GrainSizeTools

A Python script for estimating the grain size from thin sections

Information

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

Manual version:
v3.2.1
Release date:
2017/03/15

Author:
Marco A. Lopez-Sanchez

Licenses:
This document is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.
GrainSizeTools script is licensed under the Apache License, Version 2.0 (the "License").

Table of contents

Requirements

GrainSizeTools script requires Python 2.7.x (legacy) or 3.x versions and the scientific libraries *Numpy*, *Scipy*, *Pandas* and *Matplotlib*. We recommend installing the Continuum Anaconda or the Enthought Canopy (maybe more easy-friendly for novices) distributions. Both distributions are free in their basic versions and provide the most popular Python scientific packages including those named above. Both packages also provide academic free licenses for advanced versions. 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), since they are public-domain image processing programs widely used for scientific research that runs on Windows, OS X, and Linux platforms.

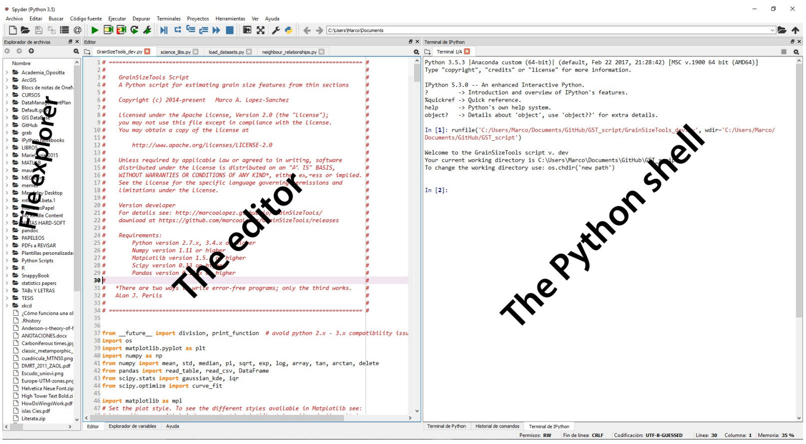
The main aim of this documentation is not to describe how to measure the areas of the grain profiles with the *ImageJ* application but to treat the data obtained from this or similar applications. If you are not familiarized with the use of *ImageJ* you have tutorial within this document.

Getting Started: A step-by-step tutorial

> **Important note:** Please, **update as soon as possible to version 1.3.x**, it contains important changes that are not fully compatible with previous versions. It is also advisable to **update matplotlib to version 2.x** since 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 v1.3.2). 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.



```
# ========================================================================= #
#                                                                           #
#    GrainSizeTools Script                                                  #
#    A Python script for estimating grain size features from thin sections  #
#                                                                           #
#    Copyright (c) 2014-present   Marco A. Lopez-Sanchez                    #
#                                                                           #
#    Licensed under the Apache License, Version 2.0 (the "License");        #
#    you may not use this file except in compliance with the License.       #
#    You may obtain a copy of the License at                                #
#                                                                           #
#        http://www.apache.org/licenses/LICENSE-2.0                         #
#                                                                           #
#    Unless required by applicable law or agreed to in writing, software    #
#    distributed under the License is distributed on an "AS IS" BASIS,      #
#    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND*, either express or implied. #
#    See the License for the specific language governing permissions and    #
#    limitations under the License.                                         #
#                                                                           #
#    Version developer                                                      #
#    For details see: http://marcoalopez.github.io/GrainSizeTools/          #
#    download at https://github.com/marcoalopez/GrainSizeTools/releases     #
#                                                                           #
#    Requirements:                                                          #
#        Python version 2.7.x, 3.4.x or higher                              #
#        Numpy version 1.11 or higher                                       #
#        Matplotlib version 1.5.[illegible] or higher                       #
#        Scipy version 0.13 or higher                                       #
#        Pandas version [illegible] or higher                               #
#                                                                           #
#   *There are two ways to write error-free programs; only the third works. #
#   Alan J. Perlis                                                          #
#                                                                           #
# ========================================================================= #


from __future__ import division, print_function  # avoid python 2.x - 3.x compatibility issu
import os
import matplotlib.pyplot as plt
import numpy as np
from numpy import mean, std, median, pi, sqrt, exp, log, array, tan, arctan, delete
from pandas import read_table, read_csv, DataFrame
from scipy.stats import gaussian_kde, iqr
from scipy.optimize import curve_fit

import matplotlib as mpl
# Set the plot style. To see the different styles available in Matplotlib see:
```

```
Python 3.5.3 |Anaconda custom (64-bit)| (default, Feb 22 2017, 21:28:42) [MSC v.1900 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 5.3.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: runfile('C:/Users/Marco/Documents/GitHub/GST_script/GrainSizeTools_dev.py', wdir='C:/Users/Marco/
Documents/GitHub/GST_script')

Welcome to the GrainSizeTools script v. dev
Your current working directory is C:\Users\Marco\Documents\GitHub\GST_script
To change the working directory use: os.chdir('new path')


In [2]:
```

