# pyrolite-meltsutil Documentation

*Release 0+u*

**Morgan Williams**

**Nov 17, 2023**

# CONTENTS

pyrolite-meltsutil is a pyrolite extension for working with alphaMELTS and its outputs.

The python package includes functions to install and run melts with specific configurations, to import output tables, and to visualise these results.

- On this site you can browse the API, or look through some of the usage examples.
- There's also a quick installation guide and some notes on where the project is heading in the near future.

Note that this package is currently experimental and the project is not affiliated with alphaMELTS or related entities. For more information on these projects, see the alphaMELTS site and relevant references[1][2][3][4][5][6][7].

[1] Ghiorso M. S. and Sack R. O. (1995). Chemical mass transfer in magmatic processes IV. A revised and internally consistent thermodynamic model for the interpolation and extrapolation of liquid-solid equilibria in magmatic systems at elevated temperatures and pressures. Contributions to Mineralogy and Petrology 119, 197–212. doi: 10.1007/BF00307281

[2] Ghiorso M. S., Hirschmann M. M., Reiners P. W. and Kress V. C. (2002). The pMELTS: A revision of MELTS for improved calculation of phase relations and major element partitioning related to partial melting of the mantle to 3 GPa. Geochemistry, Geophysics, Geosystems 3, 1–35. doi: 10.1029/2001GC000217

[3] Asimow P. D., Dixon J. E. and Langmuir C. H. (2004). A hydrous melting and fractionation model for mid-ocean ridge basalts: Application to the Mid-Atlantic Ridge near the Azores. Geochemistry, Geophysics, Geosystems 5. doi: 10.1029/2003GC000568

[4] Smith P. M. and Asimow P. D. (2005). Adiabat_1ph: A new public front-end to the MELTS, pMELTS, and pHMELTS models. Geochemistry, Geophysics, Geosystems 6. doi: 10.1029/2004GC000816

[5] Thompson R. N., Riches A. J. V., Antoshechkina P. M., Pearson D. G., Nowell G. M., Ottley C. J., Dickin A. P., Hards V. L., Nguno A.-K. and Niku-Paavola V. (2007). Origin of CFB Magmatism: Multi-tiered Intracrustal Picrite–Rhyolite Magmatic Plumbing at Spitzkoppe, Western Namibia, during Early Cretaceous Etendeka Magmatism. J Petrology 48, 1119–1154. doi: 10.1093/petrology/egm012

[6] Antoshechkina P. M., Asimow P. D., Hauri E. H. and Luffi P. I. (2010). Effect of water on mantle melting and magma differentiation, as modeled using Adiabat_1ph 3.0. AGU Fall Meeting Abstracts 53, V53C-2264.

[7] Antoshechkina P. M. and Asimow P. D. (2010). Adiabat_1ph 3.0 and the MAGMA website: educational and research tools for studying the petrology and geochemistry of plate margins. AGU Fall Meeting Abstracts 41, ED41B-0644.

CHAPTER

ONE

# INSTALLATION

pyrolite is available on PyPi, and can be downloaded with pip:

```
pip install pyrolite-meltsutil
```

**Note:** pyrolite-meltsutil is not yet packaged for Anaconda, and as such `conda install pyrolite-meltsutil` will not work.

## 1.1 Upgrading pyrolite-meltsutil

New versions of pyrolite-meltsutil will be occasionally released. You can upgrade to the latest edition on PyPi using the `--upgrade` flag:

```
pip install --upgrade pyrolite-meltsutil
```

## 1.2 Optional Dependencies

Optional dependencies (*dev*) can be specified during pip installation. For example:

```
pip install pyrolite-meltsutil[dev]
```

**See also:**

Development Installation

CHAPTER

TWO

# EXAMPLES

This example gallery includes a variety of examples for using pyrolite-meltsutil which you can copy, download and alter, or run on Binder.

## 2.1 Table Examples

This subgallery includes examples for working with alphaMELTS tables.

### 2.1.1 Import Batch Configuration

When importing batches of alphaMELTS experiments which have been run using the *pyrolite_meltsutil.automation* interface, you can also import the batch configuration file which contains the index of hashes, names, configurations and the environment for the experiments. This can be handy to refer back to when automatically visualising or searching through your experiments.

First let's find a folder with some results. In this case we'll use one of the pyrolite-meltsutil example folders which already contains some batch experiment results:

```
from pyrolite_meltsutil.util.general import get_data_example

experiment_dir = get_data_example("batch")
```

Now we can import the configuration file and explore it's contents:

```
from pyrolite_meltsutil.tables.load import import_batch_config

cfg = import_batch_config(experiment_dir)
```

The configuration is imported as a dictionary, with keys which correspond to experiment hashes, which should line up with the experiment folders:

```
cfg.keys()
```

```
dict_keys(['363f3d0a0b', '4689ca6fc3'])
```

The values of the dictionary are tuples containing the experiment title, meltsfile configuration and the environment:

```
exp_name, exp_cfg, exp_env = cfg["363f3d0a0b"]
```

```
exp_name
```

```
'Gale2013MORBisobarfrac5-5kbar1300-800CFMQ363f3d0a0b'
```

```
exp_cfg
```

```
{'Initial Temperature': 1300, 'Final Temperature': 800, 'modes': ['isobaric',
↪'fractionate solids'], 'Initial Pressure': 5000, 'Log fO2 Path': 'FMQ', 'SiO2': 50.41,
↪'Al2O3': 14.95, 'FeO': 10.07, 'MnO': 0.173, 'MgO': 7.69, 'CaO': 11.35, 'Na2O': 2.76,
↪'TiO2': 1.54, 'K2O': 0.144, 'P2O5': 0.169, 'Title': 'Gale2013MORB', 'Final Pressure':␣
↪5000, 'Increment Temperature': -5, 'Increment Pressure': 0}
```

```
exp_env
```

```
{'VERSION': 'MELTS', 'MODE': 'isobaric', 'DELTAP': 0.0, 'DELTAT': -10.0, 'MAXP': 10000.0,
↪ 'MINP': 5000.0, 'MAXT': 1500.0, 'MINT': 500.0, 'MINF': 0.005, 'MASSIN': 0.001, 'MGO_
↪TARGET': 8.0, 'DRY_ITER_PATIENCE': 100, 'HK_PXGT_TRACE_H2O': True, 'FAILED_ITER_
↪PATIENCE': 10}
```

**See also:**

Loading Melts Tables, Aggregating Tables,

**Total running time of the script:** (0 minutes 0.002 seconds)

### 2.1.2 Aggregating Tables

As one of the main use cases for using pyrolite-meltsutil is executing, interrogating and visualising multiple experiments, one of the core functionalities is importing alphaMELTS results and integrating these. Of the key functions to do this is `aggregate_tables()`. This enables you to load in all the results from an array of experiments within a single folder, enabling subsequent analysis and visualization.

First let's find a folder with some results. In this case we'll use one of the pyrolite-meltsutil example folders which already contains some batch experiment results:

```
from pyrolite_meltsutil.util.general import get_data_example

experiment_dir = get_data_example("batch")
```

Now we can import the table files from each of the experiments. Note that in the same fashion as `import_tables()`, `aggregate_tables()` returns two tables - one for `system` variables and one for `phases`, which contains information pertaining to individual phases or aggregates (e.g. 'olivine_0', 'bulk', 'liquid' etc).

```
from pyrolite_meltsutil.tables.load import aggregate_tables

system, phases = aggregate_tables(experiment_dir)
```

```
/home/docs/checkouts/readthedocs.org/user_builds/pyrolite-meltsutil/checkouts/develop/
↪pyrolite_meltsutil/tables/load.py:333: FutureWarning: The behavior of DataFrame␣
↪concatenation with empty or all-NA entries is deprecated. In a future version, this␣
↪will no longer exclude empty or all-NA columns when determining the result dtypes. To␣
↪retain the old behavior, exclude the relevant entries before the concat operation.
```

(continues on next page)

(continued from previous page)

```
  phase = pd.concat([phase, cumulate_comp])
/home/docs/checkouts/readthedocs.org/user_builds/pyrolite-meltsutil/checkouts/develop/
↪pyrolite_meltsutil/tables/load.py:333: FutureWarning: The behavior of DataFrame␣
↪concatenation with empty or all-NA entries is deprecated. In a future version, this␣
↪will no longer exclude empty or all-NA columns when determining the result dtypes. To␣
↪retain the old behavior, exclude the relevant entries before the concat operation.
  phase = pd.concat([phase, cumulate_comp])
```

In addition to the variables you'd expect from the tables, the returned dataframes also include an 'experiment' column which contains the hash-index of each experiment such that they can be easily distinguished:

```
phases.experiment.unique()
```

```
array(['4689ca6fc3', '363f3d0a0b'], dtype=object)
```

As this aggregation process can take a while for larger arrays of experiments, it's generally a good idea to save these results to disk such that they can be loaded faster:

```
import pandas as pd

system.to_csv(experiment_dir / "system.csv")
phases.to_csv(experiment_dir / "phases.csv")
```

Then next time you wish to access the data, you could simply load the tables back in with:

```
system, phases = (
    pd.read_csv(experiment_dir / "system.csv"),
    pd.read_csv(experiment_dir / "phases.csv"),
)
```

**See also:**

Loading Melts Tables, Import Batch Configuration,

**Total running time of the script:** (0 minutes 0.413 seconds)

### 2.1.3 Loading Melts Tables

pyrolite-meltsutil includes utilites to import the information from alphaMELTS-generated tables such that they can be more readily used for further interrogation and visualiastion.

This example shows how you would typically import a single experiment from the experiment directory. In this case we point you to a pyrolite-meltsutil example folder which already contains some table files. Note that the function returns two tables - one for `system` variables and one for `phases`, which contains information pertaining to individual phases or aggregates (e.g. 'olivine_0', 'bulk', 'liquid' etc).

```
from pyrolite_meltsutil.tables.load import import_tables
from pyrolite_meltsutil.util.general import get_data_example

# let's use the example batch data for this
experiment_dir = get_data_example("batch/363f3d0a0b")
system, phases = import_tables(experiment_dir)  # let's import the tables
```

```
/home/docs/checkouts/readthedocs.org/user_builds/pyrolite-meltsutil/checkouts/develop/
↪pyrolite_meltsutil/tables/load.py:333: FutureWarning: The behavior of DataFrame␣
↪concatenation with empty or all-NA entries is deprecated. In a future version, this␣
↪will no longer exclude empty or all-NA columns when determining the result dtypes. To␣
↪retain the old behavior, exclude the relevant entries before the concat operation.
  phase = pd.concat([phase, cumulate_comp])
```

The system table contains intensive variables and 'whole of system' measures.

```
system.head(3).T
```

The phases table contains information about individual components. In this case this includes the liquid, and also other aggregate measures such as 'bulk' and 'solid'.

```
phases.sample(3).T
```

The standard MELTS tables are extended to generate a 'cumulate' composiiton of integrated solids (for fractional crystallisation experiments) from the *phases* table:

```
cumulate = phases.loc[phases.phase == "cumulate", :]
```

Some of the thermodynamic variables will necessarily be missing for now, but most other relevant variables are present:

```
cumulate.sample(3).dropna(how="all", axis="columns").T
```

These cumulate compositions are generated with the `integrate_solid_composition()` function (along with a few additions provided in `import_tables()` which reindex and calcuate relative percentages using the system table). You should get similar results with:

```
from pyrolite_meltsutil.util.tables import integrate_solid_composition

cumulate_comp = integrate_solid_composition(phases)
cumulate_comp.tail()
```

Similarly, you can integrate phase proportions using `integrate_solid_proportions()`:

```
from pyrolite_meltsutil.util.tables import integrate_solid_proportions

cumulate_phases = integrate_solid_proportions(phases)
cumulate_phases.tail()
```

**See also:**

Aggregating Tables, Import Batch Configuration,

**Total running time of the script:** (0 minutes 0.287 seconds)

## 2.2 Automating alphaMELTS

This subgallery includes examples for conducting matches of alphaMELTS experiments based on a grid of configurations and compositions.

