pySurf - Python library for surface metrology and analysis

Soggetto Proponente: Vincenzo Cotroneo
Struttura INAF: OA-Brera (unica struttura partecipante)

List of Acronyms:

AFM	Atomic Force Microscope
API	Application Programming Interface
GUI	Graphical User Interface
IDL	Interactive Data Language
LTP	Long Trace Profilometer
OEM	Original Equipment Manufacturer

2. Sommario del programma (max. 1 pagina).

Surface and metrology data are of general interest in almost any field of science and technology, from optics, to microscopy, to material science and biology: data are collected by many different instruments, with a common feature of representing a set of points in the 3D space, and processed with tools, many of which are common in different disciplines.

Purpose of the proposed program is:

- extension and integration of code in an existing software project, devoted to manipulation, analysis and visualization of surface metrology data;
- its diffusion to the scientific and technical community as open source project, that can grow in time, as a framework to which functions specific to multiple disciplines can be added.

To the knowledge of the writer, while **surface data are widespread in many field of research** and a large number of operations is common to many disciplines, there is not an universally recognized framework in any language and especially not in Python, a language widely used (if not the most used one) in science.

Python is well equipped to manage and visualize 2D data, however it doesn't natively handle the matching of surface data with positional coordinates, making some operation non intuitive or quite laborious to perform. **The *pySurf* library consists in a set of core routines and classes, with a quite uniform and well defined interface, which can internally handle coordinate transformations, resulting in much more intuitive operations and enabling complex actions on data[1,2].**

The library was created for use by the author and by strict collaborators during research on X-ray mirrors[3,4,5]. The software was used to handle, integrate and compare data from a broad range of metrology instruments commonly employed in many fields (X-ray mirrors are characterized at all wavelength from AFM to LTP). The field offers a good number of use cases for manipulations and analysis operations, to a large number of which the library was successfully applied. Several examples from real use-cases are available, but need to be cleaned of development code. Documentation is available for many functions as docstrings (Python way of automatically generating documentation from comments in code), but is not uniform and not available as external documentation. Installation can be performed manually, but the user needs to make sure all dependencies (standard Python packages) are installed.

The aim of this project is to bring the existing Python code from alpha/pre-release to a documented shared codebase and library, that can perform the broad set of operations common to different fields of research interest to surface data, and can be extended to the operations specific to each field.

It is to be noticed that Python offers numerous facilities and tools available for software maintenance and documentation: tests, API documentation; the versatile code structure of the language makes the code usable by means of different interfaces (scripts, notebooks, graphical interface or integration with existing code), making it the ideal language for an easily-maintainable general-purpose project.

The goal of the project is to release an open-source beta-version, implemented through best-practices for maintenance of software and documentation **and present it to general public**. The project has the potential to start the development of a library that can serve as reference point for the community dealing with surface metrology data, or to integrate it with already existing and established software tools in use in different disciplines and connect to partners in research or industry.

Part of the project is to initiate collaborations and perform training aimed to the implementation of best practices for design of maintainable software and/or synergies with existing Python projects in advanced phase, with the added bonus of bringing the knowledge of these best practices inside INAF, transferrable to other Python projects.

3. Contesto generale e commerciale, esterno alla ricerca in astrofisica, nazionale e internazionale in cui la proposta si colloca (max. 4 pagine, incluse figure e referenze bibliografiche).

Metrology of surfaces has a crucial importance in all fields of optics manufacturing, and is fundamental in an enormous number of other fields in applied physics, science and technology (biology, material science, microelectronics, thin films etc.)[1]. In these applications as well as in X-ray optics metrology, measurements performed by means of different instruments (interferometers, microscopes, coordinate measuring machines, 3D profilometers and atomic-force microscopes) allow to cover different scales, from microscopic size (fraction of microns in side) to large areaa (tens of cm or more). Other instruments, also collect data representable as maps over two coordinate axis, that are treated with similar methods and can be subject of similar analysis.

In general, surface data describe a surface as a map of elevation, intensity, or other quantities as a function of 2D position on a surface, indicated by coordinate axis. A common (between different disciplines) set of operation (**leveling, cropping, subtraction, alignment, form removal, and many more**) can be performed for data analysis to extract relevant quantities and assess or predict performances. Often, **several related maps** (e.g. same sample measured with different instruments or setups) **need to be handled at same time, correlated, or processed on a common base or in relation one with the other.** There is sometimes a need for **batch processing of a large set of data, of interaction with other software, or conditional steps to take during analysis**[1,2].

