate factor ($|R_P|$ for P, $|R_{SV}|$ for SV, and $|R_{SH}|$ for SH). For `wave_mode="S"`, PyNetDesign evaluates SV and SH separately and keeps the more detectable branch for each source-station pair.

For microseismic network design, the source mechanism and take-off directions are often unknown. In that case, it is common to use an *effective* radiation-pattern magnitude (i.e., a somehow averaged value of $|R|$ over a suitable ensemble of mechanisms and directions). Hallo and Eisner [2013] following Boore and Boatwright [1984] suggest to use representative RMS values for DC seismic sources $R_P^{\text{eff}} = \sqrt{\langle R_P^2 \rangle} \approx 0.52$ and $R_S^{\text{eff}} = \sqrt{\langle R_S^2 \rangle} \approx 0.63$. PyNetDesign uses fixed RMS constants $R_P^{\text{eff}} = \sqrt{4/15} \approx 0.516$, $R_{SV}^{\text{eff}} = \sqrt{7/30} \approx 0.483$ for SV and $R_{SH}^{\text{eff}} = \sqrt{1/6} \approx 0.408$ for SH. See Section *Derivation of effective radiation pattern magnitudes* for more information.

2.9 Magnitude sensitivity

The magnitude sensitivity module is designed to estimate the magnitudes of microseismic events that can be theoretically detected by the network of receivers. This sensitivity (or detectability) varies in space and hence it is presented in the form of horizontal or vertical slices in the 3D space. PyNetDesign uses attenuation of the dominant signal (based on the quality factor of the medium Q), and signal-to-noise ratio (SNR) necessary for seismic event detection. Results depend on a model of attenuation, noise, wave velocity, and detection method. The resulting sensitivity can show the spatial distribution of the minimal technically detectable magnitude, or, in other words, the spatial distribution of the magnitude of detection catalogue completeness. The moment magnitude of an event, according to Kanamori [1977], is given by logarithm of the scalar value of the seismic moment:

$$M_w = \frac{2}{3} \left(\log_{10}(M_0) - 9.1 \right),$$

where M_0 is the seismic moment.

PyNetDesign computes minimal moment magnitude of an event, which will theoretically produce amplitudes on receivers necessary for the event detection. Amplitude levels for the event detection are dependent on local noise levels and signal-to-noise ratio necessary for particular detection method.

2.9.1 Homogeneous medium

In a homogeneous isotropic medium, the scalar seismic moment M_0 can be related to the magnitude of the displacement Fourier spectrum $|U(f)|$ (projected onto the arriving polarization of the considered phase P or S) at frequency f as

$$M_0 = \frac{4\pi\rho v^3 r}{|R|} \, |U(f)| \, \exp\big(\pi f \, t^*\big), \tag{2.7}$$

where ρ is density, v is the body-wave velocity ($v = \alpha$ for P waves, $v = \beta$ for S waves), and r is the source-receiver distance. The factor $|R|$ is the dimensionless radiation-pattern magnitude for the considered phase, consistent with the moment-tensor radiation convention of Aki and Richards [2002] (see Section *Radiation pattern*).

For derivation of formula (2.7), see Section *Derivation of the seismic moment in a homogeneous medium*.

The exponential term $\exp(\pi f t^*)$ corrects for intrinsic attenuation using the attenuation operator t^* (see Section *Intrinsic attenuation*).

Because P-wave velocity is typically higher than S-wave velocity, for the same propagation path and Q the term $t^* = r/(vQ)$ is smaller for P waves, so the attenuation correction $\exp(\pi f t^*)$ is closer to unity. In practice, detectability is also controlled by radiation pattern and noise characteristics; network-design sensitivity is therefore often evaluated separately for P and S phases [Hallo and Eisner, 2013].

2.10 Derivation of the seismic moment in a homogeneous medium

Here we derive Equation (2.7) from the far-field radiation of a point moment-tensor source.

In an unbounded homogeneous isotropic medium, the far-field displacement produced by a point source with moment time function $M_0(t)$ is, for the phase with radiation-pattern factor R and velocity v (Aki and Richards [2002], Section 4.3),

$$u(t) = \frac{R}{4\pi\rho v^3 r}\,\dot{M}_0\!\left(t - \frac{r}{v}\right), \tag{2.8}$$

where the $1/r$ factor is the geometrical spreading of a spherical wave in a homogeneous medium and the dot denotes the time derivative.

Taking the Fourier transform and using the fact that the moment rate spectrum tends to the scalar seismic moment at low frequency, $|\dot{M}_0(f)| \to M_0$ for $f \ll f_c$, the low-frequency plateau of the displacement amplitude spectrum is

$$\Omega_0 = \frac{M_0\,|R|}{4\pi\rho v^3 r}, \tag{2.9}$$

consistent with the source-model assumption of Section *Spectral conventions and source-model assumption*.

Intrinsic attenuation multiplies the spectrum by the dissipation filter of Section *Intrinsic attenuation*, so the spectrum actually observed at the receiver is

$$|U(f)| = \frac{M_0\,|R|}{4\pi\rho v^3 r}\,\exp\!\big(-\pi f\,t^*\big), \qquad t^* = \frac{r}{vQ}. \tag{2.10}$$

Solving Equation (2.10) for the seismic moment gives

$$M_0 = \frac{4\pi\rho v^3 r}{|R|}\,|U(f)|\,\exp\!\left(\pi f\,\frac{r}{vQ}\right),$$

which is Equation (2.7).

Note

The exponential appears with a **positive** exponent in the moment formula: the amplitude observed at the receiver has already been reduced by attenuation, so recovering the source moment means undoing that reduction.

Note also that $|U(f)|$ is an amplitude **spectrum**. The conversion of a recorded noise level to $|U(f)|$ is carried out once, by the coefficient $S(f)$ of Section *Detection threshold*, and must not be applied a second time here.

2.11 Derivation of effective radiation pattern magnitudes

For microseismic network design with unknown focal mechanisms, we utilize effective averaged radiation factors. While literature often name these as average magnitudes, they are mathematically evaluated as the **Root Mean Square (RMS)** averages over the entire focal sphere to preserve wave energy [Boore and Boatwright, 1984].

We assume a canonical double-couple (DC) source. In a coordinate system where the fault plane normal is aligned with the x-axis and the slip direction is aligned with the y-axis, the seismic moment tensor $\mathbf{M}$ is:

$$\mathbf{M} = M_0 \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

Let γ be the unit take-off vector parameterized by polar angle θ and azimuthal angle ϕ:

$$\gamma = \begin{pmatrix} \sin\theta\cos\phi \\ \sin\theta\sin\phi \\ \cos\theta \end{pmatrix}.$$

2.11.1 P-wave average

The P-wave radiation pattern factor is the projection of the equivalent force $\mathbf{M}\gamma$ onto the longitudinal polarization γ:

$$R_P = \frac{\gamma^T \mathbf{M}\gamma}{M_0} = 2\sin^2\theta\sin\phi\cos\phi = \sin^2\theta\sin 2\phi.$$

To find the RMS average, we integrate the square of R_P over the unit sphere (where the averaging operator is $\frac{1}{4\pi}\int_0^{2\pi}\int_0^{\pi}\ldots\sin\theta\,d\theta\,d\phi$):

$$\langle R_P^2\rangle = \frac{1}{4\pi}\int_0^{2\pi}\int_0^{\pi}\left(\sin^2\theta\sin 2\phi\right)^2\sin\theta\,d\theta\,d\phi.$$

Separating the integrals:

$$\langle R_P^2\rangle = \frac{1}{4\pi}\left(\int_0^{\pi}\sin^5\theta\,d\theta\right)\left(\int_0^{2\pi}\sin^2 2\phi\,d\phi\right).$$

Evaluating the integrals yields $16/15$ for the polar part and π for the azimuthal part:

$$\langle R_P^2\rangle = \frac{1}{4\pi}\left(\frac{16}{15}\right)(\pi) = \frac{4}{15}.$$

Taking the square root provides the effective P-wave magnitude:

$$R_P^{\text{rms}} = \sqrt{\frac{4}{15}} \approx 0.516 \approx 0.52.$$

2.11.2 S-wave average

For S-waves, the total radiation pattern squared, R_S^2, is the squared magnitude of the transverse component of the equivalent force. This is the total equivalent force squared minus the longitudinal (P-wave) component squared:

$$R_S^2 = \frac{|\mathbf{M}\gamma|^2}{M_0^2} - R_P^2.$$

First, we compute the squared magnitude of the normalized equivalent force:

$$\frac{|\mathbf{M}\gamma|^2}{M_0^2} = (\sin\theta\sin\phi)^2 + (\sin\theta\cos\phi)^2 + 0^2 = \sin^2\theta.$$

Averaging this over the focal sphere:

$$\langle\frac{|\mathbf{M}\gamma|^2}{M_0^2}\rangle = \frac{1}{4\pi}\int_0^{2\pi}\int_0^{\pi}\sin^2\theta\sin\theta\,d\theta\,d\phi = \frac{1}{4\pi}\left(\frac{4}{3}\right)(2\pi) = \frac{2}{3}.$$

Subtracting the P-wave averaged from the total force average yields the total S-wave average:

$$\langle R_S^2\rangle = \frac{2}{3} - \langle R_P^2\rangle = \frac{10}{15} - \frac{4}{15} = \frac{6}{15} = \frac{2}{5}.$$

Taking the square root provides the effective total S-wave magnitude:

$$R_S^{\text{rms}} = \sqrt{\frac{2}{5}} \approx 0.632 \approx 0.63.$$

2.11.3 SV- and SH-wave average

The total S-wave coefficient derived above describes the magnitude of the full transverse radiation vector. It does **not** imply equal partition between the SV and SH components.

To obtain component-wise averages, we now switch to the standard fault-based angular convention of Aki and Richards [2002]. To avoid confusion with the spherical angles used above in the tensor-based derivation, we denote these fault-based angles by Θ and Φ.

In this convention, the standard double-couple far-field radiation-pattern expressions for P, SV, and SH waves are:

$$R_P(\Theta,\Phi) = \sin 2\Theta \cos\Phi,$$

$$R_{SV}(\Theta,\Phi) = \cos 2\Theta \cos\Phi,$$

$$R_{SH}(\Theta,\Phi) = -\cos\Theta \sin\Phi,$$

where Θ is the take-off angle and Φ is the azimuth in the fault-based coordinate system.

The effective component-wise coefficients are defined by RMS averaging over the focal sphere:

$$\langle R_{SV}^2 \rangle = \frac{1}{4\pi} \int_0^{2\pi} \int_0^{\pi} \cos^2 2\Theta \cos^2\Phi \sin\Theta \, d\Theta \, d\Phi,$$

$$\langle R_{SH}^2 \rangle = \frac{1}{4\pi} \int_0^{2\pi} \int_0^{\pi} \cos^2\Theta \sin^2\Phi \sin\Theta \, d\Theta \, d\Phi.$$

Using

$$\int_0^{2\pi} \cos^2\Phi \, d\Phi = \pi, \qquad \int_0^{2\pi} \sin^2\Phi \, d\Phi = \pi,$$

$$\int_0^{\pi} \cos^2 2\Theta \sin\Theta \, d\Theta = \frac{14}{15}, \qquad \int_0^{\pi} \cos^2\Theta \sin\Theta \, d\Theta = \frac{2}{3},$$

we obtain

$$\langle R_{SV}^2 \rangle = \frac{1}{4\pi} \, \pi \, \frac{14}{15} = \frac{7}{30},$$

$$\langle R_{SH}^2 \rangle = \frac{1}{4\pi} \, \pi \, \frac{2}{3} = \frac{1}{6}.$$

Therefore, the RMS effective coefficients are

$$R_{SV}^{\text{rms}} = \sqrt{\langle R_{SV}^2 \rangle} = \sqrt{\frac{7}{30}} \approx 0.483,$$

$$R_{SH}^{\text{rms}} = \sqrt{\langle R_{SH}^2 \rangle} = \sqrt{\frac{1}{6}} \approx 0.408.$$

As expected, the total S-wave mean-square coefficient is the sum of the two orthogonal transverse components:

$$\langle R_S^2 \rangle = \langle R_{SV}^2 \rangle + \langle R_{SH}^2 \rangle = \frac{7}{30} + \frac{1}{6} = \frac{2}{5}.$$

Hence

$$R_S^{\text{rms}} = \sqrt{\langle R_S^2 \rangle} = \sqrt{\frac{2}{5}} \approx 0.63.$$

Note

Therefore, the commonly used value 0.63 corresponds to the **combined S-wave amplitude**, not to SV alone and not to SH alone. PyNetDesign uses fixed RMS constants $R_P^{\mathrm{rms}} = \sqrt{4/15}$, $R_{SV}^{\mathrm{rms}} = \sqrt{7/30}$ for explicit SV, $R_{SH}^{\mathrm{rms}} = \sqrt{1/6}$ for explicit SH, and evaluates both S branches for composite `S` detectability.

CHAPTER

THREE

EXAMPLES

This chapter showcases the capabilities of PyNetDesign through practical, hands-on scenarios. Whether you are laying out a monitoring network, evaluating its magnitude sensitivity or visualizing the result, these examples give step-by-step guidance. Designed to be both educational and practical, the chapter bridges the gap between the theory of the *Methodology* chapter and real-world application.

The examples cover ten test monitoring network geometries in a homogeneous velocity model:

1. Simple station geometry with 3 surface stations in the form of a triangle, from Hallo [2012]
2. Simple station geometry with 4 surface stations in the form of a triangle with 1 station in the center
3. Simple DAS geometry with 1 vertical borehole
4. Combined geometry with 4 surface stations and DAS in a vertical borehole
5. Geometry with a DAS cable in an inclined borehole
6. Geometry with a DAS cable in a twice inclined borehole
7. A 1-km-long DAS cable in a vertical borehole with 1 m channel spacing and constant noise, the homogeneous benchmark of a downhole DAS monitoring study
8. Geometry with a DAS cable in a deep vertical borehole and noise decreasing with depth
9. Geometry with a single station above a DAS cable in a deep vertical borehole and noise constant along the borehole
10. Geometry with a single station above a DAS cable in a deep vertical borehole and noise decreasing with depth

3.1 01.1 Magnitude sensitivity for homogeneous model: test case 1

This is an example of computing magnitude sensitivity for a homogeneous model.

Test case 1 (from Hallo [2012]):

- Geometry: simple station geometry with 3 surface stations in the form of a triangle
- Velocity model: homogeneous, Vp = 4300 m/s, Vp/Vs = 1.73, Qp = 100, Qs = 50, rho = 2300 kg/m^3
- Processing: magnitude sensitivity computation for individual stations (for grid slices only) and for all stations (for all grid points, cutting slices after).

3.1.1 Load all necessary packages

```
import pynetdesign.visualization as pndvis
import pynetdesign.modelling as pndmod
import numpy as np
from time import time
```

(continues on next page)

(continued from previous page)

```
# sphinx_gallery_thumbnail_number = -2
```

3.1.2 Input

Read geometry

```
geometry_file_path = '../data/homo/test_01/TMPL_net_geometryA.txt'
geometry_df = pndmod.io.read_geometry(geometry_file_path)

print(geometry_df)
print(geometry_df.attrs)
```

```
  Name       Y       X    Z    NoiseLevel
0  ST1     0.0     0.0 -0.0  5.000000e-09
1  ST2     0.0  5000.0 -0.0  5.000000e-09
2  ST3  4330.0  2500.0 -0.0  5.000000e-09
{'file_path': '../data/homo/test_01/TMPL_net_geometryA.txt', 'original_coord_units':
↪'m', 'coordinate_system': 'local', 'noise_type': 'velocity', 'noise_unit': 'm/s', '_
↪is_combined': False}
```

Read velocity model

```
velocity_file_path = '../data/homo/test_01/TMPL_velocity_model.txt'
velocity_df = pndmod.io.read_velocity(velocity_file_path)

print(velocity_df)
print(velocity_df.attrs)
```

```
   Depth    Vp  VpVsRatio   Qp  Qs   Rho   Ep   Dt   Gm
0  40000  4300       1.73  100  50  2300  0.0  0.0  0.0
{'model_type': 'homo', 'model_anisotropy': 'isotropic', 'parameter_descriptions': {
↪'Depth': 'Depth', 'Vp': 'P-wave velocity', 'Vs': 'S-wave velocity', 'VpVsRatio':
↪'Vp/Vs', 'Qp': 'P-wave quality factor', 'Qs': 'S-wave quality factor', 'Rho':
↪'density', 'Ep': 'Thomsen epsilon parameter', 'Dt': 'Thomsen delta parameter', 'Gm
↪': 'Thomsen gamma parameter'}, 'parameter_units': {'Depth': 'm', 'Vp': 'm/s', 'Vs':
↪'m/s', 'VpVsRatio': '', 'Qp': '', 'Qs': '', 'Rho': 'kg/m^3', 'Ep': '', 'Dt': '', 'Gm
↪': ''}}
```

Set imaging grid parameters

```
# Imaging grid parameters
gx = np.arange(-3000, 8000 + 250, 250)
gy = np.arange(-3000, 8000 + 250, 250)
gz = np.arange(250, 10000 + 250, 250)
#gz = np.arange(250, 5000 + 250, 250)

# Generate grid points
grid_points = pndmod.utils.generate_grid(x=gx,y=gy,z=gz)
```

(continues on next page)

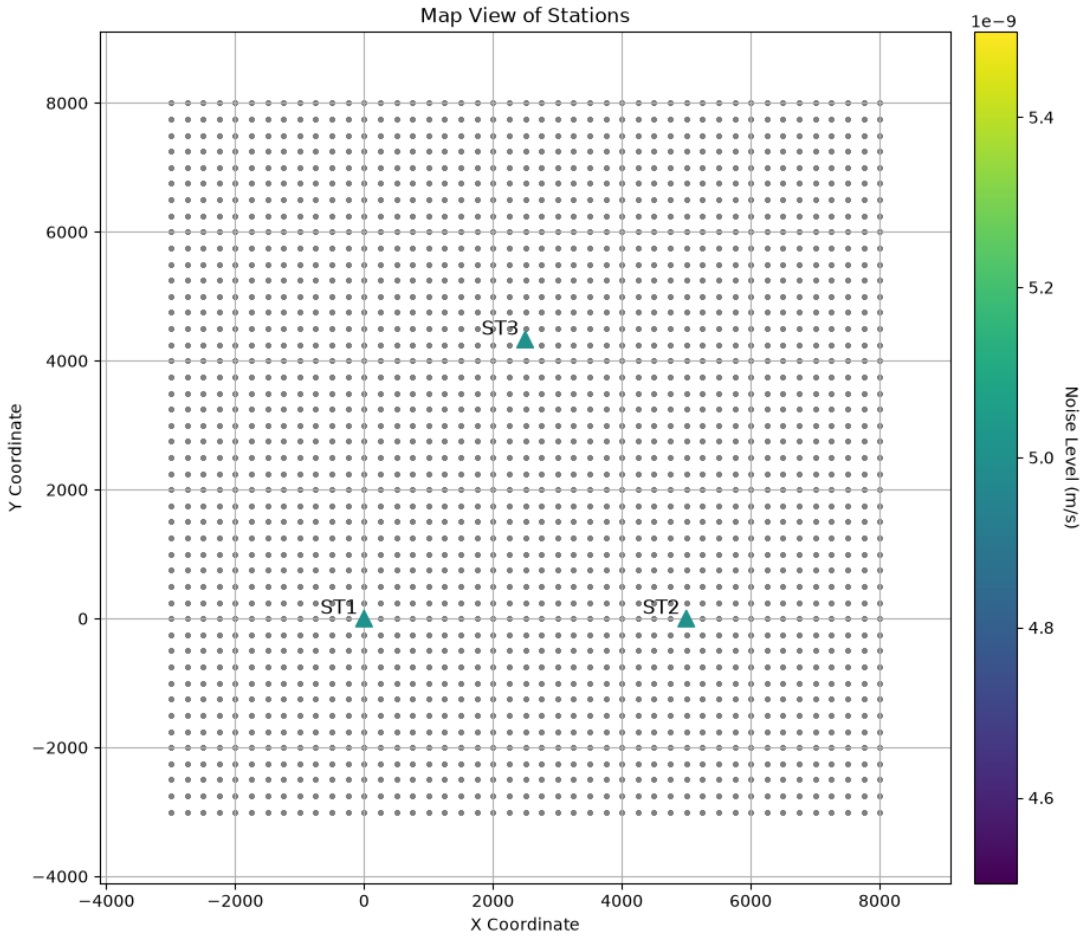
(continued from previous page)

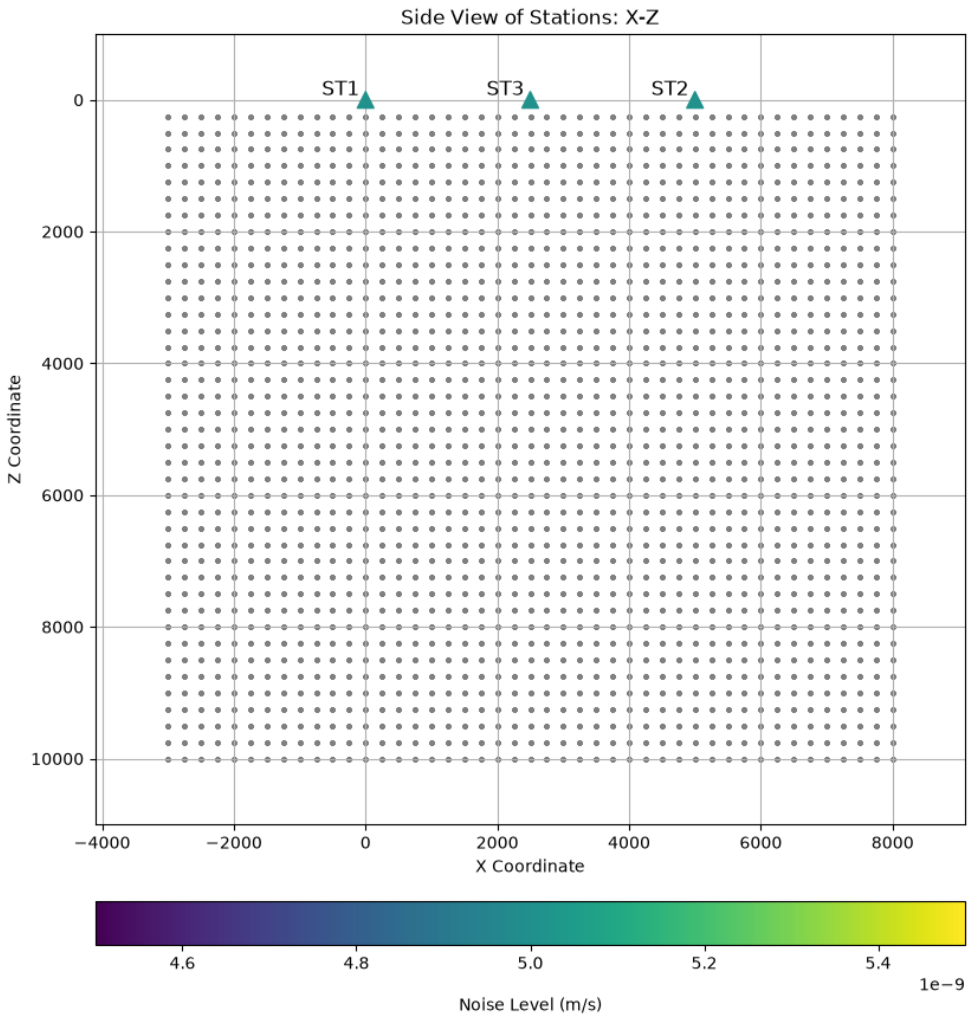
```
print("Number of grid points:",grid_points.shape[0])
print("grid_points.shape:",grid_points.shape)
```

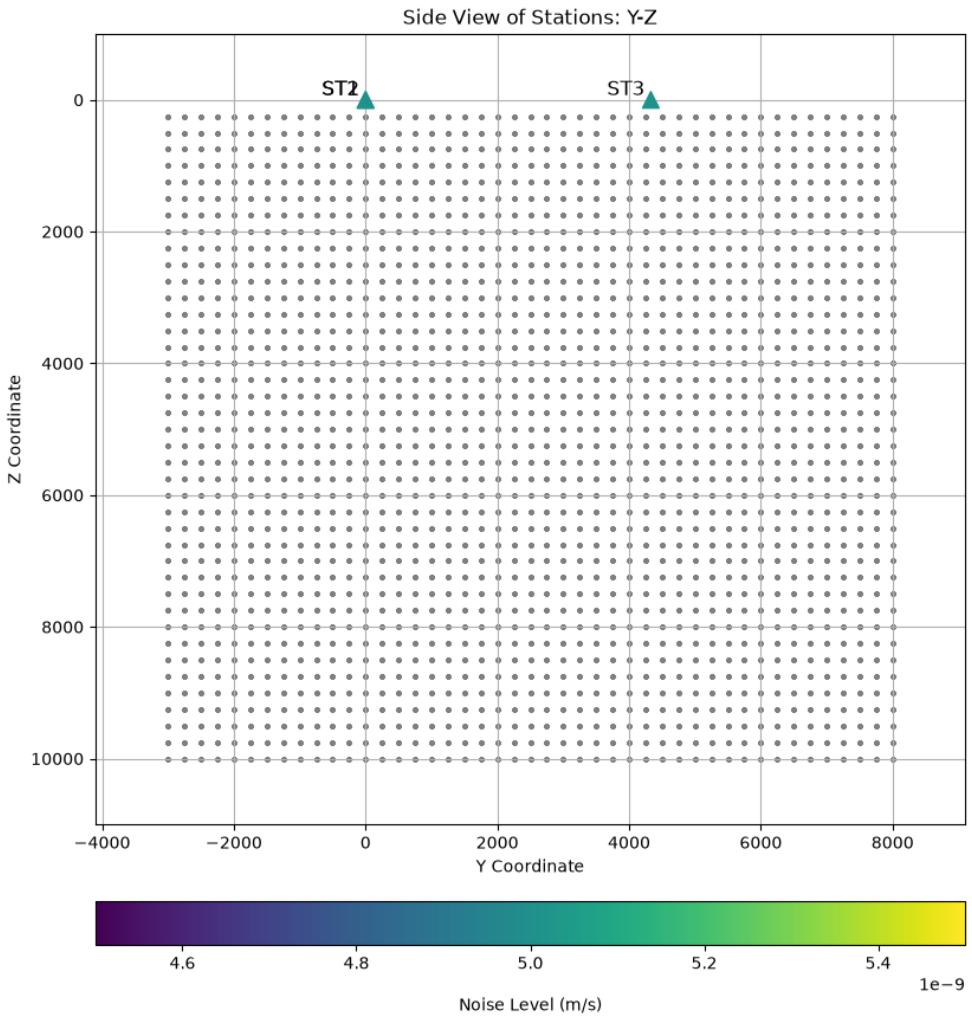
```
Number of grid points: 81000
grid_points.shape: (81000, 3)
```

Plot geometry and imaging grid

```
_ = pndvis.geometry.plot_stations_view(df=geometry_df,
                                       grid_points=grid_points,
                                       grid_marker_size = 1)
_ = pndvis.geometry.plot_stations_view(df=geometry_df,
                                       grid_points=grid_points,
                                       grid_marker_size = 1,
                                       view = 'xz')
_ = pndvis.geometry.plot_stations_view(df=geometry_df,
                                       grid_points=grid_points,
                                       grid_marker_size = 1,
                                       view = 'yz')
```


Map View of Stations
ST1
ST2
ST3
Y Coordinate
X Coordinate
Noise Level (m/s)
1e−9
8000
6000
4000
2000
0
−2000
−4000
−4000
−2000
0
2000
4000
6000
8000
5.4
5.2
5.0
4.8
4.6


Side View of Stations: X-Z
ST1
ST3
ST2
0
2000
4000
6000
8000
10000
Z Coordinate
−4000
−2000
0
2000
4000
6000
8000
X Coordinate
4.6
4.8
5.0
5.2
5.4
1e−9
Noise Level (m/s)



-

3.1.3 Prepare to compute magnitude sensitivity

```
# Specify source parameters
#frequency = 10 # Hz
frequency = None # Automatic peak frequency computation

# Minimum S/N for detection on individual channel
min_SNR_p = 2
min_SNR_s = 2

# Minimum stations on which an event must be detected
min_stations_p = 3
min_stations_s = 3

# Save parameters
```

(continues on next page)

(continued from previous page)

```
params = pndmod.classes.DetectionParameters(
    min_SNR_p=min_SNR_p,
    min_SNR_s=min_SNR_s,
    min_stations_p=min_stations_p,
    min_stations_s=min_stations_s,
    f_p=frequency,
    f_s=frequency,
)
params.validate()

# Define slices to use
slice_X = 0
slice_Y = 0
slice_Z = 2500

# Get slice planes
X_plane_points, X_plane_indices = pndmod.utils.get_slice_coordinates(grid_points=grid_
↪points,slice_dimension=0,slice_coordinate=slice_X)
Y_plane_points, Y_plane_indices = pndmod.utils.get_slice_coordinates(grid_points=grid_
↪points,slice_dimension=1,slice_coordinate=slice_Y)
Z_plane_points, Z_plane_indices = pndmod.utils.get_slice_coordinates(grid_points=grid_
↪points,slice_dimension=2,slice_coordinate=slice_Z)
```

3.1.4 Compute and plot magnitude sensitivity for single stations

```
# Get minumum detectable amplitudes
min_amps_p, min_amps_s = pndmod.utils.retrieve_min_amps(geometry_df=geometry_df,
                                                        SNR_p=min_SNR_p,
                                                        SNR_s=min_SNR_s,
                                                        f_p=frequency,
                                                        f_s=frequency)

amps_type = geometry_df.attrs.get('noise_type')

# Loop over stations
for ist in range(len(geometry_df)):
    station_coords = geometry_df[['X', 'Y', 'Z']].to_numpy()[ist]
    print("station_coords:",station_coords)
    print("min_amps_p:",min_amps_p[ist])
    print("min_amps_s:",min_amps_s[ist])


    X_slice_mag_sens_p = pndmod.homo.mag_sensitivity_grid(grid_coords=X_plane_points,
                                                          velocity_df=velocity_df,
                                                          station_coords=station_
↪coords,
                                                          min_amps_p=min_amps_p[ist],
                                                          min_stations_p=1,
                                                          f_p=frequency,
                                                          amps_type=amps_type,
                                                          wave_mode='P')
```

(continues on next page)

(continued from previous page)

```
    Y_slice_mag_sens_p = pndmod.homo.mag_sensitivity_grid(grid_coords=Y_plane_points,
                                                          velocity_df=velocity_df,
                                                          station_coords=station_
↪coords,
                                                          min_amps_p=min_amps_p[ist],
                                                          min_stations_p=1,
                                                          f_p=frequency,
                                                          amps_type=amps_type,
                                                          wave_mode='P')

    Z_slice_mag_sens_p = pndmod.homo.mag_sensitivity_grid(grid_coords=Z_plane_points,
                                                          velocity_df=velocity_df,
                                                          station_coords=station_
↪coords,
                                                          min_amps_p=min_amps_p[ist],
                                                          min_stations_p=1,
                                                          f_p=frequency,
                                                          amps_type=amps_type,
                                                          wave_mode='P')

    X_slice_mag_sens_s = pndmod.homo.mag_sensitivity_grid(grid_coords=X_plane_points,
                                                          velocity_df=velocity_df,
                                                          station_coords=station_
↪coords,
                                                          min_amps_s=min_amps_s[ist],
                                                          min_stations_s=1,
                                                          f_s=frequency,
                                                          amps_type=amps_type,
                                                          wave_mode='S')

    Y_slice_mag_sens_s = pndmod.homo.mag_sensitivity_grid(grid_coords=Y_plane_points,
                                                          velocity_df=velocity_df,
                                                          station_coords=station_
↪coords,
                                                          min_amps_s=min_amps_s[ist],
                                                          min_stations_s=1,
                                                          f_s=frequency,
                                                          amps_type=amps_type,
                                                          wave_mode='S')

    Z_slice_mag_sens_s = pndmod.homo.mag_sensitivity_grid(grid_coords=Z_plane_points,
                                                          velocity_df=velocity_df,
                                                          station_coords=station_
↪coords,
                                                          min_amps_s=min_amps_s[ist],
                                                          min_stations_s=1,
                                                          f_s=frequency,
                                                          amps_type=amps_type,
                                                          wave_mode='S')

    print("Plot magnitude sensitivity for station " + geometry_df['Name'].
```

(continues on next page)

(continued from previous page)

```
↪tolist()[ist] +" and P wave")

    _ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=X_slice_mag_sens_p, yi=gy,
↪ zi=gz,
                                                  plot_title="Magnitude sensitivity␣
↪for station " + geometry_df['Name'].tolist()[ist] + " at X=" + str(slice_X) + " m:␣
↪YZ view, P-wave",
                                                  clb_title="$M_w$")

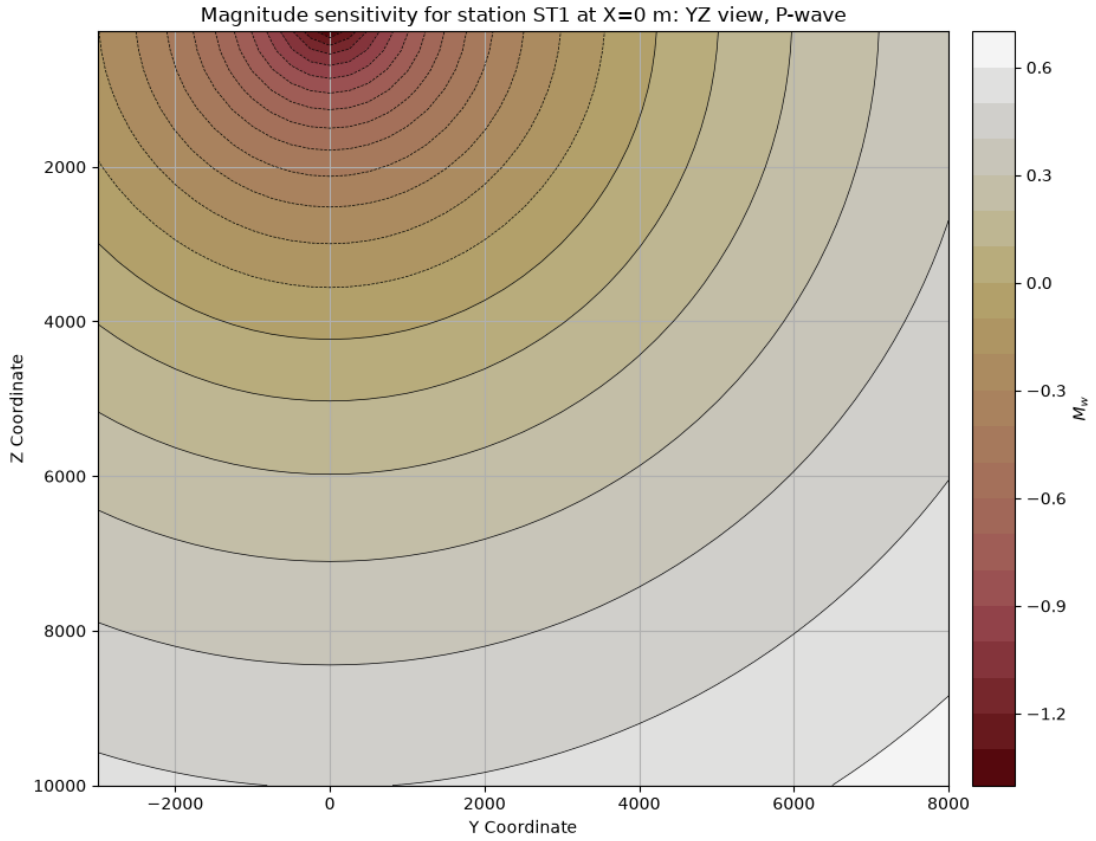
    _ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=Y_slice_mag_sens_p, xi=gx,
↪ zi=gz,
                                                  plot_title="Magnitude sensitivity␣
↪for station " + geometry_df['Name'].tolist()[ist] + " at Y=" + str(slice_Y) + " m:␣
↪XZ view, P-wave",
                                                  clb_title="$M_w$")

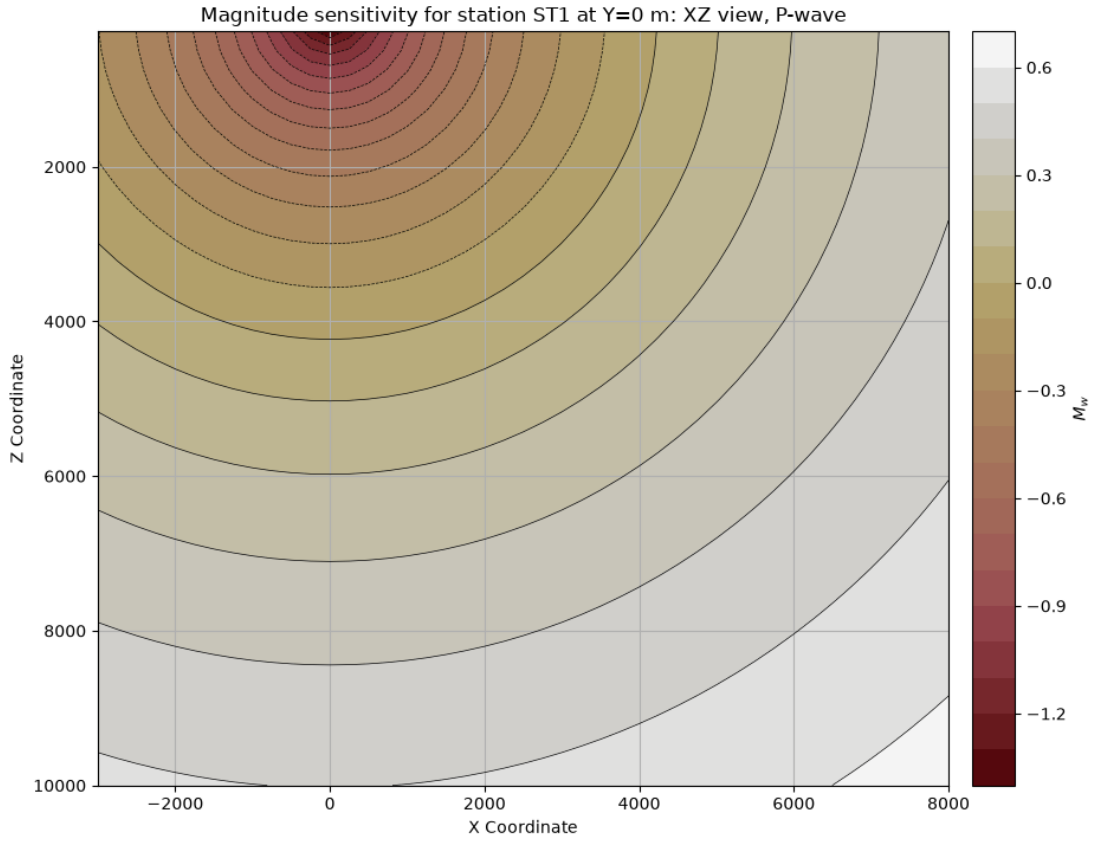
    _ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=Z_slice_mag_sens_p, xi=gx,
↪ yi=gy, geometry_df=geometry_df,
                                                  plot_title="Magnitude sensitivity␣
↪for station " + geometry_df['Name'].tolist()[ist] + " at depth " + str(slice_Z) + "␣
↪m: Map view, P-wave",
                                                  clb_title="$M_w$")

    print("Plot magnitude sensitivity for station " + geometry_df['Name'].
↪tolist()[ist] +" and S wave")
    _ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=X_slice_mag_sens_s, yi=gy,
↪ zi=gz,
                                                  plot_title="Magnitude sensitivity␣
↪for station " + geometry_df['Name'].tolist()[ist] + " at X=" + str(slice_X) + " m:␣
↪YZ view, S-wave",
                                                  clb_title="$M_w$")

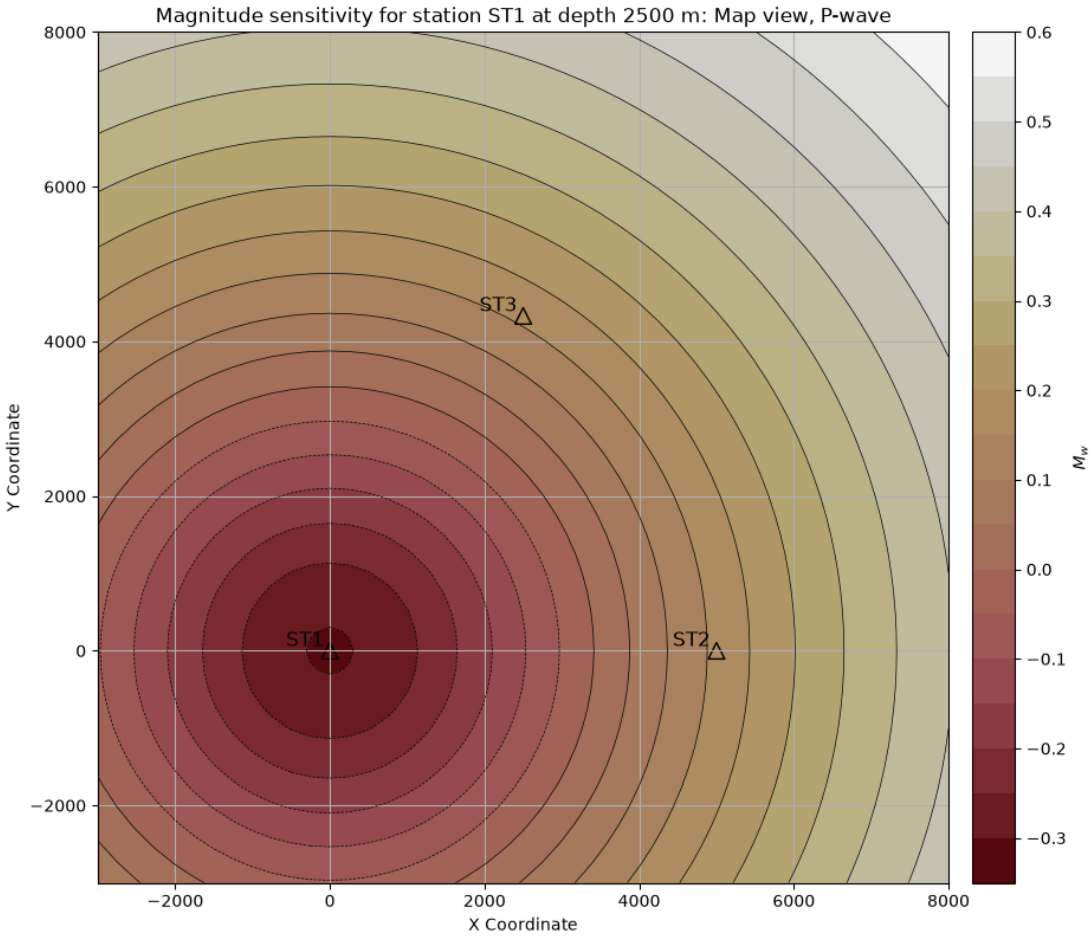
    _ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=Y_slice_mag_sens_s, xi=gx,
↪ zi=gz,
                                                  plot_title="Magnitude sensitivity␣
↪for station " + geometry_df['Name'].tolist()[ist] + " at Y=" + str(slice_Y) + " m:␣
↪XZ view, S-wave",
                                                  clb_title="$M_w$")

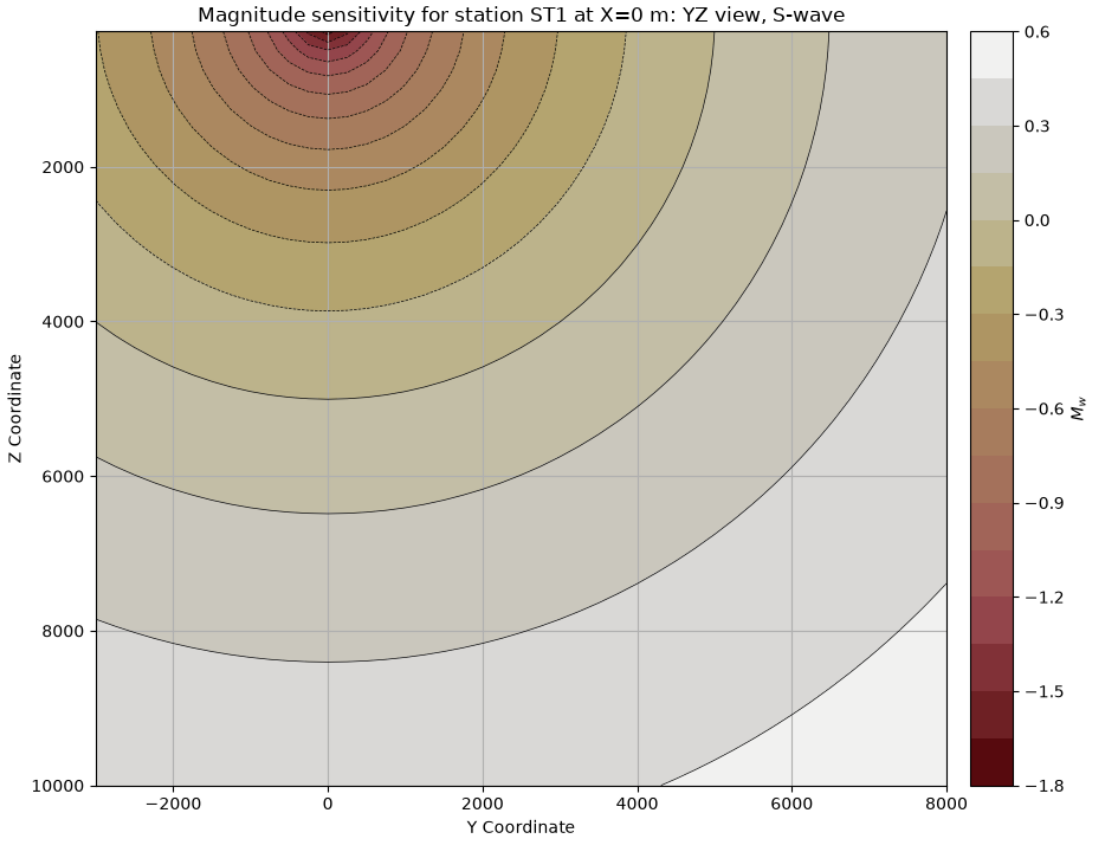
    _ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=Z_slice_mag_sens_s, xi=gx,
↪ yi=gy, geometry_df=geometry_df,
                                                  plot_title="Magnitude sensitivity␣
↪for station " + geometry_df['Name'].tolist()[ist] + " at depth " + str(slice_Z) + "␣
↪m: Map view, S-wave",
                                                  clb_title="$M_w$")
```


Magnitude sensitivity for station ST1 at X=0 m: YZ view, P-wave
Z Coordinate
Y Coordinate
2000
4000
6000
8000
10000
−2000
0
2000
4000
6000
8000
0.6
0.3
0.0
−0.3
−0.6
−0.9
−1.2
M_w

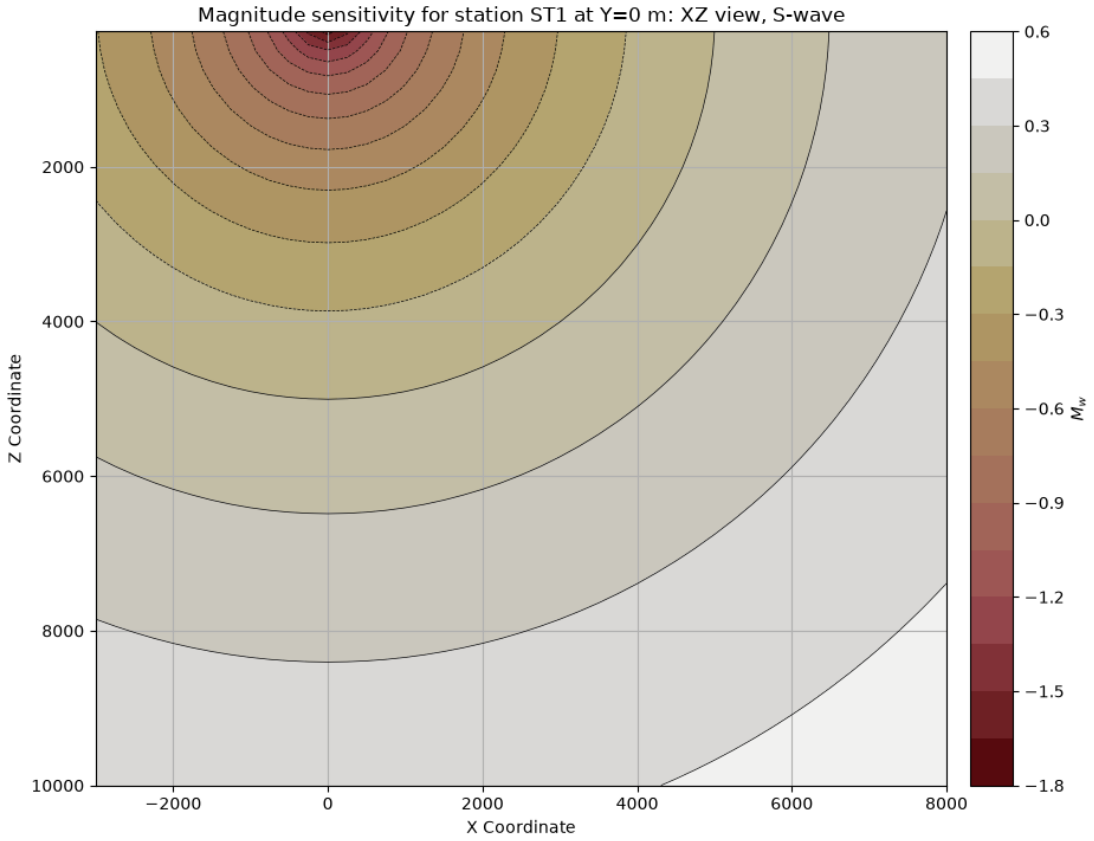

Magnitude sensitivity for station ST1 at Y=0 m: XZ view, P-wave
Z Coordinate
X Coordinate
2000
4000
6000
8000
10000
−2000
0
2000
4000
6000
8000
0.6
0.3
0.0
−0.3
−0.6
−0.9
−1.2
M_w

•

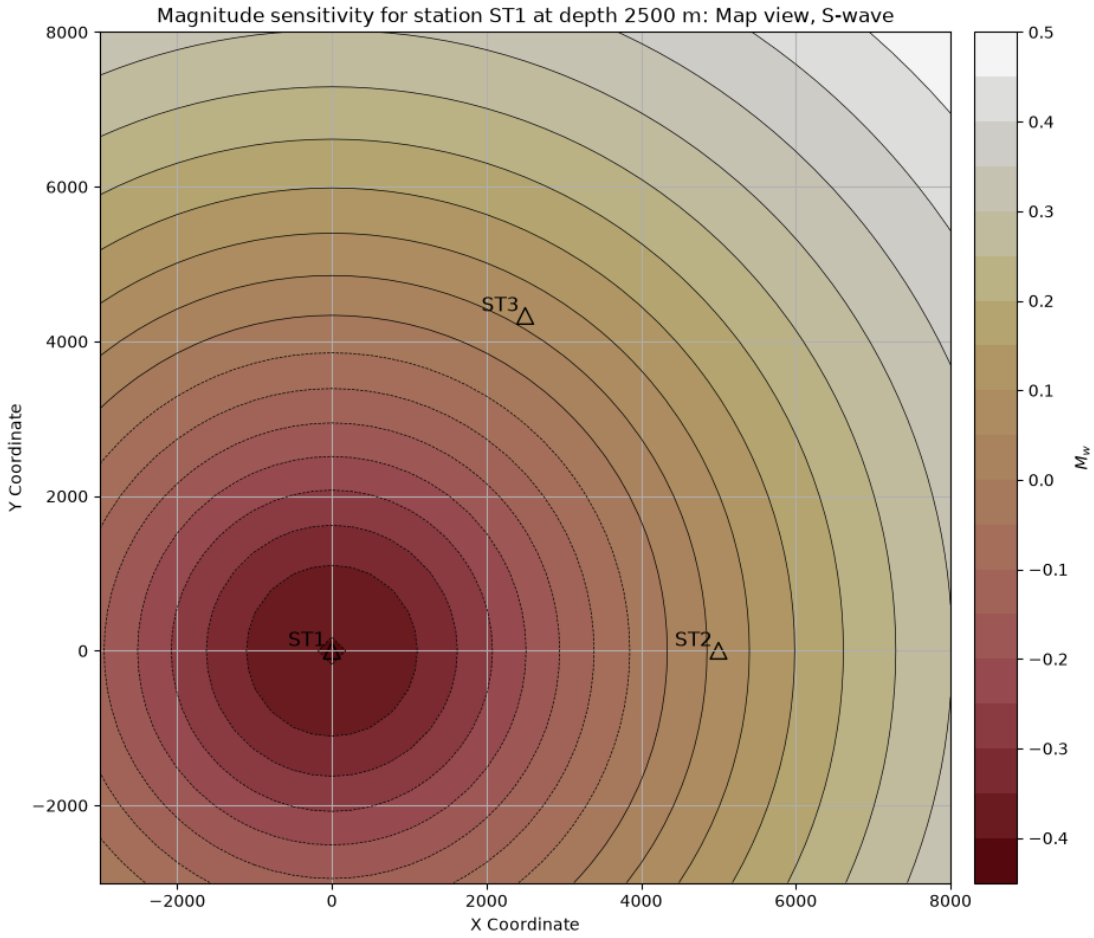

Magnitude sensitivity for station ST1 at depth 2500 m: Map view, P-wave
ST1
ST2
ST3
Y Coordinate
X Coordinate
M_w
8000
6000
4000
2000
0
−2000
−2000
0
2000
4000
6000
8000
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3

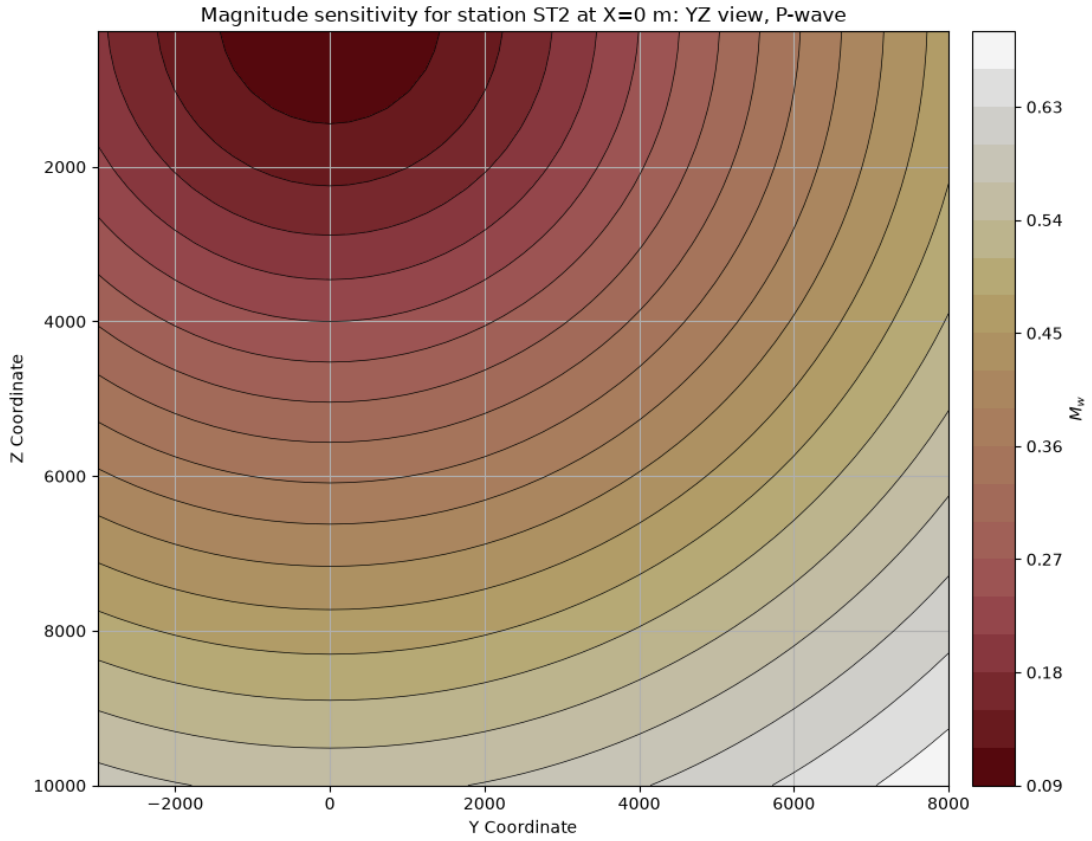

Magnitude sensitivity for station ST1 at X=0 m: YZ view, S-wave
Z Coordinate
Y Coordinate
2000
4000
6000
8000
10000
−2000
0
2000
4000
6000
8000
0.6
0.3
0.0
−0.3
−0.6
−0.9
−1.2
−1.5
−1.8
M_w

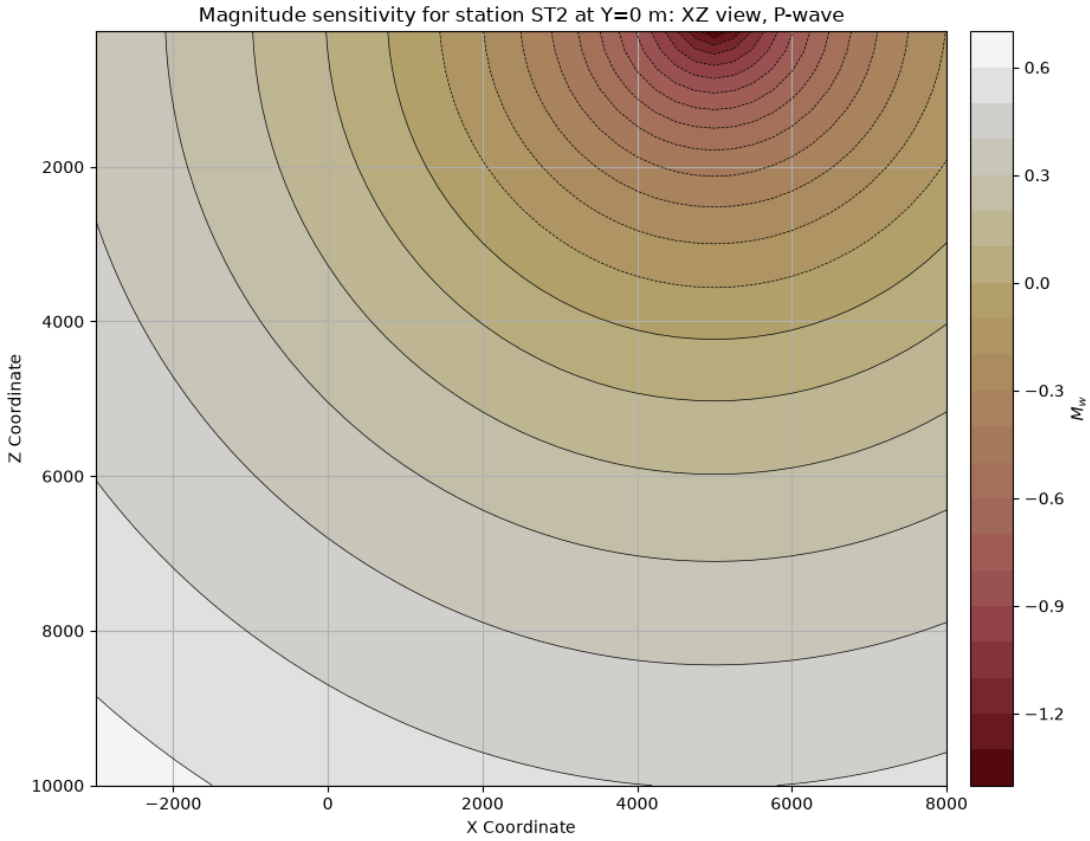
•


Magnitude sensitivity for station ST1 at Y=0 m: XZ view, S-wave
Z Coordinate
X Coordinate
2000
4000
6000
8000
10000
−2000
0
2000
4000
6000
8000
0.6
0.3
0.0
−0.3
−0.6
−0.9
−1.2
−1.5
−1.8
M_w

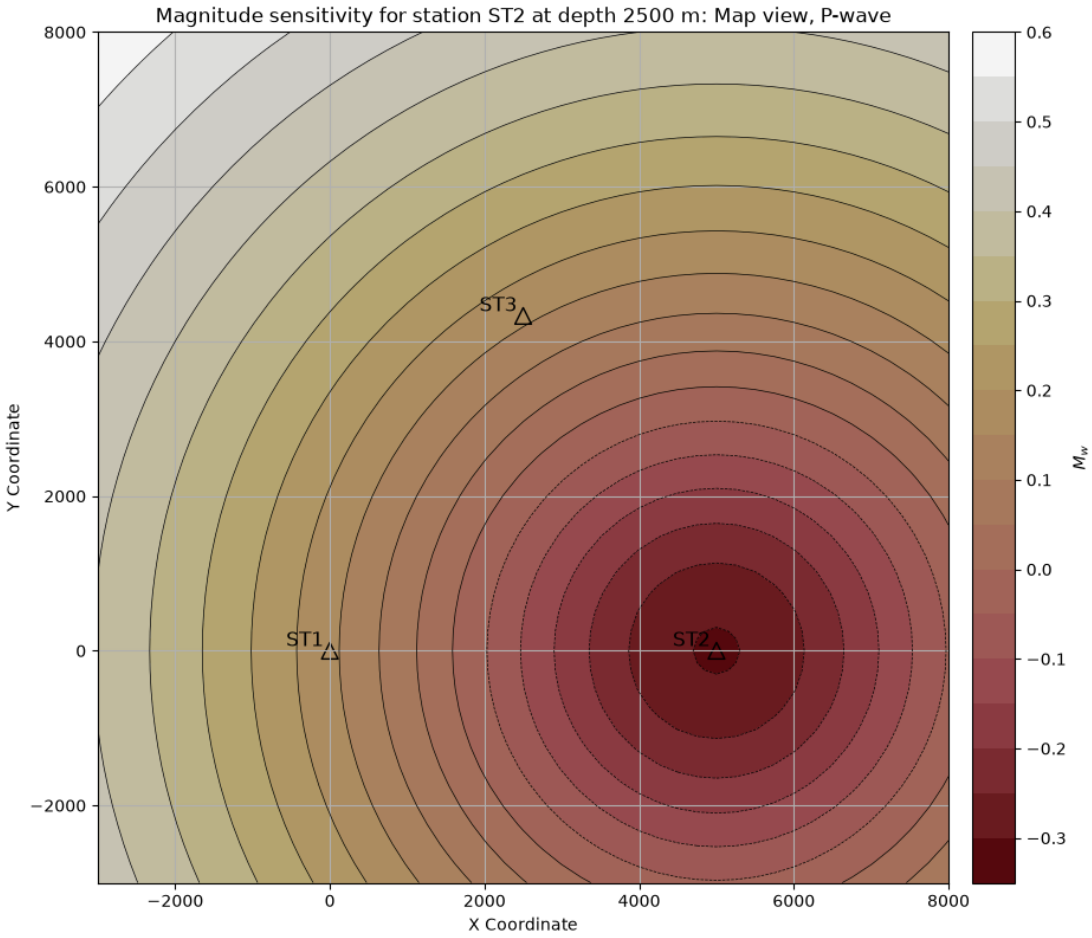
•

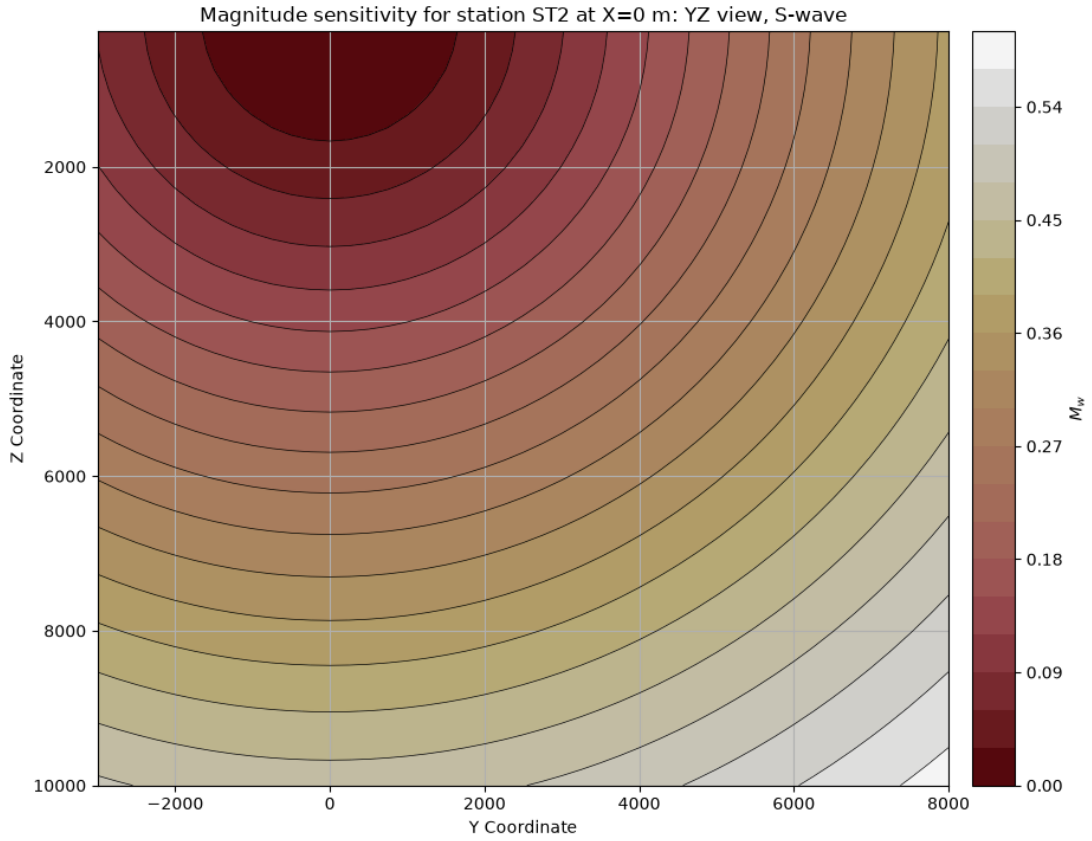

Magnitude sensitivity for station ST1 at depth 2500 m: Map view, S-wave
Y Coordinate
X Coordinate
ST1
ST2
ST3
M_w
8000
6000
4000
2000
0
−2000
−2000
0
2000
4000
6000
8000
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4

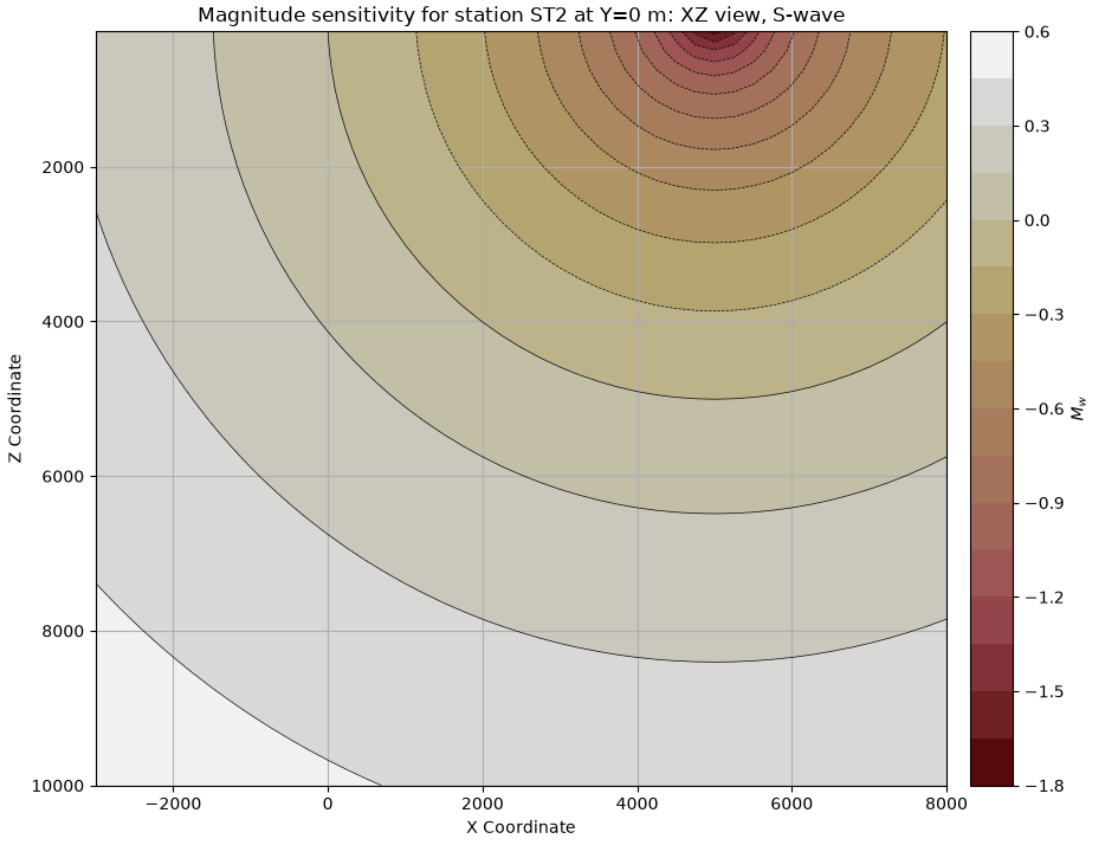

Magnitude sensitivity for station ST2 at X=0 m: YZ view, P-wave
Z Coordinate
Y Coordinate
2000
4000
6000
8000
10000
−2000
0
2000
4000
6000
8000
0.63
0.54
0.45
0.36
0.27
0.18
0.09
M_w


Magnitude sensitivity for station ST2 at Y=0 m: XZ view, P-wave
Z Coordinate
X Coordinate
2000
4000
6000
8000
10000
−2000
0
2000
4000
6000
8000
0.6
0.3
0.0
−0.3
−0.6
−0.9
−1.2
M_w

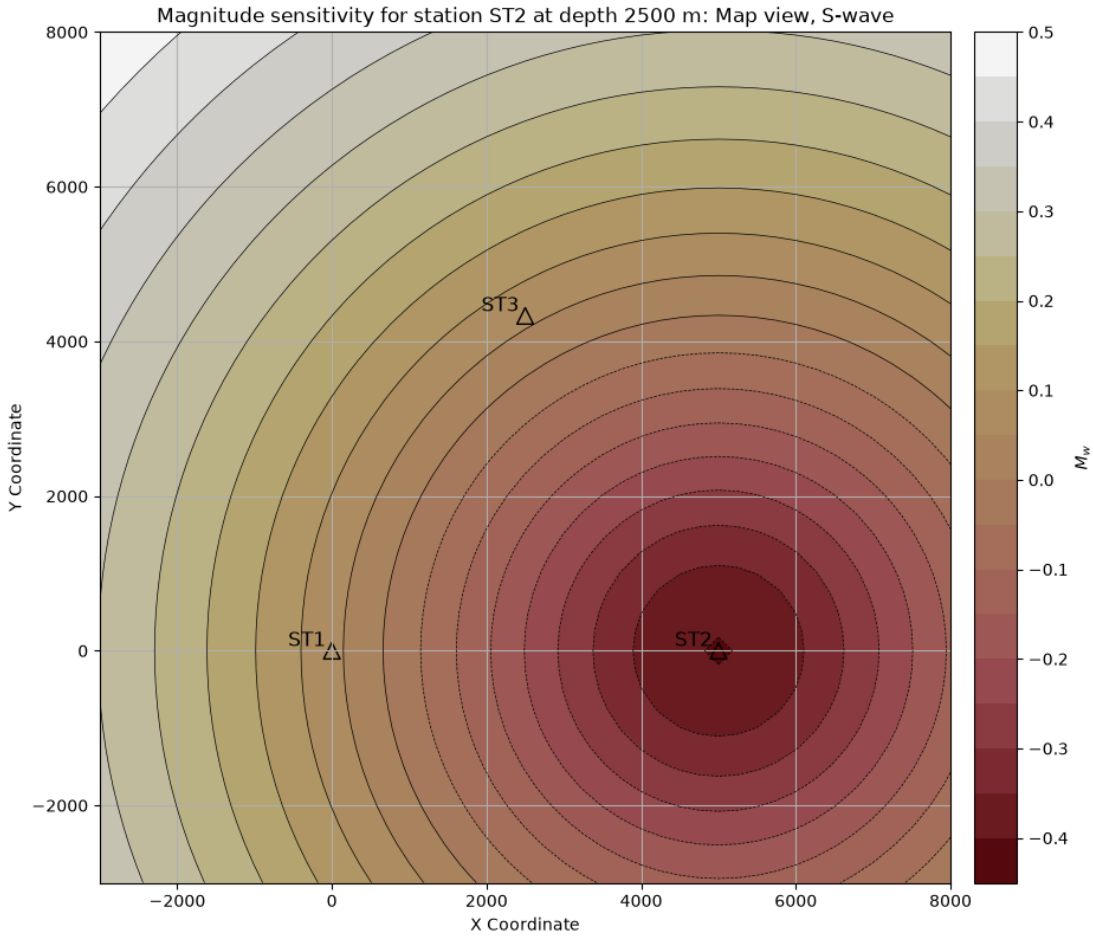
•

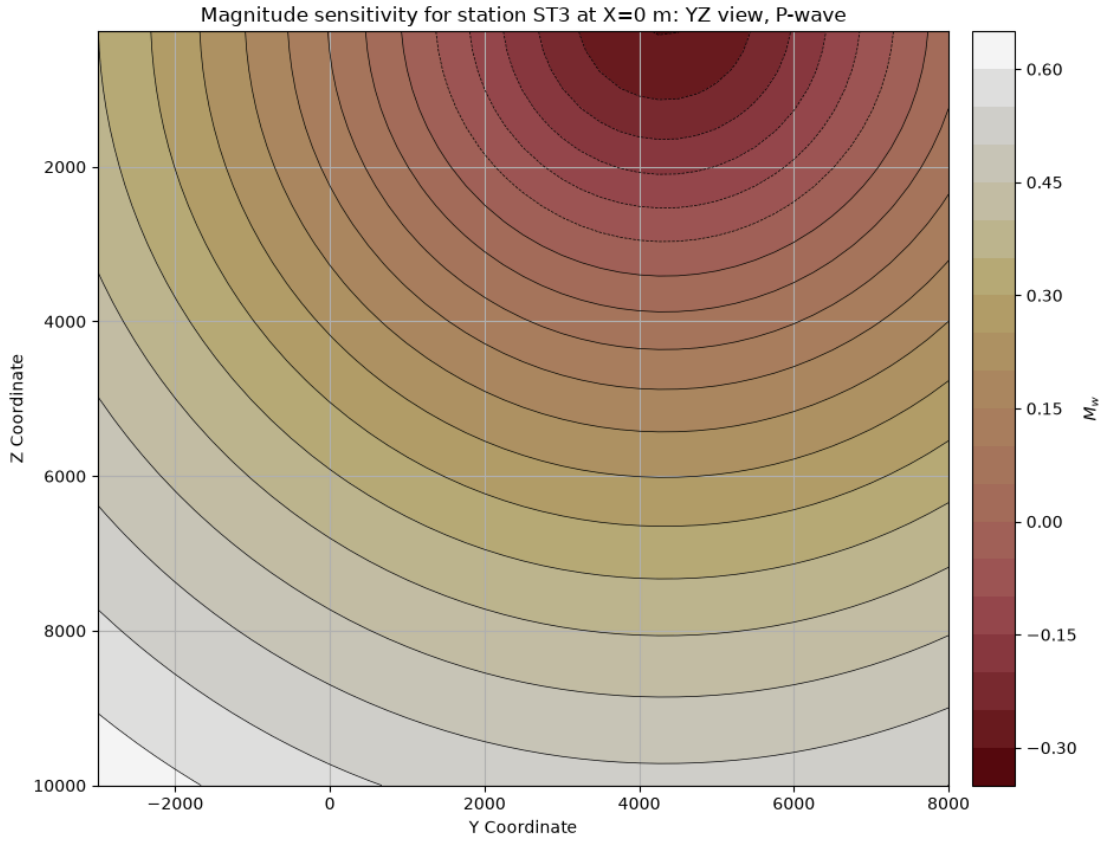

Magnitude sensitivity for station ST2 at depth 2500 m: Map view, P-wave
ST1
ST2
ST3
Y Coordinate
X Coordinate
M_w
8000
6000
4000
2000
0
−2000
−2000
0
2000
4000
6000
8000
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3

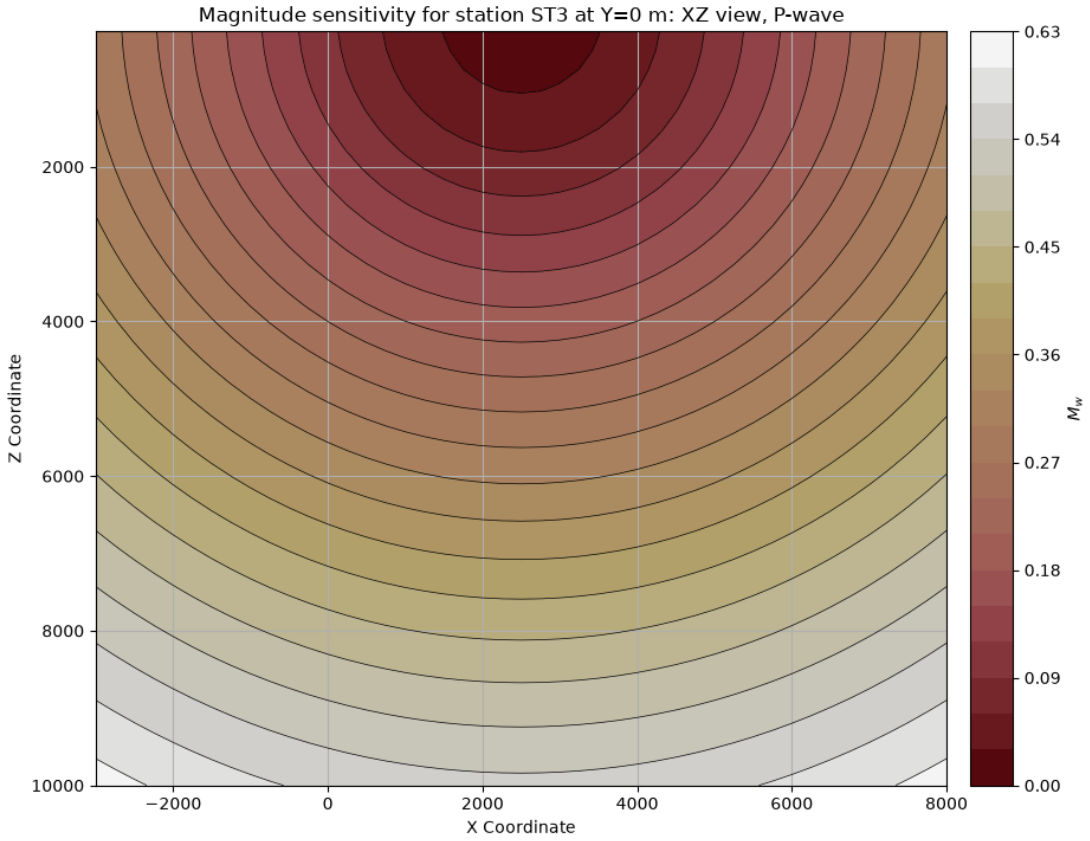

Magnitude sensitivity for station ST2 at X=0 m: YZ view, S-wave
Z Coordinate
Y Coordinate
2000
4000
6000
8000
10000
−2000
0
2000
4000
6000
8000
0.54
0.45
0.36
0.27
0.18
0.09
0.00
M_w


Magnitude sensitivity for station ST2 at Y=0 m: XZ view, S-wave
Z Coordinate
X Coordinate
2000
4000
6000
8000
10000
−2000
0
2000
4000
6000
8000
0.6
0.3
0.0
−0.3
−0.6
−0.9
−1.2
−1.5
−1.8
M_w

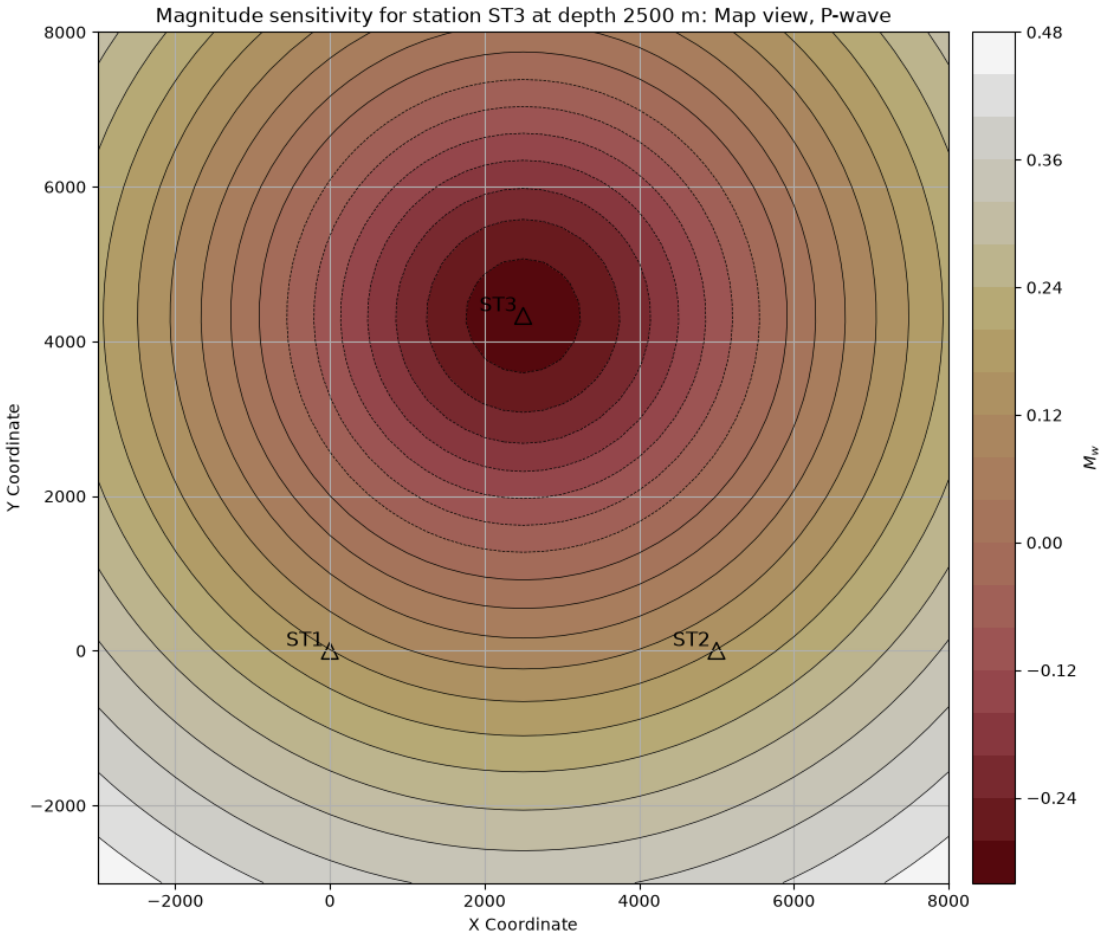
•

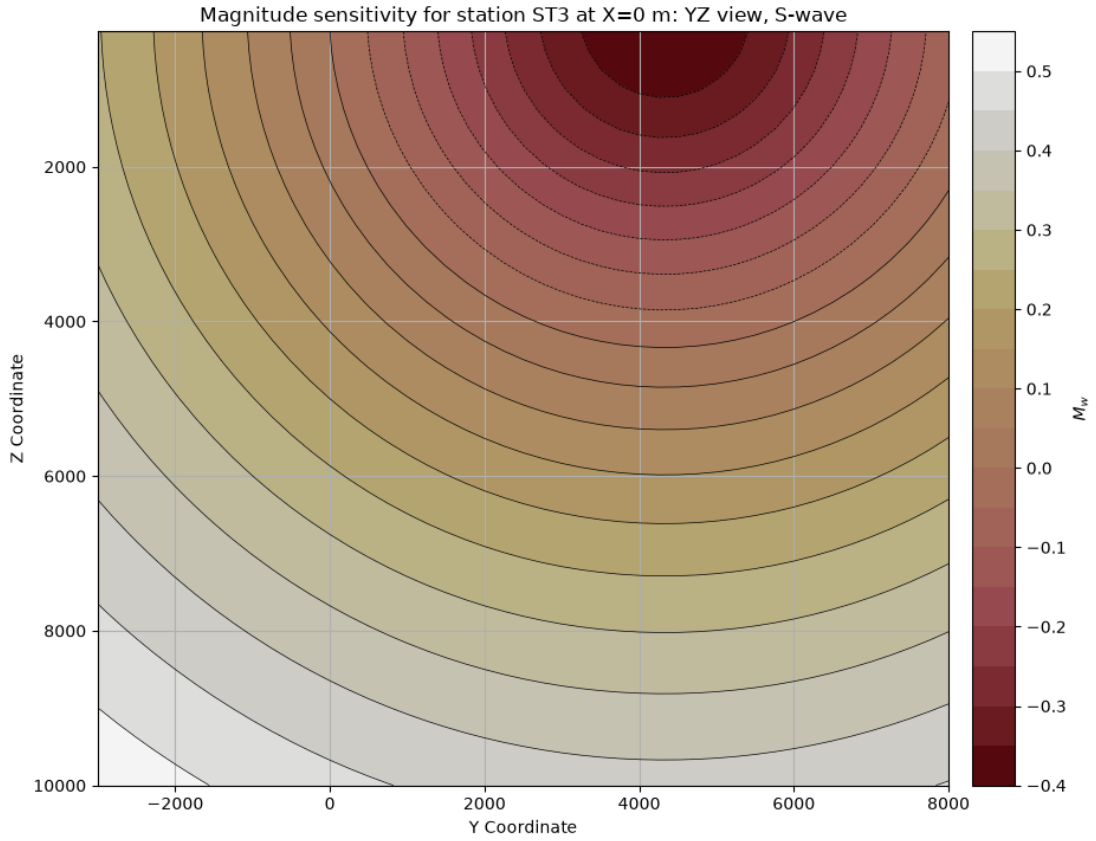

Magnitude sensitivity for station ST2 at depth 2500 m: Map view, S-wave
ST1
ST2
ST3
Y Coordinate
X Coordinate
M_w
8000
6000
4000
2000
0
−2000
−2000
0
2000
4000
6000
8000
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4


