geoana Documentation

Release 0.0.4

SimPEG Developers

Aug 20, 2018

Contents

getting_started | *connecting* | *installing* | *license* | documentation |

geoana is a collection of (mostly) analytic functions in geophysics. We take an object oriented approach with the aim of having users be able to readily interact with the functions using Jupyter

CHAPTER 1

Getting started

- If you do not already have python installed, we recommend downloading and installing it through anaconda
- Install geoana
- Browse the gallery for ideas and example usage
- Read the documentation for more information on the library and what it can do

CHAPTER 2

Connecting with the community

geoana is a part of the larger SimPEG ecosystem. There are several avenues for connecting:

- a mailing list for questions and general news items: https://groups.google.com/forum/#!forum/simpeg
- a newsletter where meeting notices and re-caps are posted: http://eepurl.com/bVUoOL
- a slack group for real-time chat with users and developers of SimPEG: http://slack.simpeg.xyz/

CHAPTER 3

Installing

geoana is available on pypi and can be installed by opening a command window and running:

```
pip install geoana
```

To install from source, you can

```
git clone https://github.com/simpeg/geoana.git
python setup.py install
```

CHAPTER 4

License

geoana is licensed under the MIT license .

CHAPTER 5

Contents

5.1 Examples

5.1.1 Electromagnetics

Note: Click *here* to download the full example code

Magnetostatic Vector Potentials: Dipole and Loop Sources

In this example, we plot the vector potential for a dipole and a loop source in a wholespace.

We can vary the magnetic permeability of the wholespace, location and orientation of the sources. For the dipole source, we can vary the moment, and for the loop source, we can vary the radius and current through the loop.

author Lindsey Heagy (@lheagy)

date June 6, 2018

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm
from scipy.constants import mu_0, epsilon_0

from geoana import utils, spatial
from geoana.em import static
```

Setup

define the location orientation and source, physical properties of the wholespace and source parameters

```
mu = mu_0  # permeability of free space (this is the default)
location=np.r_[0., 0., 0.]  # location of the dipole
orientation='Z'  # vertical dipole (can also be a unit-vector)

# dipole parameters
moment = 1

# loop source parameters
current = 1
radius = 20
```

Magnetostatic Dipole and Loop

Here, we build the geoana magnetic dipole in a wholespace and circular loop in a wholespace using the parameters defined above. For a full list of the properties you can set on a dipole, see the *geoana.em.static.MagneticDipoleWholeSpace* docs and for the circular loop source, see the *geoana.em.static.CircularLoopWholeSpace* docs

```
dipole = static.MagneticDipoleWholeSpace(
    mu=mu, location=location,
    orientation=orientation , moment=moment
)

loop = static.CircularLoopWholeSpace(
    mu=mu, location=location,
    orientation=orientation, current=current,
    radius=radius
)
```

Evaluate vector potential

Next, we construct a grid where we want to plot the vector potential and evaluate

```
x = np.linspace(-50, 50, 100)
y = np.linspace(-50, 50, 100)
xyz = utils.ndgrid([x, y, np.r_[0]])

# evaluate the vector potential
a_dipole = dipole.vector_potential(xyz)
a_loop = loop.vector_potential(xyz)
```

and define plotting code to plot an image of the amplitude of the vector field / flux as well as the streamlines

```
def plot_amplitude(ax, v):
    v = spatial.vector_magnitude(v)
    plt.colorbar(
        ax.pcolormesh(
            x, y, v.reshape(len(x), len(y), order='F'), norm=LogNorm()
        ), ax=ax
    )
    ax.axis('square')
    ax.set_xlabel('x (m)')
    ax.set_ylabel('y (m)')
```

(continues on next page)

(continued from previous page)

```
# plot streamlines
def plot_streamlines(ax, v):
    vx = v[:, 0].reshape(len(x), len(y), order='F')
    vy = v[:, 1].reshape(len(x), len(y), order='F')
    ax.streamplot(x, y, vx.T, vy.T, color='k')
```

Create subplots for plotting the results. Loop over frequencies and plot the electric and magnetic fields along a slice through the center of the dipole.

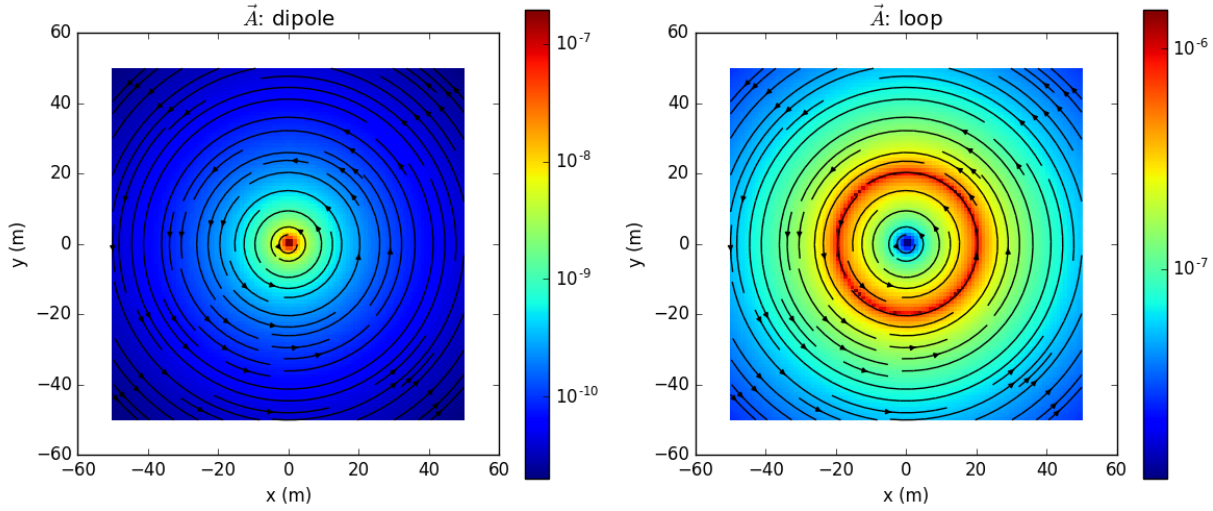
```
fig, ax = plt.subplots(1, 2, figsize=(12, 5))

# plot dipole vector potential
plot_amplitude(ax[0], a_dipole)
plot_streamlines(ax[0], a_dipole)

# plot loop vector potential
plot_amplitude(ax[1], a_loop)
plot_streamlines(ax[1], a_loop)


# set the titles
ax[0].set_title("$\\vec{A}$: dipole")
ax[1].set_title("$\\vec{A}$: loop")

# format so text doesn't overlap
fig.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 4.781 seconds)

Note: Click *here* to download the full example code

Electric Dipole in a Whole Space: Frequency Domain

In this example, we plot electric and magnetic flux density due to an electric dipole in a whole space. Note that you can also examine the current density and magnetic field.

