VarPy: A Python library for volcanology and rock physics data analysis

Rosa Filgueira and Malcolm Atkinson
School of Informatics
University of Edinburgh
Edinburgh EH8 9AB, UK
{rfilguei, mpa}@inf.ed.ac.uk

Andrew Bell and Ian Main
School of GeoSciences
University of Edinburgh
Edinburgh EH8 9XP, UK
{a.bell, ian.main}@ed.ac.uk

Branwen Snelling
Department of Earth Sciences
University of Oxford
Oxford OX1 3AN, UK
{branwen.snelling}@univ.ox.ac.uk

***Abstract*—VarPy is an open-source toolbox which provides a Python framework for analysing volcanology and rock physics data. It provides several functions, which allow users to define their own workflows to develop models, analyses and visualisations. The goal of the VarPy library is to accelerate the uptake of computational methods by researchers in volcanology and rock physics. It does this via two mechanisms:**

- **supplying a library of ready-made functions that are generally useful; and**
- **providing a context for creating, sharing and comparing additional functions.**

We anticipate two groups of VarPy users:

- **the majority who use functions already written and in the library; they will predominantly arrange to use sequences of these functions with their own parameterisations; and**
- **contributors, who, as well as using provided functions, also want to write additional functions for their own use or to add to the library.**

Keywords*—*Python library, programming frameworks, volcanology and rock physics data analysis, seismicity deformation data, e-Infrastructures, scientific workflows.

I. Introduction

The increasing prevalence of digital instrumentation in volcanology and rock physics is leading to a wealth of data, which in turn is increasing the need for computational analyses and models. Today, these are largely developed individually by each researcher. The introduction of a shared library that can be used for this purpose has several benefits:

- when an existing function in the library meets a need recognised by a researcher it is usually much less effort to re-use it than develop ones own code;
- once functions are established and multiply used they become better tested, more reliable and eventually trusted by the community;
- use of the same functions by different researchers makes it easier to compare results and to compare the skill of rival analysis and modelling methods; and
- in the longer term the cost of maintaining these functions is shared over a wider community and they therefore have greater duration.

Python is a high-level interpreted programming language, with capabilities for object-oriented programming. Although, there are many software tools available for interactive data analysis and development, there are no libraries designed specifically for volcanology and rock physics data. Therefore, we propose a new Python open-source toolbox called VarPy[1] to facilitate rapid application development for rock physicists and volcanologists, which allows users to define their own workflows to develop models, analyses and visualisations.

VarPy is focussed on analyzing seismicity and deformation data (such as stress and strain measurements). Although different in detail, the attributes of volcanic and rock phyiscs data have many commonalities, hence the desire for a shared library. We have taken the library style from the *ObsPy* [1] library, which is very successful with seismologists[2].

This proposal is triggered by our work on data assimilation in the EFFORT (Earthquake and Failure Forecasting in Real Time)[3] project using data provided by the NERC CREEP-2[4] experimental project and as a test cases.

The library must fulfil two purposes simultaneously:

- by providing a full repertoire of commonly required actions it must make it easy for volcanologists and rock physicists to write the Python scripts they need to accomplish their work, and;
- by wrapping operations it must enable the EFFORT gateway to maintain the integrity of its data.

The paper is organised as follows. Section II explains why we have selected Python for developing VarPy . Section III presents the EFFORT project. Section IV explains the data structures and main classes of VarPy. Section V presents VarPy packages and modules. Section VI describes the types of methods of VarPy and the algebra of operations. Sections VII and VIII present two examples of using VarPy. Finally, Section IX presents our conclusions and provides pointers for future work.

II. Why Python ?

Python is a high-level programming language, interpreted with capabilities for object-oriented programming. Python has

[1] https://bitbucket.org/effort/varpy/src
[2] http://docs.obspy.org
[3] http://www.effort.is.ed.ac.uk
[4] http://www.ucl.ac.uk/es/research/ripl/research/rock-physics

a simple, easy to learn syntax, emphasises readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. Often scientists choose this language to program their programs because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Besides, Python has very rich scientific computing libraries such as:

- *ObsPy*: Python library dedicated to provide a framework for processing seismological data.
- *SymPy*: Python library for symbolic mathematics
- *GeoPy*: Python Geocoding Toolbox
- *NumPy*: Python library for array processing for numbers, strings, records, and objects
- *SciPy*: Python library of algorithms and mathematical tools

More libraries can be found in[5]. As we mentioned above, there are not any libraries designed for working with volcanology and rock physics data. Therefore, our objective is to help scientists who write methods, or components to be used in methods, that will then be used in their own computers and/or the EFFORT gateway. Notice that proposal of the `VarPy` library does not attempt to replace functions provided by other libraries, such as *NumPy* and *SciPy*. `VarPy` is complementary to them.

III. EFFORT PROJECT

EFFORT [2] is a multi-disciplinary collaboration between analytical geoscientists (School of GeoSciences, University of Edinburgh (UoE)), experimental rock physicists (Department of Earth Sciences, University College London (UCL)), and informaticians (School of Informatics, UoE). Brittle rock failure plays a significant role in the timing of a range of geophysical hazards, such as volcanic eruptions; yet the predictability of brittle failure is unknown. EFFORT aims is to provide a facility for developing and testing codes to forecast brittle failure for experimental and natural data. In the project, we have developed the EFFORT gateway, which offers services for collecting and sharing volcanology and rock physics data with the intent of stimulating sharing, collaboration and comparison of methods among the practitioners in the two fields. In EFFORT we collect experimental data from controlled experiments: rock physics experiments from the UCL laboratory, sub-sea deformation experiments in the CREEP-2 project, and volcanic monitoring data from INGV observatory Etna[6] and IGN observatory El Hierro[7]. We also generate synthetic rock physics data to test the algorithms.