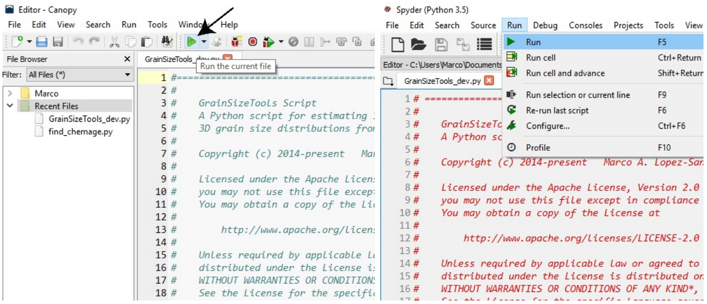
Figure 1. The editor and the Python shell (a.k.a. the console) in the the Spyder (bottom) integrated development environment (IDE). Both, the Enthough Canopy and the Spyder IDEs, are MATLAB-like IDEs optimized for numerical computing and data analysis using Python. They also provide a file explorer, 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).

The following text will appear in the shell/console:

```
Welcome to the GrainSizeTools script v. 1.3.2
Your current working directory is...
To change the working directory use: os.chdir('new path')
```

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.



```
#==========================================
#
#    GrainSizeTools Script
#    A Python script for estimating
#    3D grain size distributions fro
#
#    Copyright (c) 2014-present   Ma
#
#    Licensed under the Apache Licen
#    you may not use this file excep
#    You may obtain a copy of the Li
#
#        http://www.apache.org/licen
#
#    Unless required by applicable l
#    distributed under the License i
#    WITHOUT WARRANTIES OR CONDITION
#    See the License for the specifi
```

```
# =============
#
#    GrainSizeTo
#    A Python sc
#
#    Copyright (c) 2014-present   Marco A. Lopez-Sar
#
#    Licensed under the Apache License, Version 2.0
#    you may not use this file except in compliance
#    You may obtain a copy of the License at
#
#        http://www.apache.org/licenses/LICENSE-2.0
#
#    Unless required by applicable law or agreed to
#    distributed under the License is distributed on
#    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND*,
```

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

A brief note on the organization of the script

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

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

    PARAMETERS
    areas:
    a numpy array with the sectional areas of the grains

    correct_diameter:
    Correct the diameters estimated from the areas of the grains by adding the
    the width of the grain boundaries. If correct_diameter is not declared, it
    is considered 0. A float or integer.
    """

