

Application for absolute permeability analysis based on digital rock sample

Trgovec-Greif, Martin

Master's thesis / Diplomski rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Mining, Geology and Petroleum Engineering / Sveučilište u Zagrebu, Rudarsko-geološko-naftni fakultet**

Permanent link / Trajna poveznica: <https://urn.nsk.hr/urn:nbn:hr:169:239632>

Rights / Prava: [In copyright](#) / [Zaštićeno autorskim pravom.](#)

Download date / Datum preuzimanja: **2024-04-19**



Repository / Repozitorij:

[Faculty of Mining, Geology and Petroleum Engineering Repository, University of Zagreb](#)



UNIVERSITY OF ZAGREB

FACULTY OF MINING GEOLOGY AND PETROLEUM ENGINEERING

Master study of Petroleum Engineering

**APPLICATION FOR ABSOLUTE PERMEABILITY ANALYSIS BASED ON
DIGITAL ROCK SAMPLE**

Master thesis

Martin Trgovec-Greif

N346

Zagreb, 2021.

University of Zagreb

Master Thesis

Faculty of Mining, Geology and Petroleum Engineering

APPLICATION FOR ABSOLUTE PERMEABILITY ANALYSIS BASED ON DIGITAL
ROCK SAMPLE

Martin Trgovec-Greif

Thesis completed in: University of Zagreb

Faculty of Mining, Geology and Petroleum Engineering

Department of Petroleum and Gas Engineering and Energy

Pierottijeva 6, 10000 Zagreb

Abstract

A new algorithm for fast absolute permeability calculation was developed and implemented into computed code. The algorithm uses binarized digital micro-CT rock sample images as input data and estimates sample permeability by analysing pore geometry.

Key words: Digital core physics, digital core, micro-CT image analysis, fast permeability calculation

Thesis consists of: 46 pages, 13 Figures, 3 Tables, 29 references, appendix

Thesis archived: Library of Faculty of Mining, Geology and Petroleum Engineering
Pierottijeva 6, 10000 Zagreb

Supervisor: PhD. Domagoj Vulin, associate professor

Co-supervisor: Marko Gaćina, mag. ing. petrol.

Reviewers: 1. PhD. Domagoj Vulin, associate professor
2. PhD. Vladislav Brkić, associate professor
3. PhD. Borivoje Pašić, associate professor

Date of defense: 16th July 2021., Faculty of Mining, Geology and Petroleum Engineering,
University of Zagreb

Sveučilište u Zagrebu
rad

Diplomski

Rudarsko-geološko-naftni fakultet

APLIKACIJA ZA ANALIZU PROPUSNOSTI NA TEMELJU DIGITALNOG UZORKA
STIJENE

Martin Trgovec-Greif

Diplomski rad izrađen: Sveučilište u Zagrebu

Rudarsko-geološko-naftni fakultet

Zavod za naftno i plinsko inženjerstvo i energetiku

Pierottijeva 6, 10000 Zagreb

Sažetak

Razvijen je, i implementiran u računalni kod, novi algoritam za brzi proračun apsolutne poroznosti. Ulazni podatci uključuju digitalne binarizirane mikro-tomografske slike uzorka stijene. Geometrijskom analizom mreže pornog sustava, izračunata je apsolutna propusnost uzorka.

Ključne riječi: digitalna jezgra, digitalna fizika stijena, analiza mikro-tomografskih slika, brzi proračun propusnosti.

Završni rad sadrži: 46 stranica, 13 slika, 3 tablice, 29 referenci i dodatak

Završni rad pohranjen: Knjižnica Rudarsko-geološko-naftnog fakulteta

Pierottijeva 6, 10000 Zagreb

Voditelj: Dr. sc. Domagoj Vulin, izvanredni profesor RGNF-a

Pomoć pri izradi: Marko Gaćina, mag. naft. rud.

Ocjenjivači: 1. Dr. sc. Domagoj Vulin, izvanredni profesor RGNF-a
2. Dr. sc. Vladislav Brkić, izvanredni profesor RGNF-a
3. Dr. sc. Borivoje Pašić, izvanredni profesor RGNF-a

Datum obrane: 16. srpnja 2021., Rudarsko-geološko-naftni fakultet, Sveučilište u Zagrebu

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude towards my supervisor prof. PhD Domagoj Vulin and co-supervisor Marko Gaćina, mag. ing. petrol. for the time they invested into me and my professional development, for helping me discover my professional interests, for their non-stop help during my studies and creation of this thesis, which made all of this possible.