The EFFORT gateway, it offers facilities for running analyses and models either under a researcher's control or periodically as part of an experiment and to compare the skills of predictive methods. The gateway therefore runs code on behalf of volcanology and rock physics researchers.

[5]https://pypi.python.org/pypi?:action=browse&show=all&c=385
[6]http://www.ct.ingv.it/en/
[7]http://www.02.ign.es/ign/main/index.do

One motivation for developing `VarPy` is to increase the use of the EFFORT gateway. The `VarPy` library is intended to make it much easier for those researchers to set up the code they need to run. The library also makes it easier to arrange that code is in a form suitable for running in the EFFORT computational services. Care has been taken to ensure that the library can also be used outside of EFFORT systems, *e.g.*, on a researcher's own laptop

Figure 1 shows how the experimental data from observatories and laboratories are transferred to the *EFFORT gateway*. Geoscientists view and analyse data, contribute code, and run models.

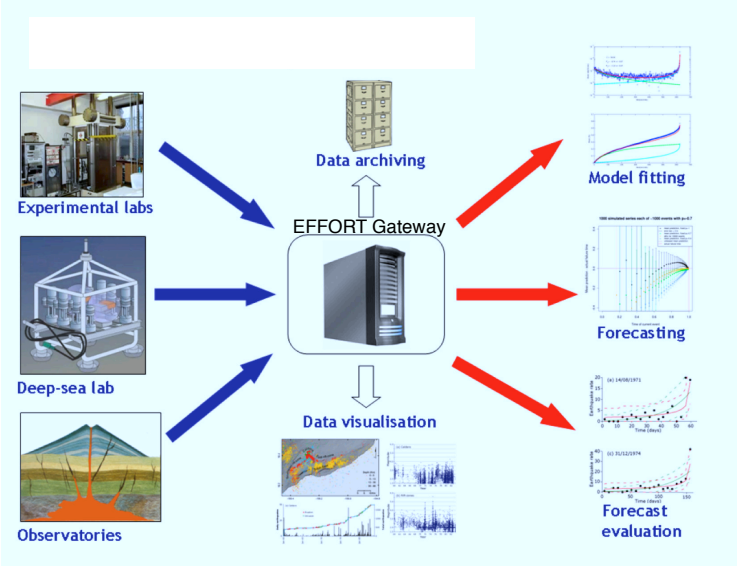

Fig. 1: The EFFORT gateway acts as hub for archiving, analysing and monitoring different types of data during experimental runs.

IV. VarPy DATA STRUCTURES AND CLASSES

Most rock physics and volcanic time-series data belong to one of two different classes: *Event catalogue data* (ECD) and *Sampled continuous data* (SCD). Volcanic observatories and rock physics laboratories can produce data of both classes in a single experiment. ECD consist of a series of events (*e.g.* acoustic emissions, earthquakes, volcanic eruptions) that occur at discrete times and have a specific attributes (*e.g.* location, depth, magnitude, duration). SCD consist of a series of times at which a continuous variable has been measured, and the value of that variable. The sample times are defined by the instruments' operator, and may (or may not) be evenly spaced (*e.g.* daily, every second). Consequently, the ECD and SCD from the rock physics laboratories are different from volcanic observatories. We represent those datasets using four different structures:

- *Event catalogue laboratory data* (ECLD)
- *Event catalogue volcanology data* (ECVD)
- *Sample continuous laboratory data* (SCLD)
- *Sample continuous volcanology data* (SCVD)

A volcanic observation may also have another data structure called *Eruption volcanic data* (EVD), which represents a time-series of volcanic eruptions, intrusions, or other events with descriptions (*e.g.* type, size, duration).

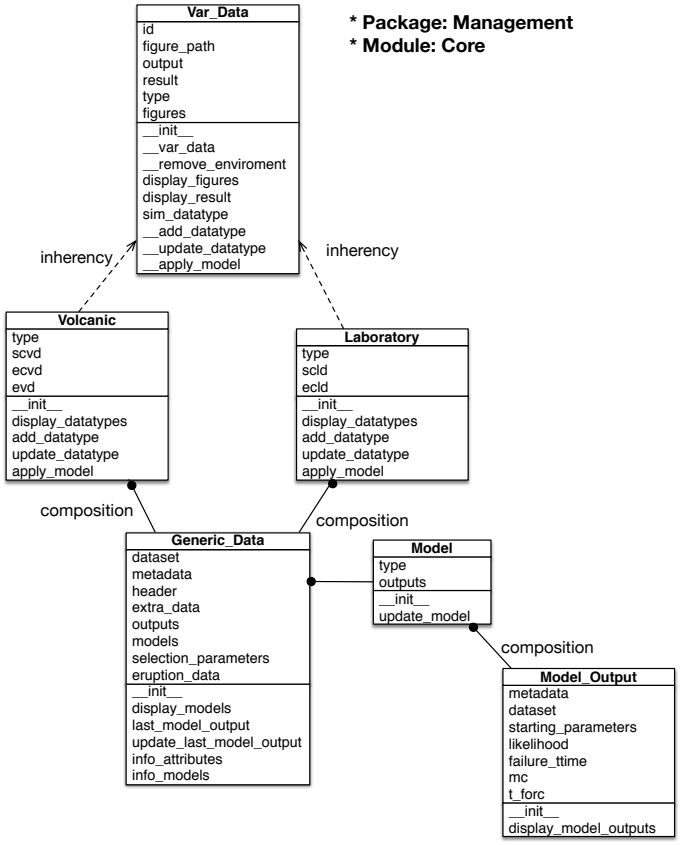

Fig. 2: UML Class Diagram of `VarPy` core classes.

