GrainSizeTools

A Python script for estimating the grain size from thin sections

This pdf manual is based on the online documentation at https://marcoalopez.github.io/GrainSizeTools/

Manual version:
v31

Release date:
2018/04/05

Author:
Marco A. Lopez-Sanchez

Table of contents

Requirements

GrainSizeTools script requires Python 2.7.x (legacy) or 3.5+ versions and the scientific libraries *Numpy*, *Scipy*, *Pandas* and *Matplotlib*. We recommend installing the Anaconda or the Enthought Canopy distributions. Both distributions have free basic versions that include all the required the scientific packages. In case you have space problems in your hard disk, there is a distribution named miniconda that only installs the packages you actually need.

The approach of the script is based on the estimation of the areas of the grain profiles obtained from thin sections. It is therefore necessary to measure them in advance and save the results in a txt/csv file. For this task, we highly encourage you to use the *ImageJ* application or one of their different flavours (see here). These are public-domain image processing programs widely used for scientific research that runs on Windows, macOS, and Linux platforms. This documentation contains a quick tutorial on how to measure the areas of the grain profiles with ImageJ, see the *Table of Contents*. The combined use of **ImageJ** and **GrainSizeTools script** is intended to ensure that all data processing steps are done through free and open-source programs/scripts that run under any operating system.

Scope

GrainSizeTools (GST) script is primarily targeted at anyone who wants to:

1. Visualize the distribution of apparent grain sizes and extract different statistical parameters to describe the features of the distribution
2. Estimate differential stress via paleopizometers (**New in version 1.4+!**)
3. Approximate the actual 3D distribution of grain sizes from thin sections. This includes an estimate of the volume occupied by a particular grain size fraction and the shape of the population of grain sizes (assuming that the distribution of grain sizes is lognormal-like)

GST script only requires as input the areas of the grain profiles measured grain-by-grain in a thin section. The script is not intended to determine the mean grain size via the planimetric (Jeffries) (i.e. the number of grains per unit area) or intercept (the number of grains intercepted by a test line per unit length of test line) methods. The reasons for using grain-by-grain methods over the planimetric or intercept ones in natural rocks are detailed in Lopez-Sanchez and Llana-Fúnez (2015). Below is a brief outline of the key assumptions to consider so that the results obtained by the script are meaningful and reliable.

Safety concerns

> All this safety concerns assume that the calibration of the microscope and the scale of the micrographs were set correctly.

Determination of "average" (mean, median, peak) apparent grain sizes

Unidimensional apparent grain size measures such as the **mean** or the **median** are only meaningful in specimens that show a **unimodal distribution of diameters** or areas. It is therefore key to **always visualize if the distribution show a single peak**. In the case multimodal distributions (two or more frequency peaks), you can use for comparative purposes the **location of the frequency peaks** based on the Kernel density estimate as proposed in Lopez-Sanchez and Llana-Fúnez (2015). Despite this, the best option when a multimodal distribution appears is to separate the different populations of grain size using image analysis methods.

Unfortunately, no general protocol exists in the earth science community on what "average" grain size measure to use when using grain-by-grain methods. For example, in the case of paleopiezometry studies, some authors have been using the mean, others the median, and some the mode. In addition, some authors applied stereological corrections and others do not, and some used logarithmic or the square root grain size instead of the linear grain size. Since it seems that this is not going to change in the short term, we propose to always report all "average" grain size measures (mean, median, freq. peak). This will allow other scientists to directly compare their data with yours without using correction factors which always involve some assumptions. In any event, since you will have to choose one of them in your study so we advise from here to follow this rule of thumb:

- use **mean and standard deviation (SD)** when your **distribution is normal-like** (e.g. when using logarithmic or square-root scales that usually produce Gaussian-like distributions)
- use **median and interquartile (or interprecentil) range** when your **distribution is skewed**

The rationale behind this rule is that the sample mean is generally more efficient than the median, but the sample median is always more robust. The latter feature makes the median more efficient when

distributions have "thick" tails as it happens in most skewed distributions. Lastly, if you want to compare grain sizes in specimens that were measured at very different conditions (e.g. different resolutions) consider using the **location of the frequency peak (peak grain size)** since this measure is not affected by issues such as the resolution limitation among others (Lopez-Sanchez and Llana-Fúnez, 2015) and hence is more robust in such situations.

When grains are equant (equiaxed) or near-equant (i.e. aspect ratios mostly < 2.0) any specimen orientation is acceptable for estimating unimodal grain size measures and a **single section** is enough to obtain a reliable estimate as long as you measure a required minimum of grain sections (see below for details). When grains show generally aspect ratios above 2.0 and preferred orientation throughout the rock volume, you will need to **estimate the grain size over three orthogonal sections and then average the results** to obtain meaningful results for every independent grain size measure. Although specimens with equant grains accept any orientation to estimate unidimensional grain size measures, it is advisable to use a principal section. Specifically, we promote the use of the XZ section, i.e. parallel to the lineation and perpendicular to the foliation, since this will allow us: (i) to estimate whether the grains are far from equant using the aspect ratio; and (ii) to provide a fairer comparison between different specimens when near-equant grains and preferred orientation of the large axes exist.

To obtain reliable grain size estimates you should measure or your grain boundary map should contain at least 433 grain sections. This sample size will ensure that 95 % of the time the mean grain size estimated will have an error equal or less than ± 4 %; if you use 965 sections, a practice that we recommend, this will ensure the 99% of the time instead (Lopez-Sanchez and Llana-Fúnez, 2015). If you want to obtain an accurate **confidence interval** for your estimates, then you have to take several representative micrographs from the same specimen (three or more) and estimate the "average" (mean, median or peak) grain size in each of them. Then use the `confidence_interval` function implemented in the script to get a robust confidence interval. For details on how this determination works see the next section or the documentation of the function using the command `help(confidence_interval)` in the console.

Regarding paleopizometry estimates, **do not use "average" values using stereological methods but directly apparent grain size measures**. The rationale behind this is that stereological methods are always built on several and ill-conditioned geometric assumptions and the results will always be, at best, only approximate. This means that the precision of the estimated 3D size distribution is **much poorer** than the precision of the original distribution of grain profiles since the latter is based on real data.

Determination of stress via paleopiezometers

When using a piezometer relation is of paramount importance to ensure what type of grain size measure should be used. For example, if you want to use the piezometric relation for quartz established in Stipp and Tullis (2003), note that they used the **root mean square apparent diameter** not the *linear nor the logarithmic mean diameter*. Also, it is important to note that the mean of the logarithmic or square-root apparent grain size is not the same as calculating the logarithm or the square root of the mean apparent grain size using a linear scale. For more details see the step-by-step tutorial.

It is always advisable to calculate a **confidence interval** for your paleopiezometry estimates, which means that you should do at least three independent measures (i.e. from different grain size maps). Since version 1.4.4+, the GST script implements a function to **estimate robust confidence intervals using the student's t-distribution** (see the step-by-step tutorial for details). This is a much robust approach than calculating the mean plus two times the standard deviation when the sample size is small (< 10) and both, the mean and the SD, cannot be estimated accurately.

Getting the shape of actual grain size distribution or the volume occupied by a particular grain size fraction

Estimating the actual grain size distribution from thin sections using stereological methods requires assuming spatial homogeneity and that **grains under study are equant or near-equant**. The Saltykov and two-step methods will not provide reliable results if most of the grains show aspect ratios above 2.0, regardless of whether a shape preference orientation exists or not. In any event, this assumption is acceptable most of the time for the most common *recrystallized (dynamically or statically)* mineral phases in crustal and mantle shear zones, including quartz, feldspar, olivine, and calcite, as well as ice. However, be careful when recrystallized grains show very irregular/lobate grain boundaries (i.e. low "solidity" values).

The Saltykov method is suitable to estimate approximately the volume of a particular grain fraction of interest (in percentage) and to visualize the aspect of the derived 3D grain size distribution using the histogram and a volume-weighted cumulative frequency curve. To provide reliable results, the method requires using a few numbers of classes and a large number of individual grain measurements. Practical experience indicates using more than 1000 grains and less than 20 classes. Unfortunately, the optimal number of classes depends on the type of distribution and the range of values; and the latter can vary largely due to presence or absence of grains with very large sizes. In conclusion, the number of classes has to be set by a trial and error approach. This will inevitably lead to different authors using a different number of classes across studies. Due to this, when estimating the volume of a grain size fraction based on a single grain boundary map it is necessary to take an absolute error of ± 5 to stay safe (see details in Lopez-Sanchez and Llana-Fúnez, 2016). If possible, take more than one representative grain boundary map and then estimate a confidence interval as explained above in this section.

The two-step method (Lopez-Sanchez and Llana-Fúnez, 2016) is suitable for describing quantitatively the shape of the actual 3D grain size distribution using a single parameter called the multiplicative standard deviation (MSD) value. The method assumes that the actual grain size distribution follows a lognormal distribution, **there is therefore critical to visualize the distribution using the Saltykov method first and ensure that the distribution is unimodal and lognormal-like**. The MSD estimate is independent of the chosen number of classes as long as the Saltykov method produces stable results (i.e. you do not lose the lognormal appearance of the distribution due to the use of an excessive number of classes) and provides a reliable uncertainty value.

Getting Started: A step-by-step tutorial

> **Important note:** Please, **update to version 1.4.5.** It is also advisable to **update the plotting library matplotlib to version 2.x** since all the plots are optimized for such version.

Open and running the script

First of all, make sure you have the required software and necessary Python libraries installed (see requirements for details), and that you downloaded the latest version of the GrainSizeTools script (currently the v1.4). If this is the case, then you need to open the script in a integrated development environment (IDE) to interact with it (Fig. 1). For this, open the Canopy editor -if you installed the Enthought package- or the Spyder IDE -if you installed the Anaconda package-, and open the GrainSizeTools script using `File>Open`. The script will appear in the editor as shown in figure 1.

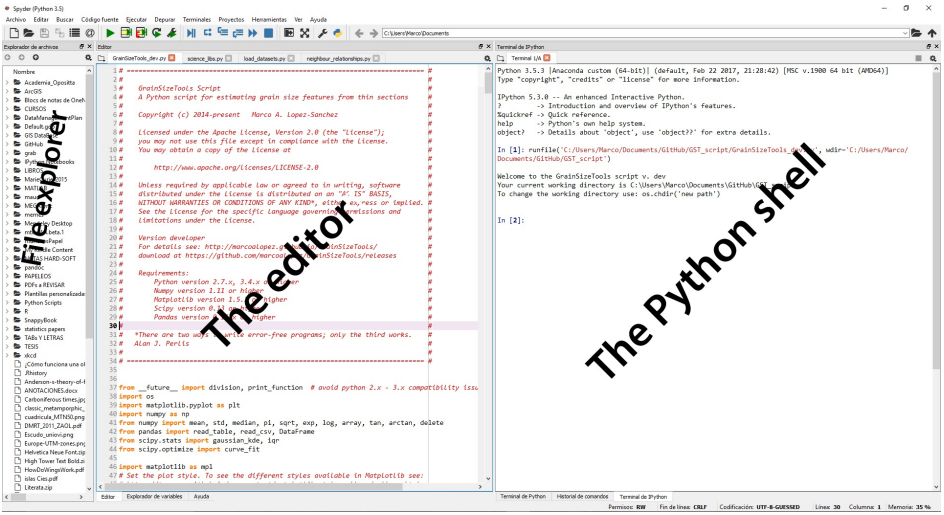


Figure 1. The editor and the Python shell (a.k.a. the console) in the the Spyder integrated development environment (IDE). The Enthough Canopy and the Spyder IDEs are both MATLAB-like IDEs optimized for numerical computing and data analysis using Python. They also provide a file explorer, a variable explorer, or a history log among other interesting features.