    # 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
```

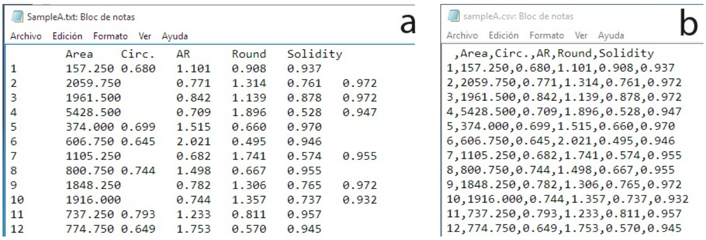
To sum up, the name following the Python keyword `def`, in this example `calc_diameters`, is the name of the function. The sequence of names within the parentheses are the formal parameters of the function, the inputs. In this case the function has two inputs, the parameter `areas` that correspond with 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 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 four, and usually less than four, functions to obtain the results. For more details, you can look at the section *Specifications of main functions in the GrainSizeTools script*.

Using the script to visualize and estimate the grain size features

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 or csv file. If you do not know how to do this, then go to the section How to measure the grain profile areas with ImageJ.


SampleA.txt: Bloc de notas
Archivo Edición Formato Ver Ayuda
a
sampleA.csv: Bloc de notas
Archivo Edición Formato Ver Ayuda
b

```
        Area    Circ.   AR      Round   Solidity
1       157.250 0.680   1.101   0.908   0.937
2       2059.750        0.771   1.314   0.761   0.972
3       1961.500        0.842   1.139   0.878   0.972
4       5428.500        0.709   1.896   0.528   0.947
5       374.000 0.699   1.515   0.660   0.970
6       606.750 0.645   2.021   0.495   0.946
7       1105.250        0.682   1.741   0.574   0.955
8       800.750 0.744   1.498   0.667   0.955
9       1848.250        0.782   1.306   0.765   0.972
10      1916.000        0.744   1.357   0.737   0.932
11      737.250 0.793   1.233   0.811   0.957
12      774.750 0.649   1.753   0.570   0.945
```

```
 ,Area,Circ.,AR,Round,Solidity
1,157.250,0.680,1.101,0.908,0.937
2,2059.750,0.771,1.314,0.761,0.972
3,1961.500,0.842,1.139,0.878,0.972
4,5428.500,0.709,1.896,0.528,0.947
5,374.000,0.699,1.515,0.660,0.970
6,606.750,0.645,2.021,0.495,0.946
7,1105.250,0.682,1.741,0.574,0.955
8,800.750,0.744,1.498,0.667,0.955
9,1848.250,0.782,1.306,0.765,0.972
10,1916.000,0.744,1.357,0.737,0.932
11,737.250,0.793,1.233,0.811,0.957
12,774.750,0.649,1.753,0.570,0.945
```

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. Consequently, we usually obtain a text file with the data in a spreadsheet-like form (Fig. 3). In our 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 previous 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 it and use instead forward slashes (e.g. "C:/yourfilelocation.../nameofthefile.txt") or double backslashes. Also note that you won’t need to specify the file type (txt or csv) as in previous versions of the script.

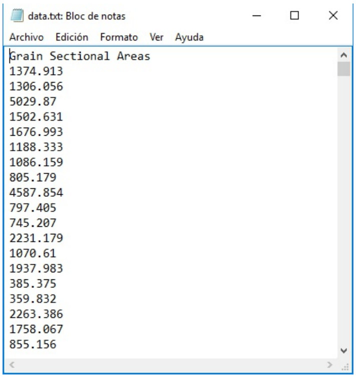
In the case that the user extracted and stored the areas of the grains in a different form from the one proposed here, this is either in a txt or csv file but without a spreadsheet-like form (Fig. 4), there is a Python/Numpy built-in method named `np.genfromtxt()` that can be used to load any text data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

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

or if 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:

```
>>> 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. You can define any number of lines to ignore.



```
Grain Sectional Areas
1374.913
1306.056
5029.87
1502.631
1676.993
1188.333
1086.159
805.179
4587.854
797.405
745.207
2231.179
1070.61
1937.983
385.375
359.832
2263.386
1758.067
855.156
```

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.