Figure 2 represents the UML class diagram of the *core* module which contains the main classes of `VarPy`. This module belongs to the *management* package (see Table I and Table II). *Var_Data* is an abstract class which contains the common attributes and methods of *Laboratory* and *Volcanic* classes. Users can not create instances of this class. The method *var_data* creates a new *tree-directory* (see Figure 4) to store the input and the output files and figures. This *tree-directory* is created under a parent directory which is by default the current directory. The name of the parent directory is the value specified by the ID parameter. This method is called by *Laboratory* and *Volcanic* classes.

Laboratory and *Volcanic* are classes for handling rock physics laboratory and volcanic observatory data. Both classes, inherits *Var_Data* attributes and methods. *Volcanic* class has three main attributes (*ecvd, scvd* and *evd*), which are composite objects. They store the three data structures that a volcanic observatory could record. Initially, the three composite objects are initialised empty, with out any values in their attributes. Later, the user by using the different methods of this class, can be used to add values to the attributes of those objects. Furthermore, the *scvd* attribute is in fact a dictionary of composite objects, because one volcanic observatory could have more than one sample continuous volcanic data information per station. *Laboratory* class has two main attributes (*ecld, scld*), which are also composite objects and initialised empty. A *type* attributed is used in both classes, for storing the type of object in each case.

Generic_Data is a class which contains several common attributes and methods for the data structures (*ecvd, scvd, evd, ecld*, and *scld*), including *dataset* (array), *metadata* (dictionary), *header* (array), and *models* (dictionary). The *dataset* attribute is used to store information about a data file. The *metadata* attribute is used to store information about a metadata file. The *header* attribute is used to store the names of the columns of the dataset attribute. Finally, the *models* attribute is a composite object used to store the models' outputs. To each dataset of each data structure can be applied different models several times, obtaining each time a new output. Those outputs are represented by the class *Model_Output*. Users can not create instances of *Generic_Data* and *Model_Output* classes.

Figures 3 shows a schema of a *Volcanic* object, where its main attributes are represented.

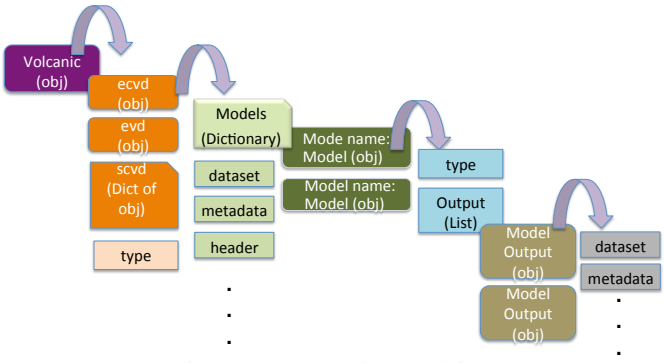

Fig. 3: `VarPy` *Volcanic* object.

In the following example, *d1 Volcanic* object has been created by using the line 5 in Listing 1. This method needs an *ID* as input, which is a string chosen by the user to help them identify their experiments *e.g. Tjornes_ex1*. As an output the method sets up the environment for storing the initial data and metadata information (as Figure 4 shows) and also returns an *d1* object. Line 6 loads the data and metadata into the *ecvd* attribute so it will be useable for analysis.

```
1 from varpy.management import core
2 ID = 'Tjornes_ex1'
3 ecvd_data_file = Iceland_IMO_C1_95.txt
4 ecvd_metadata_file = Iceland_IMO_C1_meta.txt
5 d1 = core.Volcanic(ID)
6 d1.add_datatype( ecvd , ecvd_data_file,
    ecvd_metadata_file)
```

Listing 1: A *Volcanic* object with ECVD attribute.

V. VarPy Packages and Modules

`VarPy` functionality is provided through the packages represented in Table I.

Figure 5 and Table II show the modules for *management*, *data_preparation*, and *analysis* packages. *Managment* package

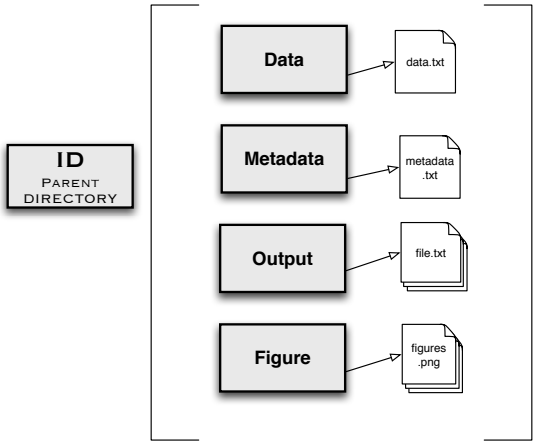

Fig. 4: `VarPy` *tree-directory* environment.

Package	Functionality
varpy.managment	Core classes
varpy.data_preparation	Filtering routines
varpy.analysis	Analyzing filtered data routines
varpy.modelling	Modelling routines
varpy.simulation	Simulating routines for seismic data
varpy.statistics	Statistical routines used by other routines
varpy.visualization	Plotting routines
varpy.write	Writing results routines

TABLE I: General `VarPy` packages.

contains the common methods and classes for `VarPy` (displayed in Figure 2). It includes the *Volcanic* and *Laboratory* classes and methods for reading data and metadata files of the five data structures introduced in Section IV. It also has several methods for converting dates and times into different formats. The *data_preparation* package includes filters of various kinds, like gap-filling, differentiation and integration. The *analysis* package contains functions to analyze the filtered data. Analysis routines include averages based on time, point processes and distributions based on location.