To use the script it is necessary to run it. To do this, just click on the green "play" icon in the tool bar or go to `Run>Run file` in the menu bar (Fig. 2).

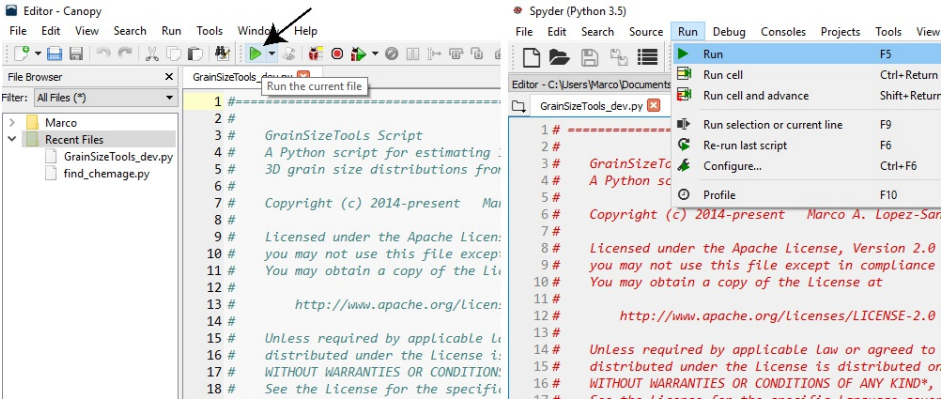


Figure 2. Running a script in the Enthought's Canopy (left) and Spyder (right) IDEs.

The following text will appear in the shell/console (Fig. 1):

```
================================================================================
Welcome to GrainSizeTools script v1.4.5
================================================================================

GrainSizeTools is a free open-source cross-platform script to visualize and
characterize
the grain size in polycrystalline materials from thin sections and estimate
differential
stresses via paleopizometers.

METHODS AVAILABLE
=================  ============================================================
Functions          Description
=================  ============================================================
extract_areas      Extract the areas of the grains from a text file (txt, csv or xlsx)
calc_diameters     Calculate the diameter via the equivalent circular diameter
find_grain_size    Estimate the apparent grain size and visualize their distribution
derive3D           Estimate the actual grain size distribution via steorology methods
quartz_piezometer  Estimate diff. stress from grain size in quartz using piezometers
olivine_piezometer Estimate diff. stress from grain size in olivine using piezometers
other_pizometers   Estimate diff. stress from grain size in other phases
confidence_interval Estimate the confidence interval using the t distribution
=================  ============================================================

You can get information on the different methods by:
    (1) Typing help(function name) in the console. e.g. help(conf_interval)
    (2) In the Spyder IDE by writing the name of the function and clicking Ctrl + I
    (3) Visit script documentation at https://marcoalopez.github.io/GrainSizeTools/


EXAMPLES
--------
Extracting data using the automatic mode:
>>> areas = extract_areas()

Estimate the equivalent circular diameters:
>>> diameters = calc_diameters(areas)
```

```
Estimate and visualize different apparent grain size measures
>>> find_grain_size(areas, diameters, plot='sqrt')

Estimate differential stress using piezometric relations
>>> quartz_piezometer(grain_size=5.7, piezometer='Stipp_Tullis')

Estimate the actual 3D grain size distribution from thin sections
>>> derive3D(diameters, numbins=15, set_limit=40)
>>> derive3D(diameters, numbins=15, set_limit=None, fit=True)
```

Once you see this text, all the tools implemented in the GrainSizeTools script will be available by typing some commands in the shell as will be explained below.

A brief note on the organization of the script

The script is organized in a modular way using Python functions helping to modify, reuse or extend the code. A Python function looks like this in the editor:

```
def calc_diameters(areas, correct_diameter=0):
    """ Calculate the diameters from the sectional areas via the equivalent circular
    diameter.

    Parameters
    ----------
    areas: array_like
        the sectional areas of the grains

    correct_diameter: a positive integer or float
        this adds the width of the grain boundaries to correct the diameters. If
        correct_diameter is not declared no correction is considered.

    Returns
    -------
    A numpy array with the equivalent circular diameters
    """

    # calculate diameters via equivalent circular diameter
    diameters = 2 * sqrt(areas/pi)

    # diameter correction adding edges (if applicable)
    if correct_diameter != 0:
        diameters += correct_diameter

    return diameters
```

To sum up, the name following the Python keyword `def`, in this example `calc_diameters`, is the name of the function. The sequence of names within the parentheses are the formal parameters of the function (i.e. the inputs). In this case the function has two inputs, the parameter `areas` that correspond with an array containing the areas of the grain profiles previously measured, and the parameter `correct_diameter` that corresponds to a number that sometimes is required for correcting the size of the grains. Note that in this case the default value is set by default to zero. The text between the triple quotation marks provides information about the function, describing the conditions that must be met by the user as well as the output obtained. This information can be accessed from the shell by using the command `help()` and specifying the name of the function within the parentheses or, in the Spyder IDE, by pressing *Ctrl+I* once you wrote the name of the function. Below, it is the code block.

The names of the Python functions in the script are self-explanatory and each one has been implemented to perform a single task. Although there are a lot of functions within the script, we will only need to call less than four functions to obtain the results.

Using the script to visualize and estimate the grain size

Loading the data and extracting the areas of the grain profiles

The first step requires to load the areas of the grain profiles measured in the thin section. It is therefore assumed that they were previously estimated using the *ImageJ* or similar software, and that the results were saved as a txt, csv, or xlsx (Fig. 3). If you do not know how to do this, then go to the section How to measure the grain profile areas with ImageJ.

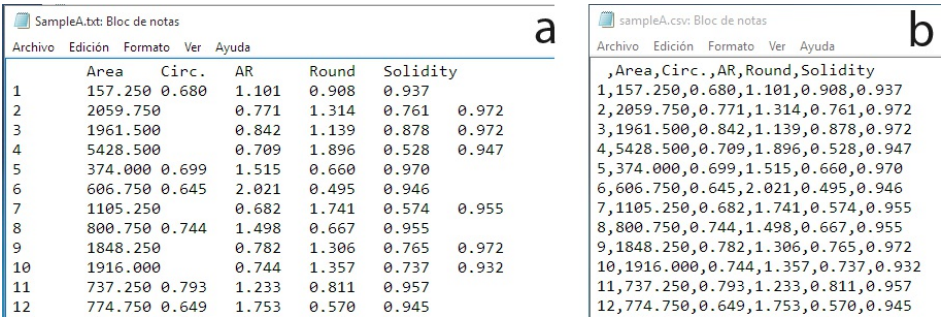


Figure 3. Tabular-like files obtaining from the ImageJ app. At left, the tab-separated txt file. At right, the csv comma-separated version.

People usually perform different types of measures at the same time in the *ImageJ* application ultimately obtained a text file with the data in a spreadsheet-like form (Fig. 3). In this case, we need to extract the information corresponding to the column named 'Area', which is the one that contains the areas of the grain profiles. To do this, the script implements a function named `extract_areas` that automatically do this for us. To invoke this function we write in the shell:

```
>>> areas = extract_areas()
```

where `areas` is just a name for a Python object, known as variable, in which the data extracted will be stored into memory. This will allow us to access and manipulate later the areas of the grain profiles using other functions. Almost any name can be used to create a variable in Python. As an example, if you want to load several datasets, you can name them `areas1`, `areas2` or `my_data1`, `my_data2` and so on. In Python, variable names can contain upper and lowercase letters (the language is case-sensitive), digits and the special character _, but cannot start with a digit. In addition, there are some special keywords reserved for the language (e.g. True, False, if, else, etc.). Do not worry about it, the shell will highlight the word if you are using one of these. The function `extract_areas` is responsible for automatically extracting the areas from the dataset and loading them into the variable defined. Since the script version 1.3.1, you do not need to introduce any parameter/input within the parentheses. Once you press the Enter key, a new window will pop up showing a file selection dialog so that you can search and open the file that contains the dataset. Then, the function will automatically extract the information corresponding to the areas of the grains profiles and store them into the variable. To check that everything is ok, the shell will return the first and last rows of the dataset and the first and last values of the areas extracted as follows:

```
>>> areas = extract_areas()

         Area  Circ.     AR  Round  Solidity
0  1   157.25  0.680  1.101  0.908     0.937
1  2  2059.75  0.771  1.314  0.761     0.972
2  3  1961.50  0.842  1.139  0.878     0.972
3  4  5428.50  0.709  1.896  0.528     0.947
4  5   374.00  0.699  1.515  0.660     0.970
...
              Area  Circ.     AR  Round  Solidity
2656  2657   452.50  0.789  1.235  0.810     0.960
2657  2658  1081.25  0.756  1.446  0.692     0.960
2658  2659   513.50  0.720  1.493  0.670     0.953
2659  2660   277.75  0.627  1.727  0.579     0.920
2660  2661   725.00  0.748  1.351  0.740     0.960

column extracted = [  157.25  2059.75  1961.5  ...,   513.5    277.75   725.  ]
n = 2661
```

The data stored in any variable can be viewed at any time by invoking its name in the shell and pressing the Enter key. Furthermore, both the Canopy and Spyder IDEs have a specific variable explorer to visualize the variables loaded in the current session. The automatic mode assumes that the column containing the areas of the grain profiles is named `'Area'` in the text file (as shown in Fig. 3), which is the default name used by the ImageJ application. If the name of the column is different you can specify it as follows:

```
>>> areas = extract_areas(file_path='auto', col_name='areas')
```

In this example, we introduced two different inputs/parameters within the parentheses. The first one is responsible for defining the file path. In this case it is set by default to 'auto', which means that the automatic mode showed above is on. The second one is the column name (col_name), in this example set to `'areas'` instead of the default `'Area'`. Note that different inputs/parameters are comma-separated.

Another option, which was the only one available in old versions (< v1.3.1), is to introduce the inputs/parameters manually. For this write in the shell:

```
>>> areas = extract_areas('C:/...yourFileLocation.../nameOfTheFile.csv', col_name='areas')
```

in this case we define the file location path in quotes, either single or double, following by the column name if required. If the column name is 'Areas' you just need to write the file path. To avoid problems in Windows OS do not use single backslashes to define the filepath but forward slashes (e.g. "C:/yourfilelocation.../nameofthefile.txt") or double backslashes.

In the case that the user extracted and stored the areas of the grains in a different form from the one proposed here, this is either in a txt or csv file but without a spreadsheet-like form (Fig. 4), there is a Python/Numpy built-in method named `np.genfromtxt()` that can be used to load text data in txt or csv into a variable in a similar way. For example:

```
>>> areas = np.genfromtxt('C:/yourFileLocation/nameOfTheFile.txt')
```

In case you need to skip the first or any other number of lines because there is text or complementary information, then use the *skip_header* parameter as follows:

```
>>> areas = np.genfromtxt('C:/yourFileLocation/nameOfTheFile.txt', skip_header=1)
```

In this example, `skip_header=1` means that the first line in the txt file will be ignored.