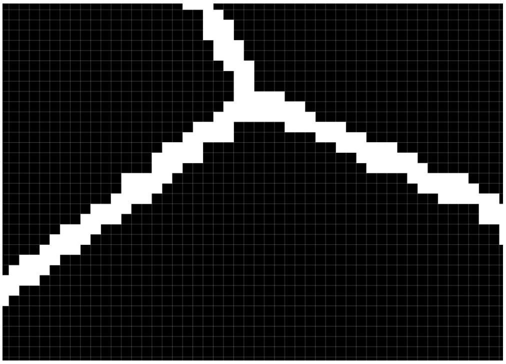


Figure 5. Example of correction of sizes in a grain boundary map. The figure is a raster showing the grain boundaries (in white) between three grains. The squares are the pixels of the image. The boundaries are two pixel wide, approximately. If, for example, each pixel corresponds to 1 micron, we will 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) or an extension of the Saltykov method named the two-step method (Lopez-Sanchez and Llana-Fúnez, 2016).

Obtaining an unidimensional value of grain size (paleopiezo/wattmetry studies)

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
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), widely used in the past, 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 type of plot also implies to obtain different grain size estimations.

Since the version 1.3 of the script, this 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. The other methods available 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. You can also use and *ad hoc* bin/class size (see an example below). We encourage you to use the default method `'auto'`. In addition, the `'doane'` and `'scott'` methods work pretty well in case you have a lognormal- or a normal-like distribution, 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 depending on the type of plot your distribution of apparent grain sizes will change (Fig. 6). Last, an example with a user-defined bin size will be 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).

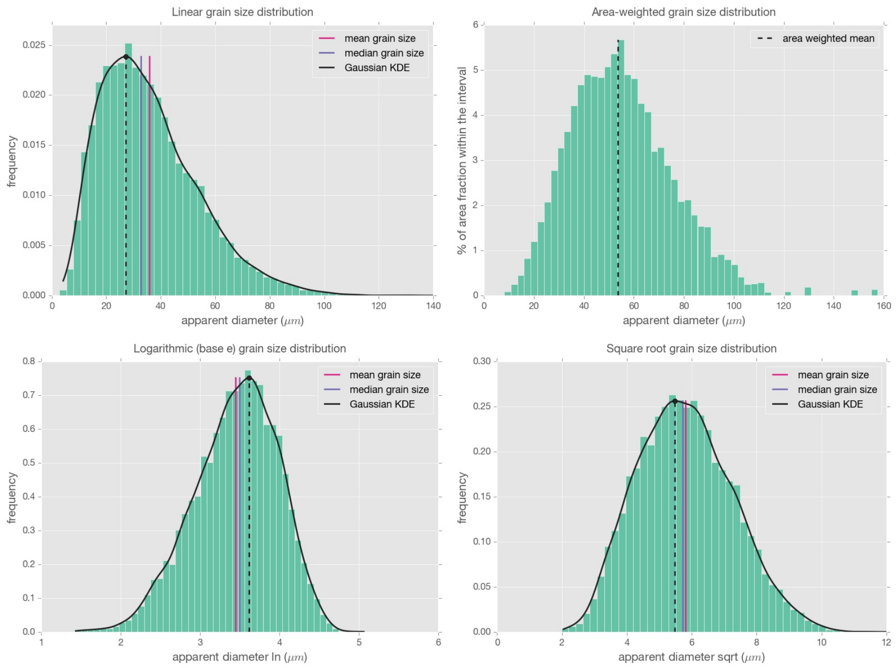


Figure 6. Different apparent grain size vs frequency plots of the same population returned by the

find_grain_size function. These include the number- (linear) and area-weighted plots (upper part) and the logarithmic and square root apparent grain sizes (lower part)

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. It is therefore necessary to check first whether the 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).

Using the Saltykov method

To derive the actual 3D population of grain sizes using the Saltykov method (Saltykov 1967), 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. Lastly, 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.