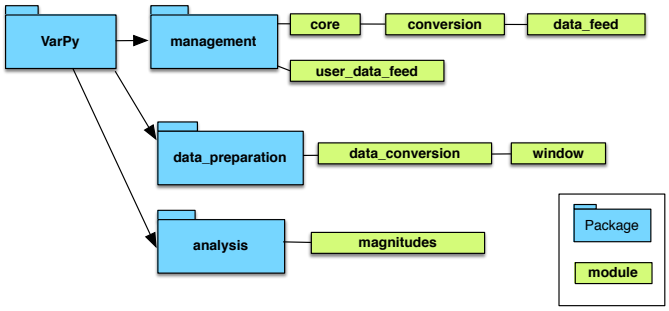

Fig. 5: Modules for *management, data preparation* and *analysis* packages.

Figure 6 and Table III summarise the packages and modules for *modelling* and *simulation* packages. *Modelling* gathers models, like *creep_model*, *exp_mle*, *iol_mle* or *hyp_mle*, and fits them to the filtered data. Those models return different rates

Module	Functionality
core	Handling `VarPy` objects and methods
conversion	Converting dates and times into different formats
data feed & user_data_feed	Importing metadata and storing it into `VarPy` object
data_conversion	Converting values to another type of data
window	Selecting a smaller sample based on a single or on a combination of variables
magnitudes	Calculating completeness magnitude

TABLE II: Description of modules for *management, data preparation* and *analysis* packages.

according to the function applied, *e.g. exp_mle* applies an exponential function to the data. A completely description of those models can be found at [3] and [4] works. *Models_applications* configures experiments that can be performed on the filtered data. We define a `VarPy` experiment as the application of a model to filtered data with some input parameters. `VarPy` has defined two types of experiment:

- *Single analysis*: This applies a model to the filtered data once. It can be classified in two types:
  - *Retrospective analysis*: When the failure (also called eruption) time is known. The output represents how well the model explains the data.
  - *Single forecast*: When the failure time is not known. The output not only explains how well the model fits, it also predicts the failure time.
- *Multiple analysis*: This applies a model to the filtered data several times. It can also be classified in two types:
  - *Prospective forecast*: The failure time is not known, and the output is the prediction of the failure time in real time.
  - *Retrospective forecast*: Although the failure time is known, it is simulated as not known. The output represents the prediction of the failure time, and we can check if the prediction is correct or not with the real failure time.

The *simulation* package has several sub-packages with different types of simulators for creating synthetic data: rock physics seismic data, volcanic seismic data or earthquake data.

Sub-package	Functionality
models	Various models that can be fitted to data
model_application	Experiments with data and models: single analysis and multiple analysis
lab_data	Rock physics seismic data simulators
volcanic_data	Volcanic seismic data simulators
earthquake	Earthquake data simulators

TABLE III: Description of *modelling* and *simulation* sub-packages.

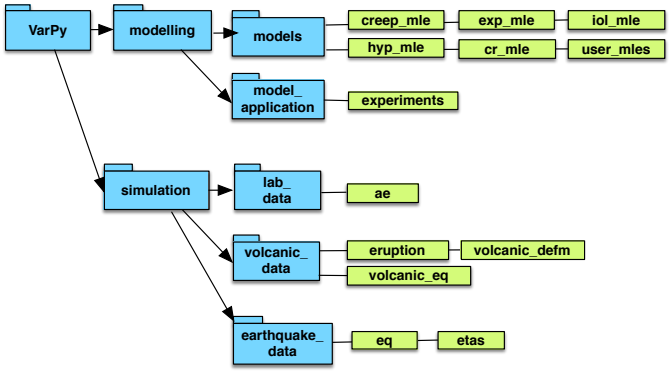

Fig. 6: Sub-packages for *modelling* and *simulation* packages.

Figure 7 and Table IV give an overview of the modules for *statistics, visualization* and *write* packages. *Statistics* includes statistical functions (*e.g. likelihood_functions*) for generating synthetic data and comparing models. It is heavily used by functions in other packages. *Visualization* enables users to plot filtered data, the results of experiments, simulated data and models. It allows multiple fitted models to be plotted against the data or against the experiments. It also plots model comparisons. Finally, *write* package allows users to write the results of experiments into text files. The names of these files are descriptive of the results they contain. Their location is automatically set inside the output directory inside the *tree-directory*.

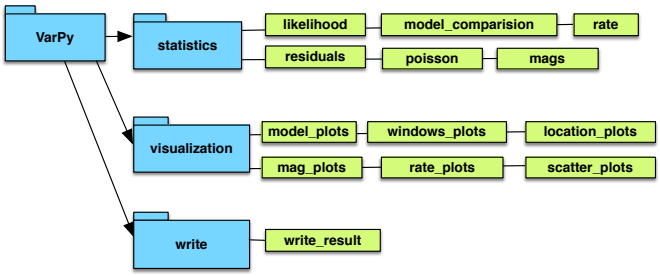

Fig. 7: Modules of *statistics, visualization* and *write* packages.

Package	Functionality
statistics	Assisting with the generation of synthetic data, fitting models and comparing models
visualisation	Plotting filtered data, results of analyses, simulated data and models
write	Writing the results of analysis and experiments into text file inside the *tree-enviroment*

TABLE IV: Description of *statistics, visualization* and *write* packages.