Several commercial software solutions exist to handle surface data and are usually provided as **OEM software** with metrology instruments. These offer a large amount of functionalities, but most of the times, they are more oriented to engineering than to science: a sequence of menu operations must be performed manually keeping internal consistency of parameters, especially when data are processed in steps using different software tools.

To overcome these difficulties, many of these tools implement some form of scripting (often Python based) or macro procedures, but these still present several drawbacks, like limited flexibility or a steep learning curve. In particular the scripting capabilities are constrained by the initial design (e.g., graphical interface) or limited in functionality or in extensibility, because embedded in the original software. Furthermore, the skills learned are not easily reusable, as the software is likely to be different on different instruments. The few software solutions that are **free or open-source** and can handle general data, present similar problem (e.g. Gwyddion, developed for nanometrology, and probably the most successful software of this kind, is based on a proprietary graphical interface), or don't use a modern and widespread language (e.g. TOPO library in IDL).

The need for an uniform and versatile processing on surface data coming from different instruments (and on different scales) during the metrology of X-ray mirrors, the analysis of coatings, and the study of active optics, led us to the development of the *pySurf* library as a flexible framework for the realization of complex surface analysis.

***pySurf* approach**

In order to illustrate ***pySurf* approach** and its relevance to different contexts in the field of surface data analysis, it can be useful to provide some minimal examples of its usage and syntax. The library is mostly made of classes, of which the most important is `Data2D`, representing a measured surface.

A Data2D object `p1` can be created by passing data, e.g.: `p1 = Data2D(data,x,y)`, but can also be read directly from file:

```
p1 = Data2D(data,x,y,units=['mm','mm','$\mu$m'], name='SAMPLE 01',
reader='4D_reader') # Several readers for the most common formats are
implemented. Scales, units and other modifiers can also be controlled as
options. A name is used for output and visualization.
```

Standard operations (e.g. rotations, crop, leveling) are implemented as methods, that return a new object:

```
p1_crop = p1.crop([10,20],[-5,5])      # data are cropped on a 2D range
p1_crop = p1.crop(interactive = True) # user interactively select area
p1_lev  = p1.level([2,5])             # data are leveled removing terms up
                                       # to given orders in x and y
```

Objects and methods can be inspected through standard Python question-mark syntax: `e.g., data2?` returns a description of object, properties and methods as extracted from documentation embedded in code as docstring format. Any object or function is inspectable in the same way.

Results can be saved and output generated, including visualization. Functions inherit all powerful Python options. For example, the returned plot object p can be modified and manipulated for publication quality figures:

```
p = p1.plot() #default options (e.g. units) are derived from Data2D object
p1.save('saved_data.txt')
```

Non-standard operations can be implemented as methods or external functions. Users can write their own and easily extend the library, or keep them in a separate package.

```
p1_psd = p1.psd2d('resultsname')  #perform a complex analysis, extracting
power spectral density (PSD) and generating outputs and visualizations.
Results are saved on files named after 'resultsname' and returned as
object(s).
```

Algebric operators are overridden, here data `p1` and `p0` are automatically resampled before subtraction, so it works even with different zoom or data size:

```
difference = p1-p0 # difference of two surfaces, fine tuning is also
available by directly accessing functions, e.g.: p1.subtract(p0, options=..)
```

Because of object model, commands can be easily chained:

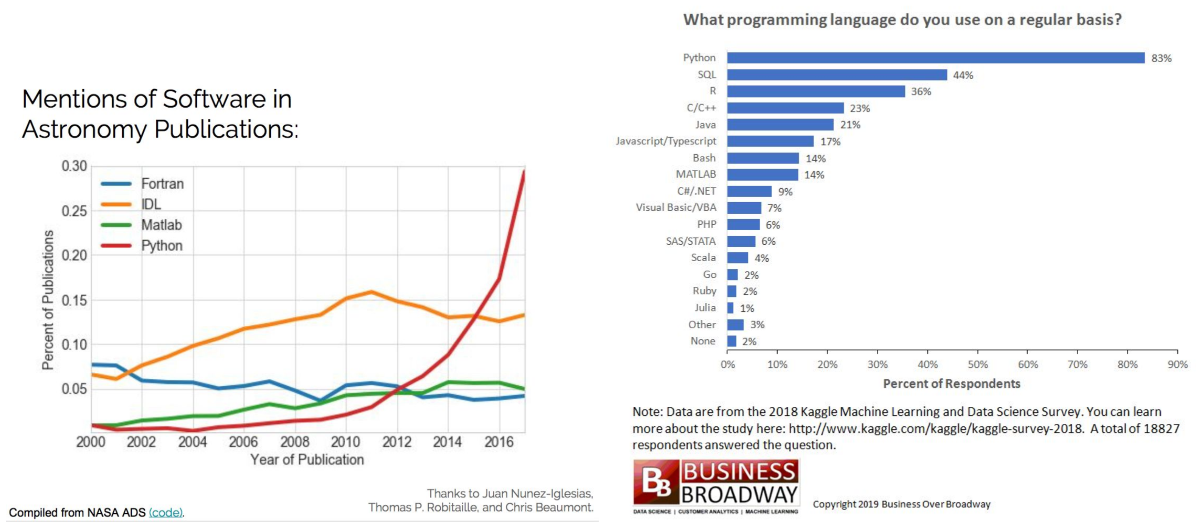
```
(p1-p0).level().crop([10,20],[-5,5]).plot()
```

The library includes additional classes and tools to handle scattered points, profiles, and functions specific to X-ray optics or metrology of optics.

<u>The Python programming language impact on science</u>

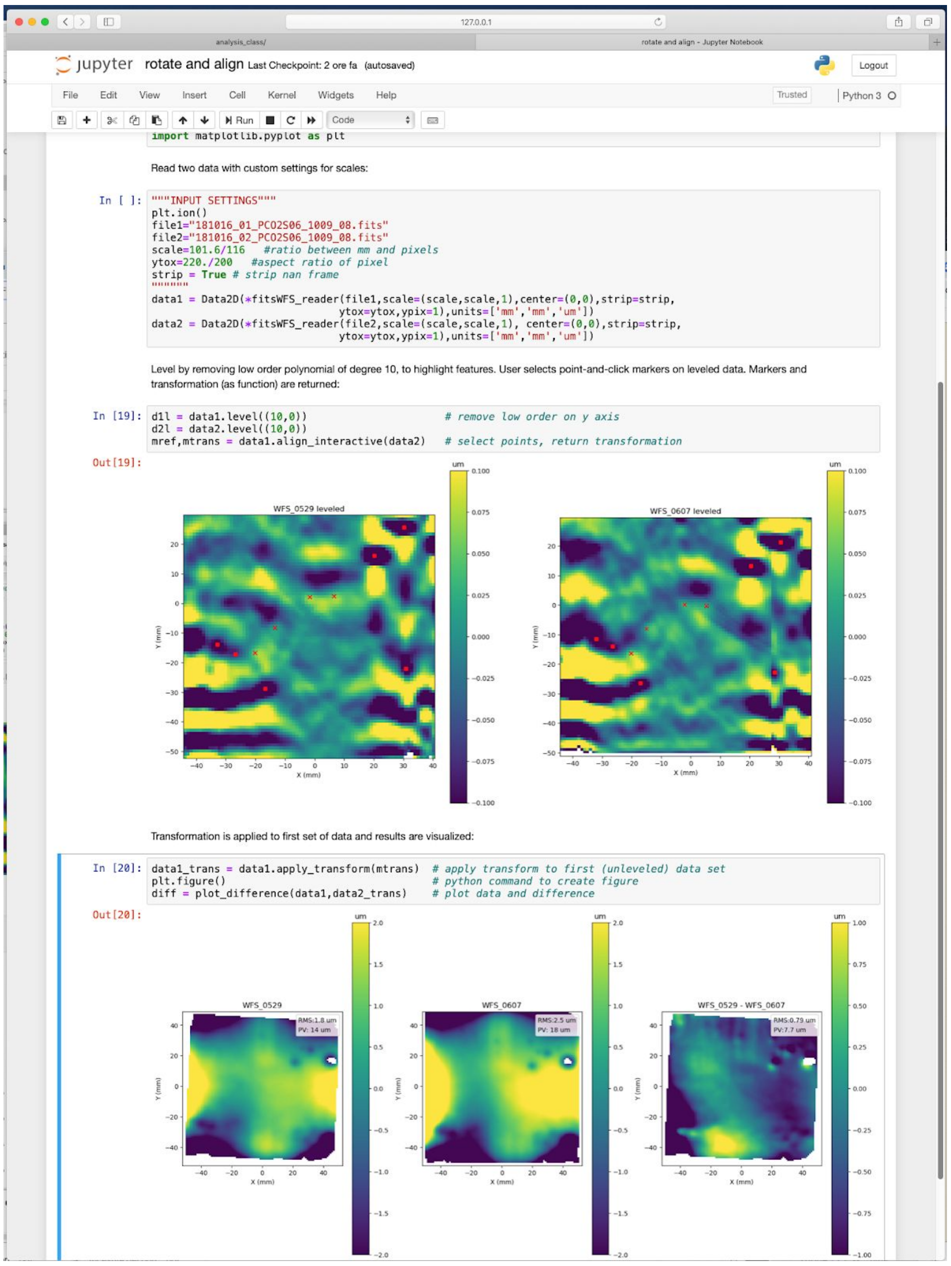
It is quite common for scientists and researchers, to write code for their own use. This code, as in the case of this proposal, can be useful to others, even in a broader context than the original, still there is little reuse (with most of the code being in early alpha-stage: not installable, not documented, and usable only by the author). A large part of effort goes in building again things that were already developed by others, rather than building on top of them.