Magnitude sensitivity for station ST3 at X=0 m: YZ view, P-wave
Z Coordinate
Y Coordinate
2000
4000
6000
8000
10000
−2000
0
2000
4000
6000
8000
0.60
0.45
0.30
0.15
0.00
−0.15
−0.30
M_w

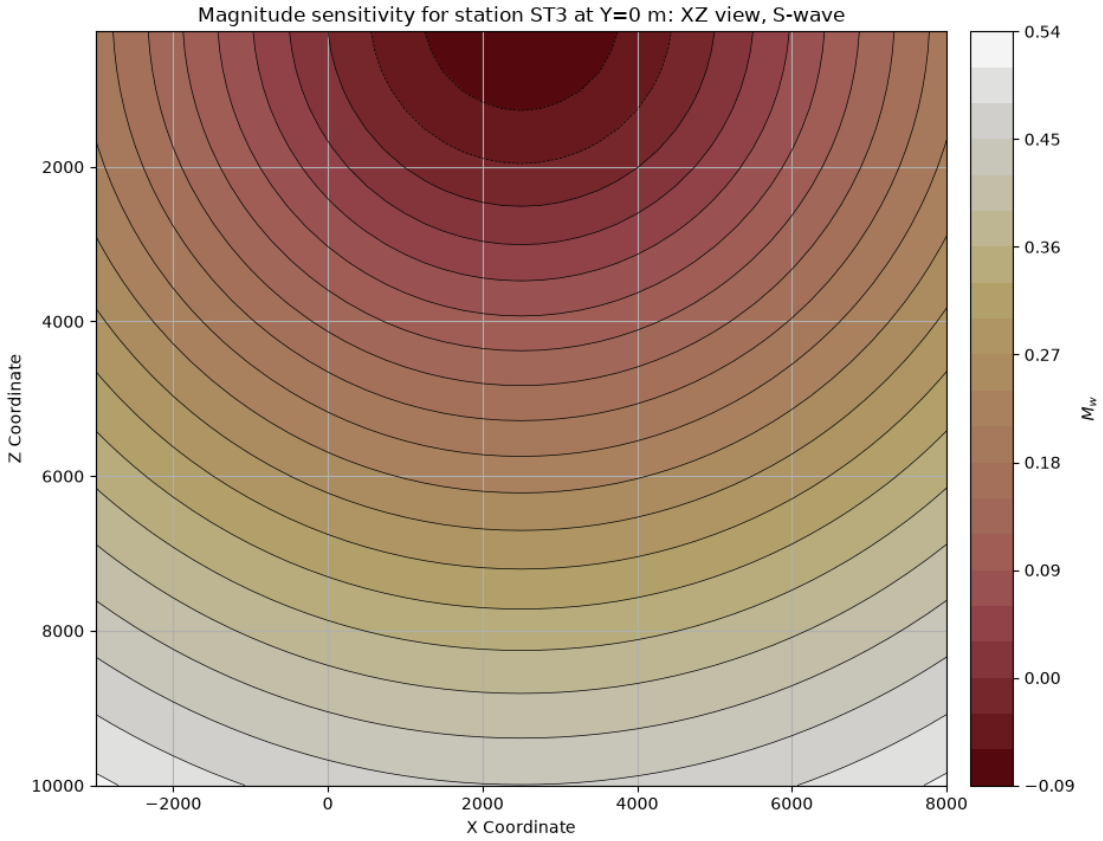

Magnitude sensitivity for station ST3 at Y=0 m: XZ view, P-wave
Z Coordinate
X Coordinate
2000
4000
6000
8000
10000
−2000
0
2000
4000
6000
8000
0.63
0.54
0.45
0.36
0.27
0.18
0.09
0.00
M_w

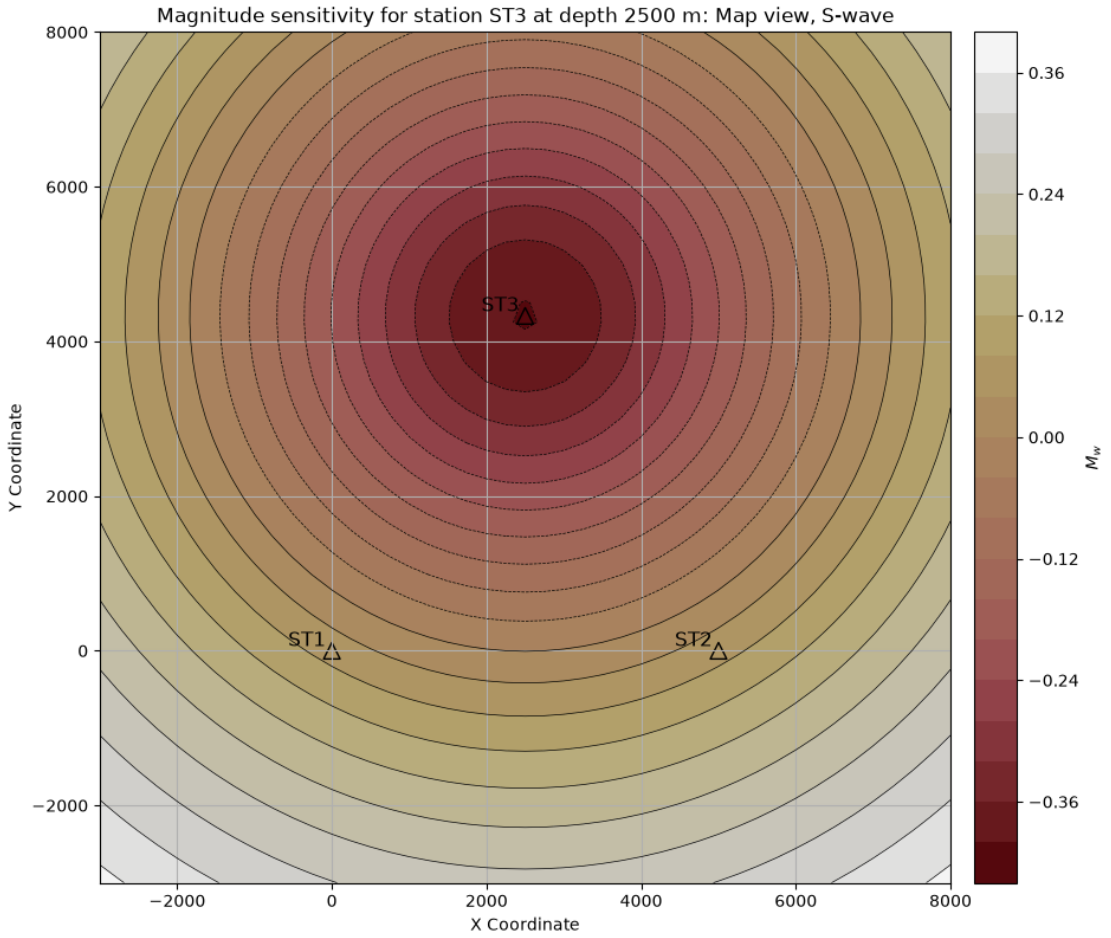
•


Magnitude sensitivity for station ST3 at depth 2500 m: Map view, P-wave
ST1
ST2
ST3
Y Coordinate
X Coordinate
M_w
8000
6000
4000
2000
0
−2000
−2000
0
2000
4000
6000
8000
0.48
0.36
0.24
0.12
0.00
−0.12
−0.24


Magnitude sensitivity for station ST3 at X=0 m: YZ view, S-wave
Z Coordinate
Y Coordinate
2000
4000
6000
8000
10000
−2000
0
2000
4000
6000
8000
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
M_w

•


Magnitude sensitivity for station ST3 at Y=0 m: XZ view, S-wave
Z Coordinate
X Coordinate
2000
4000
6000
8000
10000
−2000
0
2000
4000
6000
8000
0.54
0.45
0.36
0.27
0.18
0.09
0.00
−0.09
M_w



•

```
station_coords: [ 0.  0. -0.]
min_amps_p: 1e-08
min_amps_s: 1e-08
Plot magnitude sensitivity for station ST1 and P wave
Plot magnitude sensitivity for station ST1 and S wave
station_coords: [5000.    0.   -0.]
min_amps_p: 1e-08
min_amps_s: 1e-08
Plot magnitude sensitivity for station ST2 and P wave
Plot magnitude sensitivity for station ST2 and S wave
station_coords: [2500. 4330.   -0.]
min_amps_p: 1e-08
min_amps_s: 1e-08
Plot magnitude sensitivity for station ST3 and P wave
Plot magnitude sensitivity for station ST3 and S wave
```

3.1.5 Compute magnitude sensitivity for several stations using the whole grid

```
# Compute magnitude sensitivity for the whole grid for P
print("Computing magnitude sensitivity for the whole grid for P wave...")
start_time = time()
mag_sens_grid_p = pndmod.core.get_mag_sensitivity(grid_coords=grid_points,
                                                  geometry_df=geometry_df,
                                                  velocity_df=velocity_df,
                                                  params=params,
                                                  wave_mode='P')
end_time = time()
print(f"Computation time: {end_time - start_time} seconds")

# Compute magnitude sensitivity for the whole grid for S
print("Computing magnitude sensitivity for the whole grid for S wave...")
start_time = time()
mag_sens_grid_s = pndmod.core.get_mag_sensitivity(grid_coords=grid_points,
                                                  geometry_df=geometry_df,
                                                  velocity_df=velocity_df,
                                                  params=params,
                                                  wave_mode='S')
end_time = time()
print(f"Computation time: {end_time - start_time} seconds")


# Compute magnitude sensitivity for the whole grid for P and S waves
print("Computing magnitude sensitivity for the whole grid for both P and S waves...")
start_time = time()
mag_sens_grid_ps = pndmod.core.get_mag_sensitivity(grid_coords=grid_points,
                                                   geometry_df=geometry_df,
                                                   velocity_df=velocity_df,
                                                   params=params,
                                                   wave_mode='PS')
end_time = time()
print(f"Computation time: {end_time - start_time} seconds")
```

```
Computing magnitude sensitivity for the whole grid for P wave...
Computation time: 0.038500070571899414 seconds
Computing magnitude sensitivity for the whole grid for S wave...
Computation time: 0.033838748931884766 seconds
Computing magnitude sensitivity for the whole grid for both P and S waves...
Computation time: 0.047976016998291016 seconds
```

3.1.6 Get sensitivity slices by cutting from the grid

```
X_slice_mag_sens_stations_p = mag_sens_grid_p[X_plane_indices]
Y_slice_mag_sens_stations_p = mag_sens_grid_p[Y_plane_indices]
Z_slice_mag_sens_stations_p = mag_sens_grid_p[Z_plane_indices]

X_slice_mag_sens_stations_s = mag_sens_grid_s[X_plane_indices]
Y_slice_mag_sens_stations_s = mag_sens_grid_s[Y_plane_indices]
Z_slice_mag_sens_stations_s = mag_sens_grid_s[Z_plane_indices]
```

(continues on next page)

(continued from previous page)

```
X_slice_mag_sens_stations_ps = mag_sens_grid_ps[X_plane_indices]
Y_slice_mag_sens_stations_ps = mag_sens_grid_ps[Y_plane_indices]
Z_slice_mag_sens_stations_ps = mag_sens_grid_ps[Z_plane_indices]
```

3.1.7 Plot magnitude sensitivity for several stations

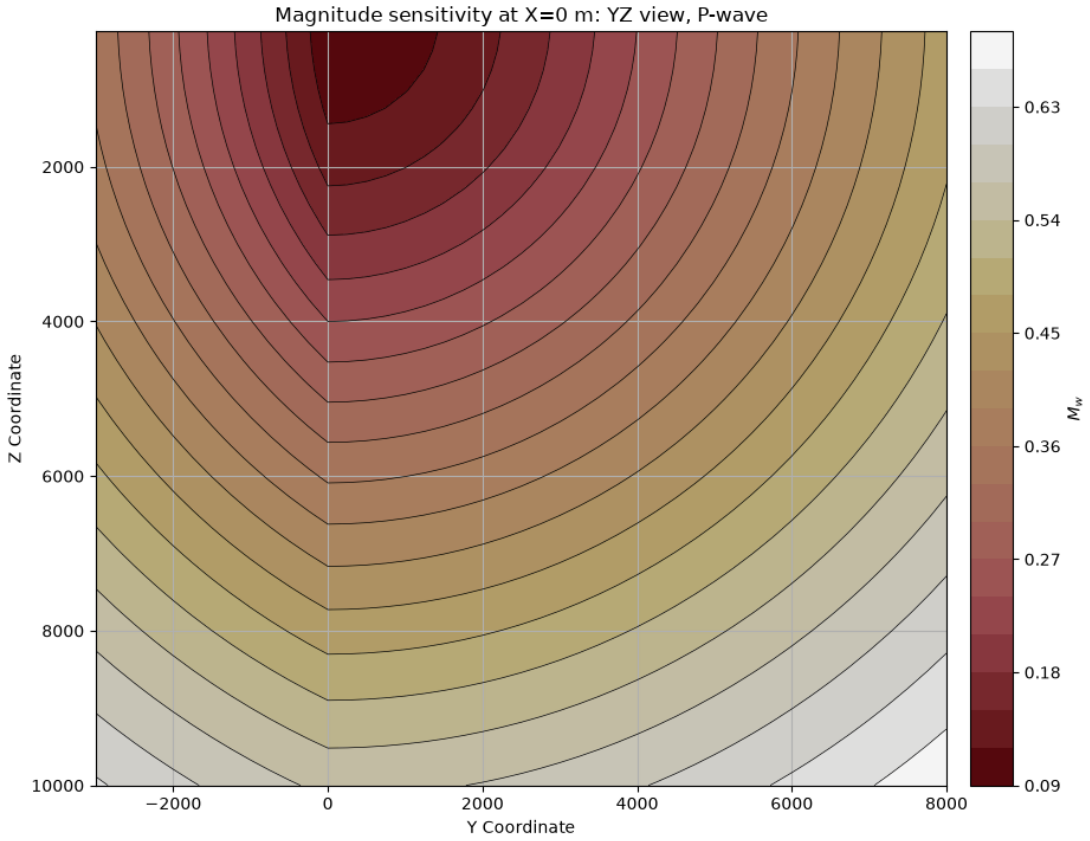
Plot magnitude sensitivity for P wave

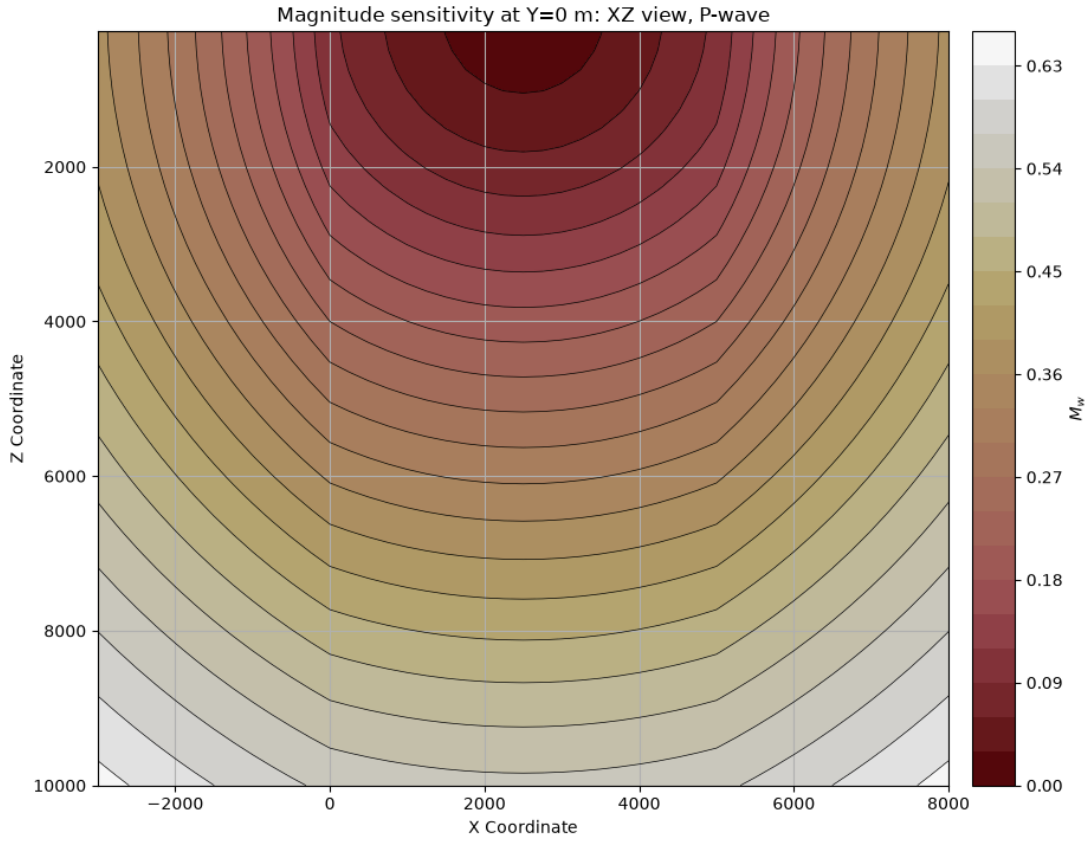
```
clb_range = None
plt_levels = None

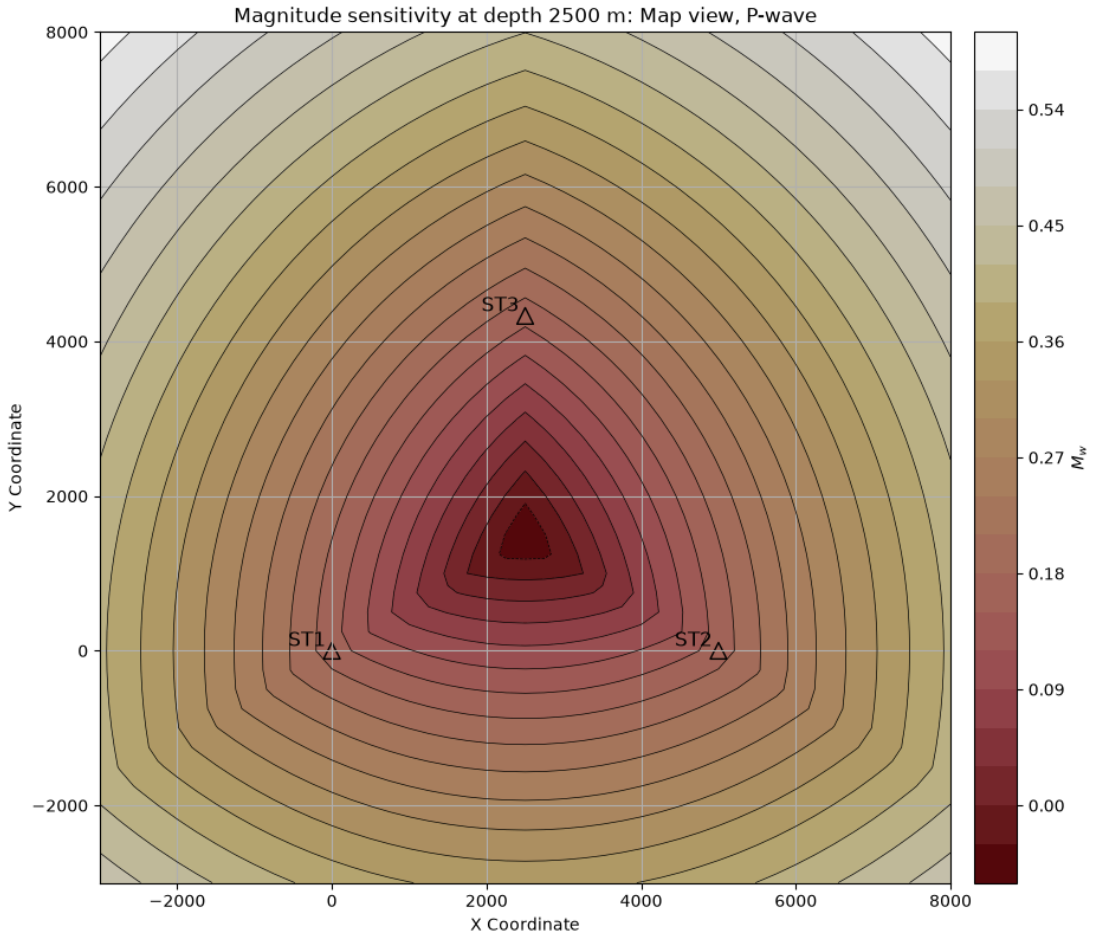
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=X_slice_mag_sens_stations_p,␣
↪yi=gy, zi=gz,
                                              plot_title="Magnitude sensitivity at X=
↪" + str(slice_X) + " m: YZ view, P-wave",
                                              clb_title="$M_w$",
                                              clb_range=clb_range,
                                              plt_levels=plt_levels)

_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=Y_slice_mag_sens_stations_p,␣
↪xi=gx, zi=gz,
                                              plot_title="Magnitude sensitivity at Y=
↪" + str(slice_Y) + " m: XZ view, P-wave",
                                              clb_title="$M_w$",
                                              clb_range=clb_range,
                                              plt_levels=plt_levels)

_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=Z_slice_mag_sens_stations_p,␣
↪xi=gx, yi=gy, geometry_df=geometry_df,
                                              plot_title="Magnitude sensitivity at␣
↪depth " + str(slice_Z) + " m: Map view, P-wave",
                                              clb_title="$M_w$",
                                              clb_range=clb_range,
                                              plt_levels=plt_levels)
```


Magnitude sensitivity at X=0 m: YZ view, P-wave
Z Coordinate
Y Coordinate
2000
4000
6000
8000
10000
−2000
0
2000
4000
6000
8000
0.63
0.54
0.45
0.36
0.27
0.18
0.09
M_w


Magnitude sensitivity at Y=0 m: XZ view, P-wave
Z Coordinate
X Coordinate
2000
4000
6000
8000
10000
−2000
0
2000
4000
6000
8000
0.63
0.54
0.45
0.36
0.27
0.18
0.09
0.00
M_w



-

Plot magnitude sensitivity for S wave

```
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=X_slice_mag_sens_stations_s,␣
↪yi=gy, zi=gz,
                                              plot_title="Magnitude sensitivity at X=
↪" + str(slice_X) + " m: YZ view, S-wave",
                                              clb_title="$M_w$",
                                              clb_range=clb_range,
                                              plt_levels=plt_levels)

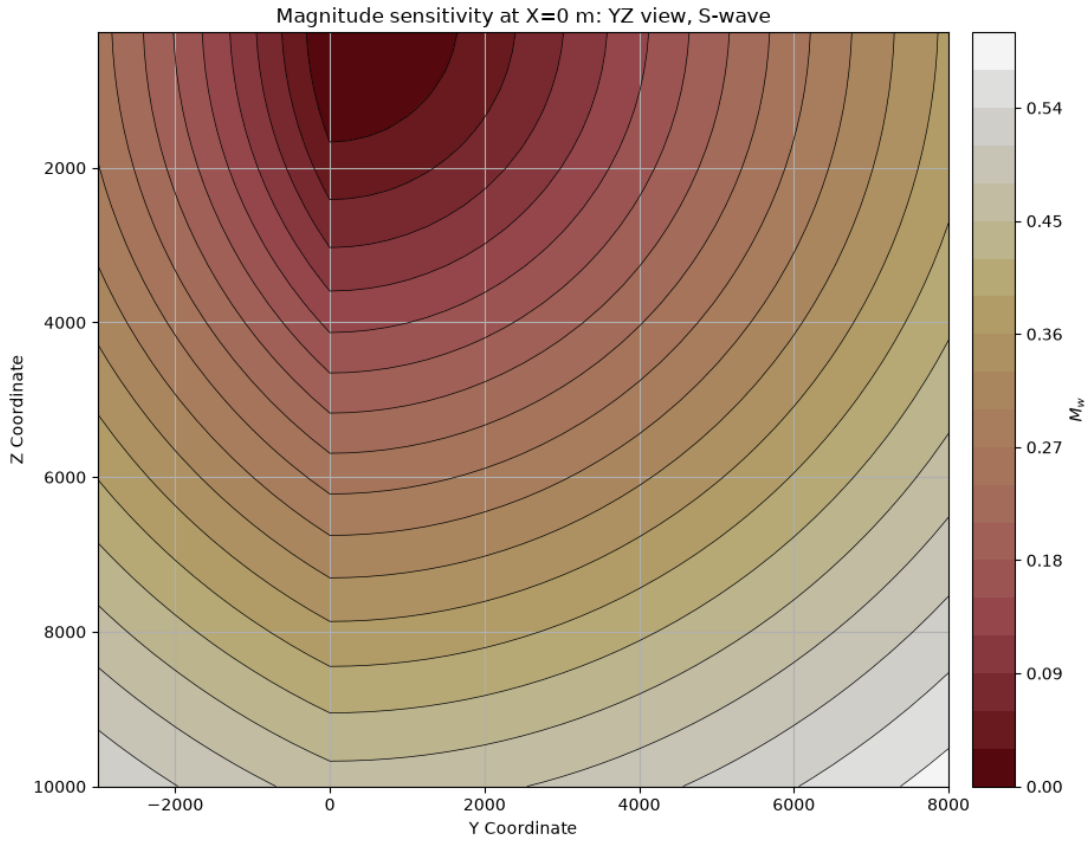
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=Y_slice_mag_sens_stations_s,␣
↪xi=gx, zi=gz,
                                              plot_title="Magnitude sensitivity at Y=
↪" + str(slice_Y) + " m: XZ view, S-wave",
                                              clb_title="$M_w$",
```

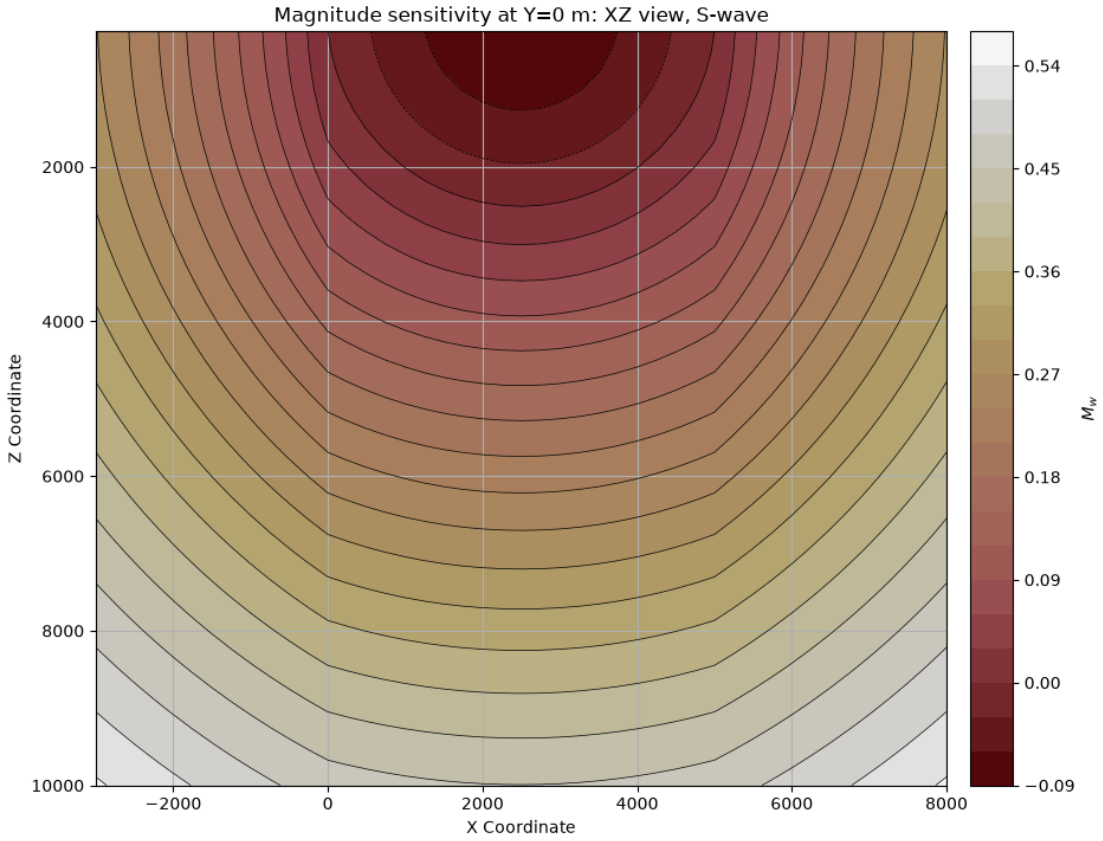
(continues on next page)

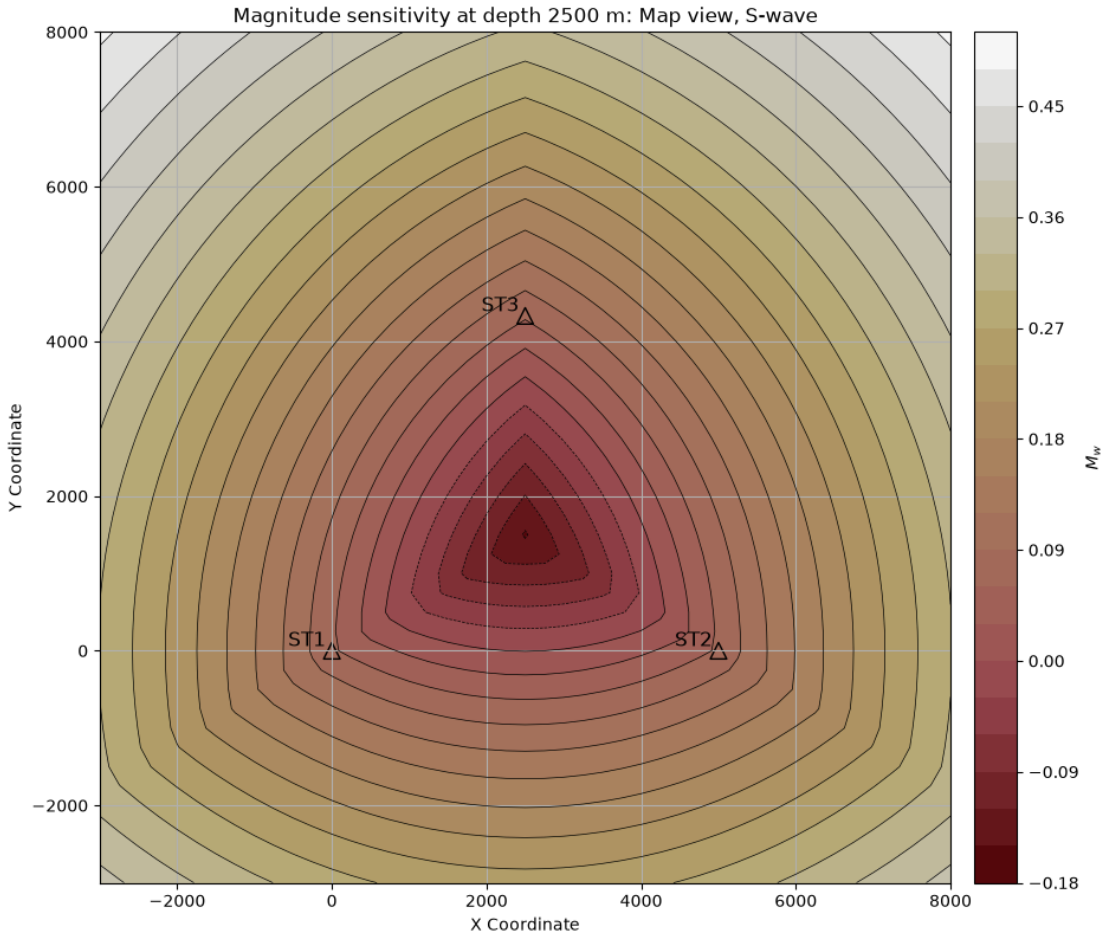
(continued from previous page)

```
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)

_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=Z_slice_mag_sens_stations_s,␣
↪xi=gx, yi=gy, geometry_df=geometry_df,
                                                plot_title="Magnitude sensitivity at␣
↪depth " + str(slice_Z) + " m: Map view, S-wave",
                                                clb_title="$M_w$",
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)
```




Magnitude sensitivity at Y=0 m: XZ view, S-wave
Z Coordinate
X Coordinate
2000
4000
6000
8000
10000
−2000
0
2000
4000
6000
8000
0.54
0.45
0.36
0.27
0.18
0.09
0.00
−0.09
M_w



-

Plot magnitude sensitivity for P and S waves

```
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=X_slice_mag_sens_stations_ps,␣
↪yi=gy, zi=gz,
                                              plot_title="Magnitude sensitivity at X=
↪" + str(slice_X) + " m: YZ view, P- and S-waves",
                                              clb_title="$M_w$",
                                              clb_range=clb_range,
                                              plt_levels=plt_levels)

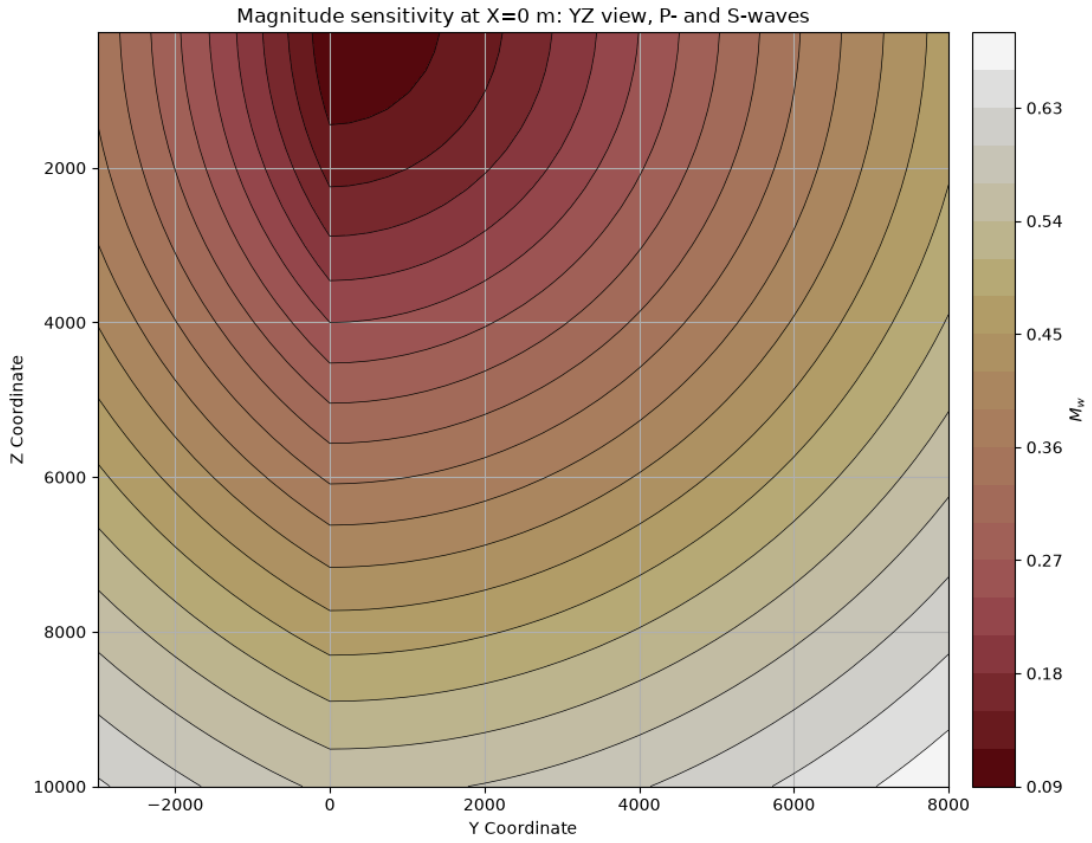
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=Y_slice_mag_sens_stations_ps,␣
↪xi=gx, zi=gz,
                                              plot_title="Magnitude sensitivity at Y=
↪" + str(slice_Y) + " m: XZ view, P- and S-waves",
                                              clb_title="$M_w$",
```

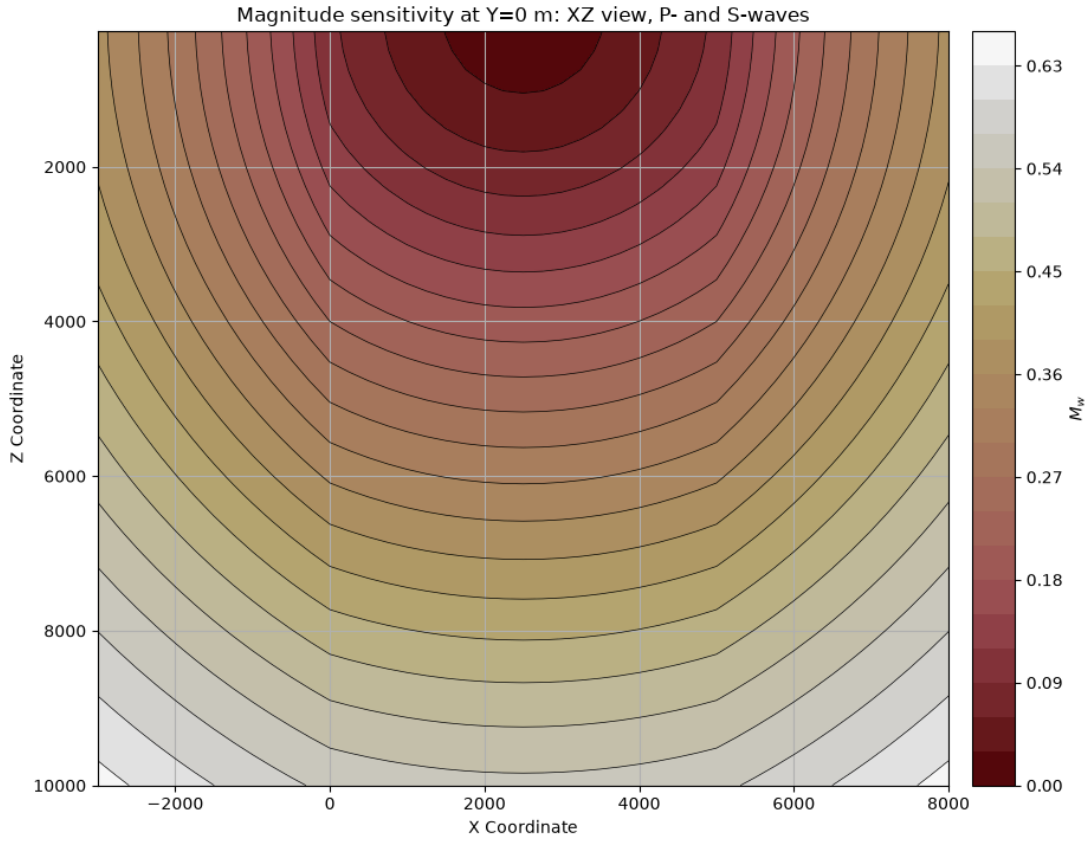
(continues on next page)

(continued from previous page)

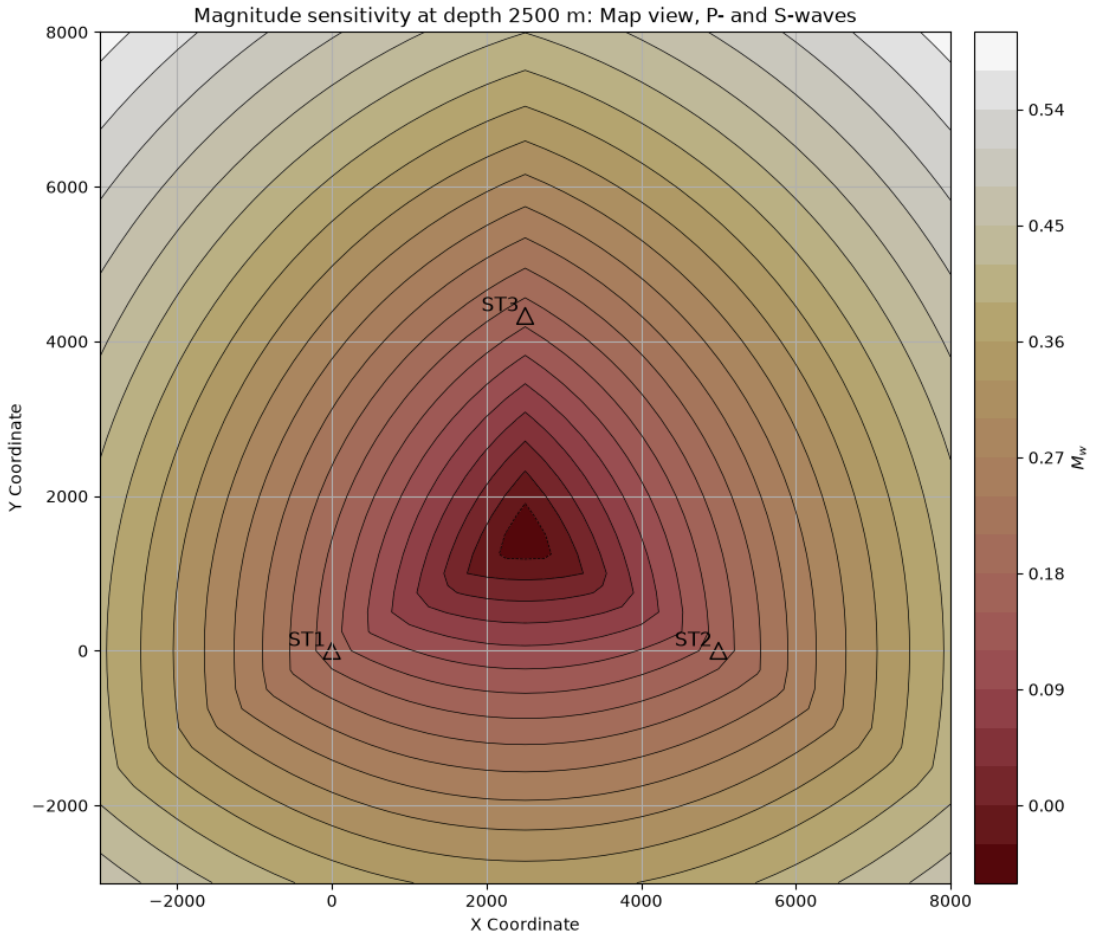
```
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)

_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=Z_slice_mag_sens_stations_ps,␣
↪xi=gx, yi=gy, geometry_df=geometry_df,
                                                plot_title="Magnitude sensitivity at␣
↪depth " + str(slice_Z) + " m: Map view, P- and S-waves",
                                                clb_title="$M_w$",
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)
```




Magnitude sensitivity at Y=0 m: XZ view, P- and S-waves
Z Coordinate
X Coordinate
2000
4000
6000
8000
10000
−2000
0
2000
4000
6000
8000
0.63
0.54
0.45
0.36
0.27
0.18
0.09
0.00
M_w

•



-

Total running time of the script: (0 minutes 4.817 seconds)

3.2 02.1 Magnitude sensitivity for homogeneous model: test case 2

This is an example of computing magnitude sensitivity for a homogeneous model.

Test case 2:

- Geometry: simple station geometry with 4 surface stations in the form of a triangle with 1 station in the center
- Velocity model: homogeneous, Vp = 3100 m/s, Vp/Vs = 1.73, Qp = 70, Qs = 50, rho = 2400 kg/m^3
- Processing: magnitude sensitivity computation for all stations (for grid slices only).

3.2.1 Load all necessary packages

```
import pynetdesign.visualization as pndvis
import pynetdesign.modelling as pndmod
import numpy as np
from time import time

# sphinx_gallery_thumbnail_number = -2
```

3.2.2 Input

Read geometry

```
geometry_file_path = '../data/homo/test_02/TMPL_net_geometryA.txt'
geometry_df = pndmod.io.read_geometry(geometry_file_path)

print(geometry_df)
print(geometry_df.attrs)
```

```
  Name       Y       X    Z    NoiseLevel
0  ST1     0.0     0.0 -0.0  1.000000e-07
1  ST2 -1470.5  2547.0 -0.0  1.000000e-07
2  ST3 -1470.5 -2547.0 -0.0  1.000000e-07
3  ST4  2941.0     0.0 -0.0  1.000000e-07
{'file_path': '../data/homo/test_02/TMPL_net_geometryA.txt', 'original_coord_units':
↪'m', 'coordinate_system': 'local', 'noise_type': 'velocity', 'noise_unit': 'm/s', '_
↪is_combined': False}
```

Read velocity model

```
velocity_file_path = '../data/homo/test_02/TMPL_velocity_model.txt'
velocity_df = pndmod.io.read_velocity(velocity_file_path)

print(velocity_df)
print(velocity_df.attrs)
```

```
   Depth    Vp  VpVsRatio  Qp  Qs   Rho   Ep   Dt   Gm
0  40000  3100       1.73  70  50  2400  0.0  0.0  0.0
{'model_type': 'homo', 'model_anisotropy': 'isotropic', 'parameter_descriptions': {
↪'Depth': 'Depth', 'Vp': 'P-wave velocity', 'Vs': 'S-wave velocity', 'VpVsRatio':
↪'Vp/Vs', 'Qp': 'P-wave quality factor', 'Qs': 'S-wave quality factor', 'Rho':
↪'density', 'Ep': 'Thomsen epsilon parameter', 'Dt': 'Thomsen delta parameter', 'Gm
↪': 'Thomsen gamma parameter'}, 'parameter_units': {'Depth': 'm', 'Vp': 'm/s', 'Vs':
↪'m/s', 'VpVsRatio': '', 'Qp': '', 'Qs': '', 'Rho': 'kg/m^3', 'Ep': '', 'Dt': '', 'Gm
↪': ''}}
```

Set imaging grid parameters

```
# Imaging grid parameters
gx = np.arange(-4500, 4500 + 250, 250)
gy = np.arange(-3750, 4750 + 250, 250)
```

(continues on next page)

(continued from previous page)

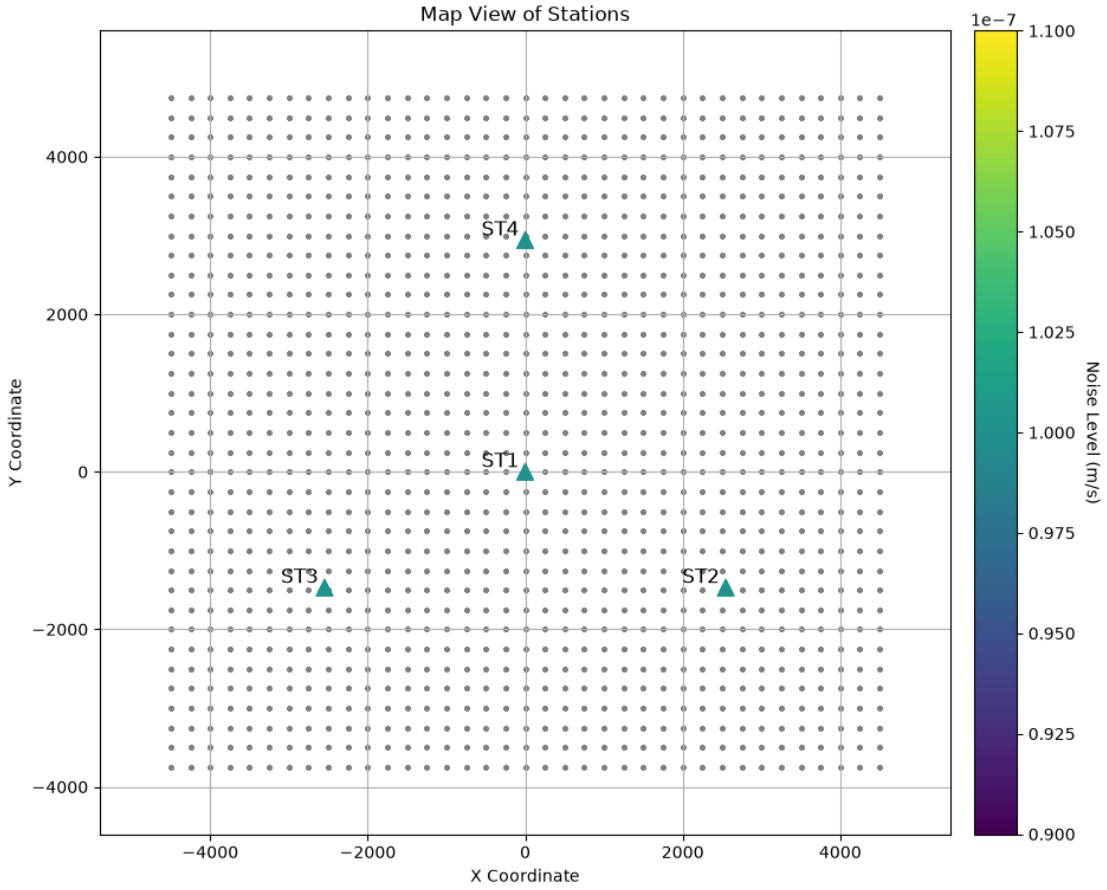
```
gz = np.arange(1500, 5000 + 250, 250)

# Generate grid points
grid_points = pndmod.utils.generate_grid(x=gx,y=gy,z=gz)
print("Number of grid points:",grid_points.shape[0])
print("grid_points.shape:",grid_points.shape)
```

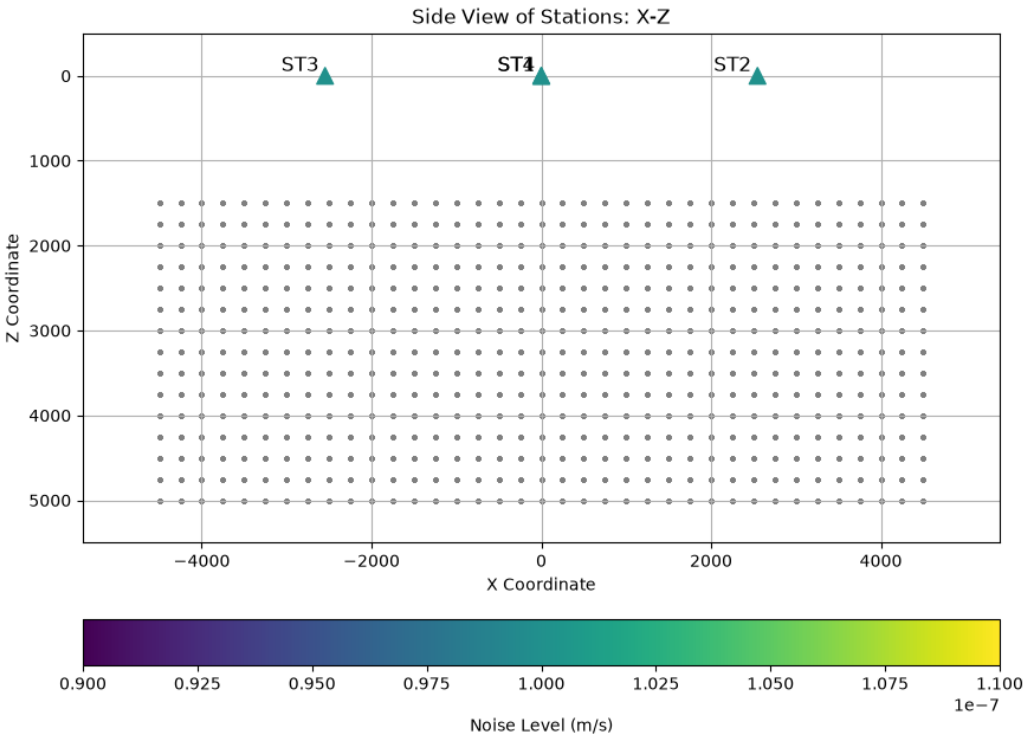
```
Number of grid points: 19425
grid_points.shape: (19425, 3)
```

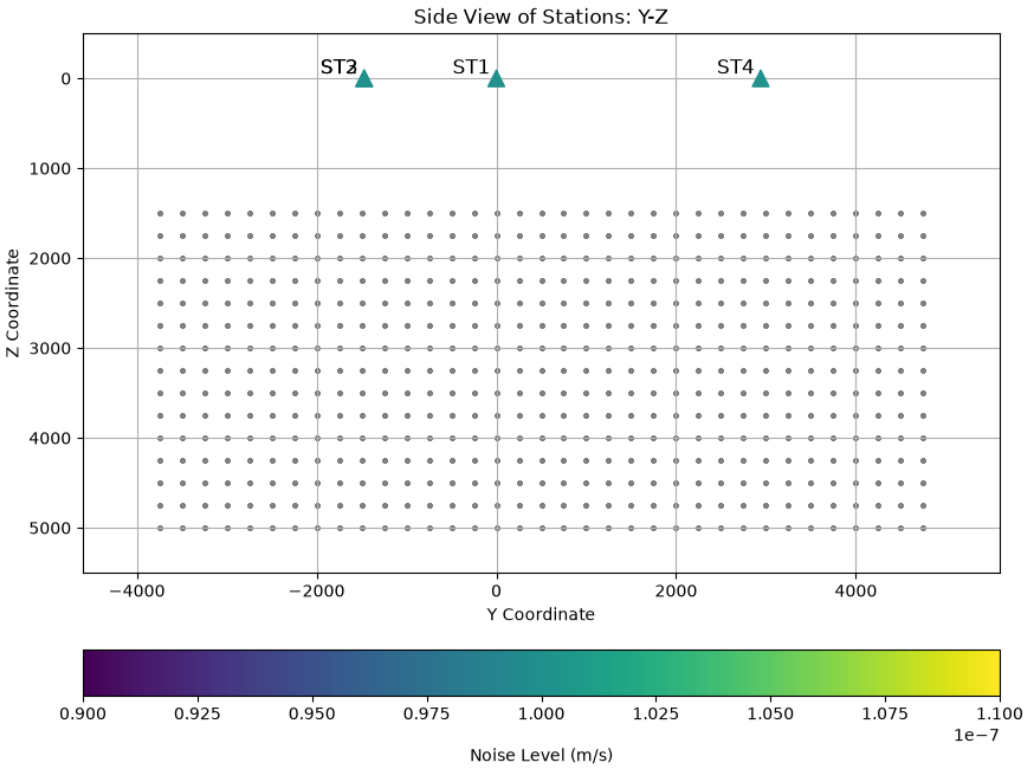
Plot geometry and imaging grid

```
_ = pndvis.geometry.plot_stations_view(df=geometry_df, grid_points=grid_points, grid_
↪marker_size = 1)
_ = pndvis.geometry.plot_stations_view(df=geometry_df, grid_points=grid_points, grid_
↪marker_size = 1, view = 'xz')
_ = pndvis.geometry.plot_stations_view(df=geometry_df, grid_points=grid_points, grid_
↪marker_size = 1, view = 'yz')
```


Map View of Stations
ST4
ST1
ST3
ST2
Y Coordinate
X Coordinate
Noise Level (m/s)
1e−7
1.100
1.075
1.050
1.025
1.000
0.975
0.950
0.925
0.900
4000
2000
0
−2000
−4000
−4000
−2000
0
2000
4000

•


Side View of Stations: X-Z
ST3
ST4
ST2
0
1000
2000
3000
4000
5000
Z Coordinate
−4000
−2000
0
2000
4000
X Coordinate
0.900
0.925
0.950
0.975
1.000
1.025
1.050
1.075
1.100
1e−7
Noise Level (m/s)



-

3.2.3 Prepare to compute magnitude sensitivity

```
# Specify source parameters
frequency = 10 # Hz

# Minimum S/N for detection on individual channel
min_SNR_p = 2
min_SNR_s = 2

# Minimum stations on which an event must be detected
min_stations_p = 3
min_stations_s = 3

# Use free surface correction
#free_surface = False
```

(continues on next page)

(continued from previous page)

```
free_surface = True

# Save parameters
params = pndmod.classes.DetectionParameters(
    f_p=frequency,
    f_s=frequency,
    min_SNR_p=min_SNR_p,
    min_SNR_s=min_SNR_s,
    min_stations_p=min_stations_p,
    min_stations_s=min_stations_s,
    free_surface=free_surface
)

# Define slices to use
slice_X = 0
slice_Y = 0
slice_Z = 2000

# Get slice planes
X_plane_points, X_plane_indices = pndmod.utils.get_slice_coordinates(grid_points=grid_
↪points,slice_dimension=0,slice_coordinate=slice_X)
Y_plane_points, Y_plane_indices = pndmod.utils.get_slice_coordinates(grid_points=grid_
↪points,slice_dimension=1,slice_coordinate=slice_Y)
Z_plane_points, Z_plane_indices = pndmod.utils.get_slice_coordinates(grid_points=grid_
↪points,slice_dimension=2,slice_coordinate=slice_Z)
```

3.2.4 Compute magnitude sensitivity

```
# Compute magnitude sensitivity for P wave
print("Computing magnitude sensitivity for P wave...")
start_time = time()
X_slice_mag_sens_stations_p = pndmod.core.get_mag_sensitivity(grid_coords=X_plane_
↪points,
                                                              geometry_df=geometry_df,
                                                              velocity_df=velocity_df,
                                                              params=params,
                                                                   wave_mode='P')
Y_slice_mag_sens_stations_p = pndmod.core.get_mag_sensitivity(grid_coords=Y_plane_
↪points,
                                                              geometry_df=geometry_df,
                                                              velocity_df=velocity_df,
                                                              params=params,
                                                              wave_mode='P')
Z_slice_mag_sens_stations_p = pndmod.core.get_mag_sensitivity(grid_coords=Z_plane_
↪points,
                                                              geometry_df=geometry_df,
                                                              velocity_df=velocity_df,
                                                              params=params,
                                                              wave_mode='P')
end_time = time()
print(f"Computation time: {end_time - start_time} seconds")
```

(continues on next page)

(continued from previous page)

```
# Compute magnitude sensitivity for S wave
print("Computing magnitude sensitivity for S wave...")
start_time = time()
X_slice_mag_sens_stations_s = pndmod.core.get_mag_sensitivity(grid_coords=X_plane_
↪points,
                                                              geometry_df=geometry_df,
                                                              velocity_df=velocity_df,
                                                              params=params,
                                                              wave_mode='S')
Y_slice_mag_sens_stations_s = pndmod.core.get_mag_sensitivity(grid_coords=Y_plane_
↪points,
                                                              geometry_df=geometry_df,
                                                              velocity_df=velocity_df,
                                                              params=params,
                                                              wave_mode='S')
Z_slice_mag_sens_stations_s = pndmod.core.get_mag_sensitivity(grid_coords=Z_plane_
↪points,
                                                              geometry_df=geometry_df,
                                                              velocity_df=velocity_df,
                                                              params=params,
                                                              wave_mode='S')
end_time = time()
print(f"Computation time: {end_time - start_time} seconds")


# Compute magnitude sensitivity for the whole grid for P and S waves
print("Computing magnitude sensitivity for both P and S waves...")
start_time = time()
X_slice_mag_sens_stations_ps = pndmod.core.get_mag_sensitivity(grid_coords=X_plane_
↪points,
                                                               geometry_df=geometry_
↪df,
                                                               velocity_df=velocity_
↪df,
                                                               params=params,
                                                               wave_mode='PS')
Y_slice_mag_sens_stations_ps = pndmod.core.get_mag_sensitivity(grid_coords=Y_plane_
↪points,
                                                               geometry_df=geometry_
↪df,
                                                               velocity_df=velocity_
↪df,
                                                               params=params,
                                                               wave_mode='PS')
Z_slice_mag_sens_stations_ps = pndmod.core.get_mag_sensitivity(grid_coords=Z_plane_
↪points,
                                                               geometry_df=geometry_
↪df,
                                                               velocity_df=velocity_
↪df,
                                                               params=params,
```

(continues on next page)

(continued from previous page)

```
                                                        wave_mode='PS')

end_time = time()
print(f"Computation time: {end_time - start_time} seconds")
```

```
Computing magnitude sensitivity for P wave...
Computation time: 0.0036513805389404297 seconds
Computing magnitude sensitivity for S wave...
Computation time: 0.0035088062286376953 seconds
Computing magnitude sensitivity for both P and S waves...
Computation time: 0.0040013790130615234 seconds
```

3.2.5 Plot magnitude sensitivity

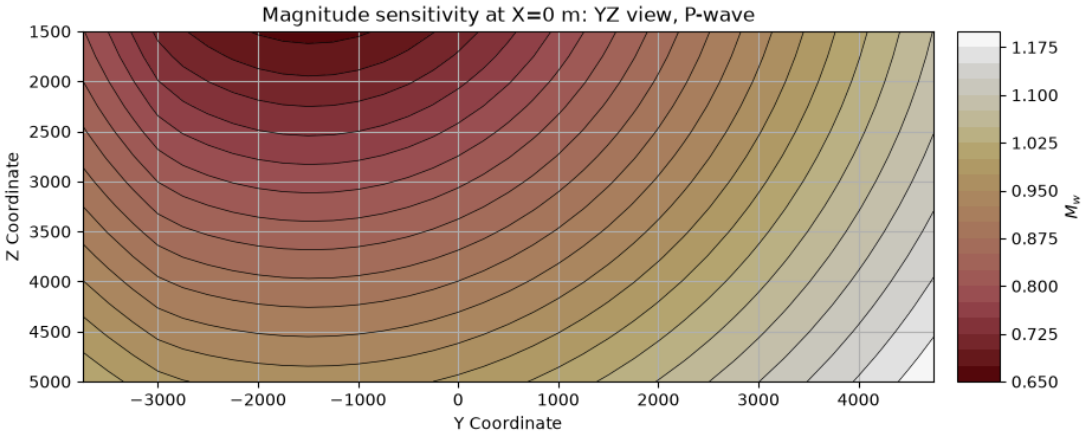
Plot magnitude sensitivity for P wave

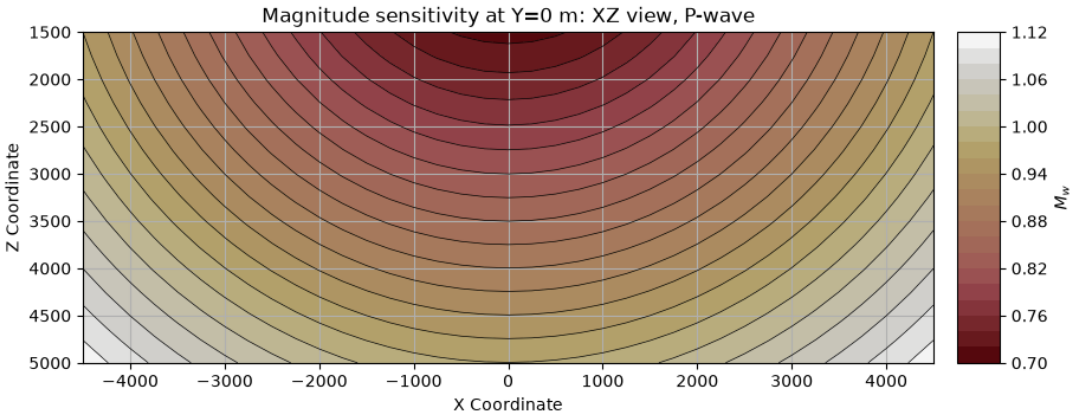
```
clb_range = None
plt_levels = None

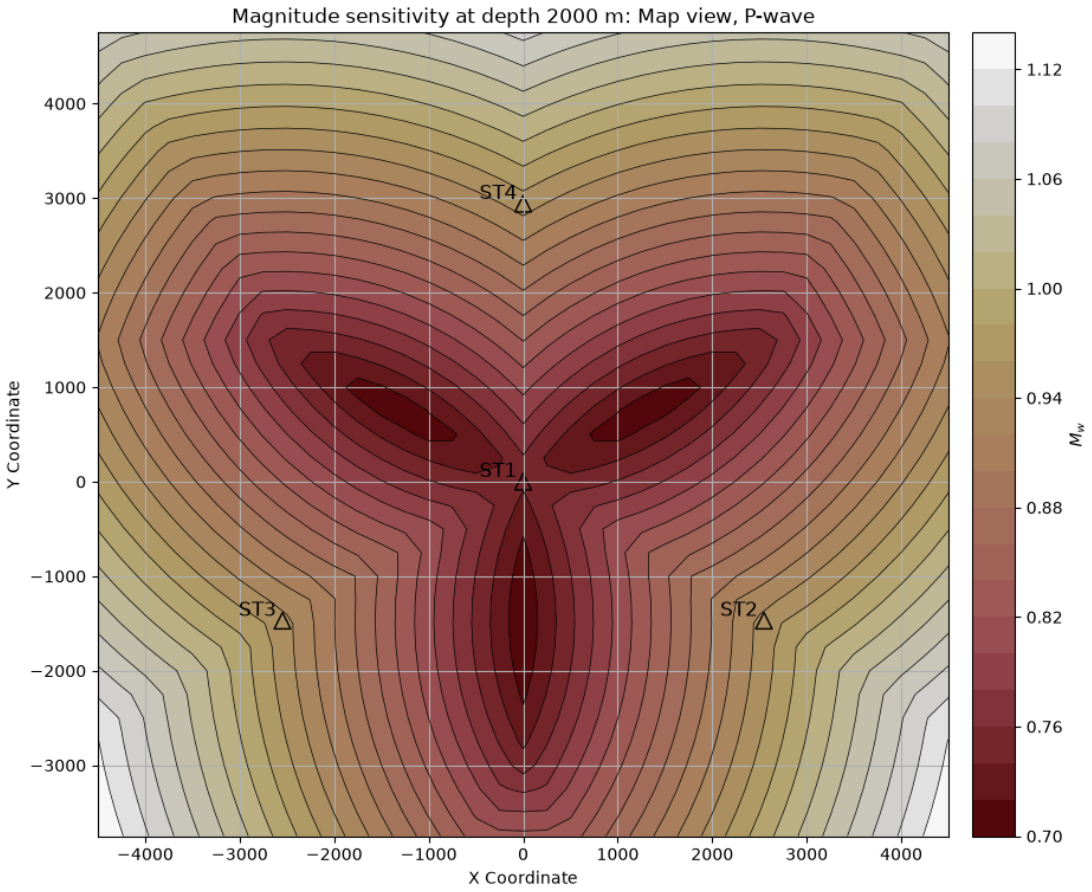
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=X_slice_mag_sens_stations_p,␣
↪yi=gy, zi=gz,
                                                    plot_title="Magnitude␣
↪sensitivity at X=" + str(slice_X) + " m: YZ view, P-wave",
                                                    clb_title="$M_w$",
                                                    clb_range=clb_range,
                                                    plt_levels=plt_levels)

_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=Y_slice_mag_sens_stations_p,␣
↪xi=gx, zi=gz,
                                                    plot_title="Magnitude␣
↪sensitivity at Y=" + str(slice_Y) + " m: XZ view, P-wave",
                                                    clb_title="$M_w$",
                                                    clb_range=clb_range,
                                                    plt_levels=plt_levels)

_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=Z_slice_mag_sens_stations_p,␣
↪xi=gx, yi=gy, geometry_df=geometry_df,
                                                    plot_title="Magnitude␣
↪sensitivity at depth " + str(slice_Z) + " m: Map view, P-wave",
                                                    clb_title="$M_w$",
                                                    clb_range=clb_range,
                                                    plt_levels=plt_levels)
```


Magnitude sensitivity at X=0 m: YZ view, P-wave
Z Coordinate
Y Coordinate
1500
2000
2500
3000
3500
4000
4500
5000
−3000
−2000
−1000
0
1000
2000
3000
4000
1.175
1.100
1.025
0.950
0.875
0.800
0.725
0.650
M_w


Magnitude sensitivity at Y=0 m: XZ view, P-wave
Z Coordinate
X Coordinate
1500
2000
2500
3000
3500
4000
4500
5000
−4000
−3000
−2000
−1000
0
1000
2000
3000
4000
1.12
1.06
1.00
0.94
0.88
0.82
0.76
0.70
M_w



-

Plot magnitude sensitivity for S wave

```
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=X_slice_mag_sens_stations_s,␣
↪yi=gy, zi=gz,
                                              plot_title="Magnitude␣
↪sensitivity at X=" + str(slice_X) + " m: YZ view, S-wave",
                                              clb_title="$M_w$",
                                              clb_range=clb_range,
                                              plt_levels=plt_levels)

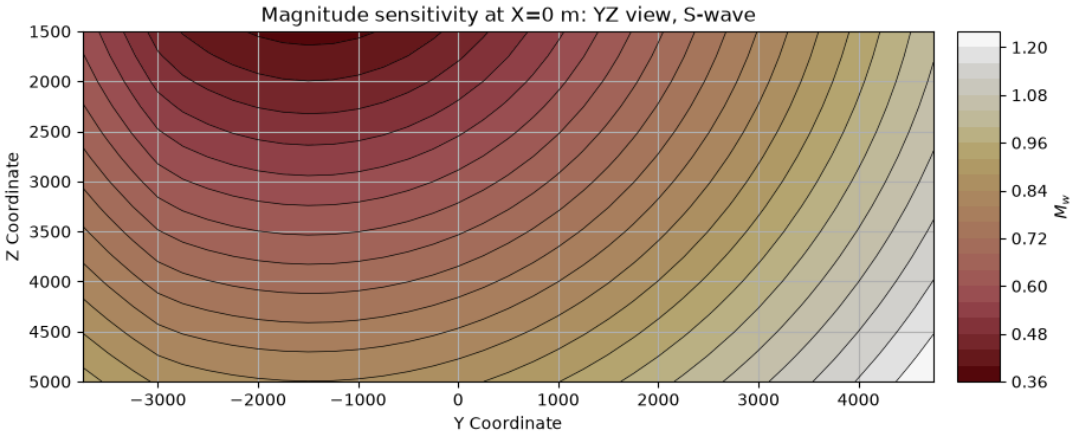
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=Y_slice_mag_sens_stations_s,␣
↪xi=gx, zi=gz,
                                              plot_title="Magnitude␣
↪sensitivity at Y=" + str(slice_Y) + " m: XZ view, S-wave",
                                              clb_title="$M_w$",
```

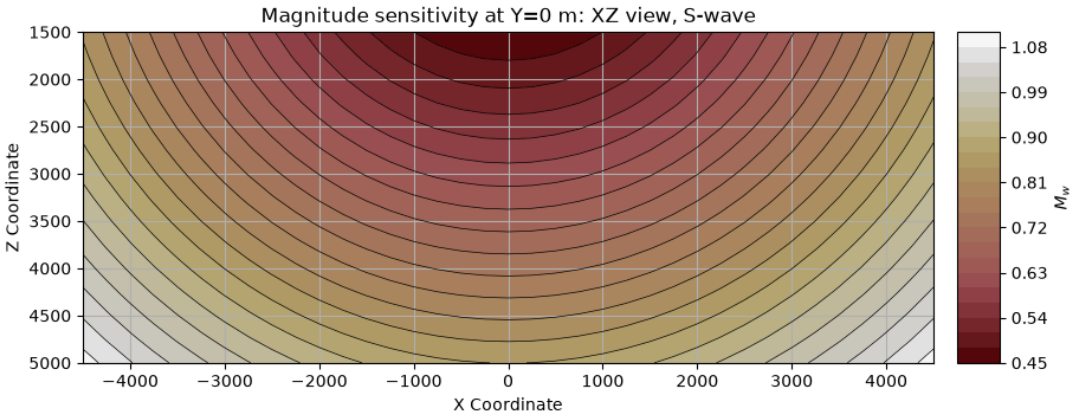
(continues on next page)

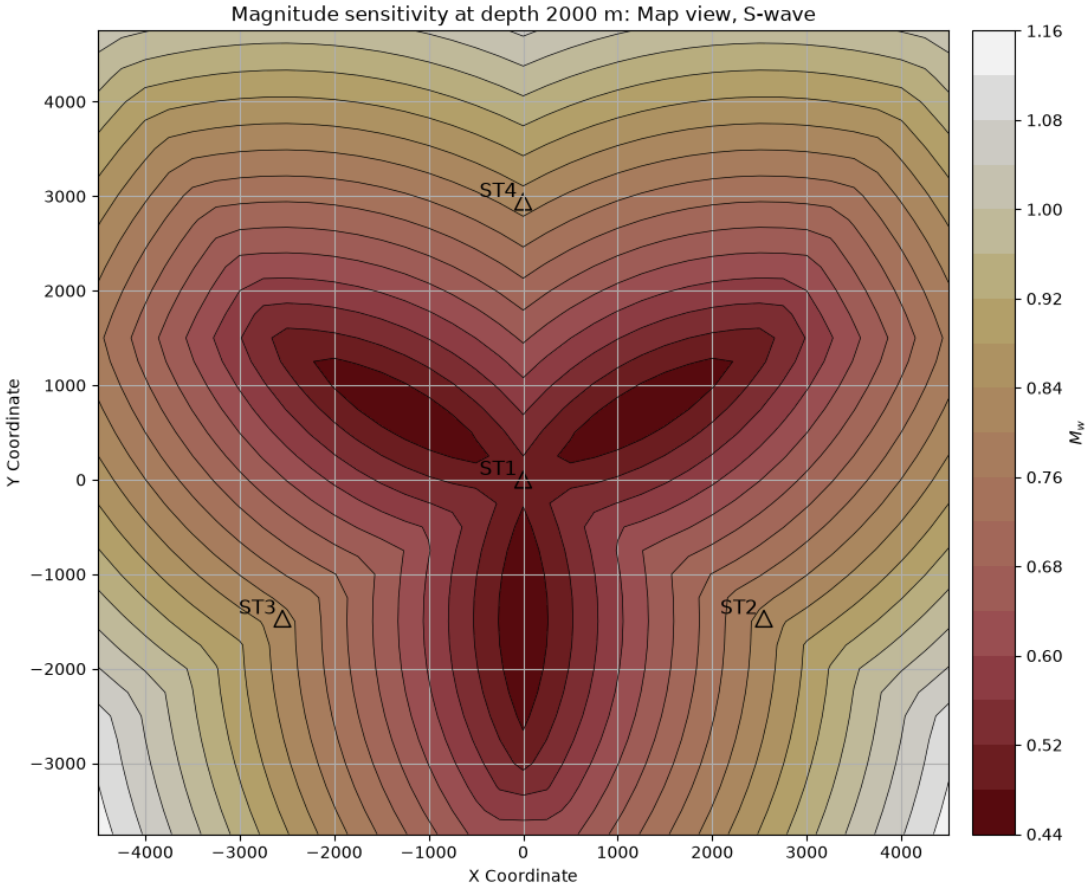
(continued from previous page)

```
                                                    clb_range=clb_range,
                                                    plt_levels=plt_levels)

_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=Z_slice_mag_sens_stations_s,␣
↪xi=gx, yi=gy, geometry_df=geometry_df,
                                                    plot_title="Magnitude␣
↪sensitivity at depth " + str(slice_Z) + " m: Map view, S-wave",
                                                    clb_title="$M_w$",
                                                    clb_range=clb_range,
                                                    plt_levels=plt_levels)
```




Magnitude sensitivity at Y=0 m: XZ view, S-wave
Z Coordinate
X Coordinate
1500
2000
2500
3000
3500
4000
4500
5000
−4000
−3000
−2000
−1000
0
1000
2000
3000
4000
1.08
0.99
0.90
0.81
0.72
0.63
0.54
0.45
M_w



-

Plot magnitude sensitivity for P and S waves

```
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=X_slice_mag_sens_stations_ps,␣
↪yi=gy, zi=gz,
                                                plot_title="Magnitude␣
↪sensitivity at X=" + str(slice_X) + " m: YZ view, P- and S-waves",
                                                clb_title="$M_w$",
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)

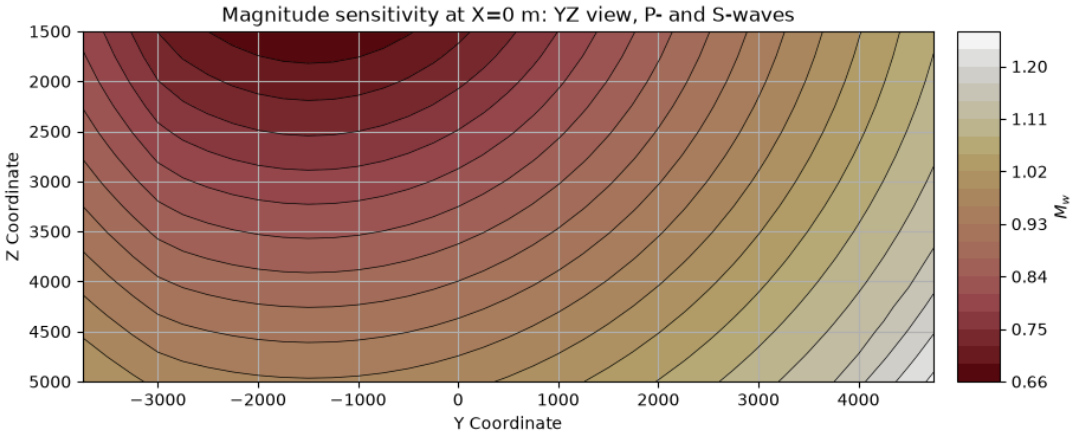
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=Y_slice_mag_sens_stations_ps,␣
↪xi=gx, zi=gz,
                                                plot_title="Magnitude␣
↪sensitivity at Y=" + str(slice_Y) + " m: XZ view, P- and S-waves",
                                                clb_title="$M_w$",
```

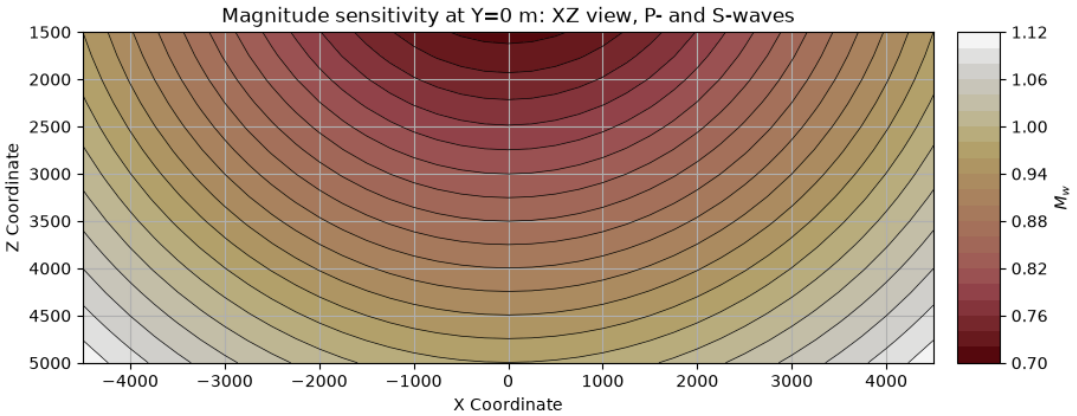
(continues on next page)

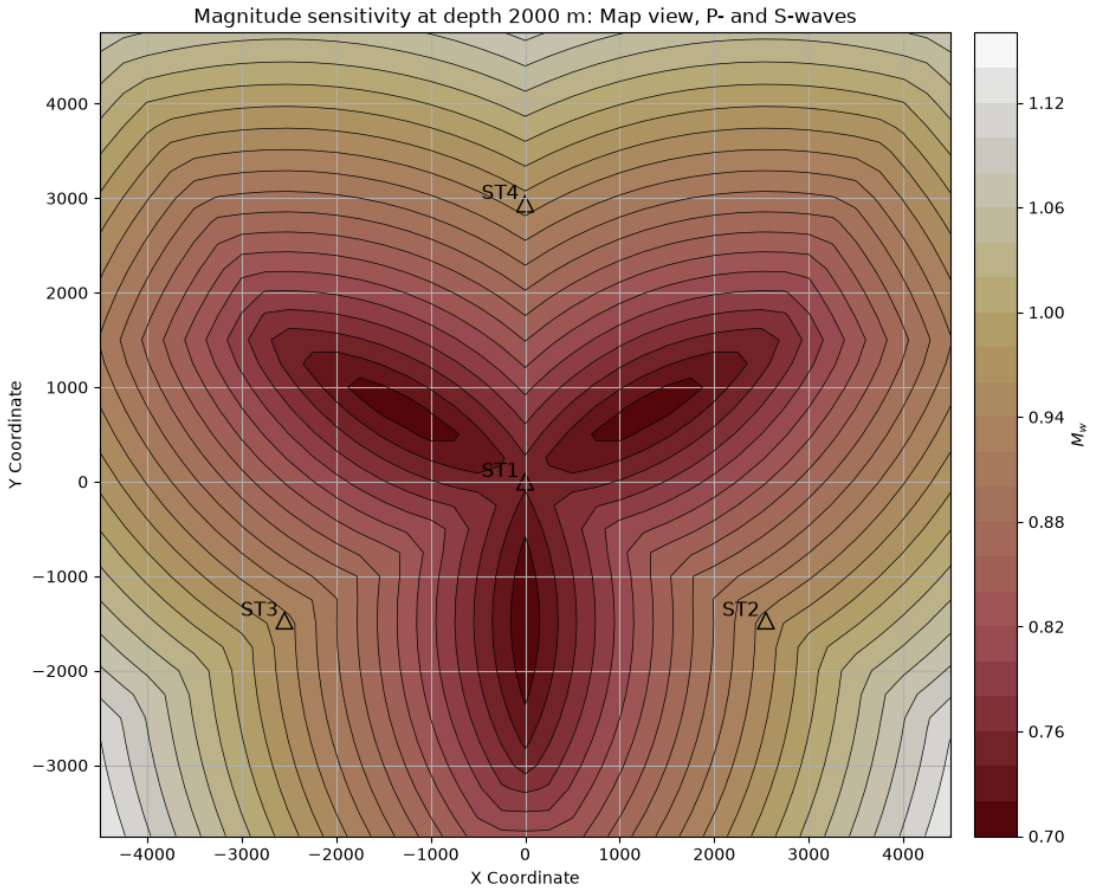
(continued from previous page)

```
                                                    clb_range=clb_range,
                                                    plt_levels=plt_levels)

_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=Z_slice_mag_sens_stations_ps,␣
↪xi=gx, yi=gy, geometry_df=geometry_df,
                                                    plot_title="Magnitude␣
↪sensitivity at depth " + str(slice_Z) + " m: Map view, P- and S-waves",
                                                    clb_title="$M_w$",
                                                    clb_range=clb_range,
                                                    plt_levels=plt_levels)
```




Magnitude sensitivity at Y=0 m: XZ view, P- and S-waves
Z Coordinate
X Coordinate
1500
2000
2500
3000
3500
4000
4500
5000
−4000
−3000
−2000
−1000
0
1000
2000
3000
4000
1.12
1.06
1.00
0.94
0.88
0.82
0.76
0.70
M_w



-

Total running time of the script: (0 minutes 1.883 seconds)

3.3 03.1 Magnitude sensitivity for homogeneous model: test case 3

This is an example of computing magnitude sensitivity for a homogeneous model.

Test case 3:

- Geometry: simple DAS geometry with 1 vertical borehole
- Velocity model: homogeneous, Vp = 4300 m/s, Vp/Vs = 1.73, Qp = 100, Qs = 50, rho = 2300 kg/m^3
- Processing: magnitude sensitivity computation with and without directionality correction (grid slices).