### 2.2.1 alphaMELTS Environment

```
from pyrolite_meltsutil.env import MELTS_Env
```

Set up an environment for an experiment stepping down temperature from 1350 to 1000 in 3 degree steps:

```
env = MELTS_Env()
env.MINP = 7000
env.MAXP = 7000
env.MINT = 1000
env.MAXT = 1350
env.DELTAT = -3
```

You can directly export these parameters to an envrionment file:

```
with open("pyrolite_envfile.txt", "w") as f:
    f.write(env.to_envfile(unset_variables=False))
```

You can then pass this to alphamelts using `run_alphamelts.command -f pyrolite_envfile.txt`

**See also:**

**Examples:**
pyrolite-hosted alphaMELTS Installation, Automating alphaMELTS Runs, Handling Outputs from Melts: Tables, Compositional Uncertainty Propagation for alphaMELTS Experiments

**Total running time of the script:** (0 minutes 0.010 seconds)

### 2.2.2 Automating alphaMELTS runs

pyrolite includes some utilities to help you run alphaMELTS with a little less hassle, especially for established workflows or repetitive calculations. Here we run multiple experiments at different conditions for a single MORB composition. Once we have the data in a `DataFrame`, we configure the default alphaMELTS environment before running the batch of experiments.

```
import os
import sys
from pathlib import Path

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

from pyrolite_meltsutil.automation import MeltsBatch
```

```
from pyrolite_meltsutil.util.synthetic import isobaricGaleMORBexample
```

(continues on next page)

(continued from previous page)

```
MORB = isobaricGaleMORBexample(title="Gale2013MORB")
MORB.T
```

```
from pyrolite_meltsutil.env import MELTS_Env

env = MELTS_Env()
env.VERSION = "MELTS"
env.MODE = "isobaric"
env.MINP = 5000
env.MAXP = 10000
env.MINT = 500
env.MAXT = 1500
env.DELTAT = -10
env.DELTAP = 0
```

Let's create a directory to run this experiment in - here we use an example folder:

```
from pyrolite_meltsutil.util.general import get_data_example

experiment_dir = get_data_example("batch")
```

Now we can set up the experiment. We're going to run a *MeltsBatch*, and we'll provide:

- The dataframe of compositions
- The default configuration
- The configuration grid with lists of parameter values - which overrides the default.
- The *MELTS_Env* or environment file to be used
- The directory to be used (defaults to the current working directory)
- Optionally, specify a logger for output

```
batch = MeltsBatch(
    MORB,
    default_config={  # things that won't change between experiments
        "Initial Temperature": 1400,
        "Final Temperature": 800,
        "modes": ["isobaric", "fractionate solids"],
    },
    config_grid={  # things that change between experiments
        "Initial Pressure": [5000, 7000],
        "Log fO2 Path": [None, "FMQ"],
        "modifychem": [None, {"H2O": 0.5}],
    },
    env=env,
    fromdir=experiment_dir,
)
```

Now the experiment is configured, we can run it:

```
batch.run(overwrite=False)  # overwrite=True if you want to update existing exp folders
```

**See also:**

**Examples:**
alphaMELTS Environment Configuration, pyrolite-hosted alphaMELTS Installation, Handling Outputs from Melts: Tables, Compositional Uncertainty Propagation for alphaMELTS Experiments

**Total running time of the script:** (0 minutes 0.030 seconds)

## 2.3 Visualisation Examples

This subgallery includes examples for visualising some of the output alphaMELTS tables.

### 2.3.1 Visualisation: Plot Templates

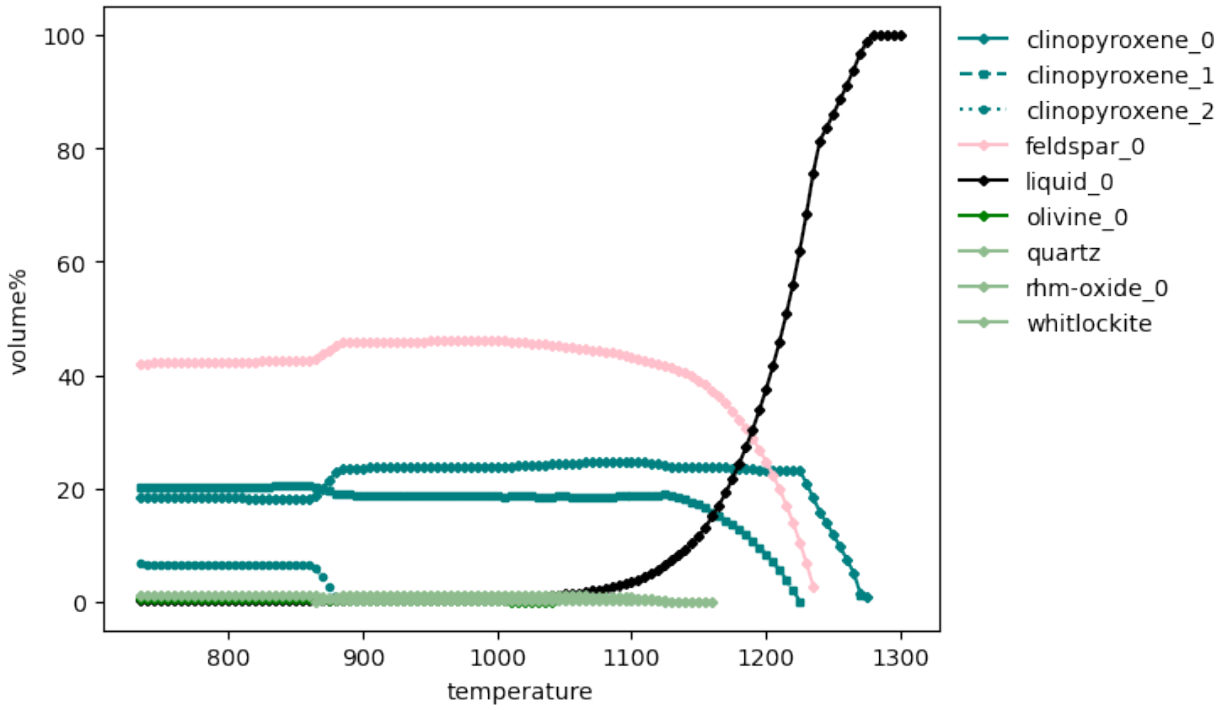
pyrolite-meltsutil includes a few plot templates to quickly visualise some melts experiment results.

First let's get a folder with some data in it. Here we use one of the isobaric crystallisation experiments from the montecarlo tutorial, and import the tables:

```
from pyrolite_meltsutil.util.general import get_data_example
from pyrolite_meltsutil.tables import import_tables


exp_dir = get_data_example("montecarlo/3149b39eee")
system, phases = import_tables(exp_dir)
```

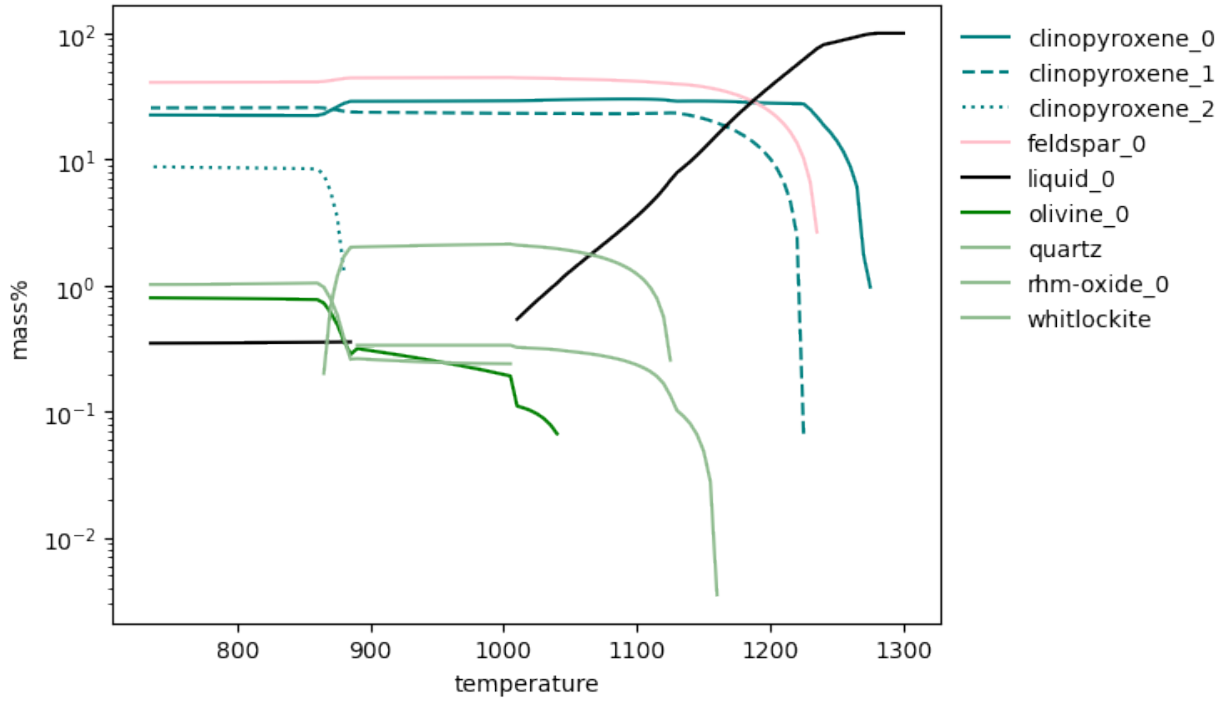
We can quickly visualise the phase volume relationships versus temperature:

```
import matplotlib.pyplot as plt

from pyrolite_meltsutil.vis.templates import plot_phasevolumes

ax, proxies = plot_phasevolumes(phases)
plt.show()
```




Similarly, for the phase mass relationships versus temperature:

```
from pyrolite_meltsutil.vis.templates import plot_phasemasses

ax, proxies = plot_phasemasses(phases, marker=None)
ax.set_yscale("log")
plt.show()
```



**Total running time of the script:** (0 minutes 2.363 seconds)

### 2.3.2 Visualisation: Distinguishing Phases

Styling by phase is largely achieved with color, here using the `phase_color()` function, which will return unique colors for each phase:

```
from pyrolite_meltsutil.vis.style import phase_color

phase_color("olivine")
```

```
'green'
```

This will also work if given a phase ID:

```
phase_color("olivine_0")
```

```
'green'
```

Similarly, to differentiate between different generations or endmembers, a linestyle or marker can be used, here generated with `phaseID_linestyle()` which takes the full phaseID:

```
from pyrolite_meltsutil.vis.style import phaseID_linestyle, phaseID_marker

[phaseID_linestyle(ol) for ol in ["olivine_0", "olivine_1", "olivine_2"]]
```

```
['-', '--', ':']
```

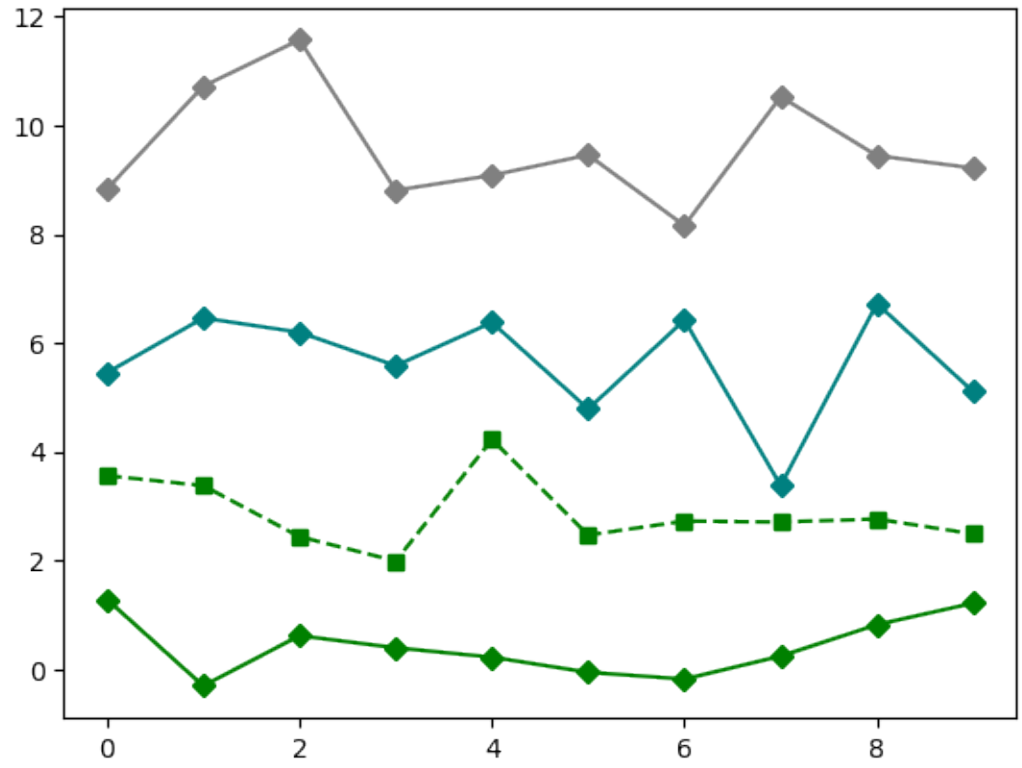
```
[phaseID_marker(ol) for ol in ["olivine_0", "olivine_1", "olivine_2"]]
```

```
['D', 's', 'o']
```

We can now use these when we're plotting to differentiate different phases:

```
phaseIDs = ["olivine_0", "olivine_1", "clinopyroxene_0", "spinel_0"]
styles = [
    dict(color=phase_color(ID), ls=phaseID_linestyle(ID), marker=phaseID_marker(ID))
    for ID in phaseIDs
]
```

