SAnTex - Seismic Anisotropy from Texture:
A Python-based library for Seismic Anisotropy Calculation

User Guide

Utpal Singh[1], Sinan Özaydın[1], Vasileios Chatzaras[1], Patrice Rey[1]

[1]The University of Sydney, School of Geosciences, Sydney, NSW, Australia

SAnTex Installation

Open the terminal and go to Downloads or Documents directory by typing *cd ~/Documents* or *cd ~/Downloads* or any other directory, where you can store the repository of SAnTex. Run the following to get the updated version of SAnTex

```
git clone https://github.com/utpal-singh/SAnTex.git
cd santex
pip install .
```

1.0 Working with SAnTex

In a python working environment, import SAnTex's modules and Python libraries:

```
from santex import EBSD
from santex import Tensor
from santex import Material
from santex import Isotropy
import pandas as pd
```

NB: In what follows, user-defined names are in blue, e.g., `ebsd.ctf`

Load the EBSD file:

```
ebsdfile = EBSD("ebsd.ctf")
```

*Note: Remember to change the parentheses in the python code, the Word document modifies the parentheses.

Commented [1]: I don't understand what you mean here

This loads the *ebsd.ctf* file in the python object *ebsdfile* of class *EBSD*, from which users are able to call methods for further processing, cleaning of the ebsd data, and ultimately calculate elastic properties.

*Class: In Python, a class is a blueprint for creating objects. It defines the structure and behavior of objects of that type. Think of it as a template or a prototype that encapsulates data (attributes) and methods (functions) that operate on that data. Object: An object, also known as an instance, is a unique entity that is created based on the structure defined by its class. When an object is instantiated, it inherits the attributes and methods defined in its class, but can also have its own unique state and behavior.

Note: the *ebsd.ctf* should be located in the directory this notebook is being run from. But you may provide a relative path like "*data/ebsd.ctf*" or an absolute import such as "*/Users/myname/Documents/ebsd/ebsd.ctf*"

1.1 Available methods in the EBSD class

To list the phases present in the ebsd file:

```
phases_available = ebsdfile.phases()
print(phases_available)
```

Load the ebsd file into a pandas dataframe, via invoking the `get_ebsd_data()` method from the *EBSD* class: Note: this dataframe df is critical as it holds information of all the euler angles

```
df = ebsdfile.get_ebsd_data()
print(df)
```

2.6 Plotting Conventions

Following are the keywords to orient sample reference plane and to store in a new dataframe

The default is `sample_ref = ["x2east", "zOutOfPlane"]`

Following are other available keywords:

```
sample_ref == ["x2east", "zOutOfPlane"]
sample_ref == ["x2west", "zOutOfPlane"]
sample_ref == ["x2north", "zOutOfPlane"]
sample_ref == ["x2south", "zOutOfPlane"]
```

Commented [2]: Change

Commented [3]: Is "Import pandas" missing?

Commented [4]: No, the import pandas isn't missing here, but its good idea to also do import pandas if the user needs to look at some statistics from the dataframe which I have not built internally. The ctf file gets a little complicated, so I have handled the pandas dataframe internally. So, when you call from sage import xxx, pandas inherently gets imported

Commented [5]: Provide explanation of how rotation angle and trie and false mean. Preferably with a figure

Preserve aspect ratio
Inside plane to be implemented

```
sample_ref == ["x2east", "zIntoPlane"]
sample_ref == ["x2west", "zIntoPlane"]
sample_ref == ["x2north", "zIntoPlane"]
sample_ref == ["x2south", "zIntoPlane"]
```

Lets now rotate the ebsd dataframe to x2east and zIntoPlane

```
rotated_df = ebsdfile.plot_rotate_ebsd(sample_ref = ["x2east",
"zIntoPlane"], ebsd_df = df)
ebsdfile.plot(rotated_df)
```

To rotate the EBSD data to match the SEM orientation in any custom angles

```
angles = (180, 0, 0)
updatedebsd = ebsdfile.rotateEBSD(ebsdfile, angles)
```

This invokes the *phases* method of the *EBSD* class and returns a dictionary with percentage abundances into a variable called *phases_available*, and then further prints the available phases and the percentage abundance within the provided ebsd file.

The above directive loads the *ebsd* dataframe into a variable called *df*, and then prints all of the pixel information, with their corresponding phases numbered from 0 to n, (n means the number of phases identified from the ebsd file), their bunge-euler angles, band contrast (BC), band slope (BS), as well as mean angular deviation (MAD), a measure of how accurate the solution assigned to each EBSD pattern is.

Commented [6]: Add a note that o is not-indexed etc

Information on the dataframe and the operations that can be performed can be found in the official documentation of the pandas https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html

*A dataframe is like a python based object for Excel based calculations, the dataframe stores tabular forms of data in which many operations can be done.