We can vary the conductivity, magnetic permeability and dielectric permittivity of the wholespace, the frequency of the source and whether or not the quasistatic assumption is imposed.

author Lindsey Heagy (@lheagy)

date June 2, 2018

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm
from scipy.constants import mu_0, epsilon_0

from geoana import utils, spatial
from geoana.em import fdem
```

Setup

define frequencies that we want to examine, physical properties of the wholespace and location and orientation of the dipole

```
frequencies = np.logspace(0, 4, 3)  # frequencies to examine
sigma = 1.  # conductivity of 1 S/m
mu = mu_0  # permeability of free space (this is the default)
epsilon=epsilon_0  # permittivity of free space (this is the default)
location=np.r_[0., 0., 0.]  # location of the dipole
orientation='Z'  # vertical dipole (can also be a unit-vector)
quasistatic=False  # don't use the quasistatic assumption
```

Electric Dipole

Here, we build the geoana electric dipole in a wholespace using the parameters defined above. For a full list of the properties you can set on an electric dipole, see the *geoana.em.fdem.ElectricDipoleWholeSpace* docs

```
edipole = fdem.ElectricDipoleWholeSpace(
    sigma=sigma, mu=mu, epsilon=epsilon,
    location=location, orientation=orientation,
    quasistatic=False
)
```

Evaluate fields and fluxes

Next, we construct a grid where we want to plot electric fields

```
x = np.linspace(-50, 50, 100)
z = np.linspace(-50, 50, 100)
xyz = utils.ndgrid([x, np.r_[0], z])
```

and define plotting code to plot an image of the amplitude of the vector field / flux as well as the streamlines

```
def plot_amplitude(ax, v):
    v = spatial.vector_magnitude(v)
    plt.colorbar(
        ax.pcolormesh(
            x, z, v.reshape(len(x), len(z), order='F'), norm=LogNorm()
        ), ax=ax
    )
    ax.axis('square')
    ax.set_xlabel('x (m)')
    ax.set_ylabel('z (m)')


# plot streamlines
def plot_streamlines(ax, v):
    vx = v[:, 0].reshape(len(x), len(z), order='F')
    vz = v[:, 2].reshape(len(x), len(z), order='F')
    ax.streamplot(x, z, vx.T, vz.T, color='k')
```

Create subplots for plotting the results. Loop over frequencies and plot the electric and magnetic fields along a slice through the center of the dipole.

```
fig_e, ax_e = plt.subplots(
    2, len(frequencies), figsize=(5*len(frequencies), 7)
)
fig_b, ax_b = plt.subplots(
    2, len(frequencies), figsize=(5*len(frequencies), 7)
)

for i, frequency in enumerate(frequencies):

    # set the frequency of the dipole
    edipole.frequency = frequency

    # evaluate the electric field and magnetic flux density
    electric_field = edipole.electric_field(xyz)
    magnetic_flux_density = edipole.magnetic_flux_density(xyz)

    # plot amplitude of electric field
    for ax, reim in zip(ax_e[:, i], ['real', 'imag']):
        # grab real or imag component
        e_plot = getattr(electric_field, reim)

        # plot both amplitude and streamlines
        plot_amplitude(ax, e_plot)
        plot_streamlines(ax, e_plot)

        # set the title
        ax.set_title(
            'E {} at {:1.1e} Hz'.format(reim, frequency)
        )

    # plot the amplitude of the magnetic field (note the magnetic field is into
    # and out of the page in this geometry, so we don't plot vectors)
    for ax, reim in zip(ax_b[:, i], ['real', 'imag']):
        # grab real or imag component
        b_plot = getattr(magnetic_flux_density, reim)
```

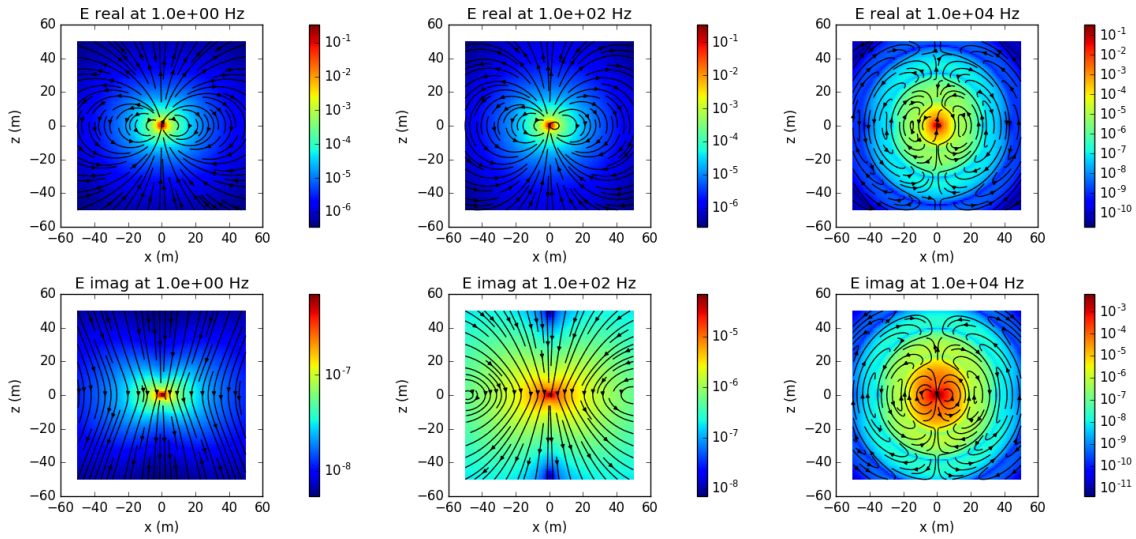
(continues on next page)

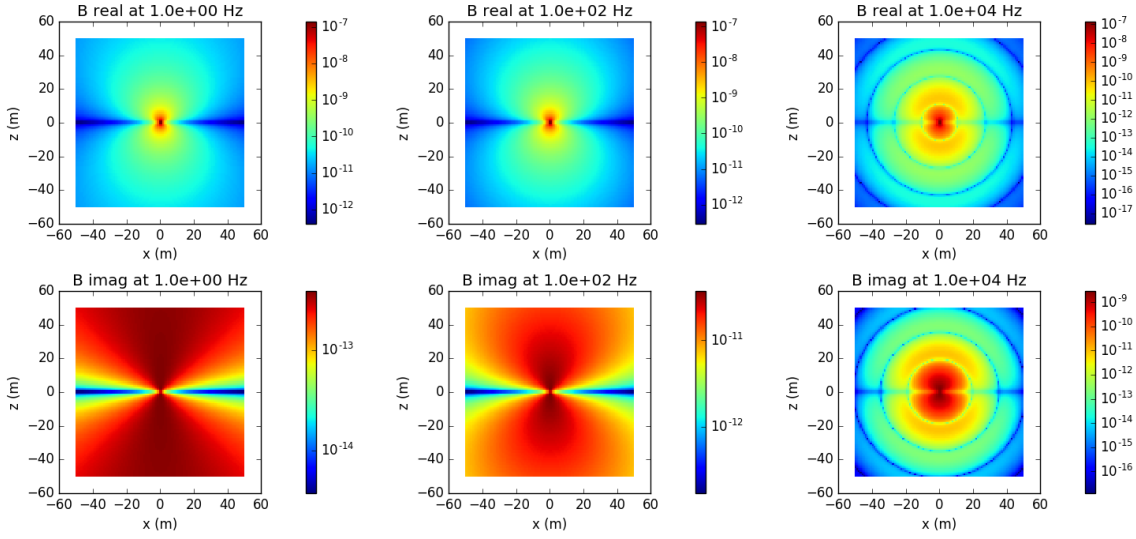
(continued from previous page)