TABLE OF CONTENTS

I.	LIST OF FIGURES	I
II.	LIST OF TABLES	I
III.	LIST OF ABBREVIATIONS AND SYMBOLS	II
1.	INTRODUCTION.....	1
2.	METHODOLOGY.....	3
2.1.	Image processing and segmentation	3
2.1.1.	<i>Image cropping.....</i>	<i>3</i>
2.1.2.	<i>Contrast enhancing.....</i>	<i>4</i>
2.1.3.	<i>Image denoising.....</i>	<i>4</i>
2.1.4.	<i>Image segmentation.....</i>	<i>6</i>
2.2.	Pore space characterization.....	8
2.2.1.	<i>Determining the representative elementary volume (REV)</i>	<i>8</i>
2.2.2.	<i>Pore space morphology</i>	<i>9</i>
2.2.3.	<i>Pore area calculation in two-dimensional images.....</i>	<i>11</i>
2.2.4.	<i>Pore perimeter calculation from two dimensional images</i>	<i>11</i>
2.2.5.	<i>Pore radius calculation.....</i>	<i>15</i>
2.3.	Absolute permeability calculation.....	15
3.	DEVELOPMENT OF APPLICATION FOR FAST PERMEABILITY CALCULATION	18
3.1.	Backend.....	18
3.2.	Frontend (Grafical User Interface)	20
4.	RESULTS AND DISCUSSION	22
5.	CONCLUSION.....	28

6. REFERENCES.....	30
7. APPENDIX – PYTHON CODE	33
7.1. binarization.py	33
7.2. permeability.py	35

I. LIST OF FIGURES

Figure 2-1. Morphological erosion and dilation; (a – original image, b – dilated image, c – eroded image)	5
Figure 2-2. Illustration of watershed algorithm: catchment basins (left) and two pores separated by ridgeline (right)	10
Figure 2-3. Influence of pore orientation on calculated perimeter	11
Figure 2-4. Pore perimeter by subtracting eroded image. Original image (left), eroded image (middle), isolated pore edges (right)	12
Figure 2-5. Convex hull (right) and original image (left)	13
Figure 2-6. Voxel categorization mechanism and assigned weights	14
Figure 3-1. Application folder structure, green boxes represent folders and red boxes represent files.....	18
Figure 3-2. 'settings' pane of the GUI	21
Figure 4-1. Unprocessed and uncropped sample slice.....	22
Figure 4-2. Cropped subsample used for the analysis	23
Figure 4-3. Mineral glaring effect.....	24
Figure 4-4. Manually binarized sample (left) vs. automatically binarized sample (right).....	24
Figure 4-5. Autocorrelation function for the subsample	25

II. LIST OF TABLES

Table 4-1. Permeability from pore area	25
Table 4-2. Permeability from pore perimeter	26
Table 4-3. Calculated permeabilities for samples previously analysed by CFD methods.....	27

III. LIST OF ABBREVIATIONS AND SYMBOLS

A	sample cross-section area	m^2
A_{pores}	area of pores	m^2
cl	circularity coefficient	-
D	Euclidian distance	m
D_{eq}	equivalent pore diameter	m
GMR	geometric mean radius	m
HR	hydraulic radius	m
I	voxel intensity	-
k	absolute permeability	mD
L	sample length	m
Q	volumetric flow rate	m^3/s
r	pore radius	m
S_2^j	autocorrelation function value	-
V_{pore}	volume of equivalent capillary tube	m^3
W_{Ti}	number of voxels within Ti region divided by total number of voxels	-
γ	gamma value	-
Δp	pressure drop	Pa
Φ	porosity	-
μ	fluid viscosity	$mPa \cdot s$
μ_{Ti}	mean voxel intensity within Ti region	-
σ_B^2	between-the-class variance	-