3.3.1 Load all necessary packages

```
import pynetdesign.visualization as pndvis
import pynetdesign.modelling as pndmod
import numpy as np
from time import time

# sphinx_gallery_thumbnail_number = -2
```

3.3.2 Input

Read geometry

```
geometry_file_path = '../data/homo/test_03/TMPL_net_geometryA.txt'
geometry_df = pndmod.io.read_geometry(geometry_file_path)

print(geometry_df)
print(geometry_df.attrs)
```

```
       Y    X       Z    NoiseLevel
0    0.0  0.0    -0.0  5.000000e-09
1    0.0  0.0    10.0  5.000000e-09
2    0.0  0.0    20.0  5.000000e-09
3    0.0  0.0    30.0  5.000000e-09
4    0.0  0.0    40.0  5.000000e-09
..   ...  ...     ...           ...
96   0.0  0.0   960.0  5.000000e-09
97   0.0  0.0   970.0  5.000000e-09
98   0.0  0.0   980.0  5.000000e-09
99   0.0  0.0   990.0  5.000000e-09
100  0.0  0.0  1000.0  5.000000e-09

[101 rows x 4 columns]
{'file_path': '../data/homo/test_03/TMPL_net_geometryA.txt', 'original_coord_units':
↪'m', 'coordinate_system': 'local', 'noise_type': 'strain', 'noise_unit': 'ε',
↪'gauge_length': 10.0, '_is_combined': False}
```

Read velocity model

```
velocity_file_path = '../data/homo/test_03/TMPL_velocity_model.txt'
velocity_df = pndmod.io.read_velocity(velocity_file_path)

print(velocity_df)
print(velocity_df.attrs)
```

```
   Depth    Vp  VpVsRatio   Qp  Qs   Rho   Ep   Dt   Gm
0  40000  4300       1.73  100  50  2300  0.0  0.0  0.0
{'model_type': 'homo', 'model_anisotropy': 'isotropic', 'parameter_descriptions': {
↪'Depth': 'Depth', 'Vp': 'P-wave velocity', 'Vs': 'S-wave velocity', 'VpVsRatio':
↪'Vp/Vs', 'Qp': 'P-wave quality factor', 'Qs': 'S-wave quality factor', 'Rho':
↪'density', 'Ep': 'Thomsen epsilon parameter', 'Dt': 'Thomsen delta parameter', 'Gm
↪': 'Thomsen gamma parameter'}, 'parameter_units': {'Depth': 'm', 'Vp': 'm/s', 'Vs':
```

(continues on next page)

(continued from previous page)

```
↪'m/s', 'VpVsRatio': '', 'Qp': '', 'Qs': '', 'Rho': 'kg/m^3', 'Ep': '', 'Dt': '', 'Gm
↪': ''}}
```

Set imaging grid parameters

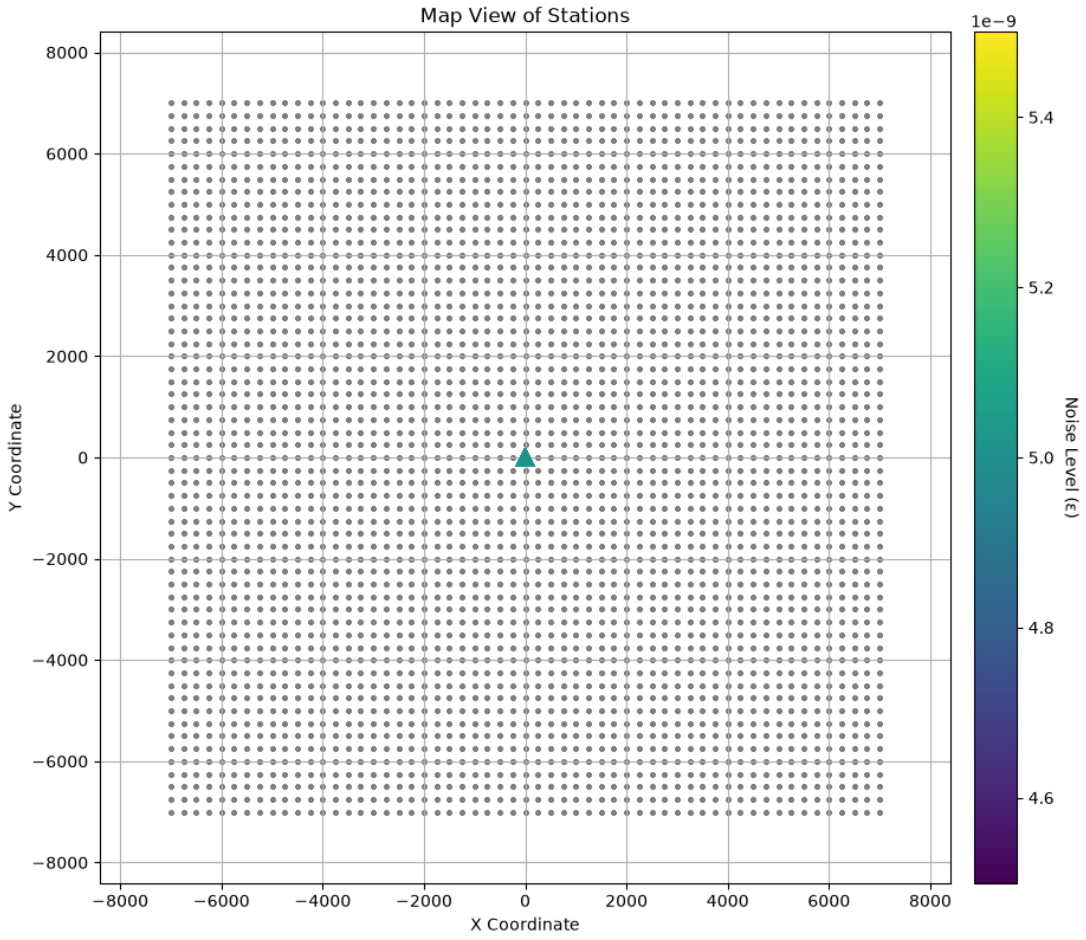
```
# Imaging grid parameters
gx = np.arange(-7000, 7000 + 250, 250)
gy = np.arange(-7000, 7000 + 250, 250)
gz = np.arange(250, 5000 + 250, 250)

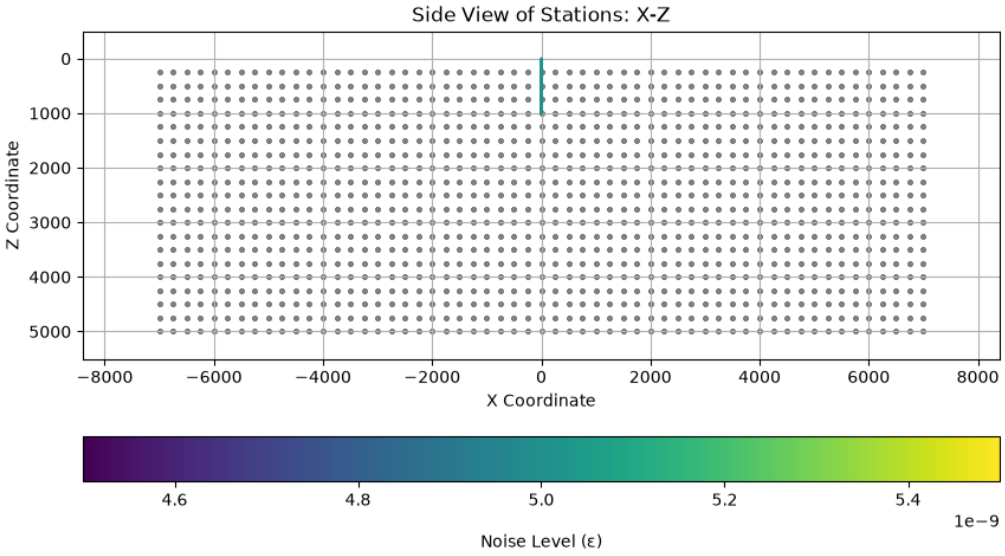
# Generate grid points
grid_points = pndmod.utils.generate_grid(x=gx,y=gy,z=gz)
print("Number of grid points:",grid_points.shape[0])
print("Number of grid points in x:",gx.size)
print("Number of grid points in y:",gy.size)
print("Number of grid points in z:",gz.size)
print("grid_points.shape:",grid_points.shape)
```

```
Number of grid points: 64980
Number of grid points in x: 57
Number of grid points in y: 57
Number of grid points in z: 20
grid_points.shape: (64980, 3)
```

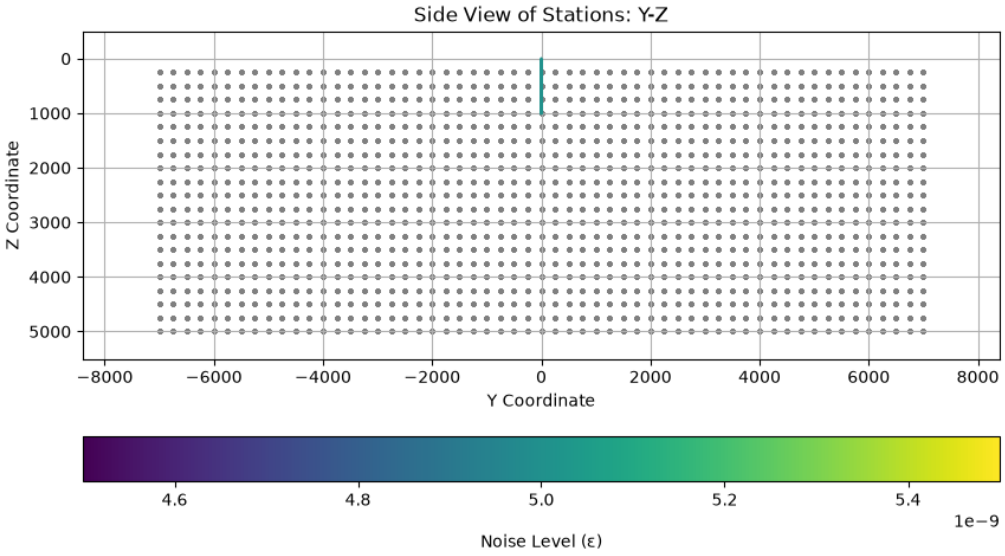
Plot geometry and imaging grid

```
_ = pndvis.geometry.plot_stations_view(geometry_df,
                                       grid_points=grid_points,
                                       grid_marker_size = 1,
                                       plot_names=False,
                                       station_marker_size=100)
_ = pndvis.geometry.plot_stations_view(geometry_df,
                                       view = 'xz',
                                       grid_points=grid_points,
                                       grid_marker_size = 1,
                                       plot_names=False,
                                       station_marker_size=1)
_ = pndvis.geometry.plot_stations_view(geometry_df,
                                       view = 'yz',
                                       grid_points=grid_points,
                                       grid_marker_size = 1,
                                       plot_names=False,
                                       station_marker_size=1)
```


Map View of Stations
1e−9
5.4
5.2
5.0
4.8
4.6
Noise Level (ε)
8000
6000
4000
2000
0
−2000
−4000
−6000
−8000
Y Coordinate
−8000
−6000
−4000
−2000
0
2000
4000
6000
8000
X Coordinate


Side View of Stations: X-Z
Z Coordinate
0
1000
2000
3000
4000
5000
−8000
−6000
−4000
−2000
0
2000
4000
6000
8000
X Coordinate
4.6
4.8
5.0
5.2
5.4
1e−9
Noise Level (ε)

-



-

3.3.3 Prepare to compute magnitude sensitivity

```
# Specify source parameters
frequency = 10 # Hz

# Minimum S/N for detection on individual channel
min_SNR_p = 2
min_SNR_s = 2

# Minimum stations on which an event must be detected
min_stations_p = 3
min_stations_s = 3

# Save parameters
params = pndmod.classes.DetectionParameters(
```

(continues on next page)

(continued from previous page)

```
    f_p=frequency,
    f_s=frequency,
    min_SNR_p=min_SNR_p,
    min_SNR_s=min_SNR_s,
    min_stations_p=min_stations_p,
    min_stations_s=min_stations_s,
)

# Define slices to use
slice_X = 250
slice_Y = 250
slice_Z = 2000

# Get slice planes
X_plane_points, X_plane_indices = pndmod.utils.get_slice_coordinates(grid_points=grid_
↪points,slice_dimension=0,slice_coordinate=slice_X)
Y_plane_points, Y_plane_indices = pndmod.utils.get_slice_coordinates(grid_points=grid_
↪points,slice_dimension=1,slice_coordinate=slice_Y)
Z_plane_points, Z_plane_indices = pndmod.utils.get_slice_coordinates(grid_points=grid_
↪points,slice_dimension=2,slice_coordinate=slice_Z)
```

3.3.4 Retrieve minimum displacement amplitudes

```
start_time = time()
min_amps_p, min_amps_s = pndmod.utils.retrieve_min_amps(geometry_df = geometry_df,
                                                        f_p = frequency,
                                                        f_s = frequency,
                                                        SNR_p = min_SNR_p,
                                                        SNR_s = min_SNR_s,
                                                        sample_interval=geometry_df.
↪attrs['gauge_length'])
end_time = time()
print(f"Computation time: {end_time - start_time} seconds")
```

```
Computation time: 0.00017786026000976562 seconds
```

3.3.5 Compute magnitude sensitivity without directionality correction

```
# Compute magnnitude sensitivity for the planes for P
print("Computing magnitude sensitivity on planes for P wave...")
start_time = time()
X_slice_mag_sens_stations_p = pndmod.homo.mag_sensitivity_grid(grid_coords=X_plane_
↪points,
                                                               velocity_df=velocity_
↪df,
                                                               geometry_df=geometry_
↪df,
                                                               min_amps_p=min_amps_p,
                                                               min_stations_p=min_
↪stations_p,
```

(continues on next page)

(continued from previous page)

```
                                                                    f_p=frequency,
                                                                    wave_mode='P')
Y_slice_mag_sens_stations_p = pndmod.homo.mag_sensitivity_grid(grid_coords=Y_plane_
↪points,
                                                                    velocity_df=velocity_
↪df,
                                                                    geometry_df=geometry_
↪df,
                                                                    min_amps_p=min_amps_p,
                                                                    min_stations_p=min_
↪stations_p,
                                                                    f_p=frequency,
                                                                    wave_mode='P')
Z_slice_mag_sens_stations_p = pndmod.homo.mag_sensitivity_grid(grid_coords=Z_plane_
↪points,
                                                                    velocity_df=velocity_
↪df,
                                                                    geometry_df=geometry_
↪df,
                                                                    min_amps_p=min_amps_p,
                                                                    min_stations_p=min_
↪stations_p,
                                                                    f_p=frequency,
                                                                    wave_mode='P')
end_time = time()
print(f"Computation time: {end_time - start_time} seconds")


# Compute magnnitude sensitivity for the planes for S
print("Computing magnitude sensitivity on planes for S wave...")
start_time = time()
X_slice_mag_sens_stations_s = pndmod.homo.mag_sensitivity_grid(grid_coords=X_plane_
↪points,
                                                                    velocity_df=velocity_
↪df,
                                                                    geometry_df=geometry_
↪df,
                                                                    min_amps_s=min_amps_s,
                                                                    min_stations_s=min_
↪stations_s,
                                                                    f_s=frequency,
                                                                    wave_mode='S')
Y_slice_mag_sens_stations_s = pndmod.homo.mag_sensitivity_grid(grid_coords=Y_plane_
↪points,
                                                                    velocity_df=velocity_
↪df,
                                                                    geometry_df=geometry_
↪df,
                                                                    min_amps_s=min_amps_s,
                                                                    min_stations_s=min_
↪stations_s,
                                                                    f_s=frequency,
```

(continues on next page)

(continued from previous page)

```
                                                                    wave_mode='S')
Z_slice_mag_sens_stations_s = pndmod.homo.mag_sensitivity_grid(grid_coords=Z_plane_
↪points,
                                                                    velocity_df=velocity_
↪df,
                                                                    geometry_df=geometry_
↪df,
                                                                    min_amps_s=min_amps_s,
                                                                    min_stations_s=min_
↪stations_s,
                                                                    f_s=frequency,
                                                                    wave_mode='S')
end_time = time()
print(f"Computation time: {end_time - start_time} seconds")

# Compute magnnitude sensitivity for the planes for P and S
print("Computing magnitude sensitivity on planes for P and S waves...")
start_time = time()
X_slice_mag_sens_stations_ps = pndmod.homo.mag_sensitivity_grid(grid_coords=X_plane_
↪points,
                                                                     velocity_df=velocity_
↪df,
                                                                     geometry_df=geometry_
↪df,
                                                                     min_amps_p=min_amps_p,
                                                                     min_amps_s=min_amps_s,
                                                                     min_stations_p=min_
↪stations_p,
                                                                     min_stations_s=min_
↪stations_s,
                                                                     f_p=frequency,
                                                                     f_s=frequency,
                                                                     wave_mode='PS')

Y_slice_mag_sens_stations_ps = pndmod.homo.mag_sensitivity_grid(grid_coords=Y_plane_
↪points,
                                                                     velocity_df=velocity_
↪df,
                                                                     geometry_df=geometry_
↪df,
                                                                     min_amps_p=min_amps_p,
                                                                     min_amps_s=min_amps_s,
                                                                     min_stations_p=min_
↪stations_p,
                                                                     min_stations_s=min_
↪stations_s,
                                                                     f_p=frequency,
                                                                     f_s=frequency,
                                                                     wave_mode='PS')

Z_slice_mag_sens_stations_ps = pndmod.homo.mag_sensitivity_grid(grid_coords=Z_plane_
↪points,
```

(continues on next page)

(continued from previous page)

```
                                                                    velocity_df=velocity_
↪df,
                                                                    geometry_df=geometry_
↪df,
                                                                    min_amps_p=min_amps_p,
                                                                    min_amps_s=min_amps_s,
                                                                    min_stations_p=min_
↪stations_p,
                                                                    min_stations_s=min_
↪stations_s,
                                                                    f_p=frequency,
                                                                    f_s=frequency,
                                                                    wave_mode='PS')
end_time = time()
print(f"Computation time: {end_time - start_time} seconds")
```

```
Computing magnitude sensitivity on planes for P wave...
Computation time: 0.02816009521484375 seconds
Computing magnitude sensitivity on planes for S wave...
Computation time: 0.21951770782470703 seconds
Computing magnitude sensitivity on planes for P and S waves...
Computation time: 0.24031352996826172 seconds
```

3.3.6 Plot magnitude sensitivity without directionality correction

Plot magnitude sensitivity for P wave

```
clb_range = None
plt_levels = None

_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=X_slice_mag_sens_stations_p,␣
↪yi=gy, zi=gz,
                                              plot_title="Magnitude sensitivity at X=
↪" + str(slice_X) + " m: YZ view, P-wave",
                                              clb_title="$M_w$",
                                              clb_range=clb_range,
                                              plt_levels=plt_levels,
                                              station_marker_size=1,
                                              plot_names=False)

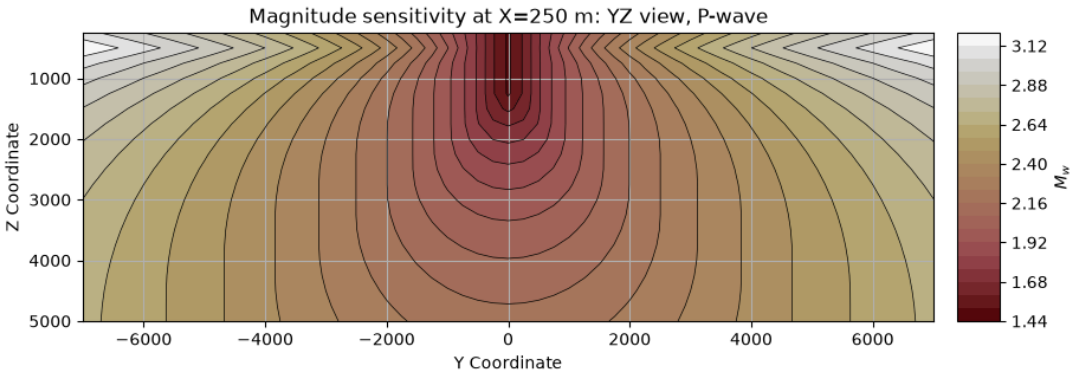
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=Y_slice_mag_sens_stations_p,␣
↪xi=gx, zi=gz,
                                              plot_title="Magnitude sensitivity at Y=
↪" + str(slice_Y) + " m: XZ view, P-wave",
                                              clb_title="$M_w$",
                                              clb_range=clb_range,
                                              plt_levels=plt_levels,
                                              station_marker_size=1,
                                              plot_names=False)

_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=Z_slice_mag_sens_stations_p,␣
```

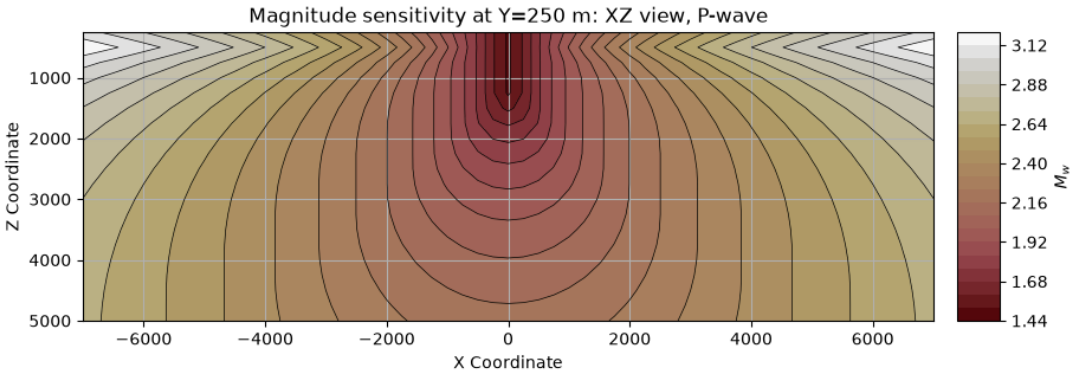
(continues on next page)

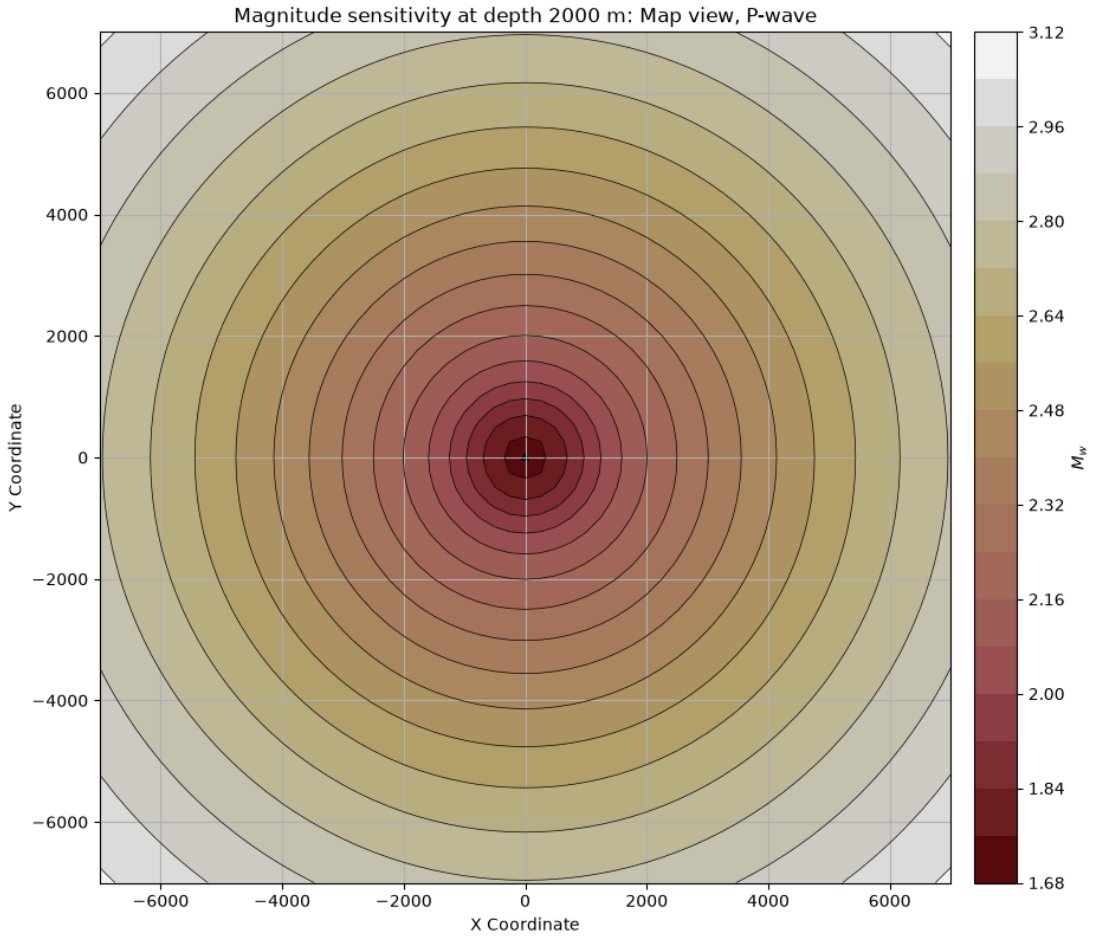
(continued from previous page)

```
↪xi=gx, yi=gy, geometry_df=geometry_df,
                                                    plot_title="Magnitude sensitivity at␣
↪depth " + str(slice_Z) + " m: Map view, P-wave",
                                                    clb_title="$M_w$",
                                                    clb_range=clb_range,
                                                    plt_levels=plt_levels,
                                                    station_marker_size=10,
                                                    plot_names=False)
```



•


Magnitude sensitivity at Y=250 m: XZ view, P-wave
Z Coordinate
1000
2000
3000
4000
5000
−6000
−4000
−2000
0
2000
4000
6000
X Coordinate
3.12
2.88
2.64
2.40
2.16
1.92
1.68
1.44
M_w



-

Plot magnitude sensitivity for S wave

```
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=X_slice_mag_sens_stations_s,␣
↪yi=gy, zi=gz,
                                              plot_title="Magnitude sensitivity at X=
↪" + str(slice_X) + " m: YZ view, S-wave",
                                              clb_title="$M_w$",
                                              clb_range=clb_range,
                                              plt_levels=plt_levels,
                                              station_marker_size=1,
                                              plot_names=False)

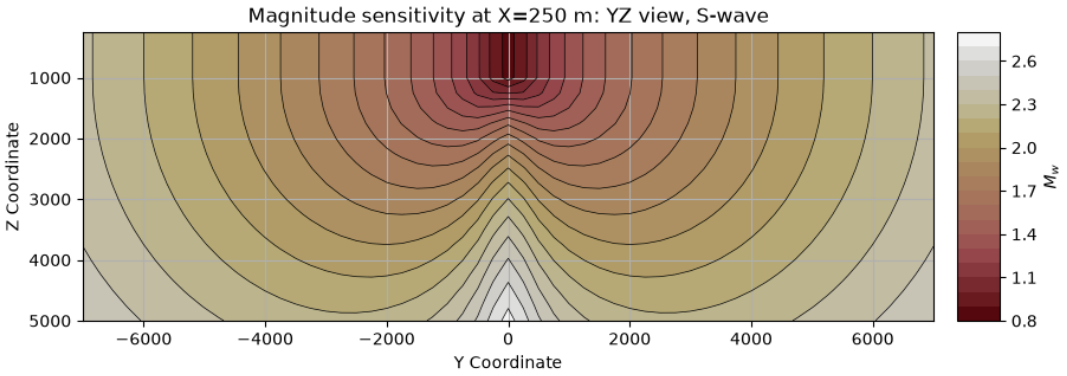
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=Y_slice_mag_sens_stations_s,␣
↪xi=gx, zi=gz,
                                              plot_title="Magnitude sensitivity at Y=
```

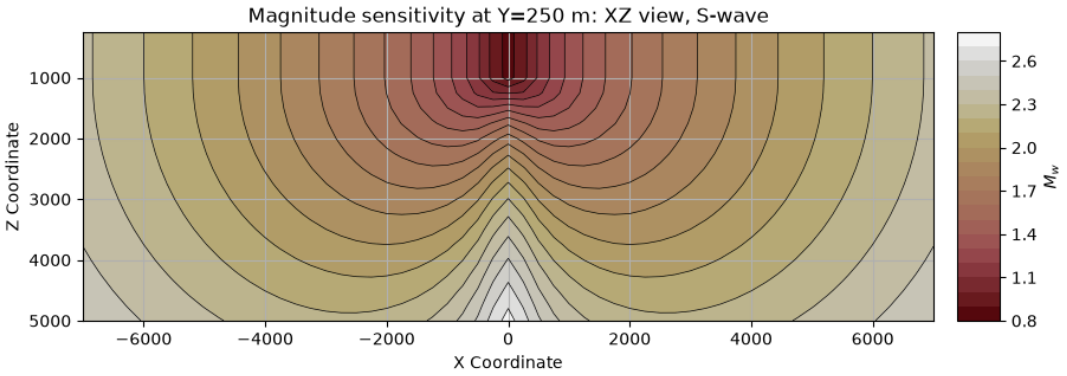
(continues on next page)

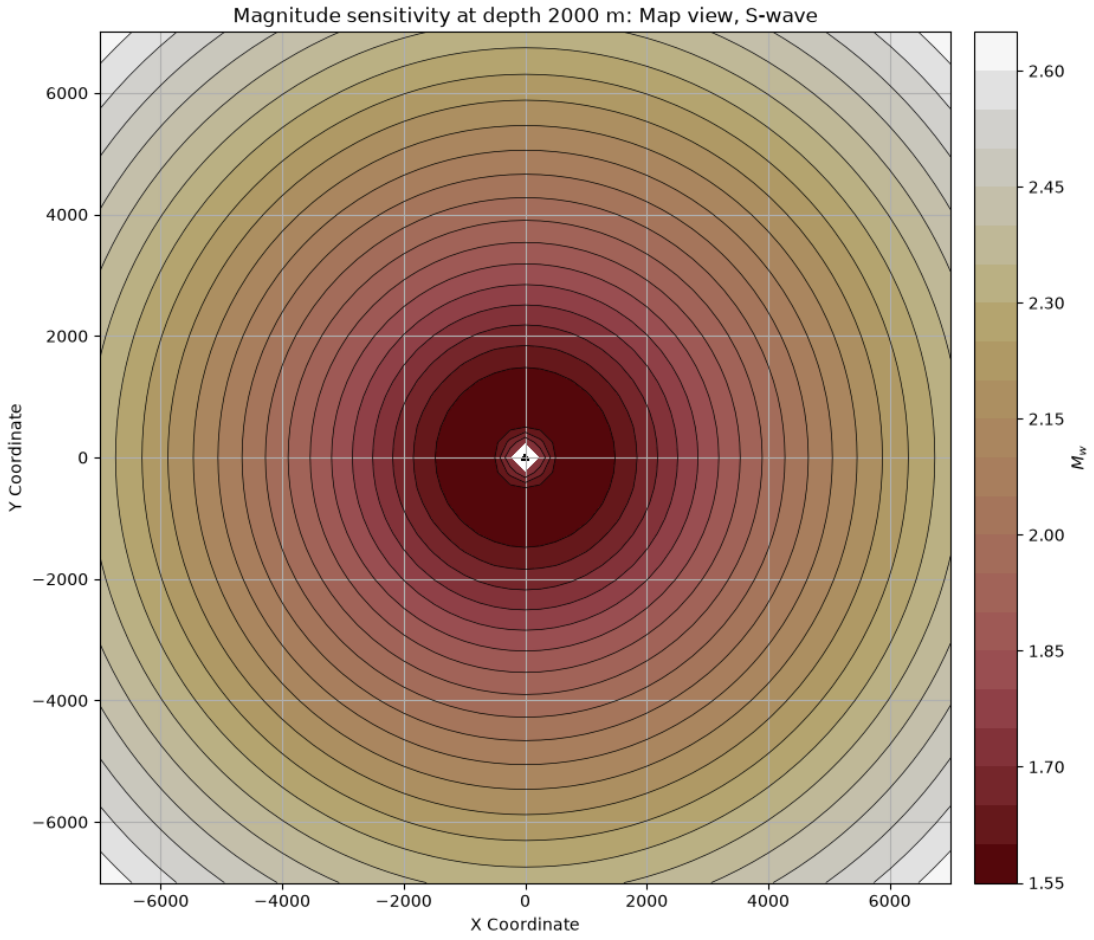
(continued from previous page)

```
↪" + str(slice_Y) + " m: XZ view, S-wave",
                                                clb_title="$M_w$",
                                                clb_range=clb_range,
                                                plt_levels=plt_levels,
                                                station_marker_size=1,
                                                plot_names=False)

_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=Z_slice_mag_sens_stations_s,␣
↪xi=gx, yi=gy, geometry_df=geometry_df,
                                                plot_title="Magnitude sensitivity at␣
↪depth " + str(slice_Z) + " m: Map view, S-wave",
                                                clb_title="$M_w$",
                                                clb_range=clb_range,
                                                plt_levels=plt_levels,
                                                station_marker_size=10,
                                                plot_names=False)
```


Magnitude sensitivity at X=250 m: YZ view, S-wave
Z Coordinate
1000
2000
3000
4000
5000
−6000
−4000
−2000
0
2000
4000
6000
Y Coordinate
2.6
2.3
2.0
1.7
1.4
1.1
0.8
M_w


Magnitude sensitivity at Y=250 m: XZ view, S-wave
Z Coordinate
1000
2000
3000
4000
5000
−6000
−4000
−2000
0
2000
4000
6000
X Coordinate
2.6
2.3
2.0
1.7
1.4
1.1
0.8
M_w



-

Plot magnitude sensitivity for P and S waves

```
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=X_slice_mag_sens_stations_ps,␣
↪yi=gy, zi=gz,
                                              plot_title="Magnitude sensitivity at X=
↪" + str(slice_X) + " m: YZ view, P- and S-waves",
                                              clb_title="$M_w$",
                                              clb_range=clb_range,
                                              plt_levels=plt_levels,
                                              station_marker_size=1,
                                              plot_names=False)

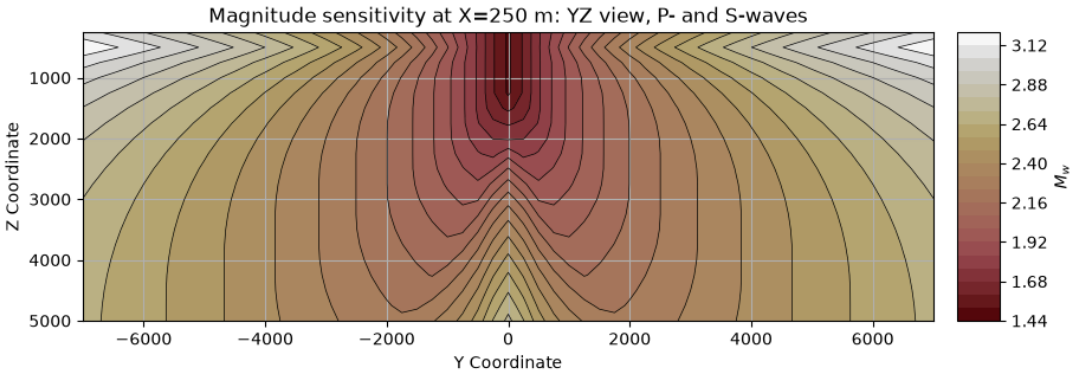
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=Y_slice_mag_sens_stations_ps,␣
↪xi=gx, zi=gz,
                                              plot_title="Magnitude sensitivity at Y=
```

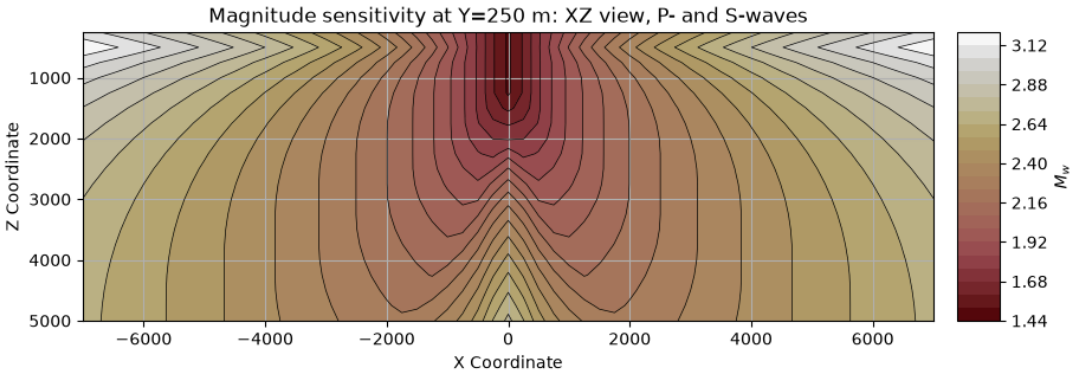
(continues on next page)

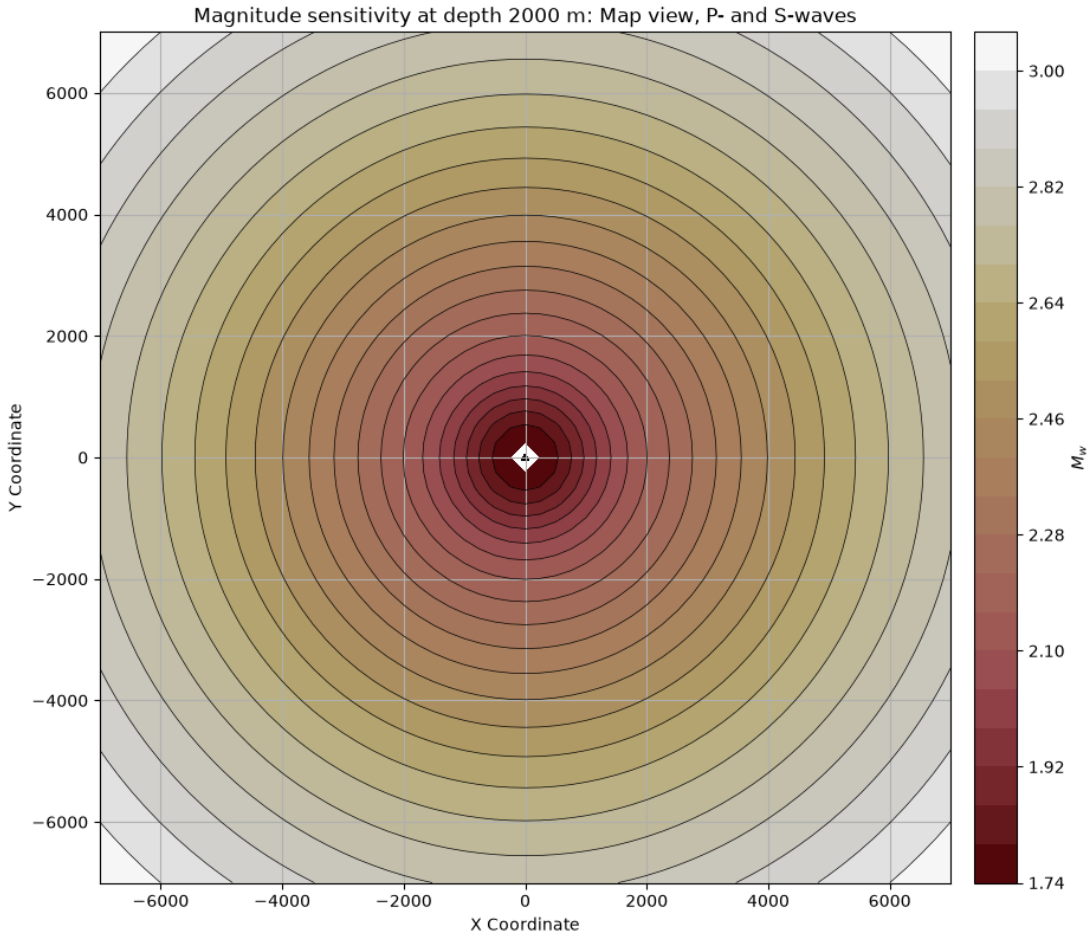
(continued from previous page)

```
↪" + str(slice_Y) + " m: XZ view, P- and S-waves",
                                                clb_title="$M_w$",
                                                clb_range=clb_range,
                                                plt_levels=plt_levels,
                                                station_marker_size=1,
                                                plot_names=False)

_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=Z_slice_mag_sens_stations_ps,␣
↪xi=gx, yi=gy, geometry_df=geometry_df,
                                                plot_title="Magnitude sensitivity at␣
↪depth " + str(slice_Z) + " m: Map view, P- and S-waves",
                                                clb_title="$M_w$",
                                                clb_range=clb_range,
                                                plt_levels=plt_levels,
                                                station_marker_size=10,
                                                plot_names=False)
```


Magnitude sensitivity at X=250 m: YZ view, P- and S-waves
Z Coordinate
1000
2000
3000
4000
5000
−6000
−4000
−2000
0
2000
4000
6000
Y Coordinate
3.12
2.88
2.64
2.40
2.16
1.92
1.68
1.44
M_w


Magnitude sensitivity at Y=250 m: XZ view, P- and S-waves
Z Coordinate
1000
2000
3000
4000
5000
−6000
−4000
−2000
0
2000
4000
6000
X Coordinate
3.12
2.88
2.64
2.40
2.16
1.92
1.68
1.44
M_w



-

3.3.7 Compute magnitude sensitivity with directionality correction

```
# Compute magnnitude sensitivity for the planes for P
print("Computing magnitude sensitivity on planes with correction for station␣
↪directionality for P wave...")
start_time = time()
X_slice_mag_sens_stations_st_dir_p = pndmod.core.get_mag_sensitivity(grid_coords=X_
↪plane_points,
                                                                     geometry_
↪df=geometry_df,
                                                                     velocity_
↪df=velocity_df,
                                                                    ␣
↪params=params,
                                                                     wave_mode='P
```

(continues on next page)

(continued from previous page)

```
↪')
Y_slice_mag_sens_stations_st_dir_p = pndmod.core.get_mag_sensitivity(grid_coords=Y_
↪plane_points,
                                                                     geometry_
↪df=geometry_df,
                                                                     velocity_
↪df=velocity_df,
                                                                    ␣
↪params=params,
                                                                     wave_mode='P
↪')
Z_slice_mag_sens_stations_st_dir_p = pndmod.core.get_mag_sensitivity(grid_coords=Z_
↪plane_points,
                                                                     geometry_
↪df=geometry_df,
                                                                     velocity_
↪df=velocity_df,
                                                                    ␣
↪params=params,
                                                                     wave_mode='P
↪')
end_time = time()
print(f"Computation time: {end_time - start_time} seconds")


# Compute magnnitude sensitivity for the planes for S
print("Computing magnitude sensitivity on planes with correction for station␣
↪directionality for S wave...")
start_time = time()
X_slice_mag_sens_stations_st_dir_s = pndmod.core.get_mag_sensitivity(grid_coords=X_
↪plane_points,
                                                                     geometry_
↪df=geometry_df,
                                                                     velocity_
↪df=velocity_df,
                                                                    ␣
↪params=params,
                                                                     wave_mode='S
↪')
Y_slice_mag_sens_stations_st_dir_s = pndmod.core.get_mag_sensitivity(grid_coords=Y_
↪plane_points,
                                                                     geometry_
↪df=geometry_df,
                                                                     velocity_
↪df=velocity_df,
                                                                    ␣
↪params=params,
                                                                     wave_mode='S
↪')
Z_slice_mag_sens_stations_st_dir_s = pndmod.core.get_mag_sensitivity(grid_coords=Z_
↪plane_points,
                                                                     geometry_
```

(continues on next page)

(continued from previous page)

```
↪df=geometry_df,
                                                                    velocity_
↪df=velocity_df,
                                                                   ␣
↪params=params,
                                                                    wave_mode='S
↪')
end_time = time()
print(f"Computation time: {end_time - start_time} seconds")

# Compute magnnitude sensitivity for the planes for S
print("Computing magnitude sensitivity on planes with correction for station␣
↪directionality for P and S waves...")
start_time = time()
X_slice_mag_sens_stations_st_dir_ps = pndmod.core.get_mag_sensitivity(grid_coords=X_
↪plane_points,
                                                                    geometry_
↪df=geometry_df,
                                                                    velocity_
↪df=velocity_df,
                                                                   ␣
↪params=params,
                                                                    wave_mode=
↪'PS')
Y_slice_mag_sens_stations_st_dir_ps = pndmod.core.get_mag_sensitivity(grid_coords=Y_
↪plane_points,
                                                                    geometry_
↪df=geometry_df,
                                                                    velocity_
↪df=velocity_df,
                                                                   ␣
↪params=params,
                                                                    wave_mode=
↪'PS')
Z_slice_mag_sens_stations_st_dir_ps = pndmod.core.get_mag_sensitivity(grid_coords=Z_
↪plane_points,
                                                                    geometry_
↪df=geometry_df,
                                                                    velocity_
↪df=velocity_df,
                                                                   ␣
↪params=params,
                                                                    wave_mode=
↪'PS')
end_time = time()
print(f"Computation time: {end_time - start_time} seconds")
```

```
Computing magnitude sensitivity on planes with correction for station directionality␣
↪for P wave...
Computation time: 0.028563976287841797 seconds
Computing magnitude sensitivity on planes with correction for station directionality␣
↪for S wave...
```

(continues on next page)

(continued from previous page)

```
Computation time: 0.23161864280700684 seconds
Computing magnitude sensitivity on planes with correction for station directionality␣
↪for P and S waves...
Computation time: 0.2688732147216797 seconds
```

3.3.8 Plot magnitude sensitivity with directionality correction

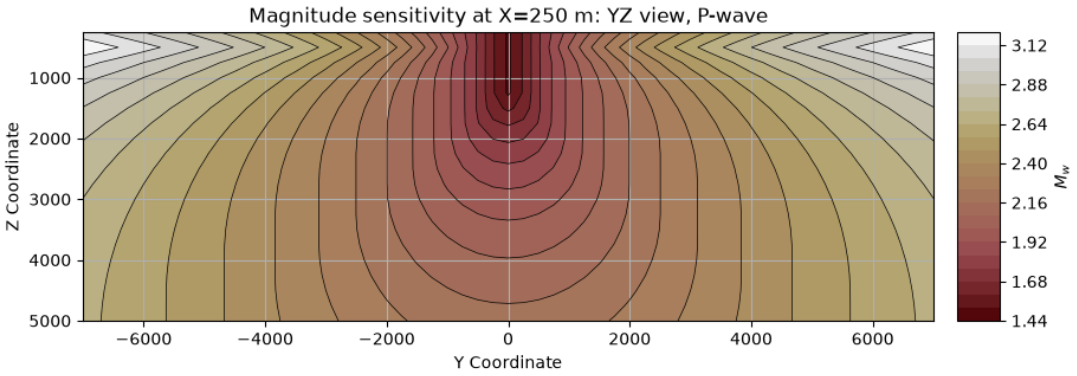
Plot magnitude sensitivity for P wave

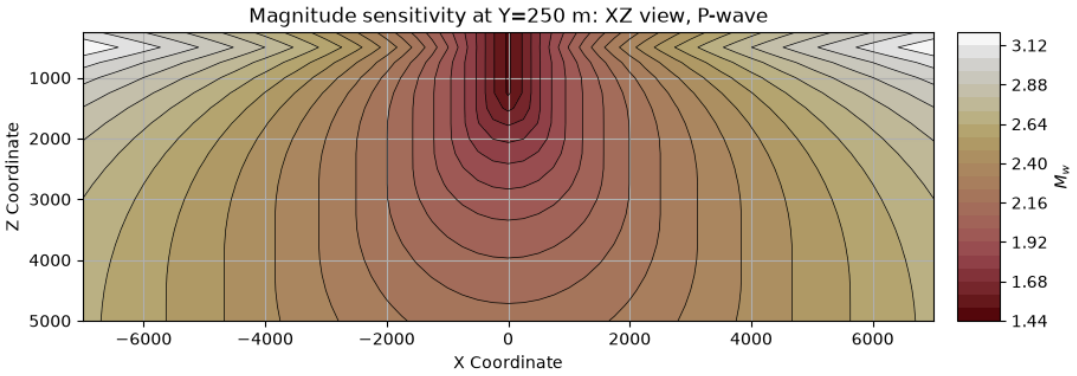
```
clb_range = None
plt_levels = None

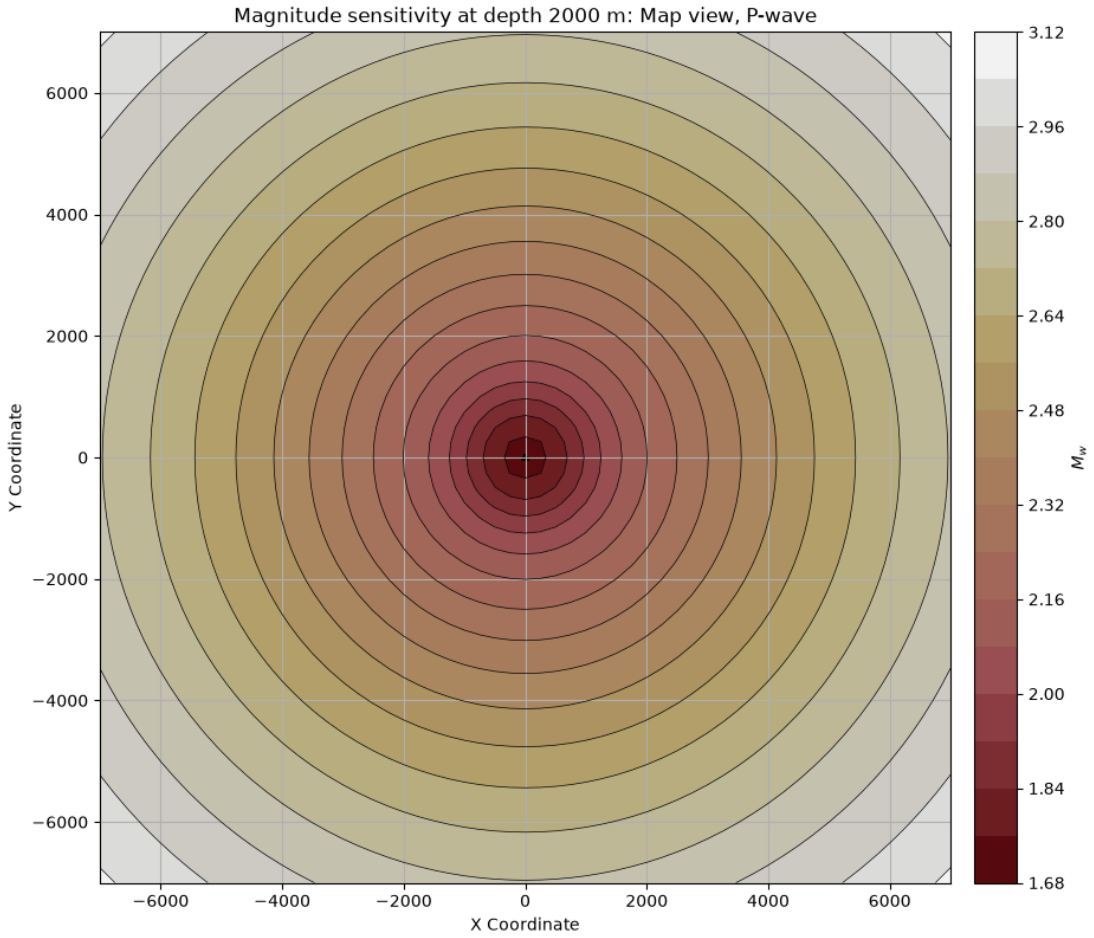
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=X_slice_mag_sens_stations_st_
↪dir_p, yi=gy, zi=gz,
                                              plot_title="Magnitude sensitivity at X=
↪" + str(slice_X) + " m: YZ view, P-wave",
                                              clb_title="$M_w$",
                                              clb_range=clb_range,
                                              plt_levels=plt_levels,
                                              station_marker_size=1,
                                              plot_names=False)

_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=Y_slice_mag_sens_stations_st_
↪dir_p, xi=gx, zi=gz,
                                              plot_title="Magnitude sensitivity at Y=
↪" + str(slice_Y) + " m: XZ view, P-wave",
                                              clb_title="$M_w$",
                                              clb_range=clb_range,
                                              plt_levels=plt_levels,
                                              station_marker_size=1,
                                              plot_names=False)

_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=Z_slice_mag_sens_stations_st_
↪dir_p, xi=gx, yi=gy, geometry_df=geometry_df,
                                              plot_title="Magnitude sensitivity at␣
↪depth " + str(slice_Z) + " m: Map view, P-wave",
                                              clb_title="$M_w$",
                                              clb_range=clb_range,
                                              plt_levels=plt_levels,
                                              station_marker_size=10,
                                              plot_names=False)
```


Magnitude sensitivity at X=250 m: YZ view, P-wave
Z Coordinate
1000
2000
3000
4000
5000
−6000
−4000
−2000
0
2000
4000
6000
Y Coordinate
3.12
2.88
2.64
2.40
2.16
1.92
1.68
1.44
M_w


Magnitude sensitivity at Y=250 m: XZ view, P-wave
Z Coordinate
1000
2000
3000
4000
5000
−6000
−4000
−2000
0
2000
4000
6000
X Coordinate
3.12
2.88
2.64
2.40
2.16
1.92
1.68
1.44
M_w



-

Plot magnitude sensitivity for S wave

```
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=X_slice_mag_sens_stations_st_
↪dir_s, yi=gy, zi=gz,
                                              plot_title="Magnitude sensitivity at X=
↪" + str(slice_X) + " m: YZ view, S-wave",
                                              clb_title="$M_w$",
                                              clb_range=clb_range,
                                              plt_levels=plt_levels,
                                              station_marker_size=1,
                                              plot_names=False)

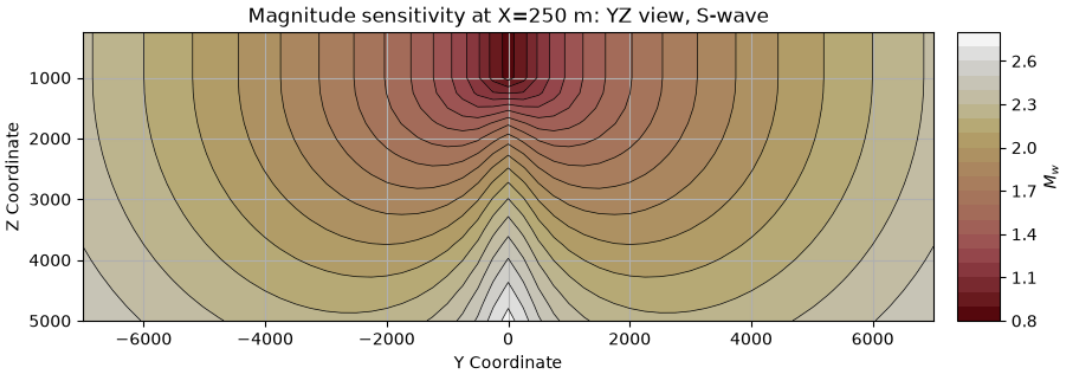
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=Y_slice_mag_sens_stations_st_
↪dir_s, xi=gx, zi=gz,
                                              plot_title="Magnitude sensitivity at Y=
```

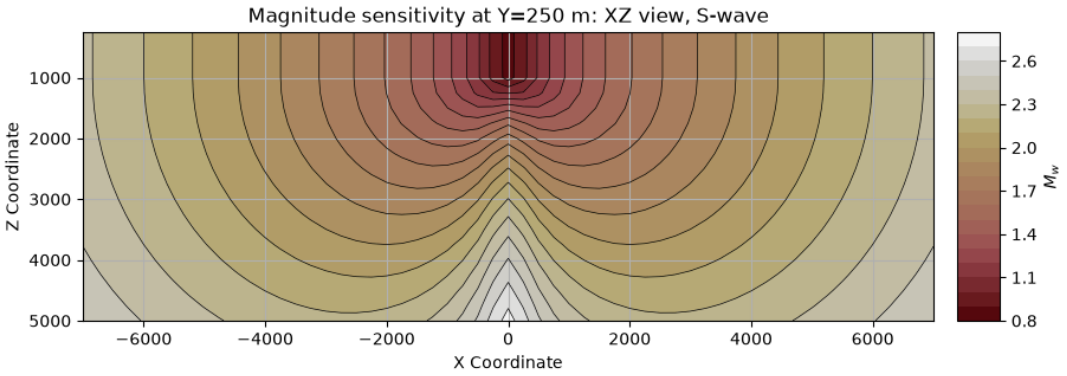
(continues on next page)

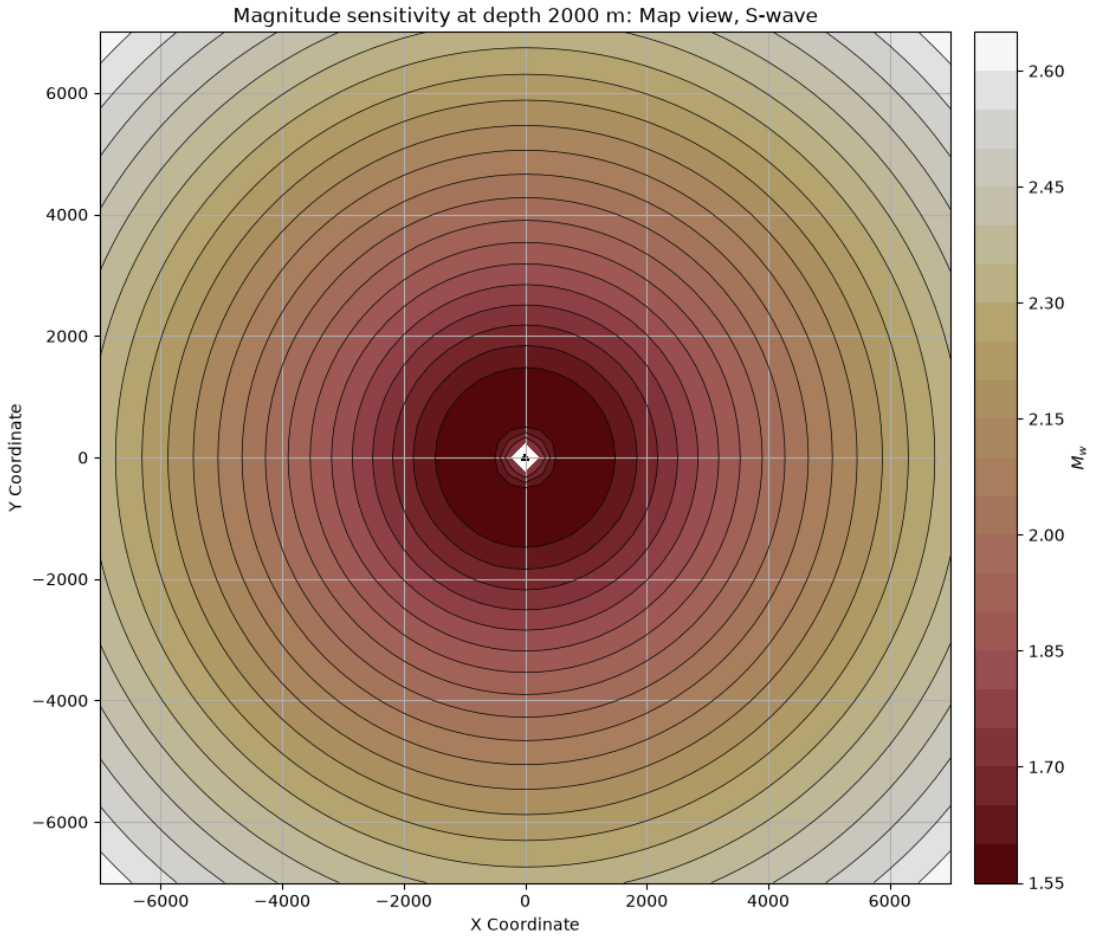
(continued from previous page)

```
↪" + str(slice_Y) + " m: XZ view, S-wave",
                                                clb_title="$M_w$",
                                                clb_range=clb_range,
                                                plt_levels=plt_levels,
                                                station_marker_size=1,
                                                plot_names=False)

_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=Z_slice_mag_sens_stations_st_
↪dir_s, xi=gx, yi=gy, geometry_df=geometry_df,
                                                plot_title="Magnitude sensitivity at␣
↪depth " + str(slice_Z) + " m: Map view, S-wave",
                                                clb_title="$M_w$",
                                                clb_range=clb_range,
                                                plt_levels=plt_levels,
                                                station_marker_size=10,
                                                plot_names=False)
```


Magnitude sensitivity at X=250 m: YZ view, S-wave
Z Coordinate
1000
2000
3000
4000
5000
−6000
−4000
−2000
0
2000
4000
6000
Y Coordinate
2.6
2.3
2.0
1.7
1.4
1.1
0.8
M_w


Magnitude sensitivity at Y=250 m: XZ view, S-wave
Z Coordinate
1000
2000
3000
4000
5000
−6000
−4000
−2000
0
2000
4000
6000
X Coordinate
2.6
2.3
2.0
1.7
1.4
1.1
0.8
M_w



-

Plot magnitude sensitivity for P and S waves

```
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=X_slice_mag_sens_stations_st_
↪dir_ps, yi=gy, zi=gz,
                                              plot_title="Magnitude sensitivity at X=
↪" + str(slice_X) + " m: YZ view, P- and S-waves",
                                              clb_title="$M_w$",
                                              clb_range=clb_range,
                                              plt_levels=plt_levels,
                                              station_marker_size=1,
                                              plot_names=False)

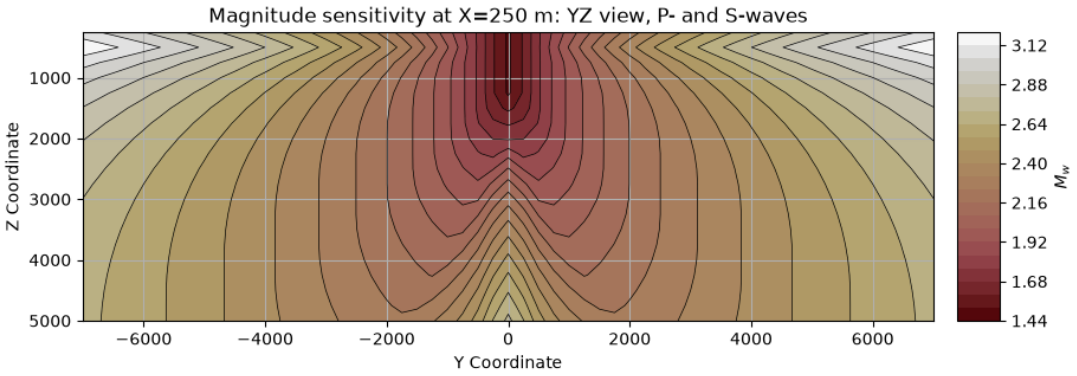
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=Y_slice_mag_sens_stations_st_
↪dir_ps, xi=gx, zi=gz,
                                              plot_title="Magnitude sensitivity at Y=
```

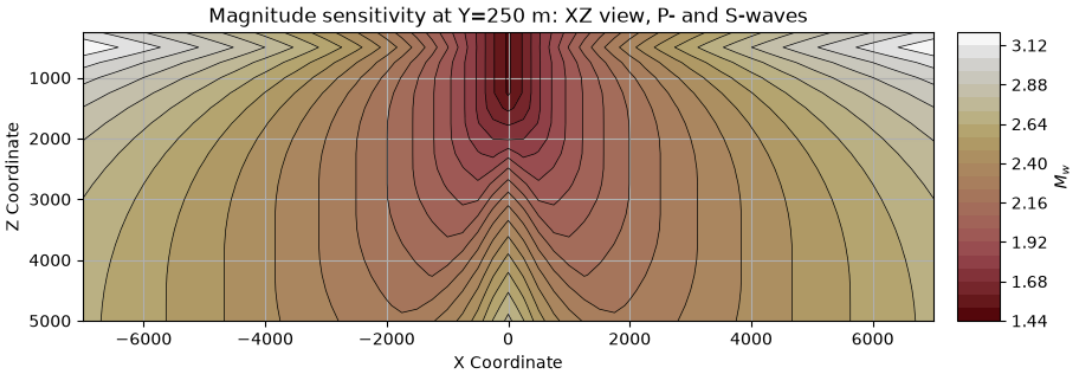
(continues on next page)

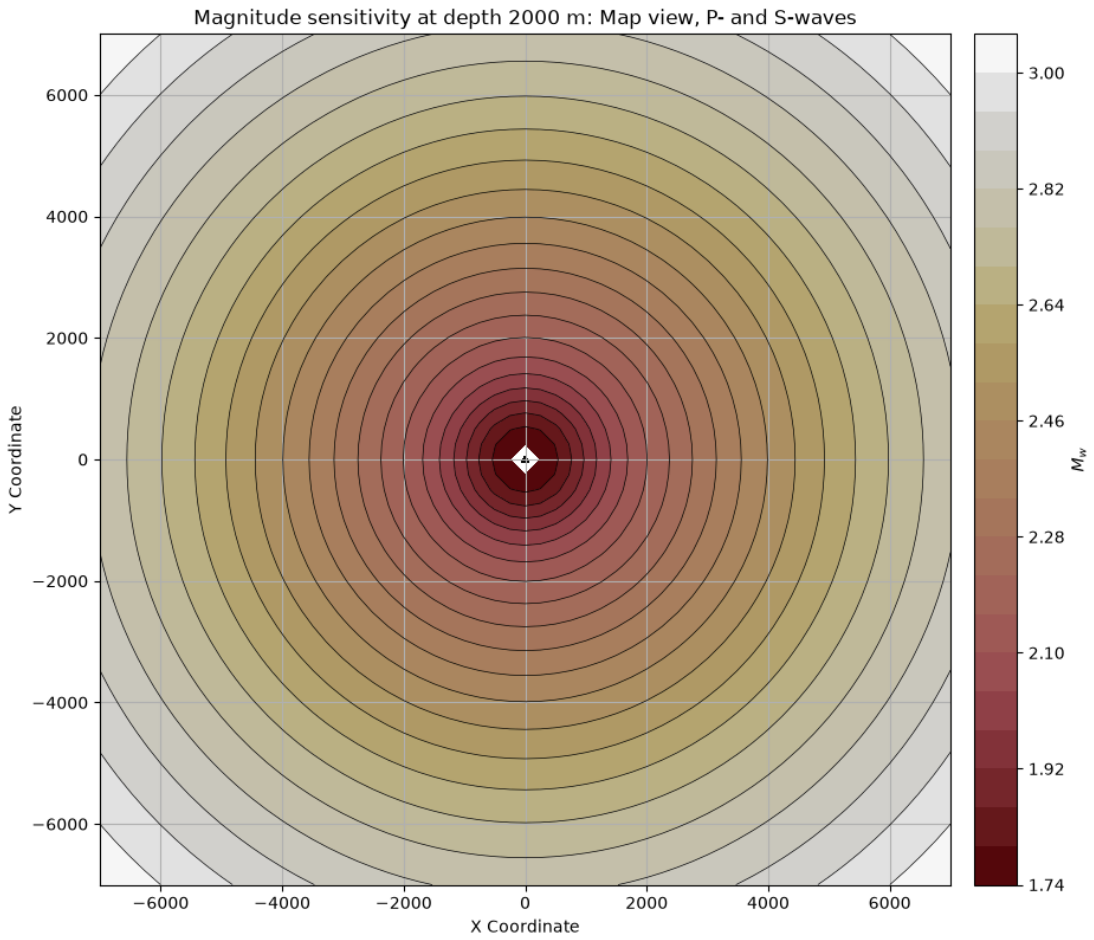
(continued from previous page)

```
↪" + str(slice_Y) + " m: XZ view, P- and S-waves",
                                                clb_title="$M_w$",
                                                clb_range=clb_range,
                                                plt_levels=plt_levels,
                                                station_marker_size=1,
                                                plot_names=False)

_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=Z_slice_mag_sens_stations_st_
↪dir_ps, xi=gx, yi=gy, geometry_df=geometry_df,
                                                plot_title="Magnitude sensitivity at␣
↪depth " + str(slice_Z) + " m: Map view, P- and S-waves",
                                                clb_title="$M_w$",
                                                clb_range=clb_range,
                                                plt_levels=plt_levels,
                                                station_marker_size=10,
                                                plot_names=False)
```


Magnitude sensitivity at X=250 m: YZ view, P- and S-waves
Z Coordinate
1000
2000
3000
4000
5000
−6000
−4000
−2000
0
2000
4000
6000
Y Coordinate
3.12
2.88
2.64
2.40
2.16
1.92
1.68
1.44
M_w


Magnitude sensitivity at Y=250 m: XZ view, P- and S-waves
Z Coordinate
1000
2000
3000
4000
5000
−6000
−4000
−2000
0
2000
4000
6000
X Coordinate
3.12
2.88
2.64
2.40
2.16
1.92
1.68
1.44
M_w



-

Total running time of the script: (0 minutes 4.116 seconds)

3.4 04.1 Magnitude sensitivity for homogeneous model: test case 4

This is an example of computing magnitude sensitivity for a homogeneous model.

Test case 4:

- Geometry: combination of station geometry and two DAS geometries (2 vertical boreholes)
- Velocity model: homogeneous, Vp = 4300 m/s, Vp/Vs = 1.73, Qp = 100, Qs = 50, rho = 2300 kg/m^3
- Processing: magnitude sensitivity computation for individual geometries and for combined geometry (for grid slices only).

3.4.1 Load all necessary packages

```
import pynetdesign.visualization as pndvis
import pynetdesign.modelling as pndmod
import numpy as np
from time import time

# sphinx_gallery_thumbnail_number = -2
```

3.4.2 Input

Read geometry

```
station_geometry_file_path = '../data/homo/test_04/TMPL_net_geometryA.txt'
das_geometry_file_path_1 = '../data/homo/test_04/TMPL_net_geometryB.txt'
das_geometry_file_path_2 = '../data/homo/test_04/TMPL_net_geometryC.txt'

station_geometry_df = pndmod.io.read_geometry(station_geometry_file_path)
das_geometry_df_1 = pndmod.io.read_geometry(das_geometry_file_path_1)
das_geometry_df_2 = pndmod.io.read_geometry(das_geometry_file_path_2)

### Show
print("\nStation geometry:")
print(station_geometry_df.head())
print(station_geometry_df.attrs)
print("\nDAS 1 geometry:")
print(das_geometry_df_1.head())
print(das_geometry_df_1.attrs)
print("\nDAS 2 geometry:")
print(das_geometry_df_2.head())
print(das_geometry_df_2.attrs)
```

```
Station geometry:
  Name       Y       X    Z    NoiseLevel
0  ST1     0.0     0.0 -0.0  5.000000e-09
1  ST2     0.0  5000.0 -0.0  5.000000e-09
2  ST3  4330.0  2500.0 -0.0  5.000000e-09
{'file_path': '../data/homo/test_04/TMPL_net_geometryA.txt', 'original_coord_units':
↪'m', 'coordinate_system': 'local', 'noise_type': 'velocity', 'noise_unit': 'm/s', '_
↪is_combined': False}

DAS 1 geometry:
     Y       X     Z    NoiseLevel
0  0.0  2500.0  -0.0  5.000000e-12
1  0.0  2500.0  20.0  5.000000e-12
2  0.0  2500.0  40.0  5.000000e-12
3  0.0  2500.0  60.0  5.000000e-12
4  0.0  2500.0  80.0  5.000000e-12
{'file_path': '../data/homo/test_04/TMPL_net_geometryB.txt', 'original_coord_units':
↪'m', 'coordinate_system': 'local', 'noise_type': 'strain', 'noise_unit': 'ε',
↪'gauge_length': 10.0, '_is_combined': False}

DAS 2 geometry:
```

(continues on next page)

(continued from previous page)

```
        Y       X     Z    NoiseLevel
0  2000.0  2500.0  -0.0  5.000000e-12
1  2000.0  2500.0  20.0  5.000000e-12
2  2000.0  2500.0  40.0  5.000000e-12
3  2000.0  2500.0  60.0  5.000000e-12
4  2000.0  2500.0  80.0  5.000000e-12
{'file_path': '../data/homo/test_04/TMPL_net_geometryC.txt', 'original_coord_units':
↪'m', 'coordinate_system': 'local', 'noise_type': 'strain', 'noise_unit': 'ε',
↪'gauge_length': 10.0, '_is_combined': False}
```

Combine geometry

```
combined_geometry_df = pndmod.io.combine_geometry(station_geometry_df,
                                                  das_geometry_df_1,
                                                  das_geometry_df_2,
                                                  names=['Stations',
                                                         'DAS_1',
                                                         'DAS_2'])

### Show combined
print("\nCombined geometry:")
print(combined_geometry_df)
print(combined_geometry_df.attrs)
print("\nGeometry names:")
print(combined_geometry_df.get_names())
print("\nCombined status:",
      pndmod.io.is_combined_geometry(combined_geometry_df))

### Show recovered
print("\nStation geometry (recovered):")
print(combined_geometry_df.recover('Stations').head())
print("\nCombined status:",
      pndmod.io.is_combined_geometry(combined_geometry_df.recover('Stations')))

print("\nDAS 1 geometry (recovered):")
print(combined_geometry_df.recover('DAS_1').head())
print("\nCombined status:",
      pndmod.io.is_combined_geometry(combined_geometry_df.recover('DAS_1')))
print("\nAttributes of DAS 1 (recovered):")
print(combined_geometry_df.recover('DAS_1').attrs)

print("\nDAS 2 geometry (recovered):")
print(combined_geometry_df.recover('DAS_2').head())
print("\nCombined status:",
      pndmod.io.is_combined_geometry(combined_geometry_df.recover('DAS_2')))
print("\nAttributes of DAS 2 (recovered):")
print(combined_geometry_df.recover('DAS_2').attrs)
```

```
Combined geometry:
            Name       Y       X       Z    NoiseLevel
source
```

(continues on next page)

(continued from previous page)

```
Stations 0   ST1     0.0     0.0    -0.0  5.000000e-09
         1   ST2     0.0  5000.0    -0.0  5.000000e-09
         2   ST3  4330.0  2500.0    -0.0  5.000000e-09
DAS_1    0   NaN     0.0  2500.0    -0.0  5.000000e-12
         1   NaN     0.0  2500.0    20.0  5.000000e-12
...          ...     ...     ...     ...           ...
DAS_2    46  NaN  2000.0  2500.0   920.0  5.000000e-12
         47  NaN  2000.0  2500.0   940.0  5.000000e-12
         48  NaN  2000.0  2500.0   960.0  5.000000e-12
         49  NaN  2000.0  2500.0   980.0  5.000000e-12
         50  NaN  2000.0  2500.0  1000.0  5.000000e-12

[105 rows x 5 columns]
{'original_attributes': {'Stations': {'file_path': '../data/homo/test_04/TMPL_net_
↪geometryA.txt', 'original_coord_units': 'm', 'coordinate_system': 'local', 'noise_
↪type': 'velocity', 'noise_unit': 'm/s', '_is_combined': False}, 'DAS_1': {'file_path
↪': '../data/homo/test_04/TMPL_net_geometryB.txt', 'original_coord_units': 'm',
↪'coordinate_system': 'local', 'noise_type': 'strain', 'noise_unit': 'ε', 'gauge_
↪length': 10.0, '_is_combined': False}, 'DAS_2': {'file_path': '../data/homo/test_04/
↪TMPL_net_geometryC.txt', 'original_coord_units': 'm', 'coordinate_system': 'local',
↪'noise_type': 'strain', 'noise_unit': 'ε', 'gauge_length': 10.0, '_is_combined':␣
↪False}}, '_is_combined': True}

Geometry names:
['Stations', 'DAS_1', 'DAS_2']

Combined status: True

Station geometry (recovered):
  Name       Y       X    Z    NoiseLevel
0  ST1     0.0     0.0 -0.0  5.000000e-09
1  ST2     0.0  5000.0 -0.0  5.000000e-09
2  ST3  4330.0  2500.0 -0.0  5.000000e-09

Combined status: False

DAS 1 geometry (recovered):
     Y       X     Z    NoiseLevel
0  0.0  2500.0  -0.0  5.000000e-12
1  0.0  2500.0  20.0  5.000000e-12
2  0.0  2500.0  40.0  5.000000e-12
3  0.0  2500.0  60.0  5.000000e-12
4  0.0  2500.0  80.0  5.000000e-12

Combined status: False

Attributes of DAS 1 (recovered):
{'file_path': '../data/homo/test_04/TMPL_net_geometryB.txt', 'original_coord_units':
↪'m', 'coordinate_system': 'local', 'noise_type': 'strain', 'noise_unit': 'ε',
↪'gauge_length': 10.0, '_is_combined': False}

DAS 2 geometry (recovered):
```

(continues on next page)

(continued from previous page)

```
        Y       X     Z    NoiseLevel
0  2000.0  2500.0  -0.0  5.000000e-12
1  2000.0  2500.0  20.0  5.000000e-12
2  2000.0  2500.0  40.0  5.000000e-12
3  2000.0  2500.0  60.0  5.000000e-12
4  2000.0  2500.0  80.0  5.000000e-12

Combined status: False

Attributes of DAS 2 (recovered):
{'file_path': '../data/homo/test_04/TMPL_net_geometryC.txt', 'original_coord_units':
↪'m', 'coordinate_system': 'local', 'noise_type': 'strain', 'noise_unit': 'ε',
↪'gauge_length': 10.0, '_is_combined': False}
```

Read velocity model

```
velocity_file_path = '../data/homo/test_04/TMPL_velocity_model.txt'
velocity_df = pndmod.io.read_velocity(velocity_file_path)

print(velocity_df)
print(velocity_df.attrs)
```

```
   Depth    Vp  VpVsRatio   Qp  Qs   Rho   Ep   Dt   Gm
0  40000  4300       1.73  100  50  2300  0.0  0.0  0.0
{'model_type': 'homo', 'model_anisotropy': 'isotropic', 'parameter_descriptions': {
↪'Depth': 'Depth', 'Vp': 'P-wave velocity', 'Vs': 'S-wave velocity', 'VpVsRatio':
↪'Vp/Vs', 'Qp': 'P-wave quality factor', 'Qs': 'S-wave quality factor', 'Rho':
↪'density', 'Ep': 'Thomsen epsilon parameter', 'Dt': 'Thomsen delta parameter', 'Gm
↪': 'Thomsen gamma parameter'}, 'parameter_units': {'Depth': 'm', 'Vp': 'm/s', 'Vs':
↪'m/s', 'VpVsRatio': '', 'Qp': '', 'Qs': '', 'Rho': 'kg/m^3', 'Ep': '', 'Dt': '', 'Gm
↪': ''}}
```

Set imaging grid parameters

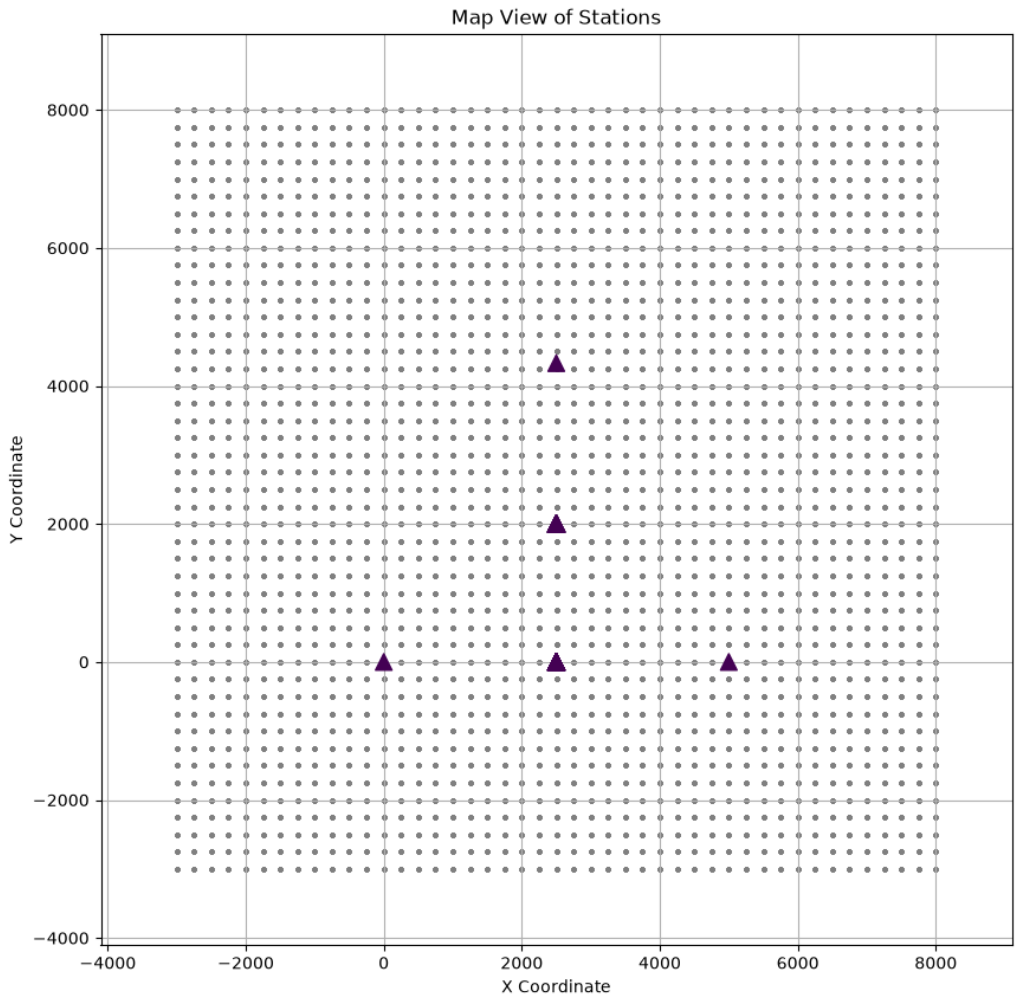
```
# Imaging grid parameters
gx = np.arange(-3000, 8000 + 250, 250)
gy = np.arange(-3000, 8000 + 250, 250)
gz = np.arange(250, 5000 + 250, 250)

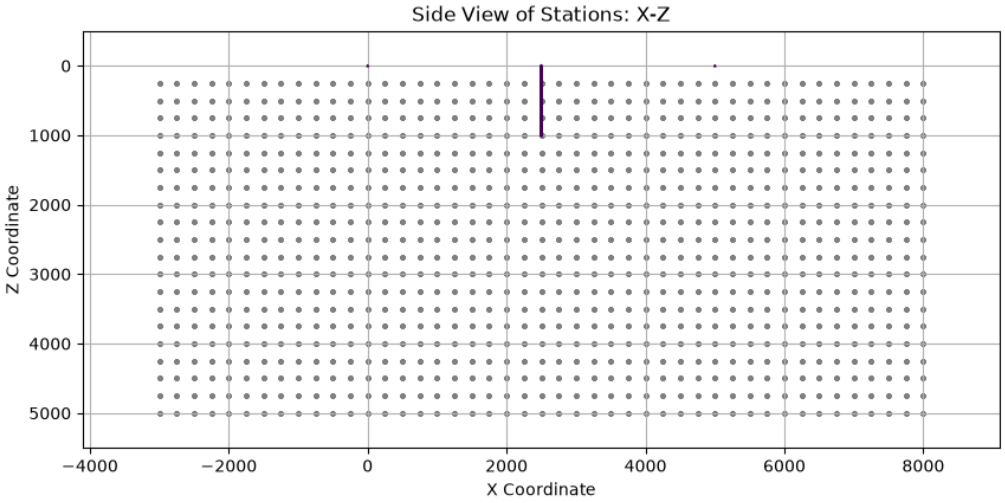
# Generate grid points
grid_points = pndmod.utils.generate_grid(x=gx,y=gy,z=gz)
print("Number of grid points:",grid_points.shape[0])
print("grid_points.shape:",grid_points.shape)
```

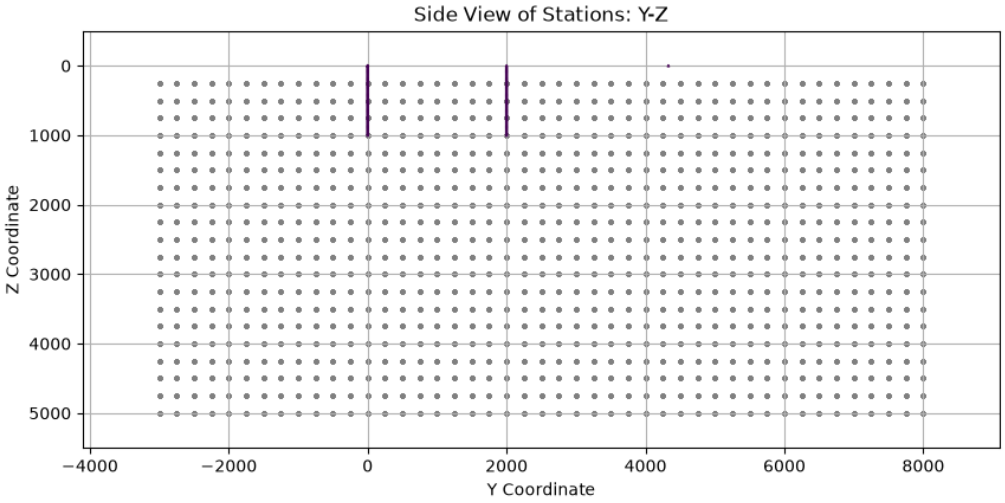
```
Number of grid points: 40500
grid_points.shape: (40500, 3)
```