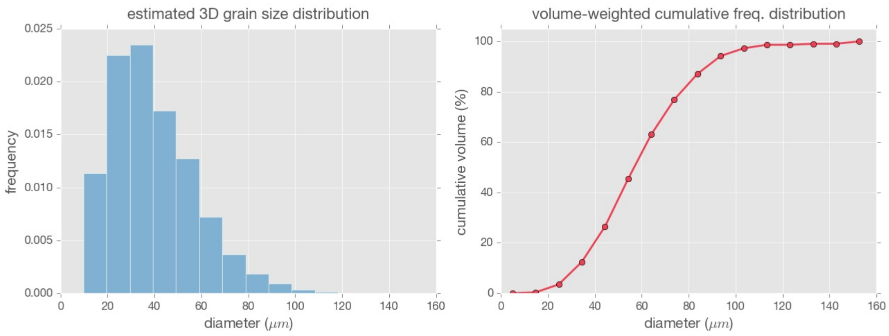


Figure 7. The derived 3D grain size distribution and the volume-weighted cumulative grain size distribution using the Saltykov method.

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 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. 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.

Using 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 script return two parameters, the **MSD** and the theoretical **median**, both enough to fully describe the lognormal distribution at their original 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, when the algorithm fails it would be necessary to change these default values by adding a new parameter as follows:

```
>>> 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).

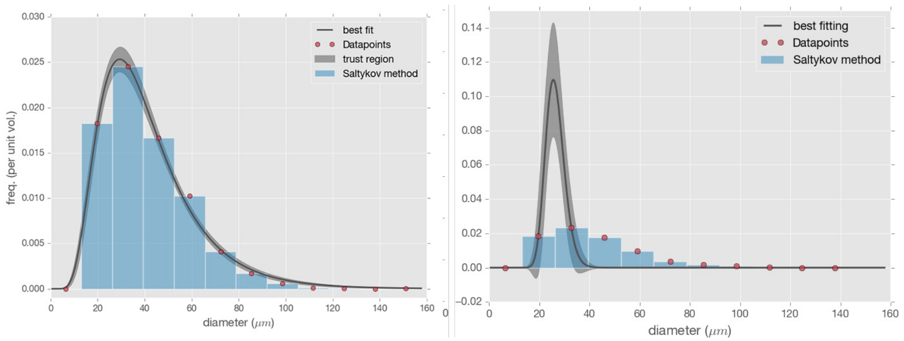


Figure 8. Plots obtained using the two-step method. At left, an example with the lognormal pdf well fitted to the data points. The shadow zone is the trust region for the fitting procedure. At right, an example with a wrong fit due to the use of unsuitable initial guess values. Note the discrepancy between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

Other general methods of interest

Estimate 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 installed). Hence, in general 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, use any other name if desired.
> The commands shown in this tutorial are for the GST script v1.3. 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 an unidimensional value of grain size (paleopiezo/wattmetry studies)

```
# 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)
```

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

```python
# 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 the ImageJ app

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 (for example Fiji, see here for a summary) . As a cautionary note, this is not a detailed tutorial on image analysis using ImageJ at all, 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 in a grain size study is to obtain a grain boundary map from the thin section (Fig. 9).

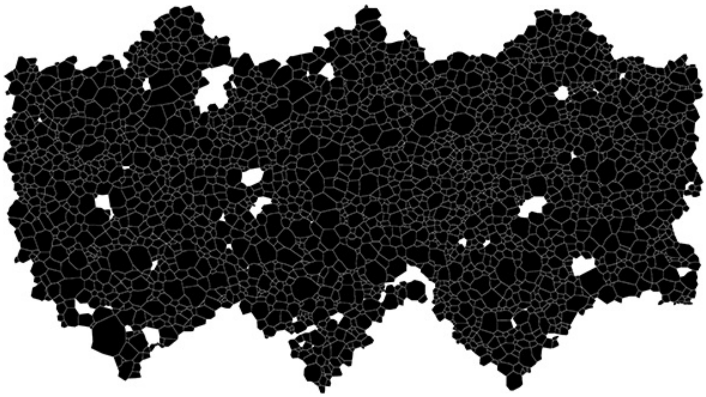


Figure 9. An example of a grain boundary map

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 'gray' value in a range that goes from 0 (white) to 256 (black) (i.e. it allows 256 different gray intensities). In the case of a grain boundary map (Fig. 9), 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 on 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 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 ppi = 4 per unit area). In contrast, 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, color *vs* gray scale, nature of the artefacts, grain size boundary *vs* phase maps, etc.). The presentation of this image analysis 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 book *Image Analysis in Earth Sciences* by Heilbronner and Barret (2014) and Russ (2011) for a more general treatise on the subject. Instead, this tutorial is focused 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 more pixels (Fig. 5). 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.