1. INTRODUCTION

Because of limitations of traditional experimental core analysis methods, digital core analysis has rapidly become a valuable, commercially used tool for calculations of rock properties (Blunt et al., 2013; Fredrich et al., 2014). Furthermore, the value in digital core analysis is not only in predicting petrophysical rock properties, but it also gives insight into phenomena occurring at pore-scale. The main advantages of digital core analysis, in comparison to traditional laboratory experiments are:

- less time required for sample preparation and analysis,
- lower costs for analysis,
- possibility of conducting multiple numerical experiments on the same sample,
- possibility of conducting multiple numerical experiments simultaneously, on the same sample,
- possibility of examining multiple production strategies with the goal of maximizing fluid recovery,
- quality and integrity of the sample remain unchanged by conducted experiments, while some laboratory experiments can cause sample degradation or disintegration.

In the last fifteen or so years, micro-focused X-ray computed tomography (micro-CT) is used to obtain digital rock samples for analysis of pore space structure and residual fluids in porous medium. In the standard workflow, rock samples are scanned with micro-CT. The scan represents spatial distribution of sample density which enables the distinguishment of pore space and rock grains. The final product of the scanning is three-dimensional image (stack of two-dimensional images) that represents digital reconstruction of inner sample structures (Mukunoki et al., 2016). This digital representation of a rock sample, if correctly prepared for further analyses, is called a *digital core*, the term that whole branch of research derives its name from, and that is *digital core analysis*.

For the purpose of digital core analysis, numerical and computer methods are used for determining rock properties like pore size distribution, absolute and effective porosity as well as for some of the more complicated laboratory experiment simulations like mercury intrusion

capillary pressure analysis (MICP) and single or multiphase flow simulation for determining permeability.

Digital core analysis workflow consists of image preparation, segmentation and binarization, defining sub-volume of the sample on which the analyses and simulations are to be performed and, if numerical simulations are performed, the necessary step is choosing the numerical solver method and defining boundary conditions. Computing power and runtime required for the analysis is dependent on the sample size, therefore, the choice of representative elementary volume (REV) is of critical importance (Blunt et al., 2013; Andrä et al., 2013a). Since there is no standard workflow that could offer precise data for broad range of porosities and permeabilities with acceptable runtime, the majority of challenges is connected to image processing for pore space reconstruction. Even though there are technological possibilities for sample scanning on nanoscale (using scanning electron microscope, SEM), these images are inappropriate for numerical simulations and pore-scale sample analysis, because they either require very high computing power or do not fulfil the condition that the sample is representative. Very high image resolution implies that the digital sample is very small, often smaller than the determined REV and, despite the fact that these images show more detail and are more similar to real rock sample, using these images for rock properties estimation can lead to results that show no value in practical application. To illustrate, while modern reservoir scale models contain about $5 \cdot 10^5$ to $5 \cdot 10^6$ cells, CT sample with dimension of 1x1x1 cm with voxel resolution of 1 μm contains 10^{12} voxels.

The idea of this thesis is to determine absolute permeability of a rock sample by analysing geometrical properties of interconnected pores. Usually, rock permeability is determined by running numerical flow simulations (finite-element, finite-volume, Lattice-Boltzmann method) (Neumann et al., 2021). While these methods may produce more precise results, they are much more demanding in terms of computer power and runtime. The proposed approach is suitable for fast calculations of absolute permeability, absolute and effective porosity while also showing permeability data at pore-scale, which enables detecting restrictions in flow paths.

2. METHODOLOGY

Standard digital rock physics workflow consists of:

1. image processing and analysis for pore and grain segmentation
2. computational grid generation
3. running numerical simulations for effective rock properties calculation

2.1. Image processing and segmentation

Image processing is usually performed in four steps: (1) cropping, (2) contrast enhancing, (3) denoising, (4) segmentation.

After micro-CT scanning, images are processed by contrast enhancing, gamma correction, brightness adjustment and/or histogram equalization (Verri et al., 2017). At the present moment, there are several proposed techniques for automatic image processing and segmentation (Schlüter et al., 2014; Iassonov et al., 2009; Eibenberg et al., 2008; Buades et al., 2005; Huang and Chau, 2008), but porosity estimation for samples where significant fraction of pore space is not detectable by CT scanning (pores that smaller than voxel resolution) is still imprecise. Manual image thresholding usually targets the compromise value between the need for conservation of pore geometry and low resolution-caused detail loss. With that in mind, sample characterization is subject to high degree of uncertainty; comparative studies show that porosity and permeability estimates vary by 30% for porosity and by 40% and more for permeability (Andrä et al., 2013a, b).