Plot geometry and imaging grid

```
_ = pndvis.geometry.plot_stations_view(df=combined_geometry_df,
                                       grid_points=grid_points,
                                       plot_names=False,
                                       grid_marker_size = 1,
                                       station_marker_size=100)
_ = pndvis.geometry.plot_stations_view(df=combined_geometry_df,
                                       grid_points=grid_points,
                                       plot_names=False,
                                       grid_marker_size = 1,
                                       station_marker_size=1,
                                       view = 'xz')
_ = pndvis.geometry.plot_stations_view(df=combined_geometry_df,
                                       grid_points=grid_points,
                                       plot_names=False,
                                       grid_marker_size = 1,
                                       station_marker_size=1,
                                       view = 'yz')
```


Map View of Stations
Y Coordinate
X Coordinate
8000
6000
4000
2000
0
−2000
−4000
−4000
−2000
0
2000
4000
6000
8000


Side View of Stations: X-Z
Z Coordinate
X Coordinate
0
1000
2000
3000
4000
5000
−4000
−2000
0
2000
4000
6000
8000



-

3.4.3 Prepare to compute magnitude sensitivity

```
# Get medium parameters
density = velocity_df.at[0,'Rho']
v_p = velocity_df.at[0,'Vp']
Q_p = velocity_df.at[0,'Qp']
v_s = velocity_df.at[0,'Vp']/velocity_df.at[0,'VpVsRatio']
Q_s = velocity_df.at[0,'Qs']
print("density, v_p, Q_p, v_s, Q_s:",density,v_p,Q_p,v_s,Q_s)

# Specify source paramerers
frequency = 10 # Hz

# Minimum S/N for detection on individual channel
min_SNR_p = 2
```

(continues on next page)

(continued from previous page)

```
min_SNR_s = 2

# Minimum stations on which an event must be detected
min_stations_p = 3
min_stations_s = 3

# Save parameters
params = pndmod.classes.DetectionParameters(
    f_p=frequency,
    f_s=frequency,
    min_SNR_p=min_SNR_p,
    min_SNR_s=min_SNR_s,
    min_stations_p=min_stations_p,
    min_stations_s=min_stations_s,
)

# Define slices to use
slice_X = 250
slice_Y = 250
slice_Z = 2000

# Get slice planes
X_plane_points, X_plane_indices = pndmod.utils.get_slice_coordinates(grid_points=grid_
↪points,slice_dimension=0,slice_coordinate=slice_X)
Y_plane_points, Y_plane_indices = pndmod.utils.get_slice_coordinates(grid_points=grid_
↪points,slice_dimension=1,slice_coordinate=slice_Y)
Z_plane_points, Z_plane_indices = pndmod.utils.get_slice_coordinates(grid_points=grid_
↪points,slice_dimension=2,slice_coordinate=slice_Z)
```

```
density, v_p, Q_p, v_s, Q_s: 2300 4300 100 2485.549132947977 50
```

3.4.4 Compute and plot magnitude sensitivity for independent geometries

```
print("Computing and plotting magnitude sensitivity for independent geometries...")

# Loop over geometries
for geometry in combined_geometry_df.get_names():
    print(f"Computing and plotting magnitude sensitivity for {geometry} geometry...")
    start_time = time()
    for wave_mode in ['P','S','PS']:
        X_slice_mag_sens = pndmod.core.get_mag_sensitivity(grid_coords=X_plane_points,
                                                           geometry_df=combined_
↪geometry_df.recover(geometry),
                                                           velocity_df=velocity_
↪df,
                                                           params=params,
                                                           wave_mode=wave_mode)
        Y_slice_mag_sens = pndmod.core.get_mag_sensitivity(grid_coords=Y_plane_points,
                                                           geometry_df=combined_
↪geometry_df.recover(geometry),
                                                           velocity_df=velocity_
```

(continues on next page)

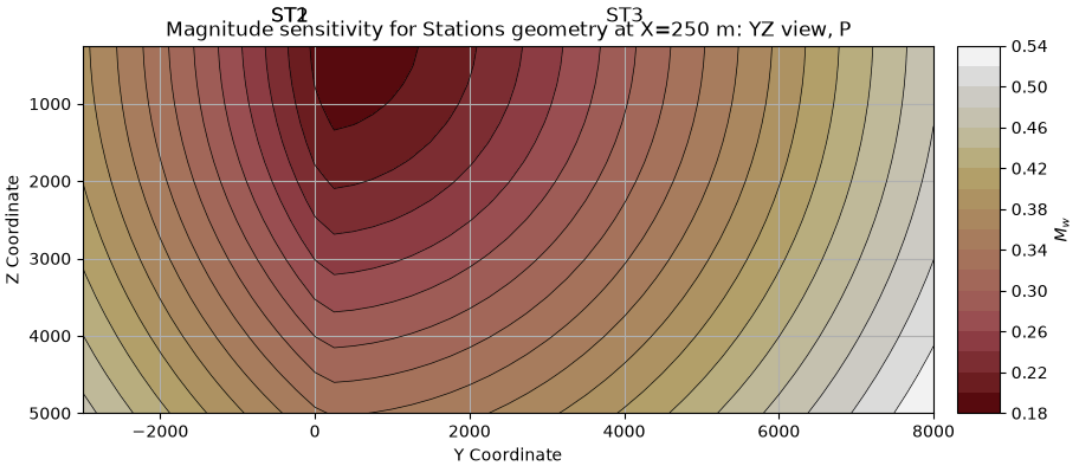
(continued from previous page)

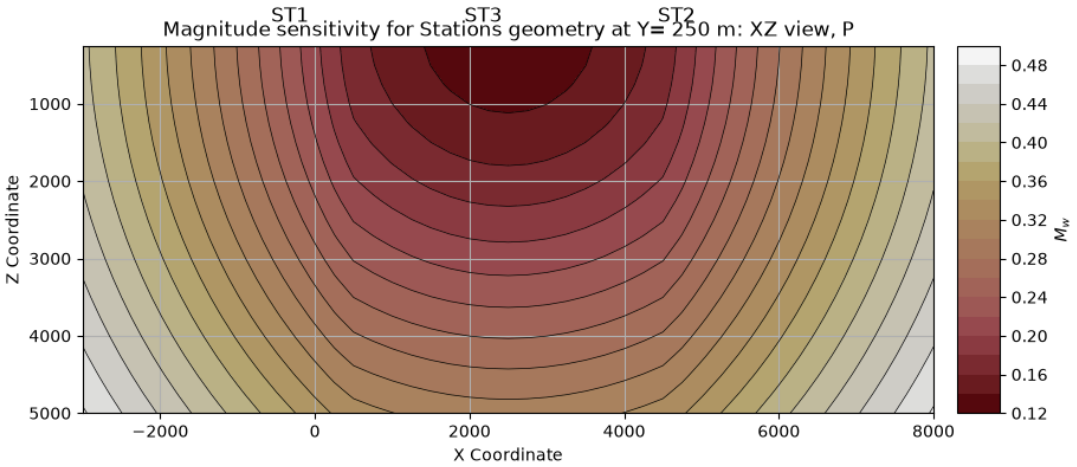
```
↪df,
                                                                    params=params,
                                                                    wave_mode=wave_mode)
        Z_slice_mag_sens = pndmod.core.get_mag_sensitivity(grid_coords=Z_plane_points,
                                                                    geometry_df=combined_
↪geometry_df.recover(geometry),
                                                                    velocity_df=velocity_
↪df,
                                                                    params=params,
                                                                    wave_mode=wave_mode)

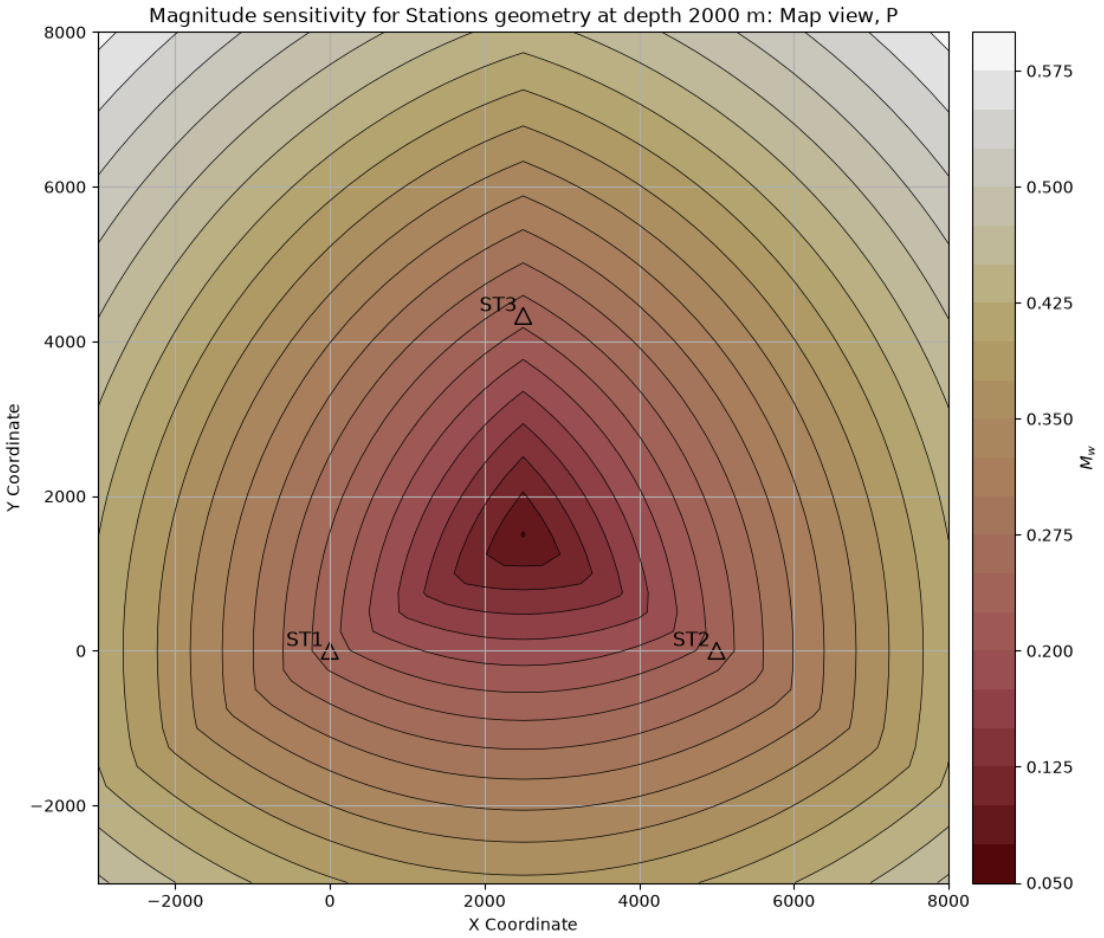
        _ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=X_slice_mag_sens,␣
↪yi=gy, zi=gz,
                                                      plot_title=f"Magnitude␣
↪sensitivity for {geometry} geometry at X={slice_X} m: YZ view, {wave_mode}",
                                                      clb_title="$M_w$",
                                                      geometry_df=combined_geometry_
↪df.recover(geometry),
                                                      plot_names = True if 'Name' in␣
↪combined_geometry_df.recover(geometry).columns else False,
                                                      station_marker_size = 100 if
↪'gauge_length' not in combined_geometry_df.recover(geometry).attrs else 10)

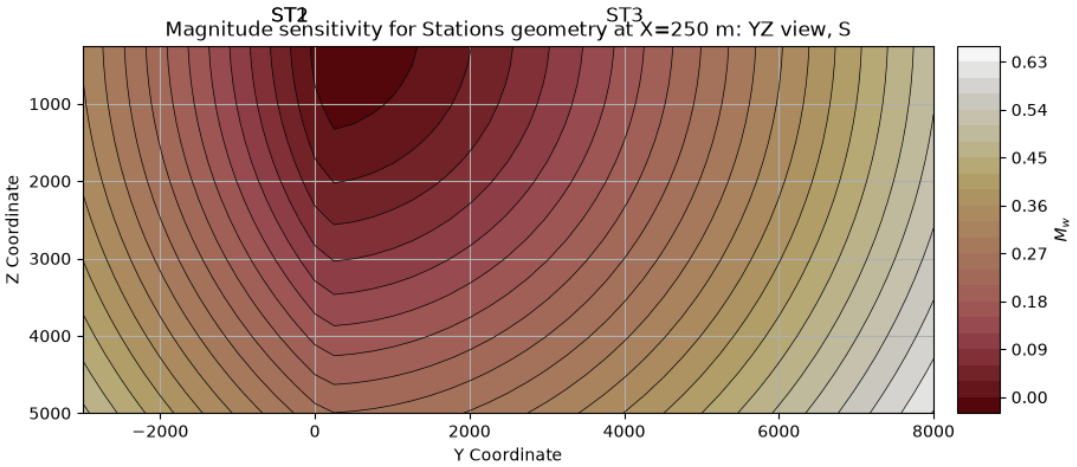
        _ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=Y_slice_mag_sens,␣
↪xi=gx, zi=gz,
                                                      plot_title=f"Magnitude␣
↪sensitivity for {geometry} geometry at Y= {slice_Y} m: XZ view, {wave_mode}",
                                                      clb_title="$M_w$",
                                                      geometry_df=combined_geometry_
↪df.recover(geometry),
                                                      plot_names = True if 'Name' in␣
↪combined_geometry_df.recover(geometry).columns else False,
                                                      station_marker_size = 100 if
↪'gauge_length' not in combined_geometry_df.recover(geometry).attrs else 10)

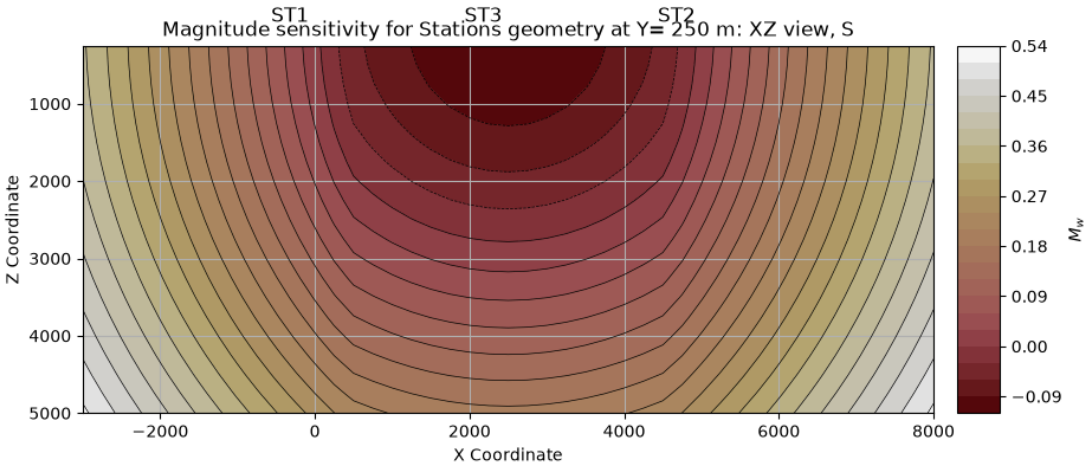
        _ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=Z_slice_mag_sens,␣
↪xi=gx, yi=gy,
                                                      plot_title=f"Magnitude␣
↪sensitivity for {geometry} geometry at depth {slice_Z} m: Map view, {wave_mode}",
                                                      clb_title="$M_w$",
                                                      geometry_df=combined_geometry_
↪df.recover(geometry),
                                                      plot_names = True if 'Name' in␣
↪combined_geometry_df.recover(geometry).columns else False,
                                                      station_marker_size = 100 if
↪'gauge_length' not in combined_geometry_df.recover(geometry).attrs else 10)
    end_time = time()
    print(f"Computation time: {end_time - start_time} seconds")
```

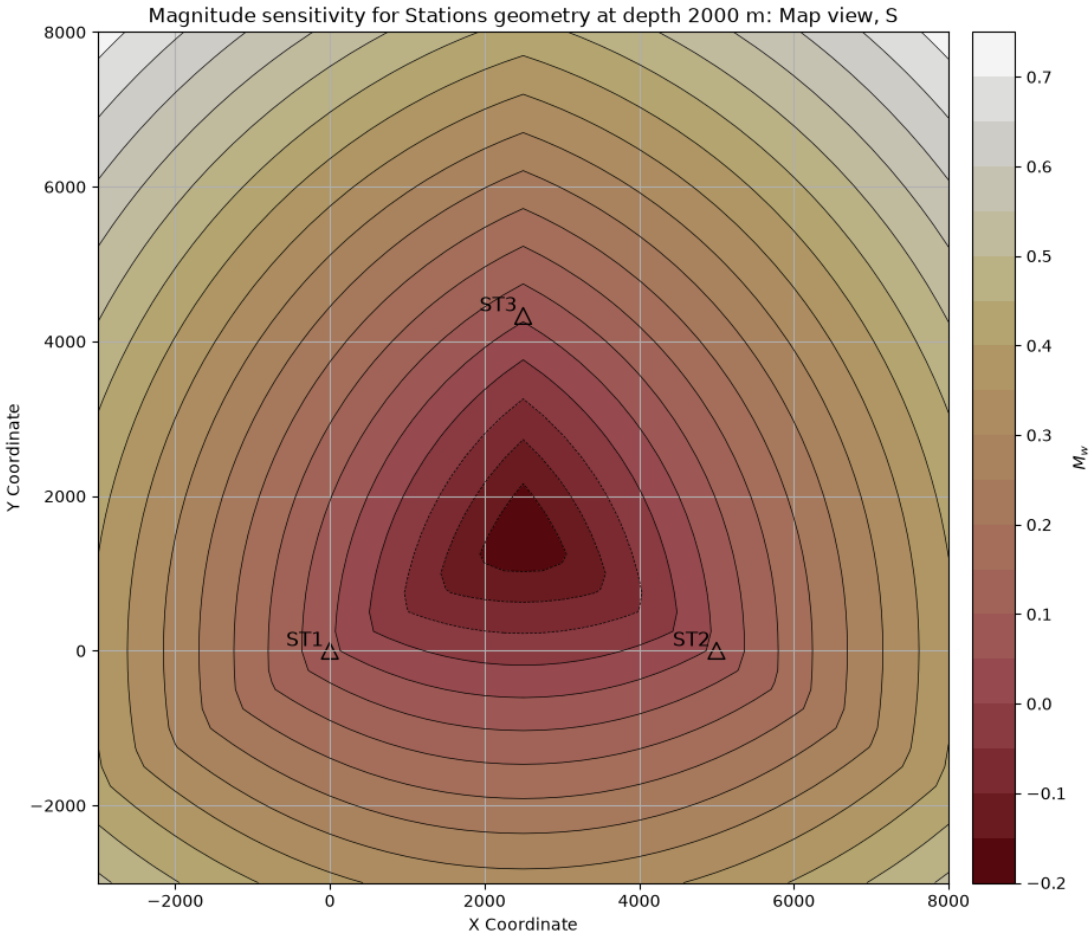

ST2
ST3
Magnitude sensitivity for Stations geometry at X=250 m: YZ view, P
Z Coordinate
1000
2000
3000
4000
5000
−2000
0
2000
4000
6000
8000
Y Coordinate
0.54
0.50
0.46
0.42
0.38
0.34
0.30
0.26
0.22
0.18
M_w


ST1
ST3
ST2
Magnitude sensitivity for Stations geometry at Y= 250 m: XZ view, P
Z Coordinate
1000
2000
3000
4000
5000
−2000
0
2000
4000
6000
8000
X Coordinate
0.48
0.44
0.40
0.36
0.32
0.28
0.24
0.20
0.16
0.12
M_w

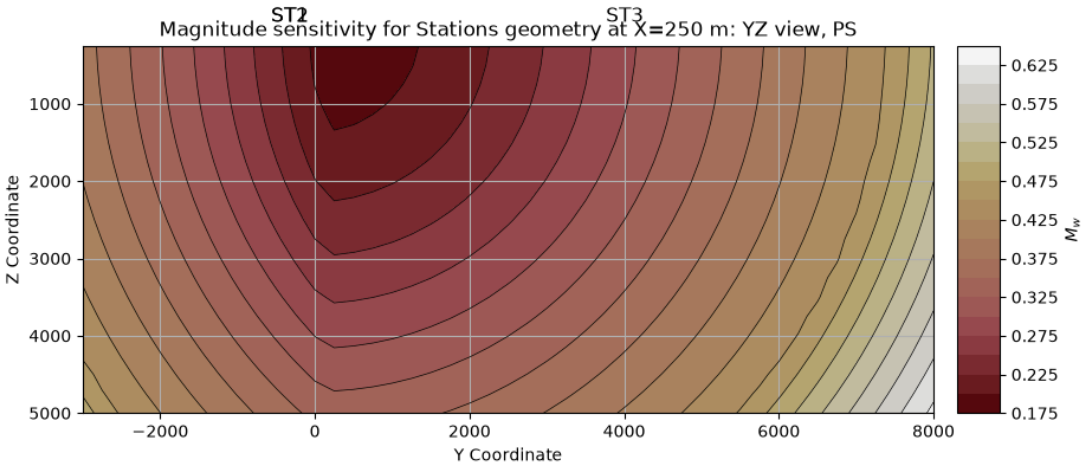

Magnitude sensitivity for Stations geometry at depth 2000 m: Map view, P
ST1
ST2
ST3
Y Coordinate
X Coordinate
M_w
8000
6000
4000
2000
0
−2000
−2000
0
2000
4000
6000
8000
0.575
0.500
0.425
0.350
0.275
0.200
0.125
0.050

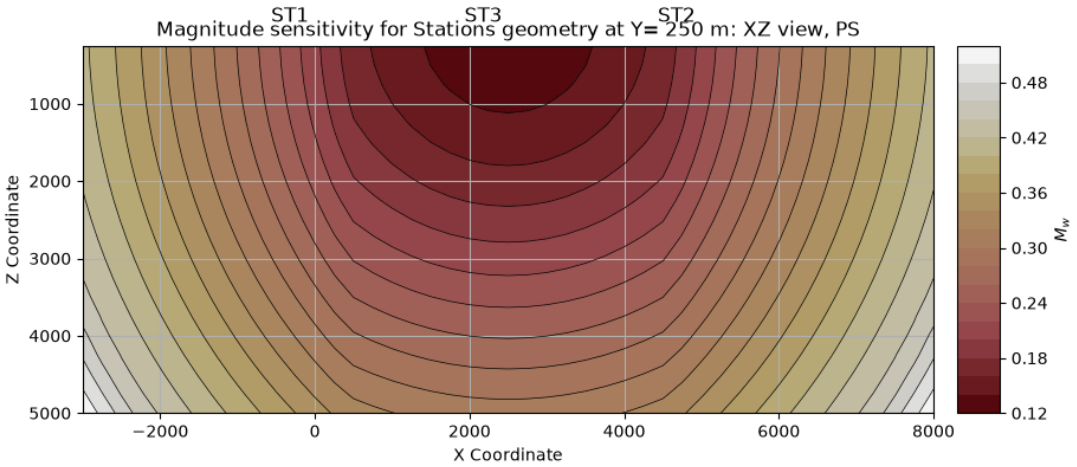

ST2
ST3
Magnitude sensitivity for Stations geometry at X=250 m: YZ view, S
Z Coordinate
1000
2000
3000
4000
5000
−2000
0
2000
4000
6000
8000
Y Coordinate
0.63
0.54
0.45
0.36
0.27
0.18
0.09
0.00
M_w

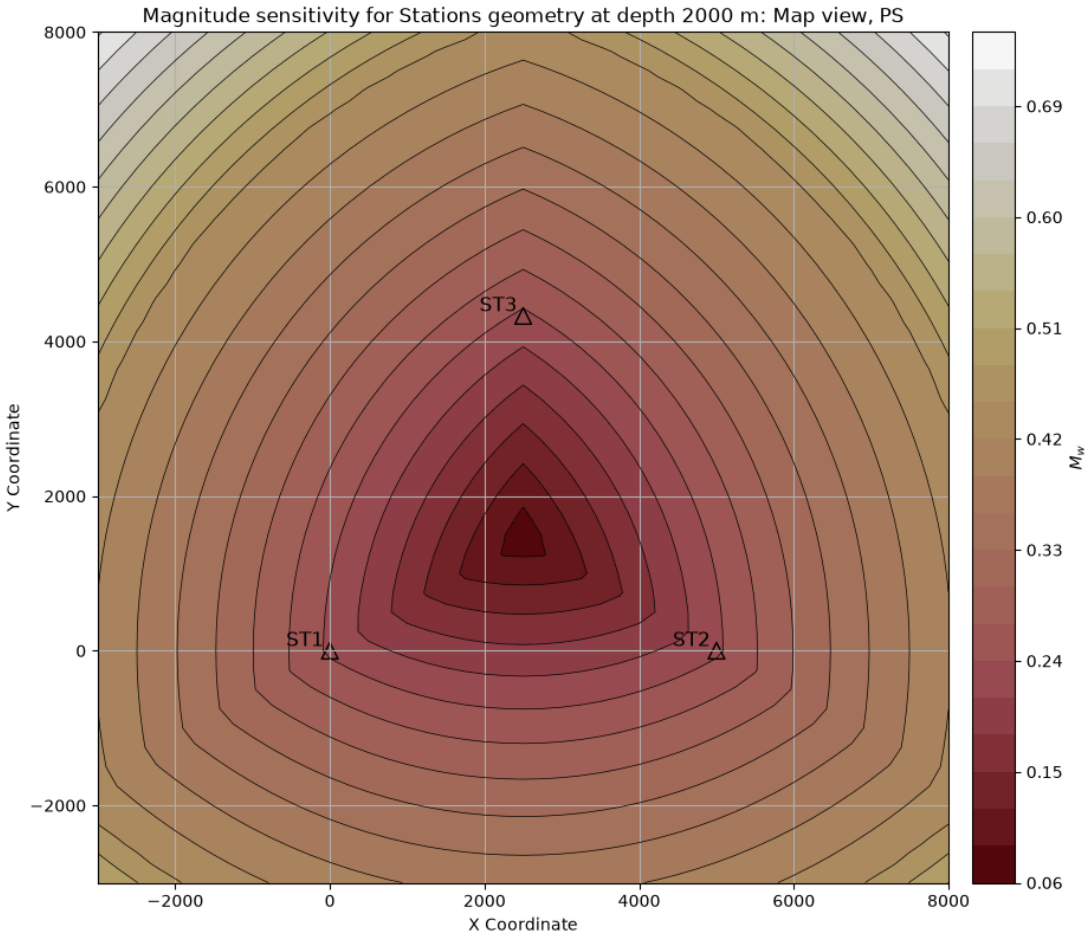

ST1
ST3
ST2
Magnitude sensitivity for Stations geometry at Y= 250 m: XZ view, S
Z Coordinate
1000
2000
3000
4000
5000
−2000
0
2000
4000
6000
8000
X Coordinate
0.54
0.45
0.36
0.27
0.18
0.09
0.00
−0.09
M_w

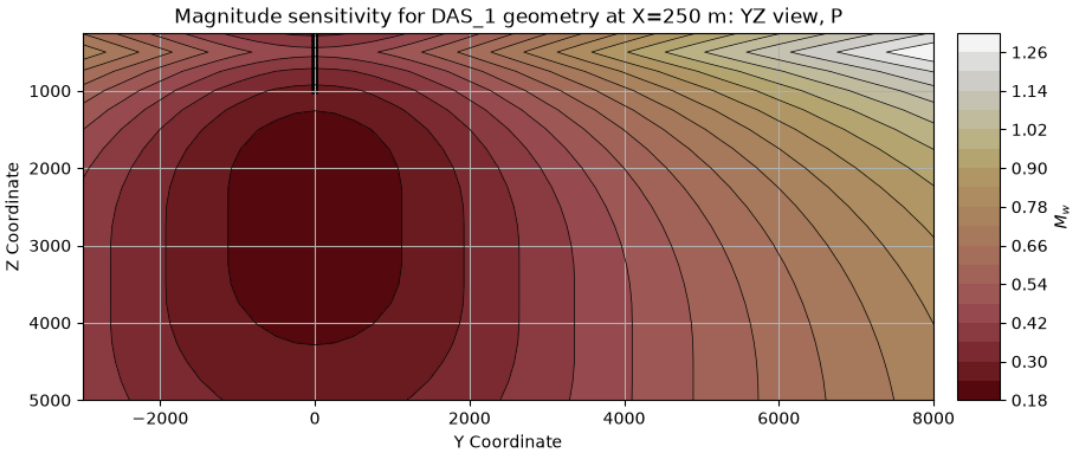

Magnitude sensitivity for Stations geometry at depth 2000 m: Map view, S
ST1
ST2
ST3
Y Coordinate
X Coordinate
M_w
8000
6000
4000
2000
0
−2000
−2000
0
2000
4000
6000
8000
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2

•

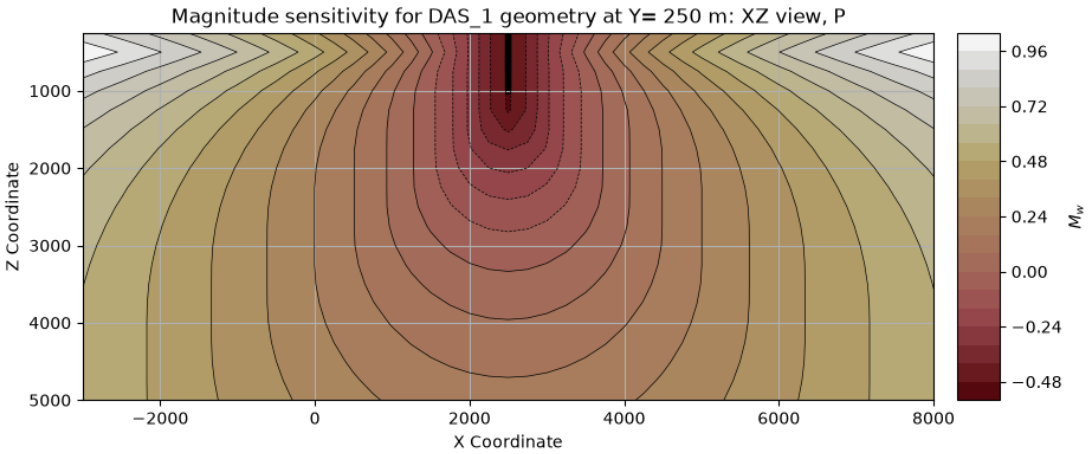

ST2
ST3
Magnitude sensitivity for Stations geometry at X=250 m: YZ view, PS
Z Coordinate
1000
2000
3000
4000
5000
−2000
0
2000
4000
6000
8000
Y Coordinate
0.625
0.575
0.525
0.475
0.425
0.375
0.325
0.275
0.225
0.175
M_w

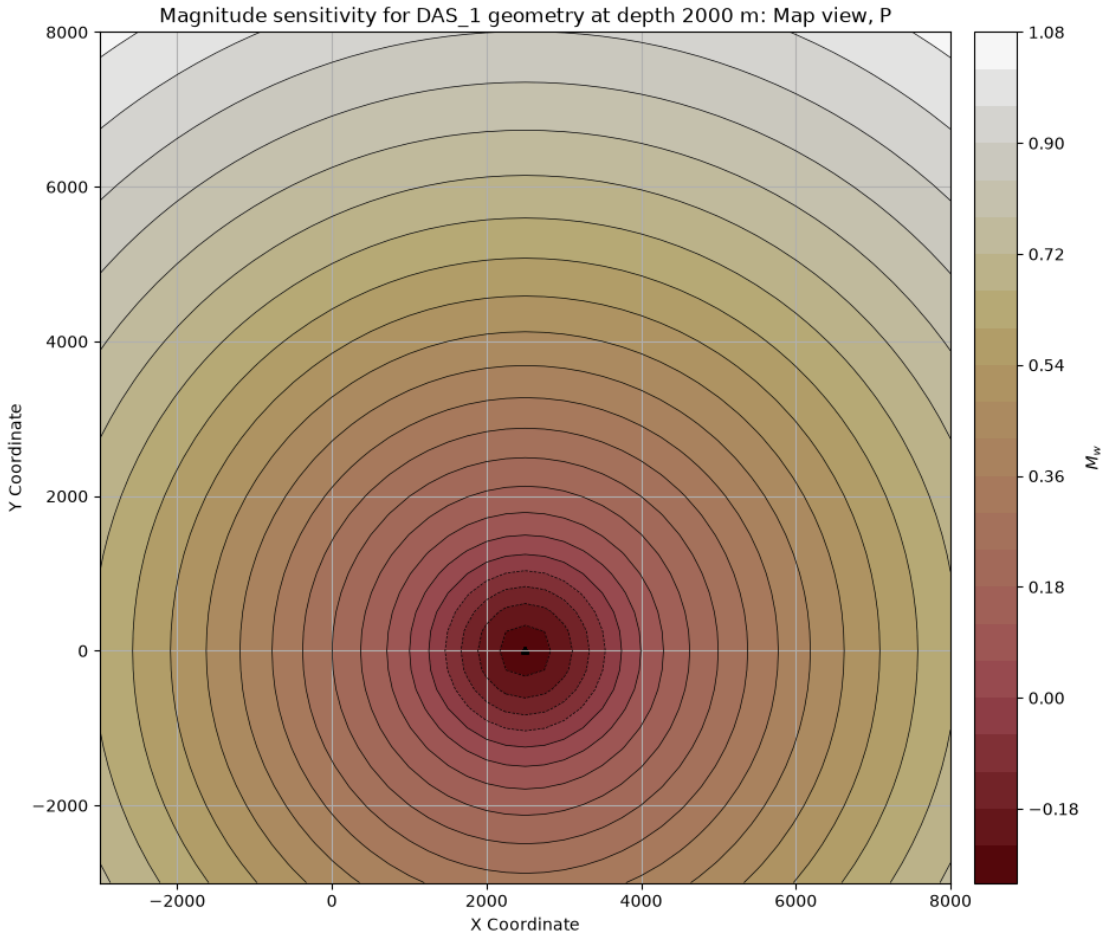

ST1
ST3
ST2
Magnitude sensitivity for Stations geometry at Y= 250 m: XZ view, PS
Z Coordinate
X Coordinate
1000
2000
3000
4000
5000
−2000
0
2000
4000
6000
8000
0.48
0.42
0.36
0.30
0.24
0.18
0.12
M_w

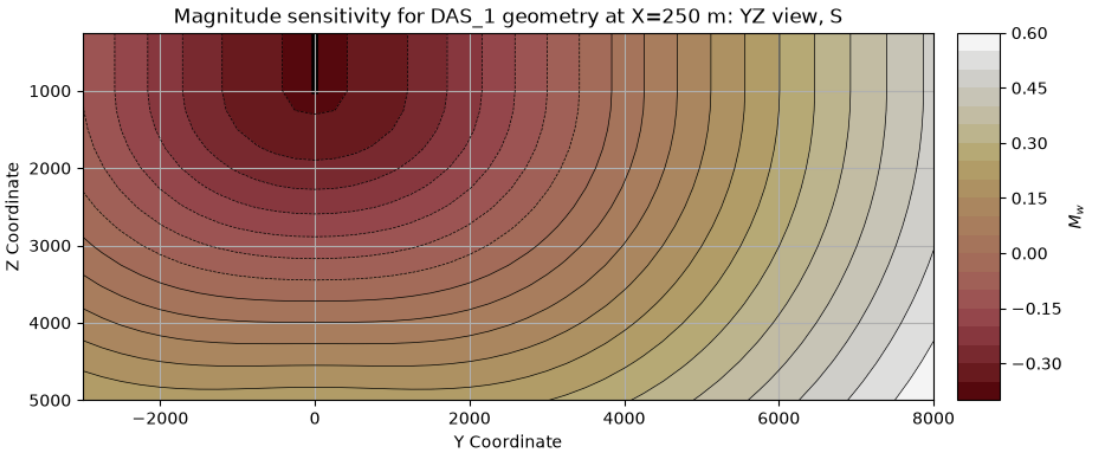

Magnitude sensitivity for Stations geometry at depth 2000 m: Map view, PS
ST1
ST2
ST3
Y Coordinate
X Coordinate
M_w
8000
6000
4000
2000
0
−2000
−2000
0
2000
4000
6000
8000
0.69
0.60
0.51
0.42
0.33
0.24
0.15
0.06

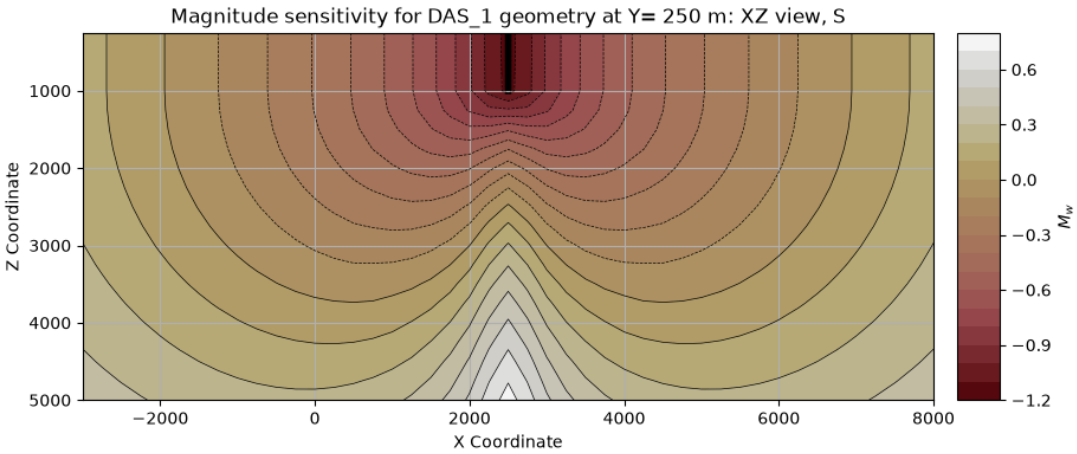

Magnitude sensitivity for DAS_1 geometry at X=250 m: YZ view, P
Z Coordinate
1000
2000
3000
4000
5000
−2000
0
2000
4000
6000
8000
Y Coordinate
1.26
1.14
1.02
0.90
0.78
0.66
0.54
0.42
0.30
0.18
M_w

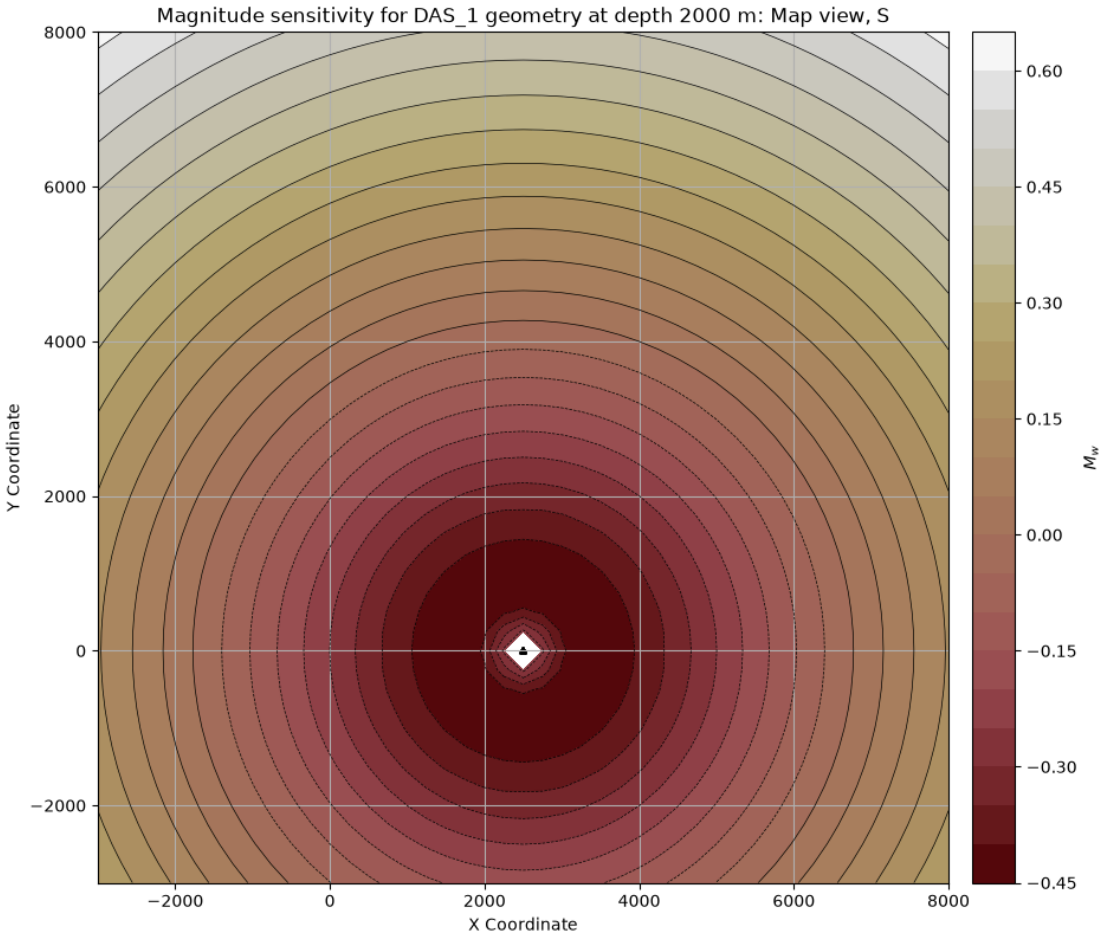
•

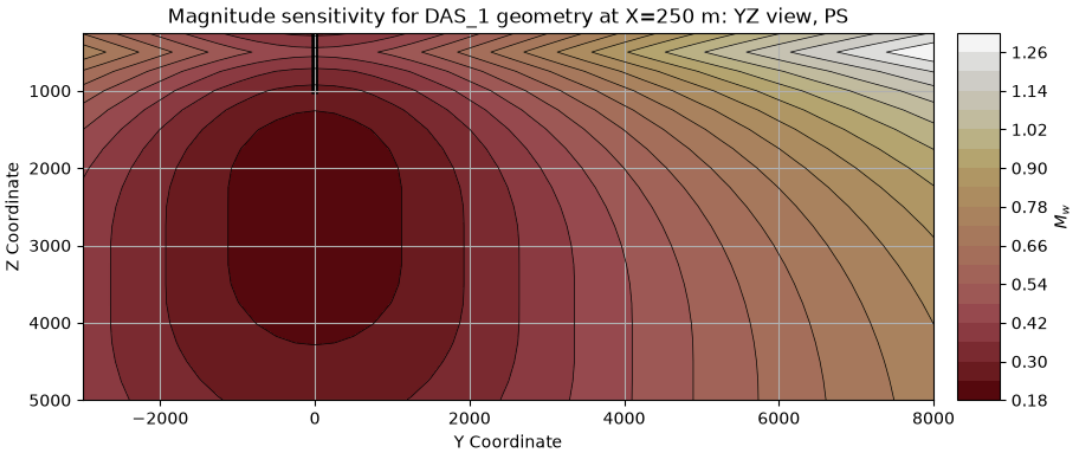

Magnitude sensitivity for DAS_1 geometry at Y= 250 m: XZ view, P
Z Coordinate
1000
2000
3000
4000
5000
−2000
0
2000
4000
6000
8000
X Coordinate
0.96
0.72
0.48
0.24
0.00
−0.24
−0.48
M_w


Magnitude sensitivity for DAS_1 geometry at depth 2000 m: Map view, P
Y Coordinate
X Coordinate
8000
6000
4000
2000
0
−2000
−2000
0
2000
4000
6000
8000
1.08
0.90
0.72
0.54
0.36
0.18
0.00
−0.18
M_w

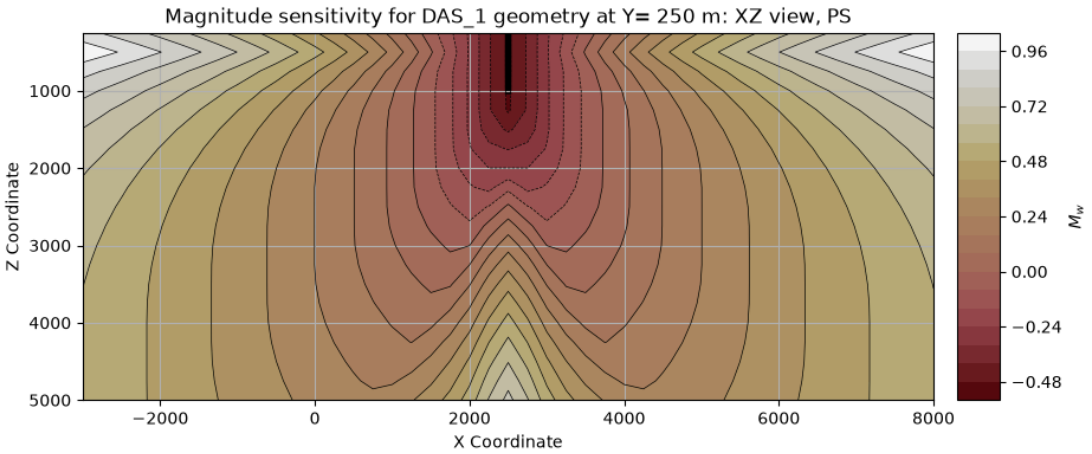

Magnitude sensitivity for DAS_1 geometry at X=250 m: YZ view, S
Z Coordinate
1000
2000
3000
4000
5000
−2000
0
2000
4000
6000
8000
Y Coordinate
0.60
0.45
0.30
0.15
0.00
−0.15
−0.30
M_w

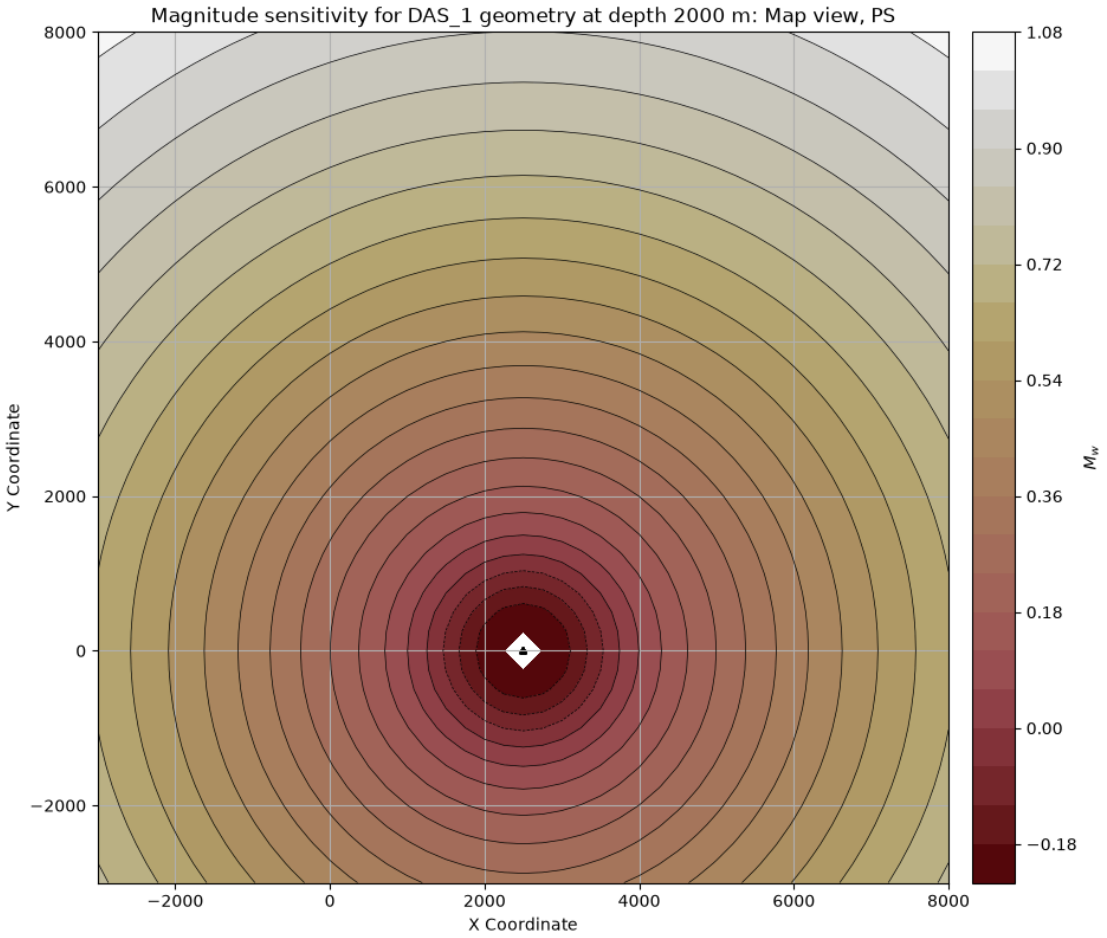

Magnitude sensitivity for DAS_1 geometry at Y= 250 m: XZ view, S
Z Coordinate
X Coordinate
1000
2000
3000
4000
5000
−2000
0
2000
4000
6000
8000
0.6
0.3
0.0
−0.3
−0.6
−0.9
−1.2
M_w


Magnitude sensitivity for DAS_1 geometry at depth 2000 m: Map view, S
Y Coordinate
X Coordinate
8000
6000
4000
2000
0
−2000
−2000
0
2000
4000
6000
8000
0.60
0.45
0.30
0.15
0.00
−0.15
−0.30
−0.45
M_w

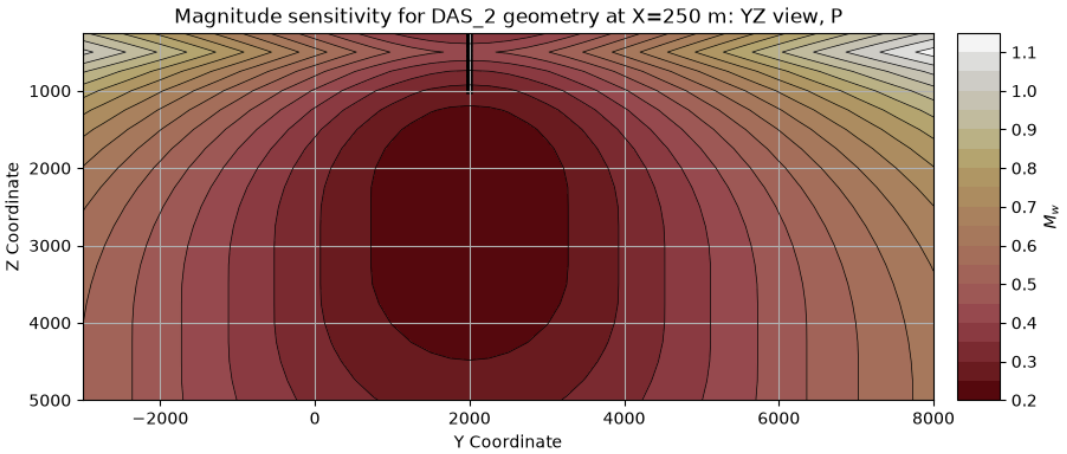

Magnitude sensitivity for DAS_1 geometry at X=250 m: YZ view, PS
Z Coordinate
1000
2000
3000
4000
5000
−2000
0
2000
4000
6000
8000
Y Coordinate
1.26
1.14
1.02
0.90
0.78
0.66
0.54
0.42
0.30
0.18
M_w

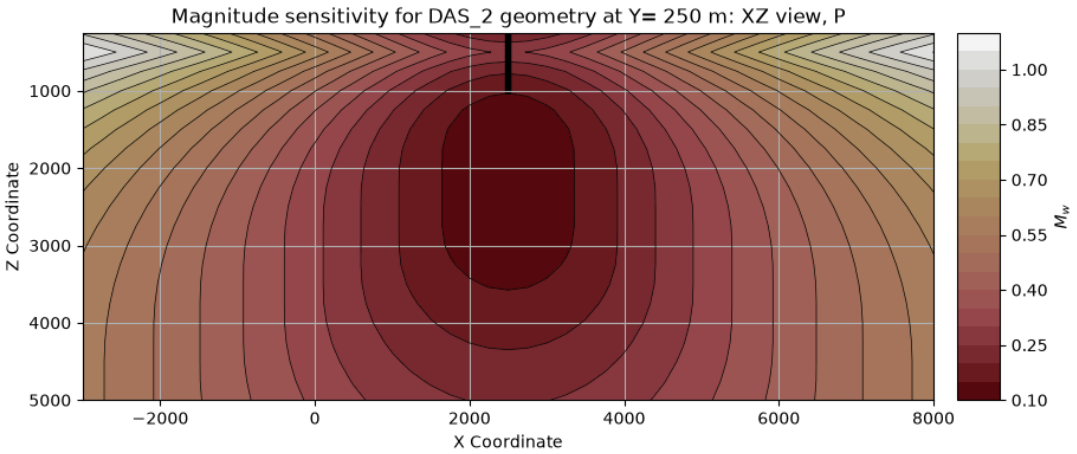
•


Magnitude sensitivity for DAS_1 geometry at Y= 250 m: XZ view, PS
Z Coordinate
1000
2000
3000
4000
5000
−2000
0
2000
4000
6000
8000
X Coordinate
0.96
0.72
0.48
0.24
0.00
−0.24
−0.48
M_w

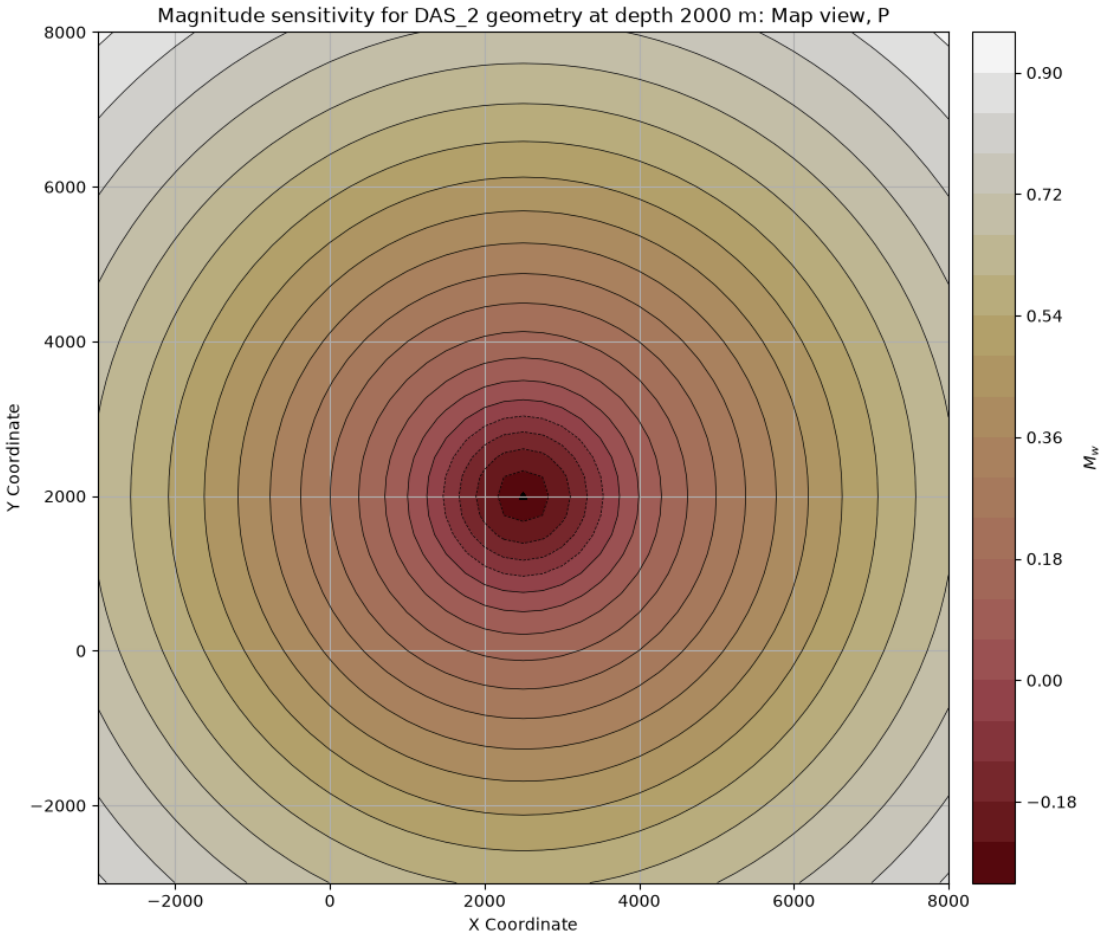

Magnitude sensitivity for DAS_1 geometry at depth 2000 m: Map view, PS
Y Coordinate
X Coordinate
8000
6000
4000
2000
0
−2000
−2000
0
2000
4000
6000
8000
1.08
0.90
0.72
0.54
0.36
0.18
0.00
−0.18
M_w

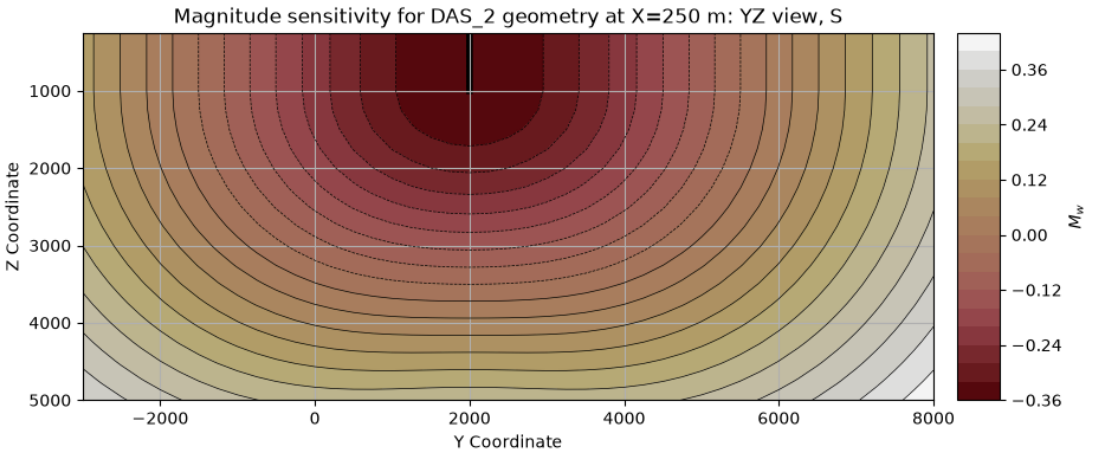
•

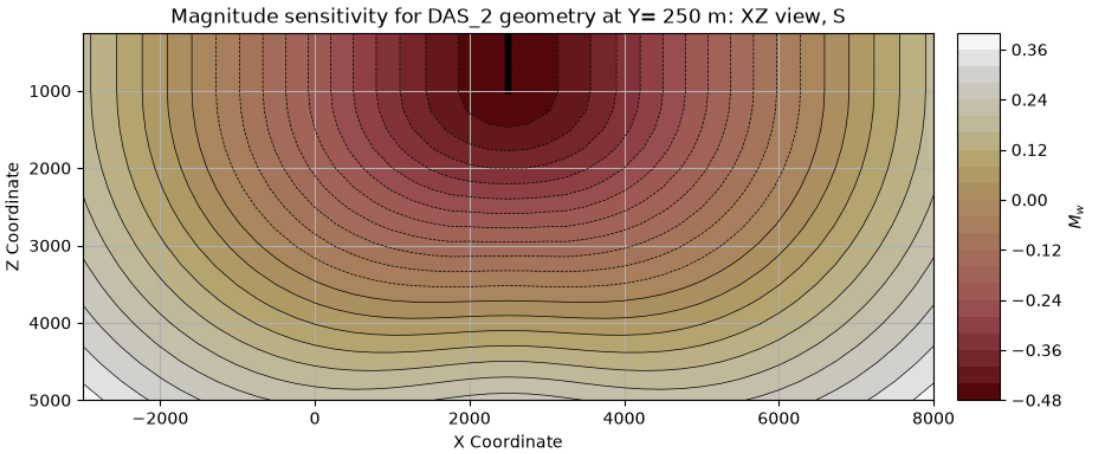

Magnitude sensitivity for DAS_2 geometry at X=250 m: YZ view, P
Z Coordinate
1000
2000
3000
4000
5000
−2000
0
2000
4000
6000
8000
Y Coordinate
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
M_w

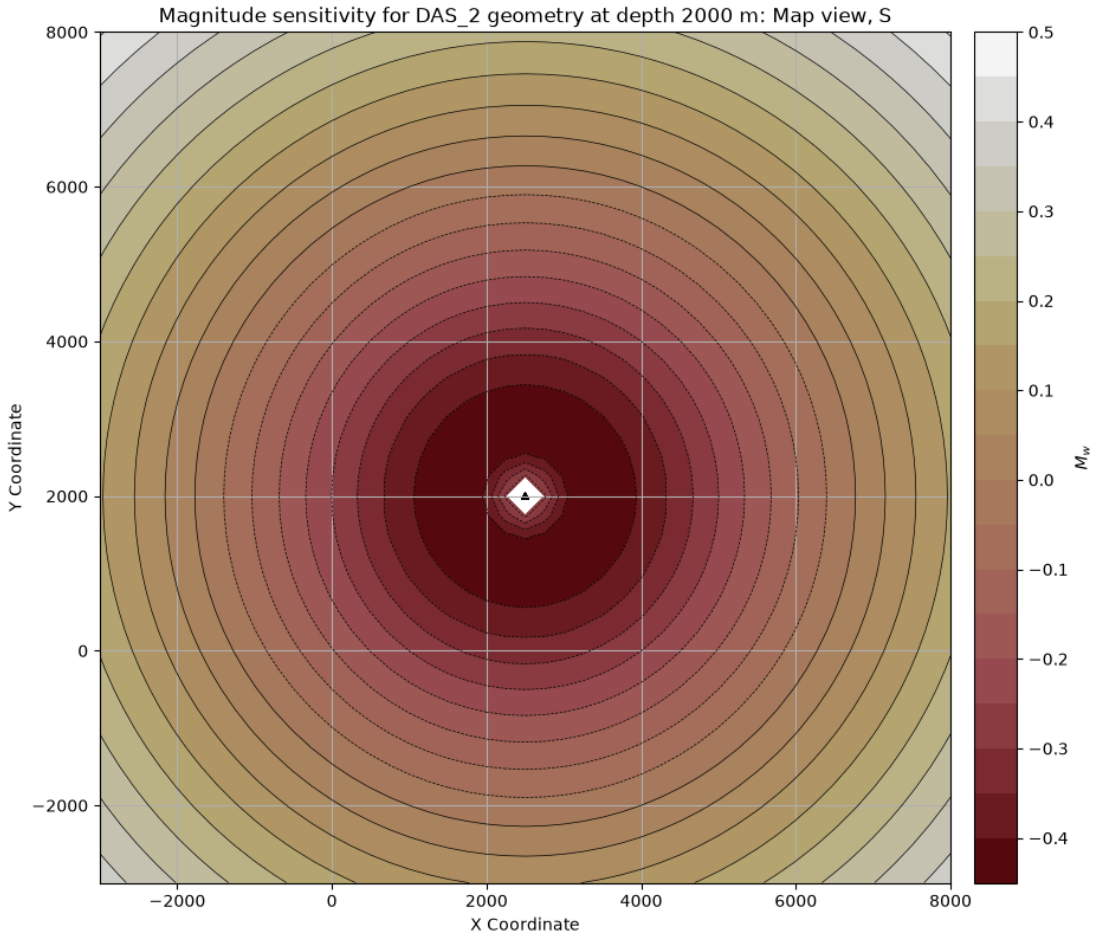

Magnitude sensitivity for DAS_2 geometry at Y= 250 m: XZ view, P
Z Coordinate
X Coordinate
1000
2000
3000
4000
5000
−2000
0
2000
4000
6000
8000
1.00
0.85
0.70
0.55
0.40
0.25
0.10
M_w

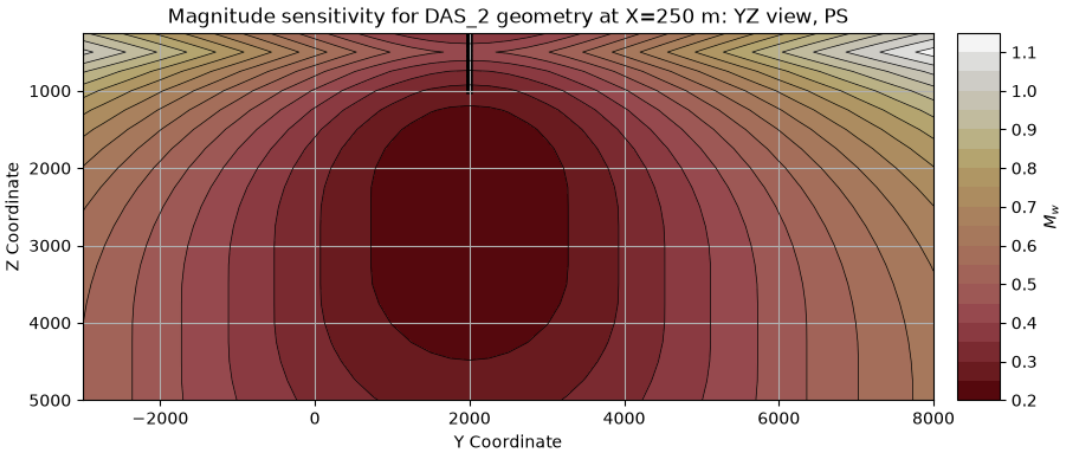
•

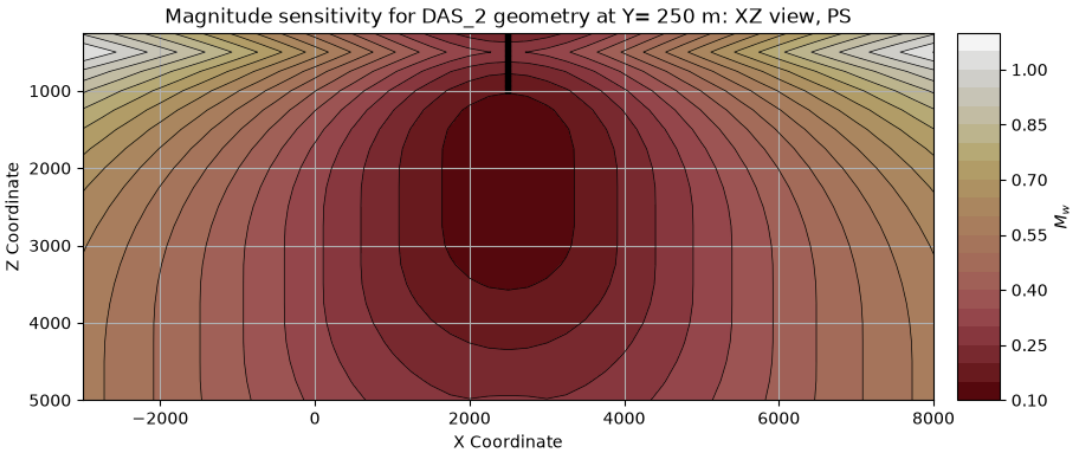

Magnitude sensitivity for DAS_2 geometry at depth 2000 m: Map view, P
Y Coordinate
X Coordinate
8000
6000
4000
2000
0
−2000
−2000
0
2000
4000
6000
8000
0.90
0.72
0.54
0.36
0.18
0.00
−0.18
M_w

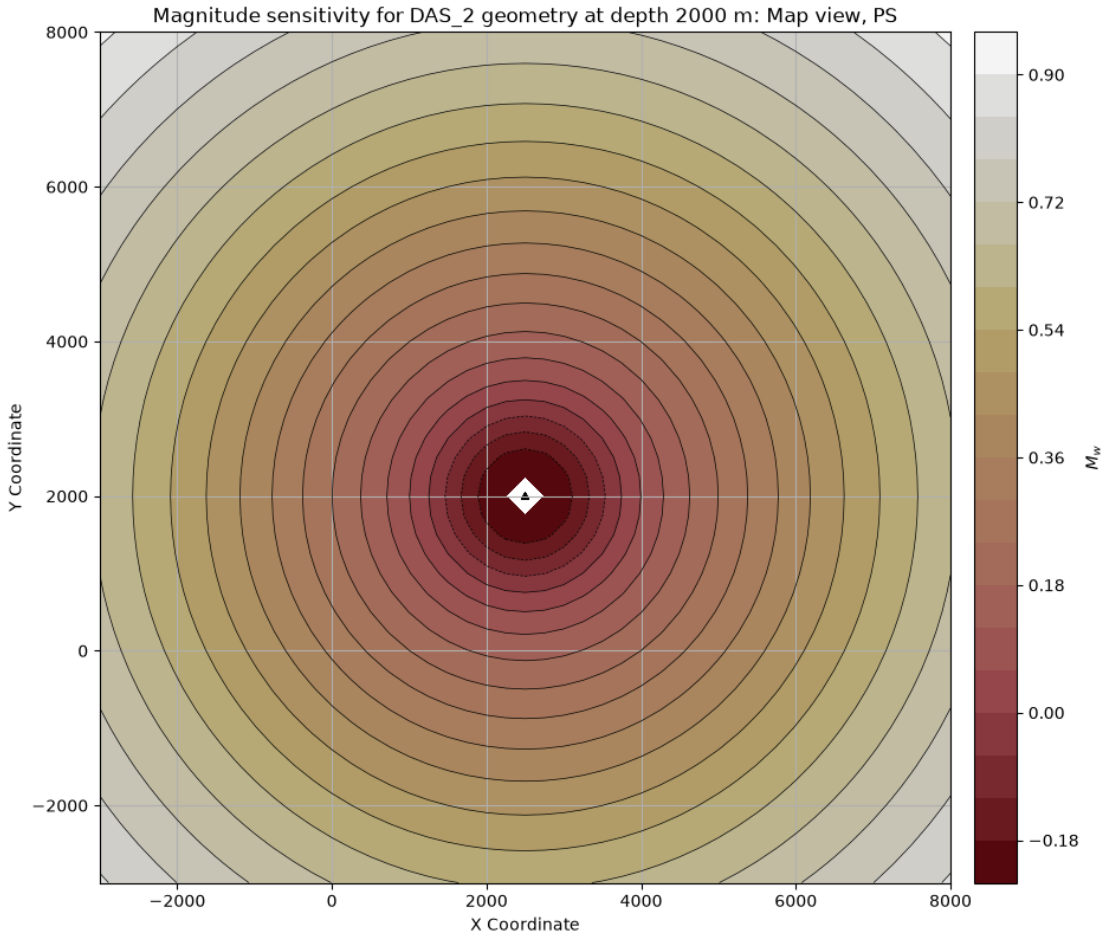

Magnitude sensitivity for DAS_2 geometry at X=250 m: YZ view, S
Z Coordinate
1000
2000
3000
4000
5000
−2000
0
2000
4000
6000
8000
Y Coordinate
0.36
0.24
0.12
0.00
−0.12
−0.24
−0.36
M_w


Magnitude sensitivity for DAS_2 geometry at Y= 250 m: XZ view, S
Z Coordinate
X Coordinate
1000
2000
3000
4000
5000
−2000
0
2000
4000
6000
8000
0.36
0.24
0.12
0.00
−0.12
−0.24
−0.36
−0.48
M_w


Magnitude sensitivity for DAS_2 geometry at depth 2000 m: Map view, S
Y Coordinate
X Coordinate
8000
6000
4000
2000
0
−2000
−2000
0
2000
4000
6000
8000
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
M_w


Magnitude sensitivity for DAS_2 geometry at X=250 m: YZ view, PS
Z Coordinate
1000
2000
3000
4000
5000
−2000
0
2000
4000
6000
8000
Y Coordinate
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
M_w


Magnitude sensitivity for DAS_2 geometry at Y= 250 m: XZ view, PS
Z Coordinate
X Coordinate
1000
2000
3000
4000
5000
−2000
0
2000
4000
6000
8000
1.00
0.85
0.70
0.55
0.40
0.25
0.10
M_w




```
Computing and plotting magnitude sensitivity for independent geometries...
Computing and plotting magnitude sensitivity for Stations geometry...
Computation time: 0.21479272842407227 seconds
Computing and plotting magnitude sensitivity for DAS_1 geometry...
Computation time: 0.3876361846923828 seconds
Computing and plotting magnitude sensitivity for DAS_2 geometry...
/home/runner/work/PyNetDesign/PyNetDesign/pynetdesign/visualization/sensitivity.
↪py:115: RuntimeWarning: More than 20 figures have been opened. Figures created␣
↪through the pyplot interface (`matplotlib.pyplot.figure`) are retained until␣
↪explicitly closed and may consume too much memory. (To control this warning, see␣
↪the rcParam `figure.max_open_warning`). Consider using `matplotlib.pyplot.close()`.
  fig, ax = plt.subplots(figsize=(10, 10))
Computation time: 0.48102450370788574 seconds
```

3.4.5 Compute and plot magnitude sensitivity for combined geometry

```
print("Computing and plotting magnitude sensitivity for combined geometry...")

start_time = time()
for wave_mode in ['P','S','PS']:
    X_slice_mag_sens_comb = pndmod.core.get_mag_sensitivity(grid_coords=X_plane_
↪points,
                                                            geometry_df=combined_
↪geometry_df,
                                                            velocity_df=velocity_
↪df,
                                                            params=params,
                                                            wave_mode=wave_mode)
    Y_slice_mag_sens_comb = pndmod.core.get_mag_sensitivity(grid_coords=Y_plane_
↪points,
                                                            geometry_df=combined_
↪geometry_df,
                                                            velocity_df=velocity_
↪df,
                                                            params=params,
                                                            wave_mode=wave_mode)
    Z_slice_mag_sens_comb = pndmod.core.get_mag_sensitivity(grid_coords=Z_plane_
↪points,
                                                            geometry_df=combined_
↪geometry_df,
                                                            velocity_df=velocity_
↪df,
                                                            params=params,
                                                            wave_mode=wave_mode)

    _ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=X_slice_mag_sens_comb,␣
↪yi=gy, zi=gz,
                                                  plot_title=f"Magnitude sensitivity␣
↪for combined geometry at X={slice_X} m: YZ view, {wave_mode}",
                                                  clb_title="$M_w$",
                                                  geometry_df=combined_geometry_df.
↪recover(geometry),
                                                  plot_names = True if 'Name' in␣
↪combined_geometry_df.recover(geometry).columns else False,
                                                  station_marker_size = 100 if 'gauge_
↪length' not in combined_geometry_df.recover(geometry).attrs else 10)

    _ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=Y_slice_mag_sens_comb,␣
↪xi=gx, zi=gz,
                                                  plot_title=f"Magnitude sensitivity␣
↪for combined geometry at Y= {slice_Y} m: XZ view, {wave_mode}",
                                                  clb_title="$M_w$",
                                                  geometry_df=combined_geometry_df.
↪recover(geometry),
                                                  plot_names = True if 'Name' in␣
↪combined_geometry_df.recover(geometry).columns else False,
                                                  station_marker_size = 100 if 'gauge_
```

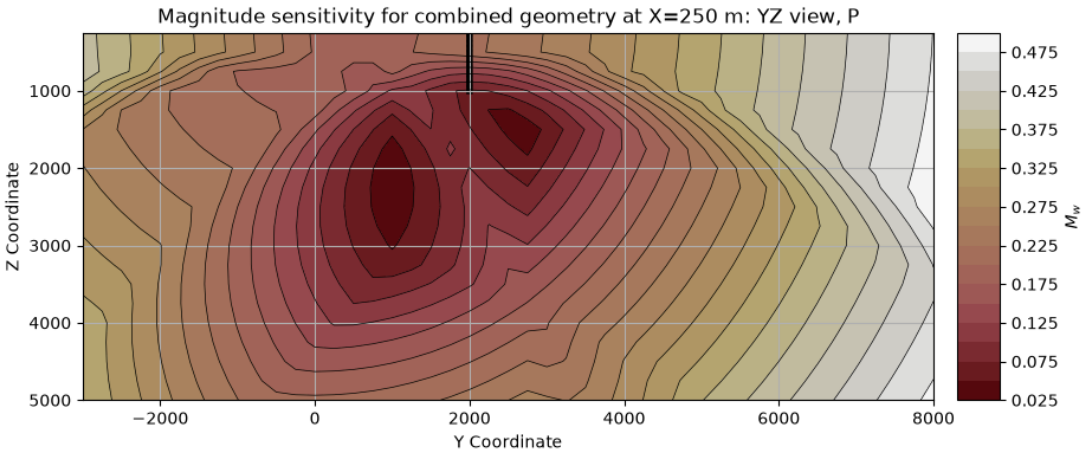
(continues on next page)

(continued from previous page)

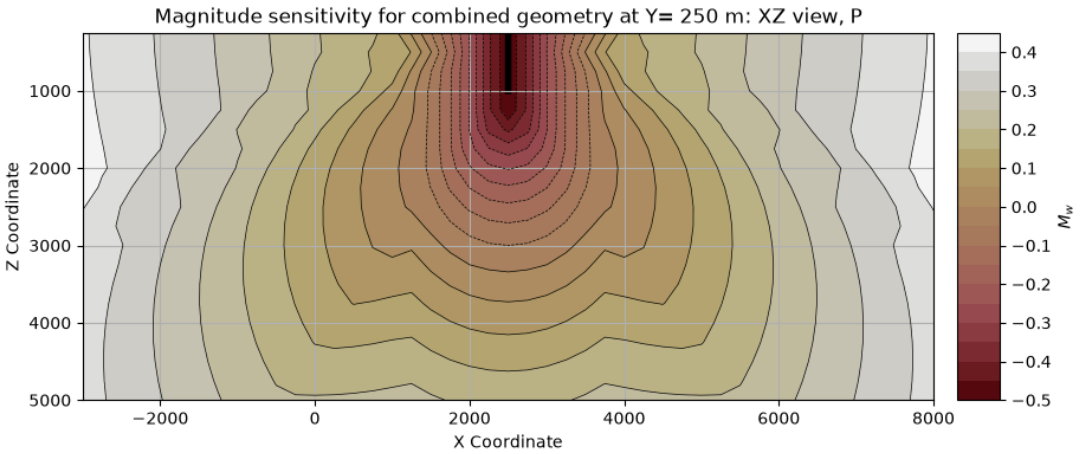
```
↪length' not in combined_geometry_df.recover(geometry).attrs else 10)

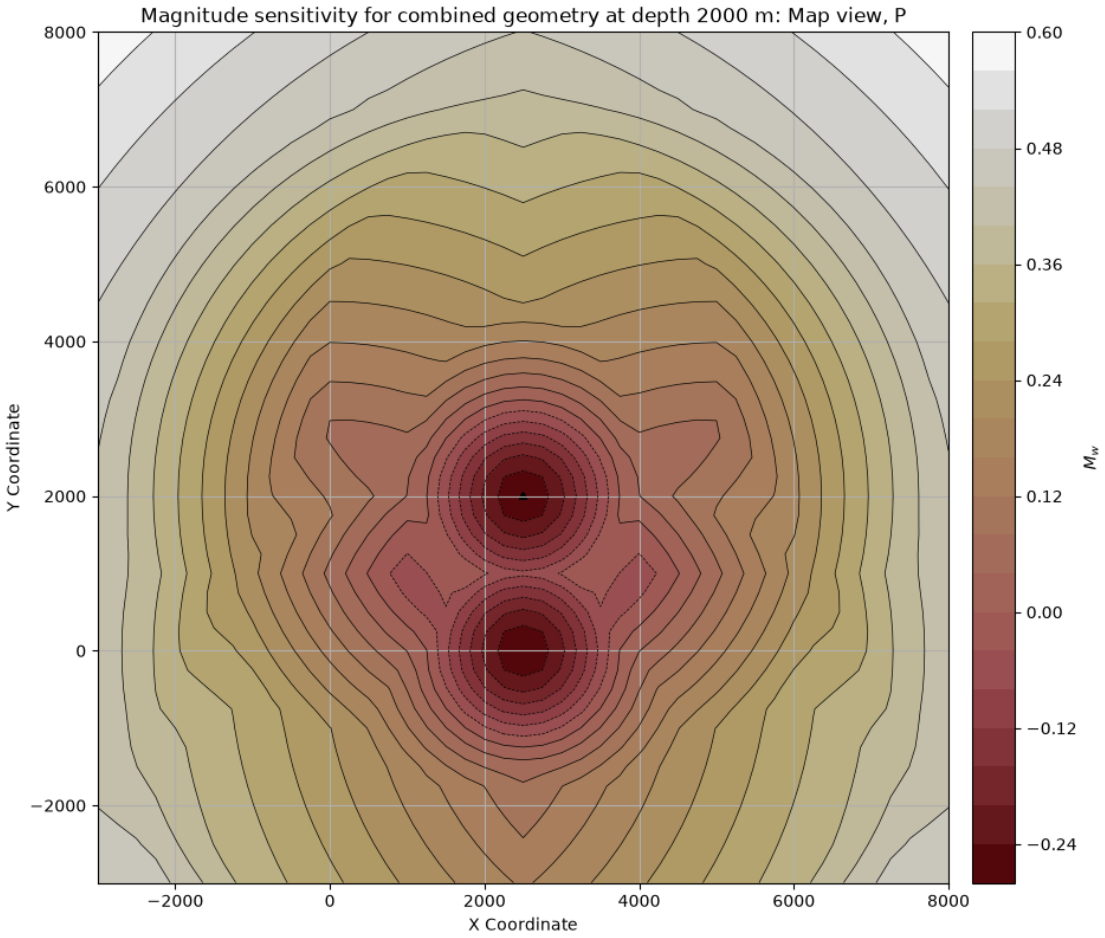
    _ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=Z_slice_mag_sens_comb,␣
↪xi=gx, yi=gy,
                                                  plot_title=f"Magnitude sensitivity␣
↪for combined geometry at depth {slice_Z} m: Map view, {wave_mode}",
                                                  clb_title="$M_w$",
                                                  geometry_df=combined_geometry_df.
↪recover(geometry),
                                                  plot_names = True if 'Name' in␣
↪combined_geometry_df.recover(geometry).columns else False,
                                                  station_marker_size = 100 if 'gauge_
↪length' not in combined_geometry_df.recover(geometry).attrs else 10)

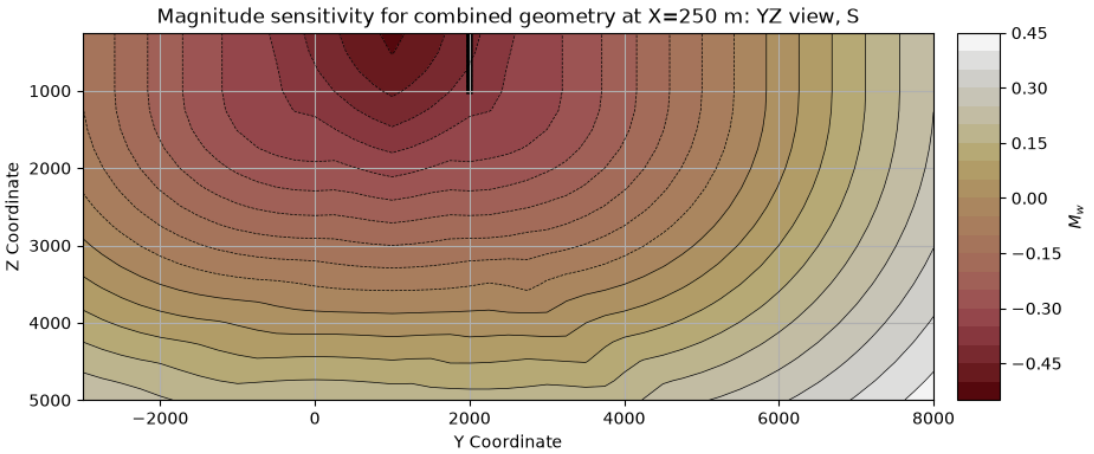
end_time = time()
print(f"Computation time: {end_time - start_time} seconds")
```