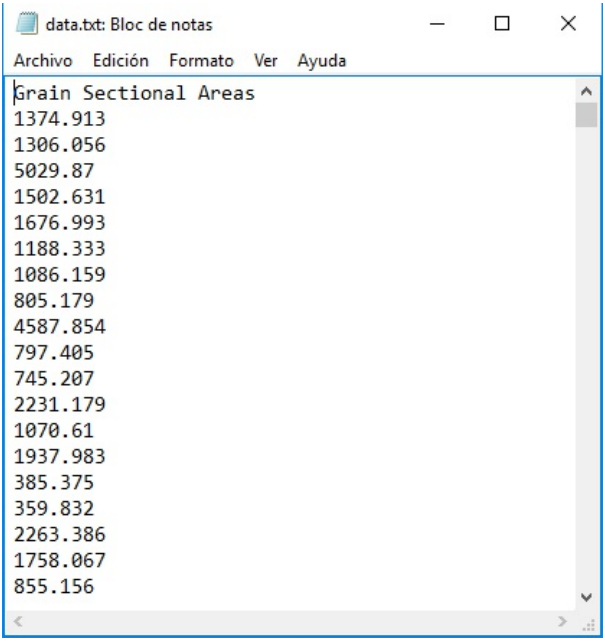


Figure 4. A txt file without spreadsheet-like form. The first line, which is informative, has to be ignored when loading the data

Estimating the apparent diameters from the areas of the grain profiles

The second step is to convert the areas into diameters via the equivalent circular diameter. This is done by a function named `calc_diameters`. To invoke this function we write in the shell:

```
>>> diameters = calc_diameters(areas)
```

The parameter declared within the parenthesis are the name of the variable that contains the areas of the grain profiles. Also, we need to define a new variable to store the diameters estimated from the areas, `diameters` in this example. In some cases, we would need to correct the size of the grain profiles (Fig. 5). For this, you need to add a new parameter within the parentheses:

```
>>> diameters = calc_diameters(areas, correct_diameter=0.05)
```

This example means that for each apparent diameter calculated from the sectional areas, 0.05 will be added. If the parameter `correct_diameter` is not declared within the function, as in the first example, it is assumed that no diameter correction is needed.

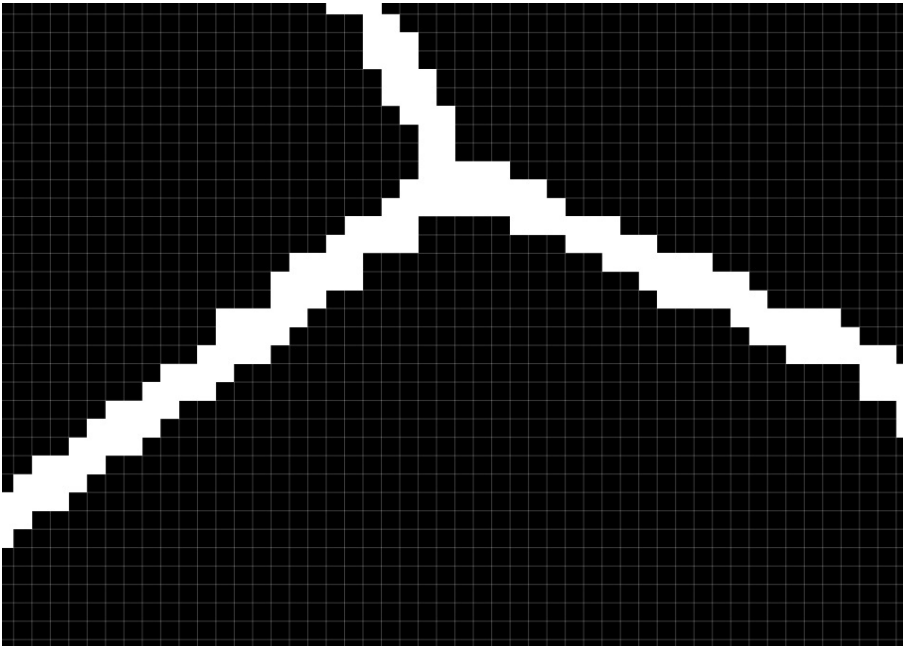


Figure 5. Example of correction of sizes in a grain boundary map. The figure is a raster showing the grain boundaries (in white) between three grains. The squares are the pixels of the image. The boundaries are two pixel wide, approximately. If, for example, each pixel corresponds to 1 micron, we will need to add 2 microns to the diameters estimated from the equivalent circular areas.

Once we estimated and stored the apparent grain sizes, we have several choices: (1) estimate an unidimensional value of grain size for paleopiezometry/paleowattmetry studies, or (2) derive the actual 3D grain size distribution from the population of apparent grain sizes using the Saltykov method (Saltykov, 1967; Sahagian and Proussevitch, 1998) or an extension of the Saltykov method named the two-step method (Lopez-Sanchez and Llana-Fúnez, 2016).

Obtaining apparent grain size measures

For this, we need to call the function `find_grain_size`. This function returns several grain size measures and plots, depending on your needs. The default mode returns a frequency *vs* apparent grain size plot together with the mean, median, and frequency peak grain sizes; the latter using a Gaussian kernel density estimator (see details in Lopez-Sanchez and Llana-Fúnez 2015). Other parameters of interest are also provided, such as the bin size, the number of classes, the method used to estimate the bin size, and the bandwidth used for the Gaussian kde according to the Silverman rule (Silverman 1986). As stated in Lopez-Sanchez and Llana-Fúnez 2015, **to obtain consistent results a minimum of 433 measured grain profiles are required** (error < 4% at a 95% confidence), although we recommend to measure a minimum of 965 when possible (99% confidence).

To estimate a 1D apparent grain size value we write in the shell:

```
>>> find_grain_size(areas, diameters)
```

First note that contrary to what was shown so far, the function is called directly in the shell since it is no

longer necessary to store any data into an object/variable. The inputs are the arrays containing the areas and diameters. After pressing the Enter key, the shell will display something like this:

```
NUMBER WEIGHTED APPROACH (linear apparent grain size):

Mean grain size = 35.79 microns
Standard deviation = xx (1-sigma)
Median grain size = 32.53 microns
Interquartile range (IQR) = 23.98

HISTOGRAM FEATURES
The modal interval is 27.02 - 31.52
The number of classes are 35
The bin size is 4.5 according to the scott rule

GAUSSIAN KERNEL DENSITY ESTIMATOR FEATURES
KDE peak (peak grain size) =  25.24 microns
Bandwidth = 4.01 (Silverman rule)
```

Also, a new window with a plot will appear. The plots will show the apparent grain size distribution and the location of the different grain size measures (Fig. 6). You can save the plots by clicking the floppy disk icon in the tool bar as bitmap (8 file types to choose) or to post-editing in vector image (5 file types to choose). Another interesting option is to modify the appearance of the plot within the *Matplotlib* environment before saving by clicking on the configuration icon in the toolbar.

Although we promote the use of frequency *vs* apparent grain size linear plot (Fig. 6a), the function allows to use other options such as the logarithmic and square-root grain sizes (Figs. 6c, d) or the area-weighted grain size (e.g. Berger et al. 2011) (Fig. 6b). The advantages and disadvantages of the area weighted plot are explained in detail in Lopez-Sanchez and Llana-Fúnez 2015. To do this, we need to specify the type of plot as follows:

```
>>> find_grain_size(areas, diameters, plot='area')
```

in this example setting to use the area-weighted plot. The name of the different plots available are `'lin'` for the linear number-weighted plot (the default), `'area'` for the area-weighted plot (as in the example above), `'sqrt'` for the square-root grain size plot, and `'log'` for the logarithmic grain size plot. Note that the selection of different scales also implies to obtain different grain size estimations. Last, it is very important to note that **the mean of the square root or logarithmic grain sizes is not the same as the square root or the logarithm of the grain size mean**!

The function includes different plug-in methods to estimate an "optimal" bin size, including an automatic mode. The default automatic mode `'auto'` use the Freedman-Diaconis rule when using large datasets (> 1000) and the Sturges rule for small datasets. Other available rules are the Freedman-Diaconis `'fd'`, Scott `'scott'`, Rice `'rice'`, Sturges `'sturges'`, Doane `'doane'`, and square-root `'sqrt'` bin sizes. For more details on the methods see here. We encourage you to use the default method `'auto'`. Empirical experience indicates that the `'doane'` and `'scott'` methods work also pretty well when you have a lognormal- or a normal-like distributions, respectively. To specify the method we write in the shell:

```
>>> find_grain_size(areas, diameters, plot='lin', binsize='doane')
```

note that you have to define first the type of plot you want and that the type of plot will change the appearance of your distribution (Fig. 6). You can also use and *ad hoc* bin/class size of type integer or float

(*i.e.* an irrational number) as follows:

```
>>> find_grain_size(areas, diameters, plot='lin', binsize=10.0)
```

The user-defined bin size can be any number of type integer or float (*i.e.* an irrational number).

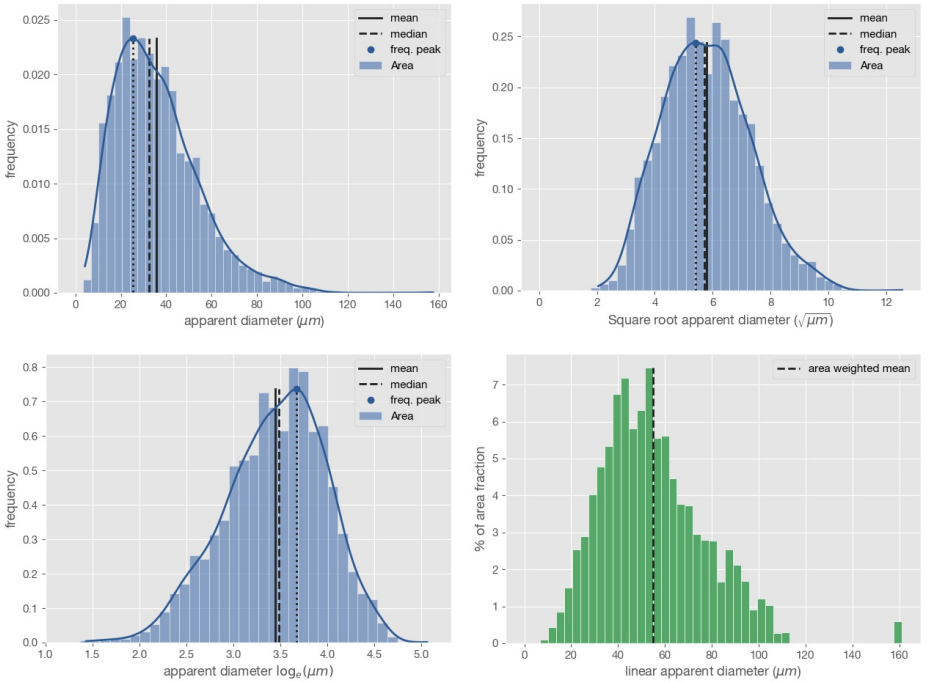


Figure 6. Different apparent grain size vs frequency plots of the same population returned by the find_grain_size function. These include the number-weighted grain size distribution using linear (upper left), square root (upper right), or logarithmic (lower left) scales, and area-weighted distributions (lower right)

Estimating differential stress using piezometric relations (paleopiezometry)

The script includes several functions for estimating differential stress from apparent grain size (i.e. using piezometric relations). This includes mineral phases such as quartz, calcite, olivine and albite (other phases will be added in the future). The function requires measuring the grain size as equivalent circular diameters and entering the apparent grain sizes ***in microns***. There are three different functions for this: `quartz_piezometer` for quartz, `olivine_piezometer` for olivine, and `other_piezometers` for other mineral phases. We provide some examples below:

```
>>> quartz_piezometer(grain_size=5.7, piezometer='Stipp_Tullis')
Ensure that you have entered the apparent grain size as the square root mean!

differential stress = 169.16 MPa

>>> olivine_piezometer(grain_size=35, piezometer='Jung_Karato')
Ensure that you have entered the apparent grain size as the linear scale mean!
```

```
differential stress = 208.8 MPa

>>> other_piezometers(grain_size=5.7, piezometer='calcite_Rutter_SGR')
Ensure that you have entered the apparent grain size as the square root mean!

differential stress = 175.72 MPa
```

It is key to note that different piezometers require entering **different types of apparent grain sizes** to provide meaningful estimates of differential stress. For example, the piezometer relation of Stipp and Tullis (2003) requires entering the grain size as *the square root mean grain size from equivalent circular diameters with no stereological correction* (i.e. mean sqrt apparent grain size), and so on. Table 1 show all the implemented piezometers in GrainSizeTools v1.4.2 and the apparent grain size required for each one. Despite some piezometers were originally calibrated using linear intercepts (LI), the script will always require entering a specific apparent grain size measured as equivalent circular diameters (ECD), the script will automatically convert this ECD value to linear intercepts using the De Hoff and Rhines (1968) empirical relation.