2.1.1. *Image cropping*

Since micro-CT obtained images contain not only the rock sample, but also black background, these images need to be cropped before continuing with the analysis. These images also contain artefacts on sample edges which need to be removed by cropping. The cropped images are used for determining REV.

2.1.2. Contrast enhancing

There are several techniques for contrast enhancing of micro-CT images, some of which are histogram equalization, gamma correction and brightness adjustment. Histogram equalization is performed when voxel intensities (which range from 0-255 for 8-bit images) are located in a narrow range of intensities. This is characterized by steep slope in cumulative density of voxel intensities. Histogram equalization will distribute voxel intensities across the whole intensity range. Micro-CT images of rock samples contain a lot of voxels in dark-grey colour range, but some minerals may luminesce under influence of X-rays. This phenomenon causes very light grey or even white voxels to occur in the images. If this occurs, it is better to use gamma correction than histogram equalization for contrast enhancing. Gamma correction for 8-bit images is defined by following expression:

$$I' = 255 \times \left(\frac{I}{255}\right)^{\gamma} \quad (2-1)$$

where:

- I' is voxel intensity after applying gamma correction,
- I is original voxel intensity,
- γ is gamma correction value.

Gamma corrected image is more suitable for automatic image thresholding if there are very bright voxels occurring. Brightness adjustment increments every voxel intensity by a constant, which translates the intensity histogram towards extreme values (0 if the constant is below 0, 255 if the constant is greater than 0). This can cause accumulation of voxels in the extreme value histogram bins. Although contrast enhancing causes loss of structural detail (Schlüter et al., 2014), it is still a necessary step for increasing the quality of image segmentation, especially if a global threshold is chosen (Iassonov et al., 2009).

2.1.3. Image denoising

Image denoising is performed to improve signal to noise ratio. Techniques for denoising usually include some sort of averaging between voxels, but that is not always the case. Denoising also diminishes image details, by blurring edges of objects and removing small objects on the image. In order to avoid significant loss of detail while still improving

segmentation possibilities, edge-preserving filters have been developed (Eibenberger et al., 2008). Among these filters are *Non-Local Means* (NLM) filter and *Bilateral filter* (BF). The main difference between the two is that the latter performs averaging around the target voxel, while the former performs the averaging based on every voxel in the image. For that reason, NLM is expected to be more robust than BF while processing images with low signal to noise ratio (Buades et al, 2005). Three dimensional versions of these filters are also available, where the averaging algorithm considers three consecutive two dimensional cross sections in three main directions. Other filters using for image denoising are median filter and mean filter. Median filter sets the targeted voxel's intensity to the value of median of voxel intensities within the specified radius, while mean filter sets the intensity to the value of mean of voxel intensities within the specified radius. The main advantage of median filter, in comparison to mean filter, is its lower sensitivity to extreme values (Mandzhieva, 2017).

Another approach to image denoising is using morphological operations such as dilation and erosion. Both of these operations require a structuring element, often called a kernel. Kernel represents a pattern by which image objects are modified. In case of dilation, image objects are dilated by moving the structuring element along the edges of the object and changing voxel values of voxels that are not part of the object, but overlap with structuring element, to the value of targeted voxel within the object. Image erosion is an inverse procedure where object edges are eroded by modifying their values to the values of neighbouring voxels outside the object. Figure 2-1 illustrates these two operations.