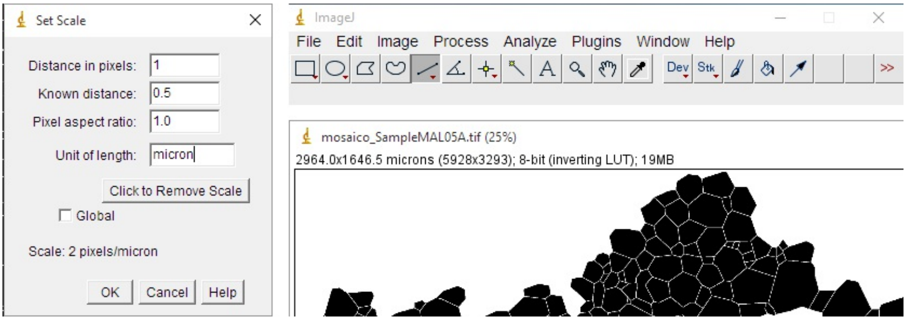
Measuring the areas of the grain profiles

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. 10). 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. 10) 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.


Set Scale

Distance in pixels: 1

Known distance: 0.5

Pixel aspect ratio: 1.0

Unit of length: micron

Click to Remove Scale

☐ Global

Scale: 2 pixels/micron

OK Cancel Help

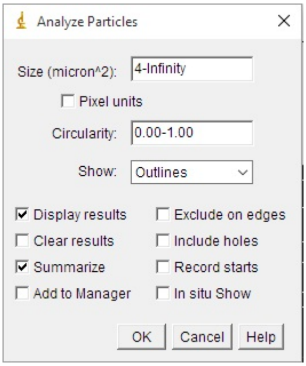
mosaico_SampleMAL05A.tif (25%)

2964.0x1646.5 microns (5928x3293); 8-bit (inverting LUT); 19MB

Figure 10. 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. 11). 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.


Analyze Particles

Size (micron^2): 4-Infinity

☐ Pixel units

Circularity: 0.00-1.00

Show: Outlines

☑ Display results ☐ Exclude on edges

☐ Clear results ☐ Include holes

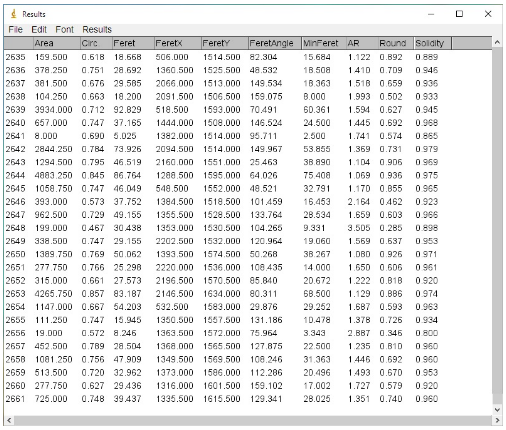
☑ Summarize ☐ Record starts

☐ Add to Manager ☐ In situ Show

OK Cancel Help

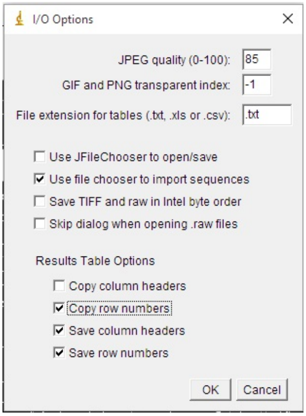
Figure 11. Analyze particles window showing the different options

6. After a while, several windows will appear. At least, one containing the results of the measures (Fig. 12), 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. 13). 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 12. Window with the ImageJ output showing all the measures done on the grains.