In many fields of science, the Python programming language, with his approach aimed at reusability, inverted this tendency: in the last years, several Python packages, tools and libraries emerged and established themselves as standard tools, bringing the language to be one of the most popular in science, if not the most popular. The result is an enormous base of users, and many of the projects bringing considerable return in terms of image and citations to the host Institutions. Python is likely also the most commonly used language for scripting of surface-metrology instruments proprietary software.



Note: Data are from the 2018 Kaggle Machine Learning and Data Science Survey. You can learn more about the study here: http://www.kaggle.com/kaggle/kaggle-survey-2018. A total of 18827 respondents answered the question.

Fig. 1: plots about use of Python in astronomy (left panel) and data science (right panel).



```
import matplotlib.pyplot as plt
```

Read two data with custom settings for scales:

```
In [ ]:  """INPUT SETTINGS"""
         plt.ion()
         file1="181016_01_PCO2S06_1009_08.fits"
         file2="181016_02_PCO2S06_1009_08.fits"
         scale=101.6/116   #ratio between mm and pixels
         ytox=220./200    #aspect ratio of pixel
         strip = True # strip nan frame
         """"""
         data1 = Data2D(*fitsWFS_reader(file1,scale=(scale,scale,1),center=(0,0),strip=strip,
                               ytox=ytox,ypix=1),units=['mm','mm','um'])
         data2 = Data2D(*fitsWFS_reader(file2,scale=(scale,scale,1), center=(0,0),strip=strip,
                               ytox=ytox,ypix=1),units=['mm','mm','um'])
```

Level by removing low order polynomial of degree 10, to highlight features. User selects point-and-click markers on leveled data. Markers and transformation (as function) are returned:

```
In [19]: d1l = data1.level((10,0))                   # remove low order on y axis
         d2l = data2.level((10,0))
         mref,mtrans = data1.align_interactive(data2)   # select points, return transformation
Out[19]:
```

Transformation is applied to first set of data and results are visualized:

```
In [20]: data1_trans = data1.apply_transform(mtrans)  # apply transform to first (unleveled) data set
         plt.figure()                                 # python command to create figure
         diff = plot_difference(data1,data2_trans)    # plot data and difference
Out[20]:
```

Fig.2: Example of usage from notebook interface: commands are run in a browser. Images are retained and text can be added to code for report generation (code can be hidden). In the example, two sets of data with different alignment are compared. A removal of low order legendre polynomials is performed to highlight features. Markers are interactively (by zoom, point and click) placed on features. Markers are then used to determine alignment transformation, that is applied to original data The second set is then subtracted. Note that the two operation of alignment and subtraction are performed on two different sets of data (leveled and unleveled. This operation, not trivial, if feasible at all, with many of the commercial software, can be performed with a few lines of code.

The implementation of this project in python, gives to the library the following advantages:

- open source: Python programming language enable and enforces free access to source code, allowing to obtain improvement, extension and review from the community of users.
- extensible: the software consists in a set of class, functions and procedures, as characteristic of Python libraries:
  - calls to functions and class methods can be called interactively from command line, making the library usable as a “programming language” for surface analysis.
  - the “language” of the previous point can be scripted or used in batch processing, allowing the realization of complex analysis scripts and data processing pipelines, with or without user interaction.