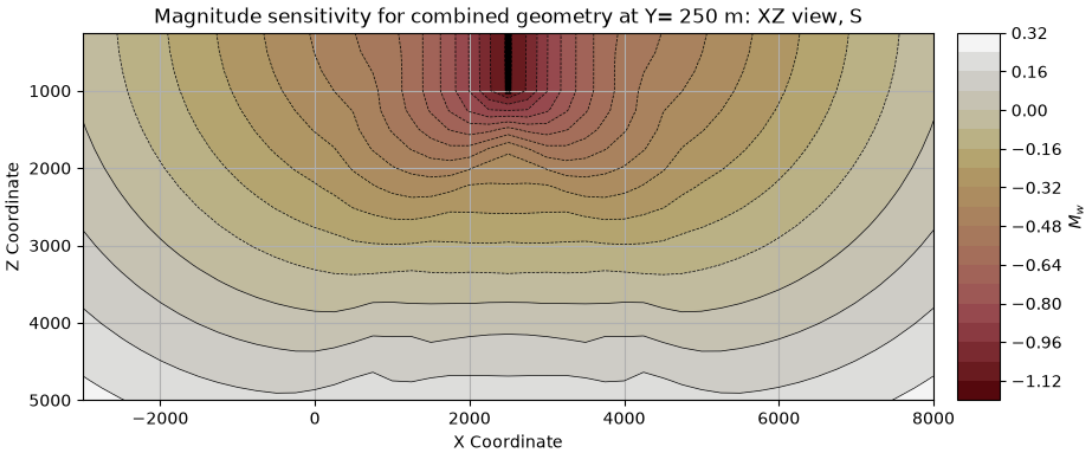

Magnitude sensitivity for combined geometry at X=250 m: YZ view, P
Z Coordinate
1000
2000
3000
4000
5000
−2000
0
2000
4000
6000
8000
Y Coordinate
0.475
0.425
0.375
0.325
0.275
0.225
0.175
0.125
0.075
0.025
M_w

•

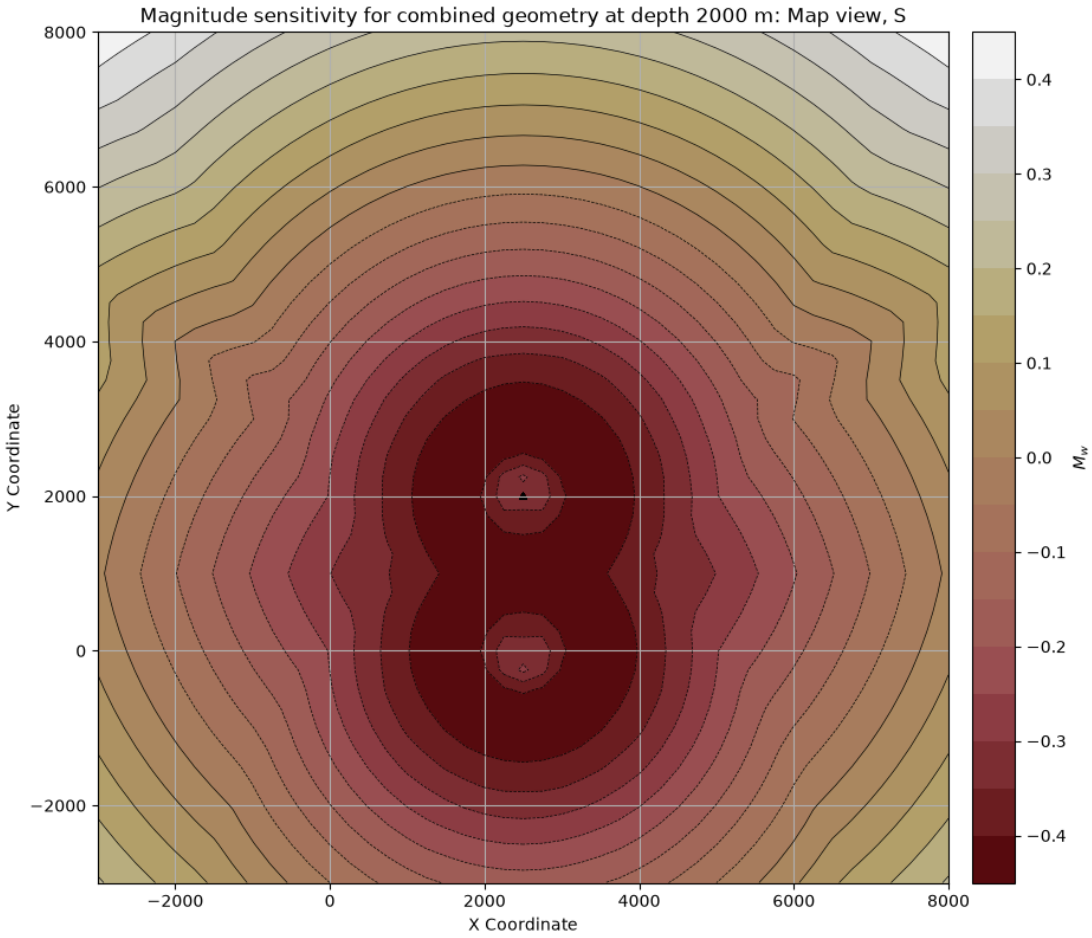

Magnitude sensitivity for combined geometry at Y= 250 m: XZ view, P
Z Coordinate
1000
2000
3000
4000
5000
−2000
0
2000
4000
6000
8000
X Coordinate
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
−0.5
M_w

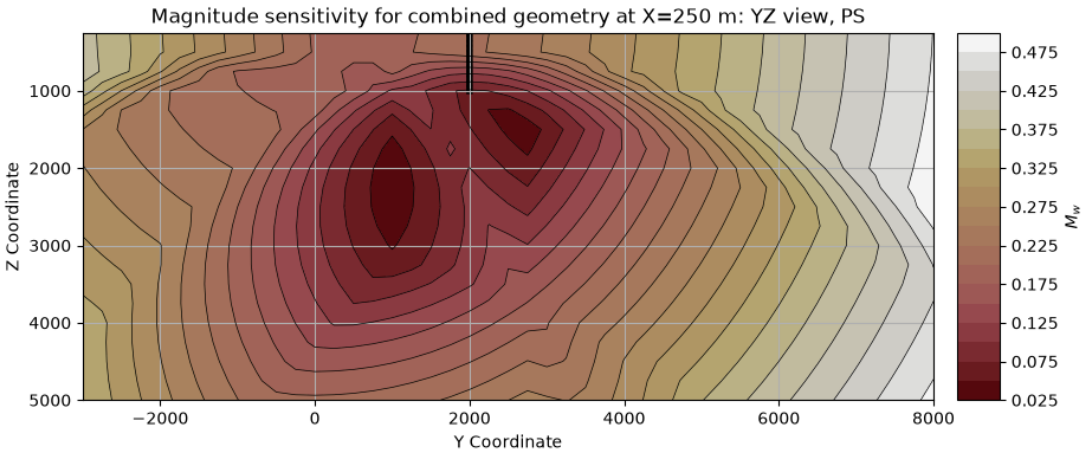

Magnitude sensitivity for combined geometry at depth 2000 m: Map view, P
Y Coordinate
X Coordinate
8000
6000
4000
2000
0
−2000
−2000
0
2000
4000
6000
8000
0.60
0.48
0.36
0.24
0.12
0.00
−0.12
−0.24
M_w


Magnitude sensitivity for combined geometry at X=250 m: YZ view, S
Z Coordinate
1000
2000
3000
4000
5000
−2000
0
2000
4000
6000
8000
Y Coordinate
0.45
0.30
0.15
0.00
−0.15
−0.30
−0.45
M_w

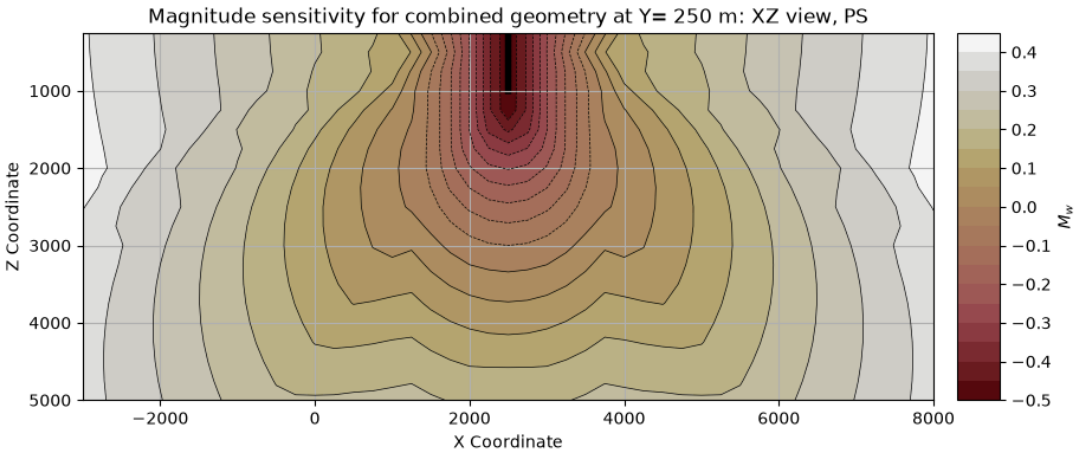

Magnitude sensitivity for combined geometry at Y= 250 m: XZ view, S
Z Coordinate
X Coordinate
1000
2000
3000
4000
5000
−2000
0
2000
4000
6000
8000
0.32
0.16
0.00
−0.16
−0.32
−0.48
−0.64
−0.80
−0.96
−1.12
M_w

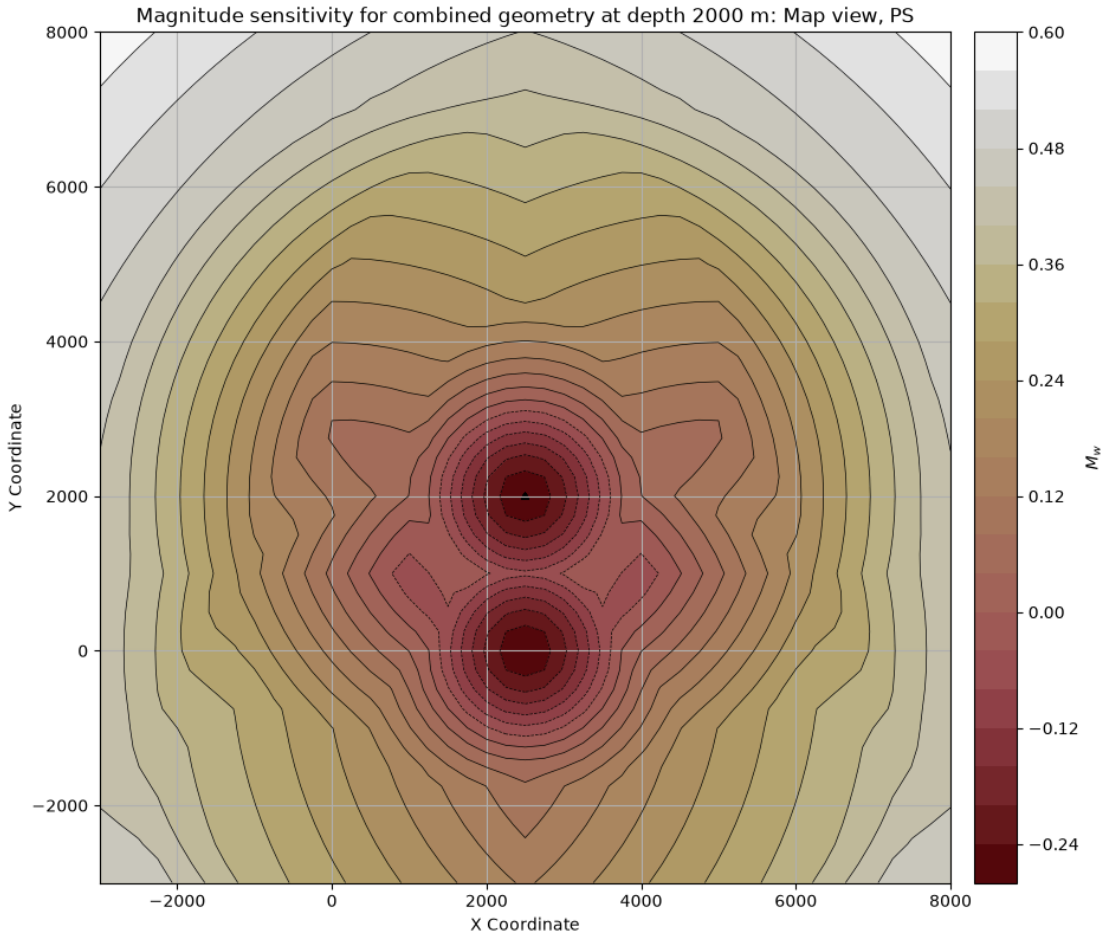
•


Magnitude sensitivity for combined geometry at depth 2000 m: Map view, S
Y Coordinate
X Coordinate
8000
6000
4000
2000
0
−2000
−2000
0
2000
4000
6000
8000
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
M_w


Magnitude sensitivity for combined geometry at X=250 m: YZ view, PS
Z Coordinate
1000
2000
3000
4000
5000
−2000
0
2000
4000
6000
8000
Y Coordinate
0.475
0.425
0.375
0.325
0.275
0.225
0.175
0.125
0.075
0.025
M_w

•


Magnitude sensitivity for combined geometry at Y= 250 m: XZ view, PS
Z Coordinate
1000
2000
3000
4000
5000
−2000
0
2000
4000
6000
8000
X Coordinate
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
−0.5
M_w




```
Computing and plotting magnitude sensitivity for combined geometry...
Computation time: 0.7384374141693115 seconds
```

Total running time of the script: (0 minutes 6.533 seconds)

3.5 05.1 Magnitude sensitivity for homogeneous model: test case 5

This is an example of computing magnitude sensitivity for a homogeneous model.

Test case 5:

- Geometry: DAS cable in an inclined borehole
- Velocity model: homogeneous, Vp = 4300 m/s, Vp/Vs = 1.73, Qp = 100, Qs = 50, rho = 2300 kg/m^3
- Processing: magnitude sensitivity computation for all stations (for grid slices only).

3.5.1 Load all necessary packages

```
import pynetdesign.visualization as pndvis
import pynetdesign.modelling as pndmod
import numpy as np
from time import time

# sphinx_gallery_thumbnail_number = -2
```

3.5.2 Input

Read geometry

```
geometry_file_path = '../data/homo/test_05/TMPL_net_geometry_Inclined.txt'
geometry_df = pndmod.io.read_geometry(geometry_file_path)

print(geometry_df)
print(geometry_df.attrs)
```

```
      Y      X       Z    NoiseLevel
0   0.0    0.0    10.0  5.000000e-11
1   0.0   10.0    60.0  5.000000e-11
2   0.0   20.0   110.0  5.000000e-11
3   0.0   30.0   160.0  5.000000e-11
4   0.0   40.0   210.0  5.000000e-11
..  ...    ...     ...           ...
56  0.0  560.0  2810.0  5.000000e-11
57  0.0  570.0  2860.0  5.000000e-11
58  0.0  580.0  2910.0  5.000000e-11
59  0.0  590.0  2960.0  5.000000e-11
60  0.0  600.0  3010.0  5.000000e-11

[61 rows x 4 columns]
{'file_path': '../data/homo/test_05/TMPL_net_geometry_Inclined.txt', 'original_coord_
↪units': 'm', 'coordinate_system': 'local', 'noise_type': 'strain', 'noise_unit': 'ε
↪', 'gauge_length': 10.0, '_is_combined': False}
```

Read velocity model

```
velocity_file_path = '../data/homo/test_05/TMPL_velocity_model.txt'
velocity_df = pndmod.io.read_velocity(velocity_file_path)

print(velocity_df)
print(velocity_df.attrs)
```

```
   Depth    Vp  VpVsRatio   Qp  Qs   Rho   Ep   Dt   Gm
0  40000  4300       1.73  100  50  2300  0.0  0.0  0.0
{'model_type': 'homo', 'model_anisotropy': 'isotropic', 'parameter_descriptions': {
↪'Depth': 'Depth', 'Vp': 'P-wave velocity', 'Vs': 'S-wave velocity', 'VpVsRatio':
↪'Vp/Vs', 'Qp': 'P-wave quality factor', 'Qs': 'S-wave quality factor', 'Rho':
↪'density', 'Ep': 'Thomsen epsilon parameter', 'Dt': 'Thomsen delta parameter', 'Gm
↪': 'Thomsen gamma parameter'}, 'parameter_units': {'Depth': 'm', 'Vp': 'm/s', 'Vs':
```

(continues on next page)

(continued from previous page)

```
↪'m/s', 'VpVsRatio': '', 'Qp': '', 'Qs': '', 'Rho': 'kg/m^3', 'Ep': '', 'Dt': '', 'Gm
↪': ''}}
```

Set imaging grid parameters

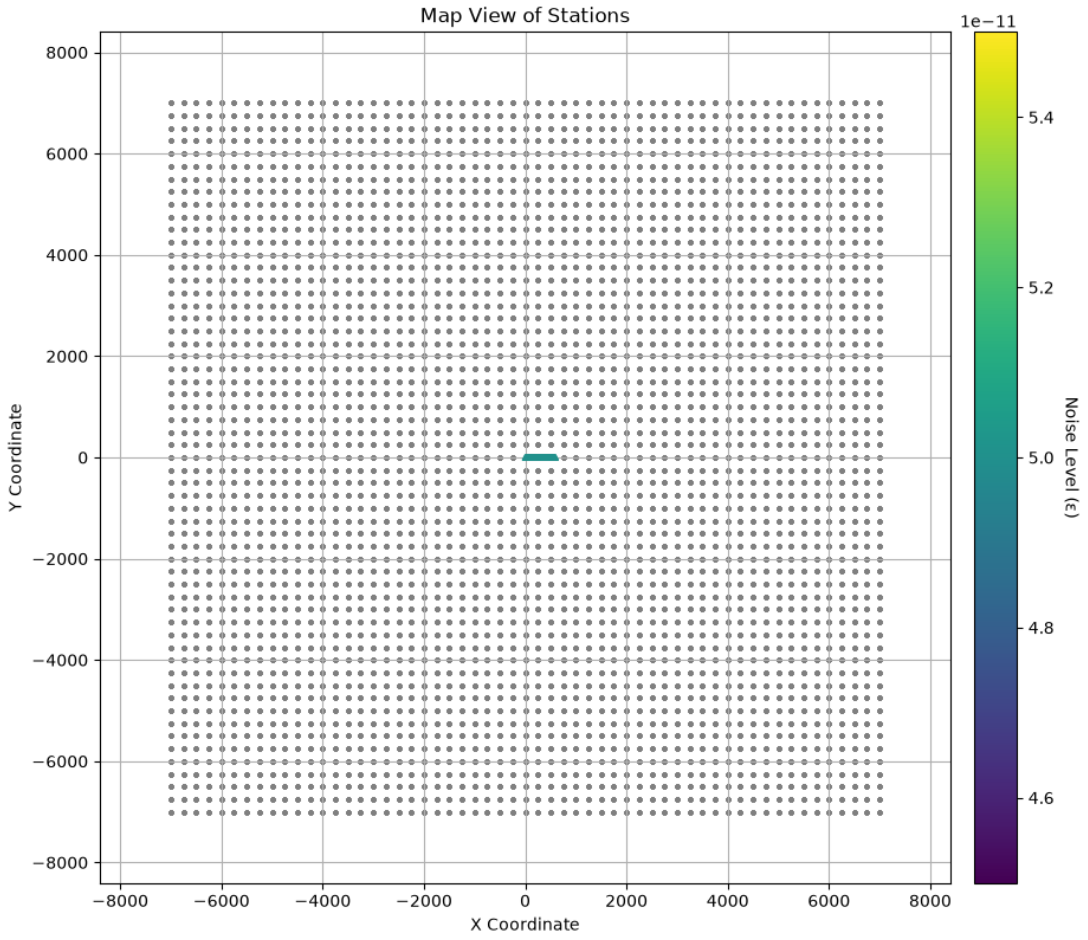
```
# Imaging grid parameters
gx = np.arange(-7000, 7000 + 250, 250)
gy = np.arange(-7000, 7000 + 250, 250)
gz = np.arange(250, 5000 + 250, 250)

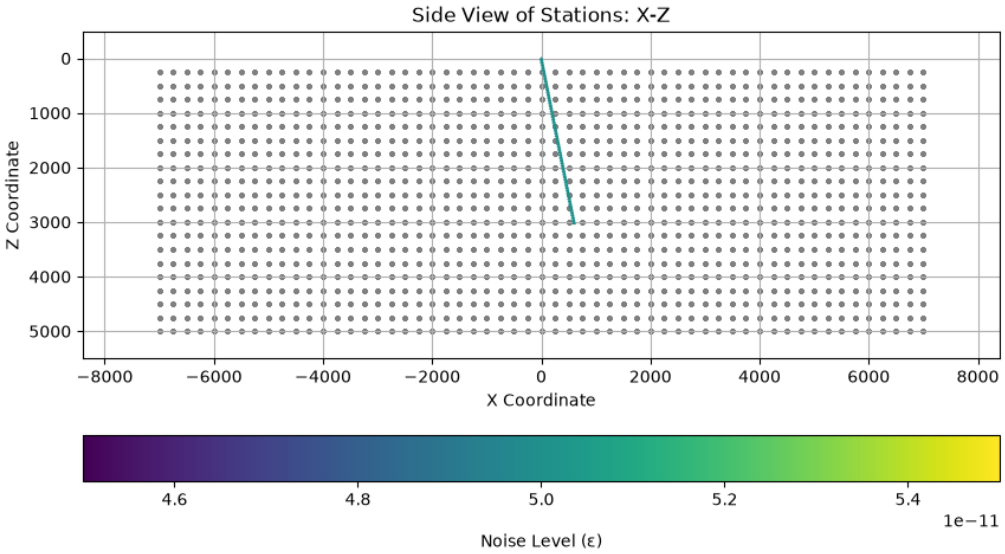
# Generate grid points
grid_points = pndmod.utils.generate_grid(x=gx,y=gy,z=gz)
print("Number of grid points:",grid_points.size)
print("Number of grid points in x:",gx.size)
print("Number of grid points in y:",gy.size)
print("Number of grid points in z:",gz.size)
print("grid_points.shape:",grid_points.shape)
```

```
Number of grid points: 194940
Number of grid points in x: 57
Number of grid points in y: 57
Number of grid points in z: 20
grid_points.shape: (64980, 3)
```

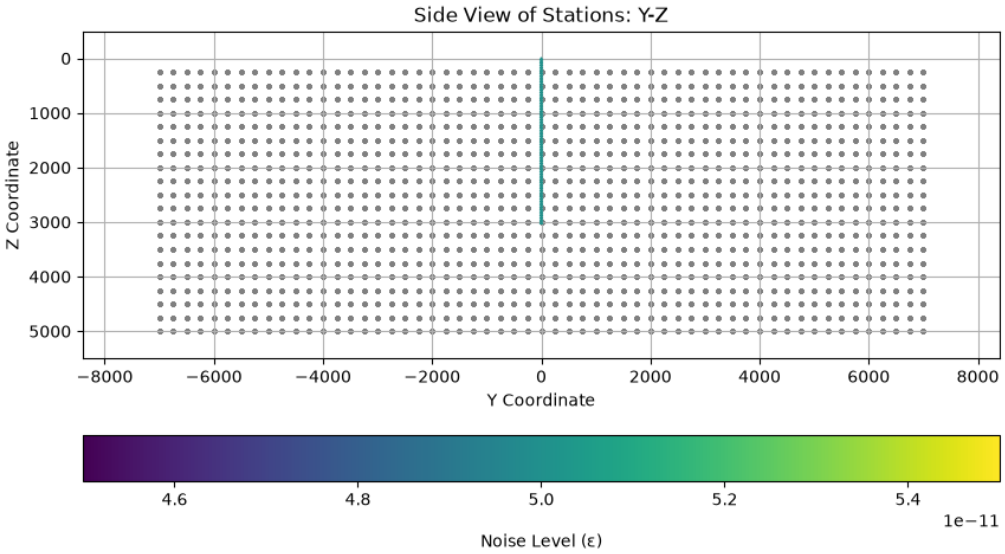
Plot geometry and imaging grid

```
_ = pndvis.geometry.plot_stations_view(geometry_df,
                                       view='xy',
                                       grid_points=grid_points,
                                       grid_marker_size=1,
                                       plot_names=False,
                                       station_marker_size=10)
_ = pndvis.geometry.plot_stations_view(geometry_df,
                                       view='xz',
                                       grid_points=grid_points,
                                       grid_marker_size=1,
                                       plot_names=False,
                                       station_marker_size=2)
_ = pndvis.geometry.plot_stations_view(geometry_df,
                                       view='yz',
                                       grid_points=grid_points,
                                       grid_marker_size=1,
                                       plot_names=False,
                                       station_marker_size=2)
```


Map View of Stations
1e−11
5.4
5.2
5.0
4.8
4.6
Noise Level (ε)
8000
6000
4000
2000
0
−2000
−4000
−6000
−8000
Y Coordinate
−8000
−6000
−4000
−2000
0
2000
4000
6000
8000
X Coordinate


Side View of Stations: X-Z
Z Coordinate
0
1000
2000
3000
4000
5000
−8000
−6000
−4000
−2000
0
2000
4000
6000
8000
X Coordinate
4.6
4.8
5.0
5.2
5.4
1e−11
Noise Level (ε)

•



-

3.5.3 Prepare to compute magnitude sensitivity

```
# Specify source parameters
frequency = 10 # Hz

# Minimum S/N for detection on individual channel
min_SNR_p = 2
min_SNR_s = 2

# Minimum stations on which an event must be detected
min_stations_p = 3
min_stations_s = 3

# Save parameters
params = pndmod.classes.DetectionParameters(
```

(continues on next page)

(continued from previous page)

```
    f_p=frequency,
    f_s=frequency,
    min_SNR_p=min_SNR_p,
    min_SNR_s=min_SNR_s,
    min_stations_p=min_stations_p,
    min_stations_s=min_stations_s,
)

# Define slices to use
slice_X = 0
slice_Y = 0
slice_Z = 2500

# Get slice planes
X_plane_points, X_plane_indices = pndmod.utils.get_slice_coordinates(grid_points=grid_
↪points,slice_dimension=0,slice_coordinate=slice_X)
Y_plane_points, Y_plane_indices = pndmod.utils.get_slice_coordinates(grid_points=grid_
↪points,slice_dimension=1,slice_coordinate=slice_Y)
Z_plane_points, Z_plane_indices = pndmod.utils.get_slice_coordinates(grid_points=grid_
↪points,slice_dimension=2,slice_coordinate=slice_Z)
```

3.5.4 Compute magnitude sensitivity

```
print("Computing magnitude sensitivity...")


start_time = time()
X_slice_mag_sens_p = pndmod.core.get_mag_sensitivity(grid_coords=X_plane_points,
                                                     geometry_df=geometry_df,
                                                     velocity_df=velocity_df,
                                                     params=params,
                                                     wave_mode='P')
X_slice_mag_sens_s = pndmod.core.get_mag_sensitivity(grid_coords=X_plane_points,
                                                     geometry_df=geometry_df,
                                                     velocity_df=velocity_df,
                                                     params=params,
                                                     wave_mode='S')
X_slice_mag_sens_ps = pndmod.core.get_mag_sensitivity(grid_coords=X_plane_points,
                                                      geometry_df=geometry_df,
                                                      velocity_df=velocity_df,
                                                      params=params,
                                                      wave_mode='PS')
Y_slice_mag_sens_p = pndmod.core.get_mag_sensitivity(grid_coords=Y_plane_points,
                                                     geometry_df=geometry_df,
                                                     velocity_df=velocity_df,
                                                     params=params,
                                                     wave_mode='P')
Y_slice_mag_sens_s = pndmod.core.get_mag_sensitivity(grid_coords=Y_plane_points,
                                                     geometry_df=geometry_df,
                                                     velocity_df=velocity_df,
                                                     params=params,
```

(continues on next page)

(continued from previous page)

```
                                                        wave_mode='S')
Y_slice_mag_sens_ps = pndmod.core.get_mag_sensitivity(grid_coords=Y_plane_points,
                                                       geometry_df=geometry_df,
                                                       velocity_df=velocity_df,
                                                       params=params,
                                                       wave_mode='PS')
Z_slice_mag_sens_p = pndmod.core.get_mag_sensitivity(grid_coords=Z_plane_points,
                                                     geometry_df=geometry_df,
                                                     velocity_df=velocity_df,
                                                     params=params,
                                                     wave_mode='P')
Z_slice_mag_sens_s = pndmod.core.get_mag_sensitivity(grid_coords=Z_plane_points,
                                                     geometry_df=geometry_df,
                                                     velocity_df=velocity_df,
                                                     params=params,
                                                     wave_mode='S')
Z_slice_mag_sens_ps = pndmod.core.get_mag_sensitivity(grid_coords=Z_plane_points,
                                                       geometry_df=geometry_df,
                                                       velocity_df=velocity_df,
                                                       params=params,
                                                       wave_mode='PS')
end_time = time()
print(f"Computation time: {end_time - start_time} seconds")
```

```
Computing magnitude sensitivity...
Computation time: 0.29751157760620117 seconds
```

3.5.5 Plot magnitude sensitivity

Plot magnitude sensitivity for P wave

```
station_marker_size = 25
clb_range = None
plt_levels = None

_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= X_slice_mag_sens_p, yi=gy,␣
↪zi=gz,
                                              plot_title=f"Magnitude sensitivity at X=
↪{slice_X} m: YZ view, P-waves",
                                              clb_title="$M_w$",
                                              geometry_df=geometry_df,
                                              plot_names = False,
                                              station_marker_size = station_marker_
↪size,
                                              clb_range=clb_range,
                                              plt_levels=plt_levels)

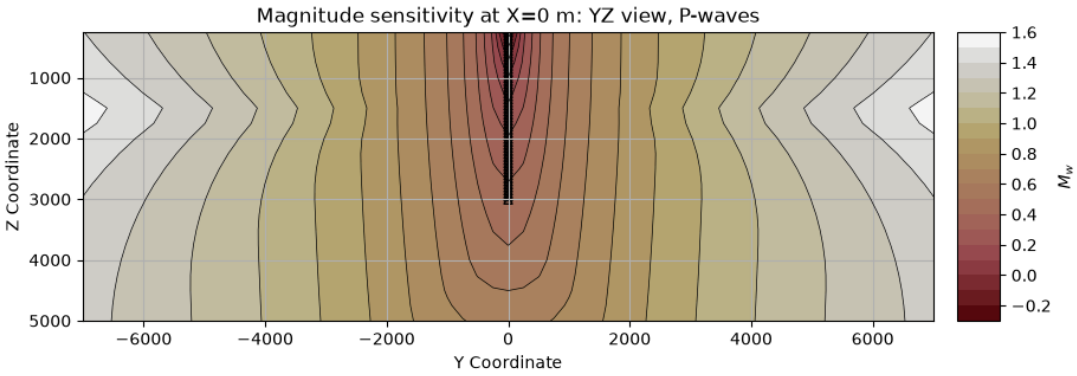
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= Y_slice_mag_sens_p, xi=gx,␣
↪zi=gz,
                                              plot_title=f"Magnitude sensitivity at Y=
↪{slice_Y} m: XZ view, P-waves",
```

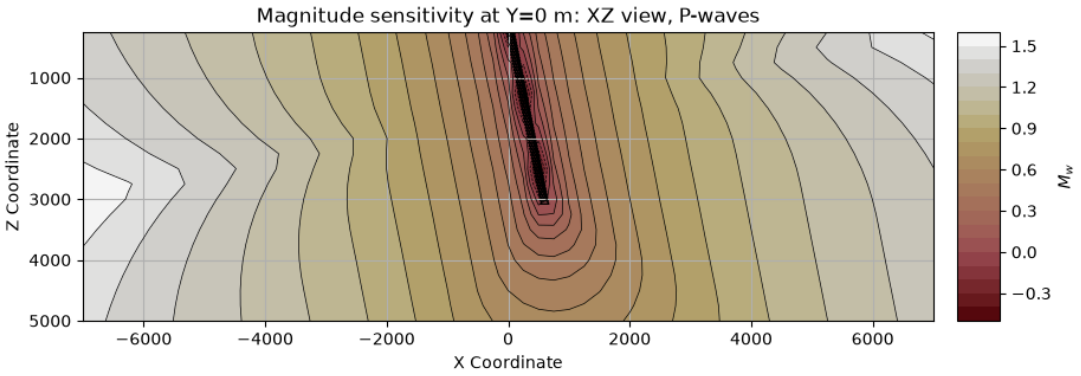
(continues on next page)

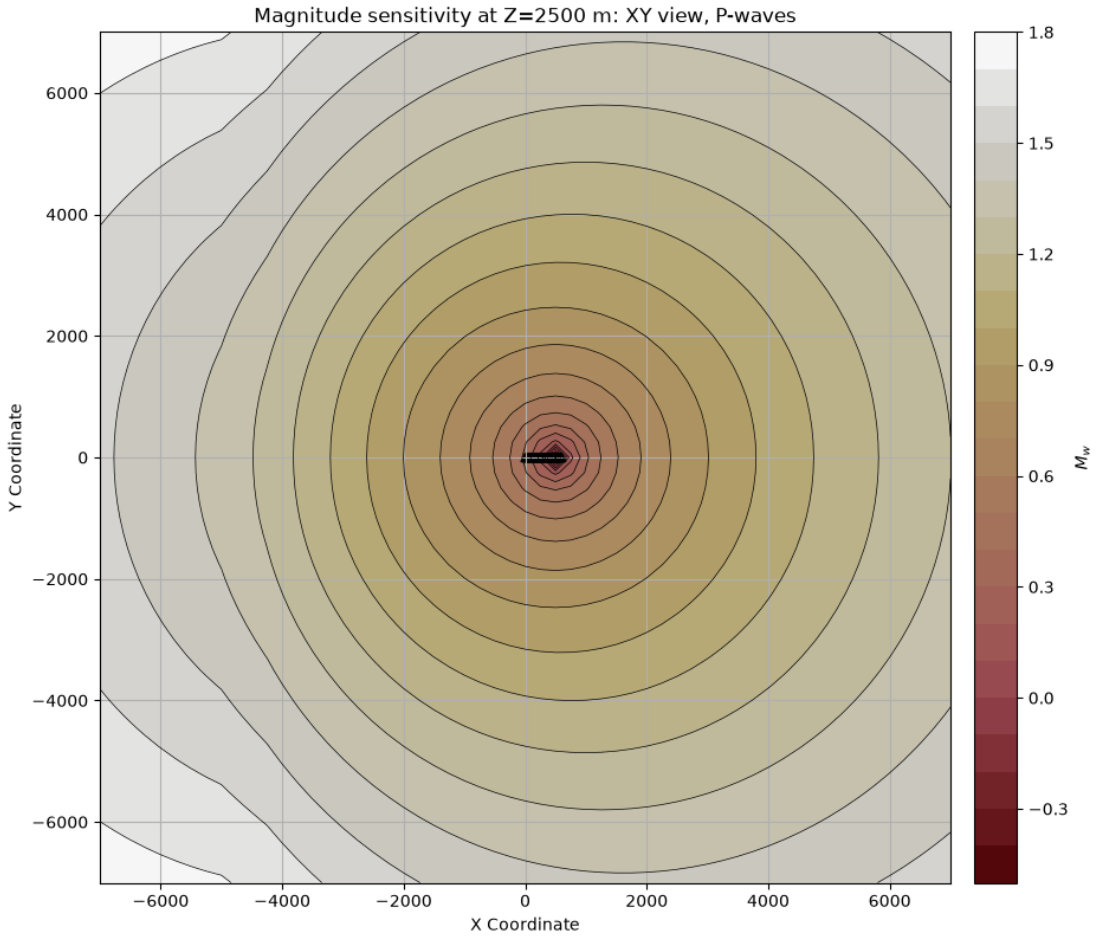
(continued from previous page)

```
                                                clb_title="$M_w$",
                                                geometry_df=geometry_df,
                                                plot_names = False,
                                                station_marker_size = station_marker_
↪size,
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)

_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= Z_slice_mag_sens_p, xi=gx,␣
↪yi=gy,
                                                plot_title=f"Magnitude sensitivity at Z=
↪{slice_Z} m: XY view, P-waves",
                                                clb_title="$M_w$",
                                                geometry_df=geometry_df,
                                                plot_names = False,
                                                station_marker_size = station_marker_
↪size,
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)
```


Magnitude sensitivity at X=0 m: YZ view, P-waves
Z Coordinate
1000
2000
3000
4000
5000
−6000
−4000
−2000
0
2000
4000
6000
Y Coordinate
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
M_w


Magnitude sensitivity at Y=0 m: XZ view, P-waves
Z Coordinate
1000
2000
3000
4000
5000
−6000
−4000
−2000
0
2000
4000
6000
X Coordinate
1.5
1.2
0.9
0.6
0.3
0.0
−0.3
M_w



-

Plot magnitude sensitivity for S wave

```
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= X_slice_mag_sens_s, yi=gy,␣
↪zi=gz,
                                              plot_title=f"Magnitude sensitivity at X=
↪{slice_X} m: YZ view, S-waves",
                                              clb_title="$M_w$",
                                              geometry_df=geometry_df,
                                              plot_names = False,
                                              station_marker_size = station_marker_
↪size,
                                              clb_range=clb_range,
                                              plt_levels=plt_levels)

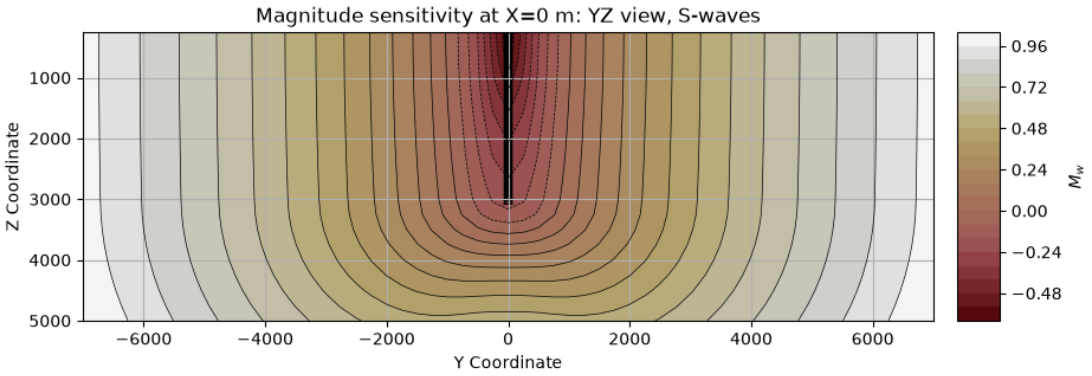
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= Y_slice_mag_sens_s, xi=gx,␣
```

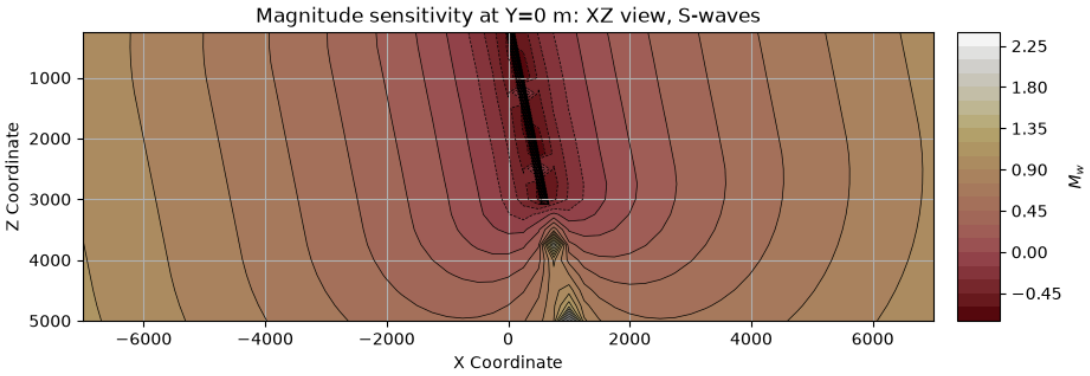
(continues on next page)

(continued from previous page)

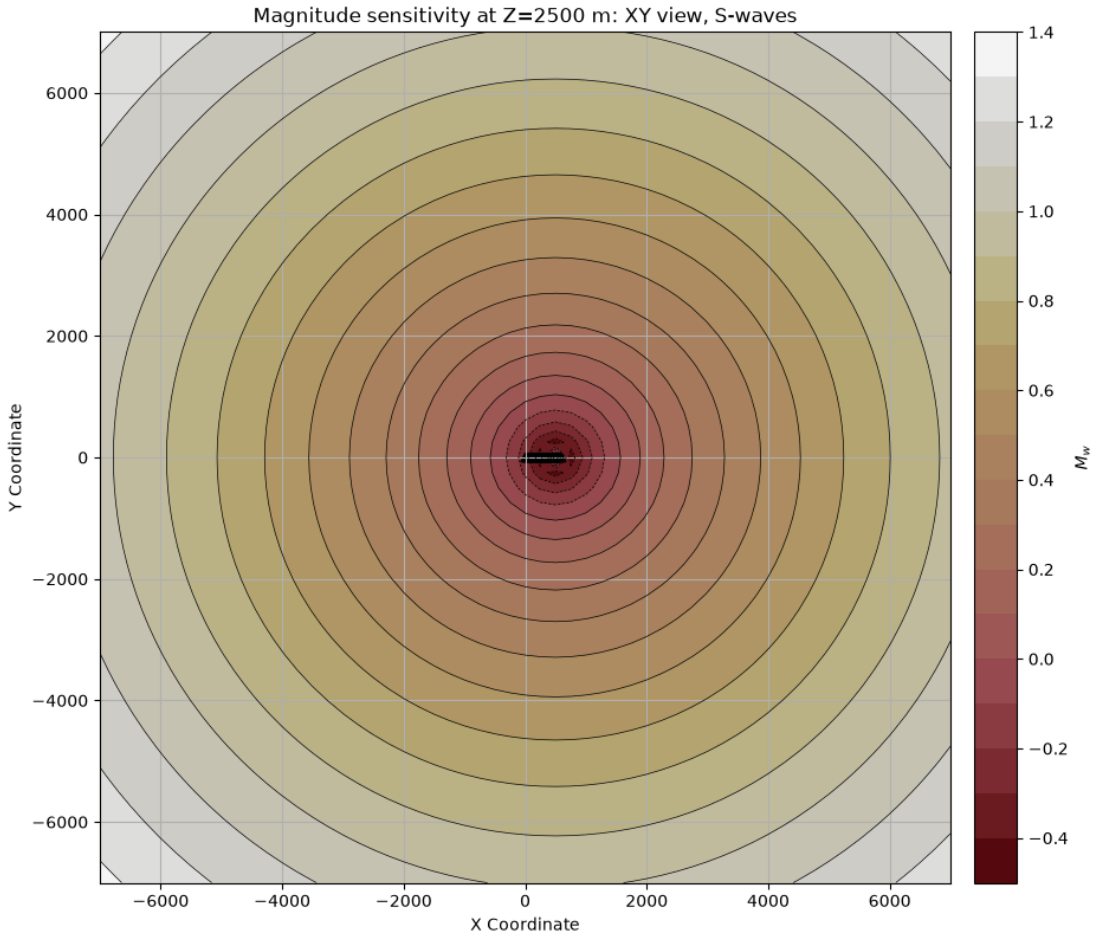
```
↪zi=gz,
                                                plot_title=f"Magnitude sensitivity at Y=
↪{slice_Y} m: XZ view, S-waves",
                                                clb_title="$M_w$",
                                                geometry_df=geometry_df,
                                                plot_names = False,
                                                station_marker_size = station_marker_
↪size,
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)

_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= Z_slice_mag_sens_s, xi=gx,␣
↪yi=gy,
                                                plot_title=f"Magnitude sensitivity at Z=
↪{slice_Z} m: XY view, S-waves",
                                                clb_title="$M_w$",
                                                geometry_df=geometry_df,
                                                plot_names = False,
                                                station_marker_size = station_marker_
↪size,
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)
```


Magnitude sensitivity at X=0 m: YZ view, S-waves
Z Coordinate
1000
2000
3000
4000
5000
−6000
−4000
−2000
0
2000
4000
6000
Y Coordinate
0.96
0.72
0.48
0.24
0.00
−0.24
−0.48
M_w


Magnitude sensitivity at Y=0 m: XZ view, S-waves
Z Coordinate
1000
2000
3000
4000
5000
−6000
−4000
−2000
0
2000
4000
6000
X Coordinate
2.25
1.80
1.35
0.90
0.45
0.00
−0.45
M_w

•



-

Plot magnitude sensitivity for P and S waves

```
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= X_slice_mag_sens_ps, yi=gy,␣
↪zi=gz,
                                              plot_title=f"Magnitude sensitivity at X=
↪{slice_X} m: YZ view, P- and S-waves",
                                              clb_title="$M_w$",
                                              geometry_df=geometry_df,
                                              plot_names = False,
                                              station_marker_size = station_marker_
↪size,
                                              clb_range=clb_range,
                                              plt_levels=plt_levels)

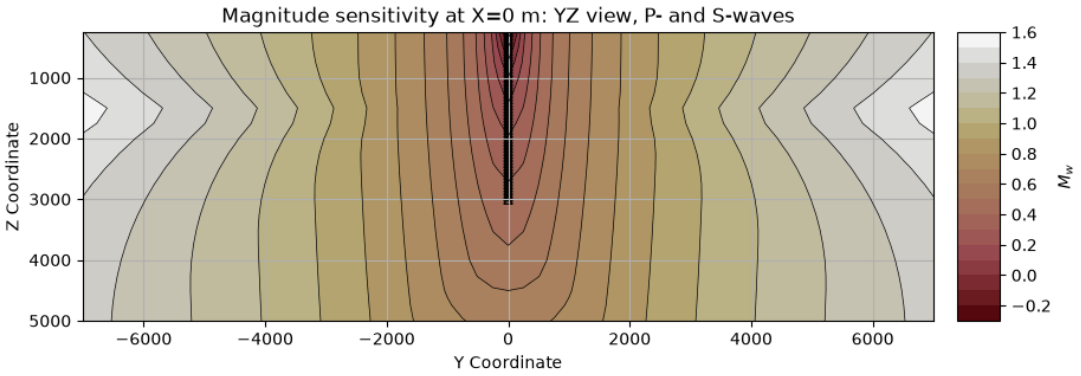
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= Y_slice_mag_sens_ps, xi=gx,␣
```

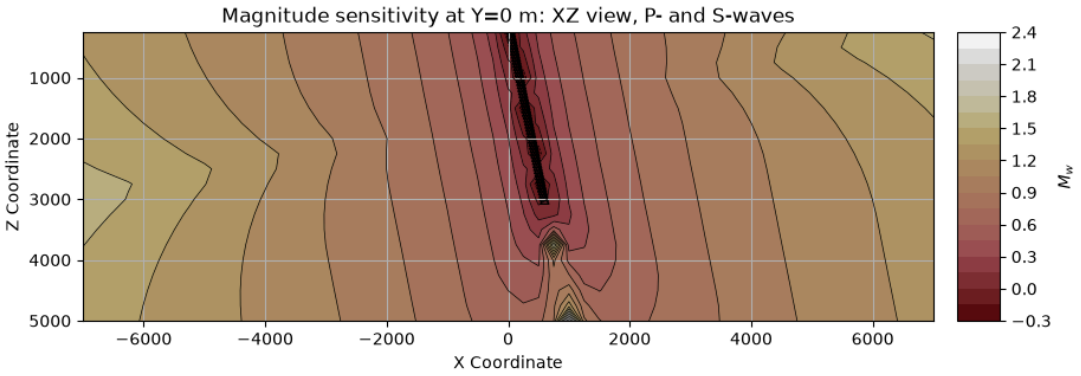
(continues on next page)

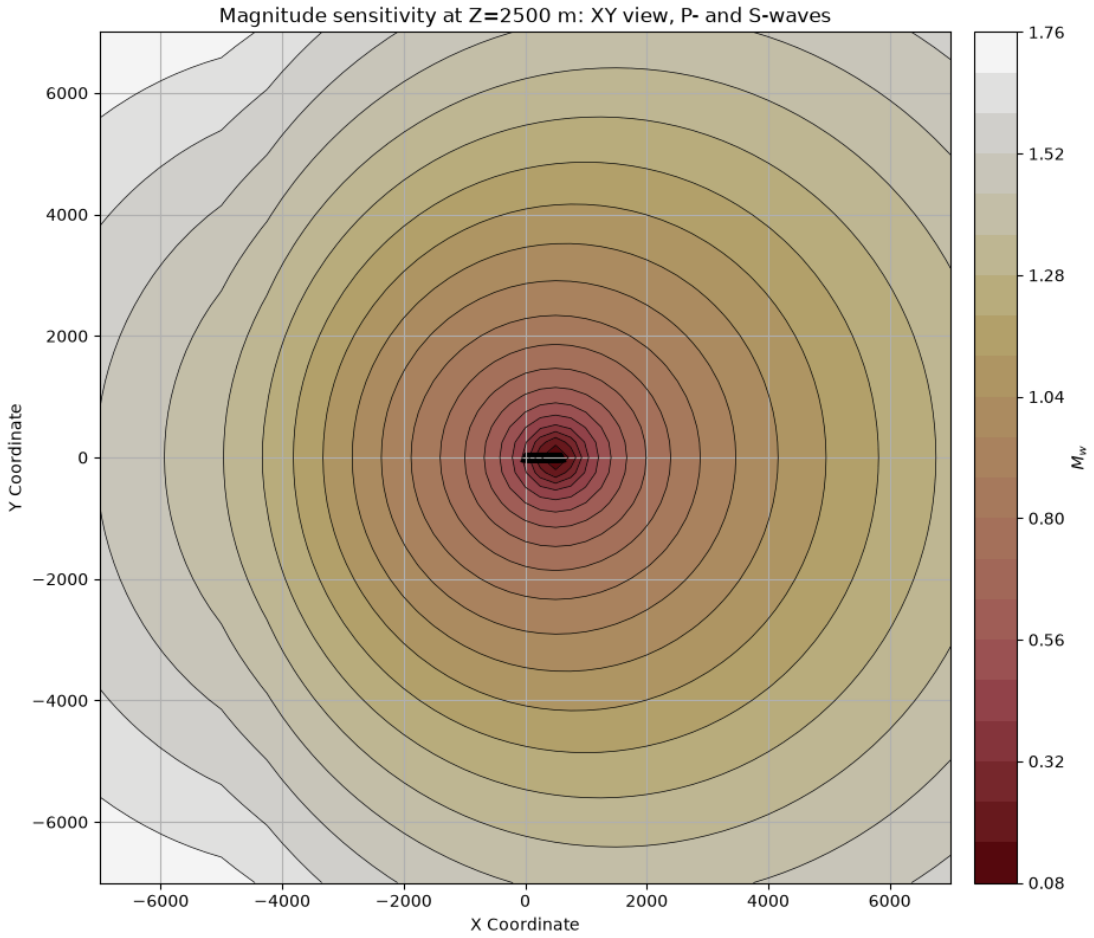
(continued from previous page)

```
↪zi=gz,
                                                plot_title=f"Magnitude sensitivity at Y=
↪{slice_Y} m: XZ view, P- and S-waves",
                                                clb_title="$M_w$",
                                                geometry_df=geometry_df,
                                                plot_names = False,
                                                station_marker_size = station_marker_
↪size,
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)

_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= Z_slice_mag_sens_ps, xi=gx,␣
↪yi=gy,
                                                plot_title=f"Magnitude sensitivity at Z=
↪{slice_Z} m: XY view, P- and S-waves",
                                                clb_title="$M_w$",
                                                geometry_df=geometry_df,
                                                plot_names = False,
                                                station_marker_size = station_marker_
↪size,
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)
```


Magnitude sensitivity at X=0 m: YZ view, P- and S-waves
Z Coordinate
1000
2000
3000
4000
5000
−6000
−4000
−2000
0
2000
4000
6000
Y Coordinate
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
M_w


Magnitude sensitivity at Y=0 m: XZ view, P- and S-waves
Z Coordinate
X Coordinate
1000
2000
3000
4000
5000
−6000
−4000
−2000
0
2000
4000
6000
2.4
2.1
1.8
1.5
1.2
0.9
0.6
0.3
0.0
−0.3
M_w



-

Total running time of the script: (0 minutes 2.054 seconds)

3.6 06.1 Magnitude sensitivity for homogeneous model: test case 6

This is an example of computing magnitude sensitivity for a homogeneous model.

Test case 6:

- Geometry: DAS cable in a twice inclined borehole
- Velocity model: homogeneous, Vp = 4300 m/s, Vp/Vs = 1.73, Qp = 100, Qs = 50, rho = 2300 kg/m^3
- Processing: magnitude sensitivity computation for all stations (for grid slices only).

3.6.1 Load all necessary packages

```
import pynetdesign.visualization as pndvis
import pynetdesign.modelling as pndmod
import numpy as np
from time import time

# sphinx_gallery_thumbnail_number = -2
```

3.6.2 Input

Read geometry

```
geometry_file_path = '../data/homo/test_06/TMPL_net_geometry_DoubleInclined.txt'
geometry_df = pndmod.io.read_geometry(geometry_file_path)

print(geometry_df)
print(geometry_df.attrs)
```

```
      Y     X       Z    NoiseLevel
0   0.0   0.0    10.0  5.000000e-11
1   0.0  20.0    60.0  5.000000e-11
2   0.0  40.0   110.0  5.000000e-11
3   0.0  60.0   160.0  5.000000e-11
4   0.0  80.0   210.0  5.000000e-11
..  ...   ...     ...           ...
56  0.0  40.0  2810.0  5.000000e-11
57  0.0  20.0  2860.0  5.000000e-11
58  0.0   0.0  2910.0  5.000000e-11
59  0.0 -20.0  2960.0  5.000000e-11
60  0.0 -40.0  3010.0  5.000000e-11

[61 rows x 4 columns]
{'file_path': '../data/homo/test_06/TMPL_net_geometry_DoubleInclined.txt', 'original_
↪coord_units': 'm', 'coordinate_system': 'local', 'noise_type': 'strain', 'noise_unit
↪': 'ε', 'gauge_length': 10.0, '_is_combined': False}
```

Read velocity model

```
velocity_file_path = '../data/homo/test_06/TMPL_velocity_model.txt'
velocity_df = pndmod.io.read_velocity(velocity_file_path)

print(velocity_df)
print(velocity_df.attrs)
```

```
   Depth    Vp  VpVsRatio   Qp  Qs   Rho   Ep   Dt   Gm
0  40000  4300       1.73  100  50  2300  0.0  0.0  0.0
{'model_type': 'homo', 'model_anisotropy': 'isotropic', 'parameter_descriptions': {
↪'Depth': 'Depth', 'Vp': 'P-wave velocity', 'Vs': 'S-wave velocity', 'VpVsRatio':
↪'Vp/Vs', 'Qp': 'P-wave quality factor', 'Qs': 'S-wave quality factor', 'Rho':
↪'density', 'Ep': 'Thomsen epsilon parameter', 'Dt': 'Thomsen delta parameter', 'Gm
↪': 'Thomsen gamma parameter'}, 'parameter_units': {'Depth': 'm', 'Vp': 'm/s', 'Vs':
```

(continues on next page)

(continued from previous page)

```
↪'m/s', 'VpVsRatio': '', 'Qp': '', 'Qs': '', 'Rho': 'kg/m^3', 'Ep': '', 'Dt': '', 'Gm
↪': ''}}
```

Set imaging grid parameters

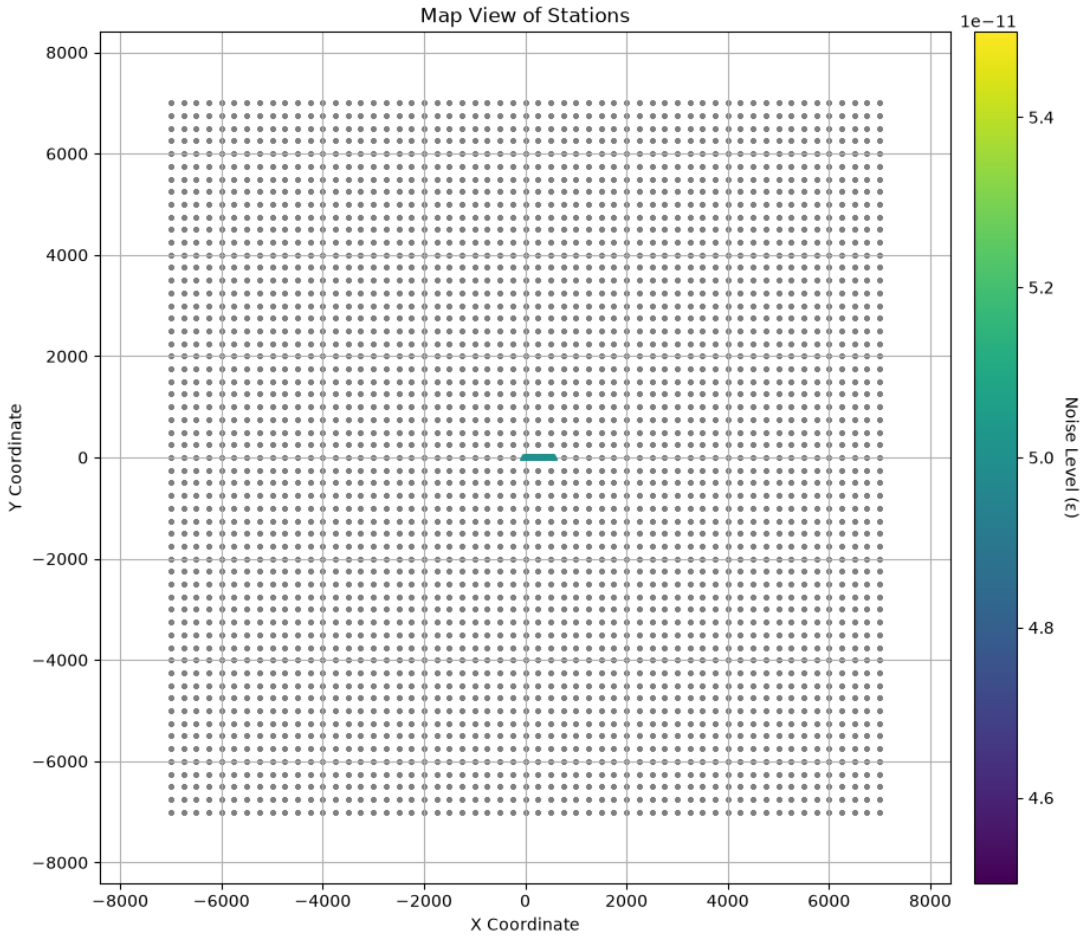
```
# Imaging grid parameters
gx = np.arange(-7000, 7000 + 250, 250)
gy = np.arange(-7000, 7000 + 250, 250)
gz = np.arange(250, 5000 + 250, 250)

# Generate grid points
grid_points = pndmod.utils.generate_grid(x=gx,y=gy,z=gz)
print("Number of grid points:",grid_points.size)
print("Number of grid points in x:",gx.size)
print("Number of grid points in y:",gy.size)
print("Number of grid points in z:",gz.size)
print("grid_points.shape:",grid_points.shape)
```

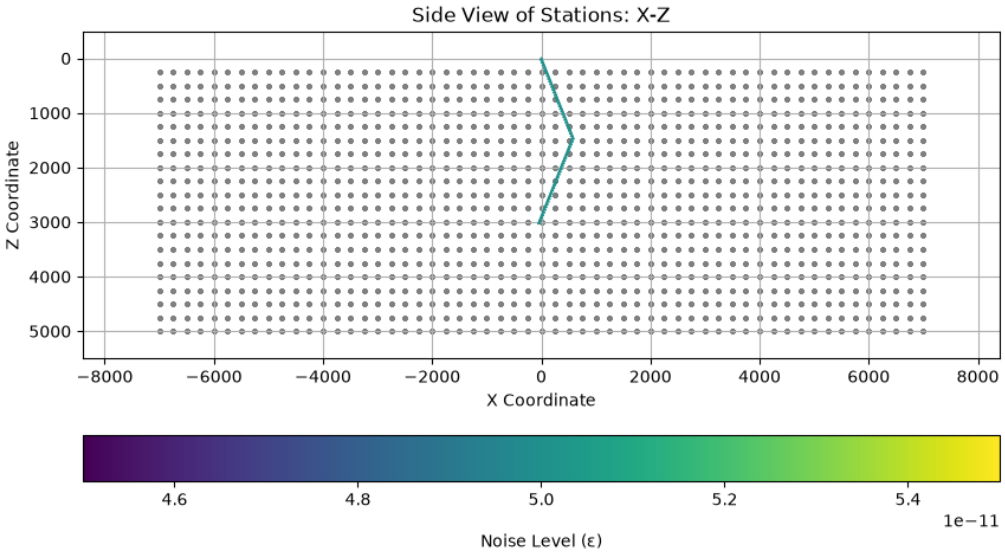
```
Number of grid points: 194940
Number of grid points in x: 57
Number of grid points in y: 57
Number of grid points in z: 20
grid_points.shape: (64980, 3)
```

Plot geometry and imaging grid

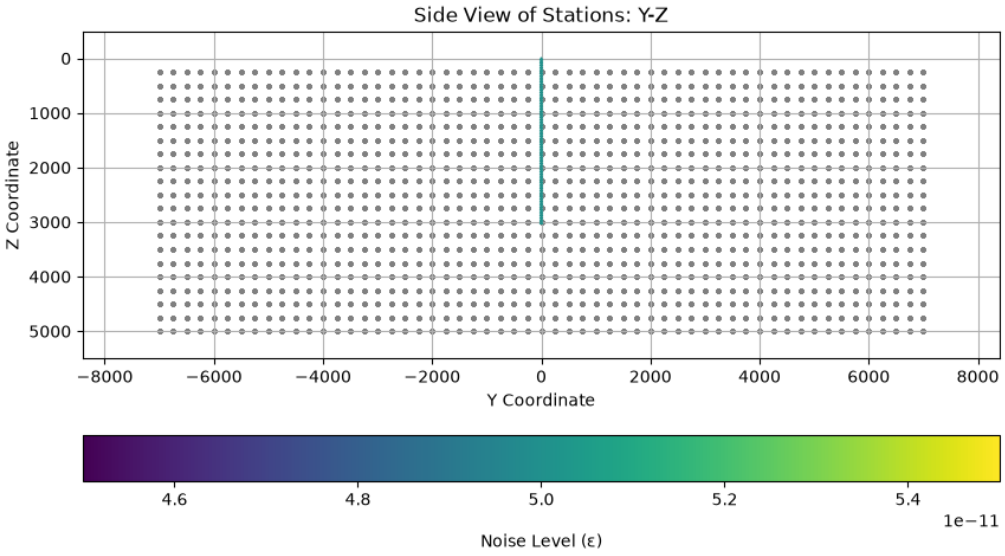
```
_ = pndvis.geometry.plot_stations_view(geometry_df,
                                       view='xy',
                                       grid_points=grid_points,
                                       grid_marker_size=1,
                                       plot_names=False,
                                       station_marker_size=10)
_ = pndvis.geometry.plot_stations_view(geometry_df,
                                       view='xz',
                                       grid_points=grid_points,
                                       grid_marker_size=1,
                                       plot_names=False,
                                       station_marker_size=2)
_ = pndvis.geometry.plot_stations_view(geometry_df,
                                       view='yz',
                                       grid_points=grid_points,
                                       grid_marker_size=1,
                                       plot_names=False,
                                       station_marker_size=2)
```


Map View of Stations
1e−11
5.4
5.2
5.0
4.8
4.6
Noise Level (ε)
8000
6000
4000
2000
0
−2000
−4000
−6000
−8000
Y Coordinate
−8000
−6000
−4000
−2000
0
2000
4000
6000
8000
X Coordinate

•


Side View of Stations: X-Z
Z Coordinate
0
1000
2000
3000
4000
5000
−8000
−6000
−4000
−2000
0
2000
4000
6000
8000
X Coordinate
4.6
4.8
5.0
5.2
5.4
1e−11
Noise Level (ε)

•



-

3.6.3 Prepare to compute magnitude sensitivity

```
# Get medium parameters
density = velocity_df.at[0,'Rho']
v_p = velocity_df.at[0,'Vp']
Q_p = velocity_df.at[0,'Qp']
v_s = velocity_df.at[0,'Vp']/velocity_df.at[0,'VpVsRatio']
Q_s = velocity_df.at[0,'Qs']
print("density, v_p, Q_p, v_s, Q_s:",density,v_p,Q_p,v_s,Q_s)

# Specify source parameters
frequency = 10 # Hz

# Minimum S/N for detection on individual channel
min_SNR_p = 2
```

(continues on next page)

(continued from previous page)

```
min_SNR_s = 2

# Minimum stations on which an event must be detected
min_stations_p = 3
min_stations_s = 3

# Save parameters
params = pndmod.classes.DetectionParameters(
    f_p=frequency,
    f_s=frequency,
    min_SNR_p=min_SNR_p,
    min_SNR_s=min_SNR_s,
    min_stations_p=min_stations_p,
    min_stations_s=min_stations_s,
)

# Define slices to use
slice_X = 0
slice_Y = 0
slice_Z = 2500

# Get slice planes
X_plane_points, X_plane_indices = pndmod.utils.get_slice_coordinates(grid_points=grid_
↪points,slice_dimension=0,slice_coordinate=slice_X)
Y_plane_points, Y_plane_indices = pndmod.utils.get_slice_coordinates(grid_points=grid_
↪points,slice_dimension=1,slice_coordinate=slice_Y)
Z_plane_points, Z_plane_indices = pndmod.utils.get_slice_coordinates(grid_points=grid_
↪points,slice_dimension=2,slice_coordinate=slice_Z)
```

```
density, v_p, Q_p, v_s, Q_s: 2300 4300 100 2485.549132947977 50
```

3.6.4 Compute magnitude sensitivity

```
print("Computing magnitude sensitivity...")


start_time = time()
X_slice_mag_sens_p = pndmod.core.get_mag_sensitivity(grid_coords=X_plane_points,
                                                     geometry_df=geometry_df,
                                                     velocity_df=velocity_df,
                                                     params=params,
                                                     wave_mode='P')
X_slice_mag_sens_s = pndmod.core.get_mag_sensitivity(grid_coords=X_plane_points,
                                                     geometry_df=geometry_df,
                                                     velocity_df=velocity_df,
                                                     params=params,
                                                     wave_mode='S')
X_slice_mag_sens_ps = pndmod.core.get_mag_sensitivity(grid_coords=X_plane_points,
                                                      geometry_df=geometry_df,
                                                      velocity_df=velocity_df,
                                                      params=params,
```

(continues on next page)

(continued from previous page)

```
                                                        wave_mode='PS')
Y_slice_mag_sens_p = pndmod.core.get_mag_sensitivity(grid_coords=Y_plane_points,
                                                     geometry_df=geometry_df,
                                                     velocity_df=velocity_df,
                                                     params=params,
                                                     wave_mode='P')
Y_slice_mag_sens_s = pndmod.core.get_mag_sensitivity(grid_coords=Y_plane_points,
                                                     geometry_df=geometry_df,
                                                     velocity_df=velocity_df,
                                                     params=params,
                                                     wave_mode='S')
Y_slice_mag_sens_ps = pndmod.core.get_mag_sensitivity(grid_coords=Y_plane_points,
                                                      geometry_df=geometry_df,
                                                      velocity_df=velocity_df,
                                                      params=params,
                                                      wave_mode='PS')
Z_slice_mag_sens_p = pndmod.core.get_mag_sensitivity(grid_coords=Z_plane_points,
                                                     geometry_df=geometry_df,
                                                     velocity_df=velocity_df,
                                                     params=params,
                                                     wave_mode='P')
Z_slice_mag_sens_s = pndmod.core.get_mag_sensitivity(grid_coords=Z_plane_points,
                                                     geometry_df=geometry_df,
                                                     velocity_df=velocity_df,
                                                     params=params,
                                                     wave_mode='S')
Z_slice_mag_sens_ps = pndmod.core.get_mag_sensitivity(grid_coords=Z_plane_points,
                                                      geometry_df=geometry_df,
                                                      velocity_df=velocity_df,
                                                      params=params,
                                                      wave_mode='PS')
end_time = time()
print(f"Computation time: {end_time - start_time} seconds")
```

```
Computing magnitude sensitivity...
Computation time: 0.29700303077697754 seconds
```

3.6.5 Plot magnitude sensitivity

Plot magnitude sensitivity for P wave

```
station_marker_size = 25
clb_range = None
plt_levels = None

_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= X_slice_mag_sens_p, yi=gy,␣
↪zi=gz,
                                              plot_title=f"Magnitude sensitivity at X=
↪{slice_X} m: YZ view, P-waves",
                                              clb_title="$M_w$",
                                              geometry_df=geometry_df,
```

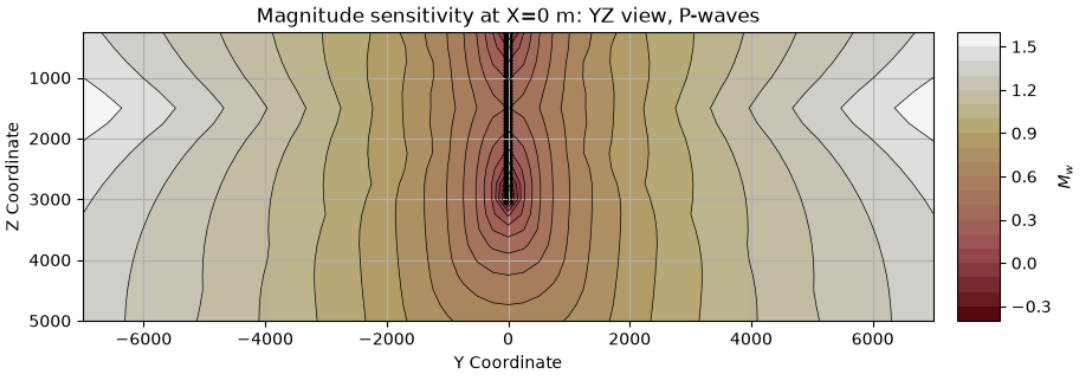
(continues on next page)

(continued from previous page)

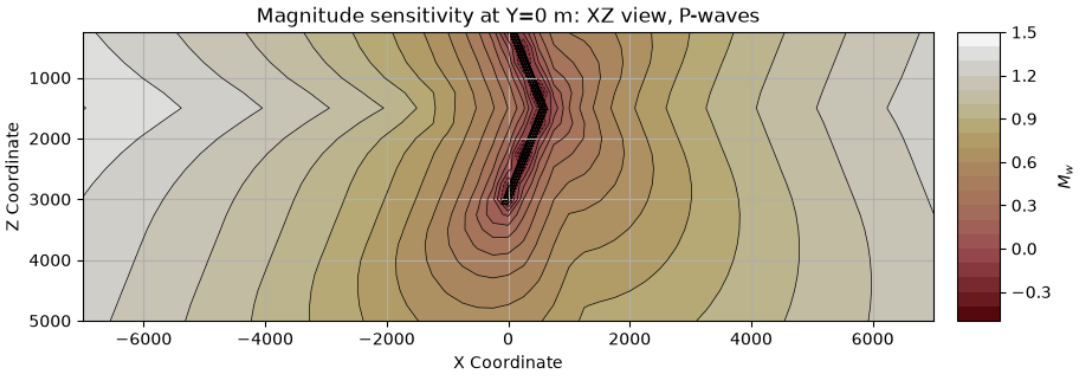
```
                                                plot_names = False,
                                                station_marker_size = station_marker_
↪size,
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)

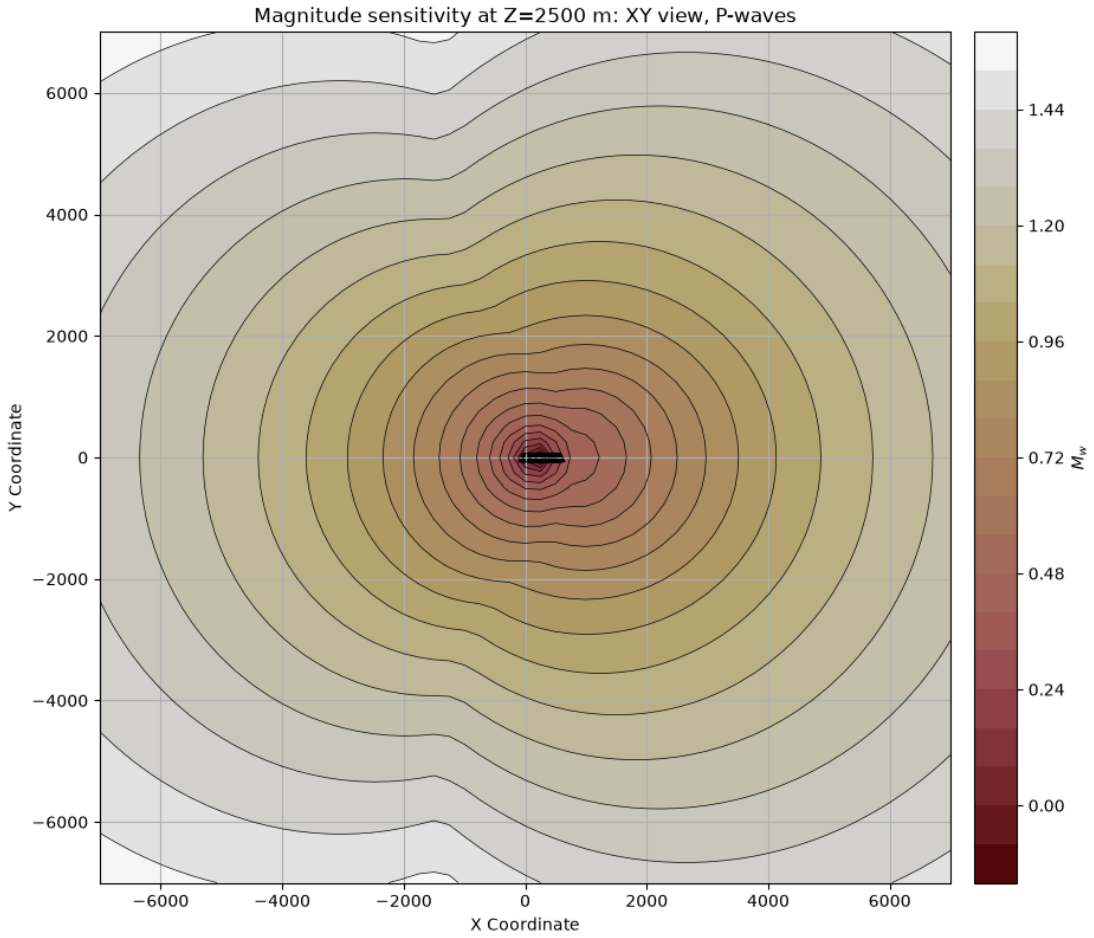
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= Y_slice_mag_sens_p, xi=gx,␣
↪zi=gz,
                                                plot_title=f"Magnitude sensitivity at Y=
↪{slice_Y} m: XZ view, P-waves",
                                                clb_title="$M_w$",
                                                geometry_df=geometry_df,
                                                plot_names = False,
                                                station_marker_size = station_marker_
↪size,
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)

_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= Z_slice_mag_sens_p, xi=gx,␣
↪yi=gy,
                                                plot_title=f"Magnitude sensitivity at Z=
↪{slice_Z} m: XY view, P-waves",
                                                clb_title="$M_w$",
                                                geometry_df=geometry_df,
                                                plot_names = False,
                                                station_marker_size = station_marker_
↪size,
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)
```


Magnitude sensitivity at X=0 m: YZ view, P-waves
Z Coordinate
1000
2000
3000
4000
5000
−6000
−4000
−2000
0
2000
4000
6000
Y Coordinate
1.5
1.2
0.9
0.6
0.3
0.0
−0.3
M_w

•


Magnitude sensitivity at Y=0 m: XZ view, P-waves
Z Coordinate
1000
2000
3000
4000
5000
−6000
−4000
−2000
0
2000
4000
6000
X Coordinate
1.5
1.2
0.9
0.6
0.3
0.0
−0.3
M_w



-

Plot magnitude sensitivity for S wave

```
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= X_slice_mag_sens_s, yi=gy,␣
↪zi=gz,
                                              plot_title=f"Magnitude sensitivity at X=
↪{slice_X} m: YZ view, S-waves",
                                              clb_title="$M_w$",
                                              geometry_df=geometry_df,
                                              plot_names = False,
                                              station_marker_size = station_marker_
↪size,
                                              clb_range=clb_range,
                                              plt_levels=plt_levels)

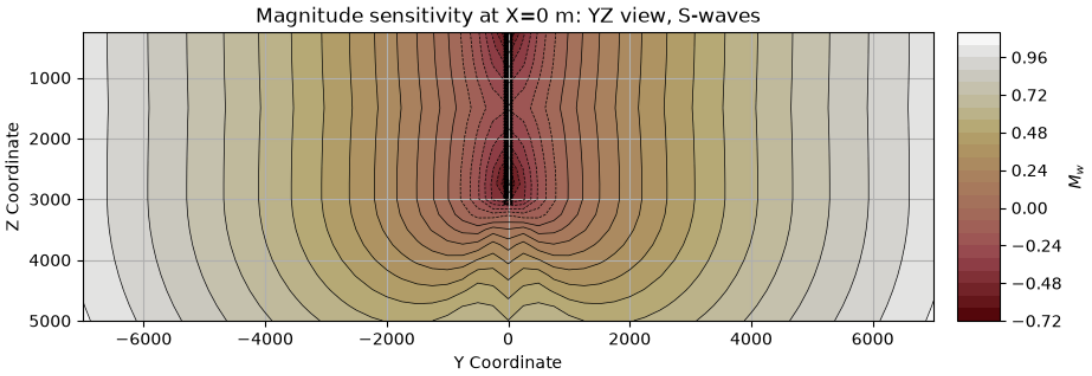
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= Y_slice_mag_sens_s, xi=gx,␣
```

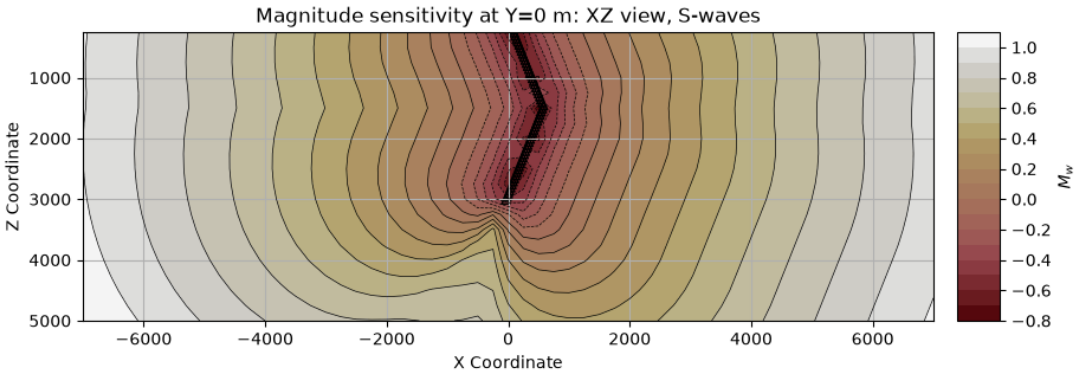
(continues on next page)

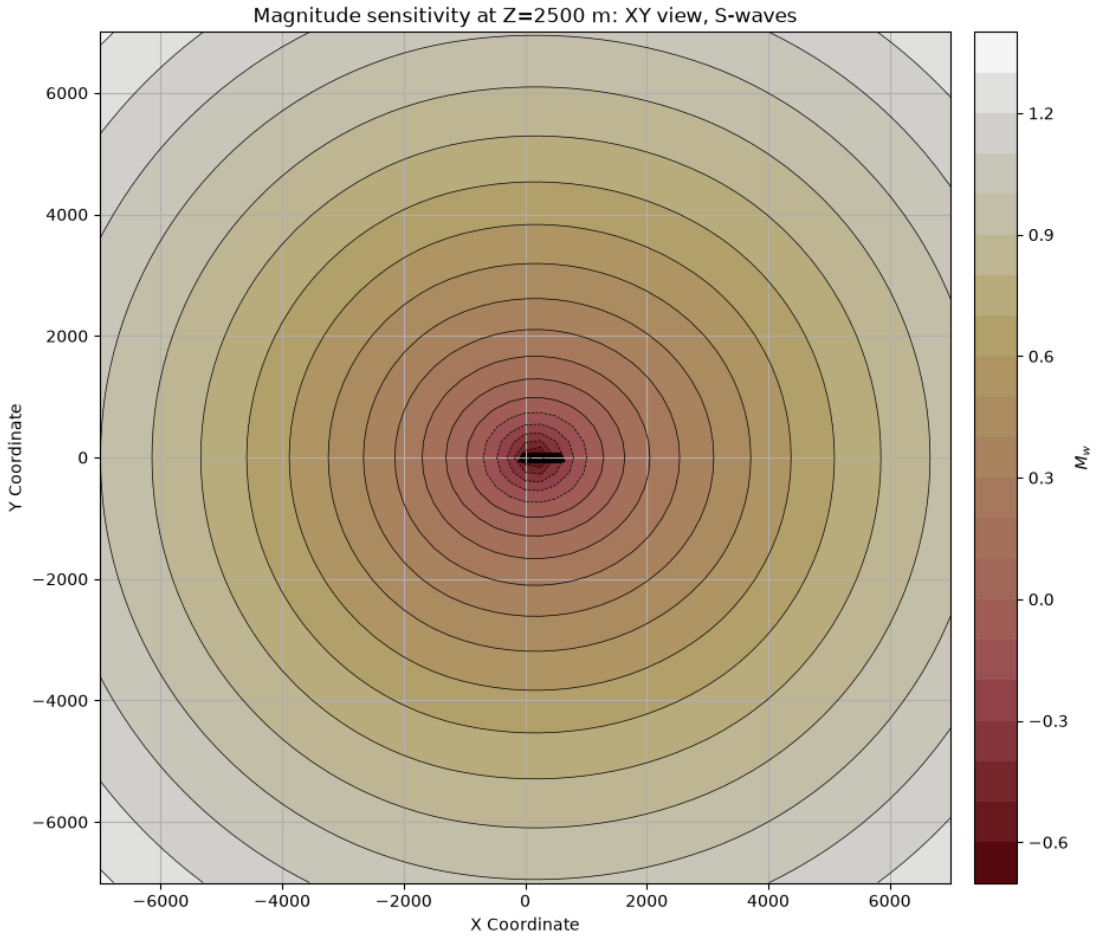
(continued from previous page)

```
↪zi=gz,
                                                plot_title=f"Magnitude sensitivity at Y=
↪{slice_Y} m: XZ view, S-waves",
                                                clb_title="$M_w$",
                                                geometry_df=geometry_df,
                                                plot_names = False,
                                                station_marker_size = station_marker_
↪size,
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)

_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= Z_slice_mag_sens_s, xi=gx,␣
↪yi=gy,
                                                plot_title=f"Magnitude sensitivity at Z=
↪{slice_Z} m: XY view, S-waves",
                                                clb_title="$M_w$",
                                                geometry_df=geometry_df,
                                                plot_names = False,
                                                station_marker_size = station_marker_
↪size,
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)
```


Magnitude sensitivity at X=0 m: YZ view, S-waves
Z Coordinate
1000
2000
3000
4000
5000
−6000
−4000
−2000
0
2000
4000
6000
Y Coordinate
0.96
0.72
0.48
0.24
0.00
−0.24
−0.48
−0.72
M_w


Magnitude sensitivity at Y=0 m: XZ view, S-waves
Z Coordinate
1000
2000
3000
4000
5000
−6000
−4000
−2000
0
2000
4000
6000
X Coordinate
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
M_w



-

Plot magnitude sensitivity for P and S waves

```
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= X_slice_mag_sens_ps, yi=gy,␣
↪zi=gz,
                                              plot_title=f"Magnitude sensitivity at X=
↪{slice_X} m: YZ view, P- and S-waves",
                                              clb_title="$M_w$",
                                              geometry_df=geometry_df,
                                              plot_names = False,
                                              station_marker_size = station_marker_
↪size,
                                              clb_range=clb_range,
                                              plt_levels=plt_levels)

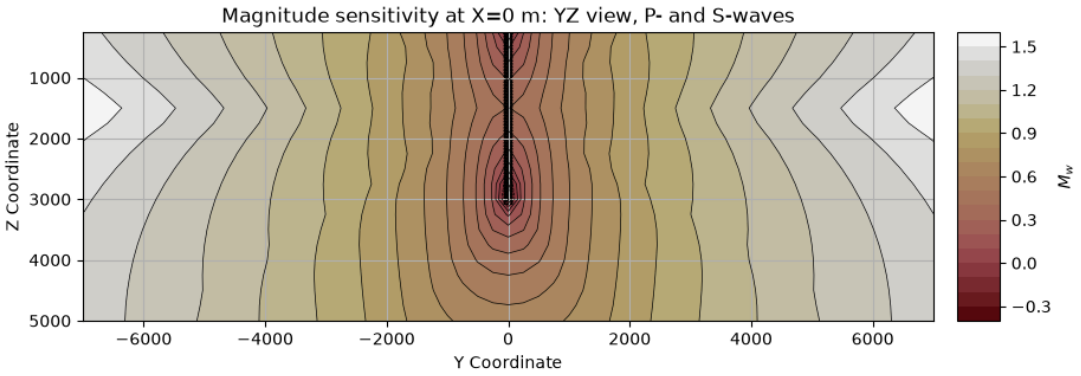
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= Y_slice_mag_sens_ps, xi=gx,␣
```

(continues on next page)

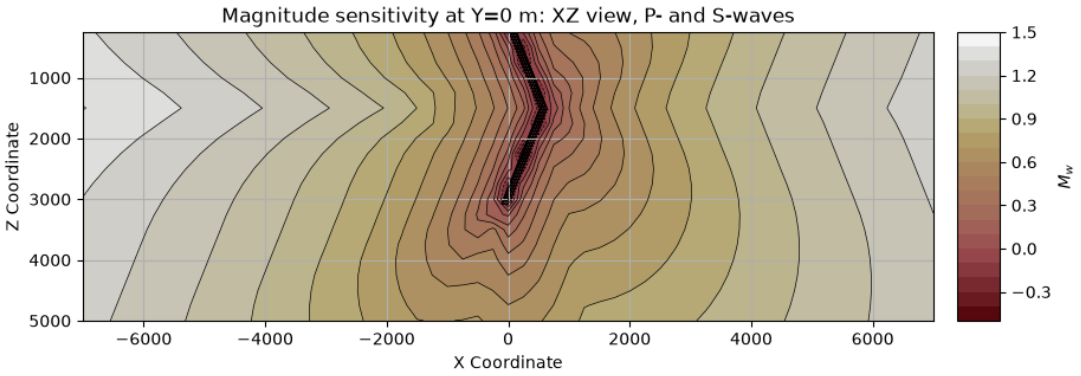
(continued from previous page)

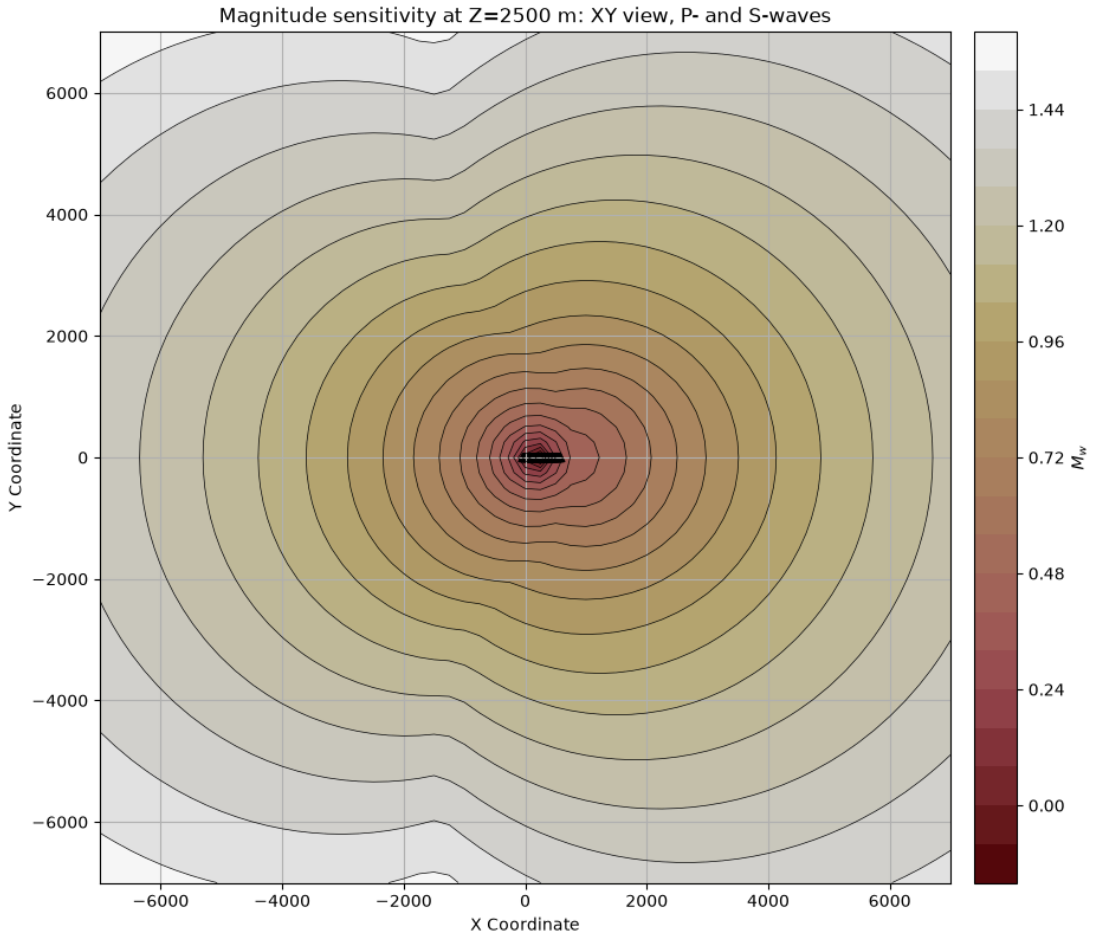
```
↪zi=gz,
                                                plot_title=f"Magnitude sensitivity at Y=
↪{slice_Y} m: XZ view, P- and S-waves",
                                                clb_title="$M_w$",
                                                geometry_df=geometry_df,
                                                plot_names = False,
                                                station_marker_size = station_marker_
↪size,
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)

_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= Z_slice_mag_sens_ps, xi=gx,␣
↪yi=gy,
                                                plot_title=f"Magnitude sensitivity at Z=
↪{slice_Z} m: XY view, P- and S-waves",
                                                clb_title="$M_w$",
                                                geometry_df=geometry_df,
                                                plot_names = False,
                                                station_marker_size = station_marker_
↪size,
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)
```


Magnitude sensitivity at X=0 m: YZ view, P- and S-waves
Z Coordinate
1000
2000
3000
4000
5000
−6000
−4000
−2000
0
2000
4000
6000
Y Coordinate
1.5
1.2
0.9
0.6
0.3
0.0
−0.3
M_w

•


Magnitude sensitivity at Y=0 m: XZ view, P- and S-waves
Z Coordinate
1000
2000
3000
4000
5000
−6000
−4000
−2000
0
2000
4000
6000
X Coordinate
1.5
1.2
0.9
0.6
0.3
0.0
−0.3
M_w



-

Total running time of the script: (0 minutes 2.038 seconds)

3.7 07.1 Magnitude sensitivity for homogeneous model: 1-km DAS array benchmark

This is an example of computing magnitude sensitivity for a homogeneous model.