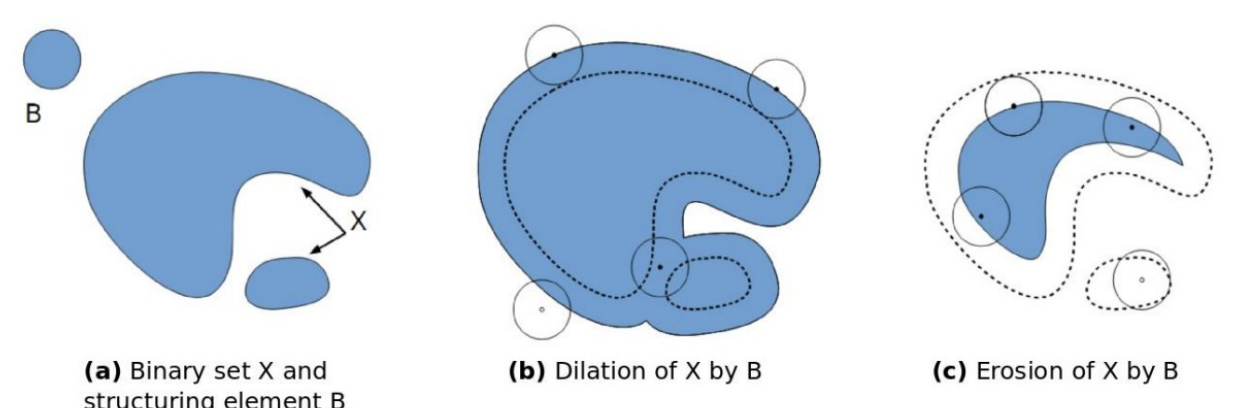


Figure 2-1. Morphological erosion and dilation; (a – original image, b – dilated image, c – eroded image) (<https://imagej.net/plugins/morpholibj>)

These operations are often used in pair. If erosion is performed first, followed by dilation, the operation is called morphological opening. On the contrary, if dilation is performed first, followed by erosion, the operation is called morphological closing. Coupling erosion and dilation gives a powerful tool for removing speckles from images while preserving object shapes and dimensions.

2.1.4. Image segmentation

Image segmentation is the final step of image processing, and the main goal of segmentation is identifying pore space of the sample. This is accomplished by image thresholding. Images that underwent the process of thresholding are binary, meaning that there are only two voxel values present, minimal and maximal values (0 and 255 for 8-bit images). Therefore, this process is also called image binarization. After the threshold value is set, every voxel intensity in the image below that value is set to 0, and every voxel with intensity greater than that value is set to 255. There are several approaches to image thresholding. The simplest and most widely used approach being setting a global threshold value. This means that the whole image will be binarized using the same threshold value. Threshold value can be determined manually or automatically. Manual thresholding usually results in more precise image segmentation. General recommendation for manual thresholding is to pick a threshold value near the first local minimum value on the intensity histogram. Automatic thresholding implies using some algorithm for determining the threshold value. Most commonly used algorithm for this purpose is Otsu's binarization algorithm. Otsu's method segments the image into two regions, T0 and T1, where region T0 is a set of intensities from 0 to t and region T1 from t to 255. Value t is the threshold value. Otsu's method scans all the possible thresholding values and searches for the value with maximum *between-the-class variance* which is defined by the following expression:

$$\sigma_B^2 = W_{T0}W_{T1}(\mu_{T0} - \mu_{T1}) \quad (2-2)$$

where:

- σ_B^2 is *between-the-class variance*,
- W_{T0} is number of voxels in region T0 divided by total number of voxels,

- W_{T1} is number of voxels in region T1 divided by total number of voxels,
- μ_{T0} is mean intensity of region T0 and
- μ_{T1} is mean intensity of region T1.

Otsu's thresholding method is more suitable for images with more complex, multimodal intensity histograms, while it could produce unsatisfying results for simpler, bimodal histogram images (Otsu, 1979; Yousefi, 2015).

Global threshold can be applied to the whole image stack, or the threshold value can be adjusted for each thin section of the three-dimensional image. If the latter method is used, setting the threshold manually for every image slice could be very time consuming and ineffective, therefore it is recommended to use some sort of automatic thresholding, but it is also important to consider that some image slices could still have significant artefacts which could impede binarization quality even if the image has already undergone denoising process.

Adaptive thresholding is a thresholding method in which a separate threshold value is calculated for each voxel. This technique is suitable for images that have significant variations in illumination. Threshold values are determined by statistical data, like mean or median, of neighbouring voxels within the specified radius. If the value of targeted voxel is lower than the average/median by t percent of neighbouring voxels, where t is an arbitrary number, the voxel value is set to black (0), otherwise it is set to white (255) (Bradley and Roth, 2007).