Table 1. Relation of piezometers put in the GrainSizeTools script and the apparent grain size required to obtain meaningful differential stress estimates

Piezometer	Apparent grain size†	DRX mechanism	Phase	Reference
`'Stipp_Tullis'`	Square root mean	Regimes 2, 3	Quartz	Stipp & Tullis (2003)
`'Stipp_Tullis_BLG'`	Square root mean	Regime 1 (BLG)	Quartz	Stipp & Tullis (2003)
`'Holyoke'`	Square root mean	Regimes 2, 3	Quartz	Holyoke and Kronenberg (2010)
`'Holyoke_BLG'`	Square root mean	Regime 1 (BLG)	Quartz	Holyoke and Kronenberg (2010)
`'Shimizu'` ‡	Logarithmic median	SGR + GBM	Quartz	Shimizu (2008)
`'Cross'` and `'Cross_hr'`	Square root mean	BLG, SGR	Quartz	Cross et al. (2017)
`'Twiss'` §	Logarithmic mean	Regimes 2, 3	Quartz	Twiss (1977)
`'calcite_Rutter_SGR'`	Square root mean	SGR	Calcite	Rutter (1995)

`'calcite_Rutter_GBM'`	Square root mean	GBM	Calcite	Rutter (1995)
`'albite_PostT_BLG'` §	Median (linear scale)	BLG	Albite	Post and Tullis (1999)
`'VanderWal_wet'` §	Mean (linear scale)		Olivine, wet	Van der Wal et al. (1993)
`'Jung_Karato'` §	Mean (linear scale)	BLG	Olivine, wet	Jung & Karato (2001)

† *Apparent grain size measured as equivalent circular diameters (ECD) with no stereological correction and reported in microns either in linear, square root or logarithmic scales*
‡ *Shimizu piezometer requires to provide the temperature during deformation in K*
§ *These piezometers were originally calibrated using linear intercepts (LI) instead of ECD*

For more details on the piezometers and the assumption made use the command `help()` in the console as follows:

```
>>> help(quartz_piezometer)
```

The constant values as put in the script are described in Table 2 below.

Table 2. Parameters relating the apparent size of dynamically recrystallized grains and the differential stress using a relation in the form $d = A\sigma^{-p}$ or $\sigma = Bd^{-m}$

Reference	phase	DRX	A†,‡	p†	B†,‡	m†
Stipp and Tullis (2003)	quartz	Regimes 2, 3	3630.8	1.26	669.0	0.79
Stipp and Tullis (2003)	quartz	Regime 1	78	0.61	1264.1	1.64
Holyoke & Kronenberg (2010)§	quartz	Regimes 2, 3	2451	1.26	490.3	0.79
Holyoke & Kronenberg (2010)§	quartz	Regime 1	39	0.54	883.9	1.85
Shimizu (2008)	quartz	SGR + GBM	1525	1.25	352	0.8
Cross et al. (2017)¶	quartz	Regimes 2, 3	8128.3	1.41	593.0	0.71
Cross et al. (2017)¶			16595.9	1.59	450.9	0.63

	quartz, hr	Regimes 2, 3				
Twiss (1977)	quartz	Regimes 2, 3	12.3	1.47	5.5	0.68
Jung and Karato (2001)	olivine, wet	BLG	25704	1.18	5461.03	0.85
Van der Wal et al. (1993)	olivine, wet		0.0148	1.33	0.0425	0.75
Rutter (1995)	calcite	SGR	2026.8	1.14	812.83	0.88
Rutter (1995)	calcite	GBM	7143.8	1.12	2691.53	0.89
Post and Tullis (1999)	albite	BLG	55	0.66	433.4	1.52

*† **B** and **m** relate to **A** and **p** as follows: $B = A^{1/p}$ and $m = 1/p$*
*‡ **A** and **B** are in [μm $MPa^{p, m}$] excepting Twiss (1977) in [mm $MPa^{p, m}$] and Van der Wal et al. (1993) in [m $MPa^{p, m}$]*
§ Holyoke and Kronenberg (2010) is a linear recalibration of the Stipp and Tullis (2003) piezometer
¶ Cross et al. (2017) reanalysed the samples of Stipp and Tullis (2003) using EBSD data for reconstructing the grains. Specifically, they use grain maps with a 1 μm and a 200 nm (hr - high resolution) step sizes . This is the preferred piezometer for quartz when grain size data comes from EBSD maps

Estimating a robust confidence interval

As pointed out in the scope section, the optimal approach is to obtain several measures of stress or grain sizes and then estimate a confidence interval. Since v1.4.4+, the script implements a function called `confidende_interval` for this. The script assume that the sample size is small (< 10) and hence it uses the student's t-distribution with n-1 degrees of freedom to estimate a robust confidence interval. For large datasets the t-distribution approaches the normal distribution so you can also use this method for large datasets. The function has two inputs, the dataset, required, and the confidence interval, optional and set at 0.95 by default. For example:

```
>>> my_results = [165.3, 174.2, 180.1]
>>> confidence_interval(data=my_results, confidence=0.95)
```

The function will return the following information in the console:

```
Confidence set at 95.0 %
Mean = 173.2 ± 18.51
Max / min = 191.71 / 154.69
Coefficient of variation = 10.7 %
```

The coefficient of variation express the confidence interval in percentage respect to the mean and thus it can be used to compare confidence intervals between samples with different mean values.

Derive the actual 3D distribution of grain sizes from thin sections

The function responsible to unfold the distribution of apparent grain sizes into the actual 3D grain size distribution is named `derive3D`. The script implements two methods to do this, the Saltykov and the two-step methods. The Saltykov method is the best option for exploring the dataset and for estimating the volume of a particular grain size fraction. The two-step method is suitable to describe quantitatively the shape of the grain size distribution assuming that they follow a lognormal distribution. This means that the two-step method only yield consistent results when the population of grains considered are completely recrystallized or when the non-recrystallized grains can be previously discarded using shape descriptors or any other relevant paramater such as the density of dislocations. It is therefore necessary to check first whether the linear distribution of grain sizes is unimodal and lognormal-like (i.e. skewed to the right as in the example shown in figure 7). For more details see Lopez-Sanchez and Llana-Fúnez (2016).

Applying the Saltykov method

To derive the actual 3D population of grain sizes using the Saltykov method (Saltykov, 1967; Sahagian and Proussevitch, 1998), we need to call the function `derive3D` as follows:

```
>>> derive3D(diameters, numbins=14)
```

Since the Saltykov method uses the histogram to derive the actual 3D grain size distribution, the inputs are an array containing the population of apparent diameters of the grains and the number of classes. If the number of classes is not declared is set to ten by default. The user can use any positive **integer** value. However, it is usually advisable to choose a number not exceeding 20 classes (see later for details). After pressing the Enter key, the function will return something like this in the shell:

```
sample size = 2661
bin size = 11.26

A file named Saltykov_output.csv containing the midpoints and the class frequencies,
was generated
```

The text indicates the sample size, the bin size, and that a tabular-like csv file containing different numerical values was generated. Also, a window will pop up showing two plots (Fig 7). On the left it is the frequency plot with the estimated 3D grain size distribution in the form of a histogram, and on the right the volume-weighted cumulative density curve. The latter allows the user to estimate qualitatively the percentage of volume occupied by a defined fraction of grain sizes. If the user wants to estimate quantitatively the volume of a particular grain fraction (i.e. the volume occupied by a fraction of grains less or equal to a certain value) we need to add a new parameter within the function as follows:

```
>>> derive3D(diameters, numbins=12, set_limit=40)
```

In the example above, the grain fraction is set to 40 microns, which means that the script will also return an estimation of the percentage of volume occupied by all the grain fractions up to 40 microns. A new line will appear in the shell:

```
volume fraction (up to 40 microns) = 22.8 %
```

As a cautionary note, if we use a different number of bins/classes, in this example set randomly at 12, we will obtain slightly different volume estimates. This is normal due to the inaccuracies of the Saltykov method. In any event, Lopez-Sanchez and Llana-Fúnez (2016) proved that the absolute differences between the volume estimations using a range of classes between 10 and 20 are below ±5. This means that to stay safe we should always take an absolute error margin of ±5 in the volume estimations.

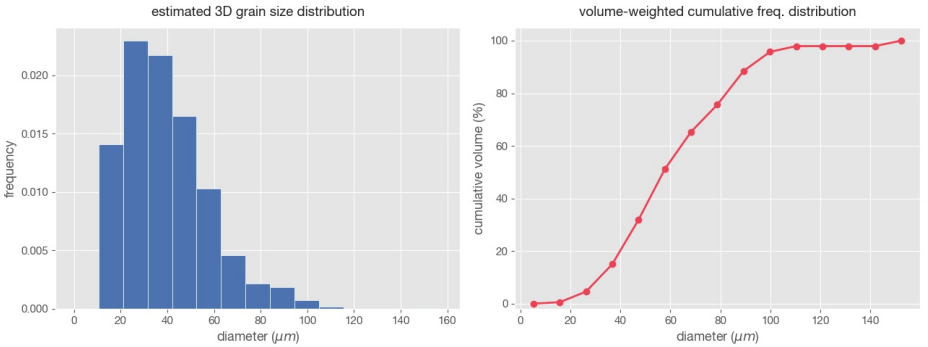


Figure 7. The derived 3D grain size distribution and the volume-weighted cumulative grain size distribution using the Saltykov method.

Lastly, due to the nature of the Saltykov method, the smaller the number of classes the better the numerical stability of the method, and the larger the number of classes the better the approximation of the wanted distribution. Ultimately, the strategy to follow is about finding the maximum number of classes (i.e. the best resolution) that produces stable results. Based on experience, previous works proposed to use between 10 to 15 classes (e.g. Exner 1972), although depending on the quality and the size of the dataset it seems that it can be used safely up to 20 classes or even more (e.g. Heilbronner and Barret 2014, Lopez-Sanchez and Llana-Fúnez 2016). Yet, no method (i.e. algorithm) appears to be generally best for choosing an optimal number of classes (or bin size) for the Saltykov method. Hence, the only way to find the maximum number of classes with consistent results is to use a trial and error strategy and observe if the appearance is coherent or not. As a last cautionary note, **the Saltykov method requires large samples (n ~ 1000 or larger)** to obtain consistent results, even when using a small number of classes.