It reproduces the homogeneous benchmark of a downhole DAS monitoring study: a 1-km-long vertical fibre with a constant noise level, used to isolate the effect of the DAS directional sensitivity before any velocity structure is introduced.

Test case 7:

- Geometry: 1-km-long DAS cable in a vertical borehole, 1 m channel spacing, so 1000 channels between 1 m and 1000 m depth
- Data: constant strain-rate noise level of 1.95e-9 1/s on every channel, gauge length of 10 m

- Velocity model: homogeneous, Vp = 2370 m/s, Vp/Vs = 2 (so Vs = 1185 m/s), Qp = Qs = 100, rho = 2500 kg/m^3
- Processing: magnitude sensitivity on a vertical section through the well, requiring detection on at least 60 channels with a signal-to-noise ratio of at least 2, for P-waves, S-waves and both together.

Three features of the resulting sections are worth looking for, and all three follow from the fact that a DAS cable only records the strain component along the fibre:

1. The minimum detectable magnitude grows with offset from the monitoring well.
2. P-wave sensitivity is worst near the mid-depth of the array, but only at offsets beyond half the array length, where the ray direction that a vertical fibre records best is no longer available anywhere along the cable.
3. S-wave sensitivity degrades below the array, and vanishes on the cable axis itself, where the S-wave polarization is perpendicular to the fibre.

3.7.1 Load all necessary packages

```
import pynetdesign.visualization as pndvis
import pynetdesign.modelling as pndmod
import numpy as np
from time import time

# sphinx_gallery_thumbnail_number = -1
```

3.7.2 Input

Read geometry

```
geometry_file_path = '../data/homo/test_07/TMPL_net_geometry_Noise_Constant.txt'
geometry_df = pndmod.io.read_geometry(geometry_file_path)

print(geometry_df)
print(geometry_df.attrs)
```

```
       Y    X       Z    NoiseLevel
0    0.0  0.0     1.0  1.950000e-09
1    0.0  0.0     2.0  1.950000e-09
2    0.0  0.0     3.0  1.950000e-09
3    0.0  0.0     4.0  1.950000e-09
4    0.0  0.0     5.0  1.950000e-09
..   ...  ...     ...           ...
995  0.0  0.0   996.0  1.950000e-09
996  0.0  0.0   997.0  1.950000e-09
997  0.0  0.0   998.0  1.950000e-09
998  0.0  0.0   999.0  1.950000e-09
999  0.0  0.0  1000.0  1.950000e-09

[1000 rows x 4 columns]
{'file_path': '../data/homo/test_07/TMPL_net_geometry_Noise_Constant.txt', 'original_
↪coord_units': 'm', 'coordinate_system': 'local', 'noise_type': 'strain rate',
↪'noise_unit': '1/s', 'gauge_length': 10.0, '_is_combined': False}
```

Read velocity model

```
velocity_file_path = '../data/homo/test_07/TMPL_velocity_model.txt'
velocity_df = pndmod.io.read_velocity(velocity_file_path)

print(velocity_df)
print(velocity_df.attrs)
```

```
   Depth    Vp  VpVsRatio   Qp   Qs   Rho   Ep   Dt   Gm
0  40000  2370          2  100  100  2500  0.0  0.0  0.0
{'model_type': 'homo', 'model_anisotropy': 'isotropic', 'parameter_descriptions': {
↪'Depth': 'Depth', 'Vp': 'P-wave velocity', 'Vs': 'S-wave velocity', 'VpVsRatio':
↪'Vp/Vs', 'Qp': 'P-wave quality factor', 'Qs': 'S-wave quality factor', 'Rho':
↪'density', 'Ep': 'Thomsen epsilon parameter', 'Dt': 'Thomsen delta parameter', 'Gm
↪': 'Thomsen gamma parameter'}, 'parameter_units': {'Depth': 'm', 'Vp': 'm/s', 'Vs':
↪'m/s', 'VpVsRatio': '', 'Qp': '', 'Qs': '', 'Rho': 'kg/m^3', 'Ep': '', 'Dt': '', 'Gm
↪': ''}}
```

Set imaging grid parameters

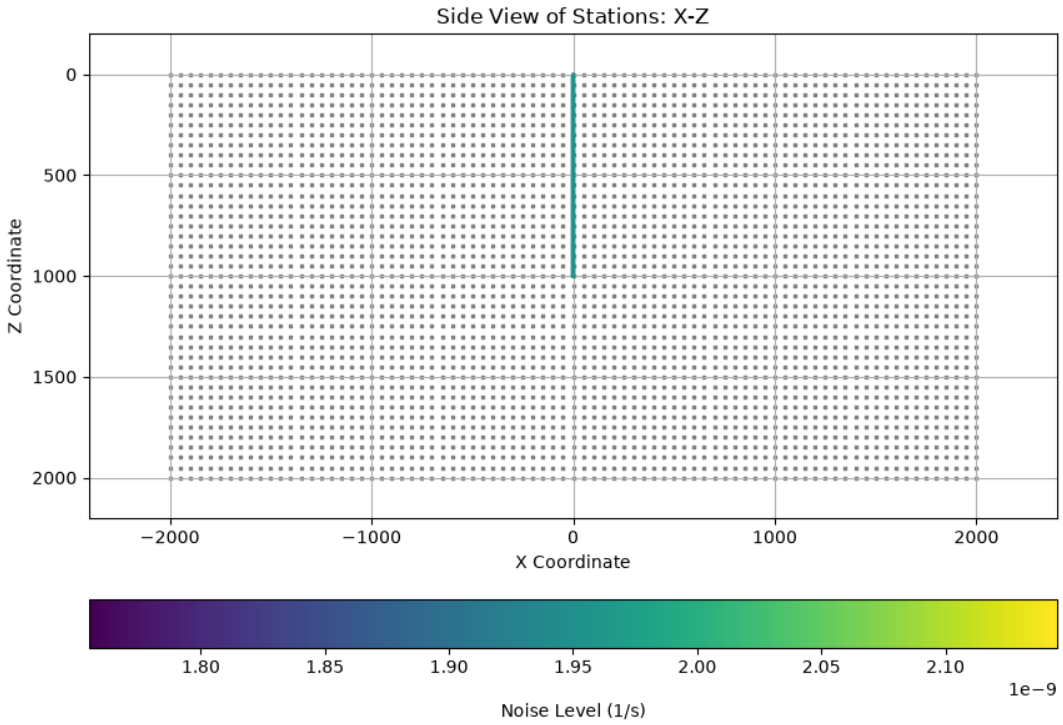
```
# Imaging grid parameters: a vertical section through the monitoring well,
# reaching twice the array length in offset and in depth
gx = np.arange(-2000, 2000 + 50, 50)
gy = np.array([0])
gz = np.arange(0, 2000 + 50, 50)

# Generate grid points
grid_points = pndmod.utils.generate_grid(x=gx, y=gy, z=gz)
print("Number of grid points:", grid_points.size)
print("Number of grid points in x:", gx.size)
print("Number of grid points in z:", gz.size)
print("grid_points.shape:", grid_points.shape)
```

```
Number of grid points: 9963
Number of grid points in x: 81
Number of grid points in z: 41
grid_points.shape: (3321, 3)
```

Plot geometry and imaging grid

```
_ = pndvis.geometry.plot_stations_view(geometry_df,
                                       view='xz',
                                       grid_points=grid_points,
                                       grid_marker_size=1,
                                       plot_names=False,
                                       station_marker_size=2)
```



3.7.3 Prepare to compute magnitude sensitivity

```
# Get medium parameters
density = velocity_df.at[0, 'Rho']
v_p = velocity_df.at[0, 'Vp']
Q_p = velocity_df.at[0, 'Qp']
v_s = velocity_df.at[0, 'Vp']/velocity_df.at[0, 'VpVsRatio']
Q_s = velocity_df.at[0, 'Qs']
print("density, v_p, Q_p, v_s, Q_s:", density, v_p, Q_p, v_s, Q_s)

# Minimum S/N for detection on an individual channel
min_SNR_p = 2
min_SNR_s = 2
```

(continues on next page)

(continued from previous page)

```
# Minimum channels on which an event must be detected
min_stations_p = 60
min_stations_s = 60

# Save parameters. The source frequency is left unset, so that the representative
# frequency is the peak frequency of each arrival, clamped at the corner frequency
# of 100 Hz. The free surface correction is switched off: every channel of this
# cable is downhole, so none of them sees the doubling of a surface receiver.
params = pndmod.classes.DetectionParameters(
    min_SNR_p=min_SNR_p,
    min_SNR_s=min_SNR_s,
    min_stations_p=min_stations_p,
    min_stations_s=min_stations_s,
    fs_mode='off',
)

# Define the slice to use
slice_Y = 0

# Get the slice plane
Y_plane_points, Y_plane_indices = pndmod.utils.get_slice_coordinates(grid_points=grid_
↪points,
                                                                     slice_
↪dimension=1,
                                                                     slice_
↪coordinate=slice_Y)
```

```
density, v_p, Q_p, v_s, Q_s: 2500 2370 100 1185.0 100
```

3.7.4 Compute magnitude sensitivity

```
print("Computing magnitude sensitivity...")

start_time = time()
Y_slice_mag_sens_p = pndmod.core.get_mag_sensitivity(grid_coords=Y_plane_points,
                                                     geometry_df=geometry_df,
                                                     velocity_df=velocity_df,
                                                     params=params,
                                                     wave_mode='P')
Y_slice_mag_sens_s = pndmod.core.get_mag_sensitivity(grid_coords=Y_plane_points,
                                                     geometry_df=geometry_df,
                                                     velocity_df=velocity_df,
                                                     params=params,
                                                     wave_mode='S')
Y_slice_mag_sens_ps = pndmod.core.get_mag_sensitivity(grid_coords=Y_plane_points,
                                                      geometry_df=geometry_df,
                                                      velocity_df=velocity_df,
                                                      params=params,
                                                      wave_mode='PS')
end_time = time()
```

(continues on next page)

(continued from previous page)

```
print(f"Computation time: {end_time - start_time} seconds")
```

```
Computing magnitude sensitivity...
Computation time: 3.6952273845672607 seconds
```

3.7.5 Plot magnitude sensitivity

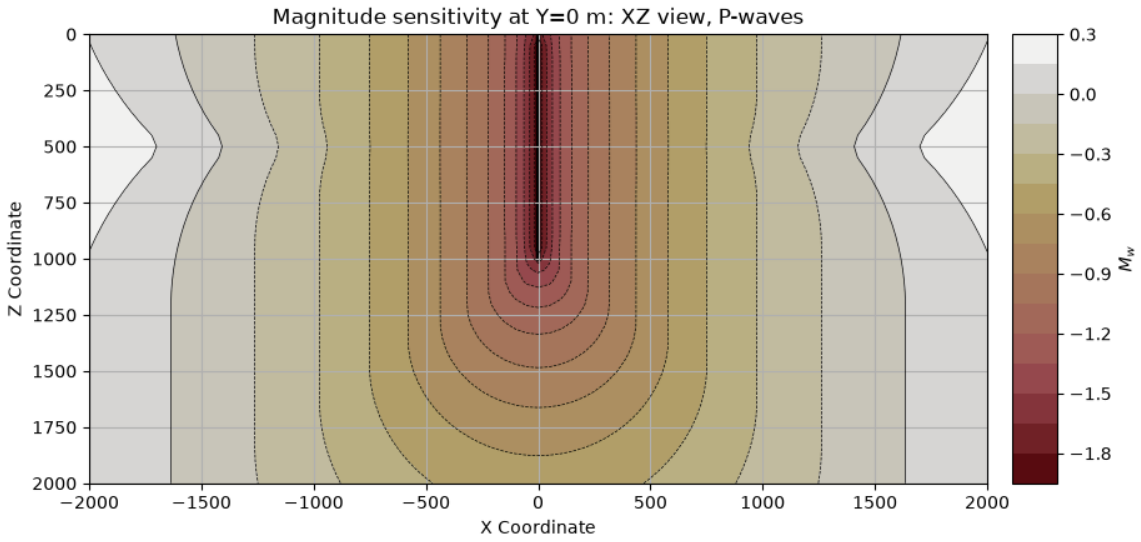
The three panels correspond to the three panels of the benchmark figure: P-waves only, S-waves only, and detection of both.

```
station_marker_size = 2
clb_range = None
plt_levels = None
```

Plot magnitude sensitivity for P wave

The sensitivity is worst at the mid-depth of the array, and that loss only appears at offsets larger than half the array length.

```
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=Y_slice_mag_sens_p, xi=gx,␣
↪zi=gz,
                                              plot_title=f"Magnitude sensitivity at Y=
↪{slice_Y} m: XZ view, P-waves",
                                              clb_title="$M_w$",
                                              geometry_df=geometry_df,
                                              plot_names=False,
                                              station_marker_size=station_marker_size,
                                              clb_range=clb_range,
                                              plt_levels=plt_levels)
```



Plot magnitude sensitivity for S wave

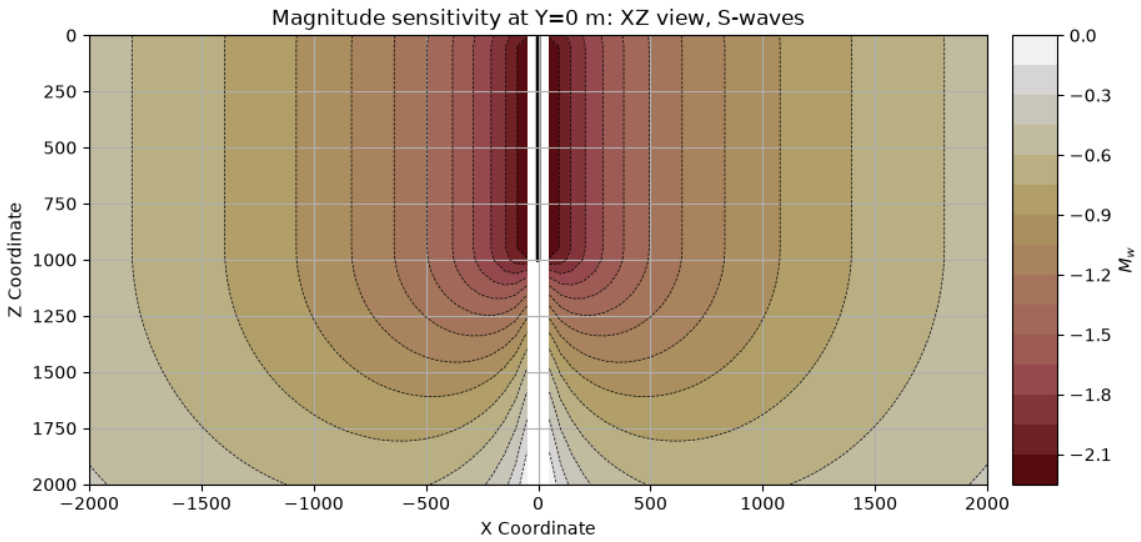
The S-wave polarization is perpendicular to the fibre on the cable axis, so the cable cannot record an S-wave arriving from directly below it.

```
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=Y_slice_mag_sens_s, xi=gx,␣
↪zi=gz,
                                              plot_title=f"Magnitude sensitivity at Y=
↪{slice_Y} m: XZ view, S-waves",
                                              clb_title="$M_w$",
                                              geometry_df=geometry_df,
                                              plot_names=False,
                                              station_marker_size=station_marker_size,
```

(continues on next page)

(continued from previous page)

```
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)
```



Plot magnitude sensitivity for P and S waves

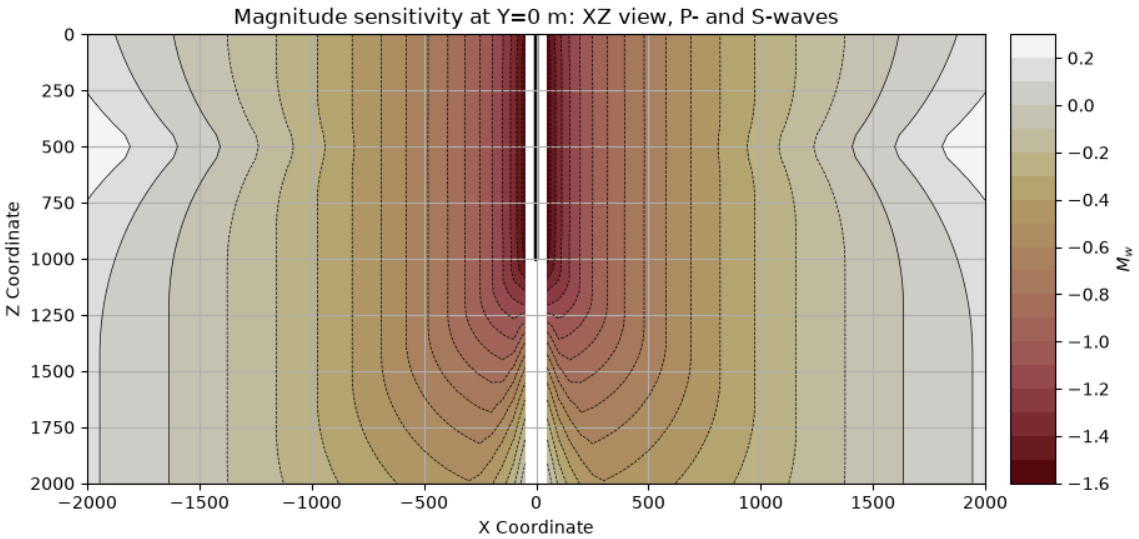
Requiring both phases takes the worse of the two, so this panel inherits the P-wave loss at mid-depth and the S-wave loss below the array.

```
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data=Y_slice_mag_sens_ps, xi=gx,␣
↪zi=gz,
                                                plot_title=f"Magnitude sensitivity at Y=
↪{slice_Y} m: XZ view, P- and S-waves",
```

(continues on next page)

(continued from previous page)

```
                                                clb_title="$M_w$",
                                                geometry_df=geometry_df,
                                                plot_names=False,
                                                station_marker_size=station_marker_size,
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)
```



Total running time of the script: (0 minutes 4.301 seconds)

3.8 08.1 Magnitude sensitivity for homogeneous model: test case 8

This is an example of computing magnitude sensitivity for a homogeneous model.

Test case 8:

- Geometry: DAS cable in a deep vertical borehole
- Data: Noise is decreasing with depth (from 5e-7 to 8.87e-12)
- Velocity model: homogeneous, Vp = 4300 m/s, Vp/Vs = 1.73, Qp = 100, Qs = 50, rho = 2300 kg/m^3
- Processing: magnitude sensitivity computation for all stations (for grid slices only).

3.8.1 Load all necessary packages

```
import pynetdesign.visualization as pndvis
import pynetdesign.modelling as pndmod
import numpy as np
from time import time

# sphinx_gallery_thumbnail_number = -2
```

3.8.2 Input

Read geometry

```
geometry_file_path = '../data/homo/test_08/TMPL_net_geometry_Noise_Decreasing.txt'
geometry_df = pndmod.io.read_geometry(geometry_file_path)

print(geometry_df)
print(geometry_df.attrs)
```

```
      Y    X       Z    NoiseLevel
0   0.0  0.0    10.0  5.000000e-07
1   0.0  0.0    60.0  4.170000e-07
2   0.0  0.0   110.0  3.470000e-07
3   0.0  0.0   160.0  2.890000e-07
4   0.0  0.0   210.0  2.410000e-07
..  ...  ...     ...           ...
56  0.0  0.0  2810.0  1.840000e-11
57  0.0  0.0  2860.0  1.530000e-11
58  0.0  0.0  2910.0  1.280000e-11
59  0.0  0.0  2960.0  1.060000e-11
60  0.0  0.0  3010.0  8.870000e-12

[61 rows x 4 columns]
{'file_path': '../data/homo/test_08/TMPL_net_geometry_Noise_Decreasing.txt',
↪'original_coord_units': 'm', 'coordinate_system': 'local', 'noise_type': 'strain',
↪'noise_unit': 'ε', 'gauge_length': 10.0, '_is_combined': False}
```

Read velocity model

```
velocity_file_path = '../data/homo/test_08/TMPL_velocity_model.txt'
velocity_df = pndmod.io.read_velocity(velocity_file_path)

print(velocity_df)
print(velocity_df.attrs)
```

```
   Depth    Vp  VpVsRatio   Qp  Qs   Rho   Ep   Dt   Gm
0  40000  4300       1.73  100  50  2300  0.0  0.0  0.0
{'model_type': 'homo', 'model_anisotropy': 'isotropic', 'parameter_descriptions': {
↪'Depth': 'Depth', 'Vp': 'P-wave velocity', 'Vs': 'S-wave velocity', 'VpVsRatio':
↪'Vp/Vs', 'Qp': 'P-wave quality factor', 'Qs': 'S-wave quality factor', 'Rho':
↪'density', 'Ep': 'Thomsen epsilon parameter', 'Dt': 'Thomsen delta parameter', 'Gm
↪': 'Thomsen gamma parameter'}, 'parameter_units': {'Depth': 'm', 'Vp': 'm/s', 'Vs':
↪'m/s', 'VpVsRatio': '', 'Qp': '', 'Qs': '', 'Rho': 'kg/m^3', 'Ep': '', 'Dt': '', 'Gm
↪': ''}}
```

Set imaging grid parameters

```
# Imaging grid parameters
gx = np.arange(-7000, 7000 + 250, 250)
gy = np.arange(-7000, 7000 + 250, 250)
gz = np.arange(250, 5000 + 250, 250)

# Generate grid points
grid_points = pndmod.utils.generate_grid(x=gx,y=gy,z=gz)
print("Number of grid points:",grid_points.size)
print("Number of grid points in x:",gx.size)
print("Number of grid points in y:",gy.size)
print("Number of grid points in z:",gz.size)
print("grid_points.shape:",grid_points.shape)
```

```
Number of grid points: 194940
Number of grid points in x: 57
Number of grid points in y: 57
Number of grid points in z: 20
grid_points.shape: (64980, 3)
```

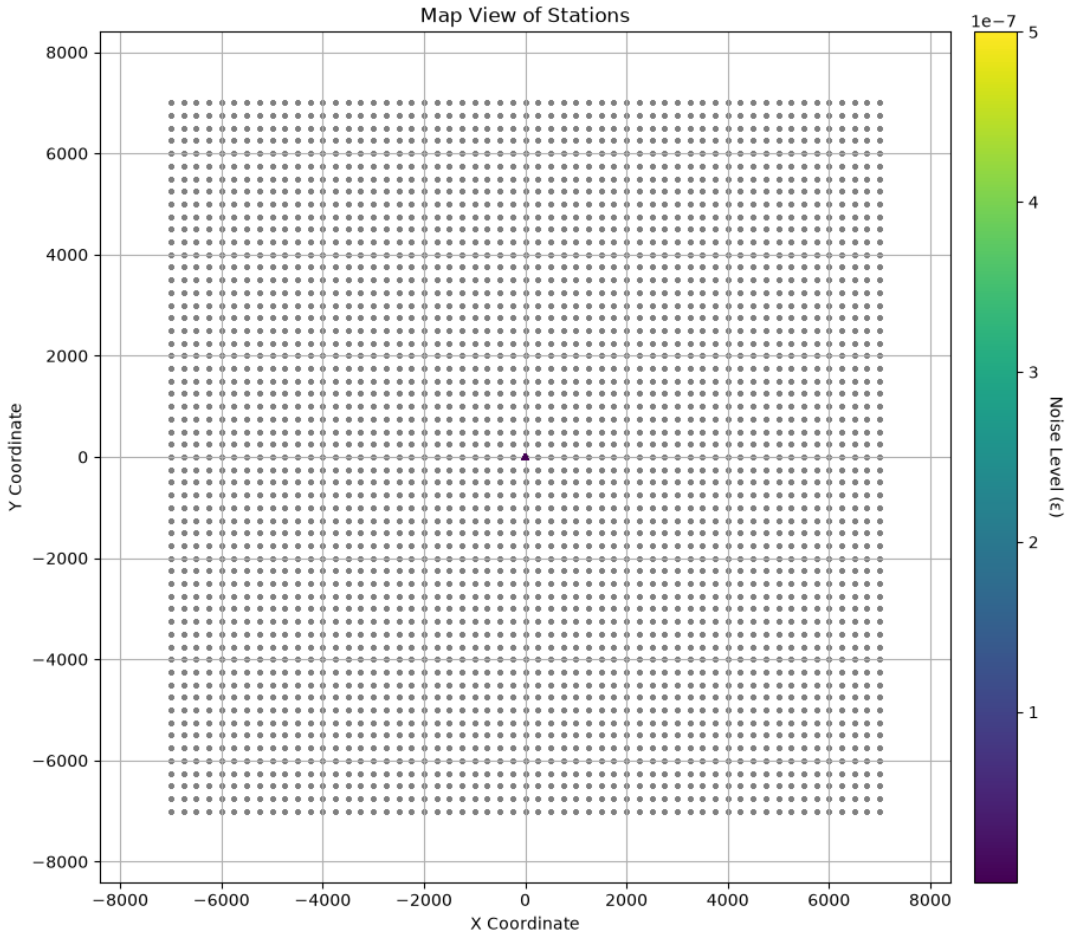
Plot geometry and imaging grid

```
_ = pndvis.geometry.plot_stations_view(geometry_df,
                                       view='xy',
                                       grid_points=grid_points,
                                       grid_marker_size=1,
                                       plot_names=False,
                                       station_marker_size=10)
_ = pndvis.geometry.plot_stations_view(geometry_df,
                                       view='xz',
                                       grid_points=grid_points,
                                       grid_marker_size=1,
                                       plot_names=False,
```

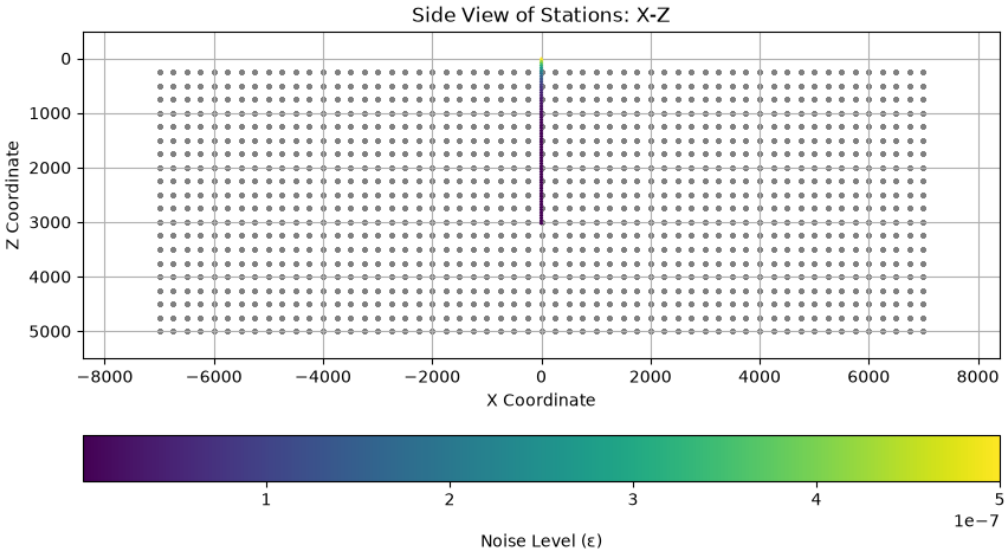
(continues on next page)

(continued from previous page)

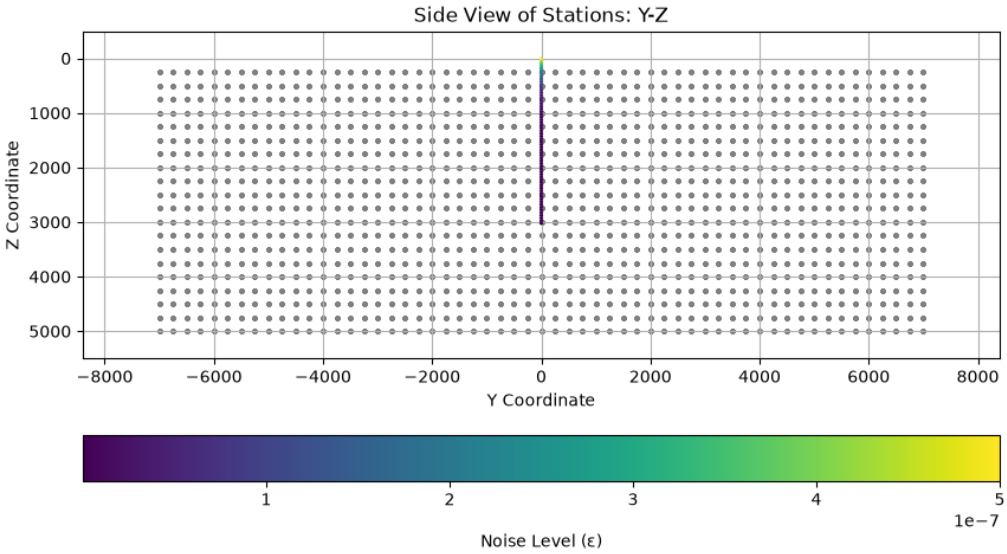
```
                                        station_marker_size=2)
_ = pndvis.geometry.plot_stations_view(geometry_df,
                                        view='yz',
                                        grid_points=grid_points,
                                        grid_marker_size=1,
                                        plot_names=False,
                                        station_marker_size=2)
```



-


Side View of Stations: X-Z
Z Coordinate
0
1000
2000
3000
4000
5000
−8000
−6000
−4000
−2000
0
2000
4000
6000
8000
X Coordinate
1
2
3
4
5
1e−7
Noise Level (ε)

•



-

3.8.3 Prepare to compute magnitude sensitivity

```
# Get medium parameters
density = velocity_df.at[0,'Rho']
v_p = velocity_df.at[0,'Vp']
Q_p = velocity_df.at[0,'Qp']
v_s = velocity_df.at[0,'Vp']/velocity_df.at[0,'VpVsRatio']
Q_s = velocity_df.at[0,'Qs']
print("density, v_p, Q_p, v_s, Q_s:",density,v_p,Q_p,v_s,Q_s)

# Specify source parameters
frequency = 10 # Hz

# Minimum S/N for detection on individual channel
min_SNR_p = 2
```

(continues on next page)

(continued from previous page)

```
min_SNR_s = 2

# Minimum stations on which an event must be detected
min_stations_p = 3
min_stations_s = 3

# Save parameters
params = pndmod.classes.DetectionParameters(
    f_p=frequency,
    f_s=frequency,
    min_SNR_p=min_SNR_p,
    min_SNR_s=min_SNR_s,
    min_stations_p=min_stations_p,
    min_stations_s=min_stations_s,
)

# Define slices to use
slice_X = 0
slice_Y = 0
slice_Z = 2500

# Get slice planes
X_plane_points, X_plane_indices = pndmod.utils.get_slice_coordinates(grid_points=grid_
↪points,slice_dimension=0,slice_coordinate=slice_X)
Y_plane_points, Y_plane_indices = pndmod.utils.get_slice_coordinates(grid_points=grid_
↪points,slice_dimension=1,slice_coordinate=slice_Y)
Z_plane_points, Z_plane_indices = pndmod.utils.get_slice_coordinates(grid_points=grid_
↪points,slice_dimension=2,slice_coordinate=slice_Z)
```

```
density, v_p, Q_p, v_s, Q_s: 2300 4300 100 2485.549132947977 50
```

3.8.4 Compute magnitude sensitivity

```
print("Computing magnitude sensitivity...")


start_time = time()
X_slice_mag_sens_p = pndmod.core.get_mag_sensitivity(grid_coords=X_plane_points,
                                                     geometry_df=geometry_df,
                                                     velocity_df=velocity_df,
                                                     params=params,
                                                     wave_mode='P')
X_slice_mag_sens_s = pndmod.core.get_mag_sensitivity(grid_coords=X_plane_points,
                                                     geometry_df=geometry_df,
                                                     velocity_df=velocity_df,
                                                     params=params,
                                                     wave_mode='S')
X_slice_mag_sens_ps = pndmod.core.get_mag_sensitivity(grid_coords=X_plane_points,
                                                      geometry_df=geometry_df,
                                                      velocity_df=velocity_df,
                                                      params=params,
```

(continues on next page)

(continued from previous page)

```
                                                          wave_mode='PS')
Y_slice_mag_sens_p = pndmod.core.get_mag_sensitivity(grid_coords=Y_plane_points,
                                                     geometry_df=geometry_df,
                                                     velocity_df=velocity_df,
                                                     params=params,
                                                     wave_mode='P')
Y_slice_mag_sens_s = pndmod.core.get_mag_sensitivity(grid_coords=Y_plane_points,
                                                     geometry_df=geometry_df,
                                                     velocity_df=velocity_df,
                                                     params=params,
                                                     wave_mode='S')
Y_slice_mag_sens_ps = pndmod.core.get_mag_sensitivity(grid_coords=Y_plane_points,
                                                      geometry_df=geometry_df,
                                                      velocity_df=velocity_df,
                                                      params=params,
                                                      wave_mode='PS')
Z_slice_mag_sens_p = pndmod.core.get_mag_sensitivity(grid_coords=Z_plane_points,
                                                     geometry_df=geometry_df,
                                                     velocity_df=velocity_df,
                                                     params=params,
                                                     wave_mode='P')
Z_slice_mag_sens_s = pndmod.core.get_mag_sensitivity(grid_coords=Z_plane_points,
                                                     geometry_df=geometry_df,
                                                     velocity_df=velocity_df,
                                                     params=params,
                                                     wave_mode='S')
Z_slice_mag_sens_ps = pndmod.core.get_mag_sensitivity(grid_coords=Z_plane_points,
                                                      geometry_df=geometry_df,
                                                      velocity_df=velocity_df,
                                                      params=params,
                                                      wave_mode='PS')
end_time = time()
print(f"Computation time: {end_time - start_time} seconds")
```

```
Computing magnitude sensitivity...
Computation time: 0.2943897247314453 seconds
```

3.8.5 Plot magnitude sensitivity

Plot magnitude sensitivity for P wave

```
station_marker_size = 25
clb_range = None
plt_levels = None

_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= X_slice_mag_sens_p, yi=gy,␣
↪zi=gz,
                                              plot_title=f"Magnitude sensitivity at X=
↪{slice_X} m: YZ view, P-waves",
                                              clb_title="$M_w$",
                                              geometry_df=geometry_df,
```

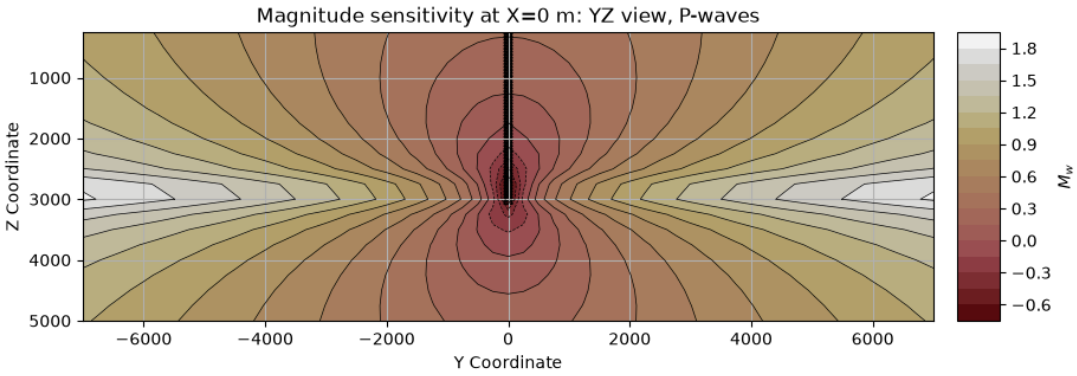
(continues on next page)

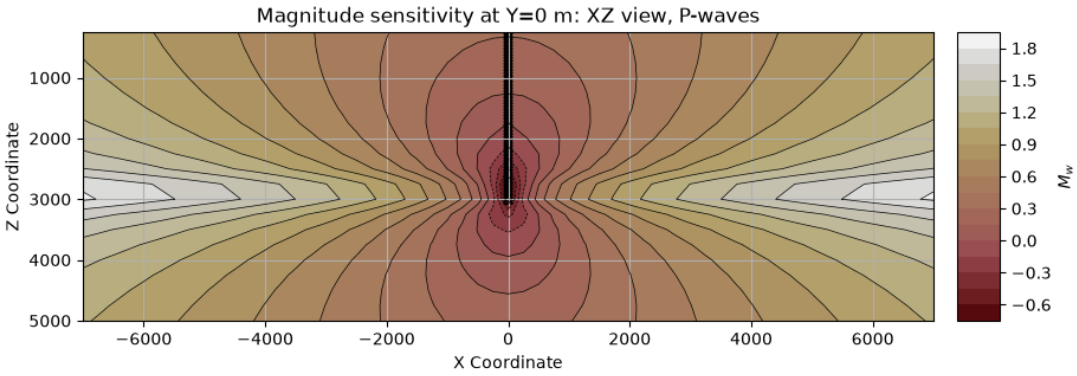
(continued from previous page)

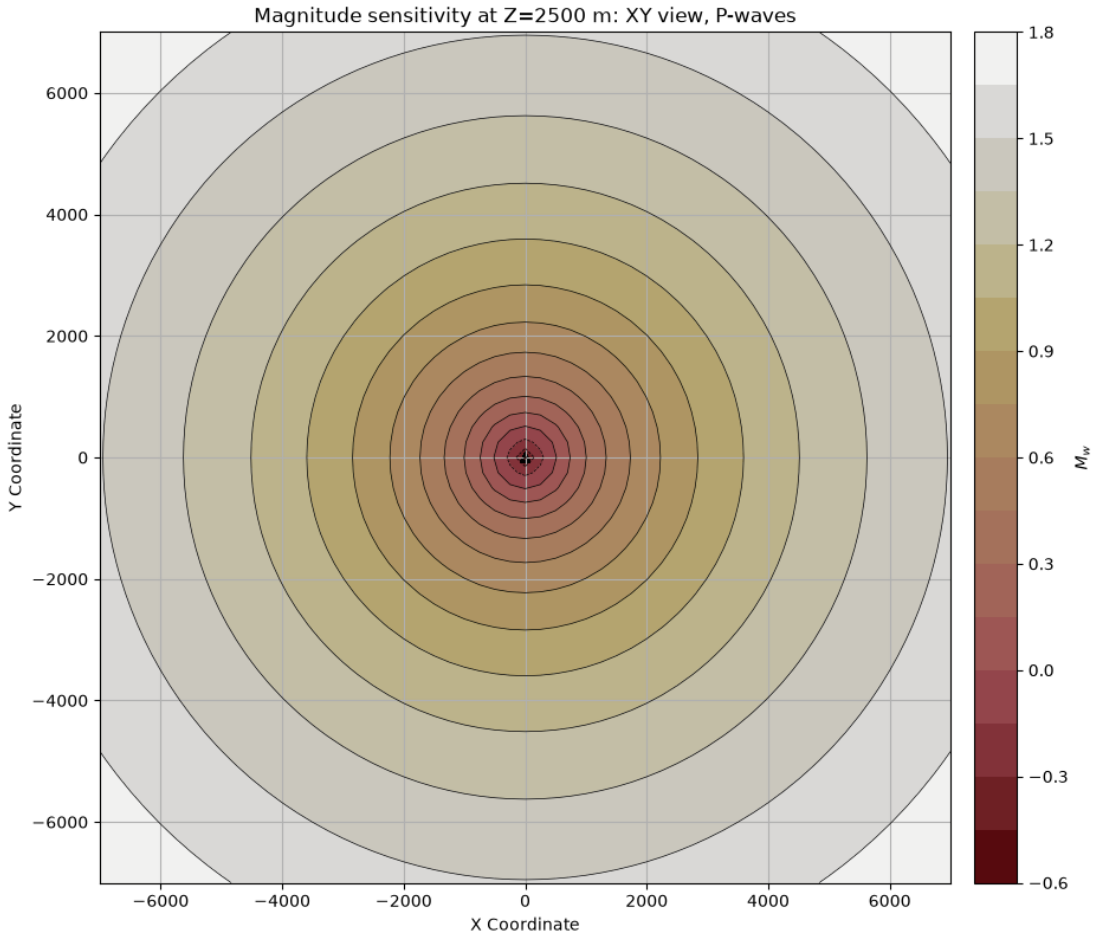
```
                                                plot_names = False,
                                                station_marker_size = station_marker_
↪size,
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)

_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= Y_slice_mag_sens_p, xi=gx,␣
↪zi=gz,
                                                plot_title=f"Magnitude sensitivity at Y=
↪{slice_Y} m: XZ view, P-waves",
                                                clb_title="$M_w$",
                                                geometry_df=geometry_df,
                                                plot_names = False,
                                                station_marker_size = station_marker_
↪size,
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)

_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= Z_slice_mag_sens_p, xi=gx,␣
↪yi=gy,
                                                plot_title=f"Magnitude sensitivity at Z=
↪{slice_Z} m: XY view, P-waves",
                                                clb_title="$M_w$",
                                                geometry_df=geometry_df,
                                                plot_names = False,
                                                station_marker_size = station_marker_
↪size,
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)
```


Magnitude sensitivity at X=0 m: YZ view, P-waves
Z Coordinate
1000
2000
3000
4000
5000
−6000
−4000
−2000
0
2000
4000
6000
Y Coordinate
1.8
1.5
1.2
0.9
0.6
0.3
0.0
−0.3
−0.6
M_w


Magnitude sensitivity at Y=0 m: XZ view, P-waves
Z Coordinate
1000
2000
3000
4000
5000
−6000
−4000
−2000
0
2000
4000
6000
X Coordinate
1.8
1.5
1.2
0.9
0.6
0.3
0.0
−0.3
−0.6
M_w



-

Plot magnitude sensitivity for S wave

```
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= X_slice_mag_sens_s, yi=gy,␣
↪zi=gz,
                                              plot_title=f"Magnitude sensitivity at X=
↪{slice_X} m: YZ view, S-waves",
                                              clb_title="$M_w$",
                                              geometry_df=geometry_df,
                                              plot_names = False,
                                              station_marker_size = station_marker_
↪size,
                                              clb_range=clb_range,
                                              plt_levels=plt_levels)

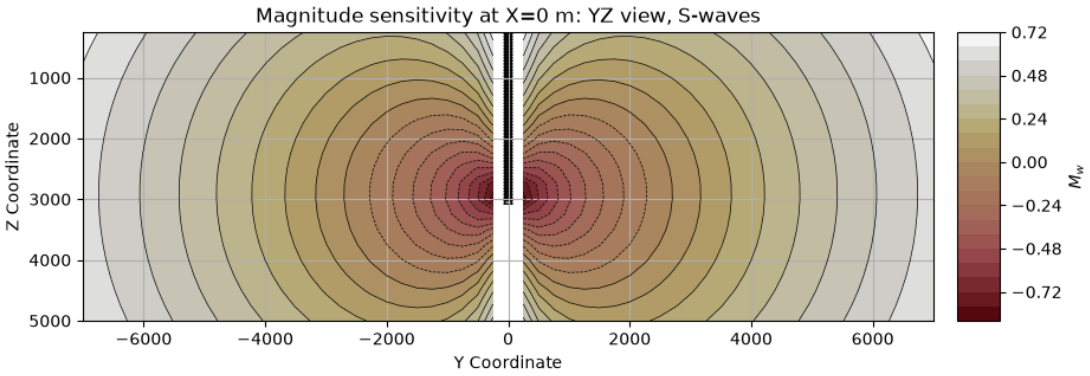
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= Y_slice_mag_sens_s, xi=gx,␣
```

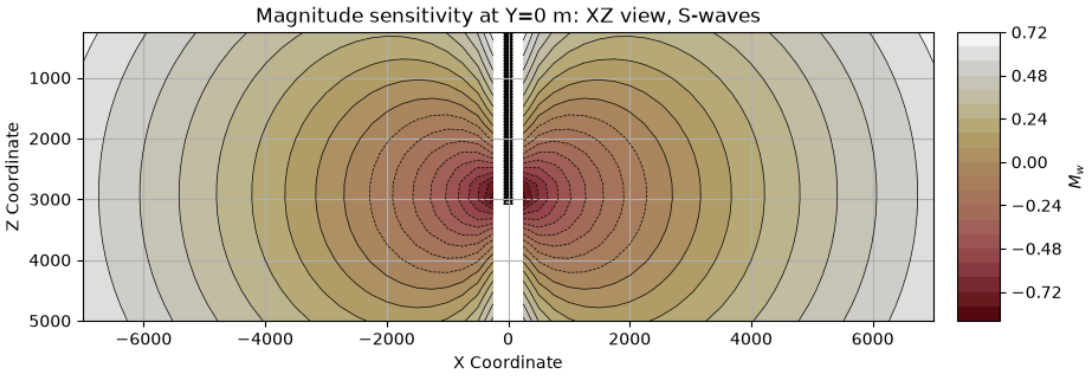
(continues on next page)

(continued from previous page)

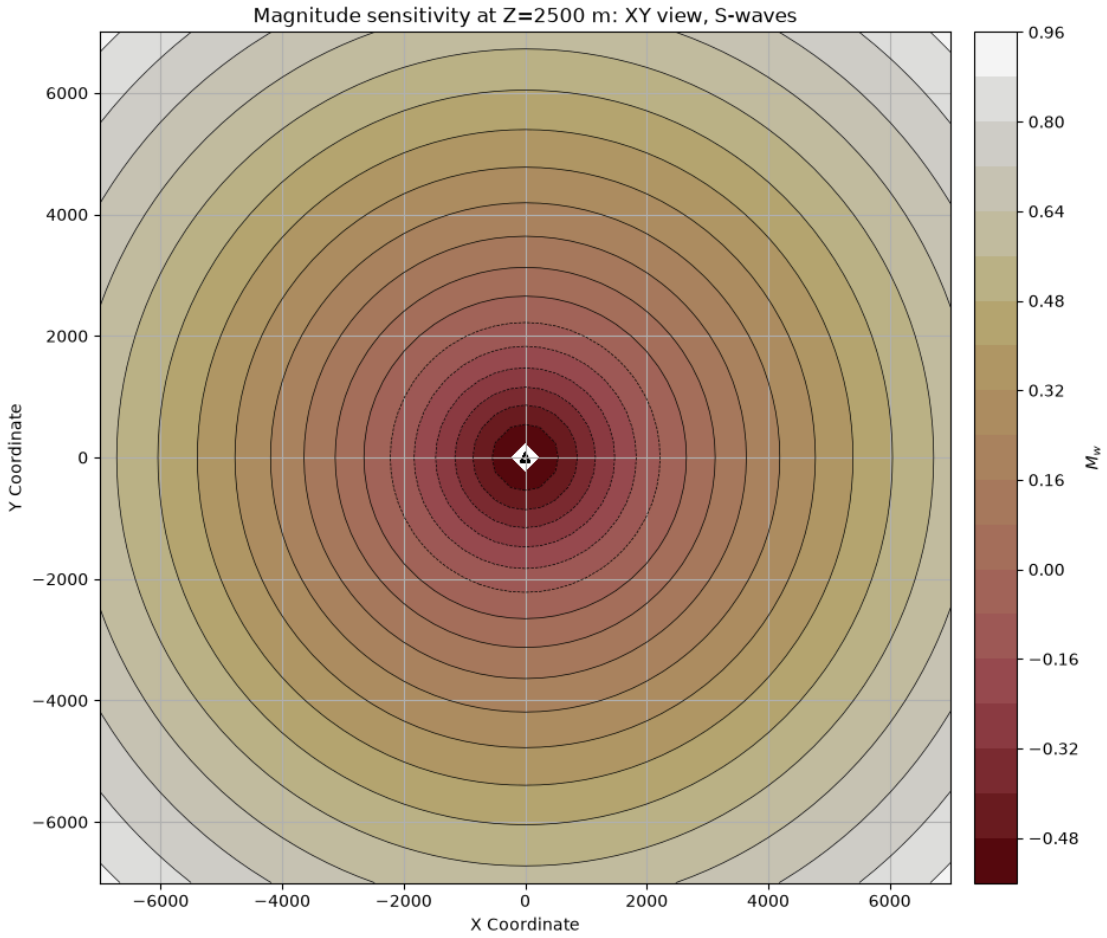
```
↪zi=gz,
                                                plot_title=f"Magnitude sensitivity at Y=
↪{slice_Y} m: XZ view, S-waves",
                                                clb_title="$M_w$",
                                                geometry_df=geometry_df,
                                                plot_names = False,
                                                station_marker_size = station_marker_
↪size,
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)

_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= Z_slice_mag_sens_s, xi=gx,␣
↪yi=gy,
                                                plot_title=f"Magnitude sensitivity at Z=
↪{slice_Z} m: XY view, S-waves",
                                                clb_title="$M_w$",
                                                geometry_df=geometry_df,
                                                plot_names = False,
                                                station_marker_size = station_marker_
↪size,
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)
```


Magnitude sensitivity at X=0 m: YZ view, S-waves
Z Coordinate
1000
2000
3000
4000
5000
−6000
−4000
−2000
0
2000
4000
6000
Y Coordinate
0.72
0.48
0.24
0.00
−0.24
−0.48
−0.72
M_w


Magnitude sensitivity at Y=0 m: XZ view, S-waves
Z Coordinate
X Coordinate
1000
2000
3000
4000
5000
−6000
−4000
−2000
0
2000
4000
6000
0.72
0.48
0.24
0.00
−0.24
−0.48
−0.72
M_w

•



-

Plot magnitude sensitivity for P and S waves

```
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= X_slice_mag_sens_ps, yi=gy,␣
↪zi=gz,
                                              plot_title=f"Magnitude sensitivity at X=
↪{slice_X} m: YZ view, P- and S-waves",
                                              clb_title="$M_w$",
                                              geometry_df=geometry_df,
                                              plot_names = False,
                                              station_marker_size = station_marker_
↪size,
                                              clb_range=clb_range,
                                              plt_levels=plt_levels)

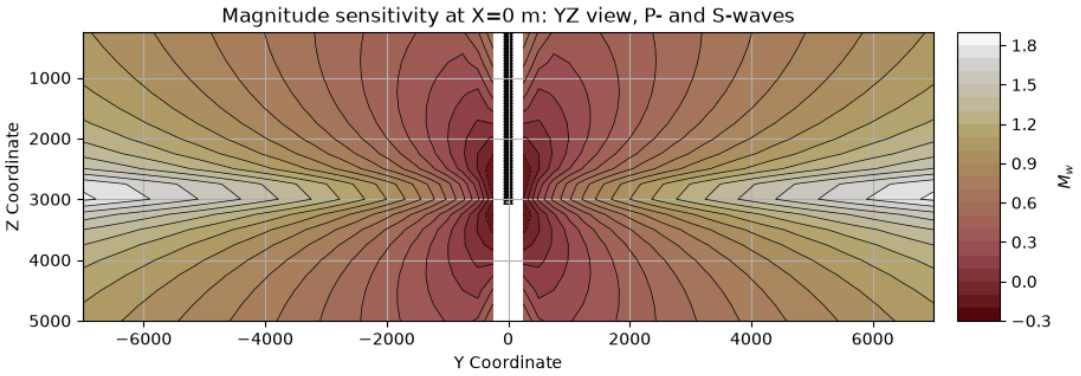
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= Y_slice_mag_sens_ps, xi=gx,␣
```

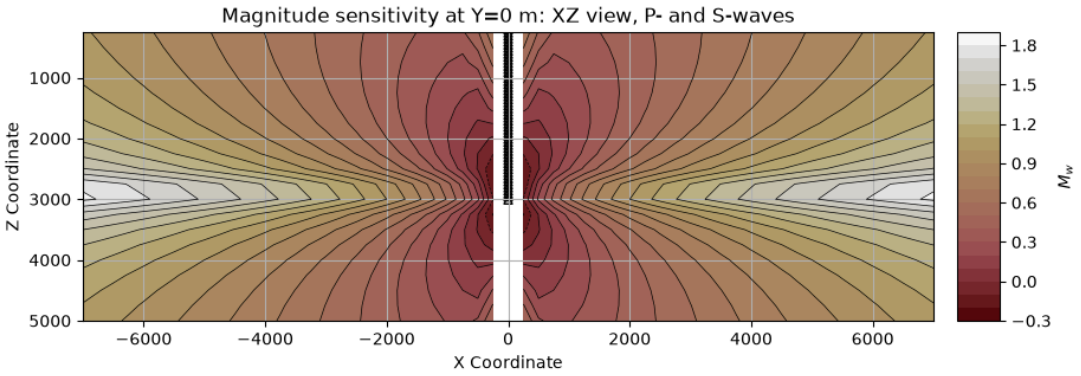
(continues on next page)

(continued from previous page)

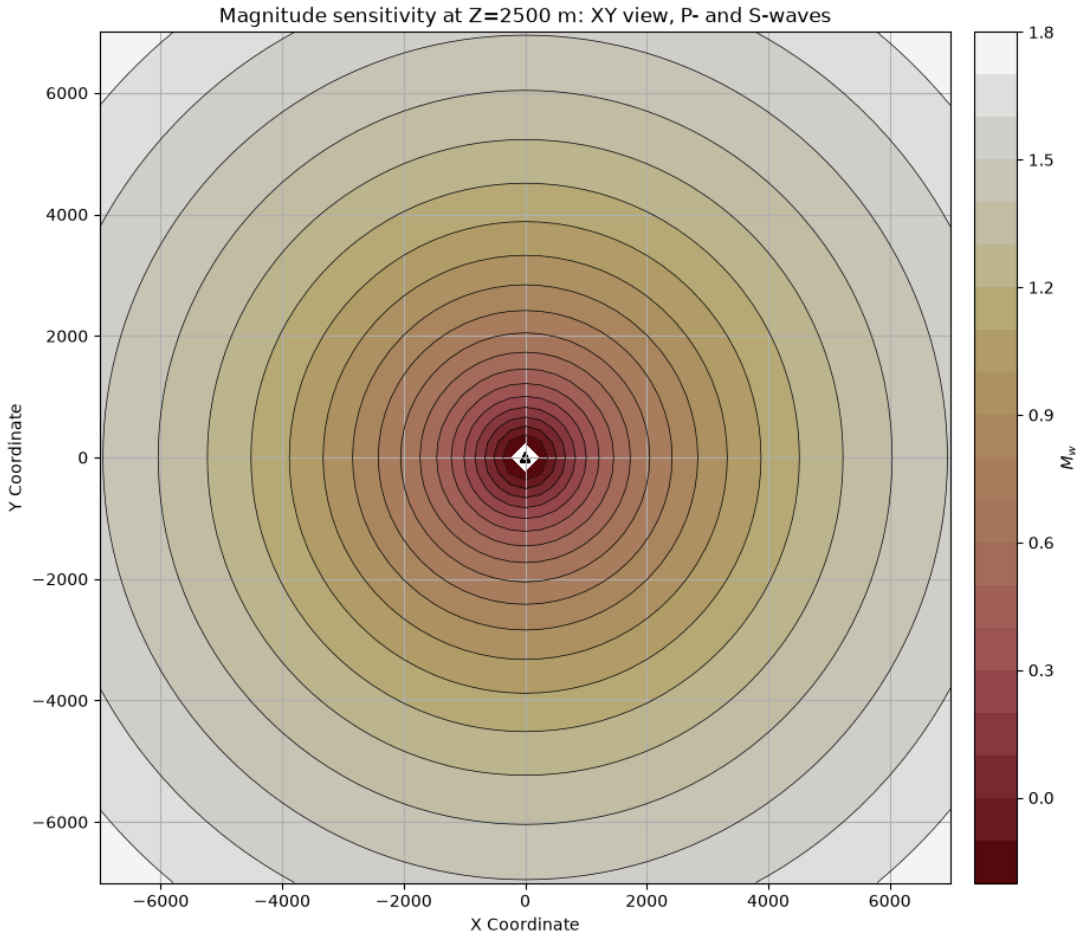
```
↪zi=gz,
                                                plot_title=f"Magnitude sensitivity at Y=
↪{slice_Y} m: XZ view, P- and S-waves",
                                                clb_title="$M_w$",
                                                geometry_df=geometry_df,
                                                plot_names = False,
                                                station_marker_size = station_marker_
↪size,
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)

_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= Z_slice_mag_sens_ps, xi=gx,␣
↪yi=gy,
                                                plot_title=f"Magnitude sensitivity at Z=
↪{slice_Z} m: XY view, P- and S-waves",
                                                clb_title="$M_w$",
                                                geometry_df=geometry_df,
                                                plot_names = False,
                                                station_marker_size = station_marker_
↪size,
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)
```


Magnitude sensitivity at X=0 m: YZ view, P- and S-waves
Z Coordinate
1000
2000
3000
4000
5000
−6000
−4000
−2000
0
2000
4000
6000
Y Coordinate
1.8
1.5
1.2
0.9
0.6
0.3
0.0
−0.3
M_w


Magnitude sensitivity at Y=0 m: XZ view, P- and S-waves
Z Coordinate
1000
2000
3000
4000
5000
−6000
−4000
−2000
0
2000
4000
6000
X Coordinate
1.8
1.5
1.2
0.9
0.6
0.3
0.0
−0.3
M_w

•



-

Total running time of the script: (0 minutes 2.050 seconds)

3.9 09.1 Magnitude sensitivity for homogeneous model: test case 9

This is an example of computing magnitude sensitivity for a homogeneous model.

Test case 9:

- Geometry: a single station above the DAS cable in a deep vertical borehole
- Data: Noise level at the station: 5e-9 m/s, for DAS - constant noise level along the borehole (5e-11)
- Velocity model: homogeneous, Vp = 4300 m/s, Vp/Vs = 1.73, Qp = 100, Qs = 50, rho = 2300 kg/m^3
- Processing: magnitude sensitivity computation for all stations (for grid slices only).

3.9.1 Load all necessary packages

```
import pynetdesign.visualization as pndvis
import pynetdesign.modelling as pndmod
import numpy as np
from time import time

# sphinx_gallery_thumbnail_number = -2
```

3.9.2 Input

Read geometry

```
### Read geometry
station_geometry_file_path = '../data/homo/test_09/TMPL_net_geometry_Single.txt'
station_geometry_df = pndmod.io.read_geometry(station_geometry_file_path)
print(station_geometry_df)
print(station_geometry_df.attrs)

das_geometry_file_path = '../data/homo/test_09/TMPL_net_geometry_Noise_Constant.txt'
das_geometry_df = pndmod.io.read_geometry(das_geometry_file_path)
print(das_geometry_df)
print(das_geometry_df.attrs)

### Show
print("\nStation geometry:")
print(station_geometry_df.head())
print(station_geometry_df.attrs)
print("\nDAS geometry:")
print(das_geometry_df.head())
print(das_geometry_df.attrs)
```

```
  Name    Y    X    Z    NoiseLevel
0  ST1  0.0  0.0 -0.0  5.000000e-09
{'file_path': '../data/homo/test_09/TMPL_net_geometry_Single.txt', 'original_coord_
↪units': 'm', 'coordinate_system': 'local', 'noise_type': 'velocity', 'noise_unit':
↪'m/s', '_is_combined': False}
      Y    X       Z    NoiseLevel
0   0.0  0.0    10.0  5.000000e-11
1   0.0  0.0    60.0  5.000000e-11
2   0.0  0.0   110.0  5.000000e-11
3   0.0  0.0   160.0  5.000000e-11
4   0.0  0.0   210.0  5.000000e-11
..  ...  ...     ...           ...
56  0.0  0.0  2810.0  5.000000e-11
57  0.0  0.0  2860.0  5.000000e-11
58  0.0  0.0  2910.0  5.000000e-11
59  0.0  0.0  2960.0  5.000000e-11
60  0.0  0.0  3010.0  5.000000e-11

[61 rows x 4 columns]
{'file_path': '../data/homo/test_09/TMPL_net_geometry_Noise_Constant.txt', 'original_
↪coord_units': 'm', 'coordinate_system': 'local', 'noise_type': 'strain', 'noise_unit
```

(continues on next page)

(continued from previous page)

```
↪': 'ε', 'gauge_length': 10.0, '_is_combined': False}

Station geometry:
  Name    Y    X    Z    NoiseLevel
0  ST1  0.0  0.0 -0.0  5.000000e-09
{'file_path': '../data/homo/test_09/TMPL_net_geometry_Single.txt', 'original_coord_
↪units': 'm', 'coordinate_system': 'local', 'noise_type': 'velocity', 'noise_unit':
↪'m/s', '_is_combined': False}

DAS geometry:
     Y    X      Z    NoiseLevel
0  0.0  0.0   10.0  5.000000e-11
1  0.0  0.0   60.0  5.000000e-11
2  0.0  0.0  110.0  5.000000e-11
3  0.0  0.0  160.0  5.000000e-11
4  0.0  0.0  210.0  5.000000e-11
{'file_path': '../data/homo/test_09/TMPL_net_geometry_Noise_Constant.txt', 'original_
↪coord_units': 'm', 'coordinate_system': 'local', 'noise_type': 'strain', 'noise_unit
↪': 'ε', 'gauge_length': 10.0, '_is_combined': False}
```

Combine geometry

```
combined_geometry_df = pndmod.io.combine_geometry(station_geometry_df,
                                                  das_geometry_df,
                                                  names=['Stations',
                                                         'DAS'])


### Show combined
print("\nCombined geometry:")
print(combined_geometry_df)
print(combined_geometry_df.attrs)
print("\nGeometry names:")
print(combined_geometry_df.get_names())
print("\nCombined status:",
      pndmod.io.is_combined_geometry(combined_geometry_df))

### Show recovered
print("\nStation geometry (recovered):")
print(combined_geometry_df.recover('Stations').head())
print("\nCombined status:",
      pndmod.io.is_combined_geometry(combined_geometry_df.recover('Stations')))

print("\nDAS geometry (recovered):")
print(combined_geometry_df.recover('DAS').head())
print("\nCombined status:",
      pndmod.io.is_combined_geometry(combined_geometry_df.recover('DAS')))
print("\nAttributes of DAS (recovered):")
print(combined_geometry_df.recover('DAS').attrs)
```

```
Combined geometry:
           Name    Y    X       Z    NoiseLevel
```

(continues on next page)

(continued from previous page)

```
source
Stations 0   ST1  0.0  0.0    -0.0  5.000000e-09
DAS      0   NaN  0.0  0.0    10.0  5.000000e-11
         1   NaN  0.0  0.0    60.0  5.000000e-11
         2   NaN  0.0  0.0   110.0  5.000000e-11
         3   NaN  0.0  0.0   160.0  5.000000e-11
...          ...  ...  ...     ...           ...
         56  NaN  0.0  0.0  2810.0  5.000000e-11
         57  NaN  0.0  0.0  2860.0  5.000000e-11
         58  NaN  0.0  0.0  2910.0  5.000000e-11
         59  NaN  0.0  0.0  2960.0  5.000000e-11
         60  NaN  0.0  0.0  3010.0  5.000000e-11

[62 rows x 5 columns]
{'original_attributes': {'Stations': {'file_path': '../data/homo/test_09/TMPL_net_
↪geometry_Single.txt', 'original_coord_units': 'm', 'coordinate_system': 'local',
↪'noise_type': 'velocity', 'noise_unit': 'm/s', '_is_combined': False}, 'DAS': {
↪'file_path': '../data/homo/test_09/TMPL_net_geometry_Noise_Constant.txt', 'original_
↪coord_units': 'm', 'coordinate_system': 'local', 'noise_type': 'strain', 'noise_unit
↪': 'ε', 'gauge_length': 10.0, '_is_combined': False}}, '_is_combined': True}

Geometry names:
['Stations', 'DAS']

Combined status: True

Station geometry (recovered):
  Name    Y    X    Z    NoiseLevel
0  ST1  0.0  0.0 -0.0  5.000000e-09

Combined status: False

DAS geometry (recovered):
     Y    X      Z    NoiseLevel
0  0.0  0.0   10.0  5.000000e-11
1  0.0  0.0   60.0  5.000000e-11
2  0.0  0.0  110.0  5.000000e-11
3  0.0  0.0  160.0  5.000000e-11
4  0.0  0.0  210.0  5.000000e-11

Combined status: False

Attributes of DAS (recovered):
{'file_path': '../data/homo/test_09/TMPL_net_geometry_Noise_Constant.txt', 'original_
↪coord_units': 'm', 'coordinate_system': 'local', 'noise_type': 'strain', 'noise_unit
↪': 'ε', 'gauge_length': 10.0, '_is_combined': False}
```

Read velocity model

```
velocity_file_path = '../data/homo/test_09/TMPL_velocity_model.txt'
velocity_df = pndmod.io.read_velocity(velocity_file_path)
```

(continues on next page)

(continued from previous page)

```
print(velocity_df)
print(velocity_df.attrs)
```

```
   Depth    Vp  VpVsRatio   Qp  Qs   Rho   Ep   Dt   Gm
0  40000  4300       1.73  100  50  2300  0.0  0.0  0.0
{'model_type': 'homo', 'model_anisotropy': 'isotropic', 'parameter_descriptions': {
↪'Depth': 'Depth', 'Vp': 'P-wave velocity', 'Vs': 'S-wave velocity', 'VpVsRatio':
↪'Vp/Vs', 'Qp': 'P-wave quality factor', 'Qs': 'S-wave quality factor', 'Rho':
↪'density', 'Ep': 'Thomsen epsilon parameter', 'Dt': 'Thomsen delta parameter', 'Gm
↪': 'Thomsen gamma parameter'}, 'parameter_units': {'Depth': 'm', 'Vp': 'm/s', 'Vs':
↪'m/s', 'VpVsRatio': '', 'Qp': '', 'Qs': '', 'Rho': 'kg/m^3', 'Ep': '', 'Dt': '', 'Gm
↪': ''}}
```

Set imaging grid parameters

```
# Imaging grid parameters
gx = np.arange(-7000, 7000 + 100, 100)
gy = np.arange(-7000, 7000 + 100, 100)
gz = np.arange(0, 10000 + 100, 100)

# Generate grid points
grid_points = pndmod.utils.generate_grid(x=gx,y=gy,z=gz)
print("Number of grid points:",grid_points.size)
print("Number of grid points in x:",gx.size)
print("Number of grid points in y:",gy.size)
print("Number of grid points in z:",gz.size)
print("grid_points.shape:",grid_points.shape)
```

```
Number of grid points: 6023943
Number of grid points in x: 141
Number of grid points in y: 141
Number of grid points in z: 101
grid_points.shape: (2007981, 3)
```

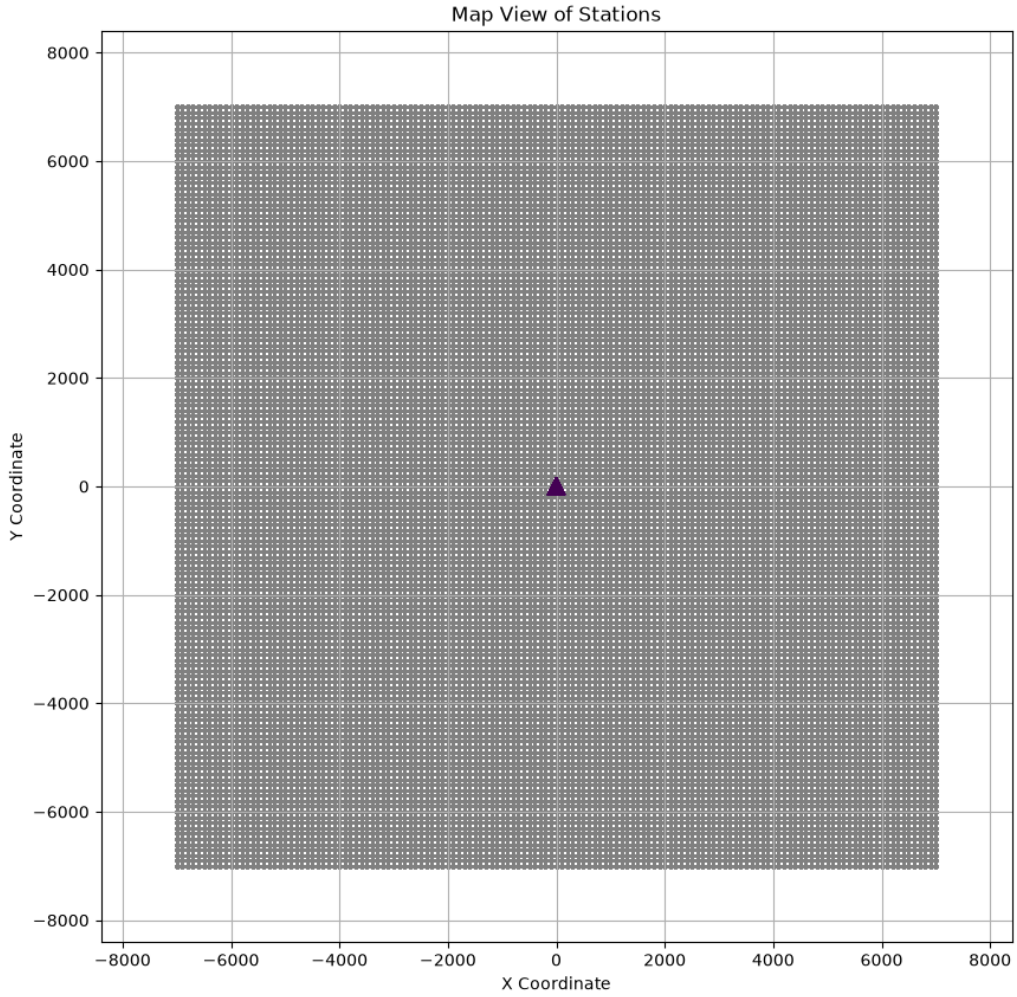
Plot geometry and imaging grid

```
_ = pndvis.geometry.plot_stations_view(df=combined_geometry_df,
                                       grid_points=grid_points,
                                       plot_names=False,
                                       grid_marker_size = 1,
                                       station_marker_size=100)
_ = pndvis.geometry.plot_stations_view(df=combined_geometry_df,
                                       grid_points=grid_points,
                                       plot_names=False,
                                       grid_marker_size = 1,
                                       station_marker_size=1,
                                       view = 'xz')
_ = pndvis.geometry.plot_stations_view(df=combined_geometry_df,
                                       grid_points=grid_points,
```

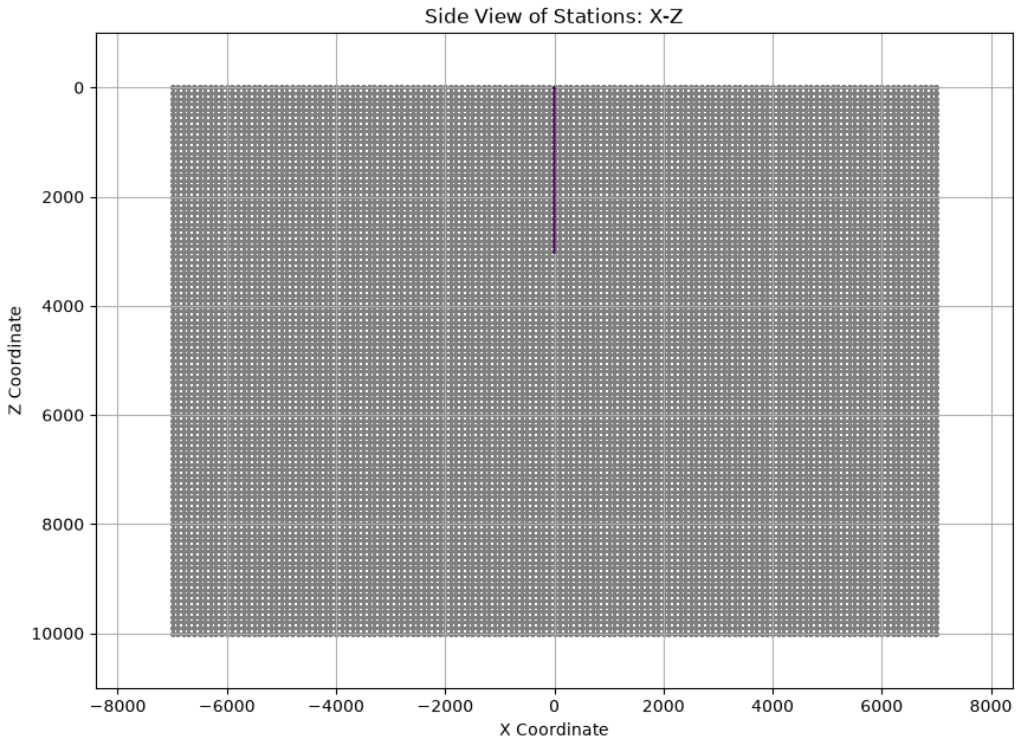
(continues on next page)

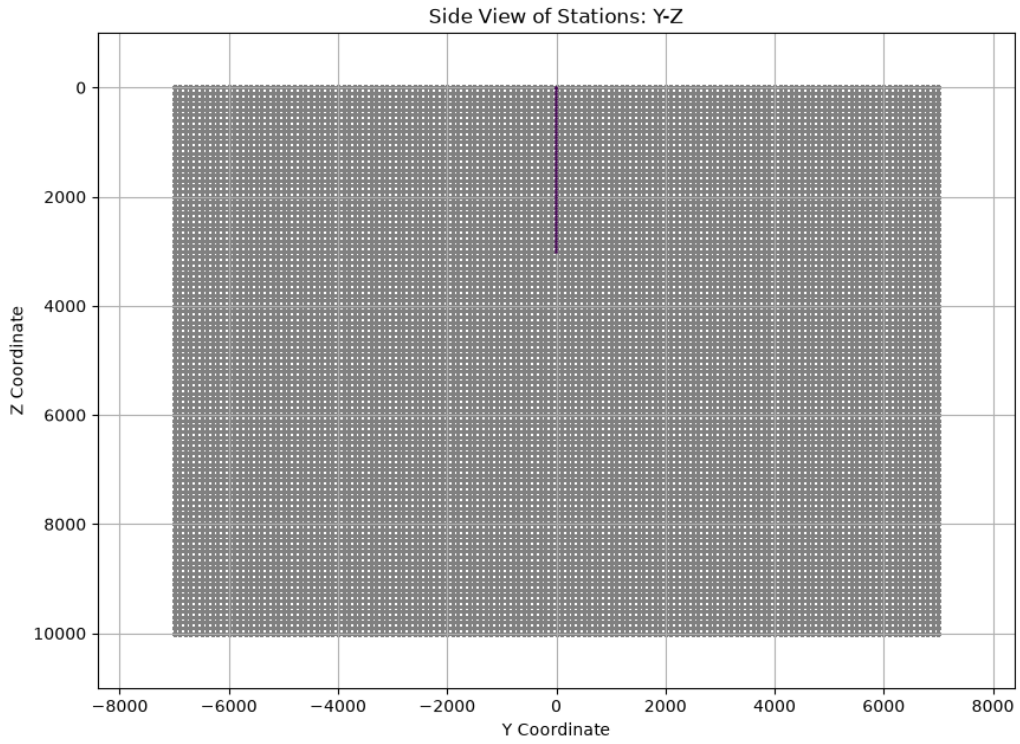
(continued from previous page)

```
                                    plot_names=False,
                                    grid_marker_size = 1,
                                    station_marker_size=1,
                                    view = 'yz')
```



•


Side View of Stations: X-Z
Z Coordinate
X Coordinate
0
2000
4000
6000
8000
10000
−8000
−6000
−4000
−2000
0
2000
4000
6000
8000



-

3.9.3 Prepare to compute magnitude sensitivity

```
# Get medium parameters
density = velocity_df.at[0,'Rho']
v_p = velocity_df.at[0,'Vp']
Q_p = velocity_df.at[0,'Qp']
v_s = velocity_df.at[0,'Vp']/velocity_df.at[0,'VpVsRatio']
Q_s = velocity_df.at[0,'Qs']
print("density, v_p, Q_p, v_s, Q_s:",density,v_p,Q_p,v_s,Q_s)

# Specify source parameters
frequency = 10 # Hz

# Minimum S/N for detection on individual channel
min_SNR_p = 2
```

(continues on next page)

(continued from previous page)

```
min_SNR_s = 2

# Minimum stations on which an event must be detected
min_stations_p = 3
min_stations_s = 3

# Save parameters
params = pndmod.classes.DetectionParameters(
    f_p=frequency,
    f_s=frequency,
    min_SNR_p=min_SNR_p,
    min_SNR_s=min_SNR_s,
    min_stations_p=min_stations_p,
    min_stations_s=min_stations_s,
)

# Define slices to use
slice_X = 0
slice_Y = 0
slice_Z = 2400

# Get slice planes
X_plane_points, X_plane_indices = pndmod.utils.get_slice_coordinates(grid_points=grid_
↪points,slice_dimension=0,slice_coordinate=slice_X)
Y_plane_points, Y_plane_indices = pndmod.utils.get_slice_coordinates(grid_points=grid_
↪points,slice_dimension=1,slice_coordinate=slice_Y)
Z_plane_points, Z_plane_indices = pndmod.utils.get_slice_coordinates(grid_points=grid_
↪points,slice_dimension=2,slice_coordinate=slice_Z)
```

```
density, v_p, Q_p, v_s, Q_s: 2300 4300 100 2485.549132947977 50
```

3.9.4 Compute magnitude sensitivity

```
print("Computing magnitude sensitivity...")


start_time = time()
X_slice_mag_sens_p = pndmod.core.get_mag_sensitivity(grid_coords=X_plane_points,
                                                     geometry_df=combined_geometry_df,
                                                     velocity_df=velocity_df,
                                                     params=params,
                                                     wave_mode='P')
X_slice_mag_sens_s = pndmod.core.get_mag_sensitivity(grid_coords=X_plane_points,
                                                     geometry_df=combined_geometry_df,
                                                     velocity_df=velocity_df,
                                                     params=params,
                                                     wave_mode='S')
X_slice_mag_sens_ps = pndmod.core.get_mag_sensitivity(grid_coords=X_plane_points,
                                                      geometry_df=combined_geometry_
↪df,
                                                      velocity_df=velocity_df,
```

(continues on next page)