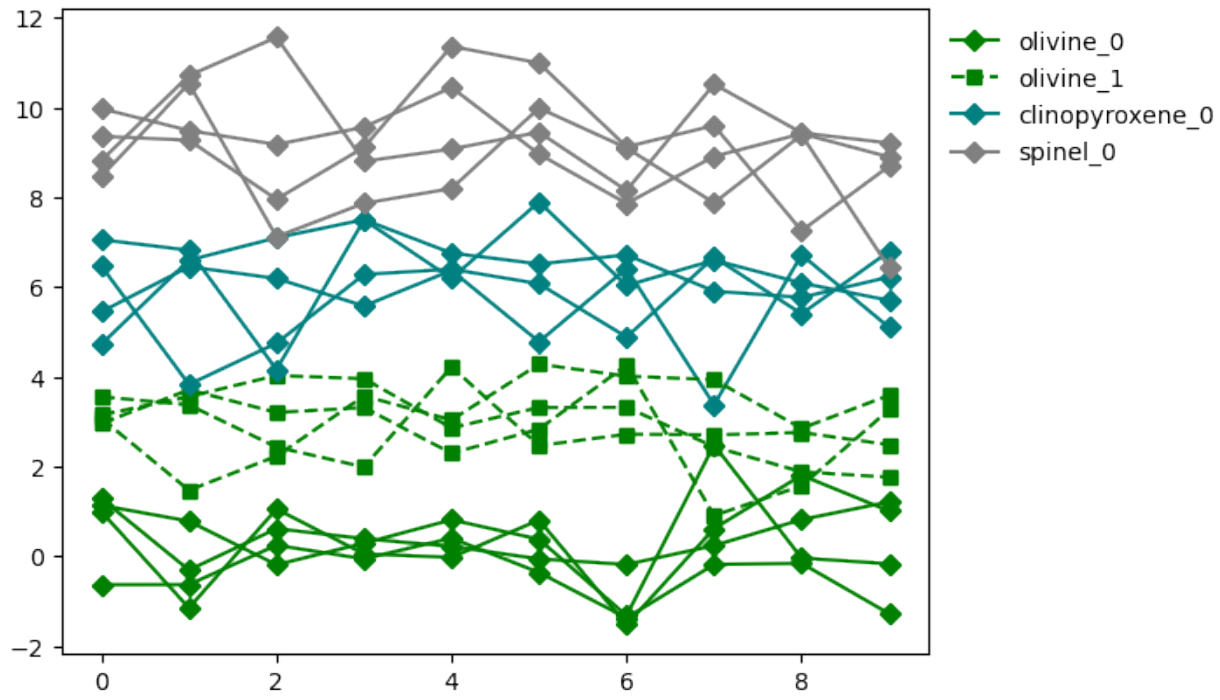
```
import numpy as np
import matplotlib.pyplot as plt
from pyrolite.util.plot.legend import proxy_line

np.random.seed(27)

fig, ax = plt.subplots(1)

for ix, (ID, style) in enumerate(zip(phaseIDs, styles)):
    ax.plot(np.arange(10), np.random.randn(10) + ix * 3, **style)
```




These are also handy for generating legend proxies, which can be used to generate summary legends where you may have multiple items with the same style:

```
proxies = [proxy_line(**sty) for sty in styles]
```

```
for ix, (ID, style) in enumerate(zip(phaseIDs, styles)):
    for i in range(3):  # make a few more lines per phaseID
        ax.plot(np.arange(10), np.random.randn(10) + ix * 3, **style)

ax.legend(  # use our proxy lines to generate a legend
    proxies,
    phaseIDs,
    frameon=False,
    facecolor=None,
    bbox_to_anchor=(1.0, 1.0),
    loc="upper left",
)
plt.show()
```




**Total running time of the script:** (0 minutes 0.332 seconds)

### 2.3.3 Visualising Cumulate Compositions

First we'll import a set of example data tables:

```
from pyrolite_meltsutil.util.general import get_data_example
from pyrolite_meltsutil.tables.load import import_tables

hsh = "3149b39eee"  # the hash index of our experiment
batchdir = get_data_example("montecarlo")  # let's use the example batch data for this
system, phases = import_tables(batchdir / hsh)  # let's import the tables
```

The cumulate composition is automatically calculated and added to the phase table:

```
phases.loc[phases.phase == "cumulate", :].head(3).T
```

We can also manually calculate the phase mass proportions for the cumulate pile:

```
from pyrolite_meltsutil.util.tables import integrate_solid_proportions

cumulate_phases = integrate_solid_proportions(phases, frac=False)
cumulate_phases.tail(3).T
```

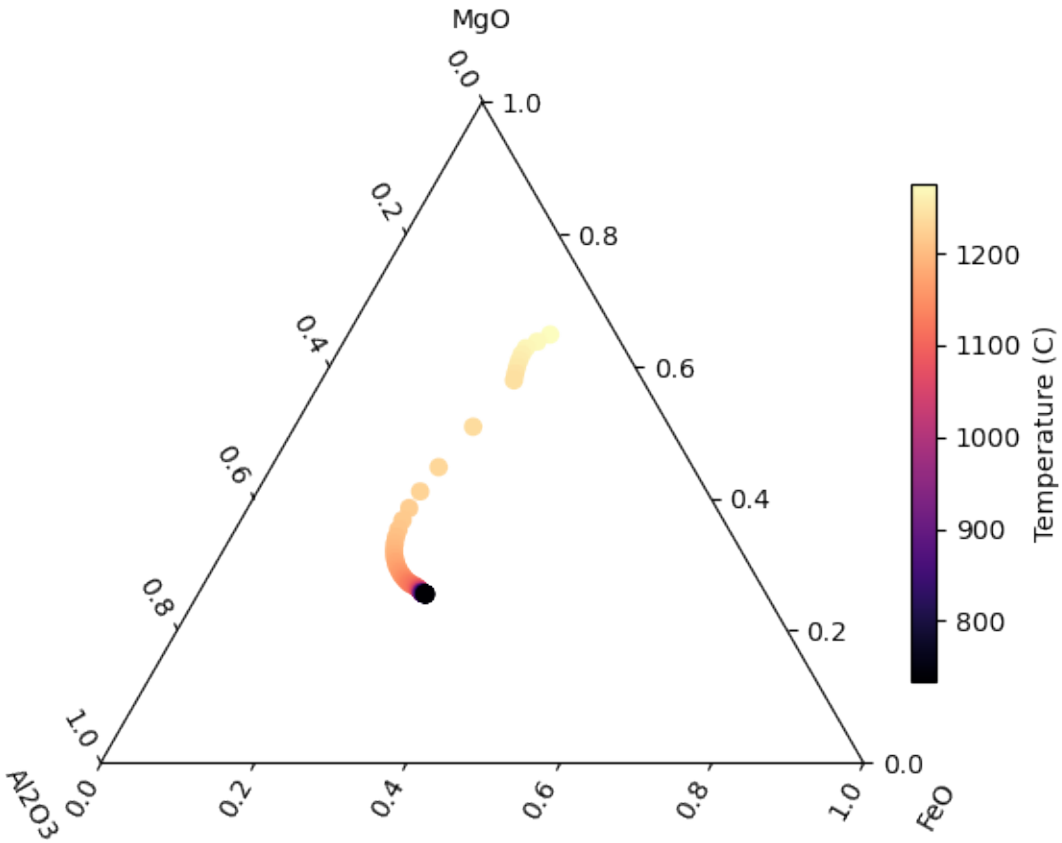
Ternary diagrams can be useful to visualise how the overal/fractional cumulates change during the experiment:

```
import matplotlib.pyplot as plt
import pyrolite.plot
from pyrolite.util.plot.style import mappable_from_values
```

(continues on next page)

(continued from previous page)

```
chemvars = ["MgO", "Al2O3", "FeO"]
cumulate_comp = phases.loc[phases.phase == "cumulate", :]
ax = cumulate_comp.loc[:, chemvars].pyroplot.scatter(
    c=cumulate_comp.temperature, cmap="magma"
)
plt.colorbar(
    mappable_from_values(cumulate_comp.temperature.dropna(), cmap="magma"),
    label="Temperature (C)",
    ax=ax,
    shrink=0.7,
)
plt.show()
```



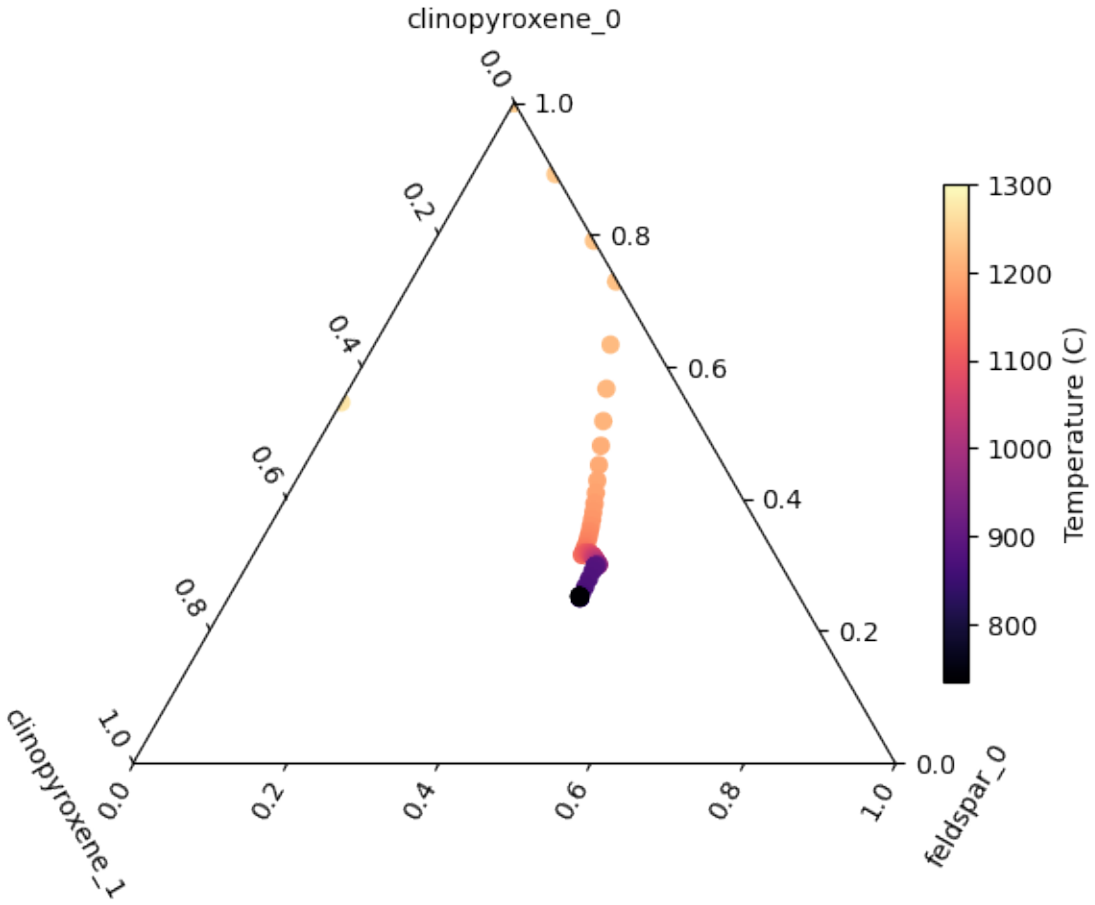

Similarly, we can plot the phase proportions:

```
phaselist = ["clinopyroxene_0", "clinopyroxene_1", "feldspar_0"]
ax = cumulate_phases.loc[:, phaselist].pyroplot.scatter(
    c=cumulate_phases.temperature, cmap="magma"
)
plt.colorbar(
    mappable_from_values(cumulate_phases.temperature, cmap="magma"),
    label="Temperature (C)",
    ax=ax,
    shrink=0.7,
```

(continues on next page)

(continued from previous page)

```
)
plt.show()
```




**Total running time of the script:** (0 minutes 7.706 seconds)

### 2.3.4 Visualising Melts Results by Phases

First we'll import a set of example data tables:

```
from pyrolite_meltsutil.util.general import get_data_example
from pyrolite_meltsutil.tables.load import import_tables, import_batch_config

hsh = "363f3d0a0b"  # the hash index of our experiment
batchdir = get_data_example("batch")  # let's use the example batch data for this
system, phases = import_tables(batchdir / hsh)  # let's import the tables
name, cfg, env = import_batch_config(batchdir)[hsh]  # and also the configuration
```

```
/home/docs/checkouts/readthedocs.org/user_builds/pyrolite-meltsutil/checkouts/develop/
↪pyrolite_meltsutil/tables/load.py:333: FutureWarning: The behavior of DataFrame␣
↪concatenation with empty or all-NA entries is deprecated. In a future version, this␣
↪will no longer exclude empty or all-NA columns when determining the result dtypes. To␣
```