```
        # plot amplitude
        plot_amplitude(ax, b_plot)

        # set the title
        ax.set_title(
            'B {} at {:1.1e} Hz'.format(reim, frequency)
        )

# format so text doesn't overlap
fig_e.tight_layout()
fig_b.tight_layout()
plt.show()
```

- 

- 

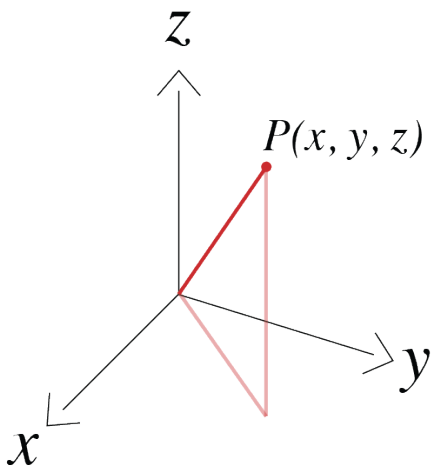
Total running time of the script: (0 minutes 8.108 seconds)

5.2 Conventions

5.2.1 Coordinate systems

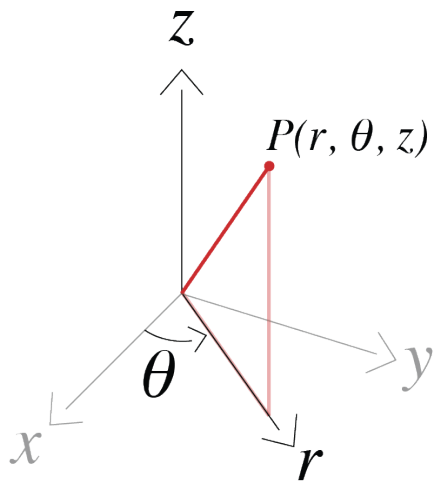
Cartesian

We use a right-handed coordinate system (x, y, z) with z-positive up as shown in the Figure below.



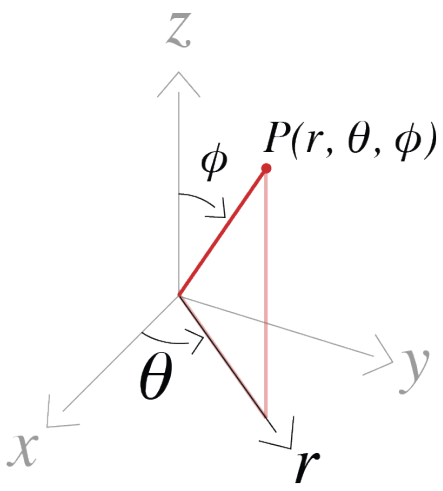
Cylindrical

We again work with z-positive up and use θ to denote the azimuthal angle, thus the coordinate system is defined as (r, θ, z).



Spherical

We use r for the radial direction, θ for the azimuthal direction, and ϕ for the polar direction as shown in the figure below.



5.2.2 Fourier Transform

For analysis and solutions in the frequency domain we use the $e^{i\omega t}$ Fourier transform convention. Thus, we define our Fourier Transform pair as

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt$$

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) e^{i\omega t} d\omega$$

where ω is angular frequency, t is time, $F(\omega)$ is the function defined in the frequency domain and $f(t)$ is the function defined in the time domain.

5.3 Electromagnetics

5.3.1 Base

class geoana.em.base.**BaseDipole**(***kwargs*)

Bases: *geoana.em.base.BaseEM*

Base class for dipoles.

Required Properties:

- **epsilon** (Float): Permitivity value (F/m), a float in range [0.0, inf], Default: 8.854187817620389e-12
- **location** (Vector3): location of the electric dipole source, a 3D Vector of <class 'float'> with shape (3), Default: ZERO
- **mu** (Float): Magnetic permeability (H/m), a float in range [0.0, inf], Default: 1.2566370614359173e-06
- **orientation** (Vector3): orientation of dipole, a 3D Vector of <class 'float'> with shape (3), Default: X
- **sigma** (Float): Electrical conductivity (S/m), a float in range [0.0, inf], Default: 1.0

cross_orientation(*xyz*)

Take the cross product between a grid and the orientation of the dipole

distance (*xyz*)
Distance from the dipole location

dot_orientation (*xyz*)
Take the dot product between a grid and the orientation of the dipole

location
location (Vector3): location of the electric dipole source, a 3D Vector of <class 'float'> with shape (3), Default: ZERO

orientation
orientation (Vector3): orientation of dipole, a 3D Vector of <class 'float'> with shape (3), Default: X

vector_distance (*xyz*)
Vector distance from the dipole location :param numpy.ndarray xyz: grid

class geoana.em.base.**BaseEM**(***kwargs*)
Bases: properties.base.base.HasProperties

Base class for electromanetics. Contains physical properties that are relevant to all problems that use Maxwell's equations

Required Properties:

- **epsilon** (Float): Permitivity value (F/m), a float in range [0.0, inf], Default: 8.854187817620389e-12
- **mu** (Float): Magnetic permeability (H/m), a float in range [0.0, inf], Default: 1.2566370614359173e-06
- **sigma** (Float): Electrical conductivity (S/m), a float in range [0.0, inf], Default: 1.0

epsilon
epsilon (Float): Permitivity value (F/m), a float in range [0.0, inf], Default: 8.854187817620389e-12

mu
mu (Float): Magnetic permeability (H/m), a float in range [0.0, inf], Default: 1.2566370614359173e-06

sigma
sigma (Float): Electrical conductivity (S/m), a float in range [0.0, inf], Default: 1.0

class geoana.em.base.**BaseElectricDipole**(***kwargs*)
Bases: *geoana.em.base.BaseDipole*

Base class for electric current dipoles

Required Properties:

- **current** (Float): magnitude of the injected current (A), a float in range [0.0, inf], Default: 1.0
- **epsilon** (Float): Permitivity value (F/m), a float in range [0.0, inf], Default: 8.854187817620389e-12
- **length** (Float): length of the dipole (m), a float in range [0.0, inf], Default: 1.0
- **location** (Vector3): location of the electric dipole source, a 3D Vector of <class 'float'> with shape (3), Default: ZERO
- **mu** (Float): Magnetic permeability (H/m), a float in range [0.0, inf], Default: 1.2566370614359173e-06
- **orientation** (Vector3): orientation of dipole, a 3D Vector of <class 'float'> with shape (3), Default: X
- **sigma** (Float): Electrical conductivity (S/m), a float in range [0.0, inf], Default: 1.0

current
current (Float): magnitude of the injected current (A), a float in range [0.0, inf], Default: 1.0

length
length (Float): length of the dipole (m), a float in range [0.0, inf], Default: 1.0

class geoana.em.base.**BaseMagneticDipole**(***kwargs*)
Bases: *geoana.em.base.BaseDipole*

Base class for magnetic dipoles

Required Properties:

- **epsilon** (Float): Permitivity value (F/m), a float in range [0.0, inf], Default: 8.854187817620389e-12
- **location** (Vector3): location of the electric dipole source, a 3D Vector of <class 'float'> with shape (3), Default: ZERO
- **moment** (Float): moment of the dipole (Am^2), a float in range [0.0, inf], Default: 1.0
- **mu** (Float): Magnetic permeability (H/m), a float in range [0.0, inf], Default: 1.2566370614359173e-06
- **orientation** (Vector3): orientation of dipole, a 3D Vector of <class 'float'> with shape (3), Default: X
- **sigma** (Float): Electrical conductivity (S/m), a float in range [0.0, inf], Default: 1.0

moment
moment (Float): moment of the dipole (Am^2), a float in range [0.0, inf], Default: 1.0