To print the header information, which includes the acquisition details of the ebsd file, first load the ebsd data header into a dataframe variable here called *df_header*:

```
df_header = ebsdfile.get_ebsd_data_header()
print(df_header)
```

To rotate the EBSD data to match the SEM orientation

```
angles = (180, 0, 0)
ebsdfile_downsample = ebsdfile.downsampleEBSD()
updatedebsd = ebsdfile.rotateEBSD(ebsdfile_downsample, angles)
```

Commented [7]: Following rotation I want to plot the EBSD map (e.g., phase map) and the CPO (plot either all orientations or 1ppg).

2.0 Cleaning EBSD data

To clean the EBSD dataset we successively remove grains with large mean angular deviation (MAD), reconstruct grains with grain boundaries misorientation ≥ 10 degrees, and remove grains smaller than 7 pixels.

2.1 Listing of the phases

Firstly, derive the phase names from the ebsd file using:

```
from santex import EBSD
ebsdfile = EBSD("ebsd.ctf")
phases_names = ebsdfile.phases_names()
print(phases_names)
```

Get the EBSD data in a dataframe

```
df = ebsd.get_ebsd_data()
```

2.2 Remove unwanted phases

Let's assume we want to remove phases whose index is [4, 5, 6, 7]

```
df = ebsd.filterByPhaseNumberList(df = df, phase_list = [4, 5, 6, 7])
```

2.3 Remove pixel data with MAD higher than a user specified value

To remove pixels with mean angular deviation (MAD) > e.g., 0.8, and store the cleaned dataset into a new dataframe called *filtered_df:*

```
filtered_df = ebsdfile.filterMAD(df, 0.8)
print(filtered_df)
```

2.4 Reconstruct grains with boundaries misorientation

Reconstruct grains with misorientation ≥ 10 degrees and store the cleaner dataset into a new dataframe called *df_grain_boundary:*

```
df_grain_boundary = ebsdfile.calcGrains(df = filtered_
df, threshold = 10, phase_names=phases_names,
downsampling_factor=100)
```

Commented [8]: Why downsampling? I think the actual code would not inclue any downsampling, but you could include an example in which downsampling is included as an option for faster calculation.

2.5 Remove small grains

Remove tiny grains smaller than e.g. 7 pixels, and store the cleaner dataset into a new dataframe called *filtered_df_grain_boundary*:

```
filtered_df_grain_boundary =
ebsdfile.filterByGrainSize(df_grain_boundary, phases_names,
min_grain_size=7)
```

2.6 Compare original and clean datasets

To compare the original ebsd dataset (*ebsdfile*) and the clean dataset (*filtered_df_grain_boundary*) users can plot them:

```
ebsdfile.plot()
ebsdfile.plot(df = filtered_df_grain_boundary)
```

2.7 Plotting Conventions

Commented [9]: Provide explanation of how rotation angle and trie and false mean. Preferably with a figure

Preserve aspect ratio
Inside plane to be implemented

```
ebsdfile.plot(data = ebsdfile.downsampleEBSD(),
rotation_angle=90, inside_plane=False)
```

2.8 Sample reference alignment

Following directive loads the EBSD data with x to east, y to north and z out of plane

ebsdfile.plot(df)

2.9 Following are the options available for plot

```
    def plot(self, data=None, rotation_angle=0, inside_plane=True,
mirror=False, save_image=False, image_filename=None, dpi=300,
cmap='viridis', legend_location="upper right"):
```

legend_location can be 'upper right', 'upper left', 'lower right', 'lower left'. Default is 'upper right'
save_image if set to True, you can give an image_filename = “image.png” to save the plot

2.10 ODF, PDF and IPF analysis

```
ebsdfile.odf(df, phase=1, crystal_symmetry='D2', random_val=True,
miller=[1, 0, 0], hemisphere = 'both', axes_labels=["Xs", "Ys"],
alpha = 0.01, figure = None, vector_labels = None, reproject=False,
show_hemisphere_label = None,
       grid = None, grid_resolution = None, return_figure = None
)

ebsdfile.pdf(df, phase=1, crystal_symmetry='D2', random_val=True,
miller=[1, 0, 0], hemisphere = 'both',
       axes_labels=["Xs", "Ys"], alpha = 0.01, figure = None,
vector_labels = None, reproject=False, show_hemisphere_label = None,
       grid = None, grid_resolution = None, return_figure = None
)

ebsdfile.ipf(df, phase=1, vector_sample=[0, 0, 1], random_val=True,
           vector_title='Z', projection='ipf', crystal_symmetry='D2'
)
```

3.0 Calculation and visualisation of seismic anisotropies

3.1 Prepare Dataframes

```
from santex import Material
```

Create a material instance from material class:

```
material_instance = Material()
phase = 'Diopside'
voigtMatrix = material_instance.voigthighPT(phase, PRESSURE =
3, TEMP = 1000)
```