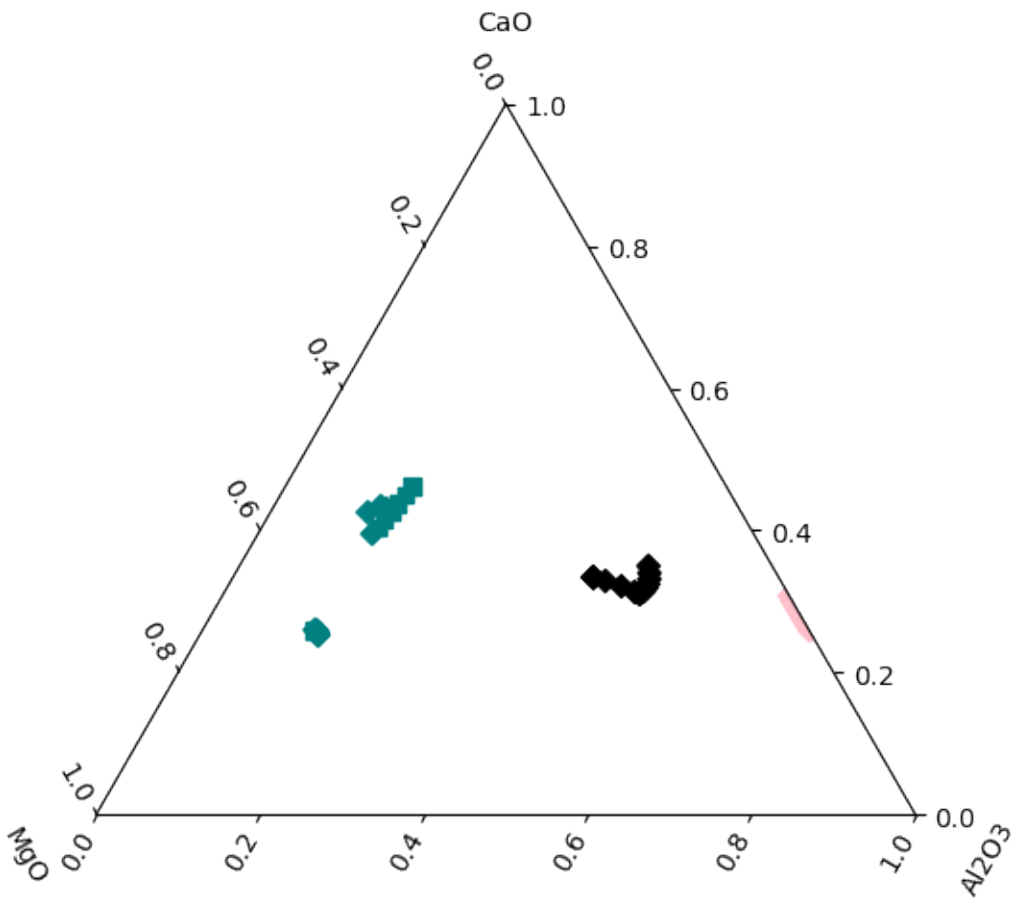
(continues on next page)

(continued from previous page)

```
↪retain the old behavior, exclude the relevant entries before the concat operation.
  phase = pd.concat([phase, cumulate_comp])
```

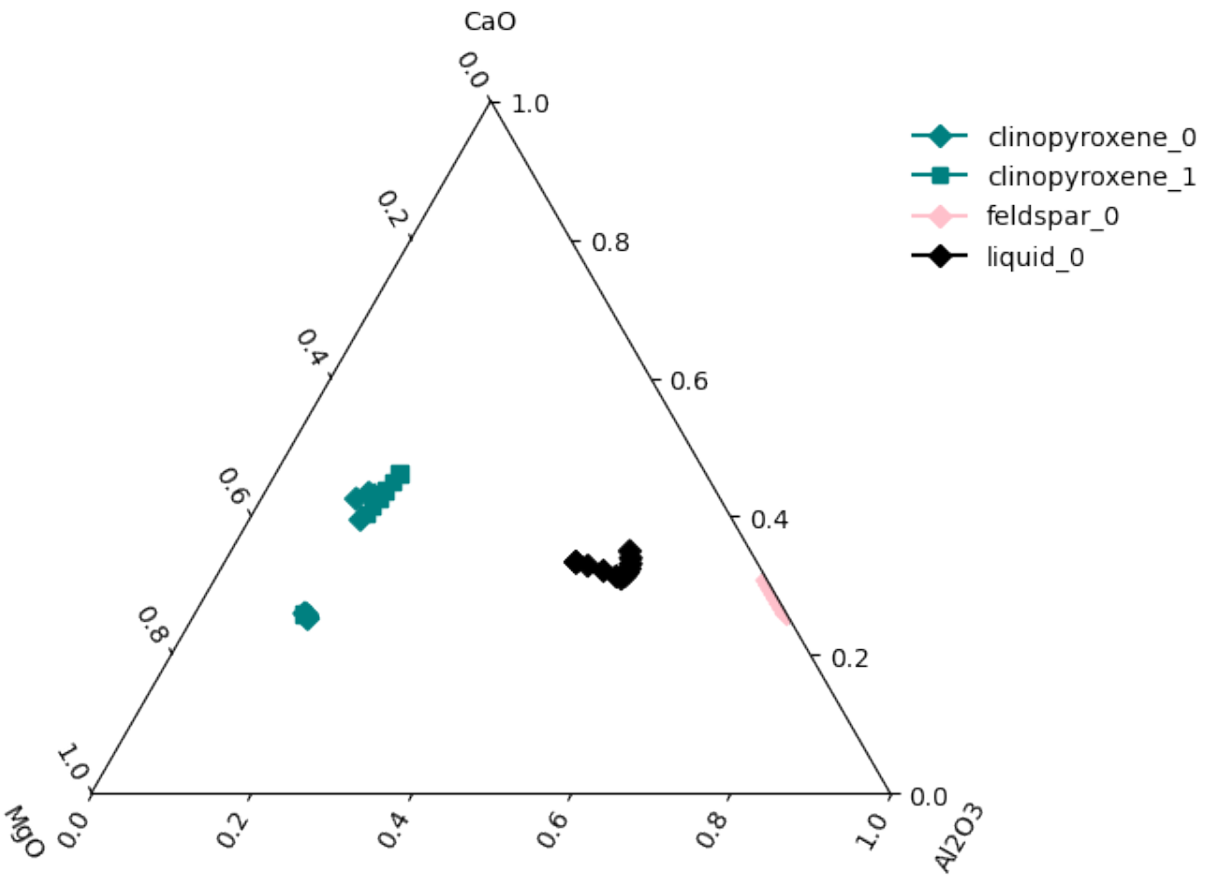
Now we can dig into some tables and plot some data for different phases.

```
import pyrolite.plot
import matplotlib.pyplot as plt
from pyrolite.util.plot.legend import proxy_line
from pyrolite_meltsutil.vis.style import phase_color, phaseID_linestyle, phaseID_marker

phasenames = ["olivine", "clinopyroxene", "feldspar", "liquid"]

fig, ax = plt.subplots(1)

proxies = {}  # proxies for creating a legend
for ix, phs in enumerate(phasenames):
    phase_data = phases.loc[phases.phase == phs, :]
    for phaseID, phaseID_data in phase_data.groupby("phaseID"):
        style = dict(c=phase_color(phaseID), marker=phaseID_marker(phaseID))
        ax = phaseID_data.loc[:, ["CaO", "MgO", "Al2O3"]].pyroplot.scatter(
            ax=ax, **style
        )

        proxies[phaseID] = proxy_line(
            ls="-", color=phase_color(phaseID), marker=phaseID_marker(phaseID)
        )
```




Finally we can generate the legend using our legend proxies:

```
ax.legend(
    list(proxies.values()),
    list(proxies.keys()),
    frameon=False,
    facecolor=None,
    bbox_to_anchor=(1, 1),
    loc="upper left",
)
plt.show()
```




**Total running time of the script:** (0 minutes 6.467 seconds)

## 2.4 Installing alphaMELTS

This subgallery includes examples illustrating how to get a local copy of alphaMELTS which pyrolite-meltsutil can use to execute batches of experiments.

### 2.4.1 Local Installation of alphaMELTS

pyrolite can download and manage its own version of alphaMELTS (without any real 'installation', *per-se*), and use this for *automation* purposes.

```
from pyrolite.util.log import stream_log

from pyrolite_meltsutil.download import install_melts
from pyrolite_meltsutil.util.general import pyrolite_meltsutil_datafolder
```

Here we can do a conditonal install - only downloading alphamelts if it doesnt exist:

```
if not (pyrolite_meltsutil_datafolder(subfolder="localinstall")).exists():
    stream_log("pyrolite-meltsutil", level="INFO")  # logger for output info
    install_melts(local=True)  # install a copy of melts to pyrolite data folder
```

**Warning:** This 'local install' method still requires that you have Perl installed, as it uses the Perl `run_alphamelts.command` script. If you're on Windows, you can use Strawberry Perl for this purpose.

**See also:**

**Examples:**

alphaMELTS Environment Configuration, Automating alphaMELTS Runs, Handling Outputs from Melts: Tables, Compositional Uncertainty Propagation for alphaMELTS Experiments

**Total running time of the script:** (0 minutes 0.857 seconds)

CHAPTER

# THREE

# TUTORIALS

This example gallery includes a variety of tutorials for using pyrolite-meltsutil which you can copy, download and alter, or run on Binder.

## 3.1 Visualising SCSS

pyrolite includes a few functions for working with magmatic liquid compositions, one of which being the Sulfur Content at Sulfur Saturation (`pyrolite.geochem.magma.SCCS()`) function. This is an empirical relationship derived by Li and Ripley (2009)[1] which enables a better understanding of relative saturation sulfur saturation state (for both sulfide and sulfate) and the prediction of sulfur saturation in evolving melts.

Here we use this function to predict sulfur saturation in a fractionally crystallizing MORB melt with a range of sulfur contents. First we'll import a set of example data tables:

```
from pyrolite_meltsutil.tables.load import import_batch_config, import_tables
from pyrolite_meltsutil.util.general import get_data_example

hsh = "363f3d0a0b"  # the hash index of our experiment
batchdir = get_data_example("batch")  # let's use the example batch data for this
system, phases = import_tables(batchdir / hsh, kelvin=False)  # let's import the tables
name, cfg, env = import_batch_config(batchdir)[hsh]  # and also the configuration
```

```
/home/docs/checkouts/readthedocs.org/user_builds/pyrolite-meltsutil/checkouts/develop/
↪pyrolite_meltsutil/tables/load.py:333: FutureWarning: The behavior of DataFrame␣
↪concatenation with empty or all-NA entries is deprecated. In a future version, this␣
↪will no longer exclude empty or all-NA columns when determining the result dtypes. To␣
↪retain the old behavior, exclude the relevant entries before the concat operation.
  phase = pd.concat([phase, cumulate_comp])
```

From this we extract only the liquid composition:

```
liquid = phases.loc[phases.phase == "liquid", :]
```

Now we can calcuate the sulfur saturation at sulfide saturation for this magma. This table also includes the relevant temperature and pressure data, noting that the temperature here is in degrees Celsius (`kelvin = False`) and the pressure is in bars (wheras this function requires kbar, hence the division by 1000):

[1] Li, C., and Ripley, E.M. (2009). Sulfur Contents at Sulfide-Liquid or Anhydrite Saturation in Silicate Melts: Empirical Equations and Example Applications. Economic Geology 104, 405–412. doi: gsecongeo.104.3.405

```
from pyrolite.geochem.magma import SCSS

sulfate, sulfide = SCSS(
    liquid, T=liquid.temperature, P=liquid.pressure / 1000, grid=None, kelvin=False
)
```

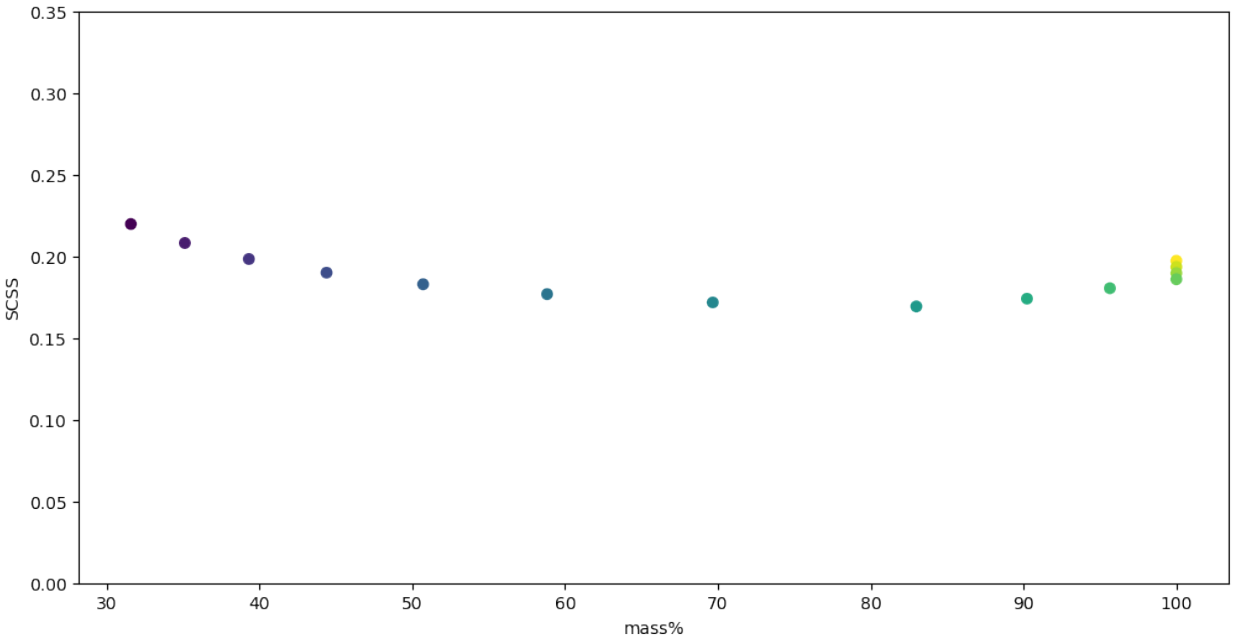
To link this back to our chemical data, let's add this measure to the dataframe:

```
liquid.loc[:, "SCSS"] = sulfide
```

```
/home/docs/checkouts/readthedocs.org/user_builds/pyrolite-meltsutil/checkouts/develop/
↪docs/source/gallery/tutorials/scss.py:43: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
↪guide/indexing.html#returning-a-view-versus-a-copy
  liquid.loc[:, "SCSS"] = sulfide
```