5.3.2 Static

class geoana.em.static.**MagneticDipoleWholeSpace**(***kwargs*)
Bases: *geoana.em.base.BaseMagneticDipole*, *geoana.em.base.BaseEM*

Static magnetic dipole in a wholespace.

Required Properties:

- **epsilon** (Float): Permitivity value (F/m), a float in range [0.0, inf], Default: 8.854187817620389e-12
- **location** (Vector3): location of the electric dipole source, a 3D Vector of <class 'float'> with shape (3), Default: ZERO
- **moment** (Float): moment of the dipole (Am^2), a float in range [0.0, inf], Default: 1.0
- **mu** (Float): Magnetic permeability (H/m), a float in range [0.0, inf], Default: 1.2566370614359173e-06
- **orientation** (Vector3): orientation of dipole, a 3D Vector of <class 'float'> with shape (3), Default: X
- **sigma** (Float): Electrical conductivity (S/m), a float in range [0.0, inf], Default: 1.0

magnetic_field(*xyz*, *coordinates='cartesian'*)
Magnetic field ($\vec{h}$) of a static magnetic dipole

Required

Parameters **xyz** (*numpy.ndarray*) – Location of the receivers(s)

Optional

Parameters **coordinates** (*str*) – coordinate system that the xyz is provided in and that the solution will be returned in (cartesian or cylindrical). Default: *"cartesian"*

Returns

Return type numpy.ndarray

Returns The magnetic field at each observation location

magnetic_flux_density(*xyz*, *coordinates='cartesian'*)
Magnetic flux ($\vec{b}$) of a static magnetic dipole

Required

Parameters **xyz** (*numpy.ndarray*) – Location of the receivers(s)

Optional

Parameters **coordinates** (*str*) – coordinate system that the xyz is provided in and that the solution will be returned in (cartesian or cylindrical). Default: *"cartesian"*

Returns

Return type numpy.ndarray

Returns The magnetic flux at each observation location

vector_potential (*xyz*, *coordinates='cartesian'*)

Vector potential of a static magnetic dipole. See Griffiths, 1999 equation 5.83

$$\vec{A}(\vec{r}) = \frac{\mu_0}{4\pi}\frac{\vec{m} \times \vec{r}}{r^3}$$

Required

Parameters **xyz** (*numpy.ndarray*) – Location at which we calculate the vector potential

Optional

Parameters **coordinates** (*str*) – coordinate system that the xyz is provided in and that the solution will be returned in (cartesian or cylindrical). Default: *"cartesian"*

Returns

Return type numpy.ndarray

Returns The magnetic vector potential at each observation location

class geoana.em.static.**CircularLoopWholeSpace**(***kwargs*)

Bases: *geoana.em.base.BaseDipole*, *geoana.em.base.BaseEM*

Static magnetic field from a circular loop in a wholespace.

Required Properties:

- **current** (Float): Electric current through the loop (A), a float, Default: 1.0
- **epsilon** (Float): Permitivity value (F/m), a float in range [0.0, inf], Default: 8.854187817620389e-12
- **location** (Vector3): location of the electric dipole source, a 3D Vector of <class 'float'> with shape (3), Default: ZERO
- **mu** (Float): Magnetic permeability (H/m), a float in range [0.0, inf], Default: 1.2566370614359173e-06
- **orientation** (Vector3): orientation of dipole, a 3D Vector of <class 'float'> with shape (3), Default: X
- **radius** (Float): radius of the loop (m), a float in range [0.0, inf], Default: 1.0
- **sigma** (Float): Electrical conductivity (S/m), a float in range [0.0, inf], Default: 1.0

current

current (Float): Electric current through the loop (A), a float, Default: 1.0

radius

radius (Float): radius of the loop (m), a float in range [0.0, inf], Default: 1.0

vector_potential (*xyz*, *coordinates='cartesian'*)

Vector potential due to the a steady-state current through a circular loop. We solve in cylindrical coordinates

$$A_\theta(\rho, z) = \frac{\mu_0 I}{\pi k}\sqrt{R/\rho^2}[(1 - k^2/2) * K(k^2) - K(k^2)]$$

where

$$k^2 = \frac{4R\rho}{(R+\rho)^2 + z^2}$$

and

- $\rho = \sqrt{x^2 + y^2}$ is the horizontal distance to the test point
- r is the distance to a test point
- I is the current through the loop
- R is the radius of the loop
- $E(k^2)$ and $K(k^2)$ are the complete elliptic integrals

Required

Parameters **xyz** (*numpy.ndarray*) – Location where we calculate the vector potential

Optional

Parameters **coordinates** (*str*) – coordinate system that the xyz is provided in and that the solution will be returned in (cartesian or cylindrical). Default: *"cartesian"*

Returns

Return type numpy.ndarray

Returns The magnetic vector potential at each observation location

5.3.3 Frequency Domain

geoana.em.fdem.**omega**(*frequency*)

Angular frequency

$$\omega = 2\pi f$$

Required :param frequency float: frequency (Hz)

geoana.em.fdem.**wavenumber**(*frequency*, *sigma*, *mu=1.2566370614359173e-06*, *epsilon=8.854187817620389e-12*, *quasistatic=False*)

Wavenumber of an electromagnetic wave in a medium with constant physical properties

$$k = \sqrt{\omega^2 \mu \varepsilon - i\omega\mu\sigma}$$

Required

Parameters

- **numpy.ndarray) frequency** ((*float,*) – frequency (Hz)
- **sigma** (*float*) – electrical conductivity (S/m)

Optional

Parameters

- **mu** (*float*) – magnetic permeability (H/m). Default: $\mu_0 = 4\pi \times 10^{-7}$ H/m
- **epsilon** (*float*) – dielectric permittivity (F/m). Default: $\epsilon_0 = 8.85 \times 10^{-12}$ F/m
- **quasistatic** (*bool*) – use the quasi-static assumption? Default: False

geoana.em.fdem.**skin_depth**(*frequency*, *sigma*, *mu=1.2566370614359173e-06*)
Distance at which an em wave has decayed by a factor of $1/e$ in a medium with constant physical properties