- ○ can be used in notebook interface (shown in example in fig.2), for reproducible analysis keeping track of all intermediate steps and final output that can be converted to report and/or saved as individual data and figures.
- ○ can be easily wrapped in more complex graphical interfaces or easily integrated with existing tools and code. Notably most of the OEM software for surface instruments has some extent of Python capabilities, and there are very well developed Python packages for data analysis, databases, image processing and feature recognition, that can easily be integrated with the project.

- versatile and user-friendly:
    - ○ analysis of metrology data is relevant in many fields, essentially any data representable as 2D array with (or without) associated axis can find an application for some of the functions and methods in the library.
    - ○ learning curve is as small as possible for who already knows Python (i.e. a constantly growing number of people in all field of science and technology), as the language offers and uniform interface and very convenient facilities for inspection of function, classes and objects API's.
- maintainable:
    - ○ Python has several tools for extraction of documentation from code, tests, debug and developing environment each user can choose from (ranging from text editor and command line to complex IDE with integrated debugger and version control systems).

Use cases and applicability of the program

A generic example of use, applicable to several fields of application, can be the alignment and subtraction of two images, as shown in fig. 2 as handled by *pySurf*, for a case in which the alignment is interactively determined on the base of a separate set of data. The case is not trivial to handle with common software: the user needs to enter and exit several menus in the software GUI, save fiducials, retrieve them, and open and close several files. This is a set of actions that, when possible, is performed differently and with different limitations, in each software. The use of different softwares, also results in a non uniform output.

A few more examples of cases that are not easy to handle with traditional software, but can be conveniently managed by *pySurf* can be[1-8]:

- Batch extraction of regions, points or profiles from data, calculation of derived quantities and analysis/visualization as series (e.g. to study the evolution of a feature in time, or to study thermal and environmental effects).
- Best fit of a surface by scaling of another surface (for example in thin films and material research, the fit of the change of shape of a silicon wafer due to film stress), or combination of multiple ones (e.g.: in optical manufacturing or in active optics, the effect of removal functions from mechanical machining or by effect of actuation).
- Comparison of two different surface maps with different sampling (e.g.: comparison of details with different zoom, cross calibration of instruments, comparison of data and simulations) or comparison of data taken at different times (i.e., when the position or registration of the sample might be non constant)
- Combination and stitching of surfaces that lies on a larger area than a single field. For example when local maps (e.g., maps of a deposited film are measured with a high magnification) are embedded in a larger map (for example recognize and count features in high magnification microscopy image distribute over the area of a larger sample).
- Building data pipelines, or in general all cases where a complex analysis on data needs to be repeated multiple times.
- Integration with existing specific code or analysis tools.

Bibliography

[2] J. Wang et al 2015 Surf. Topogr.: Metrol. Prop. 3 023001; https://doi.org/10.1088/2051-672X/3/2/023001
[1] A. Townsend et al 2016 Volume 46, October 2016, Pages 34-47; 10.1016/j.precisioneng.2016.06.001
[3] Beiwen Li, Proc. SPIE 10749 (2018); https://doi.org/10.1117/12.2321656
[4]L. R. Graves et al., Proc. SPIE 10742 (2018); https://doi.org/10.1117/12.2320745
[5] C. T. DeRoo et al., Proc. SPIE 10399 (2017); https://doi.org/10.1117/12.2275210
[6] V. Cotroneo et al., Proc. SPIE 10761, (2018); https://doi.org/10.1117/12.2323283
[7] N. Bishop et al., JATIS 5(2) 021005 (2019) https://doi.org/10.1117/1.JATIS.5.2.021005
[8] Y. Yao et al., JATelesc. Instrum. Syst. 5(2) 021011 (2019); https://doi.org/10.1117/1.JATIS.5.2.021011

4. Obiettivi che il programma si prefigge di raggiungere e quelli verificabili con specificato il ruolo dei partecipanti (max. 2 pagine).

The aim of this project is to bring the existing Python code from alpha/pre-release to a documented shared codebase and to release it on as open-source library.