Applying the two-step method

To estimate the shape of the 3D grain size distribution, the `derive3D` function implements a method called "the two-step method" (Lopez-Sanchez and Llana-Fúnez, 2016). This method assumes that the actual 3D population of grain sizes follows a lognormal distribution, a common distribution observed in recrystallized aggregates. Hence, make sure that the aggregate or the studied area within the rock/alloy/ceramic is completely recrystallized. The method applies a non-linear least squares algorithm to fit a lognormal distribution on top of the Saltykov method using the midpoints of the different classes. The method return two parameters, the **MSD** and the theoretical **median**, both enough to fully describe a lognormal distribution at their original (linear) scale, and the uncertainty associated with these estimates (see details in Lopez-Sanchez and Llana-Fúnez, 2016). In addition, it also returns a frequency plot showing the probability density function estimated (Fig. 8). In particular, the **MSD value** allows to describe the shape of the lognormal distribution independently of the scale (i.e. the range) of the grain size distribution, which is very convenient for comparative purposes. To apply the two-step method we need to invoke the function `derive3D` as follows:

```
>>> derive3D(diameters, numbins=15, set_limit=None, fit=True)
```

Note that in this case we include a new parameter named `fit` that it is set to `True` with the "T" capitalized (it is set to `False` by default). The function will return something like this in the shell:

```
sample size = 2661
bin size = 10.44

Optimal coefficients:
MSD (shape) = 1.69 ± 0.07
Median (location) = 35.51 ± 1.73 (caution: not fully realiable)

A file named twoStep_output.csv containing the midpoints, class frequencies,
and cumulative volumes was generated
```

Sometimes, the least squares algorithm will fail at fitting the lognormal distribution to the unfolded data (e.g. Fig. 8b). This is due to the algorithm used to find the optimal MSD and median values, the Levenberg–Marquardt algorithm (Marquardt, 1963), only converges to a global minimum when their initial guesses are already somewhat close to the final solution. Based on our experience in quartz aggregates, the initial guesses were set by default at 1.2 and 25.0 for the MSD and median values respectively. However, if the algorithm fails it is possible to change the default values by adding a following parameter:

```
>>> derive3D(diameters, numbins=15, set_limit=None, fit=True, initial_guess=True)
```

When the `initial_guess` parameter is set to `True`, the script will ask to set a new starting values for both parameters (it also indicates which are the default ones). Based in our experience, a useful strategy is to let the MSD value in its default value (1.2) and decreasing the median value every five units until the fitting procedure yield a good fit (*e.g.* 25 -> 20 -> 15...) (Fig. 8).

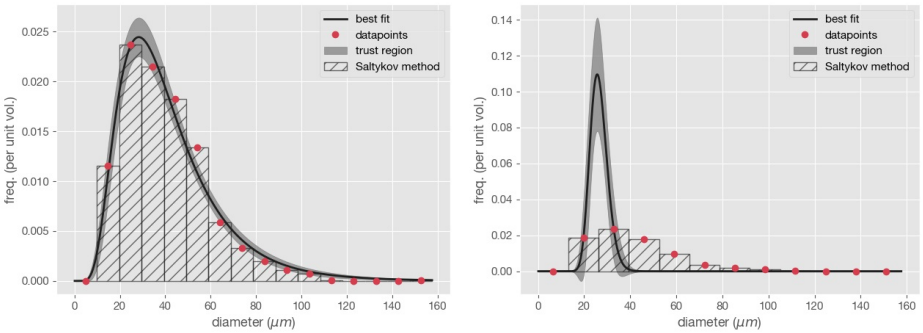


Figure 8. Plots obtained using the two-step method. At left, an example with the lognormal pdf well fitted to the data points. The shadow zone is the trust region for the fitting procedure. At right, an example with a wrong fit due to the use of unsuitable initial guess values. Note the discrepancy between the data points and the line representing the best fitting.

Comparing different grain size populations using box plots

Box (or box-and-whisker) plot is a non-parametric method to display numerical datasets through their quartiles and median, being a very efficient way for comparing **unimodal** datasets graphically. Figure 9 show the different elements represented in a typical box plot.

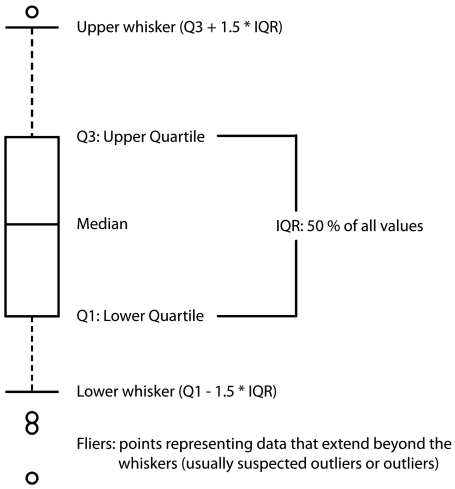


Figure 9. Box plot elements

To create a box plot using the Matplotlib library we need to create first a variable with all the data sets to represent. For this we create a Python list as follows (variable names have been chosen for convenience):

```
>>> all_data = [dataset1, dataset2, dataset3, dataset4] # Note that a Python list is a list of elements within brackets separated by commas
```

Then we create the plot (Fig. 10):

```
>>> plt.boxplot(all_data)
>>> plt.show() # write this and click return if the plot did not appear automatically
```

To create a more convenience plot (Fig. 10) we propose using the following **optional** parameters:

```
# First make a list specifying the labels of the samples (this is optional). Ensure that the number of items in the brackets coincide with the number of datasets to plot.
>>> label_list = ['SampleA', 'SampleB', 'SampleC', 'SampleD']
# Then make the plot ading the following instructions
>>> plt.boxplot(all_data, vert=False, meanline=True, showmeans=True, labels=label_list)
>>> plt.xlabel('apparent diameter ($\mu m$)') # add the x-axis label
>>> plt.show() # write this and click return if the plot did not appear automatically
```

The parameters defined in the boxplot are:

- `vert` : if False makes the boxes horizontal instead of vertical (it is True by default).
- `meanline` and `showmeans` : if True will show the location of the mean within the plots.
- `labels` : add labels to the different datasets.

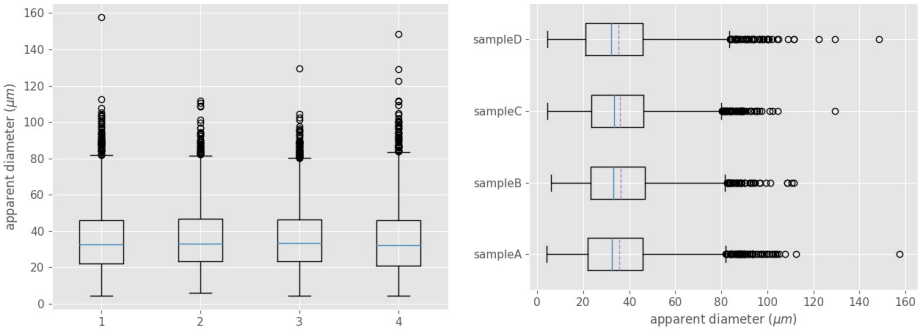


Figure 10. Box plot comparing four unimodal datasets obtained from the same sample but located in different places along the thin section. At left, a box plot with the default appearance. At right, the same box plot with the optional parameters showing above. Dashed lines are the mean. Note that the all the datasets show similar median, means, IQRs, and whisker locations. In contrast, the fliers (points) approximately above 100 microns vary greatly.

Other methods of interest

Obtain common descriptive statistic parameters

```
>>> mean(array_name)  # Estimate the mean
>>> std(array_name)  # Estimate the standard deviation
>>> median(array_name)  # Estimate the median
>>> iqr(array_name)  # Estimate the interquartile range
```

Merging datasets

A useful *Numpy* method to merge two or more datasets is called `hstack()`, which stack arrays in sequence as follows (*please, note the use of double parenthesis*):

```
>>> np.hstack((name of the array1, name of the Array2,...))
```

As an example if we have two different datasets and we want to merge the areas and the diameters estimated in a single variable we write into the Python shell (variable names are just random examples):

```
>>> all_areas = np.hstack((areas1, areas2))
>>> all_diameters = np.hstack((diameters1, diameters2))
```

Note that in this example we merged the original datasets into two new variables named `all_areas` and `all_diameters` respectively. Therefore, we do not overwrite any of the original variables and we can used them later if required. In contrast, if you use a variable name already defined:

```
>>> areas1 = np.hstack((areas1, areas2))
```

The variable `areas1` is now a new array with the values of the two datasets, and the original dataset stored in the variable `areas1` no longer exists since these variables (strictly speaking Numpy arrays) are mutable Python objects.

Loading several datasets: the fastest (appropriate) way

If you need to load a large number of datasets, you probably prefer not having to search across multiple folders in the file dialog window or to manually specify the absolute file paths of each file. Python establishes by default a current working directory in which all the files can be accessed directly by specifying just the name of the file (or a relative path if they are in a sub-folder). For example, if the current working directory is `c:/user/yourname`, you no longer need to specify the entire file path for the files stored within this directory. For example, to load a csv file named 'my_sample.csv' stored in that location you just need to write:

```
>>> areas = extract_areas('my_sample.csv')
```

When you run the script for the first time, your current working directory will appear in the Python shell. Also, you can retrieve your current working directory at any time by typing in the shell `os.getcwd()`, as well as to modify it to another path using the function `os.chdir('new default path')`. The same rules apply when using the `np.genfromtxt` method.

> Note: usually the current working directory is the same directory where the script is located (although this depends on the Python environment you have installed). Hence, sometimes it is a good idea to locate the scrip in the same directory where the datasets are located.

GST script quick tutorial

> This is a quick tutorial showing typical commands to interact with the GST script in the shell.
> You can copy and paste the command lines directly into the shell. Variable names have
> been chosen for convenience, but you can use any other name if desired.
> The commands below require the v1.4.3 or higher.

Loading the data and extracting the areas of the grain profiles

```
# The automatic (preferred) mode
areas = extract_areas()

# Using the automatic mode but defining a column name different from the default 'Area'
areas = extract_areas(file_path='auto', col_name='the name of the column to be extracted')

# The manual mode: Defining the file path
areas = extract_areas('C:/...yourFileLocation.../nameOfTheFile.csv')

# Loading data from a single-column text file (the skip_header parameter is optional)
areas = np.genfromtxt('C:/yourFileLocation/nameOfTheFile.txt', skip_header=1)
```

Estimating the apparent diameters from the areas of the grain profiles

```
# Estimating the equivalent circular diameter from the grain profiles areas
diameters = calc_diameters(areas)

# Using the diameter correction
diameters = calc_diameters(areas, correct_diameter=0.05) # 0.05 microns will be added
```

Obtaining apparent grain size measures

```
# Default mode (linear grain size distribution, automatic bin size or number of classes)
find_grain_size(areas, diameters)

# Using the area-weighted grain size with automatic bin size
find_grain_size(areas, diameters, plot='area')

# Using the logarithmic grain size with automatic bin size
find_grain_size(areas, diameters, plot='log')

# Using the square-root grain size with automatic bin size
find_grain_size(areas, diameters, plot='sqrt')

# Using a specific plug-in method for estimating the optimal bin size or number of classes
# Plug-in methods include: Freedman-Diaconis:'fd', Scott:'scott', Rice:'rice',
# Sturges:'sturges', Doane:'doane', and square-root:'sqrt' bin sizes
find_grain_size(areas, diameters, plot='lin', binsize='doane')

# Using an ad hoc bin size
find_grain_size(areas, diameters, plot='lin', binsize=10.0)
```

Estimate differential stresses using paleopizometers

```
# Estimate differential stress using quartz and the piezometric
# piezometer of Stipp and Tullis (2003)
quartz_piezometer(grain_size=9.0, piezometer='Stipp_Tullis')

# Estimate differential stress using olivine and the piezometric
# piezometer of Jung and Karato (2001)
olivine_piezometer(grain_size=9.0, piezometer='Jung_Karato')

# Estimate differential stress in other mineral/materials
# piezometer of Rutter (1995)
other_piezometer(grain_size=9.0, piezometer='calcite_Rutter_SGR')
```

Derive the actual 3D grain size distribution from the apparent grain size distribution

```
# Using the Saltykov method with ad hoc number of classes
# if numbins is not declared is set to ten classes by default
derive3D(diameters, numbins=12)

# Estimating the volume of a particular grain size fraction
derive3D(diameters, numbins=12, set_limit=40)

# Using the two-step method
derive3D(diameters, numbins=12, set_limit=None, fit=True)

# Using ad hoc initial guess values for correcting a wrong fit
# The script will ask you about the new guessing values
derive3D(diameters, numbins=12, set_limit=None, fit=True, initial_guess=True)
```

How to measure the areas of the grain profiles with ImageJ

> **Before you start:** This tutorial assumes that you have installed the ImageJ application. If this is not the case, go here to download and install it. You can also install different flavours of the ImageJ application that will work in a similar way (see here for a summary) . As a cautionary note, this is not a detailed tutorial on image analysis methods using ImageJ, but a quick systematic tutorial on how to measure the areas of the grain profiles from a thin section to later estimate the grain size and grain size distribution using the GrainSizeTools script. If you are interested in image analysis methods (e.g. grain segmentation techniques, shape characterization, etc.) you should have a look at the list of references at the end of this tutorial.