$$\sqrt{\frac{2}{\omega\sigma\mu}}$$

Required

Parameters

- **frequency** (*float*) – frequency (Hz)
- **sigma** (*float*) – electrical conductivity (S/m)

Optional :param float mu: magnetic permeability (H/m). Default: $\mu_0 = 4\pi \times 10^{-7}$ H/m

geoana.em.fdem.**sigma_hat**(*frequency*, *sigma*, *epsilon=8.854187817620389e-12*, *quasistatic=False*)
conductivity with displacement current contribution

$$\hat{\sigma} = \sigma + i\omega\varepsilon$$

Required

Parameters

- **numpy.array) frequency** (*(float,*) – frequency (Hz)
- **sigma** (*float*) – electrical conductivity (S/m)

Optional

Parameters

- **epsilon** (*float*) – dielectric permittivity. Default ε_0
- **quasistatic** (*bool*) – use the quasi-static assumption? Default: False

class geoana.em.fdem.**BaseFDEM**(***kwargs*)
Bases: *geoana.em.base.BaseEM*

Base frequency domain electromagnetic class

Required Properties:

- **epsilon** (Float): Permitivity value (F/m), a float in range [0.0, inf], Default: 8.854187817620389e-12
- **frequency** (Float): Source frequency (Hz), a float in range [0.0, inf], Default: 1.0
- **mu** (Float): Magnetic permeability (H/m), a float in range [0.0, inf], Default: 1.2566370614359173e-06
- **quasistatic** (Boolean): Use the quasi-static approximation and ignore displacement current?, a boolean, Default: False
- **sigma** (Float): Electrical conductivity (S/m), a float in range [0.0, inf], Default: 1.0

frequency
frequency (Float): Source frequency (Hz), a float in range [0.0, inf], Default: 1.0

omega
Angular frequency

$$\omega = 2\pi f$$

quasistatic
quasistatic (Boolean): Use the quasi-static approximation and ignore displacement current?, a boolean, Default: False

sigma_hat
conductivity with displacement current contribution

$$\hat{\sigma} = \sigma + i\omega\varepsilon$$

skin_depth
Distance at which an em wave has decayed by a factor of $1/e$ in a medium with constant physical properties

$$\sqrt{\frac{2}{\omega\sigma\mu}}$$

wavenumber
Wavenumber of an electromagnetic wave in a medium with constant physical properties

$$k = \sqrt{\omega * *2\mu\varepsilon - i\omega\mu\sigma}$$

class geoana.em.fdem.**ElectricDipoleWholeSpace**(***kwargs*)
Bases: *geoana.em.base.BaseElectricDipole*, *geoana.em.fdem.BaseFDEM*

Harmonic electric dipole in a whole space. The source is (c.f. Ward and Hohmann, 1988 page 173). The source current density for a dipole located at $\mathbf{r}_s$ with orientation $\hat{\mathbf{u}}$

$$\mathbf{J}(\mathbf{r}) = Ids\delta(\mathbf{r} - \mathbf{r}_s)\hat{\mathbf{u}}$$

Required Properties:

- **current** (Float): magnitude of the injected current (A), a float in range [0.0, inf], Default: 1.0
- **epsilon** (Float): Permitivity value (F/m), a float in range [0.0, inf], Default: 8.854187817620389e-12
- **frequency** (Float): Source frequency (Hz), a float in range [0.0, inf], Default: 1.0
- **length** (Float): length of the dipole (m), a float in range [0.0, inf], Default: 1.0
- **location** (Vector3): location of the electric dipole source, a 3D Vector of <class 'float'> with shape (3), Default: ZERO
- **mu** (Float): Magnetic permeability (H/m), a float in range [0.0, inf], Default: 1.2566370614359173e-06
- **orientation** (Vector3): orientation of dipole, a 3D Vector of <class 'float'> with shape (3), Default: X
- **quasistatic** (Boolean): Use the quasi-static approximation and ignore displacement current?, a boolean, Default: False
- **sigma** (Float): Electrical conductivity (S/m), a float in range [0.0, inf], Default: 1.0

current_density(*xyz*)
Current density due to a harmonic electric dipole

electric_field(*xyz*)
Electric field from an electric dipole

$$\mathbf{E} = \frac{1}{\hat{\sigma}}\nabla\nabla\cdot\mathbf{A} - i\omega\mu\mathbf{A}$$

magnetic_field(*xyz*)
Magnetic field from an electric dipole

$$\mathbf{H} = \nabla\times\mathbf{A}$$

magnetic_flux_density(*xyz*)
magnetic flux density from an electric dipole

vector_potential(*xyz*)
Vector potential for an electric dipole in a wholespace

$$\mathbf{A} = \frac{Ids}{4\pi r} e^{-ikr} \hat{\mathbf{u}}$$

class geoana.em.fdem.**MagneticDipoleWholeSpace**(***kwargs*)
Bases: *geoana.em.base.BaseMagneticDipole*, *geoana.em.fdem.BaseFDEM*

Harmonic magnetic dipole in a whole space.

Required Properties:

- **epsilon** (Float): Permitivity value (F/m), a float in range [0.0, inf], Default: 8.854187817620389e-12
- **frequency** (Float): Source frequency (Hz), a float in range [0.0, inf], Default: 1.0
- **location** (Vector3): location of the electric dipole source, a 3D Vector of <class 'float'> with shape (3), Default: ZERO
- **moment** (Float): moment of the dipole (Am^2), a float in range [0.0, inf], Default: 1.0
- **mu** (Float): Magnetic permeability (H/m), a float in range [0.0, inf], Default: 1.2566370614359173e-06
- **orientation** (Vector3): orientation of dipole, a 3D Vector of <class 'float'> with shape (3), Default: X
- **quasistatic** (Boolean): Use the quasi-static approximation and ignore displacement current?, a boolean, Default: False
- **sigma** (Float): Electrical conductivity (S/m), a float in range [0.0, inf], Default: 1.0

current_density(*xyz*)
Current density from a magnetic dipole in a wholespace

electric_field(*xyz*)
Electric field from a magnetic dipole in a wholespace

magnetic_field(*xyz*)
Magnetic field due to a magnetic dipole in a wholespace

magnetic_flux_density(*xyz*)
Magnetic flux density due to a magnetic dipole in a wholespace

vector_potential(*xyz*)