(continued from previous page)

```
                                                    params=params,
                                                    wave_mode='PS')
Y_slice_mag_sens_p = pndmod.core.get_mag_sensitivity(grid_coords=Y_plane_points,
                                                   geometry_df=combined_geometry_df,
                                                   velocity_df=velocity_df,
                                                   params=params,
                                                   wave_mode='P')
Y_slice_mag_sens_s = pndmod.core.get_mag_sensitivity(grid_coords=Y_plane_points,
                                                   geometry_df=combined_geometry_df,
                                                   velocity_df=velocity_df,
                                                   params=params,
                                                   wave_mode='S')
Y_slice_mag_sens_ps = pndmod.core.get_mag_sensitivity(grid_coords=Y_plane_points,
                                                    geometry_df=combined_geometry_
↪df,
                                                    velocity_df=velocity_df,
                                                    params=params,
                                                    wave_mode='PS')
Z_slice_mag_sens_p = pndmod.core.get_mag_sensitivity(grid_coords=Z_plane_points,
                                                   geometry_df=combined_geometry_df,
                                                   velocity_df=velocity_df,
                                                   params=params,
                                                   wave_mode='P')
Z_slice_mag_sens_s = pndmod.core.get_mag_sensitivity(grid_coords=Z_plane_points,
                                                   geometry_df=combined_geometry_df,
                                                   velocity_df=velocity_df,
                                                   params=params,
                                                   wave_mode='S')
Z_slice_mag_sens_ps = pndmod.core.get_mag_sensitivity(grid_coords=Z_plane_points,
                                                    geometry_df=combined_geometry_
↪df,
                                                    velocity_df=velocity_df,
                                                    params=params,
                                                    wave_mode='PS')
end_time = time()
print(f"Computation time: {end_time - start_time} seconds")
```

```
Computing magnitude sensitivity...
Computation time: 3.1189138889312744 seconds
```

3.9.5 Plot magnitude sensitivity

Plot magnitude sensitivity for P wave

```
station_marker_size = 25
clb_range = None
plt_levels = None

_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= X_slice_mag_sens_p, yi=gy,␣
↪zi=gz,
                                              plot_title=f"Magnitude sensitivity at X=
```

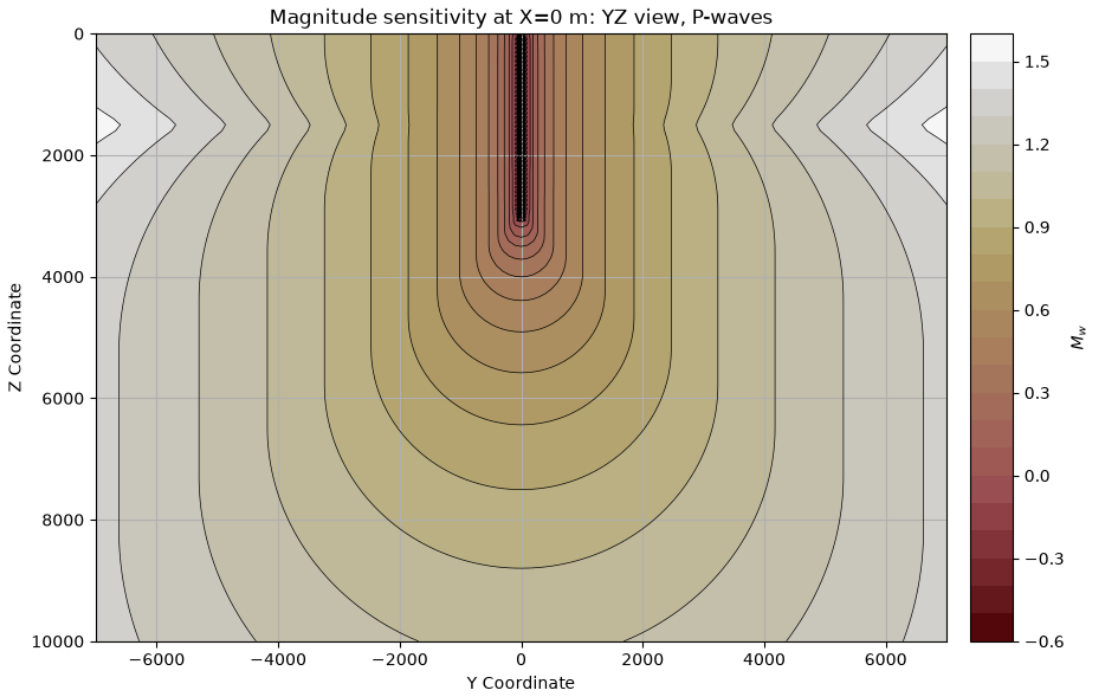
(continues on next page)

(continued from previous page)

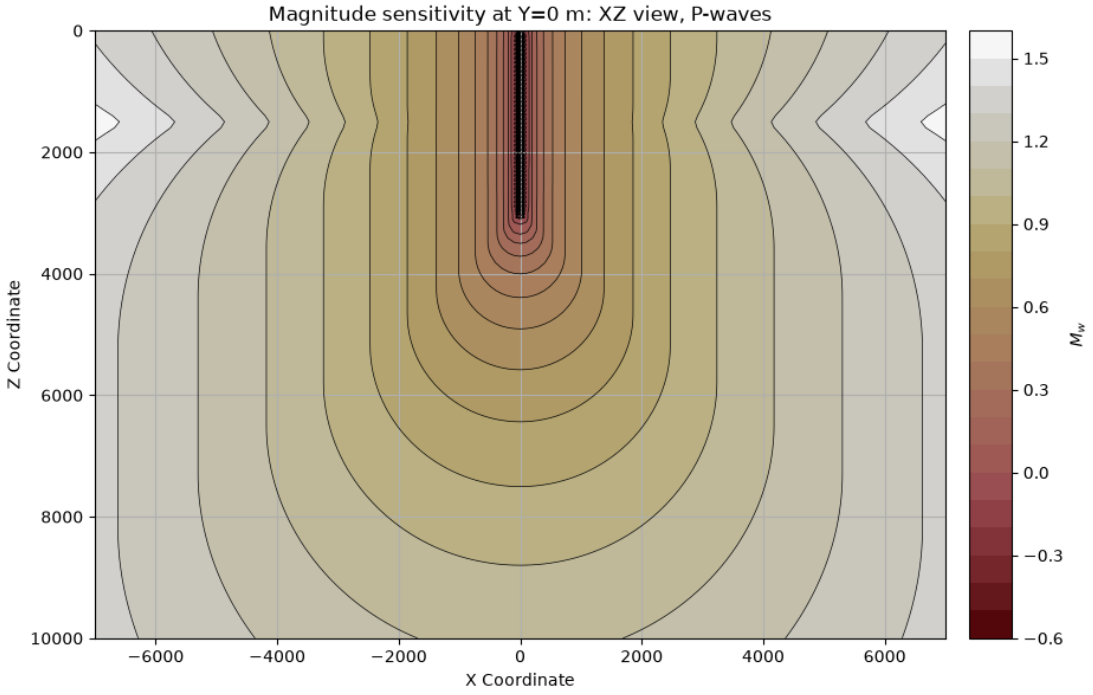
```
↪{slice_X} m: YZ view, P-waves",
                                                clb_title="$M_w$",
                                                geometry_df=combined_geometry_df,
                                                plot_names = False,
                                                station_marker_size = station_marker_
↪size,
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)

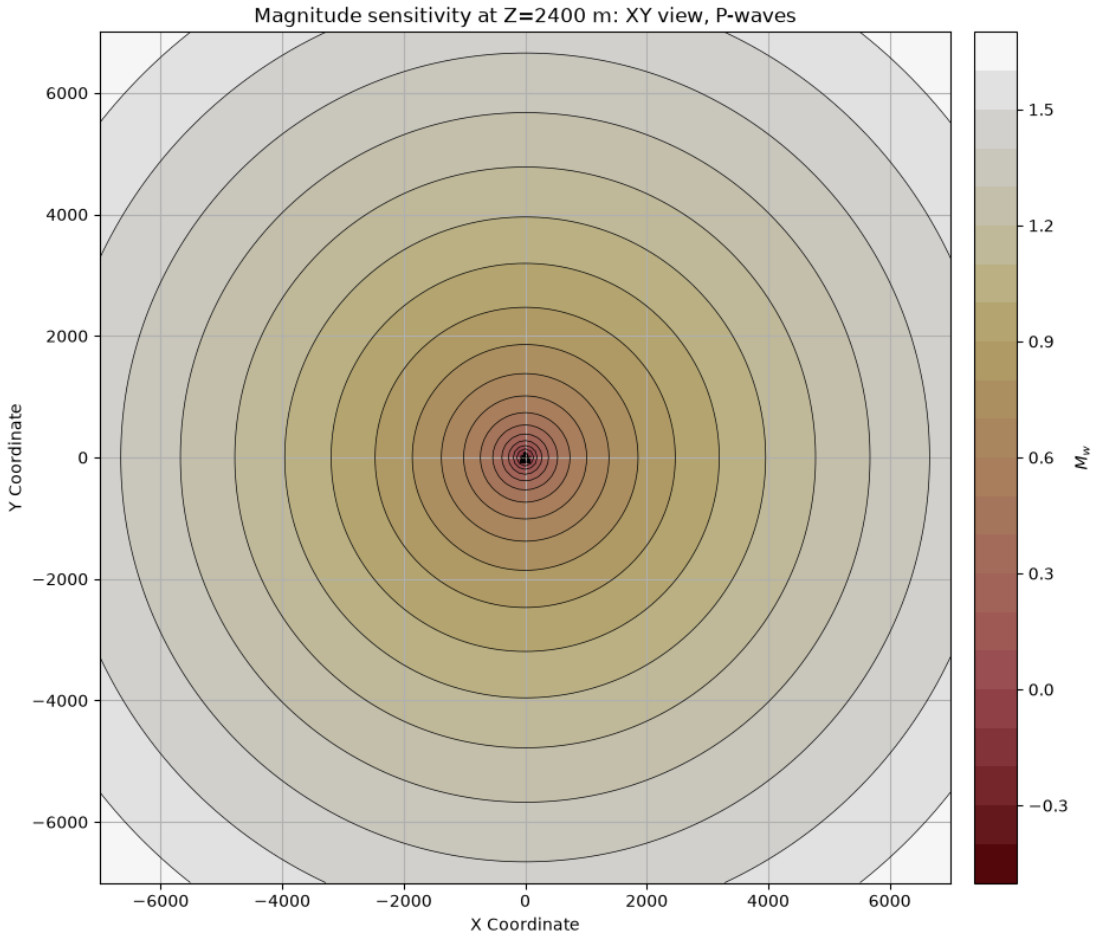
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= Y_slice_mag_sens_p, xi=gx,␣
↪zi=gz,
                                                plot_title=f"Magnitude sensitivity at Y=
↪{slice_Y} m: XZ view, P-waves",
                                                clb_title="$M_w$",
                                                geometry_df=combined_geometry_df,
                                                plot_names = False,
                                                station_marker_size = station_marker_
↪size,
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)

_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= Z_slice_mag_sens_p, xi=gx,␣
↪yi=gy,
                                                plot_title=f"Magnitude sensitivity at Z=
↪{slice_Z} m: XY view, P-waves",
                                                clb_title="$M_w$",
                                                geometry_df=combined_geometry_df,
                                                plot_names = False,
                                                station_marker_size = station_marker_
↪size,
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)
```


Magnitude sensitivity at X=0 m: YZ view, P-waves
Z Coordinate
Y Coordinate
0
2000
4000
6000
8000
10000
−6000
−4000
−2000
0
2000
4000
6000
1.5
1.2
0.9
0.6
0.3
0.0
−0.3
−0.6
M_w

•


Magnitude sensitivity at Y=0 m: XZ view, P-waves
Z Coordinate
X Coordinate
0
2000
4000
6000
8000
10000
−6000
−4000
−2000
0
2000
4000
6000
1.5
1.2
0.9
0.6
0.3
0.0
−0.3
−0.6
M_w



-

Plot magnitude sensitivity for S wave

```
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= X_slice_mag_sens_s, yi=gy,␣
↪zi=gz,
                                              plot_title=f"Magnitude sensitivity at X=
↪{slice_X} m: YZ view, S-waves",
                                              clb_title="$M_w$",
                                              geometry_df=combined_geometry_df,
                                              plot_names = False,
                                              station_marker_size = station_marker_
↪size,
                                              clb_range=clb_range,
                                              plt_levels=plt_levels)

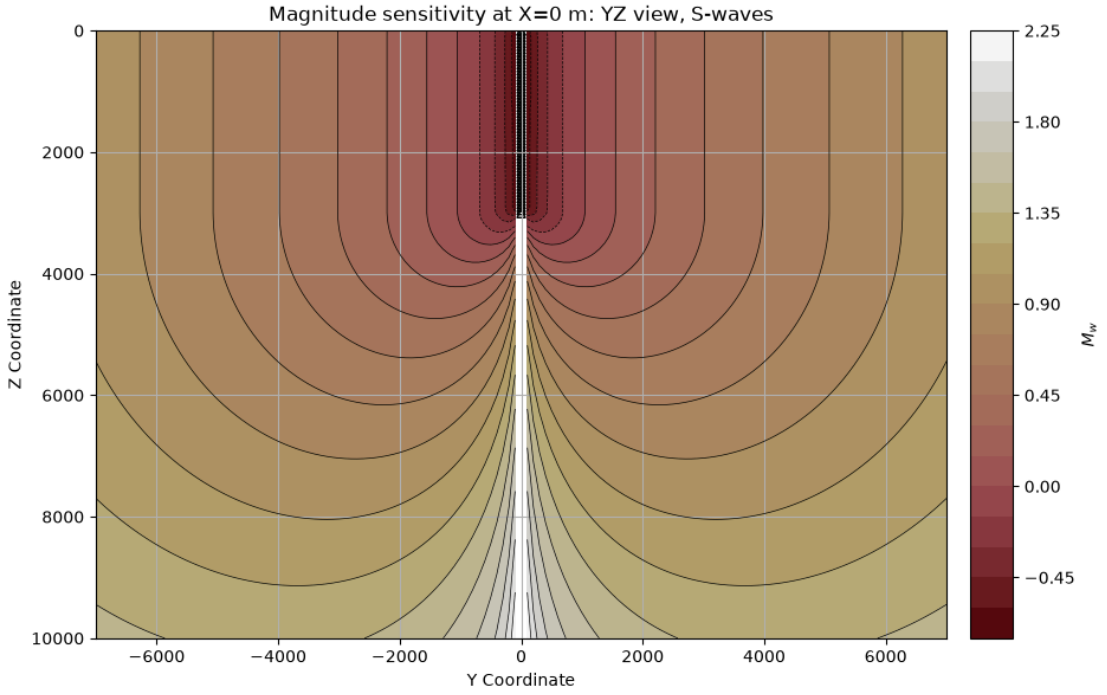
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= Y_slice_mag_sens_s, xi=gx,␣
```

(continues on next page)

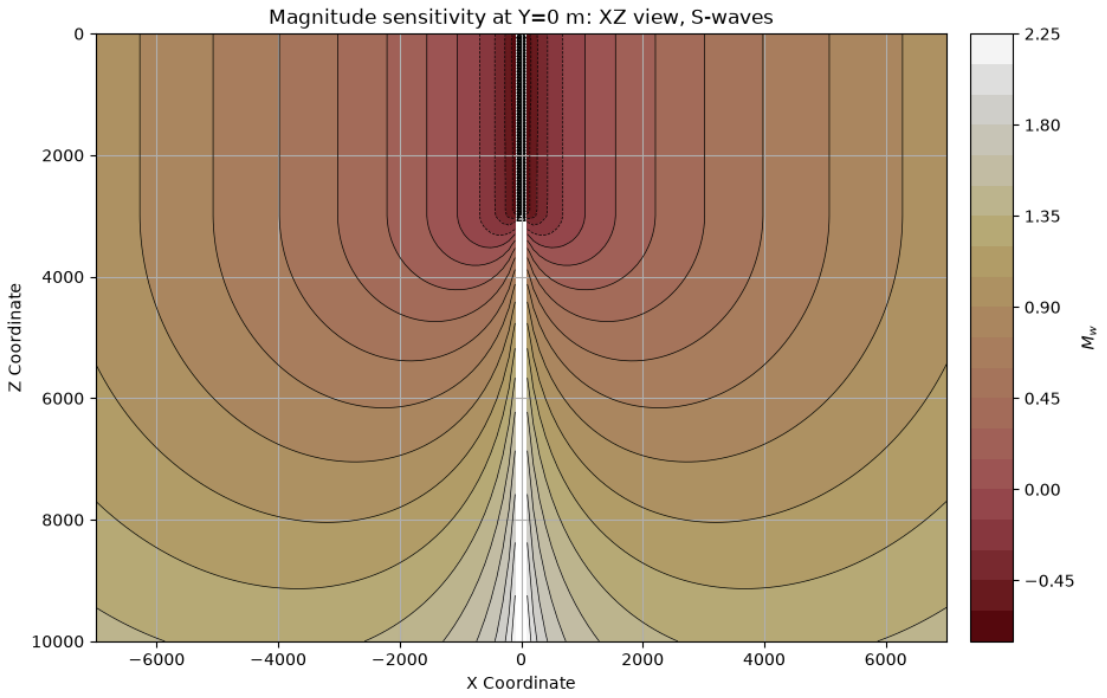
(continued from previous page)

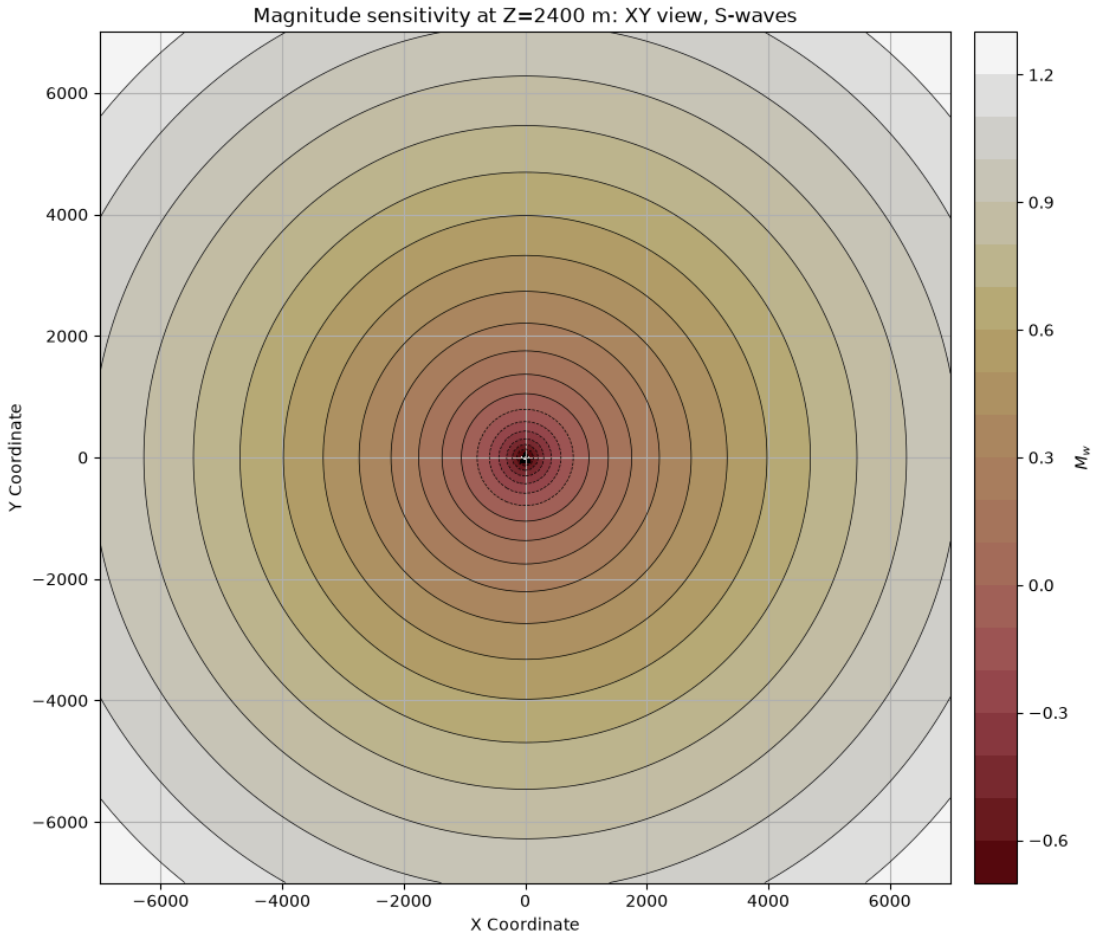
```
↪zi=gz,
                                                plot_title=f"Magnitude sensitivity at Y=
↪{slice_Y} m: XZ view, S-waves",
                                                clb_title="$M_w$",
                                                geometry_df=combined_geometry_df,
                                                plot_names = False,
                                                station_marker_size = station_marker_
↪size,
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)

_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= Z_slice_mag_sens_s, xi=gx,␣
↪yi=gy,
                                                plot_title=f"Magnitude sensitivity at Z=
↪{slice_Z} m: XY view, S-waves",
                                                clb_title="$M_w$",
                                                geometry_df=combined_geometry_df,
                                                plot_names = False,
                                                station_marker_size = station_marker_
↪size,
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)
```


Magnitude sensitivity at X=0 m: YZ view, S-waves
Z Coordinate
Y Coordinate
0
2000
4000
6000
8000
10000
−6000
−4000
−2000
0
2000
4000
6000
2.25
1.80
1.35
0.90
0.45
0.00
−0.45
M_w

•


Magnitude sensitivity at Y=0 m: XZ view, S-waves
Z Coordinate
X Coordinate
0
2000
4000
6000
8000
10000
−6000
−4000
−2000
0
2000
4000
6000
2.25
1.80
1.35
0.90
0.45
0.00
−0.45
M_w



-

Plot magnitude sensitivity for P and S waves

```
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= X_slice_mag_sens_ps, yi=gy,␣
↪zi=gz,
                                              plot_title=f"Magnitude sensitivity at X=
↪{slice_X} m: YZ view, P- and S-waves",
                                              clb_title="$M_w$",
                                              geometry_df=combined_geometry_df,
                                              plot_names = False,
                                              station_marker_size = station_marker_
↪size,
                                              clb_range=clb_range,
                                              plt_levels=plt_levels)

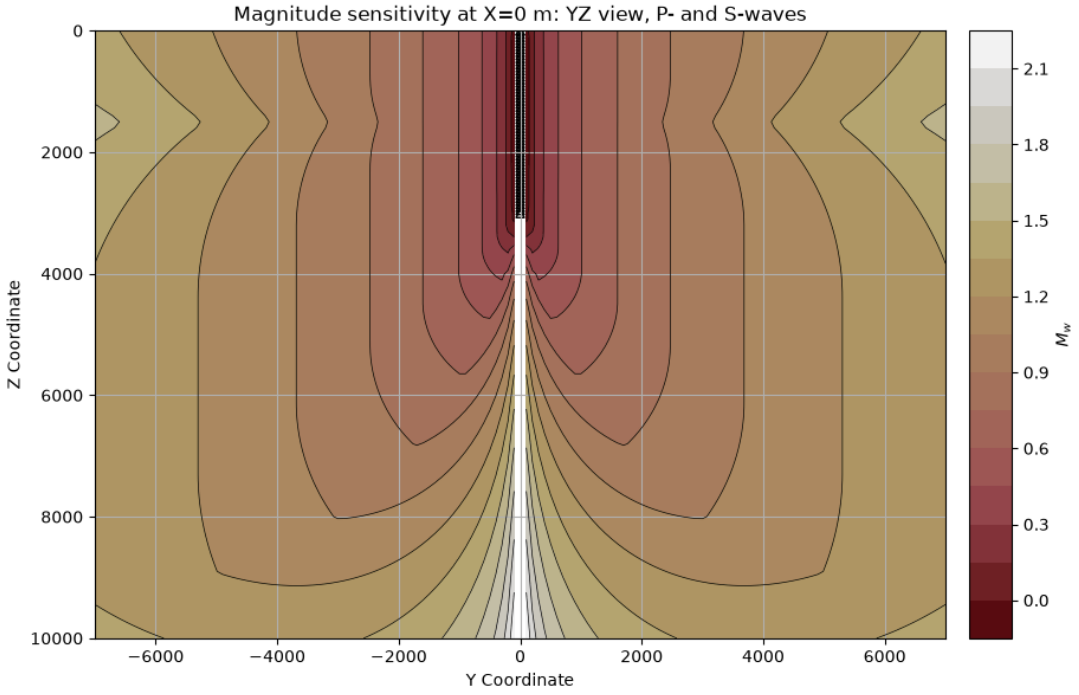
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= Y_slice_mag_sens_ps, xi=gx,␣
```

(continues on next page)

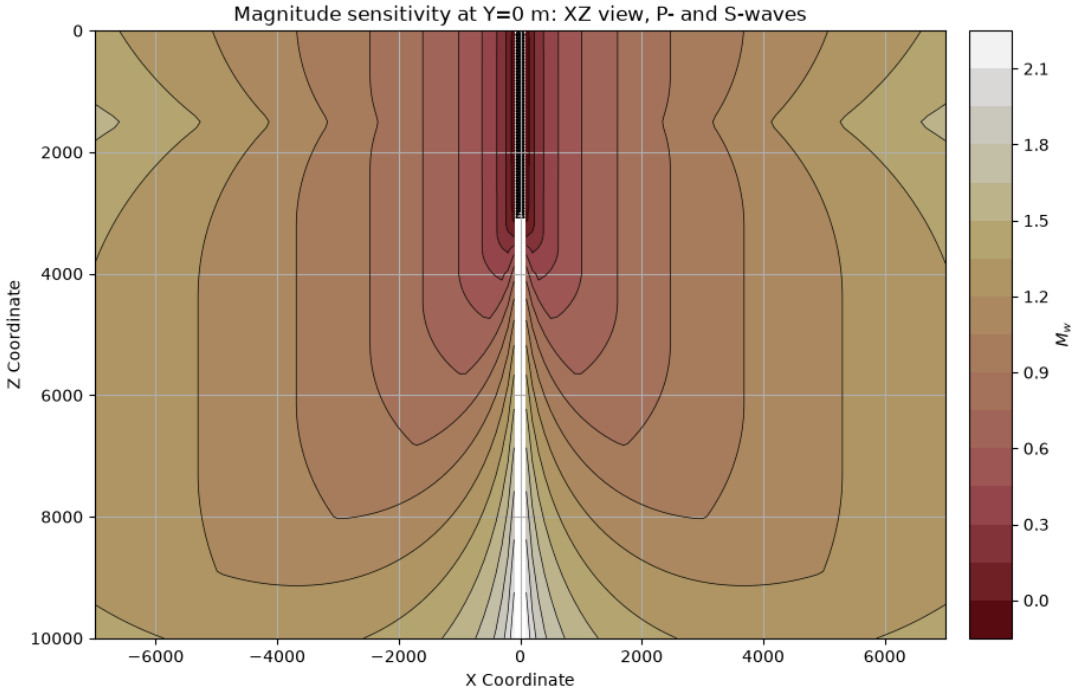
(continued from previous page)

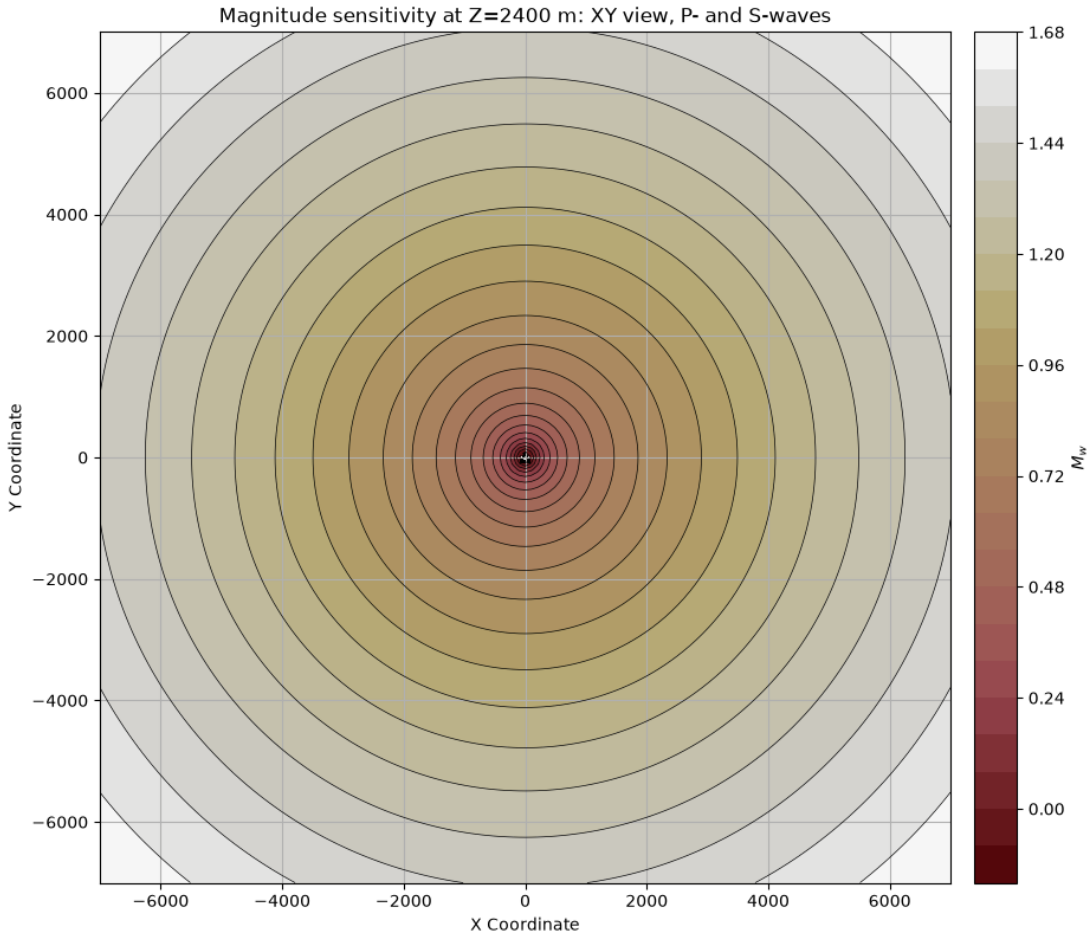
```
↪zi=gz,
                                                plot_title=f"Magnitude sensitivity at Y=
↪{slice_Y} m: XZ view, P- and S-waves",
                                                clb_title="$M_w$",
                                                geometry_df=combined_geometry_df,
                                                plot_names = False,
                                                station_marker_size = station_marker_
↪size,
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)

_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= Z_slice_mag_sens_ps, xi=gx,␣
↪yi=gy,
                                                plot_title=f"Magnitude sensitivity at Z=
↪{slice_Z} m: XY view, P- and S-waves",
                                                clb_title="$M_w$",
                                                geometry_df=combined_geometry_df,
                                                plot_names = False,
                                                station_marker_size = station_marker_
↪size,
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)
```


Magnitude sensitivity at X=0 m: YZ view, P- and S-waves
Z Coordinate
Y Coordinate
0
2000
4000
6000
8000
10000
−6000
−4000
−2000
0
2000
4000
6000
2.1
1.8
1.5
1.2
0.9
0.6
0.3
0.0
M_w

•


Magnitude sensitivity at Y=0 m: XZ view, P- and S-waves
Z Coordinate
X Coordinate
0
2000
4000
6000
8000
10000
−6000
−4000
−2000
0
2000
4000
6000
2.1
1.8
1.5
1.2
0.9
0.6
0.3
0.0
M_w



-

Total running time of the script: (0 minutes 8.342 seconds)

3.10 10.1 Magnitude sensitivity for homogeneous model: test case 10

This is an example of computing magnitude sensitivity for a homogeneous model.

Test case 10:

- Geometry: a single station above the DAS cable in a deep vertical borehole
- Data: Noise level at the station: 5e-9 m/s, for DAS - noise level decreasing with depth (from 5e-7 to 8.87e-12)
- Velocity model: homogeneous, Vp = 4300 m/s, Vp/Vs = 1.73, Qp = 100, Qs = 50, rho = 2300 kg/m^3
- Processing: magnitude sensitivity computation for all stations (for grid slices only).

3.10.1 Load all necessary packages

```
import pynetdesign.visualization as pndvis
import pynetdesign.modelling as pndmod
import numpy as np
from time import time

# sphinx_gallery_thumbnail_number = -2
```

3.10.2 Input

Read geometry

```
### Read geometry
station_geometry_file_path = '../data/homo/test_10/TMPL_net_geometry_Single.txt'
station_geometry_df = pndmod.io.read_geometry(station_geometry_file_path)
print(station_geometry_df)
print(station_geometry_df.attrs)

das_geometry_file_path = '../data/homo/test_10/TMPL_net_geometry_Noise_Decreasing.txt'
das_geometry_df = pndmod.io.read_geometry(das_geometry_file_path)
print(das_geometry_df)
print(das_geometry_df.attrs)

### Show
print("\nStation geometry:")
print(station_geometry_df.head())
print(station_geometry_df.attrs)
print("\nDAS geometry:")
print(das_geometry_df.head())
print(das_geometry_df.attrs)
```

```
  Name    Y    X    Z    NoiseLevel
0  ST1  0.0  0.0 -0.0  5.000000e-09
{'file_path': '../data/homo/test_10/TMPL_net_geometry_Single.txt', 'original_coord_
↪units': 'm', 'coordinate_system': 'local', 'noise_type': 'velocity', 'noise_unit':
↪'m/s', '_is_combined': False}
      Y    X       Z    NoiseLevel
0   0.0  0.0    10.0  5.000000e-07
1   0.0  0.0    60.0  4.170000e-07
2   0.0  0.0   110.0  3.470000e-07
3   0.0  0.0   160.0  2.890000e-07
4   0.0  0.0   210.0  2.410000e-07
..  ...  ...     ...           ...
56  0.0  0.0  2810.0  1.840000e-11
57  0.0  0.0  2860.0  1.530000e-11
58  0.0  0.0  2910.0  1.280000e-11
59  0.0  0.0  2960.0  1.060000e-11
60  0.0  0.0  3010.0  8.870000e-12

[61 rows x 4 columns]
{'file_path': '../data/homo/test_10/TMPL_net_geometry_Noise_Decreasing.txt',
↪'original_coord_units': 'm', 'coordinate_system': 'local', 'noise_type': 'strain',
```

(continues on next page)

(continued from previous page)

```
↪'noise_unit': 'ε', 'gauge_length': 10.0, '_is_combined': False}

Station geometry:
  Name    Y    X    Z    NoiseLevel
0  ST1  0.0  0.0 -0.0  5.000000e-09
{'file_path': '../data/homo/test_10/TMPL_net_geometry_Single.txt', 'original_coord_
↪units': 'm', 'coordinate_system': 'local', 'noise_type': 'velocity', 'noise_unit':
↪'m/s', '_is_combined': False}

DAS geometry:
     Y    X      Z    NoiseLevel
0  0.0  0.0   10.0  5.000000e-07
1  0.0  0.0   60.0  4.170000e-07
2  0.0  0.0  110.0  3.470000e-07
3  0.0  0.0  160.0  2.890000e-07
4  0.0  0.0  210.0  2.410000e-07
{'file_path': '../data/homo/test_10/TMPL_net_geometry_Noise_Decreasing.txt',
↪'original_coord_units': 'm', 'coordinate_system': 'local', 'noise_type': 'strain',
↪'noise_unit': 'ε', 'gauge_length': 10.0, '_is_combined': False}
```

Combine geometry

```
combined_geometry_df = pndmod.io.combine_geometry(station_geometry_df,
                                                  das_geometry_df,
                                                  names=['Stations',
                                                         'DAS'])


### Show combined
print("\nCombined geometry:")
print(combined_geometry_df)
print(combined_geometry_df.attrs)
print("\nGeometry names:")
print(combined_geometry_df.get_names())
print("\nCombined status:",
      pndmod.io.is_combined_geometry(combined_geometry_df))

### Show recovered
print("\nStation geometry (recovered):")
print(combined_geometry_df.recover('Stations').head())
print("\nCombined status:",
      pndmod.io.is_combined_geometry(combined_geometry_df.recover('Stations')))

print("\nDAS geometry (recovered):")
print(combined_geometry_df.recover('DAS').head())
print("\nCombined status:",
      pndmod.io.is_combined_geometry(combined_geometry_df.recover('DAS')))
print("\nAttributes of DAS (recovered):")
print(combined_geometry_df.recover('DAS').attrs)
```

```
Combined geometry:
           Name    Y    X       Z    NoiseLevel
```

(continues on next page)

(continued from previous page)

```
source
Stations 0   ST1  0.0  0.0    -0.0  5.000000e-09
DAS      0   NaN  0.0  0.0    10.0  5.000000e-07
         1   NaN  0.0  0.0    60.0  4.170000e-07
         2   NaN  0.0  0.0   110.0  3.470000e-07
         3   NaN  0.0  0.0   160.0  2.890000e-07
...          ...  ...  ...     ...           ...
         56  NaN  0.0  0.0  2810.0  1.840000e-11
         57  NaN  0.0  0.0  2860.0  1.530000e-11
         58  NaN  0.0  0.0  2910.0  1.280000e-11
         59  NaN  0.0  0.0  2960.0  1.060000e-11
         60  NaN  0.0  0.0  3010.0  8.870000e-12

[62 rows x 5 columns]
{'original_attributes': {'Stations': {'file_path': '../data/homo/test_10/TMPL_net_
↪geometry_Single.txt', 'original_coord_units': 'm', 'coordinate_system': 'local',
↪'noise_type': 'velocity', 'noise_unit': 'm/s', '_is_combined': False}, 'DAS': {
↪'file_path': '../data/homo/test_10/TMPL_net_geometry_Noise_Decreasing.txt',
↪'original_coord_units': 'm', 'coordinate_system': 'local', 'noise_type': 'strain',
↪'noise_unit': 'ε', 'gauge_length': 10.0, '_is_combined': False}}, '_is_combined':␣
↪True}

Geometry names:
['Stations', 'DAS']

Combined status: True

Station geometry (recovered):
  Name    Y    X    Z    NoiseLevel
0  ST1  0.0  0.0 -0.0  5.000000e-09

Combined status: False

DAS geometry (recovered):
     Y    X      Z    NoiseLevel
0  0.0  0.0   10.0  5.000000e-07
1  0.0  0.0   60.0  4.170000e-07
2  0.0  0.0  110.0  3.470000e-07
3  0.0  0.0  160.0  2.890000e-07
4  0.0  0.0  210.0  2.410000e-07

Combined status: False

Attributes of DAS (recovered):
{'file_path': '../data/homo/test_10/TMPL_net_geometry_Noise_Decreasing.txt',
↪'original_coord_units': 'm', 'coordinate_system': 'local', 'noise_type': 'strain',
↪'noise_unit': 'ε', 'gauge_length': 10.0, '_is_combined': False}
```

Read velocity model

```
velocity_file_path = '../data/homo/test_10/TMPL_velocity_model.txt'
velocity_df = pndmod.io.read_velocity(velocity_file_path)

print(velocity_df)
print(velocity_df.attrs)
```

```
   Depth    Vp  VpVsRatio   Qp  Qs   Rho   Ep   Dt   Gm
0  40000  4300       1.73  100  50  2300  0.0  0.0  0.0
{'model_type': 'homo', 'model_anisotropy': 'isotropic', 'parameter_descriptions': {
↪'Depth': 'Depth', 'Vp': 'P-wave velocity', 'Vs': 'S-wave velocity', 'VpVsRatio':
↪'Vp/Vs', 'Qp': 'P-wave quality factor', 'Qs': 'S-wave quality factor', 'Rho':
↪'density', 'Ep': 'Thomsen epsilon parameter', 'Dt': 'Thomsen delta parameter', 'Gm
↪': 'Thomsen gamma parameter'}, 'parameter_units': {'Depth': 'm', 'Vp': 'm/s', 'Vs':
↪'m/s', 'VpVsRatio': '', 'Qp': '', 'Qs': '', 'Rho': 'kg/m^3', 'Ep': '', 'Dt': '', 'Gm
↪': ''}}
```

Set imaging grid parameters

```
# Imaging grid parameters
gx = np.arange(-7000, 7000 + 100, 100)
gy = np.arange(-7000, 7000 + 100, 100)
gz = np.arange(0, 10000 + 100, 100)

# Generate grid points
grid_points = pndmod.utils.generate_grid(x=gx,y=gy,z=gz)
print("Number of grid points:",grid_points.size)
print("Number of grid points in x:",gx.size)
print("Number of grid points in y:",gy.size)
print("Number of grid points in z:",gz.size)
print("grid_points.shape:",grid_points.shape)
```

```
Number of grid points: 6023943
Number of grid points in x: 141
Number of grid points in y: 141
Number of grid points in z: 101
grid_points.shape: (2007981, 3)
```

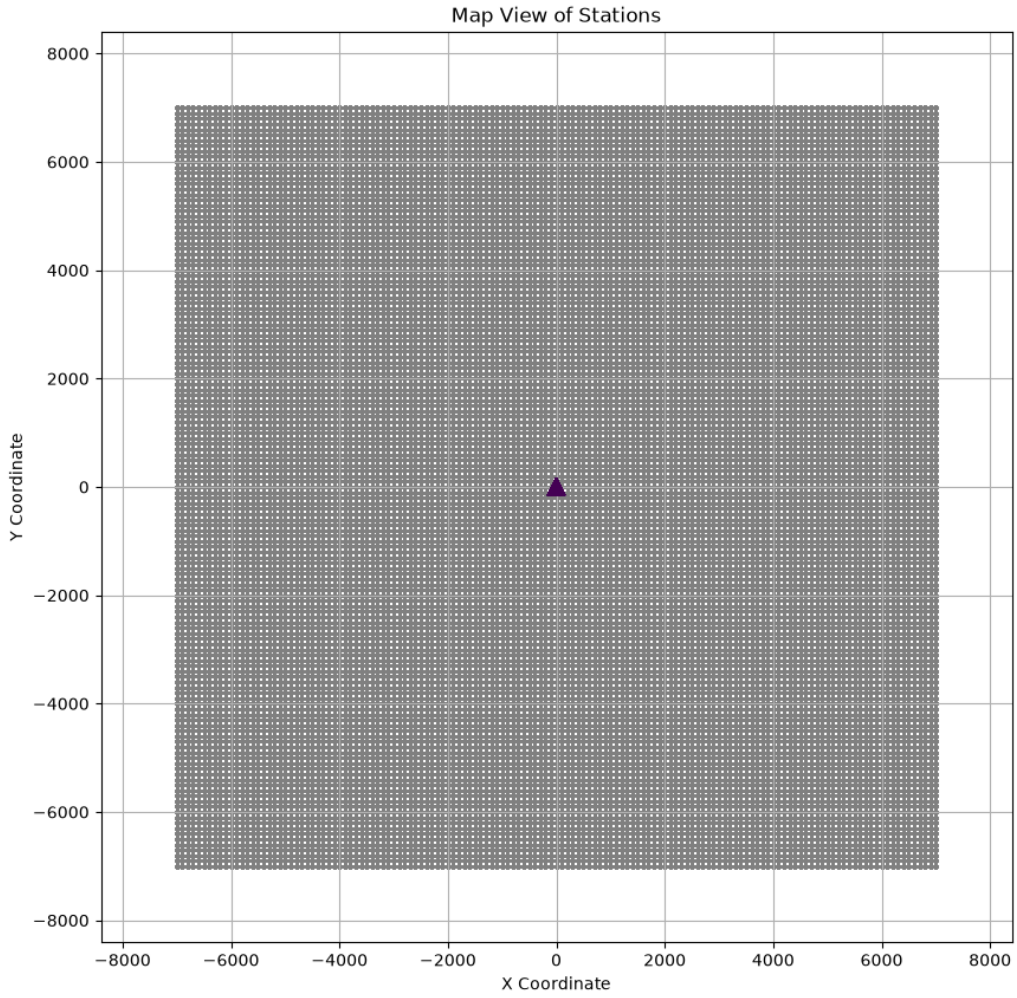
Plot geometry and imaging grid

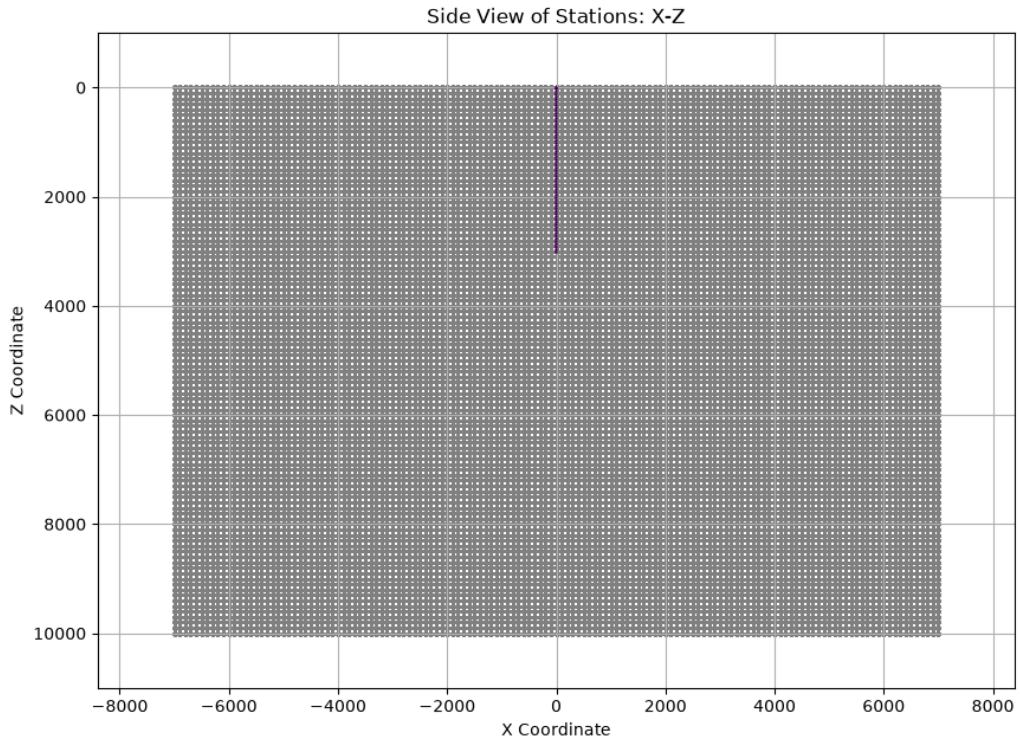
```
_ = pndvis.geometry.plot_stations_view(df=combined_geometry_df,
                                       grid_points=grid_points,
                                       plot_names=False,
                                       grid_marker_size = 1,
                                       station_marker_size=100)
_ = pndvis.geometry.plot_stations_view(df=combined_geometry_df,
                                       grid_points=grid_points,
                                       plot_names=False,
                                       grid_marker_size = 1,
                                       station_marker_size=1,
                                       view = 'xz')
```

(continues on next page)

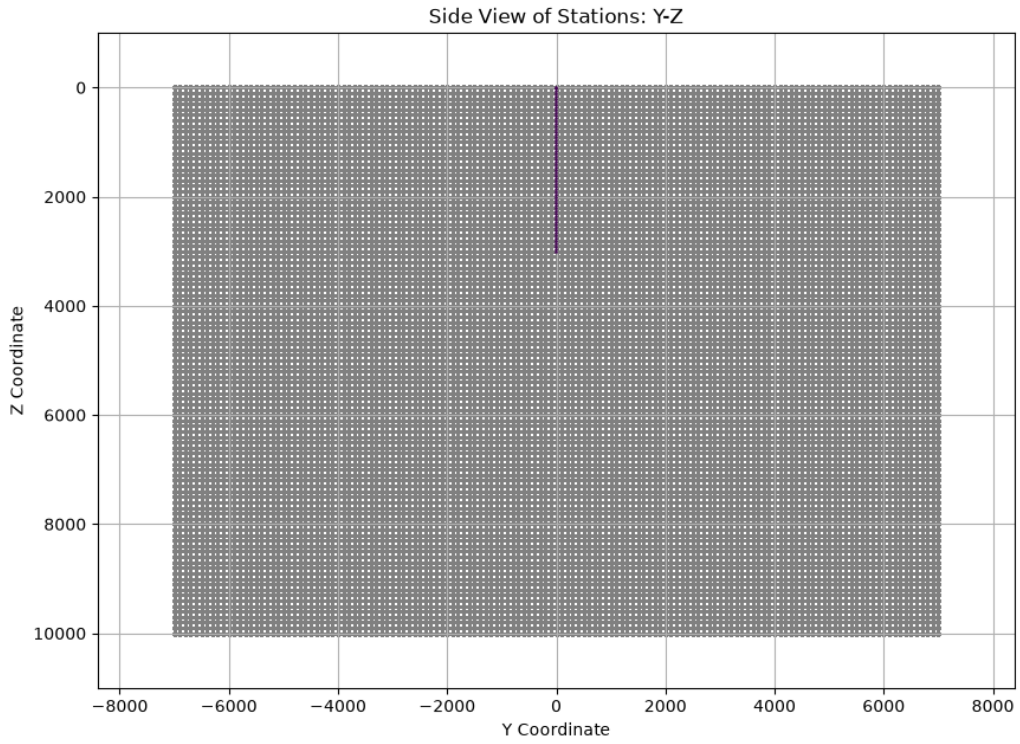
(continued from previous page)

```
_ = pndvis.geometry.plot_stations_view(df=combined_geometry_df,
                                       grid_points=grid_points,
                                       plot_names=False,
                                       grid_marker_size = 1,
                                       station_marker_size=1,
                                       view = 'yz')
```




Side View of Stations: X-Z
Z Coordinate
X Coordinate
0
2000
4000
6000
8000
10000
−8000
−6000
−4000
−2000
0
2000
4000
6000
8000

•



-

3.10.3 Prepare to compute magnitude sensitivity

```
# Get medium parameters
density = velocity_df.at[0,'Rho']
v_p = velocity_df.at[0,'Vp']
Q_p = velocity_df.at[0,'Qp']
v_s = velocity_df.at[0,'Vp']/velocity_df.at[0,'VpVsRatio']
Q_s = velocity_df.at[0,'Qs']
print("density, v_p, Q_p, v_s, Q_s:",density,v_p,Q_p,v_s,Q_s)

# Specify source parameters
frequency = 10 # Hz

# Minimum S/N for detection on individual channel
min_SNR_p = 2
```

(continues on next page)

(continued from previous page)

```
min_SNR_s = 2

# Minimum stations on which an event must be detected
min_stations_p = 3
min_stations_s = 3

# Save parameters
params = pndmod.classes.DetectionParameters(
    f_p=frequency,
    f_s=frequency,
    min_SNR_p=min_SNR_p,
    min_SNR_s=min_SNR_s,
    min_stations_p=min_stations_p,
    min_stations_s=min_stations_s,
)

# Define slices to use
slice_X = 0
slice_Y = 0
slice_Z = 2400

# Get slice planes
X_plane_points, X_plane_indices = pndmod.utils.get_slice_coordinates(grid_points=grid_
↪points,slice_dimension=0,slice_coordinate=slice_X)
Y_plane_points, Y_plane_indices = pndmod.utils.get_slice_coordinates(grid_points=grid_
↪points,slice_dimension=1,slice_coordinate=slice_Y)
Z_plane_points, Z_plane_indices = pndmod.utils.get_slice_coordinates(grid_points=grid_
↪points,slice_dimension=2,slice_coordinate=slice_Z)
```

```
density, v_p, Q_p, v_s, Q_s: 2300 4300 100 2485.549132947977 50
```

3.10.4 Compute magnitude sensitivity

```
print("Computing magnitude sensitivity...")


start_time = time()
X_slice_mag_sens_p = pndmod.core.get_mag_sensitivity(grid_coords=X_plane_points,
                                                     geometry_df=combined_geometry_df,
                                                     velocity_df=velocity_df,
                                                     params=params,
                                                     wave_mode='P')
X_slice_mag_sens_s = pndmod.core.get_mag_sensitivity(grid_coords=X_plane_points,
                                                     geometry_df=combined_geometry_df,
                                                     velocity_df=velocity_df,
                                                     params=params,
                                                     wave_mode='S')
X_slice_mag_sens_ps = pndmod.core.get_mag_sensitivity(grid_coords=X_plane_points,
                                                      geometry_df=combined_geometry_
↪df,
                                                      velocity_df=velocity_df,
```

(continues on next page)

(continued from previous page)

```
                                                    params=params,
                                                    wave_mode='PS')
Y_slice_mag_sens_p = pndmod.core.get_mag_sensitivity(grid_coords=Y_plane_points,
                                                   geometry_df=combined_geometry_df,
                                                   velocity_df=velocity_df,
                                                   params=params,
                                                   wave_mode='P')
Y_slice_mag_sens_s = pndmod.core.get_mag_sensitivity(grid_coords=Y_plane_points,
                                                   geometry_df=combined_geometry_df,
                                                   velocity_df=velocity_df,
                                                   params=params,
                                                   wave_mode='S')
Y_slice_mag_sens_ps = pndmod.core.get_mag_sensitivity(grid_coords=Y_plane_points,
                                                    geometry_df=combined_geometry_
↪df,
                                                    velocity_df=velocity_df,
                                                    params=params,
                                                    wave_mode='PS')
Z_slice_mag_sens_p = pndmod.core.get_mag_sensitivity(grid_coords=Z_plane_points,
                                                   geometry_df=combined_geometry_df,
                                                   velocity_df=velocity_df,
                                                   params=params,
                                                   wave_mode='P')
Z_slice_mag_sens_s = pndmod.core.get_mag_sensitivity(grid_coords=Z_plane_points,
                                                   geometry_df=combined_geometry_df,
                                                   velocity_df=velocity_df,
                                                   params=params,
                                                   wave_mode='S')
Z_slice_mag_sens_ps = pndmod.core.get_mag_sensitivity(grid_coords=Z_plane_points,
                                                    geometry_df=combined_geometry_
↪df,
                                                    velocity_df=velocity_df,
                                                    params=params,
                                                    wave_mode='PS')
end_time = time()
print(f"Computation time: {end_time - start_time} seconds")
```

```
Computing magnitude sensitivity...
Computation time: 3.068803310394287 seconds
```

3.10.5 Plot magnitude sensitivity

Plot magnitude sensitivity for P wave

```
station_marker_size = 25
clb_range = None
plt_levels = None

_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= X_slice_mag_sens_p, yi=gy,␣
↪zi=gz,
                                              plot_title=f"Magnitude sensitivity at X=
```

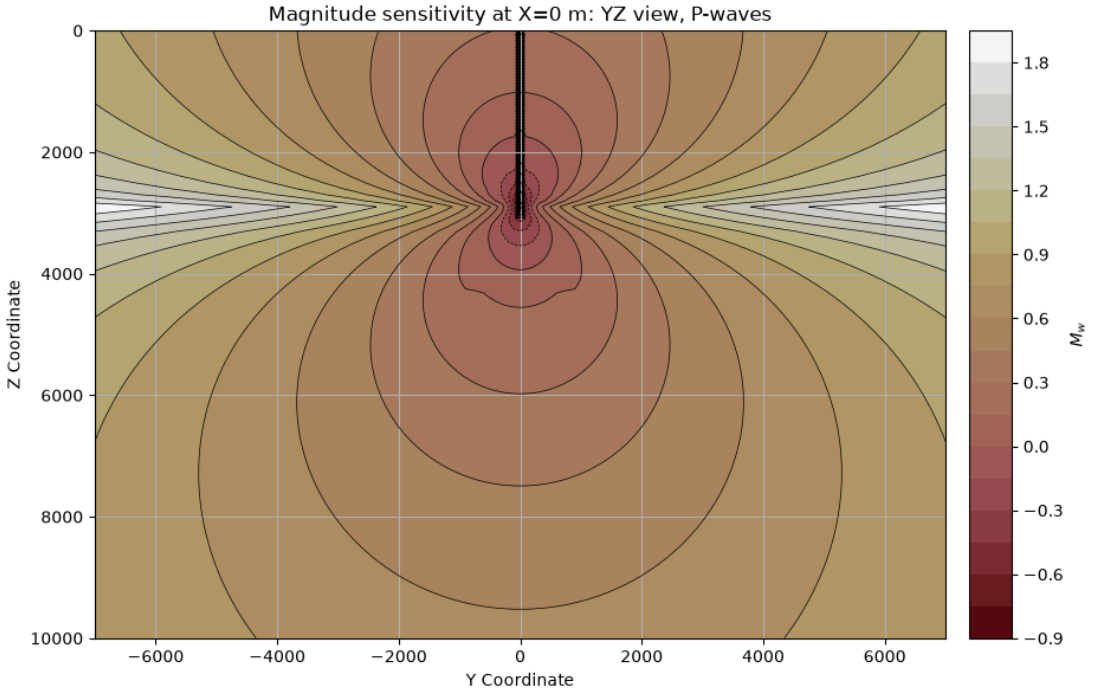
(continues on next page)

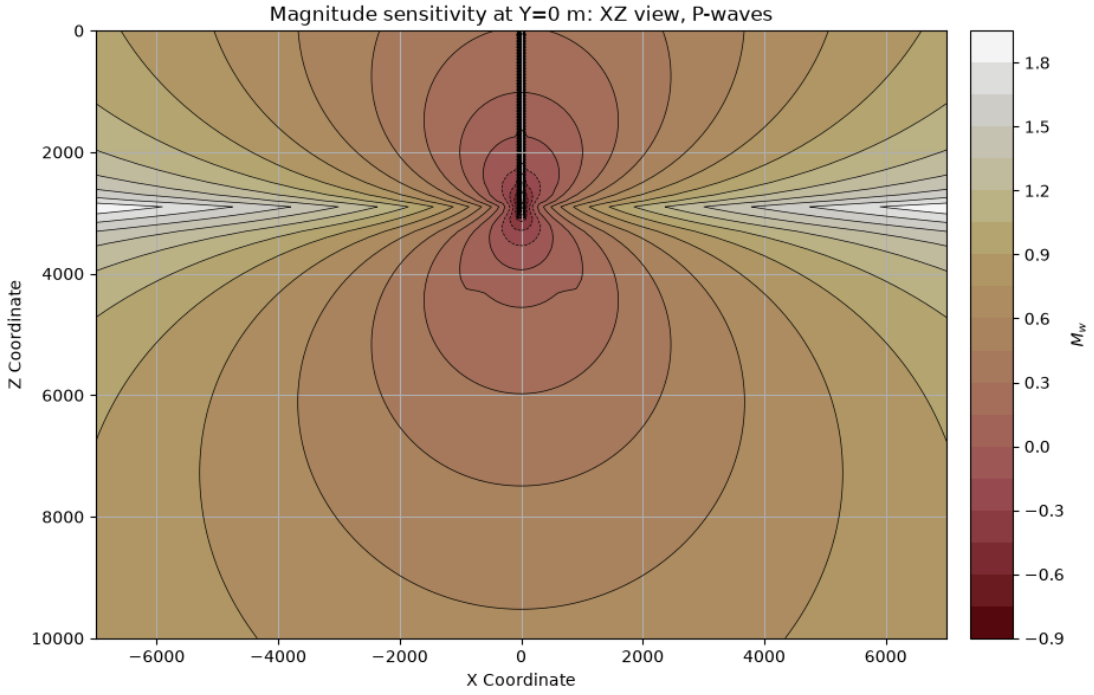
(continued from previous page)

```
↪{slice_X} m: YZ view, P-waves",
                                                clb_title="$M_w$",
                                                geometry_df=combined_geometry_df,
                                                plot_names = False,
                                                station_marker_size = station_marker_
↪size,
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)

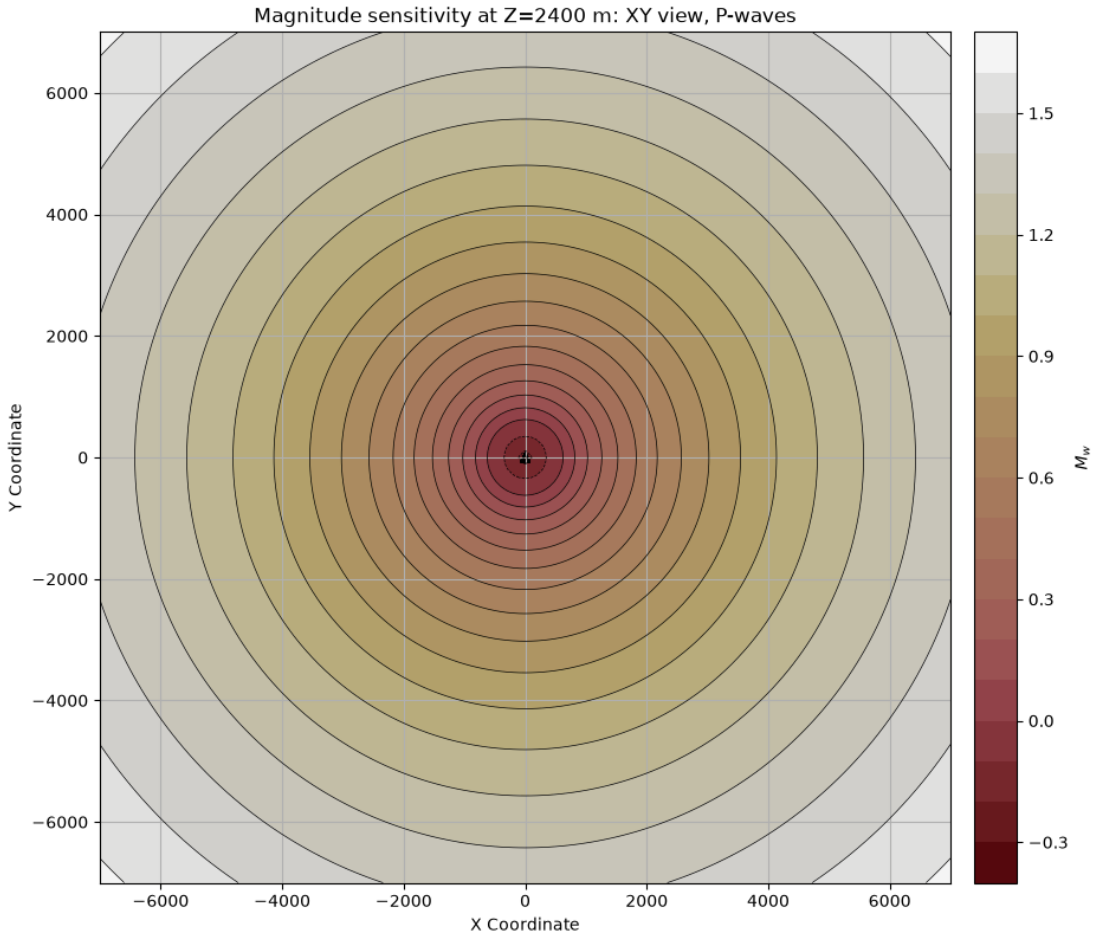
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= Y_slice_mag_sens_p, xi=gx,␣
↪zi=gz,
                                                plot_title=f"Magnitude sensitivity at Y=
↪{slice_Y} m: XZ view, P-waves",
                                                clb_title="$M_w$",
                                                geometry_df=combined_geometry_df,
                                                plot_names = False,
                                                station_marker_size = station_marker_
↪size,
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)

_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= Z_slice_mag_sens_p, xi=gx,␣
↪yi=gy,
                                                plot_title=f"Magnitude sensitivity at Z=
↪{slice_Z} m: XY view, P-waves",
                                                clb_title="$M_w$",
                                                geometry_df=combined_geometry_df,
                                                plot_names = False,
                                                station_marker_size = station_marker_
↪size,
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)
```


Magnitude sensitivity at X=0 m: YZ view, P-waves
Z Coordinate
Y Coordinate
0
2000
4000
6000
8000
10000
−6000
−4000
−2000
0
2000
4000
6000
1.8
1.5
1.2
0.9
0.6
0.3
0.0
−0.3
−0.6
−0.9
M_w


Magnitude sensitivity at Y=0 m: XZ view, P-waves
Z Coordinate
X Coordinate
0
2000
4000
6000
8000
10000
−6000
−4000
−2000
0
2000
4000
6000
1.8
1.5
1.2
0.9
0.6
0.3
0.0
−0.3
−0.6
−0.9
M_w

•



-

Plot magnitude sensitivity for S wave

```
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= X_slice_mag_sens_s, yi=gy,␣
↪zi=gz,
                                              plot_title=f"Magnitude sensitivity at X=
↪{slice_X} m: YZ view, S-waves",
                                              clb_title="$M_w$",
                                              geometry_df=combined_geometry_df,
                                              plot_names = False,
                                              station_marker_size = station_marker_
↪size,
                                              clb_range=clb_range,
                                              plt_levels=plt_levels)

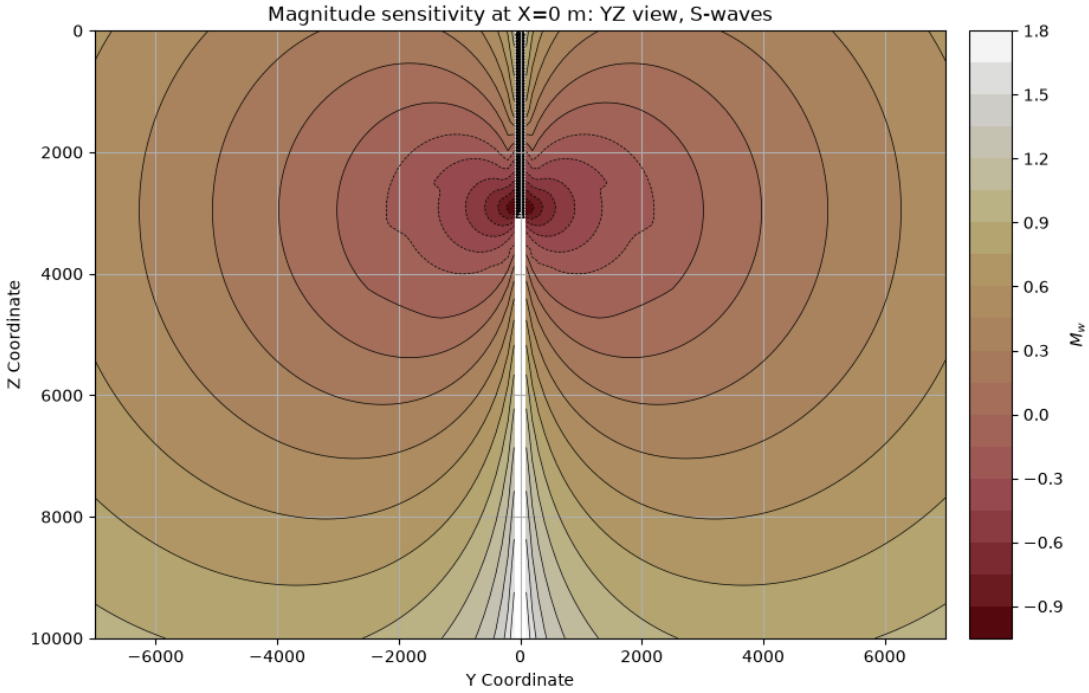
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= Y_slice_mag_sens_s, xi=gx,␣
```

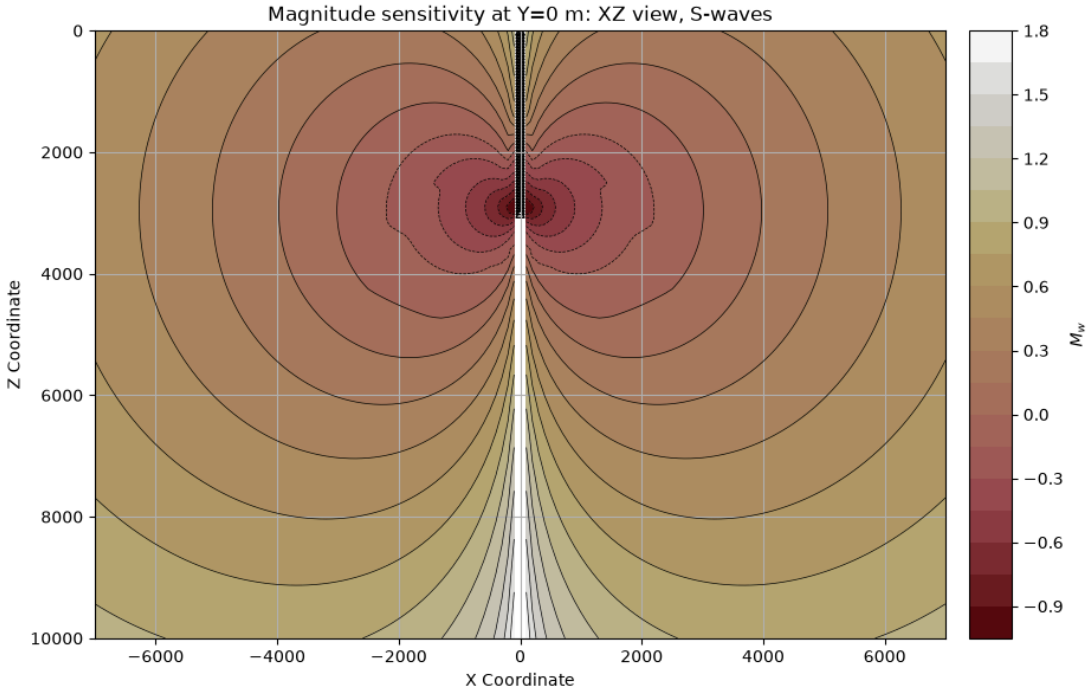
(continues on next page)

(continued from previous page)

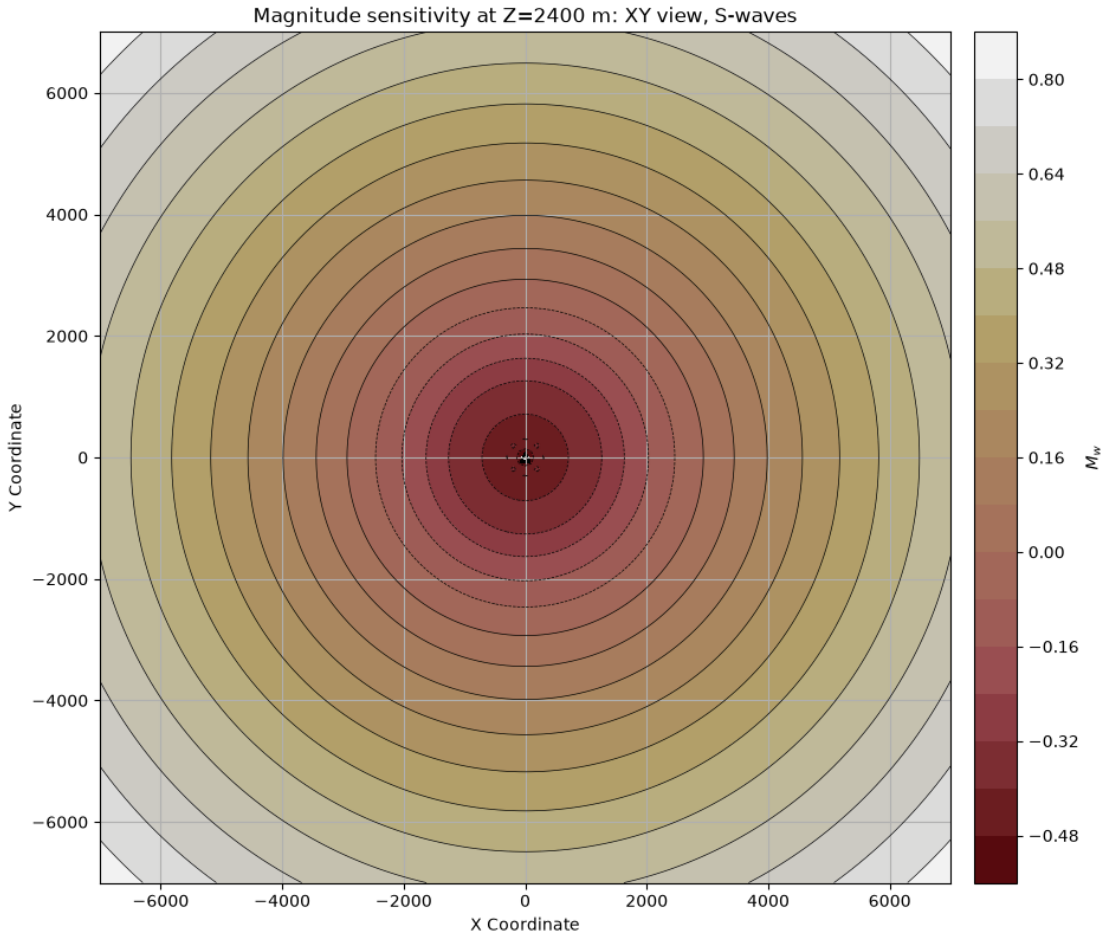
```
↪zi=gz,
                                                plot_title=f"Magnitude sensitivity at Y=
↪{slice_Y} m: XZ view, S-waves",
                                                clb_title="$M_w$",
                                                geometry_df=combined_geometry_df,
                                                plot_names = False,
                                                station_marker_size = station_marker_
↪size,
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)

_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= Z_slice_mag_sens_s, xi=gx,␣
↪yi=gy,
                                                plot_title=f"Magnitude sensitivity at Z=
↪{slice_Z} m: XY view, S-waves",
                                                clb_title="$M_w$",
                                                geometry_df=combined_geometry_df,
                                                plot_names = False,
                                                station_marker_size = station_marker_
↪size,
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)
```


Magnitude sensitivity at X=0 m: YZ view, S-waves
Z Coordinate
Y Coordinate
0
2000
4000
6000
8000
10000
−6000
−4000
−2000
0
2000
4000
6000
1.8
1.5
1.2
0.9
0.6
0.3
0.0
−0.3
−0.6
−0.9
M_w


Magnitude sensitivity at Y=0 m: XZ view, S-waves
Z Coordinate
X Coordinate
0
2000
4000
6000
8000
10000
−6000
−4000
−2000
0
2000
4000
6000
1.8
1.5
1.2
0.9
0.6
0.3
0.0
−0.3
−0.6
−0.9
M_w

•



-

Plot magnitude sensitivity for P and S waves

```
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= X_slice_mag_sens_ps, yi=gy,␣
↪zi=gz,
                                              plot_title=f"Magnitude sensitivity at X=
↪{slice_X} m: YZ view, P- and S-waves",
                                              clb_title="$M_w$",
                                              geometry_df=combined_geometry_df,
                                              plot_names = False,
                                              station_marker_size = station_marker_
↪size,
                                              clb_range=clb_range,
                                              plt_levels=plt_levels)

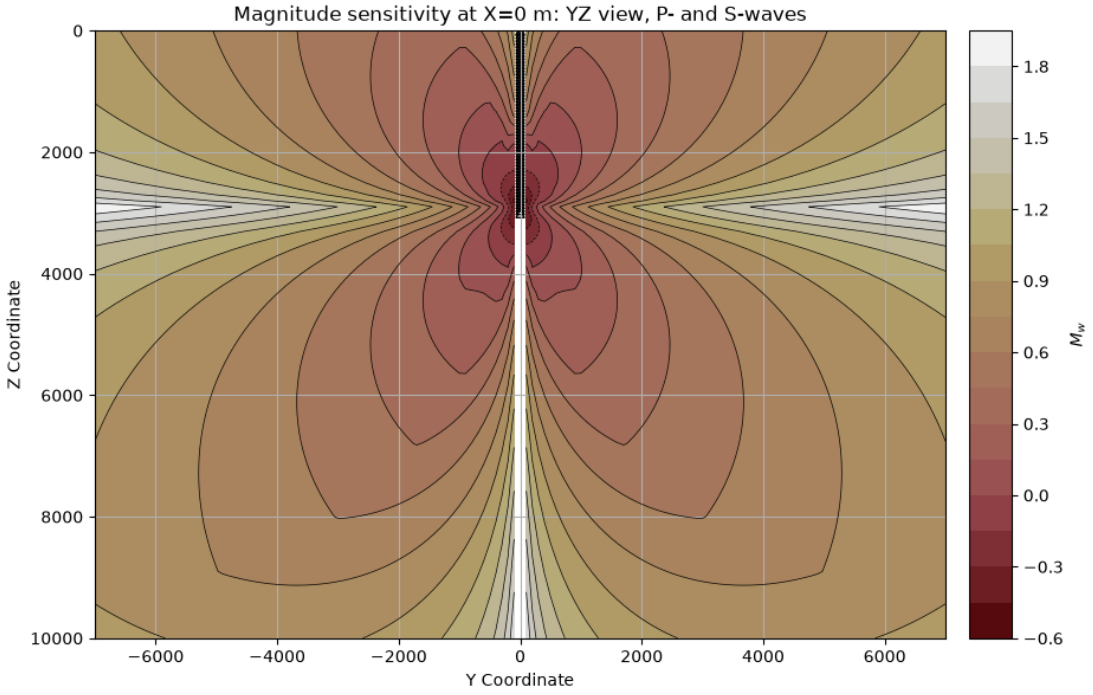
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= Y_slice_mag_sens_ps, xi=gx,␣
```

(continues on next page)

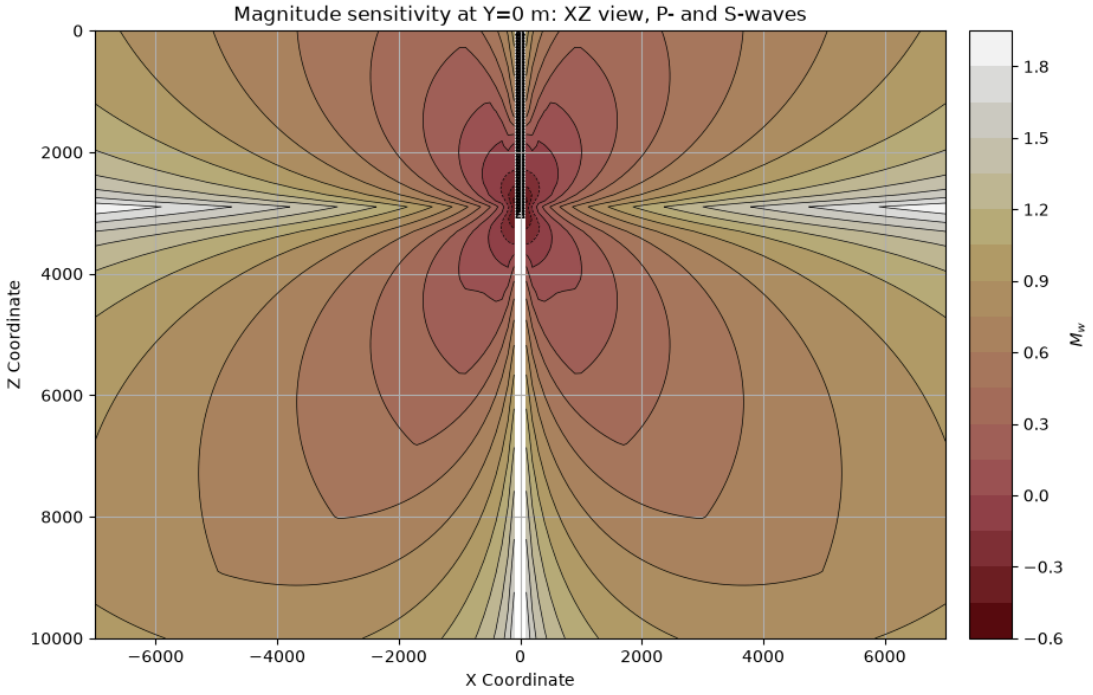
(continued from previous page)

```
↪zi=gz,
                                                plot_title=f"Magnitude sensitivity at Y=
↪{slice_Y} m: XZ view, P- and S-waves",
                                                clb_title="$M_w$",
                                                geometry_df=combined_geometry_df,
                                                plot_names = False,
                                                station_marker_size = station_marker_
↪size,
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)

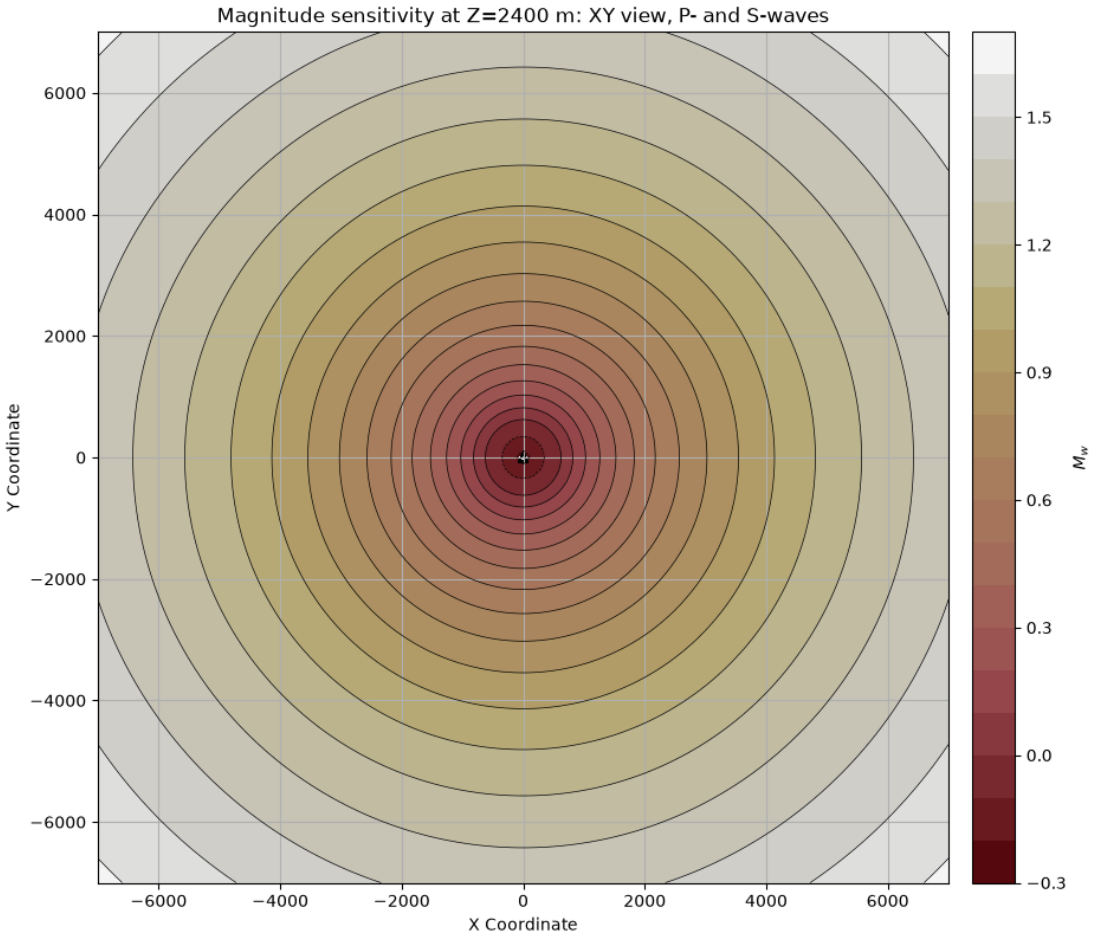
_ = pndvis.sensitivity.plot_sensitivity_slice(sens_data= Z_slice_mag_sens_ps, xi=gx,␣
↪yi=gy,
                                                plot_title=f"Magnitude sensitivity at Z=
↪{slice_Z} m: XY view, P- and S-waves",
                                                clb_title="$M_w$",
                                                geometry_df=combined_geometry_df,
                                                plot_names = False,
                                                station_marker_size = station_marker_
↪size,
                                                clb_range=clb_range,
                                                plt_levels=plt_levels)
```


Magnitude sensitivity at X=0 m: YZ view, P- and S-waves
Z Coordinate
Y Coordinate
0
2000
4000
6000
8000
10000
−6000
−4000
−2000
0
2000
4000
6000
1.8
1.5
1.2
0.9
0.6
0.3
0.0
−0.3
−0.6
M_w

•


Magnitude sensitivity at Y=0 m: XZ view, P- and S-waves
Z Coordinate
X Coordinate
0
2000
4000
6000
8000
10000
−6000
−4000
−2000
0
2000
4000
6000
1.8
1.5
1.2
0.9
0.6
0.3
0.0
−0.3
−0.6
M_w

•



-

Total running time of the script: (0 minutes 8.346 seconds)

CHAPTER

FOUR

API REFERENCE

The Application Programming Interface (API) of PyNetDesign is composed of the following modules:

- *Modelling*: algorithms for modelling related to microseismic network design
  - *Core*: high-level drivers for magnitude sensitivity
  - *Homogeneous*: algorithms for a homogeneous velocity model
  - *Geometry generators*: synthetic geometry generators
  - *Input / output*: reading and writing geometry and velocity files
  - *Classes*: parameter containers
  - *Modelling utilities*: wave modes, radiation patterns, amplitudes and grid helpers
- *Visualization*: plotting of results and input
  - *Geometry*: receiver geometry
  - *Sensitivity*: magnitude sensitivity results
  - *Visualization utilities*: general utility functions for visualization

4.1 Modelling

4.1.1 Core

`get_mag_sensitivity`(grid_coords, ...[, ...])	Computes detectable magnitude sensitivity for input geometry, velocity and parameters.
`get_mag_sensitivity_local`(grid_coords, ...)	Computes magnitude sensitivity using mag_sensitivity_grid for input local geometry, velocity and parameters.

pynetdesign.modelling.core.get_mag_sensitivity

pynetdesign.modelling.core.**get_mag_sensitivity**(*grid_coords*, *geometry_df*, *velocity_df*, *params*, *wave_mode='PS'*, *strict_nan_check=False*)

Computes detectable magnitude sensitivity for input geometry, velocity and parameters. It also deals with combined geometries.