Previous considerations on the Grain Boundary Maps

Grain size studies in rocks are usually based on measures performed in thin sections (2D data) through image analysis. Since the methods implemented in the GrainSizeTools script are based on the measure of the areas of the grain profiles, the first step is therefore to obtain a grain boundary map from the thin section (Fig. 1).

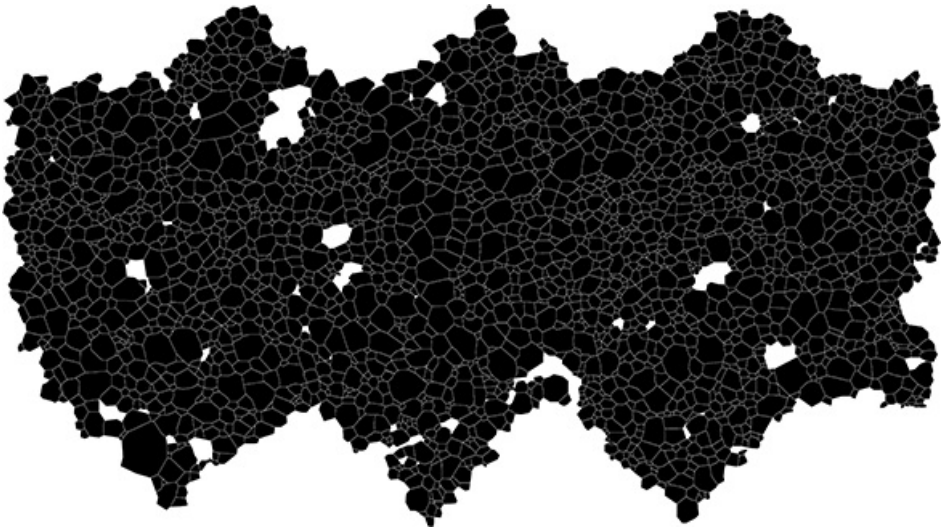


Figure 1. An example of a grain boundary map

Nowadays, these measures are mostly made on digital images made by pixels (e.g. Heilbronner and Barret 2014), also known as raster graphics image. You can obtain some information on raster graphics here. For example, in a 8-bit grayscale image -the most used type of grayscale image-, each pixel contains three values: information about its location in the image -their x and y coordinates- and its 'grey' value in a range that goes from 0 (white) to 256 (black) (i.e. it allows 256 different grey intensities). In the case of a grain boundary map (Fig. 1), we usually use a binary image where only two possible values exist, 0 for white pixels and 1 for black pixels.

One of the key points about raster images is that they are resolution dependent, which means that each

pixel have a physical dimension. Consequently, the smaller the size of the pixel, the higher the resolution. The resolution depends on the number of pixels per unit area or length, and it is usually measured in pixel per (square) inch (PPI) (more information about Image resolution and Pixel density). This concept is key since the resolution of our raw image -the image obtained directly from the microscope- will limit the precision of the measures. Known the size of the pixels is therefore essential and it will allow us to set the scale of the image to measure of the areas of the grain profiles. In addition, it will allow us to later make a perimeter correction when calculating the equivalent diameters from the areas of the grain profiles. So be sure about the image resolution at every step, from the "raw" image you get from the microscope until you get the grain boundary map.

> Note: It is important not to confuse the pixel resolution with the actual spatial resolution of the image. The spatial resolution is the actual resolution of the image and it is limited physically not by the number of pixels per unit area/length. For example, conventional SEM techniques have a maximum spatial resolution of 50 to 100 nm whatever the pixels in the image recorded. Think in a digital image of a square inch in size and made of just one black pixel (i.e. with a resolution of ppi = 1). If we double the resolution of the image, we will obtain the same image but now formed by four black pixels instead of one. The new pixel resolution per unit length will be ppi = 2 (or 4 per unit area), however, the spatial resolution of the image remains the same. Strictly speaking, the spatial resolution refers to the number of independent pixel values per unit area/length.

The techniques that make possible the transition from a raw image to a grain boundary map, known as grain segmentation, are numerous and depend largely on the type of image obtained from the microscope. Thus, digital images may come from transmission or reflected light microscopy, semi-automatic techniques coupled to light microscopy such as the CIP method (e.g. Heilbronner 2000), electron microscopy either from BSD images or EBSD grain maps, or even from electron microprobes through compositional mapping. All this techniques produce very different images (i.e. different resolutions, colour *vs* grey scale, nature of the artefacts, grain size boundary *vs* phase maps, etc.). The presentation of this segmentation techniques is beyond the scope of this tutorial and the reader is referred to the references cited at the end of this document and, particularly, to the books written by Russ (2011) and Heilbronner and Barret (2014). This tutorial is focused instead on the features of the grain boundary maps by itself not in how to convert the raw images to grain boundary maps using manual, automatic or semi-automatic grain segmentation.

Once the grain segmentation is done, it is crucial to ensure that at the actual pixel resolution the grain boundaries have a width of two or three pixels (Fig. 2). This will prevent the formation of undesirable artefacts since when two black pixels belonging to two different grains are adjacent to each other, both grains will be considered the same grain by the image analysis software.

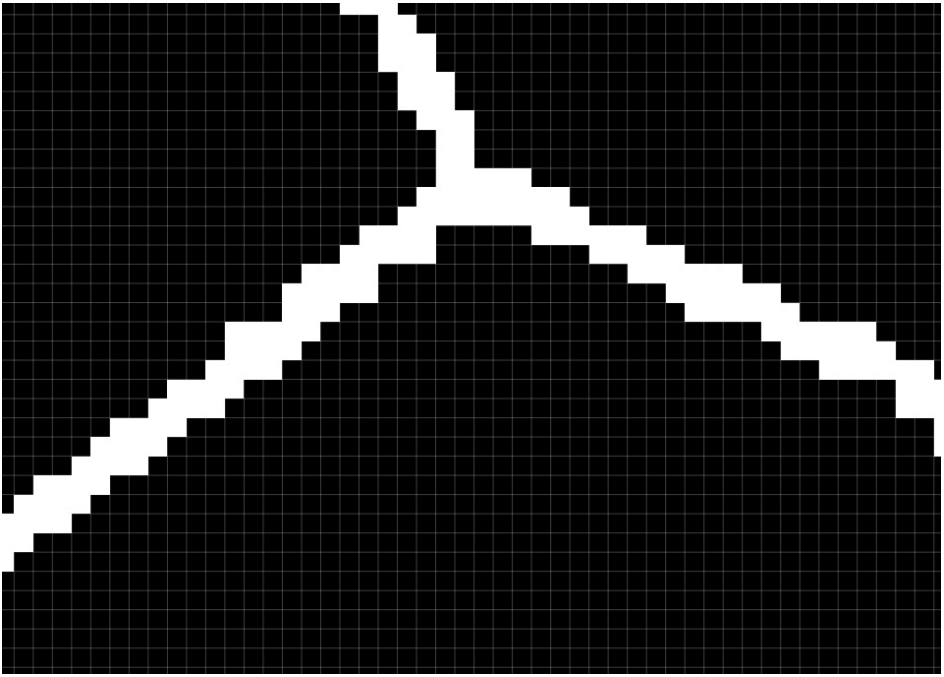


Figure 2. Detail of grain boundaries in a grain boundary map. The figure shows the boundaries (in white) between three grains in a grain boundary map. The squares represent the pixels in the image. The boundaries are two pixels wide approximately.

Measuring the areas of the grain profiles

1. Open the grain boundary map with the ImageJ application

2. To measure the areas of the grain profiles it is first necessary to convert the grain boundary map into a binary image. If this was not done previously, go to `Process>Binary` and click on `Make binary`. Also, make sure that the areas of grain profiles are in black and the grain boundaries in white and not the other way around. If not, invert the image in `Edit>Invert`.

3. Then, it is necessary to set the scale of the image. Go to `Analize>Set Scale`. A new window will appear (Fig. 3). To set the scale, you need to know the size of a feature, such as the width of the image, or the size of an object such as a scale bar. The size of the image in pixels can be check in the upper left corner of the window, within the parentheses, containing the image. To use a particular object of the image as scale the procedure is: i) Use the line selection tool in the tool bar (Fig. 3) and draw a line along the length of the feature or scale bar; ii) go to `Analize>Set Scale`; iii) the distance of the drawn line in pixels will appear in the upper box, so enter the dimension of the object/scale bar in the 'known distance' box and set the units in the 'Unit length' box; iv) do not check 'Global' unless you want that all your images have the same calibration and click ok. Now, you can check in the upper left corner of the window the size of the image in microns (millimetres or whatever) and in pixels.

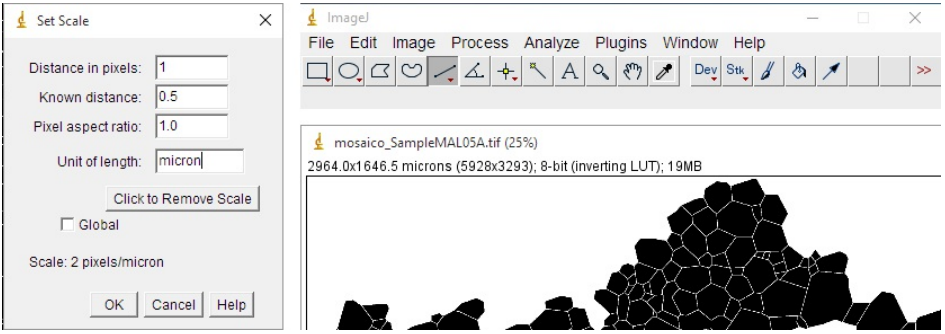


Figure 3. At left, the Set Scale window. In the upper right, the ImageJ menu and tool bars. The line selection tool is the fifth element from the left (which is actually selected). In the bottom right, the upper left corner of the window that contains the grain boundary map. The numbers are the size in microns and the size in pixels (in brackets).

4. The next step requires to set the measurements to be done. For this, go to `Analize>Set Measurements` and a new window will appear. Make sure that 'Area' is selected. You can also set at the bottom of the window the desired number of decimal places. Click ok.

5. To measure the areas of our grain profiles we need to go to `Analize>Analize Particles`. A new window will appear with different options (Fig. 4). The first two are for establishing certain conditions to exclude anything that is not an object of interest in the image. The first one is based on the size of the objects in pixels by establishing a range of size. We usually set a minimum of four pixels and the maximum set to infinity to rule out possible artefacts hard to detect by the eye. This ultimately depends on the quality and the nature of your grain boundary map. For example, people working with high-resolution EBSD maps usually discard any grain with less than ten pixels. The second option is based on the roundness of the grains. We usually leave the default range values 0.00-1.00, but again this depends on your data and purpose. For example, the roundness parameter could be useful to differentiate between non-recrystallized and recrystallized grains in some cases. Just below, the 'show' drop-down menu allows the user to obtain different types of images when the particle analysis is done. We usually set this to 'Outlines' to obtain an image with all the grains measured outlined and numbered, which can be useful later to check the data set. Finally, the user can choose between different options. In our case, it is just necessary to select 'Display results'. Click ok.