The quality of segmentation is heavily limited by image resolution. Voxels in the area of microporosity (pores undetectable by micro-CT) are grey and are easily misclassified as pores if global thresholding technique is applied. These voxels are usually the ones that make the ideal boundary between pore space and rock grains. It is also possible that, if microporosity is present, there are two valleys in low intensity area of intensity histogram. In that case, it is recommended that the global threshold value is set near the second valley (Verri et al., 2017). If microporosity is not classified as pore space, not only will it result in wrong porosity estimation, but it will also lead to wrong pore connectivity interpretation. This can result in significant misestimation of average pore radius and average flow velocity (Leu et al., 2014), but it may result in better absolute permeability estimation if geometrical analysis is performed, because micropores do not contribute much to flow and represent restrictions in pore network.

2.2. Pore space characterization

2.2.1. Determining the representative elementary volume (REV)

In order to reduce the required computing power for creation of a digital model and running simulations and other calculations, it is necessary to choose the representative elementary volume of the sample. REV is defined as the smallest subsample for which it is justified to consider its geometrical characterization representative for the whole sample. REV is determined by two-point correlation function, also called autocorrelation. Autocorrelation function for statistically inhomogeneous system is defined as follows:

$$S_2^j(r_1, r_2) = \langle I^j(r_1) I^j(r_2) \rangle \quad (2-3)$$

where:

- S_2^j is the autocorrelation function value,
- r_1 and r_2 are two arbitrary points in the system,
- angle brackets represent average value and
- $I^j(r)$ is indicator function defined as:

$$I^j(r) = \begin{cases} 1, & \text{if } r \text{ is } j \text{ phase} \\ 0, & \text{otherwise} \end{cases} \quad (2-4)$$

The value of autocorrelation function can be interpreted as probability that both points r_1 and r_2 are in the phase j . For statistically isotropic medium, autocorrelation function value is dependant solely on distance between the points r_1 and r_2 . Therefore, the function can be formulated as:

$$S_2^j = \phi_j \text{ and } \lim_{r \rightarrow \infty} S_2^j(r) = \phi_j^2 \quad (2-5)$$

where:

- ϕ_j is volume fraction of phase j .

The value of autocorrelation function can be easily evaluated by successively translating the line of r voxels by one voxel at a time and spanning the whole image, counting the number of successes of the two end points falling in phase j and dividing the number of successes by total number of trials (which is also the system size). Trials are performed across a fixed cross section, usually along the orthogonal directions. This process is repeated for different distances between

the two arbitrary points r_1 and r_2 . Autocorrelation function values are then plotted against line r length. After certain length, the values of autocorrelation function will start to oscillate inside a relatively narrow range of values. This length is considered the dimension of the representative subsample. If the same process was repeated for the subsample, different REV dimensions would be obtained, however, this difference is confirmed to be negligible for estimation of static properties. On the contrary, for dynamic properties like permeability, larger subsample could offer better pore interconnectivity data (Yeoung and Torquato, 1997).

2.2.2. Pore space morphology

Pore network models are used to describe complex pore structures with simple geometrical shapes representing pore bodies and throats. Throats are defined as narrow passages that connect pores together.

One of the methods for pore network modelling and extraction is maximal ball (MB) method (also called maximum inscribed sphere algorithm) first developed by Silin and Patzek (2006). This approach relies on the description of the pore space via clusters of inscribed spheres and has been proven to be effective in pore networks extraction from tomographic images (Dong and Blunt, 2009). The first step is to explore the entire domain to find the range of all possible values of radii of the inscribed spheres. Since every sphere is centred within the cell and is in contact with the cell wall, the range is limited by minimum and maximum distances between the cell-centres and the nearest wall. Starting from the largest possible sphere, smaller and smaller spheres are built, and if two of them touch, they are merged and clustered. At the end, the whole pore space is filled, and it is possible to obtain average pore diameter by analysing the sphere radii distribution. The average pore radius is calculated as a volume-weighted average, so that the relative contribution of each diameter is calculated as a volume fraction of occupation of the sphere. This value is then compared to theoretical estimation of the equivalent diameter:

$$D_{eq} = \frac{4V_{pore}}{S} \quad (2-6)$$

assuming that the pore space is reduced to an equivalent capillary tube of volume V_{pore} and surface area S (Verri et al., 2017).