JPEG quality (0-100): 85
GIF and PNG transparent index: -1
File extension for tables (.txt, .xls or .csv): .txt

☐ Use JFileChooser to open/save
☑ Use file chooser to import sequences
☐ Save TIFF and raw in Intel byte order
☐ Skip dialog when opening .raw files

Results Table Options

☐ Copy column headers
☑ Copy row numbers
☑ Save column headers
☑ Save row numbers

Figure 13. ImageJ I/O options window.

List of useful references (in alphabetical order)

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

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 in rocks that differs slightly from other approaches referred here. For image acquisition, he uses three different images from light microscopy with different orientations and then combine them in a false-colour RGB image. For grain segmentation it uses the anisotropic diffusion for noise reduction plus watershed segmentation (both available in the ImageJ-type apps). For grain size measures he promotes the use of geographical information systems (GIS) and vector graphics over the use of ImageJ-type applications plus raster images.

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 and is no longer under active development. The authors compare the results obtained using the LGB method and manual segmentation. The input digital images were obtained from a quartzite under light microscopy using different techniques including the CIP method. All the steps describing in the protocol can be automated using the ImageJ software and even using more sophisticated segmentation algorithms than those implemented in the LGB macro (e.g. using the Canny instead of the Sobel edge detector)

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). This book is a must if you are interested in the topic.

Herwegh, M., 2000. A new technique to automatically quantify microstructures of fine-grained carbonate mylonites: two-step etching combined with SEM imaging and image analysis. J. Struct. Geol. 22, 391-400. doi:10.1016/S0191-8141(99)00165-0

> This paper uses digital backscatter electron images on previously chemically attacked samples to distinguish between two mineral phases. The author uses the NIH Image and the LGB method (referred above) for the grain segmentation.

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 or shapes characterization among others. I find the text very clear and well-written, so if you want a general-purpose image analysis book, this is the best one I known.

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 single 1D measures of grain size to estimate the magnitude of differential stress (or rate of mechanical work) in dynamically recrystallized rocks, and/or iii) estimate the actual 3D distribution of grain sizes from the population of apparent grain sizes measured in thin sections. These features include the estimation of the volume occupied by a particular grain size fraction and the estimation of the parameters that best describe the population of grain sizes (assuming that the distribution of grain sizes follows a lognormal distribution). The stereological methods implemented within the script assume that the grains have shapes not far from near-equant. This seems acceptable most of the time for common dynamically recrystallized non-tabular grains in crustal and mantle shear zones such as olivine, quartz, feldspar, calcite or ice when bulging (BLG) or sub-grain rotation (SGR) are the main recrystallization processes. For studies involving tabular objects far from near-equant objects, we recommend other approaches such as those implemented in the *CSDCorrections* (Higgins 2000). See the references list section for details.

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 version, which is the present and future of the language. Python 2.7.x versions are still maintained for legacy reasons and they are widely used, but keep in mind that their support will be discontinued in 2020

Why the grain size distribution plots produced by the GST script do not use the same units as the classic CSD charts?
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 x crystallization time. Further, the intercept of the line at grain size = 0 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 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.

Why use apparent grain size measures instead of measures estimated from the unfolded 3D grain size distribution in paleopiezometry studies?
At first glance, one may be tempted to use the midpoint of the modal interval or any other 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 these but serious drawbacks. First, 3D grain size distributions are estimated using a stereological model. Thus, the estimation of the modal interval (or the mean or median) depends on the robustness of the model. Unfortunately, stereological methods are built on several (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 distributions, otherwise use always measures based on the apparent grain size distribution.

Can I report bugs or submit ideas to improve the script?
Definitely. If you have any problem using the script please 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

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

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

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.