Alternatively, if we want standard reference tensors:

```
voigtMatrix = material_instance.get_voigt_matrix(phase)
```

For example, if we want to get stiffness tensors and densities for Forsterite, diopside and enstatite, we write:

```
cij_Forsterite = material_instance.voigthighPT('Forsterite',
PRESSURE = 3, TEMP = 1000)
cij_Enstatite = material_instance.voigthighPT('Enstatite', PRESSURE
= 3, TEMP = 1000)
cij_Diopside = material_instance.voigthighPT('Diopside', PRESSURE =
3, TEMP = 1000)
```

```
density_Forsterite = material_instance.load_density("Forsterite")
density_Enstatite = material_instance.load_density("Enstatite")
density_Diopside = material_instance.load_density("Diopside")
```

Similarly we can invoke print keyword to look at the cij_Forsterite.

Store the densities in the list format for both cij and density. This will assemble all the stiffness tensors and densities into one variable.

```
cij = [cij_Forsterite, cij_Enstatite, cij_Diopside]
density = [density_Forsterite, density_Enstatite, density_Diopside]
```

3.2 Calculation of Anisotropy from EBSD File

Get all the euler angles for each phases and store them in a variable called euler_angles:

```
forsterite = ebsdfile.get_euler_angles(phase = 1, data =
filtered_df_grain_boundary)
enstatite = ebsdfile.get_euler_angles(phase = 2, data =
filtered_df_grain_boundary)
diopside = ebsdfile.get_euler_angles(phase = 3, data =
filtered_df_grain_boundary)

euler_angles = [forsterite, enstatite, diopside]
```

To calculate the average tensor and density:

```
from santex import Anisotropy
from santex import EBSD

average_tensor, average_density = ebsdfile.getAnisotropyForEBSD(cij
= cij, euler_angles= euler_angles, density=density)
```

The average_tensor and average_density is calculated internally, where we can parse this as input to SAnTex's Anisotropy class, and then we can generate the anisotropy plot for the EBSD file

```
anis = Anisotropy(average_tensor*10**9, average_density)
anis.plot()
```

If you want to include melt in the calculation,the modification will be

```
average_tensor, y, average_density =
ebsdfile.getAnisotropyForEBSD(cij = cij, euler_angles= euler_angles,
density=density, melt = 2)
```

3.3 Anisotropy Calculations and Visualisation

The SAnTex module *Anisotropy* requires a stiffness tensors (here *stiffness_matrix*) as an numpy array, and a density in kg.m^{-3} and returns an instance of the Anisotropy class from which seismic anisotropy is calculated.

```
from santex import Anisotropy
import numpy as np

    stiffness_matrix = np.array([[198.96, 73.595, 68.185, 0., 9.735, 0.],
                                 [73.595, 155.94, 62.23, 0., 6.295, 0.],
                                 [68.185, 62.23, 225.99, 0., 33.85, 0.],
                                 [0., 0., 0., 65.66, 0., 6.415],
                                 [9.735, 6.295, 33.85, 0., 60.23, 0.],
                                 [0., 0., 0., 6.415, 0., 65.18]]) * 10**9

    density = 3500

    # Create an instance of the Anisotropy class
    anisotropy_instance = Anisotropy(stiffness_matrix, density)
    anisotropy_instance.plot()
```

3.4 Anisotropy Calculations for Modal Rock

```
from santex import ModalAnisotropy
modal_rock = {'diopside : 0.5', 'forsterite : 0.3',
'enstatite' : 0.2}
anisotropy_instance = AnisotropyModal(modal_rock, pressure =
2, temperature = 1000)
anisotropy_instance.plot()
```

3.5 3-D Visualisations

To visualise the seismic anisotropies vp, vs1, vs2, and shear wave splitting one calls the anisotropy_instance.plot():

For vp

```
anisotropy_instance.plotter_vp(density, stiffness_matrix)
```

For vs1:

```
anisotropy_instance.plotter_vs1(density, stiffness_matrix)
```

For vs2:

```
anisotropy_instance.plotter_vs2(density, stiffness_matrix)
```

For vs splitting:

```
anisotropy_instance.plotter_vs_splitting(density, stiffness_matrix)
```

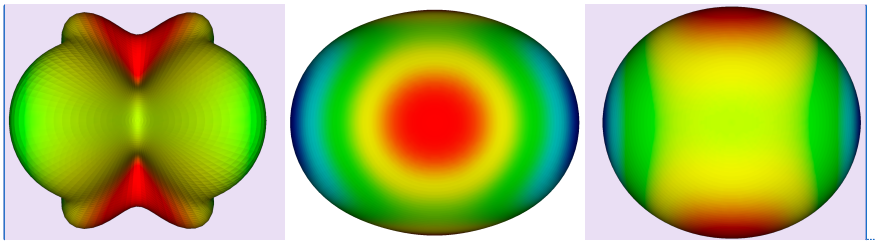

Figure 1: (a) Olivine shear wave splitting (b) Olivine Vp (c) Olivine Vs1

Commented [10]: The color scheme is a bit odd. I would have chosen red for fast, and dark blue for slow. Also the wavefronts for Vp and Vs1 looks the same. Should Vp be a little more oblate?