Now we can plot this against some of the experiment parameters. Here we plot SCSS against the remaining (non-crystallized) mass of the system and color the results by temperature:

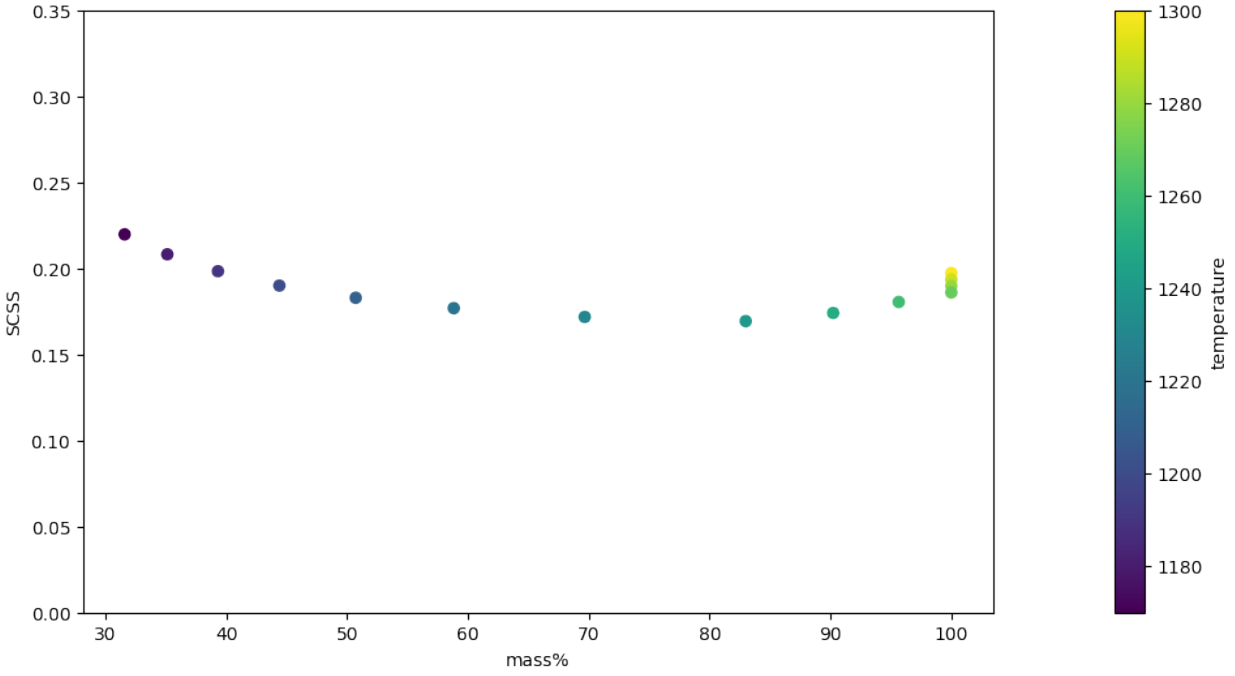
```
import matplotlib.pyplot as plt
import pyrolite.plot

from pyrolite_meltsutil.vis.scss import plot_sulfur_saturation_point

xvar, colorvar = "mass%", "temperature"

# show the SCSS for the liqud
ax = liquid.loc[:, [xvar, "SCSS"]].pyroplot.scatter(
    c=liquid[colorvar], fontsize=12, figsize=(12, 6)
)
ax.set_ylim(0, 0.35)
```




```
(0.0, 0.35)
```

To this we can add a colorbar for the temperature color mapping:

```
from pyrolite.util.plot.style import mappable_from_values
from pyrolite.util.plot.axes import add_colorbar

plt.colorbar(mappable_from_values(liquid[colorvar]), ax=ax, pad=0.1, label=colorvar)
plt.show()
```



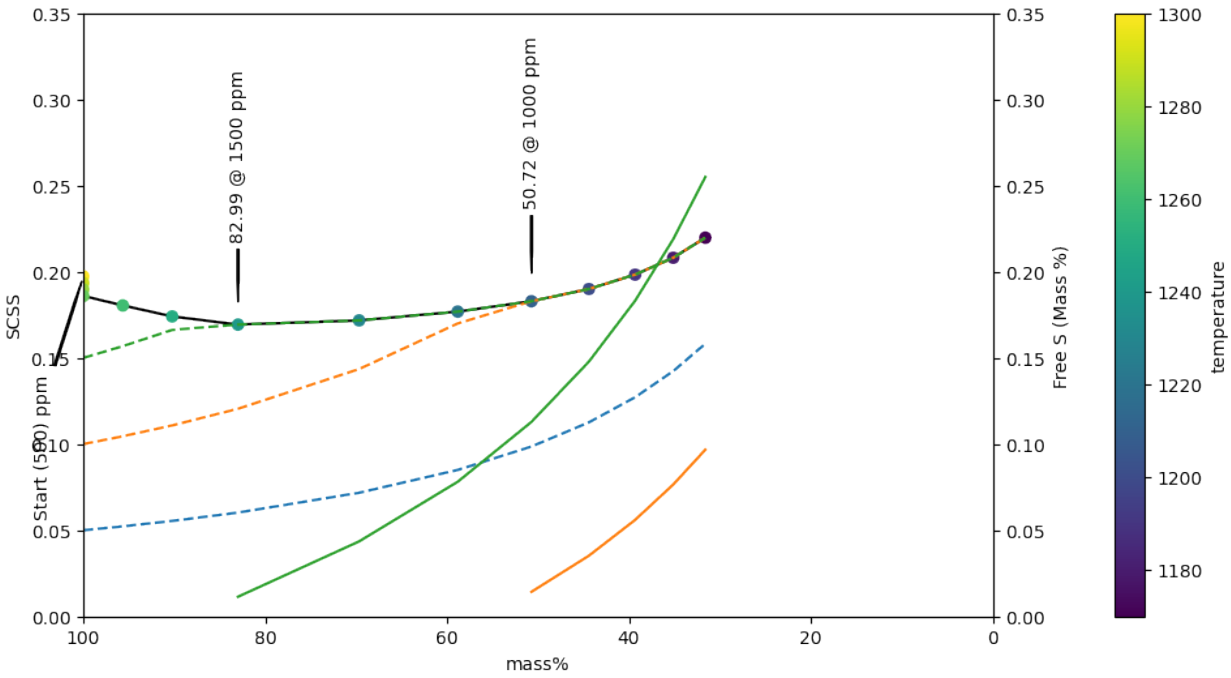
We can clean up these axes a bit by relimiting and rescaling:

```
import numpy as np

ax.set_xlim((np.nanmax(liquid["mass%"]), 0))
```

```
(100.00097488322528, 0.0)
```

To this we can also add the saturation points (where sulfur content crosses SCSS) and plot the mass of free sulfide liquid expected for a range of initial sulfur contents. Note here that the sulfur content in the liquid is plotted with dashed lines (linked to left axis), and the free sulfide liquid in solid lines (right axis). Once saturation is reached, the sulfur content in the melt will follow the SCSS curve unless the system is perturbed or becomes undersaturated again.

```
plot_sulfur_saturation_point(liquid, start=[500, 1000, 1500], xvar=xvar, ax=ax)
plt.show()
```




```
/home/docs/checkouts/readthedocs.org/user_builds/pyrolite-meltsutil/checkouts/develop/
↪pyrolite_meltsutil/vis/scss.py:73: FutureWarning: Series.__getitem__ treating keys as␣
↪positions is deprecated. In a future version, integer keys will always be treated as␣
↪labels (consistent with DataFrame behavior). To access a value by position, use `ser.
↪iloc[pos]`
  satabund = s_wtpct[satpoint]
/home/docs/checkouts/readthedocs.org/user_builds/pyrolite-meltsutil/checkouts/develop/
↪pyrolite_meltsutil/vis/scss.py:87: FutureWarning: Series.__getitem__ treating keys as␣
↪positions is deprecated. In a future version, integer keys will always be treated as␣
↪labels (consistent with DataFrame behavior). To access a value by position, use `ser.
↪iloc[pos]`
  saturation_info[xS] = dict(x=liquid[xvar][satpoint], y=s_wtpct[satpoint])
```

### 3.1.1 References

**Total running time of the script:** (0 minutes 1.007 seconds)

## 3.2 Uncertainties and alphaMELTS Experiments

While alphaMELTS is a useful tool for formulating hypotheses around magmatic processes, analytical uncertainties for compositional parameters are difficult to propagate. Here I've given an example of taking the composition of average MORB, adding 'noise' to represent multiple possible realisations under analytical uncertainties, and conducted replicate alphaMELTS experiments to provide some quantification of the uncertainties in the results. Note that the 'noise' added here is uncorrelated, and as such may usefully represent analytical uncertainty. Geological uncertainties are typically strongly correlated, and the uncertainties associated with e.g. variable mineral assemblages should be modelled differently.

```
from pathlib import Path

import numpy as np
import pandas as pd

np.random.seed(23)
```

We'll use the major element composition of MORB from Gale et al (2013) for this exercise:

```
from pyrolite_meltsutil.util.synthetic import isobaricGaleMORBexample

MORB = isobaricGaleMORBexample(title="Gale2013MORB")
MORB.T
```

As we're going to 'blur' compositions by adding compositional noise to them, it'll be handy to have a function to do so. Here's a simple one which achieves this and is sufficient for our purpose:

```
from pyrolite.comp.codata import ILR, inverse_ILR


def blur_compositions(df, noise=0.05, scale=100):
    """
    Function to add 'compositional noise' to a set of compositions. In reality, it's
    its best to use measured uncertainties to generate these simulated compositions.
    """
    # transform into compositional space, add noise, return to simplex
    xvals = ILR(df.values)
    xvals += np.random.randn(*xvals.shape) * noise
    return inverse_ILR(xvals) * scale
```

We'll replicate this composition a number of times, and then add gaussian noise to each to create a range of plausible compositions:

```
import pyrolite.geochem
from pyrolite.util.pd import accumulate
from pyrolite.util.text import slugify

reps = 10  # increase this to perform more experiments
```

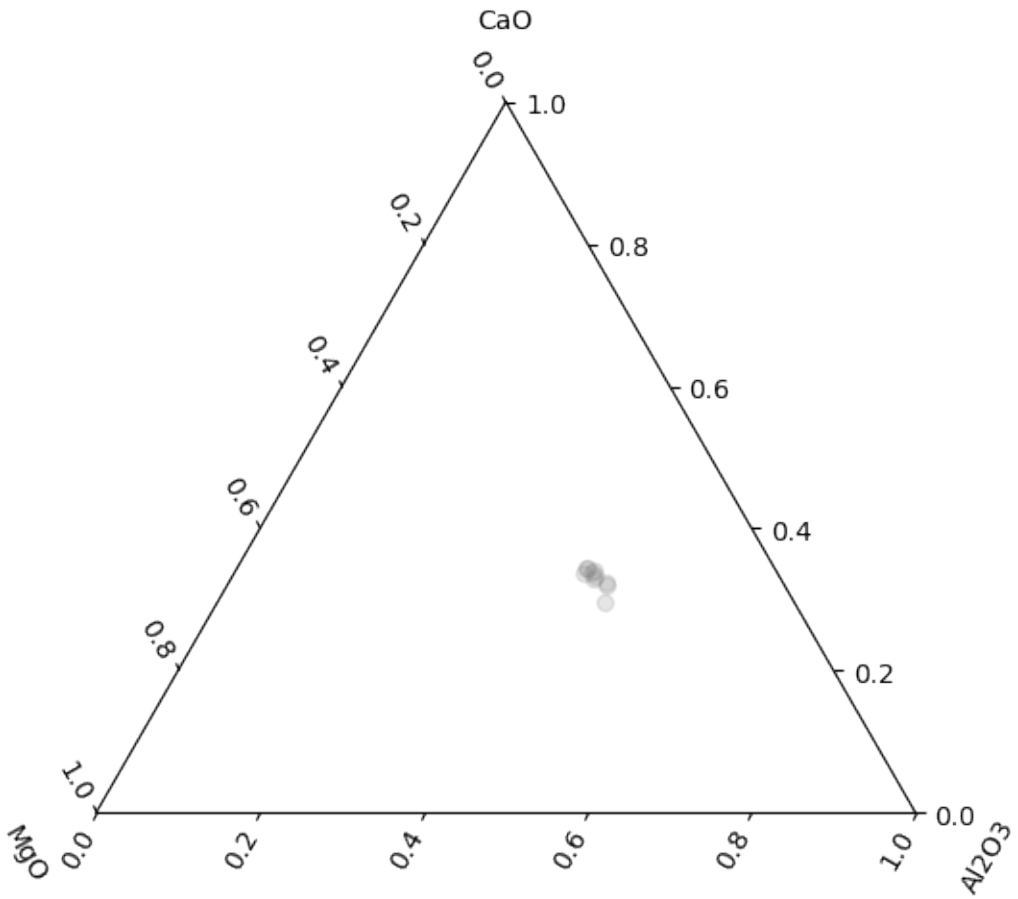
(continues on next page)

(continued from previous page)

```
df = accumulate([MORB] * reps)
df = df.reset_index().drop(columns="index")
df[df.pyrochem.list_oxides] = (
    df.loc[:, df.pyrochem.list_oxides].astype(float).pyrocomp.renormalise()
)
df[df.pyrochem.list_oxides] = blur_compositions(df[df.pyrochem.list_oxides])

df.Title = df.Title + " " + df.index.map(str)  # differentiate titles
df.Title = df.Title.apply(slugify)
```