The project has the potential to serve as reference point for the community dealing with surface metrology data; be integrated with already existing and established software tools in use in different disciplines; and offer connections to partners in research or industry. As such it can be an useful tool for many research projects, with a corresponding return in terms of citations and visibility for the host Institution.

To the purpose, it is particularly important to follow best-practices that make optimal use of python facilities for maintainable and documented python code, together with a set of facilities to make code and documentation accessible to users. Standards in this sense have been defined by the experience of similar projects that successfully grew to larger size: one of these, the Astropy project [1], has put a considerable effort to define guidelines and templates for sustainable and professional package that meets high standards[2], which can serve as guidelines to define the plan of this project.

Objectives

- Release an open-source beta-version.
- Implement best-practices for maintenance of software and documentation, that can be transferred to other projects.
- Present the library to general public.
- Identify potential partners and initiate collaborations or integration with existing software.

Potential connections and links

To investigate in first 3-6 months:

- *OrAnge SYnchrotron Suite[3]:* a visual tool for ray-tracing of beamlines and X-ray experiments. Surface data, that *pySurf* can manipulate, are used in *OASYS* for the simulation of optical elements. Also beam images or photons distributions, can be expressed as surface maps. Same group released *DABAM[4]*, an open-source database containing metrology data for X-ray mirrors that can offer test cases and interactions.
- *Glue*:[5] *Glue* is a powerful visual tool in Python for the analysis of relationships within and among datasets, born in astronomy and successfully applied to other disciplines. Its approach is oriented to integration with user written code for data input, cleaning, and analysis. A possible interaction is the use of *Glue* to correlate different features in different surface maps undergoing a common processing. *PySurf* could also be used to process different sets of data and relate the results to the processing parameters of each set, for example: the number of given features counted over each of a set of data maps could be related to temperature, time and position of the map.
- *Gwyddion:[6]* is an open source surface processing data, developed at Department of nanometrology, Czech Metrology Institute for scanning probe microscopy data visualization and analysis. The software is based on a GUI and has a large set of features for data processing, many of which are common to *pySurf* or are of interest for an implementation. *Gwyddion* is written in C++, and has experimental python scripting tools, for which interaction need to be investigated. Czech Metrology Institute organizes every two years a Nanometrology workshop.
- Established 'core' libraries for numerical processing (numpy), scientific plotting (matplotlib), tables and timeseries (pandas), image manipulation (PILLOW), feature recognition and computer vision (openCV).
- At least two of the biggest companies producers of interferometers and profilers: Zygo and 4D OEM analysis softwares (respectively Mx™ and 4Sight™) implement some scripting capabilities in python.

Roadmap and milestones

3 months: Release on github of a set of core functions and classes and of a set of working tests, from reduction and sorting of the existing codebase.

Release as open source shared codebase on github with minimal functionalities. The purpose is to establish a template for best practices obtained from a stripped-down version of the current code, to which all the remaining functions and code will be added after quality control. This development will be made on a separate branch on GitHub, that will eventually replace the current code.

The initial codebase will include:

- the main classes and functions `Data2D` and `Points`, with only core methods implemented and their dependencies
- code compliant to formatting standards PEP8
- the code will contain docstrings according to pydoc standards
- tests will be included according to pytest
- this step will not include documentation, apart from three notebooks demonstrating the functionality of the full package
- representative working classes and functions with an extended (besides to core) methods (Beta level functions and code can be optionally included)
- conduct investigation of potential collaborations

6 months: opening to developers
Starting from this step the project will be in beta-version open to public development:

- Use Sphinx to extract API documentation. Extension with narrative part. Release on *readthedocs.org*
- Initial setup of Continuous Integration of Github (test on code and documentation).
- Release of three tutorials: overview, advanced functions and complex use case
- create redistributable and installable packages for distribution with *pypi* and *conda-forge*. At least one among Windows, Linux, Mac OS X as installable (with setup) package, others installable from source.

10 months: opening to general user
Completion of codebase, after this step the library will be available for use and promotion.