Parameters

grid_coords
[numpy.ndarray] Array of grid coordinates [[xs1, ys1, zs1], [xs2, ys2, zs2], …]

geometry_df
[pandas.DataFrame] Geometry DataFrame

velocity_df
[pandas.DataFrame] Velocity model DataFrame

params
[*pynetdesign.modelling.classes.DetectionParameters*] Detection parameters

wave_mode: :obj:`str`, optional, default: 'PS'
Wave mode to use, can be 'P', 'SV', 'SH', 'S' or 'PS'

strict_nan_check: :obj:`bool`, optional, default: False
flag to check if NaN values appear in the result

Returns

Mw_min_grid
[numpy.ndarray] Array of computed detectable minimum moment magnitudes on the grid

pynetdesign.modelling.core.get_mag_sensitivity_local

pynetdesign.modelling.core.**get_mag_sensitivity_local**(*grid_coords*, *geometry_df*, *velocity_df*, *params*, *wave_mode='PS'*, *strict_nan_check=False*)

Computes magnitude sensitivity using mag_sensitivity_grid for input local geometry, velocity and parameters. Checks if the input geometry is DAS, i.e. has a gauge length and uses directionality accordingly.

Parameters

grid_coords
[numpy.ndarray] Array of grid coordinates [[xs1, ys1, zs1], [xs2, ys2, zs2], …]

geometry_df
[pandas.DataFrame] Geometry DataFrame

velocity_df
[pandas.DataFrame] Velocity model DataFrame

params
[*pynetdesign.modelling.classes.DetectionParameters*] Detection parameters

wave_mode: :obj:`str`, optional, default: 'PS'
Wave mode to use, can be 'P', 'SV', 'SH', 'S' or 'PS'

strict_nan_check: :obj:`bool`, optional, default: False
flag to check if NaN values appear in the result

Returns

Mw_min_stations
[numpy.ndarray] Array of computed minimum moment magnitudes for individual stations

Raises

ValueError
if input velocity model is not homogeneous, i.e. contains more than one layer

4.1.2 Homogeneous

mag_sensitivity_grid(grid_coords, velocity_df)	Computes magnitude sensitivity - the minimum moment magnitude (Mw) of a microseismic event which will theoretically produce particle velocity amplitudes (of P and/or S waves) on stations necessary for event detection assuming 3D elastic homogeneous medium with attenuation.
calculate_M0(density, v, Q, rad_pattern, r, amps)	Calculates the seismic moment M_0 in a homogeneous medium given the density ρ, wave velocity v, distance from the source r and the displacement amplitude spectrum $\|U(f)\|$ using:
calculate_fpeak(v, Q, r[, fcorner])	Calculates the peak frequency f_{peak} for particle velocity in a homogeneous medium given the wave velocity v, the attenuation factor Q, and the distance from a source r using:
validate_parameters(density, v_p, v_s, Q_p, ...)	Validates the input parameters.

pynetdesign.modelling.homo.mag_sensitivity_grid

pynetdesign.modelling.homo.**mag_sensitivity_grid**(*grid_coords*, *velocity_df*, *geometry_df=None*, *station_coords=None*, *min_amps_p=None*, *min_amps_s=None*, *min_stations_p=None*, *min_stations_s=None*, *rad_pattern_p=None*, *rad_pattern_s=None*, *rad_patterns=None*, *f_p=None*, *f_s=None*, *f_p_corner=None*, *f_s_corner=None*, *wave_mode='PS'*, *fs_mode='auto'*, *fs_level=None*, *fs_deviation=None*, *use_station_directionality=False*, *return_stations=False*, *strict_nan_check=False*, *amps_type=None*)

Computes magnitude sensitivity - the minimum moment magnitude (Mw) of a microseismic event which will theoretically produce particle velocity amplitudes (of P and/or S waves) on stations necessary for event detection assuming 3D elastic homogeneous medium with attenuation. It takes into account minimum number of stations on which the event should be detected. By request it is possible to output only minimum moment magnitude for each station.

Parameters

grid_coords
[numpy.ndarray] Array of grid coordinates [[xs1, ys1, zs1], [xs2, ys2, zs2], …]

velocity_df
[pandas.DataFrame] Velocity model DataFrame

geometry_df
[pandas.DataFrame, optional] Geometry DataFrame, use if station_coords is None

station_coords
[numpy.ndarray, optional] Array of station coordinates [[xr1, yr1, zr1], [xr2, yr2, zr2], …], use if geometry_df is None

min_amps_p
[numpy.ndarray, optional] Minimal measurable displacement amplitudes of on stations [a1, a2, …] for P waves

min_amps_s
[numpy.ndarray, optional] Minimal measurable displacement amplitudes on stations [a1,

a2, …] for S waves

min_stations_p
[int, optional, default: None (3)] Minimum number of stations on which event must be detected with P waves

min_stations_s
[int, optional, default: None (3)] Minimum number of stations on which event must be detected with S waves

rad_pattern_p
[float, optional] Override of the radiation pattern factor for P waves. If None, the root-mean-square value $\sqrt{4/15} \approx 0.52$ is used.

rad_pattern_s
[float, optional] Override of the radiation pattern factor applied to both S phases. If None, the root-mean-square values $\sqrt{7/30} \approx 0.48$ for SV and $\sqrt{1/6} \approx 0.41$ for SH are used. Pass 0.63 to reproduce results computed with the combined S-wave value.

rad_patterns
[dict, optional] Per-phase overrides of the radiation pattern factor, e.g. `{'SV': 0.63}`. Takes precedence over `rad_pattern_p` and `rad_pattern_s`.

f_p
[float, optional] Frequency of the P wave (Hz)

f_s
[float, optional] Frequency of the S wave (Hz)

f_p_corner
[float, optional] Corner frequency of the P wave (Hz)

f_s_corner
[float, optional] Corner frequency of the S wave (Hz)

wave_mode: :obj:`str`, optional, default: 'PS'
Wave mode to use, can be 'P', 'SV', 'SH', 'S' or 'PS'

fs_mode
[str, optional, default: 'auto'] Free surface correction mode, can be 'auto', 'on' or 'off'. The correction accounts for the amplification of amplitudes recorded at the daily surface due to the free surface boundary condition, and is applied per station so that geometries mixing surface and downhole receivers are handled correctly. See *compute_free_surface_coefficients*.

fs_level
[float, optional, default: None (0.0)] Level of the free surface in metres above the reference level.

fs_deviation
[float, optional, default: None (1.0)] Permitted deviation from the free surface level in metres.

use_station_directionality
[bool, optional, default: False] Force projection of the arriving polarization onto the local tangent defined by the adjacent stations. Projection is applied automatically for DAS geometries and for stations recording a single vertical component, so this is only needed to force tangent projection for a station geometry.

return_stations
[bool, optional, default: False] Return magnitude sensitivity for each station, don't take into account detectability

strict_nan_check: :obj:`bool`, optional, default: False
flag to check if NaN values appear in the result

amps_type
[str, optional, default: None] Type of amplitudes, can be 'displacement', 'velocity', 'acceleration', 'strain rate', 'strain'.

Returns

Mw_min_grid
[numpy.ndarray] Array of computed detectable minimum moment magnitudes on the grid, if return_stations = False

Mw_min_stations
[numpy.ndarray] Array of computed minimum moment magnitudes for individual stations, if return_stations = True Mw_min_stations is of shape (2, npairs) where npairs is a number of of source-grid pairs determined from station_coords and grid_coords. First dimension corresponds to P and S waves.

Raises

ValueError
if both station_coords and geometry_df are not None or None

ValueError
if there is inconsistency in P/S wave parameters

ValueError
if number of minimal measurable amplitudes do not match number of stations

ValueError
if density, wave velocity or attenuation factor is not positive

ValueError
if radiation pattern factor for P or S wave is less or equal to 0 or higher than 1

ValueError
if number of minimal measurable amplitudes does not match number of stations

RuntimeError
if strict_nan_check is True and NaN appears in the result

Notes

This is a refactored vectorized version of the code

pynetdesign.modelling.homo.calculate_M0

pynetdesign.modelling.homo.**calculate_M0**(*density*, *v*, *Q*, *rad_pattern*, *r*, *amps*, *f=None*, *fcorner=None*, *amps_type='displacement'*)

Calculates the seismic moment M_0 in a homogeneous medium given the density ρ, wave velocity v, distance from the source r and the displacement amplitude spectrum $|U(f)|$ using:

$$M_0 = \frac{4\pi\rho v^3 r}{|R|}\ |U(f)|\ e^{\pi f t^*},$$

where R is the provided radiation pattern factor and the exponential term corrects for the intrinsic attenuation accumulated along the ray path.

The amplitudes passed in `amps` are the displacement amplitude spectrum $|U(f)|$. Amplitudes of another type are converted to it with the coefficients of *get_scaling_displacement*, which follow from $|V(f)| = 2\pi f\,|U(f)|$.

The term t^* is the integral along the ray path of the inverse value of attenuation factor Q multiplied by the inverse of the wave velocity v:

$$t^* = \int_R \frac{dr}{v(r)Q(r)},$$

which in a homogenous medium reads as

$$t^* = \frac{r}{vQ}.$$

Frequency f is the representative frequency of the wave. If it is not provided, it defaults to the peak frequency computed from the model, and limited by the corner frequency.

Parameters

density
[`float`] Density (kg/m^3)

v
[`float`] Wave velocity (m/s)

Q
[`float`] Attenuation factor

rad_pattern
[`float`] Radiation pattern factor

r
[`float` or `numpy.ndarray`] Distance(s) from the source (m)

amps
[`float` or `numpy.ndarray`] Displacement amplitude spectrum $|U(f)|$, single value or array of amplitudes, one for each ray. If `r` is a single float, `amps` must be a single float If `r` is an array, `amps` must be an array of the same shape.

f
[`float`, optional, default: `None`] Frequency. If `None`, peak frequency is computed from the model

fcorner
[`float`, optional, default: `None`] Corner frequency, limit for peak frequency.

amps_type
[`str`, optional, default: `displacement`] Type of amplitudes in `amps`, can be 'displacement', 'velocity', 'acceleration', 'strain rate', 'strain'. Used for scaling of amplitudes in case `f` is `None`.

Returns

M0
[`float` or array_like] Seismic moment (N m)

pynetdesign.modelling.homo.calculate_fpeak

`pynetdesign.modelling.homo.`**`calculate_fpeak`**`(`*v*, *Q*, *r*, *fcorner=None*`)`

Calculates the peak frequency f_{peak} for particle velocity in a homogeneous medium given the wave velocity v, the attenuation factor Q, and the distance from a source r using:

$$f_{peak} = \frac{vQ}{\pi r}.$$

The peak frequency estimate is only valid for frequencies below the corner frequency [Eisner *et al.*, 2013].

Parameters

v
[float] wave velocity

Q
[float] attenuation factor

r
[float] distance from the source

fcorner
[float, optional, default: None] Corner frequency that limits the peak frequency. If None, defaults to 100 Hz.

Returns

fpeak
[float] Peak frequency for particle velocity

References

Eisner, L., Gei, D., Hallo, M., Opršal, I., & Ali, M. Y. (2013). The peak frequency of direct waves for microseismic events. Geophysics, 78(6), A45–A49. https://doi.org/10.1190/geo2013-0197.1

pynetdesign.modelling.homo.validate_parameters

pynetdesign.modelling.homo.**validate_parameters**(*density*, *v_p*, *v_s*, *Q_p*, *Q_s*, *min_amps_p*, *min_amps_s*, *wave_mode*, *station_coords*)

Validates the input parameters.

Parameters

density
[float] Density of the medium

v_p
[float] P wave velocity

v_s
[float] S wave velocity

Q_p
[float] P wave attenuation factor

Q_s
[float] S wave attenuation factor

min_amps_p
[numpy.ndarray] Minimal measurable displacement amplitudes of on stations [a1, a2, …] for P waves

min_amps_s
[numpy.ndarray] Minimal measurable displacement amplitudes on stations [a1, a2, …] for S waves

wave_mode: :obj:`str`
Wave mode to use, can be 'P', 'SV', 'SH', 'S' or 'PS'

station_coords
[numpy.ndarray] Array of station coordinates [[xr1, yr1, zr1], [xr2, yr2, zr2], …]

Raises

ValueError
if density, wave velocity or attenuation factor is not positive

ValueError
if number of minimal measurable amplitudes does not match number of stations

ValueError
if there is inconsistency in P/S wave parameters

4.1.3 Geometry generators

generate_geometry(mode[, center_x, ...])	Generate a DAS geometry file in PyNetDesign tab-delimited format.
generate_borehole_geometry(center_x, ...)	Generate borehole DAS channel coordinates.
generate_surface_geometry(center_x, ...)	Generate surface DAS channel coordinates.
generate_darkfiber_geometry(center_x, ...[, ...])	Generate a continuous dark-fiber cable path.

pynetdesign.modelling.geometry.generate_geometry

pynetdesign.modelling.geometry.**generate_geometry**(*mode*, *center_x=0.0*, *center_y=0.0*, *spacing=10.0*, *noise_level=5e-05*, *gauge_length=10.0*, *cable_length=5000.0*, *start_depth=0.0*, *azimuth=0.0*, *dip=90.0*, *surface_noise=None*, *noise_decay_depth=150.0*, *noise_low_depth=200.0*, *depth=0.0*, *surface_shape='zigzag'*, *zigzag_amplitude=500.0*, *zigzag_segment_length=1000.0*, *l_arm_x=3000.0*, *l_arm_y=3000.0*, *rect_width=3000.0*, *rect_height=3000.0*, *double_line_separation=500.0*, *start=None*, *turns=None*, *behavior=None*, *seed=42*)

Generate a DAS geometry file in PyNetDesign tab-delimited format.

pynetdesign.modelling.geometry.generate_borehole_geometry

pynetdesign.modelling.geometry.**generate_borehole_geometry**(*center_x*, *center_y*, *spacing*, *start_depth*, *cable_length*, *azimuth*, *dip*)

Generate borehole DAS channel coordinates.

pynetdesign.modelling.geometry.generate_surface_geometry

pynetdesign.modelling.geometry.**generate_surface_geometry**(*center_x*, *center_y*, *spacing*, *depth*, *cable_length*, *shape*, *zigzag_amplitude*, *zigzag_segment_length*, *l_arm_x*, *l_arm_y*, *rect_width*, *rect_height*, *double_line_separation*)

Generate surface DAS channel coordinates.

pynetdesign.modelling.geometry.generate_darkfiber_geometry

pynetdesign.modelling.geometry.**generate_darkfiber_geometry**(*center_x*, *center_y*, *spacing*, *depth*, *cable_length*, *start=None*, *turns=None*, *behavior=None*, *seed=42*)

Generate a continuous dark-fiber cable path.

4.1.4 Input / output

read_geometry(file_path)	Read geometry data from a whitespace-delimited CSV (ASCII) file and return a DataFrame.
save_geometry(geometry, file_path)	Save geometry to disk in a format compatible with *read_geometry*.
read_velocity(file_path)	Read velocity model data from a whitespace-delimited CSV (ASCII) file and return a DataFrame.
combine_geometry(*dataframes[, names])	Combine an arbitrary number of pandas DataFrames with geometries into one while retaining the ability to distinguish and recover the original DataFrames.
is_combined_geometry(df)	Check if the given DataFrame is a combined geometry DataFrame created by combine_geometry.
decimate_geometry(geometry_df, step)	Decimate a non-combined geometry by keeping every `step`-th station.
global2local(latY, lonX[, refLat, refLon])	Converts global latitude and longitude coordinates to Cartesian coordinates under a simple spherical Earth approximation.

pynetdesign.modelling.io.read_geometry

pynetdesign.modelling.io.**read_geometry**(*file_path*)

Read geometry data from a whitespace-delimited CSV (ASCII) file and return a DataFrame.

This function reads a CSV (ASCII) file containing geometry information, ensuring that coordinate columns (Latitude/Northing, Longitude/Easting, Elevation) and NoiseLevel column are present. The 'Name', 'Surface' and 'Components' columns are optional. The resulting DataFrame has columns renamed and units converted as necessary. If header has 'GaugeLength=', the appropriate gauge length is read and stored in the attributes.

Parameters

file_path
[`str`] The path to the input CSV (ASCII) file containing the geometry data.

Returns

pd.DataFrame
A DataFrame containing the renamed columns:

- 'Name' for Name (if present in the input file)
- 'Y' for Latitude or Northing (converted to meters)
- 'X' for Longitude or Easting (converted to meters)
- 'Z' for Depth (negative of Elevation, converted to meters)
- 'NoiseLevel' for Noise Level
- 'Surface' for the free surface indicator (if present in the input file)
- 'Components' for the recorded station components (if present in the input file)

The DataFrame also includes metadata:

- `df.attrs['file_path']`: Path of the file that was read
- `df.attrs['original_coord_units']`: Original units of coordinate columns
- `df.attrs['noise_type']`: Type of noise measurement

- `df.attrs['noise_unit']`: Original unit of noise measurement
- `df.attrs['gauge_length']`: Gauge length (if specified in the header)
- `df.attrs['_is_combined']`: Combined status (False)

Raises

ValueError
If required columns are missing, unknown columns are present, inconsistent units are found for coordinate columns, or if an unrecognized unit is encountered for the NoiseLevel column or GaugeLength.

Notes

- The function handles whitespace and tabs as delimiters.
- It recognizes and converts units for coordinate columns (m, km, ft).
- For the NoiseLevel column, it recognizes units: m (displacement), m/s (velocity), m/s^2 (acceleration), and 1/s (strain).
- The 'Z' column is converted to represent depth (negative of elevation).
- The 'Name' column, if present, is interpreted as strings.
- The 'Surface' column, if present, is a weight in [0, 1] marking receivers at the free surface, where a fractional value blends between no amplification and full free surface amplification.
- The 'Components' column, if present, must be '3C' or 'Z' and is only allowed for station geometries, not for DAS geometries.
- Only known columns ('Name', 'Latitude', 'Northing', 'Longitude', 'Easting', 'Elevation', 'NoiseLevel', 'Surface', 'Components') or combinations like 'Latitude/Northing' or 'Latitude(WGS84)/Northing(m)' are read from the file.
- For Latitude and Longitude conversion to m is not implemented, inputs are treated as meters
- The function will raise an error if any unknown columns are present in the input file.

Examples

```
>>> df = read_geometry("data/geometry.csv")
>>> print(df.head())
>>> print(df.attrs)  # To view metadata
```

pynetdesign.modelling.io.save_geometry

`pynetdesign.modelling.io.`**`save_geometry`**(*geometry*, *file_path*)

Save geometry to disk in a format compatible with *`read_geometry`*.

Accepts either the geometry text lines returned by the generators in `pynetdesign.modelling.geometry` or a geometry `pandas.DataFrame` previously read by *`read_geometry`*.

Parameters

geometry
[`list` of `str` or `pandas.DataFrame`] Geometry content to write.

file_path
[`str` or path-like] Output file path.

Returns

file_path
[str or path-like] The output path, returned for convenience.

Raises

ValueError
If the geometry content is empty or invalid for writing.

TypeError
If `geometry` is neither a list of strings nor a geometry DataFrame.

pynetdesign.modelling.io.read_velocity

`pynetdesign.modelling.io.`**`read_velocity`**(*file_path*)

Read velocity model data from a whitespace-delimited CSV (ASCII) file and return a DataFrame. This function reads a CSV (ASCII) file containing velocity information, checks if the columns match the expected format, and renames the columns according to a predefined mapping for easier analysis.

Parameters

file_path
[str] The path to the input CSV (ASCII) file containing the velocity data.

Returns

df
[pandas.DataFrame] A DataFrame containing the renamed columns:

- 'Depth' for Depth;
- 'Vp' for P-wave velocity;
- 'VpVsRatio' for Vp/Vs ratio;
- 'Qp' for P-wave quality factor;
- 'Qs' for S-wave quality factor;
- 'Rho' for density (kg/m^3);
- 'Ep' for anisotropy parameter Ep (Thomsen epsilon parameter for VTI medium, P-SV waves)
- 'Dt' for anisotropy parameter Dt (Thomsen delta parameter for VTI medium, P-SV waves)
- 'Gm' for anisotropy parameter Gm (Thomsen gamma parameter for VTI medium, SH waves)

The DataFrame will also have the following attributes:

- **'model_type'**
  [str] The type of model, either "homo" (homogeneous, single layer) or "layered" (multiple layers).
- **'model_anisotropy'**
  [str] The anisotropy type, either "isotropic" (if all anisotropy parameters are zero) or "vti" (if any are non-zero).

Raises

ValueError
If the input file columns do not match the expected format.

Notes

The function handles whitespace and tabs as delimiters.

Examples

```
>>> df = read_velocity("data/velocity.csv")
>>> print(df.head())
```

pynetdesign.modelling.io.combine_geometry

pynetdesign.modelling.io.**combine_geometry**(**dataframes*, *names=None*)

Combine an arbitrary number of pandas DataFrames with geometries into one while retaining the ability to distinguish and recover the original DataFrames.

This function adds a 'source' column to each DataFrame to identify its origin, concatenates them, and sets a multi-level index using the 'source' column and the original index.

It also stores and recovers attributes of the original dataframes.

Parameters

***dataframes**
[pandas.DataFrame] An arbitrary number of DataFrames to be combined.

names
[typing.Union[List[str], Dict[int, str]], optional] Names to use as identifiers for the DataFrames. Can be a list of strings or a dictionary mapping DataFrame positions to names. If not provided, default names 'df0', 'df1', etc. will be used.

Returns

combined: pandas.DataFrame
A combined DataFrame with a multi-level index and additional methods:

- get_names(): Returns a list of internal DataFrame names;
- recover(name): Recovers a specific DataFrame by name.

Raises

ValueError
If no DataFrames are provided, or if the number of names doesn't match the number of DataFrames.

Examples

```
>>> import pandas as pd
>>> df1 = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
>>> df2 = pd.DataFrame({'A': [5, 6], 'B': [7, 8]})
>>> df3 = pd.DataFrame({'A': [9, 10], 'B': [11, 12]})
>>> combined = combine_df(df1, df2, df3, names=['First', 'Second', 'Third'])
>>> print(combined.get_names())
['First', 'Second', 'Third']
>>> recovered_df2 = combined.recover('Second')
>>> print(recovered_df2)
   A  B
0  5  7
1  6  8
```

pynetdesign.modelling.io.is_combined_geometry

pynetdesign.modelling.io.**is_combined_geometry**(*df*)

Check if the given DataFrame is a combined geometry DataFrame created by combine_geometry.

Parameters

df
[pandas.DataFrame] The DataFrame to check.

Returns

bool
True if the DataFrame is a combined one, False otherwise.

pynetdesign.modelling.io.decimate_geometry

pynetdesign.modelling.io.**decimate_geometry**(*geometry_df*, *step*)

Decimate a non-combined geometry by keeping every `step`-th station.

This is useful for checking how the channel spacing of a dense DAS array affects sensitivity, by thinning an existing geometry rather than regenerating it.

Parameters

geometry_df
[pandas.DataFrame] The input geometry, with one row per station and metadata in `attrs`.

step
[int] Decimation factor, so that `step=5` keeps rows 0, 5, 10 and so on.

Returns

decimated_geometry_df
[pandas.DataFrame] New geometry with every `step`-th station, the index reset and all original `attrs` preserved.

Raises

ValueError
If `step` is not a positive integer, or if the geometry is combined.

pynetdesign.modelling.io.global2local

pynetdesign.modelling.io.**global2local**(*latY*, *lonX*, *refLat=None*, *refLon=None*)

Converts global latitude and longitude coordinates to Cartesian coordinates under a simple spherical Earth approximation.

Parameters

latY
[numpy.ndarray] An array of latitude coordinates in degrees.

lonX
[numpy.ndarray] An array of longitude coordinates in degrees.

refLat
[float, optional] The reference latitude for the local coordinate system. If not provided, the average of the input latitude coordinates will be used.

refLon
[float, optional] The reference longitude for the local coordinate system. If not provided, the average of the input longitude coordinates will be used.

Returns

x_local_coords
[list] A list of local x coordinates in meters.

y_local_coords
[list] A list of local y coordinates in meters.

refLat
[float] The reference latitude used for the local coordinate system.

refLon
[float] The reference longitude used for the local coordinate system.

Examples

```
>>> latY = np.array([37.7749, 34.0522, 40.7128])
>>> lonX = np.array([-122.4194, -118.2437, -74.0060])
>>> x_local_coords, y_local_coords, refLat, refLon = global2local(latY, lonX)
```

4.1.5 Classes

DetectionParameters(min_SNR_p, min_SNR_s, ...)	Container for the parameters controlling event detection.

pynetdesign.modelling.classes.DetectionParameters

class pynetdesign.modelling.classes.**DetectionParameters**(*min_SNR_p*, *min_SNR_s*, *min_stations_p*, *min_stations_s*, *f_p=None*, *f_s=None*, *f_p_corner=None*, *f_s_corner=None*, *fs_mode=None*, *fs_level=None*, *fs_deviation=None*, *free_surface=None*)

Container for the parameters controlling event detection.

Parameters

min_SNR_p
[float] Minimum signal-to-noise ratio required to detect a P wave on one receiver.

min_SNR_s
[float] Minimum signal-to-noise ratio required to detect an S wave on one receiver.

min_stations_p
[int] Minimum number of receivers on which the event must be detected with P waves.

min_stations_s
[int] Minimum number of receivers on which the event must be detected with S waves.

f_p
[float, optional] Representative frequency of the P wave (Hz). If None, the peak frequency is computed from the model.

f_s
[float, optional] Representative frequency of the S wave (Hz). If None, the peak frequency is computed from the model.

f_p_corner
[float, optional, default: None (100 Hz)] Corner frequency of the P wave (Hz), limiting the peak frequency.

f_s_corner
[float, optional, default: None (100 Hz)] Corner frequency of the S wave (Hz), limiting the peak frequency.

fs_mode
[str, optional, default: None ('auto')] Free surface correction mode, one of 'auto', 'on' or 'off'. In 'auto' mode the correction is applied to receivers that are within fs_deviation of the free surface level, or according to the Surface column of the geometry when present.

fs_level
[float, optional, default: None (0.0)] Level of the free surface in metres above the reference level.

fs_deviation
[float, optional, default: None (1.0)] Permitted deviation from the free surface level in metres.

free_surface
[bool, optional] Deprecated override kept for backward compatibility. True maps to fs_mode='on' and False to fs_mode='off'. If None, fs_mode is used.

Notes

The free surface correction is applied per receiver, so a geometry combining surface and downhole receivers is handled correctly. This differs from PyNetDesign 1.0.x, where free_surface=True amplified every receiver irrespective of its depth.

Methods

__init__(min_SNR_p, min_SNR_s, ...[, f_p, ...])	
validate()	

4.1.6 Modelling utilities

Wave modes and radiation patterns

normalize_wave_mode(wave_mode)	Return the canonical form of a wave mode.
wave_mode_phases(wave_mode)	Expand a wave mode into the physical phases that must be evaluated.
wave_mode_has_p(wave_mode)	Return whether wave_mode includes the P branch.
wave_mode_has_s(wave_mode)	Return whether wave_mode includes any S-family branch.
wave_mode_s_phases(wave_mode)	Return the S-family phases included in wave_mode.
phase_radiation_pattern(phase)	Return the root-mean-square radiation-pattern magnitude of a phase.
resolve_radiation_pattern(phase[, ...])	Resolve the radiation-pattern magnitude to use for a phase.

pynetdesign.modelling.utils.normalize_wave_mode

pynetdesign.modelling.utils.**normalize_wave_mode**(*wave_mode*)

Return the canonical form of a wave mode.

`'S'` is a composite mode representing the better recorded of the two isotropic S polarizations, SV and SH.

Parameters

wave_mode
[`str`] Wave mode, one of `'P'`, `'SV'`, `'SH'`, `'S'` or `'PS'`, in any letter case and with surrounding whitespace ignored.

Returns

mode
[`str`] Canonical upper-case wave mode.

Raises

ValueError
If `wave_mode` is not one of the supported modes.

pynetdesign.modelling.utils.wave_mode_phases

pynetdesign.modelling.utils.**wave_mode_phases**(*wave_mode*)

Expand a wave mode into the physical phases that must be evaluated.

Parameters

wave_mode
[`str`] Wave mode, see *normalize_wave_mode*.

Returns

phases
[`tuple` of `str`] Phases to evaluate, drawn from `'P'`, `'SV'` and `'SH'`.

pynetdesign.modelling.utils.wave_mode_has_p

pynetdesign.modelling.utils.**wave_mode_has_p**(*wave_mode*)

Return whether `wave_mode` includes the P branch.

pynetdesign.modelling.utils.wave_mode_has_s

pynetdesign.modelling.utils.**wave_mode_has_s**(*wave_mode*)

Return whether `wave_mode` includes any S-family branch.

pynetdesign.modelling.utils.wave_mode_s_phases

pynetdesign.modelling.utils.**wave_mode_s_phases**(*wave_mode*)

Return the S-family phases included in `wave_mode`.

pynetdesign.modelling.utils.phase_radiation_pattern

pynetdesign.modelling.utils.**phase_radiation_pattern**(*phase*)

Return the root-mean-square radiation-pattern magnitude of a phase.

The values are averages over the focal sphere of a double-couple source [Boore and Boatwright, 1984], as used for microseismicity with random source orientations [Hallo and Eisner, 2013]:

$$R_P = \sqrt{4/15} \approx 0.52, \quad R_{SV} = \sqrt{7/30} \approx 0.48, \quad R_{SH} = \sqrt{1/6} \approx 0.41.$$

Parameters

phase
[str] Phase, one of `'P'`, `'SV'` or `'SH'`.

Returns

rad_pattern
[float] Radiation-pattern magnitude of the phase.

Raises

ValueError
If `phase` is a composite mode rather than a single phase.

Notes

The value $\sqrt{2/5} \approx 0.63$ often quoted for S-waves is the magnitude of the *combined* S-wave amplitude, since $\langle R_{SV}^2 \rangle + \langle R_{SH}^2 \rangle = 2/5$, and is therefore not applicable to SV or SH individually.

pynetdesign.modelling.utils.resolve_radiation_pattern

pynetdesign.modelling.utils.**resolve_radiation_pattern**(*phase*, *rad_pattern_p=None*, *rad_pattern_s=None*, *rad_patterns=None*)

Resolve the radiation-pattern magnitude to use for a phase.

Overrides take precedence over the root-mean-square defaults of *phase_radiation_pattern*, which makes it possible to reproduce results published with other conventions, for instance the combined S-wave value $R_S = 0.63$.

Parameters

phase
[str] Phase, one of `'P'`, `'SV'` or `'SH'`.

rad_pattern_p
[float, optional] Override for the P phase. If `None`, the default is used.

rad_pattern_s
[float, optional] Override applied to both S phases. If `None`, the defaults are used.

rad_patterns
[dict, optional] Per-phase overrides, e.g. `{'SV': 0.63}`. Takes precedence over `rad_pattern_p` and `rad_pattern_s`.

Returns

rad_pattern
[float] Radiation-pattern magnitude to use.

Raises

ValueError
If a resolved value is not in the interval (0, 1].

Magnitudes and amplitudes

calculate_Mw(M0)	Calculates the moment magnitude M_w according to *Kanamori (1977)* using formula
calculate_seismic_moment(Mw)	Calculates the seismic moment M_0 from moment magnitude M_w according to *Kanamori (1977)*
get_scaling_displacement(amps_type, f)	Get scaling coefficient(s) to convert to the amplitudes to displacement provided the type of original amplitudes and the frequency (or array of frequencies).
retrieve_min_amps(geometry_df, SNR_p, SNR_s)	Retrieve minimum amplitudes for P and S waves from geometry data, scaled based on noise type.

pynetdesign.modelling.utils.calculate_Mw

pynetdesign.modelling.utils.**calculate_Mw**(*M0*)

Calculates the moment magnitude M_w according to *Kanamori (1977)* using formula

$$M_w = \frac{2}{3}\left(\log_{10}(M_0) - 9.1\right)$$

where M_0 is the seismic moment.

Parameters

M0
[array_like] seismic moment

Returns

Mw
[array_like] moment magnitude

Notes

Replaces all non-positive values of M0 by nan

References

Kanamori, H. (1977). The energy release in great earthquakes. Journal of Geophysical Research, 82(20), 2981–2987. https://doi.org/10.1029/jb082i020p02981

pynetdesign.modelling.utils.calculate_seismic_moment

pynetdesign.modelling.utils.**calculate_seismic_moment**(*Mw*)

Calculates the seismic moment M_0 from moment magnitude M_w according to *Kanamori (1977)*

$$M_0 = 10^{\frac{3}{2}M_w + 9.1}$$

where M_w is the moment magnitude.

Parameters

Mw
[array_like] moment magnitude

Returns

M0
[array_like] seismic moment

References

Kanamori, H. (1977). The energy release in great earthquakes. Journal of Geophysical Research, 82(20), 2981–2987. https://doi.org/10.1029/jb082i020p02981

pynetdesign.modelling.utils.get_scaling_displacement

pynetdesign.modelling.utils.**get_scaling_displacement**(*amps_type*, *f*)

Get scaling coefficient(s) to convert to the amplitudes to displacement provided the type of original amplitudes and the frequency (or array of frequencies). If f is an array and is used for computation of scaling, output is also an array of the same size.

Parameters

amps_type
[str] Type of amplitudes, can be 'displacement', 'velocity', 'acceleration', 'strain rate', 'strain'.

f
[float or np.ndarray] Frequency or array of frequencies.

Returns

scaling
[float or np.ndarray] Resulting scaling coefficient.

Raises

ValueError
If amps_type is not one of the supported types.

pynetdesign.modelling.utils.retrieve_min_amps

pynetdesign.modelling.utils.**retrieve_min_amps**(*geometry_df*, *SNR_p*, *SNR_s*, *f_p=None*, *f_s=None*, *sample_interval=None*)

Retrieve minimum amplitudes for P and S waves from geometry data, scaled based on noise type.

This function takes geometry data frame with noise levels, then scales these noise levels based on the noise type to displacement using the provided frequencies for P and S waves (optional), and applies SNR scaling for P and S waves.

Parameters

geometry_df
[pandas.DataFrame] The path to the input CSV (ASCII) file containing the geometry data.

SNR_p
[float] Signal-to-Noise Ratio for P waves

SNR_s
[float] Signal-to-Noise Ratio for S waves

f_p: :obj:`float`, optional, default: ``None``
Frequency of the P-wave. If None, scaling is postponed to be later done with peak frequency

f_s: :obj:`float`, optional, default: ``None``
Frequency of the S-wave. If None, scaling is postponed to be later done with peak frequency

sample_interval
[float] Interval along the cable at which strain measurements are made

Returns

min_amps_p
[numpy.ndarray] Minimum displacement amplitudes for P waves

min_amps_s
[numpy.ndarray] Minimum displacement amplitudes for S waves

Raises

ValueError
If an unsupported noise type is encountered.

ValueError
If the 'NoiseLevel' column is missing in geometry_df.

ValueError
If the sample_interval is None for strain and strain rate noise types

Detectability and receiver response

mag_detectable(Mw_min_stations[, ...])	Computes detectable minimum magnitudes from magnitude sensitivities for individual stations with respect to number of stations necessary for event detection using P and/or S waves.
check_min_stations(min_stations_p, ...[, ...])	Check that the minimum numbers of stations required for detection are valid.
compute_free_surface_coefficients(...[, ...])	Compute free surface amplification coefficients for each station and grid point.
get_phase_station_directionality(...[, ...])	Derive receiver projection coefficients for P, SV and SH polarizations.
geometry_requires_receiver_projection(...)	Return whether a geometry needs receiver-component projection.
get_ray_station_directionality(...[, ...])	Derive directionality coefficients for P and S waves along a cable.

pynetdesign.modelling.utils.mag_detectable

pynetdesign.modelling.utils.**mag_detectable**(*Mw_min_stations*, *min_stations_p=None*, *min_stations_s=None*, *wave_mode='PS'*)

Computes detectable minimum magnitudes from magnitude sensitivities for individual stations with respect to number of stations necessary for event detection using P and/or S waves.

Parameters

Mw_min_stations
[numpy.ndarray] Array of computed minimum magnitudes for individual stations

min_stations_p
[int, optional, default: 3] Minimum number of stations on which event must be detected with P waves

min_stations_s
[int, optional, default: 3] Minimum number of stations on which event must be detected with S waves

wave_mode: :obj:`str`, optional, default: 'PS'
Wave mode to use, can be 'P', 'SV', 'SH', 'S' or 'PS'

Returns

Mw_min_grid
[numpy.ndarray] Array of detectable minimum magnitudes on the grid

Raises

ValueError
if minimum number of stations on which event must be detected with P or S waves is less than 1 or exceeds number of stations

pynetdesign.modelling.utils.check_min_stations

pynetdesign.modelling.utils.**check_min_stations**(*min_stations_p*, *min_stations_s*, *n_stations*, *wave_mode='PS'*)

Check that the minimum numbers of stations required for detection are valid.

Parameters

min_stations_p
[int] Minimum number of stations on which event must be detected with P waves.

min_stations_s
[int] Minimum number of stations on which event must be detected with S waves.

n_stations
[int] Total number of stations in the geometry.

wave_mode
[str, optional, default: 'PS'] Wave mode to use, can be 'P', 'SV', 'SH', 'S' or 'PS'.

Raises

ValueError
If a minimum number of stations is not positive or exceeds the number of stations.

Notes

Each of min_stations_p and min_stations_s is checked only if it is not None and only if the corresponding branch is part of wave_mode.

pynetdesign.modelling.utils.compute_free_surface_coefficients

pynetdesign.modelling.utils.**compute_free_surface_coefficients**(*station_coords*, *grid_coords*, *station_coords_on_surface=None*, *fs_mode='auto'*, *fs_level=0.0*, *fs_deviation=1.0*)

Compute free surface amplification coefficients for each station and grid point.

A receiver recording at the free surface sees an amplitude approximately twice that of the incident wave, which is the exact amplification for normal or near-normal incidence. The coefficient is applied per station, so that a geometry combining surface receivers with receivers at depth is treated correctly.

Parameters

station_coords
[numpy.ndarray] Array of station coordinates [[xr1, yr1, zr1], [xr2, yr2, zr2], …]

grid_coords
[numpy.ndarray] Array of grid coordinates [[xs1, ys1, zs1], [xs2, ys2, zs2], …]

station_coords_on_surface
[numpy.ndarray, optional] Per-station indicator of whether the station is on the free surface, as a 1D array of the same length as the number of stations, with values in [0, 1]. Values between 0 and 1 act as a weighting factor between no amplification and full amplification. If `None`, the indicator is derived from the station depths.

fs_mode
[str, optional, default: `'auto'`] Free surface correction mode, can be `'auto'`, `'on'` or `'off'`. If `'on'`, the correction is applied to stations close to the free surface irrespective of `station_coords_on_surface`. If `'off'`, the correction is never applied. If `'auto'`, `station_coords_on_surface` is used when available, otherwise stations are considered to be on the free surface when their Z coordinate deviates from the free surface level by no more than `fs_deviation`.

fs_level
[float, optional, default: 0.0] Level of the free surface in metres above the reference level.

fs_deviation
[float, optional, default: 1.0] Permitted deviation from the free surface level in metres, must be non-negative.

Returns

fs_coef_p
[numpy.ndarray] Free surface coefficients for P waves, of shape (n_grid_points, n_stations)

fs_coef_s
[numpy.ndarray] Free surface coefficients for S waves, of shape (n_grid_points, n_stations)

Raises

ValueError
If `fs_level` is not a scalar, `fs_deviation` is negative, `station_coords_on_surface` has the wrong shape or values outside [0, 1], or `fs_mode` is not one of the supported modes.

Notes

The free surface level is negated internally to match the convention that Z increases downwards.

pynetdesign.modelling.utils.get_phase_station_directionality

pynetdesign.modelling.utils.**get_phase_station_directionality**(*station_coords*, *grid_coords*, *phases=('P', 'SV', 'SH')*, *geometry_df=None*, *force_tangent=False*, *strict_nan_check=False*)

Derive receiver projection coefficients for P, SV and SH polarizations.

The recorded amplitude of a phase is the projection of its arriving polarization onto the direction the receiver is sensitive to. For a phase ϕ the correction applied to the minimum detectable amplitude is

$$C_\phi = \frac{1}{|\mathbf{d} \cdot \mathbf{e}_\phi|},$$

where $\mathbf{e}_\phi$ is the arriving polarization and $\mathbf{d}$ is the local cable tangent for DAS, or the vertical axis for a single-component station. Three-component stations record the full vector and therefore use a coefficient of one.

Parameters

station_coords
[numpy.ndarray] Array of station coordinates [[xr1, yr1, zr1], [xr2, yr2, zr2], …]

grid_coords
[numpy.ndarray] Array of grid coordinates [[xs1, ys1, zs1], [xs2, ys2, zs2], …]

phases
[sequence of str, optional, default: ("P", "SV", "SH")] Phases to compute coefficients for. Must be drawn from 'P', 'SV' and 'SH'.

geometry_df
[pandas.DataFrame, optional] Geometry DataFrame, used to detect a DAS geometry from its gauge length and to read a Components column for station geometries. If None, projection onto the local tangent is used.

force_tangent
[bool, optional, default: False] Force projection onto the local tangent even when geometry_df describes a station geometry.

strict_nan_check
[bool, optional, default: False] Flag to check whether NaN values appear in the result.

Returns

coefficients
[dict] Mapping from phase name to an array of coefficients of shape (n_grid_points, n_stations).

Raises

ValueError
If a requested phase is not P, SV or SH, if the geometry and station coordinates disagree in length, or if a DAS geometry also carries a Components column.

RuntimeError
If strict_nan_check is True and NaN appears in the result.

Notes

A vanishing projection means the phase cannot be recorded at all, for example SH on a vertical cable, and yields NaN so that the phase is excluded rather than reported as infinitely detectable.

Straight rays between each station and each grid point are assumed, which is exact in a homogeneous medium.

pynetdesign.modelling.utils.geometry_requires_receiver_projection

pynetdesign.modelling.utils.**geometry_requires_receiver_projection**(*geometry_df*)

Return whether a geometry needs receiver-component projection.

A DAS geometry, identified by the presence of a gauge length, always requires projection onto the local cable tangent. A station geometry requires projection only if at least one station records a single vertical component.

Parameters

geometry_df
[pandas.DataFrame or None] Geometry DataFrame.

Returns

required
[bool] Whether projection is required.

pynetdesign.modelling.utils.get_ray_station_directionality

pynetdesign.modelling.utils.**get_ray_station_directionality**(*station_coords*, *grid_coords*, *strict_nan_check=False*)

Derive directionality coefficients for P and S waves along a cable.

Deprecated since version Use: *get_phase_station_directionality*, which resolves SV and SH separately and projects the actual polarization vectors.

Parameters

station_coords
[numpy.ndarray] Array of station coordinates [[xr1, yr1, zr1], [xr2, yr2, zr2], …]

grid_coords
[numpy.ndarray] Array of grid coordinates [[xs1, ys1, zs1], [xs2, ys2, zs2], …]

strict_nan_check
[bool, optional, default: False] Flag to check whether NaN values appear in the result.

Returns

dir_coef_p
[numpy.ndarray] Directionality coefficients for P waves, of shape (n_grid_points, n_stations)

dir_coef_s
[numpy.ndarray] Directionality coefficients for SV waves, of shape (n_grid_points, n_stations)

Notes

Returns the P and SV coefficients of *get_phase_station_directionality* with tangent projection forced. For a vertical cable these are the familiar $1/|\cos\theta|$ and $1/|\sin\theta|$; for an inclined or curved cable they are the correct polarization projections rather than that approximation.

Grids and helpers

generate_grid(x, y, z)	From the grid vectors x,y,z generates an array of grid point coordinates of size (nx*ny*nz,3).
reshape_grid(grid_array, nx, ny, nz)	Reshape a flat grid array into a multidimensional array based on the provided grid vectors.
reshape_data(data, x, y)	Reshape data to 2D array using axes vectors.
get_slice_coordinates(grid_points, ...)	From the array of grid point coordinates of size (nx*ny*nz,3) retrieve coordinates of the slice plane at a given slice coordinate in a given slice dimension.
are_points_inside(grid_coords1, grid_coords2)	Check if all points in grid_coords1 are inside grid_coords2.
find_indices_of_points(grid_coords1, ...)	Find indices of points in grid_coords1 that are inside grid_coords2.
nth_minimum(arr, n)	Find the n-th minimum value in each row of a given 2D numpy array.
check_zero_values(a[, ErrorMessage])	Check if the input value is close to zero and raise an error if true.
contains_only_numeric(seq)	Check if a sequence (tuple, list, or numpy array) contains only numeric values.

continues on next page

Table 10 – continued from previous page

compute_gcd_of_floats(rx, ry, rz)	Computes the greatest common divisor (GCD) of three positive float values.

pynetdesign.modelling.utils.generate_grid

pynetdesign.modelling.utils.**generate_grid**(*x*, *y*, *z*)

From the grid vectors x,y,z generates an array of grid point coordinates of size (nx*ny*nz,3).

Parameters

x
[numpy.ndarray] x axis grid vector

y
[numpy.ndarray] y axis grid vector

z
[numpy.ndarray] z axis grid vector

Returns

grid_points
[numpy.ndarray] 2D grid coordinates array

pynetdesign.modelling.utils.reshape_grid

pynetdesign.modelling.utils.**reshape_grid**(*grid_array*, *nx*, *ny*, *nz*)

Reshape a flat grid array into a multidimensional array based on the provided grid vectors.

Parameters

grid_array
[numpy.ndarray] The flat grid array to reshape. The first dimension should match the total number of grid points.

nx
[numpy.int] Number of grid points along the x axis.

ny
[numpy.int] Number of grid points along the y axis.

nz
[numpy.int] Number of grid points along the z axis.

Returns

reshaped_array
[numpy.ndarray] The reshaped array with shape (nx, ny, nz, …), where "…" corresponds to the remaining dimensions of the input array.

Raises

ValueError
If the number of grid points does not match the product of nx, ny, and nz.

pynetdesign.modelling.utils.reshape_data

pynetdesign.modelling.utils.**reshape_data**(*data*, *x*, *y*)

Reshape data to 2D array using axes vectors.

Parameters

data
[numpy.ndarray] 1D data array to be reshaped

x
[numpy.ndarray] x axis vector

y
[numpy.ndarray] y axis vector

Returns

reshaped_data
[numpy.ndarray] 2D reshaped data array

Raises

ValueError
when data array is not 1D

pynetdesign.modelling.utils.get_slice_coordinates

pynetdesign.modelling.utils.**get_slice_coordinates**(*grid_points*, *slice_dimension*, *slice_coordinate*)

From the array of grid point coordinates of size (nx*ny*nz,3) retrieve coordinates of the slice plane at a given slice coordinate in a given slice dimension.

Parameters

grid_points
[numpy.ndarray] 2D grid coordinates array

dimension
[int] integer defining the desired slice dimension (0,1 or 2)

slice_coordinate
[numpy.ndarray] coordinate of the slice to extract in the desired dimension

Returns

plane_points
[numpy.ndarray] 2D slice plane coordinates array

indices
[numpy.ndarray] Indices of plane_points in grid_points

Raises

ValueError
when slice_dimension is not 0, 1 or 2

RuntimeError
when slice can't be extracted (slice coordinate does not coincide with the grid)

pynetdesign.modelling.utils.are_points_inside

pynetdesign.modelling.utils.**are_points_inside**(*grid_coords1*, *grid_coords2*)

Check if all points in `grid_coords1` are inside `grid_coords2`.

This function checks whether all points in the array `grid_coords1` are also present in the array `grid_coords2`. Both arrays should have a shape of `(n, 3)`, where each row represents a 3D coordinate.

Parameters

grid_coords1
[numpy.ndarray] A 2D array of shape (n_grid1, 3) representing the points to check.

grid_coords2
[numpy.ndarray] A 2D array of shape (n_grid2, 3) representing the reference points.

Returns

bool
`True` if all points in `grid_coords1` are found in `grid_coords2`, `False` otherwise.

Raises

ValueError
If `grid_coords1` or `grid_coords2` does not have the required shape of `(n, 3)`.

Examples

```
>>> grid_coords1 = np.array([[1, 2, 3], [4, 5, 6]])
>>> grid_coords2 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
>>> are_points_inside(grid_coords1, grid_coords2)
True
```

```
>>> grid_coords1 = np.array([[1, 2, 3], [10, 11, 12]])
>>> grid_coords2 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
>>> are_points_inside(grid_coords1, grid_coords2)
False
```

pynetdesign.modelling.utils.find_indices_of_points

pynetdesign.modelling.utils.**find_indices_of_points**(*grid_coords1*, *grid_coords2*)

Find indices of points in `grid_coords1` that are inside `grid_coords2`.

This function finds the indices of points in `grid_coords1` that are also present in `grid_coords2`.

Parameters

grid_coords1
[numpy.ndarray] A 2D array of shape `(n_grid1, 3)` representing the points to check.

grid_coords2
[numpy.ndarray] A 2D array of shape `(n_grid2, 3)` representing the reference points.

Returns

indices
[numpy.ndarray] An array of indices corresponding to the points in `grid_coords1` that are found in `grid_coords2`.

Raises

ValueError
If `grid_coords1` or `grid_coords2` does not have the required shape of `(n, 3)`.

Examples

```
>>> grid_coords1 = np.array([[1, 2, 3], [4, 5, 6], [10, 11, 12]])
>>> grid_coords2 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
>>> find_indices_of_points(grid_coords1, grid_coords2)
array([0, 1])
```

pynetdesign.modelling.utils.nth_minimum

pynetdesign.modelling.utils.**nth_minimum**(*arr*, *n*)

Find the n-th minimum value in each row of a given 2D numpy array.

Parameters

arr
[numpy.ndarray] A 2D numpy array of shape (i, j) where i is the number of rows (features) and j is the number of columns (samples).

n
[int] The order of the minimum value to find (1-based index). For example, if n=2, the function finds the 2nd minimum value in each row.

Returns

nth_min_values
[numpy.ndarray] A 1D numpy array of shape (i,) containing the n-th minimum value from each row.

Raises

ValueError
If n is less than 1 or greater than the number of columns in the array.

pynetdesign.modelling.utils.check_zero_values

pynetdesign.modelling.utils.**check_zero_values**(*a*, *ErrorMessage=None*)

Check if the input value is close to zero and raise an error if true.

Parameters

a
[float or array_like] The input value or values to be checked.

ErrorMessage
[str , optional] Custom error message to be raised if the value is close to zero. Default is "Array values are close to zero".

Returns

bool
Returns True if the value is not close to zero.

Raises

RuntimeError
If the value is close to zero, raises a RuntimeError with the provided or default error message.

Examples

```
>>> check_zero_values(0.0000001)
RuntimeError: Array values are close to zero
```

```
>>> check_zero_values(1.0)
True
```

pynetdesign.modelling.utils.contains_only_numeric

pynetdesign.modelling.utils.**contains_only_numeric**(*seq*)

Check if a sequence (tuple, list, or numpy array) contains only numeric values.

Parameters

seq
[tuple, list, or numpy.ndarray] The sequence to be checked.

Returns

bool
True if all elements in the sequence are numeric (int, float, or complex), False otherwise.

Examples

```
>>> t1 = (1, 2.5, 3+4j)
>>> contains_only_numeric(t1)
True
```

```
>>> l1 = [1, 2.5, 3]
>>> contains_only_numeric(l1)
True
```

```
>>> a1 = np.array([1, 2.5, 3+4j])
>>> contains_only_numeric(a1)
True
```

```
>>> t2 = (1, 'a', 2.5)
>>> contains_only_numeric(t2)
False
```

pynetdesign.modelling.utils.compute_gcd_of_floats

pynetdesign.modelling.utils.**compute_gcd_of_floats**(*rx*, *ry*, *rz*)

Computes the greatest common divisor (GCD) of three positive float values.

Parameters

rx
[float] First float value.

ry
[float] Second float value.

rz
[float] Third float value.

Returns

`float`
The GCD of the three float values.

4.2 Visualization

4.2.1 Geometry

plot_stations_view(df[, view, x_offset, ...])	Plots either a map view or side view of stations as filled triangles with name labels.

pynetdesign.visualization.geometry.plot_stations_view

pynetdesign.visualization.geometry.**plot_stations_view**(*df*, *view='xy'*, *x_offset=None*, *y_offset=None*, *z_offset=None*, *offset_perc=0.1*, *plot_names=True*, *station_marker_size=100*, *grid_points=None*, *grid_marker_size=1*, *use_warnings=True*)

Plots either a map view or side view of stations as filled triangles with name labels. Optionally plots grid points as well. Color-codes the triangles based on noise level and adds a colorbar.

Parameters

df
[pandas.DataFrame] Data frame containing station names, coordinates, and noise levels. Expected columns: 'Name', 'X', 'Y', 'Z' (for side view), 'NoiseLevel'.

view
[str, optional] Specifies the view to plot. Options are 'xy' (X-Y), 'xz' (X-Z), or 'yz' (Y-Z). Default is 'xy'.

x_offset
[float, optional] Offset added to the min and max X coordinates for axis limits. If None, defaults to 10% of the X coordinate range.

y_offset
[float, optional] Offset added to the min and max Y coordinates for axis limits. If None, defaults to 10% of the Y coordinate range.

z_offset
[float, optional] Offset added to the min and max Z coordinates for axis limits (side view only). If None, defaults to 10% of the Z coordinate range.

offset_perc
[float, optional, default: 0.1] Percentage of coordinate range to use as offset. Used only if offset value is None.

plot_names
[bool, optional, default: True] Flag to plot station names or not (True or False)

station_marker_size
[int, optional, default: 100] Size of the station markers in the scatter plot

grid_points: :obj:`np.ndarray`, optional, default: None
Array of shape (ngrid, 3) representing grid points' coordinates (X, Y, Z).

grid_marker_size: :obj:`int`, optional, default: 10
Size of the grid points markers in the scatter plot

use_warnings: :obj:`bool`, optional, default: True
If True, enable warnings, if False, disable them.

Returns

fig
[matplotlib.figure.Figure] The figure object containing the plot.

ax
[matplotlib.axes.Axes] The axes object of the plot.

cax
[matplotlib.axes.Axes] The axes object for the colorbar.

cbar
[matplotlib.colorbar.Colorbar] The colorbar object.

4.2.2 Sensitivity

plot_sensitivity_slice(sens_data[, xi, yi, ...])	Plots magnitude sensitivity in map or side view with stations plotted as non-filled triangles with name labels.

pynetdesign.visualization.sensitivity.plot_sensitivity_slice

pynetdesign.visualization.sensitivity.**plot_sensitivity_slice**(*sens_data*, *xi=None*, *yi=None*, *zi=None*, *geometry_df=None*, *plot_title=None*, *clb_title=None*, *clb_range=None*, *dcolormap='cmc.bilbao'*, *plt_style='contourf'*, *plt_levels=None*, *plt_contours=True*, *plot_names=True*, *station_marker_size=100*, *use_warnings=True*)

Plots magnitude sensitivity in map or side view with stations plotted as non-filled triangles with name labels.

Parameters

sens_data
[numpy.ndarray] Data for sensitivity map

xi
[numpy.ndarray, optional] Vector of x coordinates for sensitivity

yi
[numpy.ndarray, optional] Vector of y coordinates for sensitivity

zi
[numpy.ndarray, optional] Vector of z coordinates for sensitivity

geometry_df
[pandas.dataframe, optional] Geometry data frame containing station names and coordinates

plot_title
[str, optional] Plot title (default: 'Magnitude sensitivity slice')

clb_title
[str, optional] Colorbar title (default: 'M_w')

clb_range
[tuple, optional] Colorbar range (default: automatic)

dcolormap
[str or matplotlib.colors.Colormap, default: 'cmc.batlow'] Data colormap to use for the plot

plt_style
[str, {'contourf','imshow'}, optional, default: 'contourf'] Plot style

plt_levels
[int, optional, default: 21 if None] Number of levels to plot in contours within the colorbar range. Has no effect on imshow

plt_contours
[bool, optional, default: True] Flag to plot contours

plot_names
[bool, optional, default: True] Flag to plot station names or not (True or False)

station_marker_size
[int, optional, default: 100] Size of the station markers in the scatter plot

use_warnings: :obj:`bool`, optional, default: True
If True, enable warnings, if False, disable them

Returns

fig
[matplotlib.figure.Figure] The figure object containing the plot.

ax
[matplotlib.axes.Axes] The axes object of the plot.

cax
[matplotlib.axes.Axes] The axes object for the colorbar.

cbar
[matplotlib.colorbar.Colorbar] The colorbar object.

Raises

ValueError
if two or more coordinate vectors are None

ValueError
if all three coordinate vectors are not None

ValueError
if clb_range is not a tuple with two numeric values

ValueError
if plt_style is not from the list of possible types

4.2.3 Visualization utilities

generate_colorscale(cmap, ncolors)	Generate a colorscale from a colormap and number of colors.
is_notebook()	Check if code is run in a notebook.
check_spyder()	Check if code is run inside Spyder.

pynetdesign.visualization.utils.generate_colorscale

pynetdesign.visualization.utils.**generate_colorscale**(*cmap*, *ncolors*)

Generate a colorscale from a colormap and number of colors.

Parameters

cmap
[mcolors.Colormap] A colormap instance from matplotlib.colors to be used for generating the colorscale.

ncolors
[int] The number of colors to generate in the colorscale. Must be a positive integer.

Returns

colorscale
[list] A list of colors in the format [[position, 'rgb(r, g, b)'], …] where *position* is a float between 0 and 1 representing the color position, and 'rgb(r, g, b)' is the color at that position in RGB format.

Raises

ValueError
If *ncolors* is less than 2 or if *cmap* is not an instance of mcolors.Colormap.

Examples

```
>>> from matplotlib import cm
>>> generate_colorscale(cm.viridis, 5)
[[0.0, 'rgb(0.267004, 0.004874, 0.329415)'], [0.25, 'rgb(0.229739, 0.322361, 0.
↪545706)'],
 [0.5, 'rgb(0.127568, 0.566949, 0.550556)'], [0.75, 'rgb(0.369214, 0.788888, 0.
↪382914)'],
 [1.0, 'rgb(0.993248, 0.906157, 0.143936)']]
[[0.0, 'rgb(68, 1, 84)'], [0.25, 'rgb(58, 82, 139)'], [0.5, 'rgb(32, 144, 140)'],
 [0.75, 'rgb(94, 201, 97)'], [1.0, 'rgb(253, 231, 36)']]
```

pynetdesign.visualization.utils.is_notebook

pynetdesign.visualization.utils.**is_notebook**()

Check if code is run in a notebook.

Returns

bool
True if yes (Jupyter notebook, qtconsole, google colab) or False if not

pynetdesign.visualization.utils.check_spyder

pynetdesign.visualization.utils.**check_spyder**()

Check if code is run inside Spyder.

Returns

is_spyder
[bool] True if yes or False if not

CHAPTER

FIVE

CITING

If you use PyNetDesign in your research, please cite it as follows:

Anikiev, D. (2026). *PyNetDesign: a Python framework for microseismic monitoring network design.* Zenodo. https://doi.org/10.5281/zenodo.22695001

In BibTeX format for the collective DOI

```
@software{Anikiev2026PyNetDesign,
   author    = {Anikiev, Denis},
   title     = {{PyNetDesign}: a {P}ython framework for microseismic monitoring␣
↪network design},
   year      = {2026},
   publisher = {Zenodo},
   doi       = {10.5281/zenodo.22695001},
   url       = {https://github.com/danikiev/PyNetDesign}
}
```

The DOI above represents all versions, and will always resolve to the latest one. To cite a particular version of PyNetDesign, please use the following format, for example for version 1.0.1:

Anikiev, D. (2026). *PyNetDesign: a Python framework for microseismic monitoring network design (1.0.1).* Zenodo. https://doi.org/10.5281/zenodo.22695002

In BibTeX format for a specific version

```
@software{Anikiev2026PyNetDesignVersion,
   author    = {Anikiev, Denis},
   title     = {{PyNetDesign}: a {P}ython framework for microseismic monitoring␣
↪network design},
   year      = {2026},
   publisher = {Zenodo},
   version   = {1.0.1},
   license   = {MIT},
```

```
    doi         = {10.5281/zenodo.22695002},
    url         = {https://github.com/danikiev/PyNetDesign}
}
```

5.1 Earlier release

Version 1.0.0 was deposited manually, before the automated GitHub to Zenodo integration was enabled, and therefore belongs to a separate Zenodo lineage. It is cross-linked to the current record and remains citable on its own:

Anikiev, D. (2025). *PyNetDesign: a Python framework for microseismic monitoring network design (v1.0.0)* [Computer software]. Zenodo.
https://doi.org/10.5281/zenodo.14945917

Warning

Magnitudes computed with version 1.0.0 are affected by an error in the seismic moment calculation that was corrected in version 1.0.1. If you are citing or reusing results obtained with 1.0.0, please see the *Changelog* and issue #3.

CHAPTER

SIX

CONTRIBUTING

Contributions, bug reports and suggestions are welcome.

6.1 Reporting a problem

Please open an issue at github.com/danikiev/PyNetDesign/issues. A useful report states the PyNetDesign version, the Python version, what you expected, what happened instead, and a small script or input file that reproduces the behaviour.

For a suspected error in the physics, the most useful report compares the computed result against a closed-form expectation, because that makes the discrepancy measurable.

6.2 Development setup

```
git clone https://github.com/danikiev/PyNetDesign.git
cd PyNetDesign
conda env create -f environment.yml -n pnd
conda activate pnd
pip install -e ".[dev,docs]"
```

6.3 Branches and releases

Development happens on `dev`, which is merged into `main` through a pull request. Release tags follow strict semantic versioning as `vMAJOR.MINOR.PATCH` and must point at a commit contained in `main`; the version itself is derived from the tag by `setuptools_scm`. The full procedure is described in `RELEASE.md`.

6.4 Tests

Run the suite from the repository root:

```
pytest pytests/
```

Continuous integration runs the same suite on Python 3.9 to 3.12, validates the Zenodo metadata, and builds this documentation.

Please add a test with any change to the algorithms. The convention in this project is to assert against an independently written closed form rather than against a recorded output, so that a test states the physics instead of freezing the current behaviour.

6.5 Documentation

Build the documentation locally and view it with the helper scripts:

```
./build-docs.sh          # Linux, macOS;  build-docs.bat on Windows
./serve-docs.sh          # rebuilds incrementally, then serves on port 8000
```

`serve-docs` accepts `--port` and `--no-build`, and both scripts pass `SPHINXOPTS` through to Sphinx. Without the scripts the equivalent is `python -m sphinx -M html docs/source docs/build`.

Docstrings follow the numpydoc style; equations use `.. math::` and literature is cited with `:cite:t:` or `:cite:p:` against `docs/source/references.bib`.

6.6 Changelog

`CHANGELOG.md` follows Keep a Changelog. Add your entry under `[Unreleased]` in the appropriate section, referencing the issue or pull request number. The documentation page is generated from this file, so there is nothing else to update.

CHAPTER

SEVEN

CHANGELOG

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

7.1 v1.1.0 - 2026-10-12

Compare with v1.0.1 on GitHub

Feature release: phase-resolved wave modes, per-receiver free surface and receiver projection, synthetic geometry builders, a documentation site in HTML and PDF, and branding.

7.1.1 Added

- phase-resolved wave modes: `wave_mode` accepts `'P'`, `'SV'`, `'SH'`, `'S'` and `'PS'`, where `'S'` is a composite mode keeping the more detectable of the SV and SH branches
- root-mean-square radiation-pattern magnitudes per phase, `R_P = sqrt(4/15) ~ 0.52`, `R_SV = sqrt(7/30) ~ 0.48` and `R_SH = sqrt(1/6) ~ 0.41`, averaged over the focal sphere of a double-couple source after Boore and Boatwright (1984) and Hallo and Eisner (2013), exposed through `phase_radiation_pattern`
- `rad_pattern_p`, `rad_pattern_s` and `rad_patterns` overrides on `mag_sensitivity_grid`, so that results computed with another convention stay reproducible; pass `rad_pattern_s=0.63` to use the combined S-wave value
- receiver projection from the arriving polarization vectors, through `get_phase_station_directionality`, which is correct for inclined and curved cables rather than only for a vertical one
- projection of single vertical-component stations onto the vertical axis, driven by a `Components` column; three-component stations record the full vector and are unaffected
- per-station free surface amplification through `compute_free_surface_coefficients`, with `fs_mode`, `fs_level` and `fs_deviation` parameters and an optional `Surface` column carrying per-station weights
- helpers `normalize_wave_mode`, `wave_mode_phases`, `wave_mode_has_p`, `wave_mode_has_s`, `wave_mode_s_phases`, `resolve_radiation_pattern`, `check_min_stations` and `geometry_requires_receiver_projection`
- `pynetdesign.modelling.geometry`, a module of synthetic geometry builders: `generate_geometry` dispatches to `generate_borehole_geometry` for vertical, inclined and deviated boreholes, to `generate_surface_geometry` for line, zigzag, L, square and double-line layouts, and to `generate_darkfiber_geometry` for a continuous dark-fibre path with turns. Each returns geometry text lines that `read_geometry` can read back, including a depth-dependent noise profile for boreholes
- `save_geometry`, writing either generated text lines or a geometry DataFrame back to the `read_geometry` format, with `Z` restored to an elevation so that a round trip is lossless

- `decimate_geometry`, keeping every n-th channel while preserving the metadata, for studying the effect of channel spacing on a dense array
- `read_geometry` reads the optional `Surface` and `Components` columns, so the per-station free surface weights and the single vertical component stations introduced above can be described in a geometry file
- `read_geometry` records the path it read in `attrs['file_path']`
- test modules covering wave modes and radiation patterns, receiver projection, the free surface correction, network detectability, and phase-resolved sensitivity end to end
- `test_magnitude_consistency.py`, which builds the forward displacement spectrum independently of the package and checks that `calculate_M0` inverts it exactly, that the spectral conversion `S(f)` is applied once and not twice across `retrieve_min_amps` and `calculate_M0`, and that an event at the threshold magnitude records exactly the requested signal-to-noise ratio for each of the five amplitude types, gauge-length bridge included
- Sphinx documentation under `docs/`, published to GitHub Pages at <https://danikiev.github.io/PyNetDesign/>, with an overview whose quick-link panels lead to every section and to the PDF edition, a getting-started guide covering installation and the input file formats, a methodology chapter, an example gallery and a generated API reference
- `methodology.rst` documenting the theory actually implemented: spectral conventions, the attenuation operator, the peak frequency, the detection threshold and its conversion coefficients, the free surface correction, receiver projection and DAS directionality, the radiation pattern with the full root-mean-square derivation, the homogeneous seismic moment relation with a standalone derivation from the far-field point source, and the network detectability criterion
- `docs/source/references.bib` with the seven works cited from the docstrings and the methodology, so the `:cite:` roles added earlier now resolve
- `citing`, `contributing` and `credits` documentation pages, plus a changelog page that is generated from `CHANGELOG.md` when the documentation is built, so the changelog has a single source and the generated page is not committed. On that page each version links to its changes on GitHub. These pages follow the main chapters in the order Citing, Contributing, Changelog, Credits
- `docs.yml` workflow building the documentation on every push and deploying it to GitHub Pages from `main`, including a check that the built version matches the tag and, after each deployment, a check that the live site shows the deployed version
- `push-tag.yml` workflow, so that pushing a `vX.Y.Z` tag validates it and starts a `docs.yml` run on `main` that publishes the documentation of that tag. The run is on `main` because GitHub Pages ignores a deployment made from a tag when `main` has already deployed the same commit (actions/deploy-pages#383)
- `build-docs.bat` and `build-docs.sh` for building the documentation, and `serve-docs.bat` and `serve-docs.sh` for viewing it locally. `serve-docs` rebuilds incrementally before serving, so only the pages that changed are regenerated, and accepts `--port` and `--no-build`. Both accept `SPHINXOPTS`, and neither needs `make`: they call `python -m sphinx -M` directly. Both also take `--pdf`, which builds the PDF edition and places it where the documentation links to it
- a PDF edition of the documentation, produced from the same sources through lualatex and linked from the site header and the front page, so the whole manual can be read offline as one file. Each web page keeps its own reference list, while the PDF lists the cited works once, in a bibliography at its end that its citations link to. `docs.yml` builds it on every run, publishes it with the site and keeps a copy as a workflow artifact
- `install.sh` and `uninstall.sh`, the Linux and macOS counterparts of the existing Windows scripts. `install.sh` takes `--dev` or `--no-dev` to answer the development-tools question in advance, and `uninstall.sh` takes `--yes`
- branding from the PyNetDesign brand package: the logo in the documentation header and the README, and the full browser icon set, comprising `favicon.ico`, a scalable `favicon.svg`, an Apple touch icon and a web app manifest. Both themes serve the same transparent logo, the one the README uses, so the mark keeps its own clear

space and is never filtered or recoloured by the theme, because the brand gray is fixed and must not be inverted. Hovering the header logo swaps in the brand's own orange-only lockup, which carries the same geometry at the same viewBox and differs only in colour, so the mark neither moves nor resizes and no filter is involved

- brand colours in the documentation stylesheet. Signal Orange reaches only 2.87:1 against white, so link text in light mode is darkened to `#B84806`, which measures 4.81:1 against the light page surface, while the unmodified orange is used on the dark surfaces where it measures 5.17:1

7.1.2 Changed

- **S-wave results differ from 1.0.x.** The S branch now uses the phase-resolved radiation patterns instead of the combined value 0.63, which raises S-wave thresholds by `2/3*log10(0.63/sqrt(7/30))`, about 0.077 magnitude units. Pass `rad_pattern_s=0.63` to recover the previous behaviour
- **the free surface correction is applied per receiver.** In 1.0.x `free_surface=True` amplified every receiver irrespective of its depth; it now applies only to receivers at the free surface, so geometries mixing surface and downhole receivers are treated correctly. The default mode is `'auto'`, which derives the indicator from the receiver depths or from the `Surface` column when present
- `DetectionParameters` takes `fs_mode`, `fs_level` and `fs_deviation`; the `free_surface` flag is deprecated but still accepted, mapping to `fs_mode='on'` or `'off'`
- `mag_detectable` accepts the phase-resolved wave modes
- **example 7 is now the 1-km DAS array benchmark.** It models a 1-km-long vertical fibre with 1 m channel spacing, so 1000 channels, a constant strain-rate noise level of 1.95e-9 1/s and a 10 m gauge length, in a homogeneous medium with Vp = 2370 m/s, Vp/Vs = 2, Qp = Qs = 100 and rho = 2500 kg/m^3, requiring detection on at least 60 channels. It shows the three features that follow from a fibre recording only the strain along itself: the threshold magnitude grows with offset, P-wave sensitivity is worst at the array mid-depth once the offset exceeds half the array length, and S-wave sensitivity vanishes on the cable axis and degrades below the array
- the methodology and credits pages close with a `References` section rather than a rubric, so the bibliography is reachable from the table of contents
- the credits page acknowledges the works the implementation rests on individually, adding Aki and Richards (2002) for the far-field point source, the radiation-pattern convention and the free surface, and Cerveny (2001) for the ray-theoretical global absorption factor
- receiver projection is decided from the geometry itself, so `use_station_directionality` is only needed to force tangent projection for a station geometry
- **a single environment file.** `environment.yml` now pins only the interpreter, and every dependency is declared in `pyproject.toml`, as runtime `dependencies` or in the `dev` and `docs` optional groups. Install with `conda env create -f environment.yml -n pnd` followed by `pip install -e .` or `pip install -e ".[dev,docs]"`. This removes the duplicated dependency lists, which had drifted: they advertised `scipy` and `plotly`, neither of which is imported, and omitted `cmcrameri`, which is required
- `pyproject.toml` declares `requires-python = ">=3.9,<3.13"`, the MIT license and its file, project URLs, and a fuller set of keywords and classifiers; Python 3.8 is no longer advertised, since it is not tested and the package needs 3.9
- `install.bat` creates one environment, `pnd`, from the single file and asks whether to add the development tools, rather than choosing between two conda files and two environment names
- the README documents the documentation site and the PDF download, corrects the Python range and the dependency list, and fixes the case of the clone URL

7.1.3 Removed

- `environment-dev.yml`, superseded by the `dev` and `docs` optional dependency groups
- the empty `__init__.py` at the repository root, which made the repository root look like an importable package and confused setuptools flat-layout discovery
- the `rays_p` and `rays_s` arguments of `get_ray_station_directionality`, which could never be populated because the homogeneous package has no ray tracer. The function is deprecated in favour of `get_phase_station_directionality` and now emits a `DeprecationWarning`

7.1.4 Fixed

- the `.. only:: html` blocks on the front page held their content at the wrong indentation, so the directive guarded nothing and the LaTeX build was asked to typeset icon cards it cannot render
- `examples/README.txt` promised a raytracing example that this package does not ship, and carried a stale product name
- a grid point that coincides with a receiver no longer emits a spurious divide-by-zero warning. Such a point gives `r = 0`, hence `t* = 0` and an infinite peak frequency, which the corner-frequency clamp already turned into the right answer; only the warning was wrong. Coincidence is ordinary with a dense DAS cable, where every channel sits on a grid line
- `pytest.ini` named `tests` as a directory to avoid, which does not exist here, and never named the directory that does. It now sets `testpaths = pytests`, so `examples` and `data` are not searched for tests
- `test_read_geometry.py` resolved its sample geometry files relative to the working directory, so it only passed when pytest was started from the repository root. The paths are now anchored to the test file
- the station-count guard in `mag_detectable` compared against the grid axis instead of the station axis, so a request for more receivers than the geometry has was not reported clearly
- the default dark-fibre turn distances are now fractions of the cable length, at one quarter, one half and three quarters, so that `generate_geometry(mode='darkfiber')` works for any cable. The previous fixed distances of 6, 12 and 18 km exceeded the default cable length and made the default call fail; for a 24 km cable the layout is unchanged
- `uninstall.bat` could never remove an environment. `ENV_NAME` and `CONFIRM` were assigned and read inside the same `for` block, so `cmd` substituted their pre-loop empty values: the script printed an empty environment name and always took the skip branch regardless of the answer. It now uses delayed expansion, and matches environment names anchored to the start of the line so that an unrelated environment cannot be matched through its path

7.2 v1.0.1 - 2026-09-10

Compare with v1.0.0 on GitHub

Bugfix release correcting the seismic moment calculation.

7.2.1 Fixed

- the seismic moment is no longer underestimated by a factor of `2*pi*f` (#3). `calculate_M0` applied the amplitude-to-displacement spectral conversion a second time, after `retrieve_min_amps` had already applied it through `get_scaling_displacement`. Minimum detectable moment magnitudes reported by v1.0.0 were therefore too low by `dMw = (2/3)*log10(2*pi*f_r)`, which is 1.20 at a representative frequency of 10 Hz and 1.87 at 100 Hz. Because the representative frequency `f_r = min(v*Q/(pi*r), f_c)` depends on the source-receiver distance, the correction varies across the grid and changes the shape of the sensitivity maps, not only their colour scale

- `pynetdesign.visualization` can be imported on Python 3.9 again; `plot_sensitivity_slice` used a `match` statement, which is syntax available only from Python 3.10 onwards

7.2.2 Changed

- `calculate_M0` now documents the seismic moment relation it evaluates and states that the amplitudes it takes are the displacement amplitude spectrum, with units given for the remaining arguments

7.2.3 Removed

- `loc_uncertainty_grid`, which called `calculate_pdf` and `get_uncertainty_from_pdf`. Neither function has ever been part of the package, so every call raised `NameError`. Location uncertainty will be reintroduced later together with the functions it depends on

7.2.4 Added

- `pytests/test_magnitude.py`, covering the seismic moment relation, the moment magnitude conversion and its inverse, the amplitude-type scaling coefficients, the peak frequency and its corner-frequency clamp, and the DAS strain and strain-rate detection thresholds. The closed-form checks fail on the v1.0.0 formula and so guard against a recurrence of #3

7.3 v1.0.0 - 2025-04-01

Release on GitHub

Initial public release, archived on Zenodo as 10.5281/zenodo.14945917. This changelog was added retroactively so that the release is documented alongside its archived record; the code itself is unchanged from the deposited version.

7.3.1 Added

- magnitude sensitivity modelling for a 3D elastic homogeneous medium with attenuation: the minimum moment magnitude of a microseismic event that is theoretically detectable on a given network, evaluated on a 3D grid of potential source locations
- `get_mag_sensitivity` and `get_mag_sensitivity_local` high-level drivers, dispatching on the velocity model type and handling combined geometries
- `mag_sensitivity_grid` vectorized homogeneous-medium engine computing per-station minimum detectable moment magnitudes
- seismic moment to moment magnitude conversion after Kanamori (1977), and its inverse
- peak frequency estimation after Eisner et al. (2013), limited by a corner frequency defaulting to 100 Hz
- conversion of station noise levels to minimum detectable amplitudes for displacement, particle velocity, particle acceleration, strain, and strain-rate inputs, scaled by the required signal-to-noise ratio
- distributed acoustic sensing (DAS) support: gauge length parsing from the geometry header with unit conversion, strain and strain-rate noise types, and a directionality correction accounting for sensitivity along the cable axis
- support for station networks, DAS cables, and combined geometries evaluated together
- free surface amplification correction for receivers recording at the daily surface
- network detectability criterion based on the minimum number of stations required for P-wave, S-wave, or joint P- and S-wave detection
- ASCII readers for network geometry in local and global (WGS84) coordinates, with conversion of geographic to local Cartesian coordinates

- ASCII reader for homogeneous velocity models
- `DetectionParameters` container with validation for signal-to-noise ratios, station counts, and frequencies
- station geometry visualization in map and side views, colour-coded by noise level
- magnitude sensitivity slice visualization with contour and image rendering
- ten worked examples covering surface station networks, vertical, inclined and twice-inclined borehole DAS, depth-varying DAS noise profiles, and combined station plus DAS networks
- conda environment files for user and development installations, and Windows installation scripts
- MIT license

CHAPTER

EIGHT

CREDITS

8.1 Author

PyNetDesign is developed by Denis Anikiev, github.com/danikiev.

8.2 Acknowledgments

The methodology builds on published work on seismic wave propagation and on microseismic monitoring network design, in particular:

- Aki and Richards [2002] for the far-field radiation of a point moment-tensor source, the radiation-pattern convention and the free surface boundary condition;
- Červený [2001] for ray theory and the global absorption factor;
- Kanamori [1977] for the moment magnitude scale;
- Eisner *et al.* [2013] for the peak frequency of direct waves;
- Boore and Boatwright [1984] for the average body-wave radiation coefficients;
- Hallo [2012] and Hallo and Eisner [2013] for monitoring network sensitivity and the influence of the double-couple source model.

PyNetDesign stands on the open-source scientific Python ecosystem, and the author is grateful to the developers of NumPy, Pandas, Matplotlib and cmcrameri, as well as to the Sphinx project and the PyData Sphinx theme used to build this documentation.

8.3 License

PyNetDesign is released under the MIT License.

BIBLIOGRAPHY

[1] K. Aki and P. G. Richards. *Quantitative Seismology*. University Science Books, 2002. ISBN 9780935702965.

[2] V. Červený. *Seismic Ray Theory*. Cambridge University Press, 2001. doi:10.1017/CBO9780511529399.

[3] L. Eisner, D. Gei, M. Hallo, I. Opršal, and M. Y. Ali. The peak frequency of direct waves for microseismic events. *GEOPHYSICS*, 78(6):A45–A49, November 2013. doi:10.1190/geo2013-0197.1.

[4] M. Hallo and L. Eisner. Influence of the double-couple source model on the sensitivity of microseismic monitoring networks. In *London 2013, 75th eage conference en exhibition incorporating SPE Europec*, London2013. EAGE Publications BV, 2013. doi:10.3997/2214-4609.20131023.

[5] David M. Boore and John Boatwright. Average body-wave radiation coefficients. *Bulletin of the Seismological Society of America*, 74(5):1615–1621, October 1984. doi:10.1785/bssa0740051615.

[6] H. Kanamori. The energy release in great earthquakes. *Journal of Geophysical Research*, 82(20):2981–2987, 1977. doi:10.1029/jb082i020p02981.

[7] M. Hallo. Microseismic surface monitoring network design - sensitivity and accuracy. In *74th EAGE Conference and Exhibition incorporating EUROPEC 2012*, Copenhagen2012. EAGE Publications BV, June 2012. doi:10.3997/2214-4609.20148336.

INDEX

A

are_points_inside() (*in module pynetdesign.modelling.utils*), 287

C

calculate_fpeak() (*in module pynetdesign.modelling.homo*), 266
calculate_M0() (*in module pynetdesign.modelling.homo*), 265
calculate_Mw() (*in module pynetdesign.modelling.utils*), 278
calculate_seismic_moment() (*in module pynetdesign.modelling.utils*), 278
check_min_stations() (*in module pynetdesign.modelling.utils*), 281
check_spyder() (*in module pynetdesign.visualization.utils*), 294
check_zero_values() (*in module pynetdesign.modelling.utils*), 288
combine_geometry() (*in module pynetdesign.modelling.io*), 272
compute_free_surface_coefficients() (*in module pynetdesign.modelling.utils*), 281
compute_gcd_of_floats() (*in module pynetdesign.modelling.utils*), 289
contains_only_numeric() (*in module pynetdesign.modelling.utils*), 289

D

decimate_geometry() (*in module pynetdesign.modelling.io*), 273
DetectionParameters (*class in pynetdesign.modelling.classes*), 274

F

find_indices_of_points() (*in module pynetdesign.modelling.utils*), 287

G

generate_borehole_geometry() (*in module pynetdesign.modelling.geometry*), 268
generate_colorscale() (*in module pynetdesign.visualization.utils*), 293
generate_darkfiber_geometry() (*in module pynetdesign.modelling.geometry*), 268
generate_geometry() (*in module pynetdesign.modelling.geometry*), 268
generate_grid() (*in module pynetdesign.modelling.utils*), 285
generate_surface_geometry() (*in module pynetdesign.modelling.geometry*), 268
geometry_requires_receiver_projection() (*in module pynetdesign.modelling.utils*), 283
get_mag_sensitivity() (*in module pynetdesign.modelling.core*), 261
get_mag_sensitivity_local() (*in module pynetdesign.modelling.core*), 262
get_phase_station_directionality() (*in module pynetdesign.modelling.utils*), 282
get_ray_station_directionality() (*in module pynetdesign.modelling.utils*), 284
get_scaling_displacement() (*in module pynetdesign.modelling.utils*), 279
get_slice_coordinates() (*in module pynetdesign.modelling.utils*), 286
global2local() (*in module pynetdesign.modelling.io*), 273

I

is_combined_geometry() (*in module pynetdesign.modelling.io*), 273
is_notebook() (*in module pynetdesign.visualization.utils*), 293

M

mag_detectable() (*in module pynetdesign.modelling.utils*), 280
mag_sensitivity_grid() (*in module pynetdesign.modelling.homo*), 263

N

normalize_wave_mode() (*in module pynetdesign.modelling.utils*), 276

nth_minimum() (*in module pynetdesign.modelling.utils*), 288

P

phase_radiation_pattern() (*in module pynetdesign.modelling.utils*), 277
plot_sensitivity_slice() (*in module pynetdesign.visualization.sensitivity*), 291
plot_stations_view() (*in module pynetdesign.visualization.geometry*), 290

R

read_geometry() (*in module pynetdesign.modelling.io*), 269
read_velocity() (*in module pynetdesign.modelling.io*), 271
reshape_data() (*in module pynetdesign.modelling.utils*), 286
reshape_grid() (*in module pynetdesign.modelling.utils*), 285
resolve_radiation_pattern() (*in module pynetdesign.modelling.utils*), 277
retrieve_min_amps() (*in module pynetdesign.modelling.utils*), 279

S

save_geometry() (*in module pynetdesign.modelling.io*), 270

V

validate_parameters() (*in module pynetdesign.modelling.homo*), 267

W

wave_mode_has_p() (*in module pynetdesign.modelling.utils*), 276
wave_mode_has_s() (*in module pynetdesign.modelling.utils*), 276
wave_mode_phases() (*in module pynetdesign.modelling.utils*), 276
wave_mode_s_phases() (*in module pynetdesign.modelling.utils*), 276

Online documentation



danikiev.github.io/PyNetDesign