We can visualise this variation in a ternary space:

```
import pyrolite.plot
import matplotlib.pyplot as plt

ax = df.loc[:, ["CaO", "MgO", "Al2O3"]].pyroplot.scatter(alpha=0.2, c="0.5")
plt.show()
```




Now we can setup an environment for isobaric fractional crystallisation:

```
from pyrolite_meltsutil.env import MELTS_Env

env = MELTS_Env()
```

(continues on next page)

(continued from previous page)

```
env.VERSION = "MELTS"  # crustal processes, < 1GPa/10kbar
env.MODE = "isobaric"
env.DELTAT = -5
env.MINP = 0
env.MAXP = 10000
```

Let's create a directory to run this experiment in - here we use an example folder:

```
from pyrolite_meltsutil.util.general import get_data_example

experiment_dir = get_data_example("montecarlo")
```

Let's also set up logging we can see the progression:

```
from pyrolite.util.log import stream_log
import logging

logger = logging.Logger(__name__)
stream_log(logger)
```

```
<Logger __main__ (INFO)>
```

Next we setup the alphaMELTS configuration for each of the inputs:

```
from pyrolite_meltsutil.automation import MeltsBatch

batch = MeltsBatch(
    df,
    default_config={
        "Initial Pressure": 5000,
        "Initial Temperature": 1300,
        "Final Temperature": 800,
        "modes": ["isobaric"],
    },
    config_grid={
        # "Initial Pressure": [3000, 7000],
        "Log fO2 Path": [None, "FMQ"],
        # "modifychem": [None, {"H2O": 0.5}],
    },
    env=env,
    fromdir=experiment_dir,
    logger=logger,
)
```

```
__main__ - INFO: Estimated Calculation Time: 0:02:30
```

The series of calls to alphaMELTS are now configured, and could be run as follows (`:overwrite=False` if you don't want to update existing experiment folders). Here we've already run the experiment and will load local data for the experiment to keep the documentation-building time low.

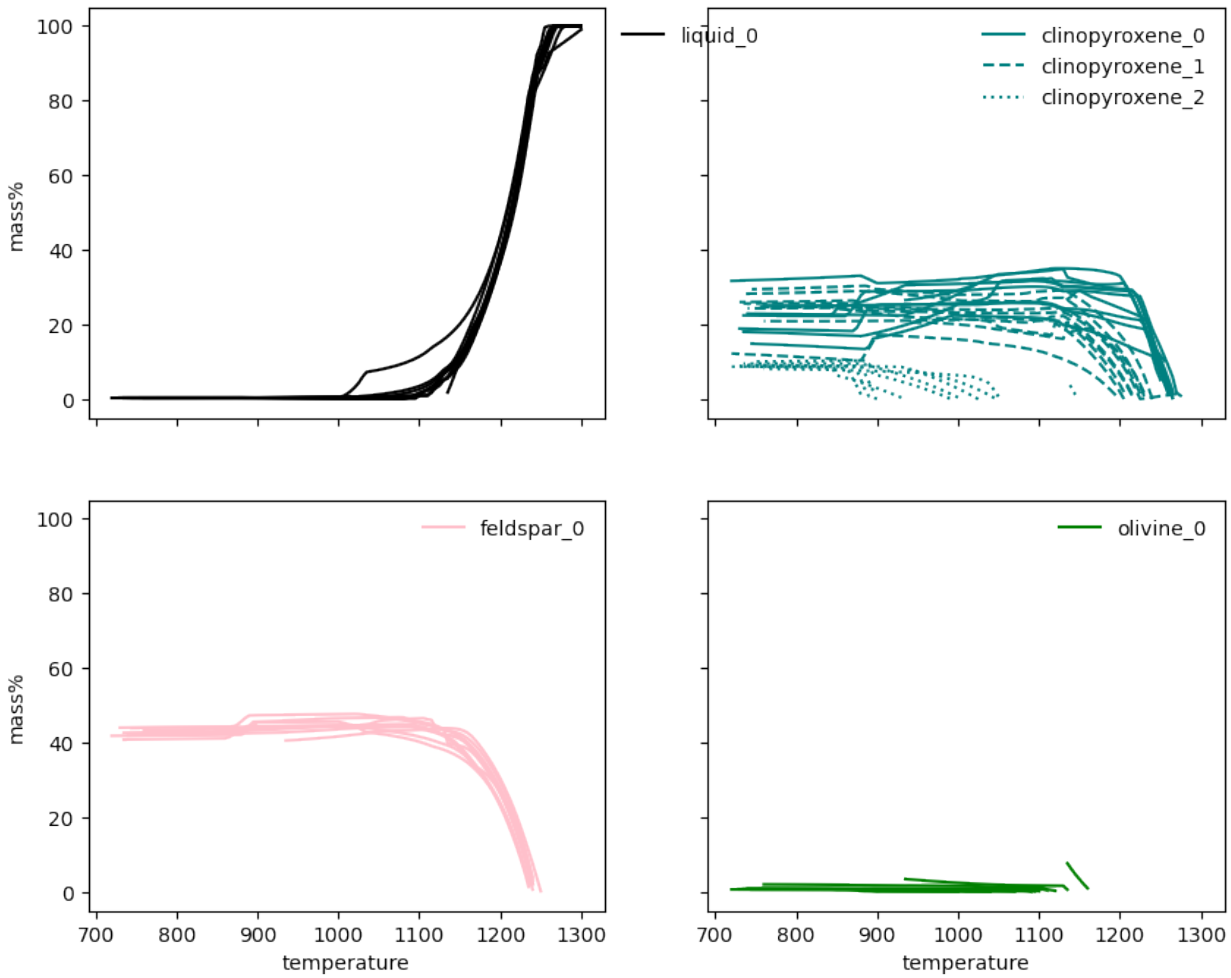
```
# batch.run(overwrite=False)
```

We can first aggregate and import these results:

```
from pyrolite_meltsutil.tables.load import (
    aggregate_tables,
    import_batch_config,
    import_tables,
)

system, phases = aggregate_tables(experiment_dir)  # let's import the tables
cfg = import_batch_config(experiment_dir)  # and also the configuration
```

And now we can visualse these tables. Let's first look at how the relative phase masses change with temperature (i.e. during crystallisation).

```
import matplotlib.pyplot as plt
from pyrolite.util.plot.legend import proxy_line

from pyrolite_meltsutil.vis.style import phase_color, phaseID_linestyle, phaseID_marker

phaselist = ["liquid", "clinopyroxene", "feldspar", "olivine"]

fig, ax = plt.subplots(
    len(phaselist) // 2, 2, sharex=True, sharey=True, figsize=(10, 8)
)
xvar, yvar = "temperature", "mass%"
[a.set_xlabel(xvar) for a in ax[-1, :]]
[a.set_ylabel(yvar) for a in ax[:, 0]]

for p, pax in zip(phaselist, ax.flat):
    pdf = phases.loc[phases.phase == p, :]
    proxies = {}
    for phaseID in pdf.phaseID.unique():
        style = dict(ls=phaseID_linestyle(phaseID), color=phase_color(phaseID))
        for expr in pdf.experiment.unique():
            e_p_df = pdf.loc[((pdf.phaseID == phaseID) & (pdf.experiment == expr)), :]
            pax.plot(e_p_df[xvar], e_p_df[yvar], **style)
            proxies[phaseID] = proxy_line(**style)

    pax.legend(proxies.values(), proxies.keys(), frameon=False, facecolor=None)
```



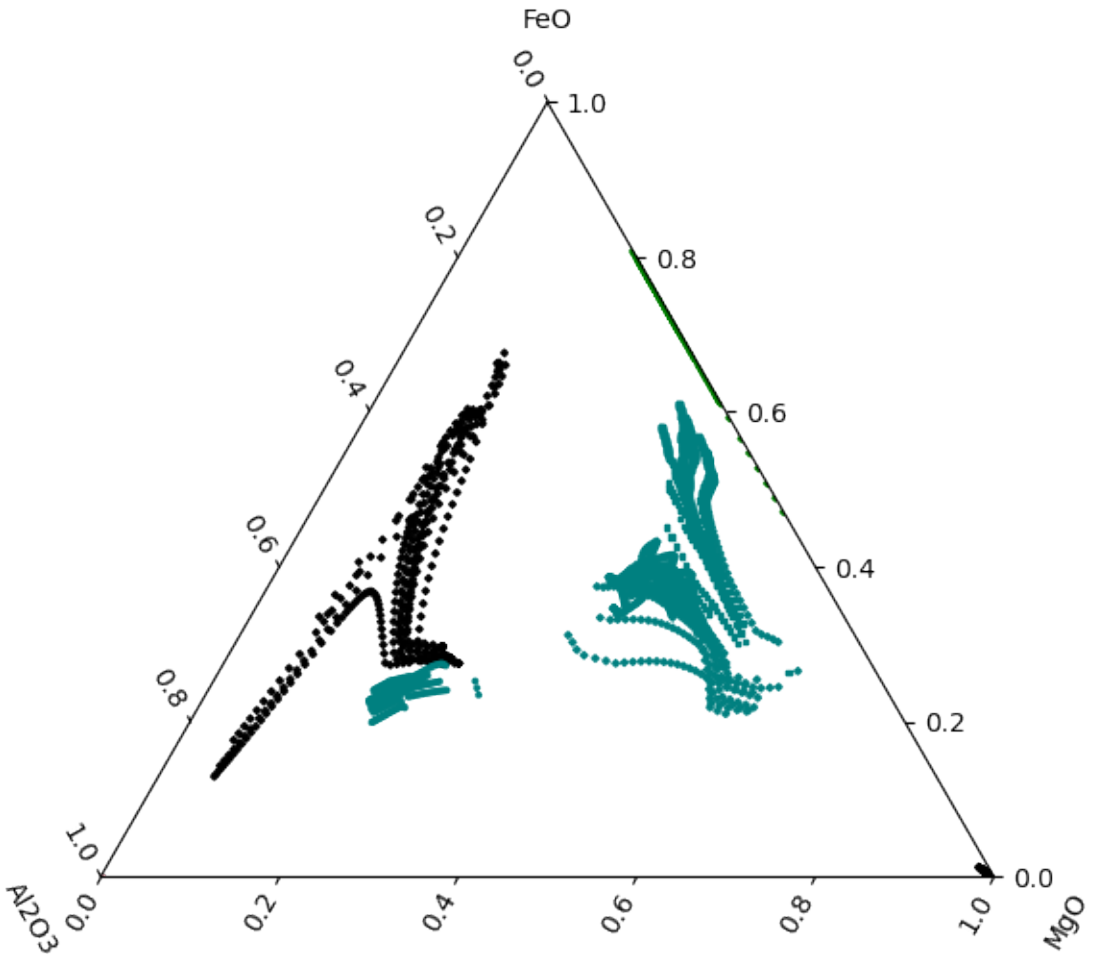

We can also see how variable the chemistry of these phases might be.

```
fig, ax = plt.subplots(1, figsize=(6, 6))
vars = "FeO", "Al2O3", "MgO"

proxies = {}
for p in phaselist:
    pdf = phases.loc[phases.phase == p, :]
    for phaseID in pdf.phaseID.unique():
        style = dict(
            marker=phaseID_marker(phaseID), c=phase_color(phaseID), markersize=3
        )
        proxies[phaseID] = proxy_line(**style)
        for expr in pdf.experiment.unique():
            e_p_df = pdf.loc[((pdf.phaseID == phaseID) & (pdf.experiment == expr)), :]
            e_p_df.loc[:, vars].pyroplot.scatter(s=3, **style, ax=ax)
proxies = {k: proxies[k] for k in sorted(proxies.keys())}
legend = ax.legend(
    proxies.values(),
    proxies.keys(),
    frameon=False,
```

(continues on next page)

(continued from previous page)

```
    facecolor=None,
    bbox_to_anchor=(1, 1),
    loc="upper left",
)
```




**Total running time of the script:** (0 minutes 19.034 seconds)

CHAPTER

FOUR

# API

Utilities for working with the alphaMELTS executable and associated tabular data. Note that these are currently experimental and not affiliated with alphaMELTS. In the future, these utilities will likely also make use of the under-development *python-melts*.

**Todo:**

- As it is developed, also make use of python-melts.
- Develop functions for automation over a grid (of P, T, H2O, fO2, X)
- Have an option to aggregate summary data, and options to discard experiment data
- Expansion of documentation

See the alphaMELTS site for more info[1][2][3][4][5][6][7].

### References

## 4.1 pyrolite_meltsutil.automation

This submodule contains functions for automated execution of alphamelts 1.9.