- inclusion of all remaining code, including `pyProfile` and `utilities` modules
- inclusion of tutorials: overview of modules `points` and `pyProfile`, add another two more use case to pySurf
- definition of tools for collaborative effort (mailing list, updates on github, etc.)
- binary installation packages working on all operating systems
- release a basic developer guide

12 months: distribution to general audience:

- Participation to one python conference/workshop Scipy/Scipy-Europe.
- One conferences related to metrology or to a potential field of application (optics, nanotechnology, biology).
- One collaboration with a related project, for integration of the two. Potential projects: Gwyddion, OASYS, or application on a different field.
- Promotion in external institution and/or initiate collaboration with related project.
- Integration of more examples and tutorials from specific use cases.
- Refine developer guide
- Write one article on technical electronic platform (e.g., Medium) for exposure
- Publish a paper about the library on a refereed journal

References

[1] A. M. Price-Whelan et al., The Astronomical Journal, Volume 156, Issue 3, article id. 123, 19 pp. (2018) 10.3847/1538-3881/aabc4f
[2] Astropy guidelines: https://www.astropy.org/affiliated/#affiliated-instructions
[3] L. Rebuffi et al., Proc. SPIE 10388, (2017) DOI: 10.1117/12.2274263
[4] M. Sánchez del Río et a., Journal of Synchrotron Radiation (2016). DOI: 10.1107/S1600577516005014
[5] Robitaille et al (2017), Glueviz v0.15.2: multidimensional data exploration, 10.5281/zenodo.774845
[6] D. Nečas et al, *Cent. Eur. J. Phys.* **10**(1) (2012) 181-188 http://gwyddion.net/

5. Impegno di personale di ruolo dedicato al programma (in FTE), distinto per qualifica e ruolo all'interno del programma e suddiviso per strutture di ricerca (per il personale Associato vale la Struttura presso la quale si è associati).

Vincenzo Cotroneo 0.3 FTE

6. Costi del programma (limitatamente ai fondi richiesti ad INAF) suddivisi per macro-voci (investimento, consumo, calcolo, missioni, promozione e divulgazione).

3000 euro: computer and hardware/software dedicated to the project. Including hardware for tests on different systems: mac, linux, windows; software or hosting services
2000 euro: workshops, books, and training on best practices for management of open source project
9000 euro: for collaborations and travels to conferences. An example of use could be:

- one month visiting scientist to work on software integration or application to specific field out of astronomy
- 15 days collaboration with industry or software collaboration
- one presentation to industry
- participation to two conferences about python/metrology/other field of application

2000 euro: consulting or services for:

- promotion, marketing, branding;
- software design, implementation, maintenance, sharing

The above cost estimate, for a total of 16000 euro, can be divided in macro-voices as (see example of budget in table 1):

Investimento: 2800 euro (hardware/software/services/books and tutorial)
Consumo: 3200 euro (services, training, publication and event participation costs)
Calcolo: 300 euro (sharing, hosting, computational services)
Missioni: 8600 euro (training, collaborations)
Promozione e Divulgazione: 1100 euro (services and consulting)

Table 1: example of budget for the project divided by destination and by macro-voices:

macro-voce	**Total macro-voce**	BY USE DESTINATION			
		hardware/software /services	training	collaborations and conferences	consulting
investimento	**2800**	2500	300	/	/
consumo	**3200**	200	500	1500	1000
calcolo	**300**	200	100	/	/
missioni	**8600**	/	1100	7500	/
promozioni e divulgazione	**1100**	100	/	/	1000
TOTAL	**16000**	2500	2000	9000	2500

7. Risorse strumentali ed eventualmente finanziarie messe a disposizione dalle Strutture di ricerca INAF ed eventuali altre fonti di finanziamento esterne.

Several surface metrology tools are installed at OAB and regularly used by the proponent as part of his activity, to which the software in this project is functional. The instruments will provide test data and use cases for the software.
It will be investigated the possibility of using INAF structures to provide support for meetings/small workshops.
pySurf code is registered at INAF.

8. Dichiarazione, datata e firmata, di accettazione da parte del Direttore della Struttura INAF di afferenza del Soggetto Promotore o Coordinatore Nazionale del Programma e nulla osta da parte dei Direttori di Struttura dei partecipanti al programma.

ALLEGATO 1

10. Assenso del Soggetto Promotore o Coordinatore Nazionale del Programma alla diffusione presso gli eventuali valutatori esterni, all'esclusivo scopo della valutazione stessa, delle informazioni riguardanti i progetti presentati a seguito di firma di opportuno accordo di riservatezza; dichiarazione ai sensi del D. Lgs. n. 196/2003 di consenso al trattamento dei dati sensibili e non.
N.B. La modulistica dovrà essere compilata con i caratteri "Times New Roman-10" e con spaziatura singola.

ALLEGATO 2