VI. Algebra of Operations

While designing VarPy, we have followed the Python Guide Style[8]. This means that all the names of the packages, modules, methods and parameters are in lower case. And the names of the classes start in upper case. As follows, we are going to describe the six types of methods that we have in our library:

- *M0 method*: Methods for creating either volcanic or laboratory objects. This type of method receives an identifier (ID) as an input data. It creates the new object and the *tree-directory* explained in Section IV.
- *M1 method*: Methods for adding an attribute of an object. This type of method receives data and metadata files as input parameters. Itt modifies the object adding the new attribute and copies the files into the *tree-directory*.
- *M2 method*: Methods for modifying an attribute (or several attributes) of an object. This type of method receives an object as an input parameter. It returns an object which is a (deep) copy of the input object with the new modifications in some attributes but it does not create another *tree-directory*. This type can be located in different packages and modules depending on the aim of the methods.
- *M3 method*: Methods for transforming data. This type of method receives some data as input parameter. It returns the transformed input data as an output parameter. This type of method does not modify any attributes of any object. They are called from other types of methods for transforming input parameters, or variables.
- *M4 method*: Methods for writing an attribute of an object. This type of method writes into a file one attribute of an object. The file is stored inside the *tree-directory*. This type of method is implemented inside the *write* package, inside the *write_result* module. The unique input parameter that this methods needs is the object.
- *M5 method*: Methods for plotting an attribute of an object. This type of method plots into a figure an attribute of an object. The figure is stored inside the *tree-directory*. This type of method is implemented inside the *visualization* package. They need an attribute as an input parameter

The idea behind this classification of methods is to be able to build an algebra of operations as shown in Figure 8. In the example, we create a new object called *object1* applying *k0* method which belongs to *M0 method* type. To *object1* we apply *k1* method for adding an attribute. This attribute needs to be modified. Therefore, *k2* method is called, which belongs to *M2 method type*. Besides, we call from *k2* method to *k3* method for transforming some parameters. As a result of *k2* method a new object called *object2* is created. *object2* has initially the same attributes as *object1*. To *object2* we apply the methods *k4* , *k7* and *k8*. The method *k4* belongs to *M2 method* type. As a results of *k4* method we have a new object called *object3* with

[8]http://www.python.org/dev/peps/pep-0008/

the new updates in the attributes. Applying the method *k5* to *object3* we write one of the attributes into a file inside the *tree-directory*. Applying the method *k6* to *object3* we plot a figure with the values of some attributes of this object. When we apply the method *k7*, by using *object2*, we can still write into a file the values of the attributes of *object2*. In the same way, we apply the method *k8* by using *object2*. Therefore, every time that we apply an *M2 method* over an object, we have a new one with updates applied to some attributes, and without modifying the original object. This allow us, to always keep the previous status of the object when we want to perform an update to one attribute of an object.

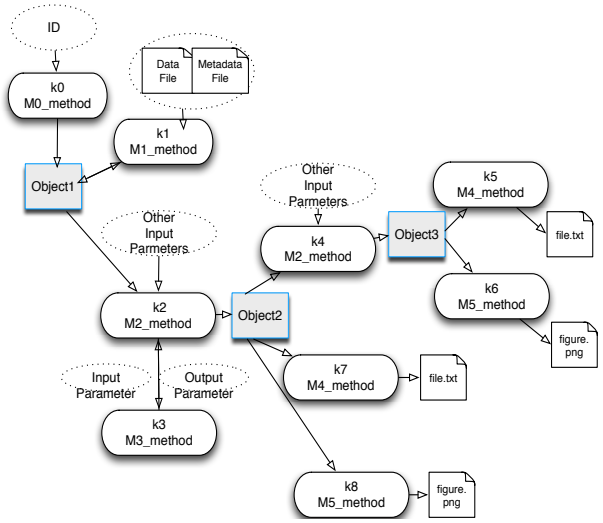

Fig. 8: Algebra of operations.

VII. Exploration and Visualization of Data

Through the following example, we explain how a user can perform data exploration and visualization based on the *Tjornes fracture zone* (Iceland) by using `VarPy`. This example is the continuation of the Listing 1 shown in Section IV. We assume that an *d1* object has been already created and seismicity data added as "earthquake catalogue volcanic data" (*ecvd* attribute).

First, *d1* object is filtered by selecting latitude and longitude for events located in Iceland (see Listing 2).

```
#continuation of  Listing 1
#Import more needed libraries
from varpy.data_preparation import window
from varpy.visualisation import rate_plots,
    mag_plots, iet_plots, map_plots,
    scatter_plots
#Boundary coordinates for Iceland
lon_min = -25.5
lon_max = -12.0
lat_min = 62.7
lat_max = 67.5
d1 = window.latlon(d1, lat_min, lat_max,
    lon_min, lon_max)
d1 = window.single_attribute(d1, 'depth', -5,
    50.0, 'ecvd')
d1 = window.single_attribute(d1, 'magnitude',
    -1.0, 10.0, 'ecvd')
```

Listing 2: Filtering data: Events located Iceland.

Later, a new *d2* object is created after selecting a smaller sample from the *d1* object based on the *magnitude* variable (see line 1 from Listing 3). Then, the regional epicentre map is shown (see line 2 from Listing 3 and Figure 9).

```
d2=window.single_attribute(d1, 'magnitude',
    3.0, 10.0, 'ecvd')
map_plots.plot_map(d2)
```

Listing 3: Show regional epicentre map of larger events.

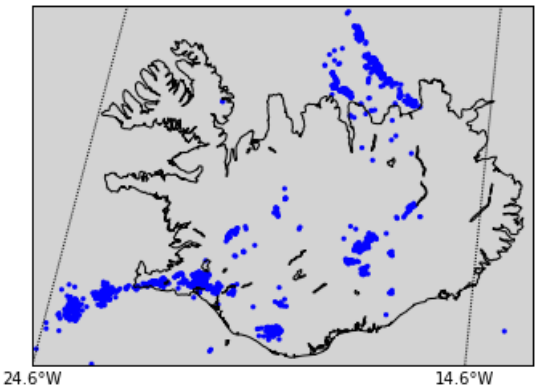

Fig. 9: Regional epicentre map of large events: listing 3, line 2.