[1] Ghiorso M. S. and Sack R. O. (1995). Chemical mass transfer in magmatic processes IV. A revised and internally consistent thermodynamic model for the interpolation and extrapolation of liquid-solid equilibria in magmatic systems at elevated temperatures and pressures. Contributions to Mineralogy and Petrology 119, 197–212. doi: 10.1007/BF00307281

[2] Ghiorso M. S., Hirschmann M. M., Reiners P. W. and Kress V. C. (2002). The pMELTS: A revision of MELTS for improved calculation of phase relations and major element partitioning related to partial melting of the mantle to 3 GPa. Geochemistry, Geophysics, Geosystems 3, 1–35. doi: 10.1029/2001GC000217

[3] Asimow P. D., Dixon J. E. and Langmuir C. H. (2004). A hydrous melting and fractionation model for mid-ocean ridge basalts: Application to the Mid-Atlantic Ridge near the Azores. Geochemistry, Geophysics, Geosystems 5. doi: 10.1029/2003GC000568

[4] Smith P. M. and Asimow P. D. (2005). Adiabat_1ph: A new public front-end to the MELTS, pMELTS, and pHMELTS models. Geochemistry, Geophysics, Geosystems 6. doi: 10.1029/2004GC000816

[5] Thompson R. N., Riches A. J. V., Antoshechkina P. M., Pearson D. G., Nowell G. M., Ottley C. J., Dickin A. P., Hards V. L., Nguno A.-K. and Niku-Paavola V. (2007). Origin of CFB Magmatism: Multi-tiered Intracrustal Picrite–Rhyolite Magmatic Plumbing at Spitzkoppe, Western Namibia, during Early Cretaceous Etendeka Magmatism. J Petrology 48, 1119–1154. doi: 10.1093/petrology/egm012

[6] Antoshechkina P. M., Asimow P. D., Hauri E. H. and Luffi P. I. (2010). Effect of water on mantle melting and magma differentiation, as modeled using Adiabat_1ph 3.0. AGU Fall Meeting Abstracts 53, V53C-2264.

[7] Antoshechkina P. M. and Asimow P. D. (2010). Adiabat_1ph 3.0 and the MAGMA website: educational and research tools for studying the petrology and geochemistry of plate margins. AGU Fall Meeting Abstracts 41, ED41B-0644.

### 4.1.1 Issues

- names are truncated for modifychem melts files?
- need a timeout so processes can keep going, add unfinished experiments to failed list

class pyrolite_meltsutil.automation.**MeltsExperiment**(*name='MeltsExperiment'*, *title='MeltsExperiment'*, *fromdir='./'*, *meltsfile=None*, *env=None*, *timeout=None*, *executable=None*)

Melts Experiment Object. For a single call to melts, with one set of outputs. Autmatically creates the experiment folder, meltsfile and environment file, runs alphaMELTS and collects the results.

---

**Todo:**

- Automated creation of folders for experiment results (see `make_meltsfolder()`)
- Being able to run melts in an automated way (see `MeltsProcess`)
- Compressed export/save function
- Post-processing functions for i) validation and ii) plotting

---

**set_meltsfile**(*meltsfile*, ***kwargs*)

Set the meltsfile for the experiment.

**Parameters**

**meltsfile** (`pandas.Series` | `str` | `pathlib.Path`) – Either a path to a valid melts file, a `pandas.Series`, or a multiline string representation of a melts file object.

**set_envfile**(*env*)

Set the environment for the experiment.

**Parameters**

**env** (`str` | `pathlib.Path`) – Either a path to a valid environment file, a `pandas.Series`, or a multiline string representation of a environment file object.

**run**(*log=False*, *superliquidus_start=True*)

Call 'run_alphamelts.command'.

**cleanup**()

pyrolite_meltsutil.automation.**process_modifications**(*cfg*)

Process modifications to an configuration composition.

**Parameters**

**cfg** (`dict`) – Configuratiion dictionary.

**Returns**

**cfg** – Configuratiion dictionary.

**Return type**

`dict`

class pyrolite_meltsutil.automation.**MeltsBatch**(*comp_df*, *default_config={}*, *config_grid={}*, *fromdir=PosixPath('.')*, *env=None*, *executable=None*, *timeout=None*, *logger=<Logger pyrolite_meltsutil.automation (WARNING)>*)

Batch of *MeltsExperiment*, which may represent evaluation over a grid of parameters or configurations.

**Parameters**

- **comp_df** (pandas.DataFrame) – Dataframe of compositions.
- **default_config** (dict) – Dictionary of default parameters.
- **config_grid** (class:*dict*) – Dictionary of parameters to systematically vary.
- **fromdir** (str | pathlib.Path) – Directory to run the set of experiments from, and where the results of each of the experiments will be saved.
- **env** (str | pathlib.Path | *pyrolite_meltsutil.env.MELTS_Env*) – Environment file to use, if not the default.
- **executable** (str | pathlib.Path) – Path to an executable to use, if not the default (specifically the path to `run_alphamelts.command` or `run_alphamelts.bat`).
- **timeout** (int) – Timeout in seconds after which to try and terminate an experiment.
- **logger** (logging.Logger) – Logger to use for logging output, if not the default.

**Variables**

- **compositions** (list of dict) – Compositions to use for the experiments.
- **configs** (list of dict) – Set of experiment configurations to use.
- **experiments** (list of dict) – Combination of compositions and configurations to generate a 'grid' of experiments.

---

**Todo:**

- Can start with a single composition or multiple compositions in a dataframe
- Enable grid search for individual parameters
- Improved output logging/reporting
- Calculate relative number of calculations to be performed for the est duration

  This is currently about correct for an isobaric calcuation at 10 degree temperature steps over few hundred degrees - but won't work for different T steps.
- Does number precision make a difference?

---

**dump**(*experiments=None*, *to_dir=None*)

Serialize the configuration to a json file.

**Parameters**

- **experiments** (dict) – Dictionary of experiments to be serialized.
- **to_dir** (str | pathlib.Path) – Directory to export file to.

**run**(*overwrite=False*, *exclude=[]*, *superliquidus_start=True*, *timeout=None*, *log=False*)

**cleanup**()

## 4.2 pyrolite_meltsutil.download

alphaMELTS download and installsation utilities.

pyrolite_meltsutil.download.**extract_zip**(*zipfile*, *output_dir*)

Extracts a zipfile without the uppermost folder.

**Parameters**

- **zipfile** (*zipfile object*) – Zipfile object to extract.
- **output_dir** (*str* | *Path*) – Directory to extract files to.

pyrolite_meltsutil.download.**download_melts**(*directory*, *version=None*)

Download and extract melts zip file to a given directory.

**Parameters**

- **directory** (str | pathlib.Path) – Directory into which to extract melts.
- **version** (str) – Which alphamelts version to use. Defaults to latest stable version.

---

**Todo:**

- Check version, enable update-style overwrite

---

pyrolite_meltsutil.download.**install_melts**(*install_dir=None*, *link_dir=None*, *eg_dir=None*, *native=True*, *temp_dir=PosixPath('/home/docs/temp/temp_melts')*, *keep_tempdir=False*, *with_readline=True*, *local=False*, *version=None*)

**Parameters**

- **install_dir** (str | pathlib.Path) – Directory into which to install melts executable.
- **link_dir** (str | pathlib.Path) – Directory into which to deposit melts links.
- **eg_dir** (str | pathlib.Path) – Directory into which to deposit melts examples.
- **native** (bool, True) – Whether to install using python (True) or the perl scripts (windows).
- **temp_dir** (str | pathlib.Path, $USER$/temp/temp_melts) – Temporary directory for melts file download and install.
- **keep_tempdir** (bool, False) – Whether to cache tempoary files and preserve the temporary directory.
- **with_readline** (bool, True) – Whether to also attempt to install with_readline.
- **local** (bool) – Whether to install a version of melts into an auxiliary pyrolite data folder. This will override

## 4.3 pyrolite_meltsutil.env

alphaMELTS environment managment.

pyrolite_meltsutil.env.**output_formatter**(*value*)

Output formatter for environment variable values.

**Parameters**
**value** – Value to format.

**Returns**
Formatted value.

**Return type**
str

class pyrolite_meltsutil.env.**MELTS_Env**(*prefix='ALPHAMELTS_'*, *variable_model=None*)

Melts environment object.

---

**Todo:**

- Implement use as context manager.

---

**export_default_env**()

Parse any environment variables which are already set. Reset environment variables after substituting defaults for unset variables.

**dump**(*unset_variables=True*, *prefix=False*, *cast=<function MELTS_Env.<lambda>>*)

Export environment configuration to a dictionary.

**Parameters**
- **unset_variables** (bool) – Whether to include variables which are currently unset.
- **prefix** (bool) – Whether to prefix environment variables (i.e with ALPHAMELTS_).
- **cast** (callable) – Function to cast environment variable values.

**Returns**
Dictionary of environent variables and their values.

**Return type**
dict

**to_envfile**(*unset_variables=False*)

Create a string representation equivalent to the alphamelts defualt environment file.

**Parameters**
**unset_variables** (bool) – Whether to include unset variables in the output (commented out).

**Returns**
String-representation of the environment which can be writen to a file.

**Return type**
str

## 4.4 pyrolite_meltsutil.meltsfile

Utilities for reading and writing .melts files.

pyrolite_meltsutil.meltsfile.**dict_to_meltsfile**(*d*, *linesep='\n'*, *writetraces=True*, *modes=[]*, *exclude=[]*, ***kwargs*)

Converts a dictionary to a MELTSfile text representation. It requires 'title' and 'initial composition' lines, major elements to be represented as oxides in Wt% and trace elements in µg/g.

**Parameters**

- **d** (dict) – Dictionary to convert to a melts file.
- **linesep** (str) – Line separation character.
- **writetraces** (bool) – Whether to include traces in the output file.
- **modes** (list) – List of modes to use (e.g. 'isobaric', 'fractionate solids').
- **exclude** (list) – List of chemical components to exclude from the meltsfile.

**Returns**
String representation of the meltsfile, which can be immediately written to a file object.

**Return type**
str

### Notes

- **Some of the parameters are one-to-many, including modes, phase fractionation,** supression and coexist-limits.

---

**Todo:**

- Parameter validation.

---

pyrolite_meltsutil.meltsfile.**ser_to_meltsfile**(*ser*, *linesep='\n'*, *writetraces=True*, *modes=[]*, *exclude=[]*, ***kwargs*)

Converts a series to a MELTSfile text representation. It requires 'title' and 'initial composition' lines, major elements to be represented as oxides in Wt% and trace elements in µg/g.

**Parameters**

- **ser** (pandas.Series) – Series to convert to a melts file.
- **linesep** (str) – Line separation character.
- **writetraces** (bool) – Whether to include traces in the output file.
- **modes** (list) – List of modes to use (e.g. 'isobaric', 'fractionate solids').
- **exclude** (list) – List of chemical components to exclude from the meltsfile.

**Returns**
String representation of the meltsfile, which can be immediately written to a file object.

**Return type**
str

**Todo:**

- Parameter validation.

pyrolite_meltsutil.meltsfile.**df_to_meltsfiles**(*df*, *linesep='\n'*, ***kwargs*)

Creates a number of melts files from a dataframe.

**Parameters**

- **df** (pandas.DataFrame) – Dataframe from which to take the rows and create melts files.
- **linesep** (str) – Line separation character.

**Returns**
List of strings which can be written to file objects.

**Return type**
list

pyrolite_meltsutil.meltsfile.**from_meltsfile**(*filename*)

Read from a meltsfile into a pandas.DataFrame.

**Parameters**
**filename** (str | pathlib.Path | io.BytesIO) – Filename, filepath or bytes object to read from.

**Returns**
Dataframe containing meltsfile parameters.

**Return type**
pandas.DataFrame

## 4.5 pyrolite_meltsutil.parse

Parsing utilities for use with alphaMELTS.

pyrolite_meltsutil.parse.**read_meltsfile**(*meltsfile*, ***kwargs*)

Read in a melts file from a Series, Path or string.

**Parameters**
**meltsfile** (pandas.Series | str | pathlib.Path) – Either a path to a valid melts file, a pandas.Series, or a multiline string representation of a melts file object.

**Returns**

- **file** (str) – Multiline string representation of a meltsfile.
- *path* – Path to the original file, if it exists.

#### Notes

This function deconvolutes the possible ways in which one can pass either a meltsfile, or reference to a meltsfile.

pyrolite_meltsutil.parse.**read_envfile**(*envfile*, ***kwargs*)

Read in a environment file from a MELTS_Env, Path or string.

**Parameters**
**envfile** (MELTS_Env | str | pathlib.Path) – Either a path to a valid environment file, a pandas.Series, or a multiline string representation of a environment file object.

**Returns**

- **file** (str) – Multiline string representation of an environment file.
- *path* – Path to the original file, if it exists.

pyrolite_meltsutil.parse.**from_melts_cstr**(*composition_str*, *formula=True*)

Parses melts composition strings to dictionaries or formulae.

**Parameters**

- **composition_str** (str) – Composition to parse.
- **formula** (bool) – Whether to output a periodictable.formula.Formula

**Returns**
Dictionary containing components, or alternatively if formula = True, a Formula representation of the composition.

**Return type**
dict | periodictable.formulas.Formula

---

**Todo:**

- Enable parsing of nested brackets in composition.

---

## 4.6 pyrolite_meltsutil.tables

Utilities for reading alphaMELTS table outputs.

## 4.7 pyrolite_meltsutil.vis

Visualisation utilities for working with alphaMELTS table outputs.

---

**Todo:**

- Pull out mineral compostional trends

---

## 4.8 pyrolite_meltsutil.util

Utility functions for pyrolite-meltsutil.