Another algorithm for pore network modelling and extraction is watershed algorithm. In case of digital core analysis, at this point, the sample is already binarized. In order to apply watershed segmentation algorithm, it is necessary to find a distance map. This is possible by applying distance transform algorithm (Mandzhieva, 2017). Euclidian distance for each voxel (x_i, y_i) inside the object is calculated from the nearest zero voxel. Euclidian distance is defined as:

$$D = \sqrt{(x_{n-z} - x_i)^2 + (y_{n-z} - y_i)^2} \quad (2-7)$$

where x_{n-z} and y_{n-z} are coordinates of the nearest zero-value voxel.

Since pores are represented as zero voxels (black), and grains are represented as maximum value voxels (white), it is necessary to invert the image before applying distance transform. After distance map is obtained, watershed algorithm for morphological segmentation can be applied. To illustrate how the algorithm works, two objects with local deepest points in their centres should be considered. These deeper areas are called catchment basins. If water would start to come up in these basins, the first contact line, before the water from the two basins would connect, is called watershed ridgeline. This line is considered a throat between two separate pore bodies. The algorithm is illustrated in Figure 2-2.

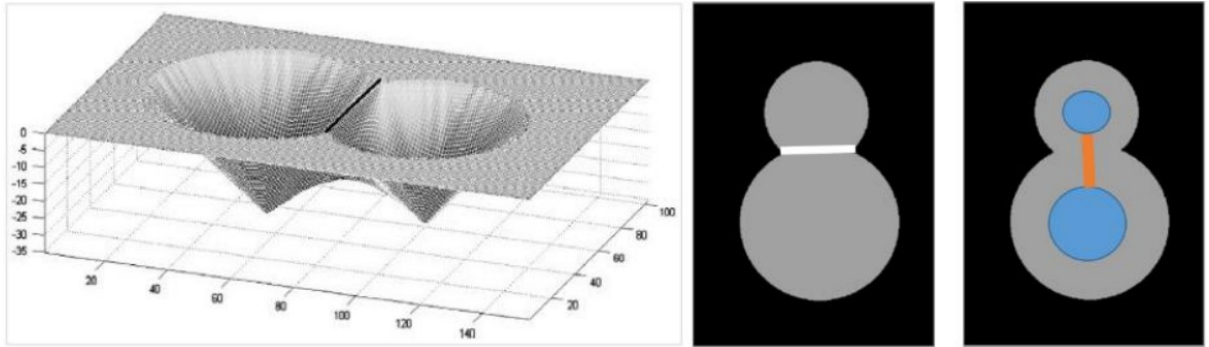


Figure 2-2. Illustration of watershed algorithm: catchment basins (left) and two pores separated by ridgeline (right) (Rabbani et al., 2014)

Medial axis extraction algorithm can also be used for extracting the pore network. This algorithm extracts flow paths using skeletonization. Skeletonization is a process of successively eroding the three-dimensional image object until only one voxel of the object per slice is left. What is then left is network of pore medial axes.

2.2.3. Pore area calculation in two-dimensional images

Pore area from binarized thin sections of the rock sample can be easily obtained. Assuming that pore space in the binarized image is represented by voxels with value 1, pore area is equal to the sum of all voxel intensities within that pore multiplied by squared voxel resolution.

2.2.4. Pore perimeter calculation from two dimensional images

Unlike pore area, calculating pore perimeter is much more complex. The value of pore perimeter varies heavily, depending on the chosen method for perimeter calculation. Perimeter values also depend on image resolution as images with higher voxel resolution show more morphological detail, and therefore detect more roughness along the pore edges which results in greater calculated perimeters. This phenomenon is more pronounced in samples with greater number of smaller pores (samples with lower permeability). Pore orientation within the sample also plays a role in calculated perimeter, for example if perimeter is calculated simply by counting pore edges, pore the size of two pixels that are arranged vertically result in smaller perimeter than if the two pixels were aligned diagonally at 45° angle. Figure 2-3 (left) shows that in the first case, calculated perimeter equals to 6, and in the second case (right), calculated perimeter equals to 8 (Images are treated as binary matrices, with values of 1 representing pore space, and values of 0 representing rock grain).