A new *d3* object is created after filtering *d1* data by selecting a sub-set of events occurring in the *Tjornes fracture zone* by latitude and longitude (see Listing 4)

```
#Boundary coordinates for Tjornes zone
lat_min_r1 = 65.5
lat_max_r1 = 67.5
lon_min_r1 = -20.0
lon_max_r1 = -16.0
d3 = window.latlon(d1, lat_min_r1, lat_max_r1,
     lon_min_r1, lon_max_r1)
```

Listing 4: Filtering data: Sub-set of events by latitude and longitude.

The next step is to check the quality of the *d3* object by examining the earthquake magnitude-frequency distribution as Figures 10, 11 and 12 show. These are basic plots in the style that has become normal in the preliminary assessment of seismicity, so patterns in space and time can be quickly detected, and the scaling relations and probabilities of different magnitude events can be observed.

```
#Plot the magnitude time series
mag_plots.mag_mc_plot(d3)
#Plot the magnitude-frequency distribution
mag_plots.mf_plot(d3)
#Plot the stability as a function of cut-off
    magnitude
mag_plots.bstab_plot(d3)
```

Listing 5: Quality check.

Finally, data exploration is performed on the *d3* object to see wether something interesting has been happening in the last two years (see Listing 6), by selecting a sub-set time period, and examining the time-series of magnitudes (see Figure 13) and the earthquake rates (see Figure 14).

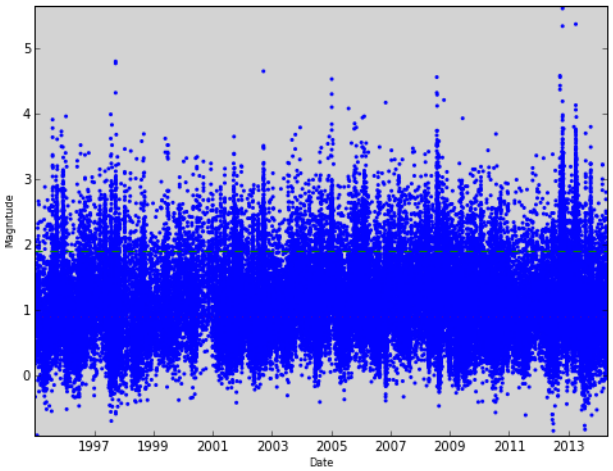

Fig. 10: Magnitude time-series showing completeness magnitudes: listing 5, line 2.

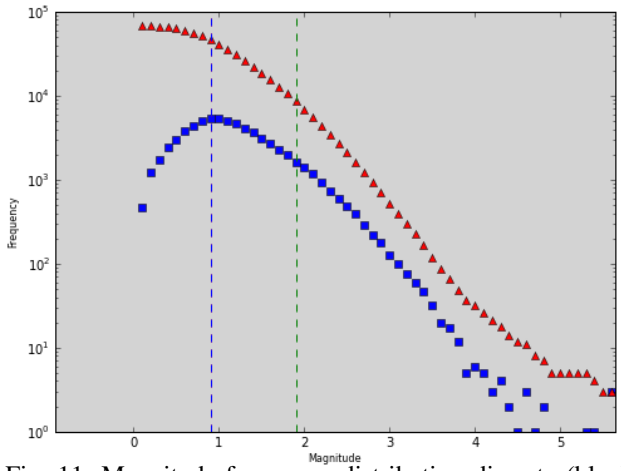

Fig. 11: Magnitude-frequency distribution discrete (blue) and cumulative (red): listing 5, line 4.

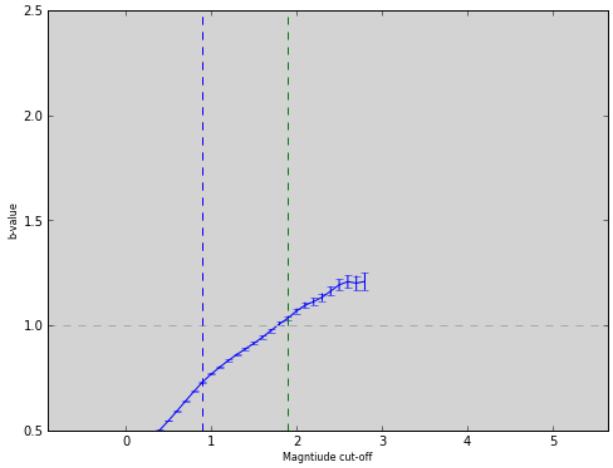

Fig. 12: B-value [9]stability as a function of cut-off magnitude: listing 5, line 6

```
start = '01-01-2012'
finish = '01-01-2014'
d4=window.datetime(d3, start, finish)
d4=window.single_attribute(d4, 'magnitude',
    1.0, 10.0, 'ecvd')
#Plot the time-series of magnitudes
mag_plots.mag_mc_plot(d4, colour='datetime')
#Plot the time-series of earthquake rates
rate_plots.ecd_rate_plot(d4)
```

Listing 6: Data Exploration.

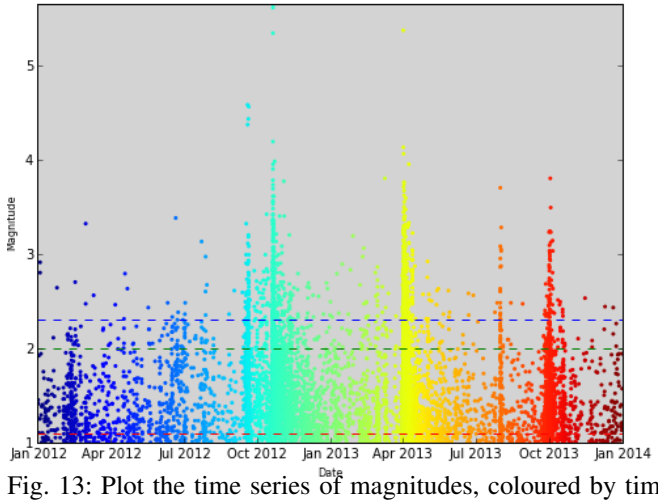

Fig. 13: Plot the time series of magnitudes, coloured by time: listing 6, line 6.