CHAPTER

FIVE

# DEVELOPMENT

## 5.1 Development History and Planning

- Changelog
- Future

## 5.2 Contributing

- Contributing
- Contributors
- Code of Conduct

## 5.3 Development Installation

To access and use the development version, you can either clone the repository or install via pip directly from GitHub:

```
pip install git+git://github.com/morganjwilliams/pyrolite-meltsutil.git@develop
↪#egg=pyrolite_meltsutil
```

## 5.4 Tests

If you clone the source repository, unit tests can be run using pytest from the root directory after installation with development dependencies (`pip install -e .[dev]`):

```
python setup.py test
```

If instead you only want to test a subset, you can call `pytest` directly from within the pyrolite-meltsutil repository:

```
pytest ./test/<path to test or test folder>
```

CHAPTER

SIX

# CHANGELOG

All notable changes to this project will be documented here.

## 6.1 Development

**Note:** Changes noted in this subsection are to be released in the next version. If you're keen to check something out before its released, you can use a development install.

## 6.2 0.1.6

- Minor import upgrades for compatibility with `pyrolite` v0.2.7
- Minor bugfix for environment validation.

## 6.3 0.1.5

- MELTS web service functionality archived.
- Updated data examples within `pyrolite_meltsutil.data.data_examples`

### 6.3.1 `pyrolite_meltsutil.tables`

- Bugfix for duplicated columns ('logfO2(absolute)').
- Added 'alloy-solid' to bug phases where 'structure' is not included as a header name.
- Bugfix for non-numeric columns produced via `import_tables` (which was breaking `plot_sulfur_saturation_point`)

### 6.3.2 pyrolite_meltsutil.vis

- Updated relevant visualisation docs examples to refer to the `((ax, proxies))` syntax where `templates` are used
- pandas-1.0 related bugfix for `templates` (#6).

### 6.3.3 pyrolite_meltsutil.util

- pandas-1.0 related bugfix for cumulate integration functions (#6).

## 6.4 0.1.4

- Web tests switched off, as the MELTS web service appears to be retired.

### 6.4.1 pyrolite_meltsutil.tables

- Added note to `integrate_solid_composition()` to assert that it's for use with singular experiments, not aggregations of experiments.

### 6.4.2 pyrolite_meltsutil.vis

- Updated `pyrolite_meltsutil.vis.templates` functions to return a tuple of axes and proxies (`(ax, proxies)`) for easier legend creation. The proxies object is a dictionary of `phaseID` : `{**styling}`.
- Updated `pyrolite_meltsutil.vis.style` functions to have expanded sets of linestyles/markers for plotting up to 8 different series (note that half of these would be duplicates)

### 6.4.3 pyrolite_meltsutil.util

- Added `log` for consistent logging handling.
- Move `get_process_tree()` and `check_perl()` over from `pyrolite`

## 6.5 0.1.3

- Updated *from_melts_cstr()* to ignore vacancies (`[]`)

### 6.5.1 pyrolite_meltsutil.automation

- Updated `exp_hash()` for consistent naming by sorting dictionary keys before taking a hash of a json-encoded configuration
- Update chemistry modifications to better handle `np.nan` in compositions, where it is now replaced by zero.
- Directory keyword argument updated to `fromdir` from `dir`, to avoid any potential conflict with the python function.
- Updated `make_meltsfolder()` directory keyword argument to `indir` from `dir`

### 6.5.2 pyrolite_meltsutil.tables

- Table files now checked for inconsistent line lengths before import into `pandas.DataFrame`. Bug fix to deal with alphaMELTS omitting a column header for 'structure' for specific minerals (here nepheline and kalsilite)
- Bug fix for dealing with duplicated column headers (specifically, this is typically 'logfO2(absolute)')

### 6.5.3 pyrolite_meltsutil.vis

- Updated templates to make plots including missing intervals.

### 6.5.4 pyrolite_meltsutil.util

- Integrate solids updated to `integrate_solid_composition()`
- Added `integrate_solid_proportions()` for integrating mineral mass proportions.
- Updated indexing for cumulate integration functions to include all experiment steps

## 6.6 0.1.2

- Updated data examples under `pyrolite_meltsutil.data.data_examples`
- Added DOI badge to readme.

### 6.6.1 pyrolite_meltsutil.automation

- Added *process_modifications()* to deal with modifications to config (e.g. chemistry), and moved modifications such that they're included in the experiment configuration grid before it's serialized.
- Bugfix for config serialization

### 6.6.2 pyrolite_meltsutil.tables

- Added try-except loop to deal with missing experiment files (e.g. if an experiment failed to run in the middle of an set of experiments)
- Improved error handling for bad tables

## 6.7 0.1.1

- Bugfix for *pyrolite_meltsutil.util*
- Fixed broken link on docs index page
- Removed support for Python 3.5

## 6.8 0.1.0

- Added `pyrolite_meltsutil.data`
- Data examples of finished experiments added to `pyrolite_meltsutil.data.data_examples`
- Updated automated docs example
- Added documentation example table styling with custom CSS
- Updated *pyrolite_meltsutil.env* to use data via `pyrolite_meltsutil.data.environment`
- Updated meltsfile export utility to be able to export variables encoded as lists, sets or tuples within singular `pandas.DataFrame` columns
- Fixed a parsing issue for *pyrolite_meltsutil.parse.from_melts_cstr()* to deal with NaN/0.0/-0.0

### 6.8.1 pyrolite_meltsutil.automation

- Split out *automation* into submodule and organised files ( `naming`, `org`, `process`, `timing`)
- Added timeouts for automated experiments within `MeltsProcess`
- Started using hashes of configuration for indexing experiments to identify which are identical and avoid duplication (`exp_hash`, `exp_name`)
- Split out the indexes of the experiment grid (`configs` & `composition`, which together form a grid of `experiments`)
- Made sure that experiment grids contain unique experiments - i.e. no duplication.
- Added `pyrolite_meltsutil.automation.MeltsExperiment.dump()` to serialize configuration for a series of experiments.

### 6.8.2 pyrolite_meltsutil.tables

- Updated table read functions
- Converted tables to a submodule including `load`: and `util`
- Added `convert_thermo_names()` to convert with single-letter thermodynamic parameter names (including V/volume, which would conflict with vanadium, S/entropy which would conflict with sulfur and H/enthalpy which could potentially conflict with hydrogen).
- Added `aggregate_tables()` to aggregate all experiments within a directory to a single `DataFrame`
- Defaults updated to lowercase column names.
- Added `import_batch_config()` for importing configurations exported on run, in order to use relevant metadata.
- Bugfixes for inconsistent table widths with specific phases, where a column name is not added for `structure` (nepheline, kalsilite, alloys)
- Added `read_phase_table()` for reading in phase tables.
- Added `phasetable_from_phasemain()` and `phasetable_from_alphameltstxt()` for reading phase tables from the *phasemain.txt* and *alphaMELTS_tbl.txt* files, respectively
- Added automatic detection of fractionation (i.e. where experiment mass changes beyond a threshold)
- Updated table percentages to be formatted as 0-100% (rather than fractional 0-1.)

### 6.8.3 pyrolite_meltsutil.vis

- Added submodule for visualisation components
- Added styling functions in `style`
- Added SCSS function in `scss`
- Added `plot_xy_phase_groupby()` and the convenience functions `plot_phasevolumes()` and `plot_phasemasses()`
- Added `phaseID_marker()` and updated `pyrolite_meltsutil.vis.style.phaseID_linestyle()` for modulating styling based on ID.

### 6.8.4 pyrolite_meltsutil.util

- Added `pyrolite_meltsutil.util.general.pyrolite_meltsutil_datafolder()` to identify the relevant data folder.
- Added `pyrolite_meltsutil.util.synthetic.isobaricGaleMORBexample()` for generating a `DataFrame` based on the Gale (2013) MORB dataset for general use with *pyrolite_meltsutil*.
- Added `get_local_example()` for loading examples installed with `alphaMELTS`, and `get_local_link()` for identifying the link files created upon `alphaMELTS` installation.
- Added `get_data_example()` to get the folder of an example already-finished experiment folder

## 6.9 0.0.2

- Split out the `pyrolite-meltsutil` project from `pyrolite`
- Updated and refactored documentation

CHAPTER

SEVEN

# FUTURE

This page details some of the under-development and planned features for `pyrolite-meltsutil`. Note that while no schedules are attached, features under development are likely to be completed with weeks to months, while those 'On The Horizon' may be significantly further away (or in some cases may not make it to release).

## 7.1 Under Development

These features are under development and should be released in the near future.

## 7.2 On the Horizon, Potential Future Updates

These are a number of features which are in various stages of development, which are planned be integrated over the longer term.

CHAPTER

EIGHT

# CODE OF CONDUCT

## 8.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

## 8.2 Our Standards

Examples of behaviour that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behaviour by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

## 8.3 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behaviour and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behaviour.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviours that they deem inappropriate, threatening, offensive, or harmful.

## 8.4 Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

## 8.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behaviour may be reported by contacting the project admins. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

## 8.6 Attribution

This Code of Conduct is adapted from the Contributor Covenant Version 1.4, available at https://www.contributor-covenant.org/version/1/4/code-of-conduct.html. The Contributor Covenant is released under the Creative Commons Attribution 4.0 License.

For answers to common questions about this code of conduct, see https://www.contributor-covenant.org/faq.

CHAPTER

NINE

# CONTRIBUTING

The long-term aim of this project is to be designed, built and supported by (and for) the geochemistry community. In the present, the majority of the work involves incorporating geological knowledge and frameworks into a practically useful core set of tools which can be later be expanded. As such, requests for features and bug reports are particularly valuable contributions, in addition to code and expanding the documentation. All individuals contributing to the project are expected to follow the Code of Conduct, which outlines community expectations and responsibilities.

Also, be sure to add your name or GitHub username to the contributors list.

**Note:** This project is currently in *beta*, and as such there's much work to be done.

## 9.1 Feature Requests

If you're new to Python, and want to implement a specific process, plot or framework as part of *pyrolite_meltsutil*, you can submit a Feature Request. Perhaps also check the Issues Board first to see if someone else has suggested something similar (or if something is in development), and comment there.

## 9.2 Bug Reports

If you've tried to do something with *pyrolite_meltsutil*, but it didn't work, and googling erro messages didn't help (or, if the error messages are full of `pyrolite_meltsutil.XX.xx`), you can submit a Bug Report . Perhaps also check the Issues Board first to see if someone else is having the same issue, and comment there.

## 9.3 Contributing to Documentation

The documentation and examples for *pyrolite_meltsutil* are gradually being developed, and any contributions or corrections would be greatly appreciated. Currently the examples are patchy, and a 'getting started' guide would be a helpful addition. If you'd like to edit an existing page, the easiest way to get started is via the 'Edit on GitHub' links:

**These pages serve multiple purposes:**

- A human-readable reference of the source code (compiled from docstrings).
- A set of simple examples to demonstrate use and utility.
- A place for developing extended examples[1]

[1] Such examples could easily be distributed as educational resources showcasing the utility of programmatic approaches to geochemistry

## 9.4 Contributing Code

Code contributions are always welcome, whether it be small modifications or entire features. As the project gains momentum, check the Issues Board for outstanding issues, features under development. If you'd like to contribute, but you're not so experienced with Python, look for `good first issue` tags or email the maintainer for suggestions.

To contribute code, the place to start will be forking the source for `pyrolite-meltsutil` from GitHub. Once forked, clone a local copy and from the repository directory you can install a development (editable) copy via `python setup.py develop`. To incorporate suggested changes back to into the project, push your changes to your remote fork, and then submit a pull request onto pyrolite-meltsutil/develop .

---

**Note:**

- See Installation for directions for installing extra dependencies for development, and Development for information on development environments and tests.
- `pyrolite-meltsutil` development roughly follows a gitflow workflow. `pyrolite-meltsutil/master` is only used for releases, and large separable features should be build on `feature` branches off `develop`.
- Contributions introducing new functions, classes or entire features should also include appropriate tests where possible (see *Writing Tests*, below).
- `pyrolite-meltsutil` uses Black for code formatting, and submissions which have passed through `Black` are appreciated, although not critical.

---

## 9.5 Writing Tests

There is currently a broad unit test suite for `pyrolite-meltsutil`, which guards against breaking changes and assures baseline functionality. `pyrolite-meltsutil` uses continuous integration via Travis, where the full suite of tests are run for each commit and pull request, and test coverage output to Coveralls.

Adding or expanding tests is a helpful way to ensure `pyrolite-meltsutil` does what is meant to, and does it reproducibly. The unit test suite one critical component of the package, and necessary to enable sufficient trust to use `pyrolite-meltsutil` for scientific purposes.

CHAPTER

TEN

# CONTRIBUTORS

This list includes people who have contributed to the project in the form of code, comments, testing, bug reports, feature requests.

- Morgan Williams
- Louise Schoneveld
- Yajing Mao

CHAPTER

# ELEVEN

# CITATION

**Note:** `pyrolite` began as a personal project, but as the project has developed others have since contributed. Check the Contributors list here for `pyrolite-meltsutil` and also the `pyrolite` contributors list and add major contributors to citations where appropriate.

If you use `pyrolite` extensively for your research, citation of the software would be particularly appreciated. It helps quantify the impact of the project (assisting those contributing through paid and volunteer work), and is one way to get the message out and help build the pyrolite community.

For notes on citation, please see the pyrolite citation page.

**Note:** This documentation is a work in progress and is updated regularly. Contact the maintainer with any specific questions/requests.

CHAPTER

TWELVE

# REFERENCES

# PYTHON MODULE INDEX

## p

# INDEX

## S



## T