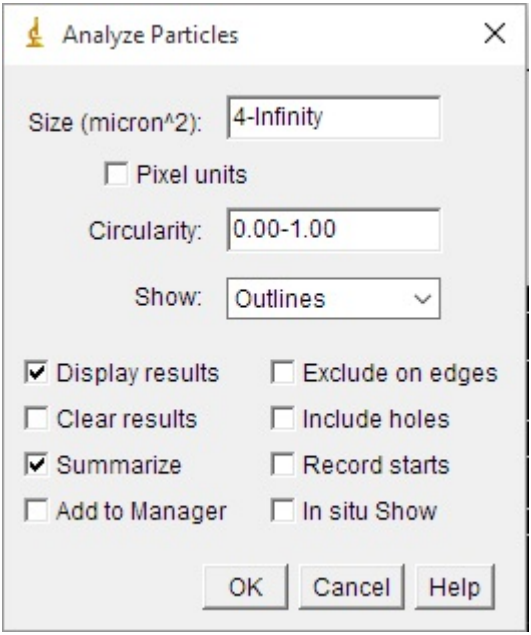
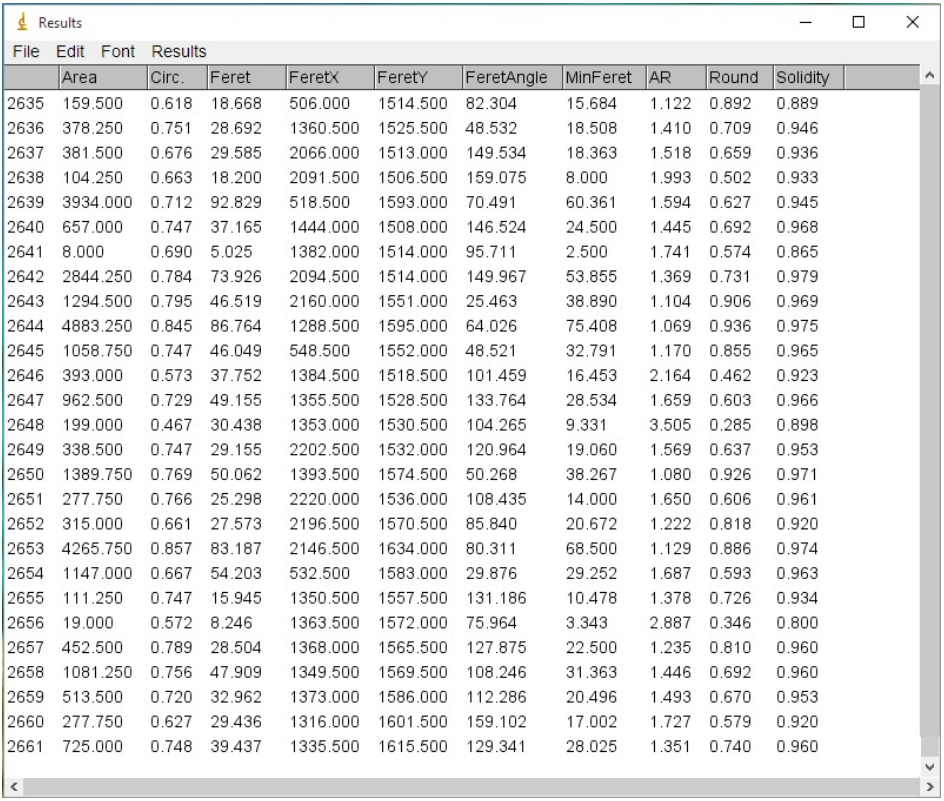


Figure 4. Analyze particles window showing the different options

6. After a while, several windows will appear. At least, one containing the results of the measures (Fig. 5), and other containing the image with the grains outlined and numbered. Note that the numbers displayed within the grains in the image correspond to the values showed in the first column of the results. To save the image go to the ImageJ menu bar, click on `File>Save As`, and choose the file type you prefer (we encourage you to use PNG or TIFF for such type of image). To save the results we have different options. In the menu bar of the window containing the results, go to `Results>Options` and a new window will appear (Fig. 6). In the third line, you can choose to save the results as a text (.txt), csv comma-separated (.csv) or excel (.xls) file types. We encourage you to choose either *txt* or *csv* since both are widely supported formats to exchange tabular data. Regarding the 'Results Table Options' at the bottom, make sure that 'Save column headers' are selected since this headers will be used by the GrainSizeTools script to automatically extract the data from the column 'Area'. Finally, in the same window go to `File>Save As` and choose a name for the file. You are done.



	Area	Circ.	Feret	FeretX	FeretY	FeretAngle	MinFeret	AR	Round	Solidity
2635	159.500	0.618	18.668	506.000	1514.500	82.304	15.684	1.122	0.892	0.889
2636	378.250	0.751	28.692	1360.500	1525.500	48.532	18.508	1.410	0.709	0.946
2637	381.500	0.676	29.585	2066.000	1513.000	149.534	18.363	1.518	0.659	0.936
2638	104.250	0.663	18.200	2091.500	1506.500	159.075	8.000	1.993	0.502	0.933
2639	3934.000	0.712	92.829	518.500	1593.000	70.491	60.361	1.594	0.627	0.945
2640	657.000	0.747	37.165	1444.000	1508.000	146.524	24.500	1.445	0.692	0.968
2641	8.000	0.690	5.025	1382.000	1514.000	95.711	2.500	1.741	0.574	0.865
2642	2844.250	0.784	73.926	2094.500	1514.000	149.967	53.855	1.369	0.731	0.979
2643	1294.500	0.795	46.519	2160.000	1551.000	25.463	38.890	1.104	0.906	0.969
2644	4883.250	0.845	86.764	1288.500	1595.000	64.026	75.408	1.069	0.936	0.975
2645	1058.750	0.747	46.049	548.500	1552.000	48.521	32.791	1.170	0.855	0.965
2646	393.000	0.573	37.752	1384.500	1518.500	101.459	16.453	2.164	0.462	0.923
2647	962.500	0.729	49.155	1355.500	1528.500	133.764	28.534	1.659	0.603	0.966
2648	199.000	0.467	30.438	1353.000	1530.500	104.265	9.331	3.505	0.285	0.898
2649	338.500	0.747	29.155	2202.500	1532.000	120.964	19.060	1.569	0.637	0.953
2650	1389.750	0.769	50.062	1393.500	1574.500	50.268	38.267	1.080	0.926	0.971
2651	277.750	0.766	25.298	2220.000	1536.000	108.435	14.000	1.650	0.606	0.961
2652	315.000	0.661	27.573	2196.500	1570.500	85.840	20.672	1.222	0.818	0.920
2653	4265.750	0.857	83.187	2146.500	1634.000	80.311	68.500	1.129	0.886	0.974
2654	1147.000	0.667	54.203	532.500	1583.000	29.876	29.252	1.687	0.593	0.963
2655	111.250	0.747	15.945	1350.500	1557.500	131.186	10.478	1.378	0.726	0.934
2656	19.000	0.572	8.246	1363.500	1572.000	75.964	3.343	2.887	0.346	0.800
2657	452.500	0.789	28.504	1368.000	1565.500	127.875	22.500	1.235	0.810	0.960
2658	1081.250	0.756	47.909	1349.500	1569.500	108.246	31.363	1.446	0.692	0.960
2659	513.500	0.720	32.962	1373.000	1586.000	112.286	20.496	1.493	0.670	0.953
2660	277.750	0.627	29.436	1316.000	1601.500	159.102	17.002	1.727	0.579	0.920
2661	725.000	0.748	39.437	1335.500	1615.500	129.341	28.025	1.351	0.740	0.960

Figure 5. The results windows showing all the measures done on the grains by the ImageJ application.

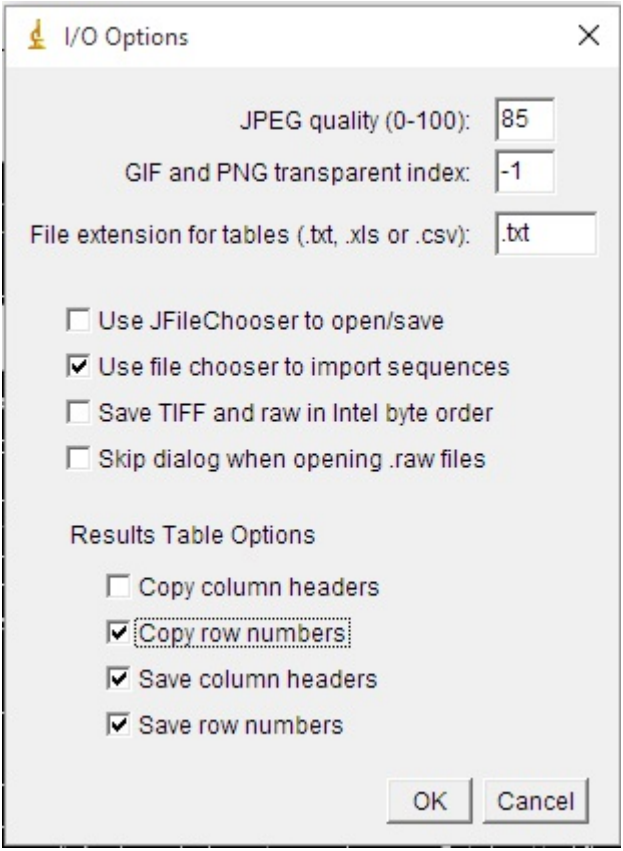


Figure 6. ImageJ I/O options window.

List of useful references

Note: This list of references is not intended to be exhaustive in any way. It simply reflects some books, articles or webpages that I find interesting on the topic. My intention is to expand this list over time. Regarding the ImageJ application, there are many tutorials on the web, see for example here or here

Russ, J.C., 2011. The image processing handbook. CRC Press. Taylor & Francis Group

> This is a general-purpose book on image analysis written by professor John C. Russ from the Material Sciences and Engineering at North Carolina State University. Although the book is not specifically focused on structural geology, thin sections, or even rocks, it covers a wide variety of procedures in image analysis and contain very nice examples of image enhancement, segmentation techniques and shape characterization. I find the text very clear and well-written, so if you are looking for a general-purpose image analysis book, this is the best one I known.

Heilbronner, R., Barret, S., 2014. Image Analysis in Earth Sciences. Springer-Verlag Berlin Heidelberg. doi:10.1007/978-3-642-10343-8

> This book focuses on image analysis related with Earth Sciences putting much emphasis on methods used in structural geology. The first two chapters deals with image processing and grain segmentation techniques using the software Image SXM, which is a different flavour of the ImageJ family applications (see here).

Heilbronner, R., 2000. Automatic grain boundary detection and grain size analysis using polarization micrographs or orientation images. *J. Struct. Geol.* 22, 969–981. doi:10.1016/S0191-8141(00)00014-6

> This paper explains a simple procedure for creating grain boundary maps from thin sections using a semi-automatic method implemented in a NIH Image macro named Lazy Grain Boundary (LGB). NIH Image is the predecessor of ImageJ (no longer under active development). The authors compare the results obtained using the LGB method and manual segmentation using digital images obtained from a quartzite under light microscopy using different techniques. Interestingly, all the steps described in the protocol can be automated using the ImageJ software and using more sophisticated segmentation algorithms than those originally implemented in the LGB macro (e.g. using the Canny edge detector instead of the Sobel one).

Barraud, J., 2006. The use of watershed segmentation and GIS software for textural analysis of thin sections. *Journal of Volcanology and Geothermal Research* 154, 17–33. doi:10.1016/j.jvolgeores.2005.09.017

> This paper propose a workflow for grain segmentation and grain analysis that differs slightly from other approaches referred here. For the acquisition, it uses three different images from light microscopy with different orientations, combining them in a false-colour RGB image. For grain segmentation it uses the anisotropic diffusion for noise reduction plus watershed methods (both available in ImageJ).