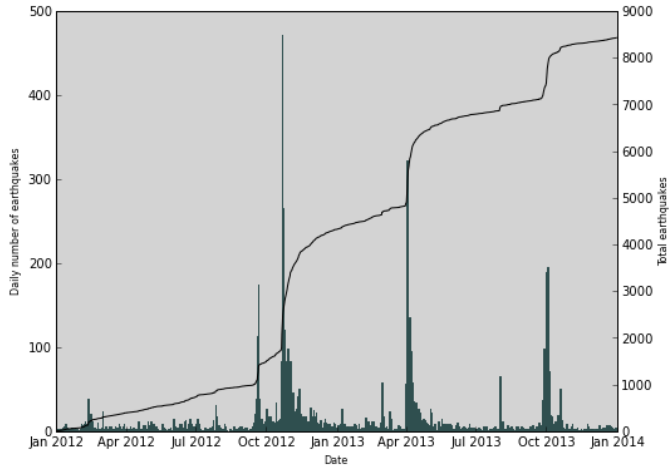

Fig. 14: Plot the time-series of earthquake rates (daily and total): listing 6, line 8.

VIII. Forecasting Methods

In this Section we illustrate with an example how to apply forecasting methods for analysing volcanic data from *Mt. Etna* (Sicily) with both types of `VarPy` experiments, single analysis and multiple analysis. In the example, we assume that an *d1*

[9] B-value is the slope of the best fitting line on Figure 12. Users can judge when the data is complete, i.e. when the line flattens at m 2.5

object is already created with *ecvd*, *scvd*, and *evd* attributes, and the necessaries libraries have been imported.

First, *d1* object is filtered by latitude and longitude to include the "greater Etna area", as line 6 of Listing 7 shows, creating *d2* object. Once filtered the data, a smaller period of time is selected by using line 10 of Listing 7, getting *d3* object.

Then, a single forecast analysis is applied to *ecvd* attribute of *d3* object, by using inverse omori's law (*iol_mle*) model (see line 1 in Listing 8). Once the model has been applied, the output of the analysis can be printed with the output parameters (see line 2 of Listing 8, and Figure 15). There are several model plots available in `VarPy`. Lines 4 and 6 of Listing 8 are two examples of them. The first plot, as Figure 16 shows, represents how well the model explains (red) the data (black). In this case a clear accelerating trend of cumulative event number with time can be seen in then data and the best fit. This could be an indication of an impending eruption. The second plot, Figure 17, represents the earthquake rates (blue points), best-fit model (red), and boot-strapped confidence limits, which in this example is 95%.

```
1  lat_min = 37.5
2  lat_max = 37.9
3  lon_min = 14.7
4  lon_max = 15.3
5  #Select data by latitude and longitude
6  d2=window.latlon(d1, lat_min, lat_max, lon_min
       , lon_max)
7  start = '01-06-2011'
8  finish = '01-01-2013'
9  #Select data by time period of interest
10 d3=window.datetime(d2, start, finish, 'ecvd',
       'evd', 'scvd')
```

Listing 7: Filtering and selecting data.

```
1  d4 = experiments.single_analysis(d3, 'ecvd', '
       iol_mle', t_min='01-08-2011', t_max=finish,
        mag_comp='GFT')
2  d4.ecvd.display_models()
3  #Show best-fit model and data
4  model_plots.model_plot(d4, 'ecvd', 'iol_mle',
       plot_type='cumulative')
5  #Show earthquake rates, best-fit model, and
       confidence limits
6  model_plots.model_error_plot(d4, 'ecvd', '
       iol_mle')
```

Listing 8: Single analysis.

```
model name: iol_mle
model type: single_forecast
model outputs:
-- Dataset: [ 1107.45637955   700.57012416     1.22779809]
-- rate_func ['iol_rate']
-- parameters ['k', 'tf', 'p']
-- total_func ['iol_total']
-- likelihood: -457.780086675
-- failure time: 01-07-2013
-- mc: None
-- t_forc: None
starting parameters
-- initial_parameters [50.0, 1038.0, 1.1]
-----
```

Fig. 15: Output of single forecast analysis: listing 8, line 2.

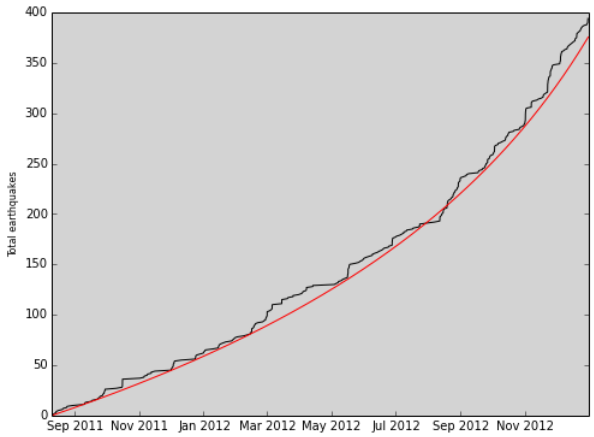

Fig. 16: Model plot for showing best-fit model and data analysis: listing 8, line 4.