Vector potential for a magnetic dipole in a wholespace

$$\mathbf{F} = \frac{i\omega\mu m}{4\pi r} e^{-ikr} \hat{\mathbf{u}}$$

5.3.4 Time Domain

class geoana.em.tdem.**BaseTDEM**(***kwargs*)
Bases: *geoana.em.base.BaseEM*

Required Properties:

- **epsilon** (Float): Permitivity value (F/m), a float in range [0.0, inf], Default: 8.854187817620389e-12
- **mu** (Float): Magnetic permeability (H/m), a float in range [0.0, inf], Default: 1.2566370614359173e-06
- **sigma** (Float): Electrical conductivity (S/m), a float in range [0.0, inf], Default: 1.0

- **time** (Float): time after shut-off at which we are evaluating the fields (s), a float, Default: 0.0001

diffusion_distance

peak_time(*z*)

theta

time
time (Float): time after shut-off at which we are evaluating the fields (s), a float, Default: 0.0001

class geoana.em.tdem.**ElectricDipoleWholeSpace**(***kwargs*)
Bases: *geoana.em.base.BaseElectricDipole*, *geoana.em.tdem.BaseTDEM*

Harmonic electric dipole in a whole space. The source is (c.f. Ward and Hohmann, 1988 page 173). The source current density for a dipole located at $\mathbf{r}_s$ with orientation $\hat{\mathbf{u}}$

$$\mathbf{J}(\mathbf{r}) = Ids\delta(\mathbf{r} - \mathbf{r}_s)\hat{\mathbf{u}}$$

Required Properties:

- **current** (Float): magnitude of the injected current (A), a float in range [0.0, inf], Default: 1.0
- **epsilon** (Float): Permitivity value (F/m), a float in range [0.0, inf], Default: 8.854187817620389e-12
- **length** (Float): length of the dipole (m), a float in range [0.0, inf], Default: 1.0
- **location** (Vector3): location of the electric dipole source, a 3D Vector of <class 'float'> with shape (3), Default: ZERO
- **mu** (Float): Magnetic permeability (H/m), a float in range [0.0, inf], Default: 1.2566370614359173e-06
- **orientation** (Vector3): orientation of dipole, a 3D Vector of <class 'float'> with shape (3), Default: X
- **sigma** (Float): Electrical conductivity (S/m), a float in range [0.0, inf], Default: 1.0
- **time** (Float): time after shut-off at which we are evaluating the fields (s), a float, Default: 0.0001

current_density(*xyz*)
Current density due to a harmonic electric dipole

electric_field(*xyz*)
Electric field from an electric dipole

$$\mathbf{E} = \frac{1}{\hat{\sigma}}\nabla\nabla\cdot\mathbf{A} - i\omega\mu\mathbf{A}$$

magnetic_field(*xyz*)
Magnetic field from an electric dipole

magnetic_field_time_deriv(*xyz*)
Time derivative of the magnetic field, $\frac{\partial \mathbf{h}}{\partial t}$

magnetic_flux_density(*xyz*)
Magnetic flux density from an electric dipole

magnetic_flux_density_time_deriv(*xyz*)
Time derivative of the magnetic flux density from an electric dipole

geoana.em.tdem.**diffusion_distance**(*time*, *sigma*, *mu=1.2566370614359173e-06*)
Diffusion distance: Distance at which the signal amplitude is largest for a given time after shut off. Also referred to as the peak distance

geoana.em.tdem.**peak_time**(*z*, *sigma*, *mu=1.2566370614359173e-06*)

Peak time: Time at which the maximum signal amplitude is observed at a particular location for a transient plane wave through a homogeneous medium.

Required

Parameters

- **z** (*float*) – distance from source (m)
- **sigma** (*float*) – electrical conductivity (S/m)

Optional

Parameters **mu** (*float*) – magnetic permeability (H/m). Default: $\mu_0 = 4\pi \times 10^{-7}$ H/m

geoana.em.tdem.**theta**(*time*, *sigma*, *mu=1.2566370614359173e-06*)

Analog to wavenumber in the frequency domain. See Ward and Hohmann, 1988 pages 174-175

5.4 Earthquake

oksar3

Program to calcuate forward models of interferograms, strain tensor, etc. from Okada subroutine.

Heritage:

- originally fringes.c written by Barry Parsons
- updated to oksar tjw
- oksar_strain: added strain tensor calculation tjw
- oksar3: added new line of sight calculator tjw feb 2003
- Modified into Python by RowanCockett, 3point Science Aug 2014

class geoana.earthquake.oksar.**EarthquakeInterferogram**(***kwargs*)

Bases: properties.base.base.HasProperties

Required Properties:

- **copyright** (String): copyright, a unicode string
- **data** (Array): Processed interferogram data (unwrapped), a list or numpy array of <class 'float'> with shape (*)
- **data_source** (String): data_source, a unicode string
- **date1** (DateTime): date1, a datetime object
- **date2** (DateTime): date2, a datetime object
- **event_country** (String): Earthquake country, a unicode string
- **event_date** (DateTime): Date of the earthquake, a datetime object
- **event_gcmt_id** (String): GCMT ID, a unicode string
- **event_name** (String): Earthquake name, a unicode string
- **local_earth_radius** (Float): Earth radius, a float, Default: 6371000.0
- **local_rigidity** (Float): Local rigidity, a float, Default: 30000000000.0

- **location** (Vector2): interferogram location (bottom N, left E), a 2D Vector of <class 'float'> with shape (2)
- **location_UTM_zone** (Integer): UTM zone, an integer
- **pixel_size** (Array): Size of each pixel (northing, easting), a list or numpy array of <class 'float'> with shape (2)
- **processed_by** (String): processed_by, a unicode string
- **processed_date** (DateTime): processed_date, a datetime object
- **ref** (Vector2): interferogram reference, a 2D Vector of <class 'float'> with shape (2)
- **ref_incidence** (Float): Incidence angle, a float
- **satellite_altitude** (Float): satellite_altitude, a float
- **satellite_azimuth** (Float): satellite_azimuth, a float