Commented [11]: yes, thats correct. I will change the color code

3.6 Plot velocity Grid

```
plot_velocities(self, pressure_range, temperature_range, return_type, is_ebsd = False, phase = None, grid = [5, 5], filename = None, *args)
```

e.g.: `pressure_range = [2, 2.5], temperature_range = [1000, 2000]`

```
x = anisotropy_instance.plot_velocities([2, 2.5], [1000, 1200], grid = [3, 3], phase = "Forsterite", return_type="maxvp")
```

return_type can be ‘maxvp’, ‘minvp’, ‘maxvs1’, ‘minvs1’, ‘maxvs2’, ‘minvs2’, ‘max_vs_anisotropy_percent’, ‘min_vs_anisotropy_percent’, ‘p_wave_anisotropy_percent’, ‘s1_wave_anisotropy_percent’, ‘s2_wave_anisotropy_percent’, ‘maxdvs’, ‘AVpVs1

4. Tensor Analysis

There are two notations, the tensor notation and voigt notation. We can rotate the tensor with any given phi1, phi and phi2 to obtain the rotated tensor. For that, we need to first import Tensor module

```
import numpy as np
import pandas as pd

from santex import Tensor
```

Lets define some tensor voigt form which is a 6*6 form

```
cij_forsterite = np.array([[320.5,  68.15,  71.6,     0,     0,     0],
  [ 68.15,  196.5,  76.8,     0,     0,     0],
  [  71.6,   76.8, 233.5,     0,     0,     0],
  [   0,       0,      0,    64,     0,     0],
  [   0,       0,      0,     0,    77,     0],
  [   0,       0,      0,     0,     0,  78.7]])
```

To instantiate the Tensor class,

```
tensor = Tensor()
```

Getting the stiffness tensor (3*3*3*3)

```
cijkl_forsterite = tensor.voigt_to_tensor(cij_forsterite)
```

Rotate Tensors

```
alpha = 10
beta = 20
gamma = 30 # in degrees

rotated_forsterite = tensor.rotate_tensor(cijkl_forsterite, alpha,
beta, gamma)

voigt_rotated_forsterite = tensor.tensor_to_voigt(rotated_forsterite)
```

5. Isotropic Velocities Analysis

Importing the module and from santex and defining isotropy instance

```
from santex import Isotropy
isotropy = Isotropy()
```

5.1 Check available phases

```
isotropy.get_available_phases()
```

5. 2 Get phase constants for eg Forsterite

Following are the information we get when we invoke the method get_phase_constants()

1. rho0: initial density
2. ao: coefficient of thermal expansion
3. akt0: isothermal bulk modulus, which is a measure of a material's resistance to compression under uniform pressure
4. dkdp: pressure derivative of the bulk modulus, indicating how the bulk modulus changes with pressure

5. amu0: shear modulus of the mineral. The shear modulus measures a material's resistance to deformation by shear stress
6. dmudp: pressure derivative of the shear modulus, indicating how the shear modulus changes with pressure
7. gam: gamma, first thermodynamic Gruinesen parameter
8. grun: second Gruneisen parameter, which is a measure of how a material's volume changes with temperature
9. delt: Debye temperature, which is a measure of the average vibrational energy of atoms in a solid.

```
isotropy.get_phase_constants("Forsterite")
```

5. 3 Get velocities and densities

We can get the following quantities at any given temperature and pressure for a material after invoking the method calculate_seismic_properties()

1. density: density of material at any given pressure and temperature
2. aks: bulk modulus, The bulk modulus indicates how much a material will compress under pressure.
3. amu: Shear Modulus, The shear modulus is essential for understanding a material's response to shear stress
4. vp: P-wave velocity at any given pressure and temperature
5. vs: swave velocity at any given pressure and temperature
6. vbulk: Bulk sound velocity, The velocity of sound waves traveling through a material
7. akt: Isothermal bulk modulus, Similar to the bulk modulus, but specifically refers to the resistance to compression under constant 8. temperature conditions

```
density, aks, amu, vp, vs, vbulk, akt =
isotropy.calculate_seismic_properties('Forsterite', temperature=2000,
pressure=2, return_vp_vs_vbulk=True, return_aktout=True)
print(density, aks, amu, vp, vs, vbulk, akt)
```