Frequently Asked Questions

Who is this script for?
This script is targeted at anyone who wants to: i) visualize grain size features, ii) obtain a set of apparent grain size measures to estimate the magnitude of differential stress (or rate of mechanical work) in dynamically recrystallized rocks, and iii) estimate the actual (3D) distribution of grain sizes from thin sections using stereological methods. The latter point includes an estimate of the volume occupied by a particular grain size fraction and a parameter to measure the shape of the population of grain sizes (assuming that the distribution of grain sizes follows a lognormal distribution). The stereological methods assume that grains have equant or near-equant (AR < 2.0) shapes, which includes grains produced during static annealing and dynamically recrystallized grains when bulging (BLG) or sub-grain rotation (SGR) are the main recrystallization process. For igneous studies involving tabular grains far from near-equant objects, we recommend other approaches such as those implemented in the *CSDCorrections* software (Higgins 2000). See the references list section for details.

Why use apparent grain size measures instead of measures estimated from unfolded 3D grain size distributions in paleopiezometry studies?
One may be tempted to use a stereological method to estimate the midpoint of the modal interval or any other unidimensional parameter based on the actual grain size distribution rather than using the mean, median, or frequency peak of the apparent grain size distribution. We think that there is no advantage in doing this but serious drawbacks. The rationale behind this is that 3D grain size distributions are estimated using a stereological model and, hence, the accuracy of the estimates depends not only in the introduction of errors during measuring but also on the robustness of the model. Unfortunately, stereological methods are built on "weak" geometric assumptions and the results will always be, at best, only approximate. This means that the precision of the estimated 3D size distribution is **much poorer** than the precision of the original distribution of grain profiles since the latter is based on real data. To sum up, use stereological methods only when you need to estimate the volume occupied by a particular grain size fraction or investigating the shape of the actual grain size distribution, otherwise use estimates based on the apparent grain size distribution.

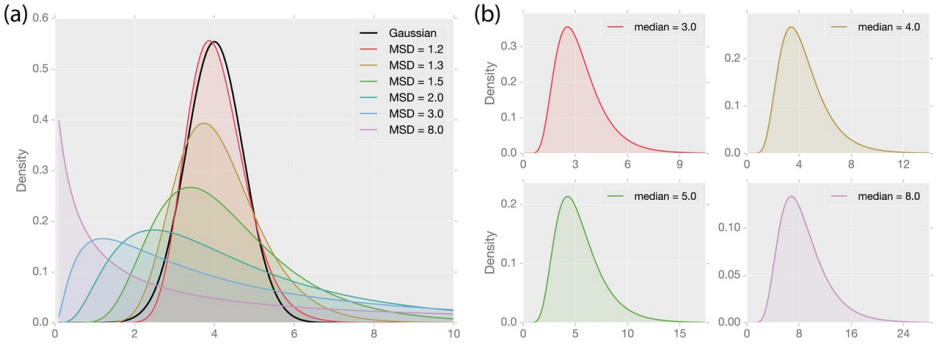
What "average" (mean, median, peak) apparent grain size measure do I use?

This depends on the features of the grain size distribution. In the case of unimodal distributions the following rule of thumb should be considered:

- use **mean and standard deviation (SD)** when your **distribution is normal-like**
- use **median and interquartile (or interprecentil) range** when your **distribution is skewed**

If your distribution is multimodal (two or more frequency peaks), you can use for comparative purposes the **location of the frequency peaks** based on the Kernel density estimate. For more details read the Scope section.

What is an MSD value? What is it used for?
MSD stands for *Multiplicative Standard Deviation* and it is a parameter that allows to define the shape of the grain size distribution using a single value assuming that it follows a lognormal distribution. In plain language, the MSD value gives a measure of the asymmetry (or skewness) of the grain size distribution. For example, an MSD value equal to one corresponds to a normal (Gaussian) distribution and values greater than one with log-normal distributions of different shapes, being the higher the MSD value the greater the asymmetry of the distribution (Figure a). The advantage of this approach is that by using a single parameter we can define the shape of the grain size distribution independently of its scale (Fig. b), which is very convenient for comparing the shape of two or more grain size distributions.



Probability density functions of selected lognormal distributions taken from Lopez-Sanchez and Llana-Fúnez (2016). (a) Lognormal distributions with different MSD values (shapes) and the same median (4). (b) Lognormal distributions with the same shape corresponding to an MSD value (1.5) and different medians (note that different medians imply different scales in the horizontal and vertical directions).

Why the grain size distribution plots produced by the GST script and the classic CSD charts do not use the same units on the y axis?

As you may noticed classic CSDs charts (Marsh, 1988) show in the vertical axis the logarithmic variation in population density or log(frequency) in mm^{-4}, while the stereological methods put in the GrainSizeTools (GST) script returns plots with a linear frequency (per unit volume). This is due to the different aims of the CSDs and the plots returned by the GST script. Originally, CSDs were built for deriving two things in magmatic systems: i) nucleation rates and ii) crystal growth rates. In these systems, small grains are more abundant than the large ones and the increase in quantity is typically exponential. The use of the logarithm in the vertical axis helps to obtain a straight line with the slope being the negative inverse of the crystal growth times the time of crystallization. Further, the intercept of the line at grain size equal to zero allows estimating the nuclei population density. In recrystallized rocks, there is no grain size equal to zero and we usually unknown the crystallization time, so the use of the CSDs are not optimal. Furthermore, the use of the logarithm in the vertical axis has two main disadvantages for microstructural studies: (i) it obscures the reading of the volume of a particular grain fraction, a common target in microtectonic studies, and (ii) it prevents the easy identification of the features of grain size distribution, which is relevant for applying the two-step method.

Why the sum of all frequencies in the histograms is not equal to one?

This is because the script normalized the frequencies of the different classes so that the integral over the range is one. In other words, once the frequencies are normalized to one, the frequency values are divided by the bin size. This means that the sum of all frequency values will not be equal to one unless the bin size is one. We have chosen this normalization method because it allows comparing similar distributions using a different number of classes or bin size, and it is required to properly apply the two-step method.

Is it necessary to specify the version of the script used in a publication? How can this be indicated?

Yes, it is always desirable to indicate the version of the script used. The rationale behind this is that codes may contain bugs and versioning allow to track them and correct the results of already published studies in case a bug is discovered later. The way to indicate the version is twofold:

- The easy way we advise you to follow is by explicitly indicating the version in your manuscript as follows: "*...we used the GrainSizeTools script version 1.4.4...*" and then use the general citation in the reference list.

- The other way is by adding a termination in the form *.v plus a number* in the general DOI link. This is, instead of using the general reference of the script:

  Lopez-Sanchez, Marco A. (2018): GrainSizeTools script. figshare.

  https://doi.org/10.6084/m9.figshare.1383130

  You can modify the doi link as follows:

  Lopez-Sanchez, Marco A. (2017): GrainSizeTools script. figshare.

  https://doi.org/10.6084/m9.figshare.1383130.v15 (Note the termination *.v15*)

  To get the specific doi link, go to the script repository at https://figshare.com/articles/GrainSizeTools_script/1383130 , find the version you used in your study, and then click on "Cite" to obtain the full citation.

Does the script work with Python 2.7.x and 3.x versions? Which version do I choose?
Despite both Python versions are not fully compatible, *GrainSizeTools script* has been written to run on both. As a rule of thumb, if you do not have previous experience with the Python language go with 3.x versions, which is the present and future of the language. Python 2.7.x versions are still maintained for legacy reasons, but keep in mind that their support will be discontinued in 2020. Also note that since version 1.3, the script is only tested by the maintainer using Python 3.

I get the results but not the plots when using the Spyder IDE
This issue is produced because the size of the figures returned by the script are too large to show them inside the console using the **inline** mode. To fix this go to the Spyder menu bar and in `Tools>Preferences>IPython console>Graphics` find *Graphics backend* and select *Automatic*.

Can I report bugs or submit ideas to improve the script?
Definitely. If you have any problem using the script please just let me know (see an email address here: http://marcoalopez.github.io/). Feedback from users is always welcome and important to develop a better script. Lastly, you can also create a fork of the project and develop your own tools based on the GST script since it is open source and free.

References cited

Cross AJ, Prior, DJ, Stipp M and Kidder S (2017) The recrystallized grain size piezometer for quartz: An EBSD-based calibration *Geophys. Res. Lett.* 44, 6667–6674. doi:10.1002/2017GL073836

Exner HE (1972) Analysis of Grain- and Particle-Size Distributions in Metallic Materials. *International Metallurgical Reviews* 17, 25-42.

Heilbronner R and Barret S (2014) Image Analysis in Earth Sciences. Springer-Verlag Berlin Heidelberg. doi:10.1007/978-3-642-10343-8

Higgins MD (2000) Measurement of crystal size distributions. *American Mineralogist* 85, 1105-1116. doi: 10.2138/am-2000-8-901

Holyoke CW and Kronenberg AK (2010) Accurate differential stress measurement using the molten salt cell and solid salt assemblies in the Griggs apparatus with applications to strength, piezometers and rheology. *Tectonophysics* 494(1–2), 17–31, doi:10.1016/j.tecto.2010.08.001

Jung H and Karato S (2001) Effects of water on dynamically recrystallized grain-size of olivine. *Journal of Structural Geology* 23, 1337-1344, doi:10.1016/S0191-8141(01)00005-0

Lopez-Sanchez MA and Llana-Funez S (2015) An evaluation of different measures of dynamically recrystallized grain size for paleopiezometry or paleowattmetry studies. *Solid Earth* 6, 475-495. doi: 10.5194/se-6-475-2015

Lopez-Sanchez MA and Llana-Fúnez S An extension of the Saltykov method to quantify 3D grain size distributions in mylonites. *Journal of Structural Geology* 93, 149-161. doi: 10.1016/j.jsg.2016.10.008

Marquardt DW (1963) An Algorithm for Least-Squares Estimation of Nonlinear Parameters. *J. Soc. Ind. Appl. Math.* 11, 431–441. doi: 10.1137/0111030

Post A and Tullis J (1999) A recrystallized grain size piezometer for experimentally deformed feldspar aggregates. *Tectonophysics* 303, 159-173, doi:10.1016/S0040-1951(98)00260-1

Rutter E (1995) Experimental study of the influence of stress, temperature, and strain on the dynamic recrystallization of Carrara marble. *Journal of Geophysical Research: Solid Earth*, 100, 24651-24663. doi: 10.1029/95JB02500

Sahagian D and Proussevitch AA (1998) 3D particle size distributions from 2D observations: stereology for natural applications. *Journal of Volcanology and Geothermal Research* 84, 173-196. doi: 10.1029/95JB02500

Saltykov SA (1967) The determination of the size distribution of particles in an opaque material from a measurment of the size distribution of their secions. In: Elias, H. (Ed.), *Proceedings of the Second International Congress for STEREOLOGY*. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 163–173. doi: 10.1007/978-3-642-88260-9_31

Silverman BW (1986) Density estimation for statistics and data analysis. Monographs on Statistics and Applied Probability, Chapman and Hall, London.

Shimizu I (2008) Theories and applicability of grain size piezometers: The role of dynamic recrystallization mechanisms. *J. Struct. Geol.* 30(7), 899–917, doi:10.1016/j.jsg.2008.03.004

Stipp M and Tullis J (2003) The recrystallized grain size piezometer for quartz. *Geophysical Research Letters* 30, 1-5. doi: 10.1029/2003GL018444

Twiss RJ (1977) Theory and Applicability of a Recrystallized Grain Size Paleopiezometer. *Pure Appl. Geophys. PAGEOPH* 115(1–2), 227–244, doi:10.1007/BF01637105

Van der Wal D, Chopra M, Drury M and Fitz-Gerald J (1993) Relationships between dynamically recrystallized grain size and deformation conditions in experimentally deformed olivine rocks. *Geophysical Research Letters* 20, 1479-1482, doi:10.1029/93GL01382