- **satellite_fringe_interval** (Float): Fringe interval, a float, Default: 0.028333
- **satellite_name** (String): Name of the satelite., a unicode string
- **scaling** (Float): Scaling of the interferogram, a float, Default: 1.0
- **shape** (Array): number of pixels in the interferogram, a list or numpy array of <class 'int'> with shape (2)
- **title** (String): name of the earthquake, a unicode string

Optional Properties:

- **description** (String): description of the event, a unicode string

copyright
copyright (String): copyright, a unicode string

data
data (Array): Processed interferogram data (unwrapped), a list or numpy array of <class 'float'> with shape (*)

data_source
data_source (String): data_source, a unicode string

date1
date1 (DateTime): date1, a datetime object

date2
date2 (DateTime): date2, a datetime object

description
description (String): description of the event, a unicode string

event_country
event_country (String): Earthquake country, a unicode string

event_date
event_date (DateTime): Date of the earthquake, a datetime object

event_gcmt_id
event_gcmt_id (String): GCMT ID, a unicode string

event_name
event_name (String): Earthquake name, a unicode string

get_LOS_vector(*locations*)
calculate beta - the angle at earth center between reference point and satellite nadir

local_earth_radius
local_earth_radius (Float): Earth radius, a float, Default: 6371000.0

local_rigidity
local_rigidity (Float): Local rigidity, a float, Default: 30000000000.0

location
location (Vector2): interferogram location (bottom N, left E), a 2D Vector of <class 'float'> with shape (2)

location_UTM_zone
location_UTM_zone (Integer): UTM zone, an integer

pixel_size
pixel_size (Array): Size of each pixel (northing, easting), a list or numpy array of <class 'float'> with shape (2)

plot_interferogram(*wrap=True*, *ax=None*)

plot_mask(*ax=None*, *opacity=0.2*)

processed_by
processed_by (String): processed_by, a unicode string

processed_date
processed_date (DateTime): processed_date, a datetime object

ref
ref (Vector2): interferogram reference, a 2D Vector of <class 'float'> with shape (2)

ref_incidence
ref_incidence (Float): Incidence angle, a float

satellite_altitude
satellite_altitude (Float): satellite_altitude, a float

satellite_azimuth
satellite_azimuth (Float): satellite_azimuth, a float

satellite_fringe_interval
satellite_fringe_interval (Float): Fringe interval, a float, Default: 0.028333

satellite_name
satellite_name (String): Name of the satelite., a unicode string

scaling
scaling (Float): Scaling of the interferogram, a float, Default: 1.0

shape
shape (Array): number of pixels in the interferogram, a list or numpy array of <class 'int'> with shape (2)

title
title (String): name of the earthquake, a unicode string

class geoana.earthquake.oksar.**Oksar**(***kwargs*)
Bases: properties.base.base.HasProperties

Required Properties:

- **O** (Vector2): Origin of the simulation domain, a 2D Vector of <class 'float'> with shape (2)

- **U** (Vector2): U direction of the simulation domain, a 2D Vector of <class 'float'> with shape (2)
- **V** (Vector2): V direction of the simulation domain, a 2D Vector of <class 'float'> with shape (2)
- **beta** (Float): beta, a float, Default: 30000000000.0
- **center** (Vector2): Center of the fault plane., a 2D Vector of <class 'float'> with shape (2)
- **depth_bottom** (Float): Bottom of fault, a float in range [0, inf], Default: 10000
- **depth_top** (Float): Top of fault, a float in range [0, inf]
- **dip** (Float): Dip, a float in range [0, 90], Default: 45
- **length** (Float): Fault length, a float in range [0, inf], Default: 10000.0
- **mu** (Float): mu, a float, Default: 30000000000.0
- **rake** (Float): Rake, a float in range [-180, 180], Default: 90
- **shape** (Array): number of pixels in the simulation, a list or numpy array of <class 'int'> with shape (2), Default: (300, 300)
- **slip** (Float): Slip, a float in range [0, inf], Default: 0.5
- **strike** (Float): Strike, a float in range [0, 360]

O
: **O** (Vector2): Origin of the simulation domain, a 2D Vector of <class 'float'> with shape (2)

U
: **U** (Vector2): U direction of the simulation domain, a 2D Vector of <class 'float'> with shape (2)

V
: **V** (Vector2): V direction of the simulation domain, a 2D Vector of <class 'float'> with shape (2)

beta
: **beta** (Float): beta, a float, Default: 30000000000.0

center
: **center** (Vector2): Center of the fault plane., a 2D Vector of <class 'float'> with shape (2)

depth_bottom
: **depth_bottom** (Float): Bottom of fault, a float in range [0, inf], Default: 10000

depth_top
: **depth_top** (Float): Top of fault, a float in range [0, inf]

dip
: **dip** (Float): Dip, a float in range [0, 90], Default: 45

displacement_vector

length
: **length** (Float): Fault length, a float in range [0, inf], Default: 10000.0

mu
: **mu** (Float): mu, a float, Default: 30000000000.0

plot_displacement(*eq=None*, *ax=None*, *wrap=True*, *mask_opacity=0.2*)

rake
: **rake** (Float): Rake, a float in range [-180, 180], Default: 90

shape
: **shape** (Array): number of pixels in the simulation, a list or numpy array of <class 'int'> with shape (2), Default: (300, 300)

simulation_grid

slip
slip (Float): Slip, a float in range [0, inf], Default: 0.5

strike
strike (Float): Strike, a float in range [0, 360]

geoana.earthquake.oksar.**example**()

5.5 Utilities

5.5.1 Core Utils

geoana.utils.**mkvc**(*x*, *numDims=1*)
Creates a vector with the number of dimension specified

e.g.:

```
a = np.array([1, 2, 3])

mkvc(a, 1).shape
    > (3, )

mkvc(a, 2).shape
    > (3, 1)

mkvc(a, 3).shape
    > (3, 1, 1)
```

geoana.utils.**ndgrid**(**args*, ***kwargs*)
Form tensorial grid for 1, 2, or 3 dimensions.

Returns as column vectors by default.

To return as matrix input:

ndgrid(. . . , vector=False)

The inputs can be a list or separate arguments.

e.g.:

```
a = np.array([1, 2, 3])
b = np.array([1, 2])

XY = ndgrid(a, b)
    > [[1 1]
       [2 1]
       [3 1]
       [1 2]
       [2 2]
       [3 2]]

X, Y = ndgrid(a, b, vector=False)
    > X = [[1 1]
           [2 2]
           [3 3]]
```

(continues on next page)

(continued from previous page)

```
> Y = [[1 2]
       [1 2]
       [1 2]]
```

5.5.2 Spatial

geoana.spatial.**cartesian_2_cylindrical**(*grid*, *vec=None*)

Takes a grid or vector (if provided) defined in cartesian coordinates (x, y, z) and transform it to cylindrical coordinates, (r, θ, z).

Required

Parameters **grid** (*numpy.ndarray*) – grid in cartesian coordinates (x, y, z)

Optional

Parameters **vec** (*numpy.ndarray*) – (optional) vector defined in cartesian coordinates

Returns

Returns grid or vector (if provided) in cylindrical coordinates (r, θ, z)

Return type numpy.ndarray

geoana.spatial.**cartesian_2_spherical**(*grid*, *vec=None*)

Takes a grid or vector (if provided) defined in cartesian coordinates (x, y, z) and transform it to spherical coordinates, (r, θ, ϕ).

Required

Parameters **grid** (*numpy.ndarray*) – grid in cartesian coordinates (x, y, z)

Optional

Parameters **vec** (*numpy.ndarray*) – (optional) vector defined in cartesian coordinates

Returns

Returns grid or vector (if provided) in spherical coordinates (r, θ, ϕ)

Return type numpy.ndarray

geoana.spatial.**cylindrical_2_cartesian**(*grid*, *vec=None*)

Take a grid or vector (if provided) defined in cylindrical coordinates (r, θ, z) and transform it to cartesian coordinates, (x, y, z).

Required

Parameters **grid** (*numpy.ndarray*) – grid in cylindrical coordinates (r, θ, z)

Optional

Parameters **vec** (*numpy.ndarray*) – (optional) vector defined in cylindrical coordinates

Returns

Returns grid or vector (if provided) in cartesian coordinates (x, y, z)

Return type numpy.ndarray

geoana.spatial.**distance**(*xyz*, *origin=array([0., 0., 0.])*)

Radial distance from an grid of points to the origin

Required

Parameters **xyz** (*numpy.ndarray*) – grid (npoints x 3)

Optional

Parameters **origin** (*numpy.ndarray*) – origin (default: [0., 0., 0.])

Returns

Returns distance between each point and the origin (npoints x 1)

Return type numpy.ndarray

geoana.spatial.**repeat_scalar**(*scalar*, *dim=3*)
Repeat a spatially distributed scalar value dim times to simplify multiplication with a vector.

Required

Parameters **scalar** (*numpy.ndarray*) – (n x 1) array of scalars

Optional

Parameters **dim** (*int*) – dimension of the second axis for the output (default = 3)

Returns

Returns (n x dim) array of the repeated vector

Return type numpy.ndarray

geoana.spatial.**rotate_points_from_normals**(*xyz*, *n0*, *n1*, *x0=array([0., 0., 0.])*)
rotates a grid so that the vector n0 is aligned with the vector n1

Required

Parameters

- **xyz** (*numpy.ndarray*) –
- **n0** (*numpy.ndarray*) – vector of length 3, should have norm 1
- **n1** (*numpy.ndarray*) – vector of length 3, should have norm 1

Optional

Parameters **x0** (*numpy.ndarray*) – vector of length 3, point about which we perform the rotation

Returns

Return type numpy.ndarray

Returns (3x3) rotation matrix which rotates the frame so that n0 is aligned with n1

geoana.spatial.**rotation_matrix_from_normals**(*v0*, *v1*, *tol=1e-20*)
Performs the minimum number of rotations to define a rotation from the direction indicated by the vector n0 to the direction indicated by n1. The axis of rotation is n0 x n1 https://en.wikipedia.org/wiki/Rodrigues%27_rotation_formula

Parameters

- **v0** (*numpy.ndarray*) – vector of length 3
- **v1** (*numpy.ndarray*) – vector of length 3
- **tol** (*float*) – tolerance. If the norm of the cross product between the two vectors is below this, no rotation is performed default = 1e-20

Return type numpy.ndarray

Returns 3 x 3 rotation matrix which rotates the frame so that n0 is aligned with n1

geoana.spatial.**spherical_2_cartesian**(*grid*, *vec=None*)

Take a grid or vector (if provided) defined in spherical coordinates (r, θ, ϕ) and transform it to cartesian coordinates, (x, y, z).

Required

Parameters **grid** (*numpy.ndarray*) – grid in spherical coordinates (r, θ, ϕ)

Optional

Parameters **vec** (*numpy.ndarray*) – (optional) vector defined in spherical coordinates

Returns

Returns grid or vector (if provided) in cartesian coordinates (x, y, z)

Return type numpy.ndarray

geoana.spatial.**vector_distance**(*xyz*, *origin=array([0., 0., 0.])*)

Vector distance of a grid, xyz from an origin origin.

Required

Parameters **xyz** (*numpy.ndarray*) – grid (npoints x 3)

Optional

Parameters **origin** (*numpy.ndarray*) – origin (default: [0., 0., 0.])

Returns

Returns vector distance from a grid of points from the origin (npoints x 3)

Return type numpy.ndarray

geoana.spatial.**vector_dot**(*xyz*, *vector*)

Take a dot product between an array of vectors, xyz and a vector [x, y, z]

Required

Parameters

- **xyz** (*numpy.ndarray*) – grid (npoints x 3)
- **vector** (*numpy.ndarray*) – vector (1 x 3)

Returns

Returns dot product between the grid and the (1 x 3) vector, returns an (npoints x 1) array

Return type numpy.ndarray

geoana.spatial.**vector_magnitude**(*v*)

Amplitude of a vector, v.

Required

Parameters **v** (*numpy.ndarray*) – vector array

Returns

Returns magnitude of a vector (n, 1)

Return type numpy.ndarray

Python Module Index

g

Index

G

L

M

N

O

P

Q

R

S

T

U

V

W