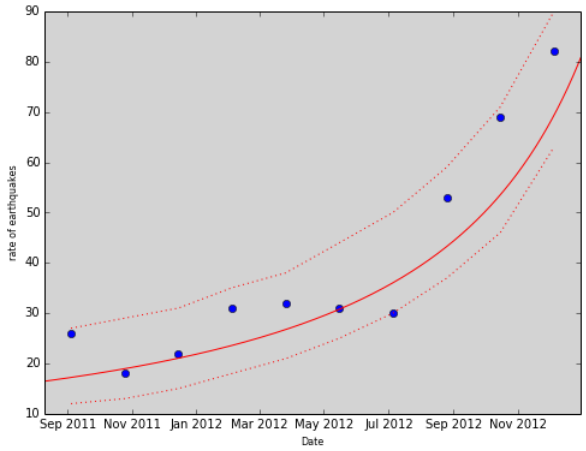

Fig. 17: Model plot for showing earthquake rates, best-fit model, and confidence limits: listing 8, line 6.

Finally, a multiple retrospective forecast analysis is applied to the d3 object. The *iol_mle* model is applied every two days (*t_step=2*) between the initial day (*t_min*) and the last day (*t_max*). The idea of this analysis is to simulate the evolving forecast that would result from data that was being producing in real-time. Every two days, the *iol_mle* model is applied to the collected data with no knowledge of the following data. Figure 18 displays the output of this analysis (see Listing 9).

```
1  d5 = experiments.multiple_analysis(d3, 'ecvd',
       'iol_mle', t_min=start, t_max=finish,
      mag_comp='GFT', t_step=2.0)
2  d5.ecvd.display_models()
3  #Show best-fit model and data
4  model_plots.model_plot(d5, 'ecvd', 'iol_mle')
```

Listing 9: Multiple analysis.

Figure 19 shows how well the model fits (blue) to the data (black). Note that in line 4 of Listing 9 the *cumulative* option has not been used. Therefore, in this case, this routine also determines the plot rates and reports the correct "daily rate" (green bars). This figure illustrates details of the temporal clustering and the intermittent nature of the data. These

```
model name: iol_mle
model type: retrosepective_forecast
model outputs:
-- Dataset: [ 687.93558343 662.00911965 1.15153509]
-- rate_func ['iol_rate']
-- parameters ['k', 'tf', 'p']
-- total_func ['iol_total']
-- likelihood: -436.358732726
-- failure time: 24-06-2013
-- mc: 1.2
-- t_forc: 734867.0
starting parameters
-- initial_parameters [ 50.0, 972.0, 1.1]
- - - - -
```

Fig. 18: Output of multiple retrospective forecast analysis: listing 9, line 2.

characteristics control the predictability much more than the information from the cumulative curve, so both need to be evaluated.

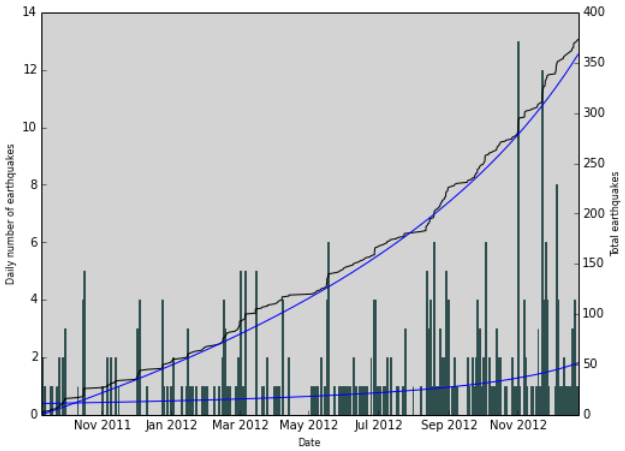

Fig. 19: Model plot for showing best-fit model, daily rates, and data analysis: listing 9, line 4.

IX. Conclusion

In this work we have presented a new Python open-source toolbox called `VarPy`, which is the first library designed specifically for volcanology and rock physics seismicity data. `VarPy` provides a full repertoire of commonly required actions, like analysis and modelling in real time and retrospective. `VarPy` offers capabilities for data exploration, data analysis and quality checking. With `VarPy` users can define their own workflows to develop models, analyses and visualizations. `VarPy` enables an easy method to analyze seismicity data, and standardizes different tasks like standards for data format, methodologies for processing data. When researchers use the same functions it is easier to compare the results and performance of models. Additionally, the cost of maintaining the library is shared among the longer community.

With `VarPy` we encourage the collaboration between rock physicists and volcanologists researchers. Researchers could also contribute to `VarPy` adding new models, simulators of seismic data, or filters.

`VarPy` is still under development, and we continuously aim to improve it. In the coming months we will add new models, and analysis facilities. It is an open source project and we would welcome assistance with its development.

Acknowledgment

The research described here has been supported by the NERC UK Grant (NE/H02297X/1) and the EC project VERCE (RI-283543).

References

[1] M. Beyreuther, R. Barsch, L. Krischer, T. Megies, Y. Behr, and J. Wassermann, "ObsPy: A Python Toolbox for Seismology," *SRL*, vol. 81, no. 3, pp. 530–533, may 2010.

[2] Rosa Filgueira, Malcolm Atkinson, Andrew Bell, Ian Main, Steve Boon, Christopher Kilburn and Philip Meredith, "escience gateway stimulating collaboration in rock physics and volcanology," 2014, iEEE escience 2014.

[3] Andrew F. Bell, John Greenhough, and Michael J. Heap, and Ian G. Main, "Challenges for forecasting based on accelerating rates of earthquakes at volcanoes and laboratory analogues," 2011, geophysical Journal International.

[4] Andrew F. Bell, Mark Naylor, and Ian G. Main, "The limits of predictability of volcanic eruptions from accelerating rates of earthquakes," 2013, geophysical Journal International.