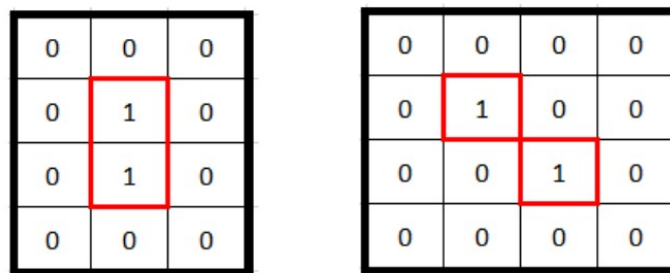


Figure 2-3. Influence of pore orientation on calculated perimeter

It is also questionable if pore elements on the sample edges should be considered when calculating perimeter as the pore shape outside the analysed subsample sample is not known. With all that in mind, twenty-two different methods for determining pore perimeter were considered (some algorithms are developed internally at the faculty) for the purpose of this

thesis. These methods were implemented into a *Python* script for permeability estimation and are described in the follow-up:

1. Pore edges are isolated by comparing the matrix without the first column to the matrix without the last column. Perimeter is then incremented by 1 for each corresponding voxel pair with different values. The process is repeated for the matrices without the first and the last row. Finally, pore voxels on the sample edges are added to the total perimeter. The problem of this method is that it overestimates perimeters, especially for irregular pore shapes.
2. Pore perimeter is calculated similarly to the first method apart from adding pore voxels on the sample edges to the total perimeter. Perimeter overestimation is still present in this method.
3. Since pore orientation plays a role in calculated perimeter, this method calculates the pore perimeter as in method number one and repeats the process for the matrix rotated by 45°. Perimeter is then calculated as the average value between the two.
4. Perimeter is calculated as the average value of perimeter calculated by method number 2 and perimeter of the matrix rotated by 45° calculated by the same method.
5. Perimeter is calculated similarly to the method number 3, but minimal value between the two is chosen as the perimeter, instead of calculating average value.
6. Perimeter is calculated similarly to the method number 4, but minimal value between the two is chosen as the perimeter, instead of calculating average value.
7. Pore edges are isolated by subtracting the matrix obtained by binary erosion of the original image from the original image (Figure 2-4). Perimeter then equals to the sum of remaining voxels of value 1.

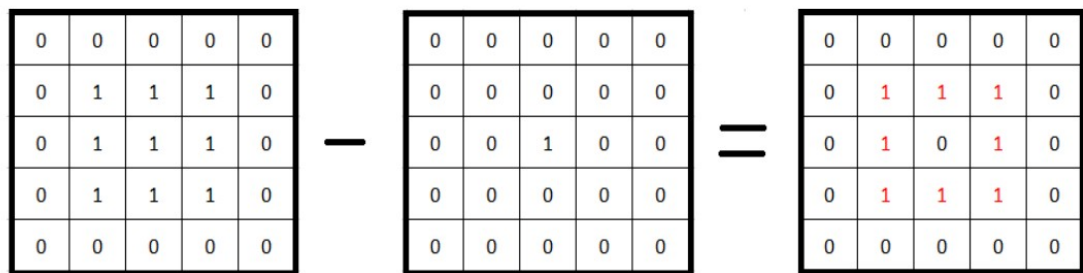


Figure 2-4. Pore perimeter by subtracting eroded image. Original image (left), eroded image (middle), isolated pore edges (right)

8. Perimeter is calculated as the average value between the perimeter obtained by method number 7 and perimeter from the matrix rotated by 45° obtained by the same method.
9. Pore edges are isolated by subtracting the original matrix from the matrix obtained by binary dilation of the original image. Perimeter is then equal to sum of remaining voxels of value 1.
10. Perimeter is calculated as the average value between the perimeter obtained by method number 9 and perimeter from the matrix rotated by 45° obtained by the same method.
11. Perimeter is calculated similarly as in method number 8, but minimal value between the two is chosen, instead of calculating the average value.
12. Perimeter is calculated similarly as in method number 9, but minimal value between the two is chosen, instead of calculating the average value.
13. Perimeter is calculated as in method number 1, but it is corrected by solidity coefficient. Solidity coefficient describes degree of convexity of an object and is defined as a ratio between object area and area of object's convex hull. Convex hull of an object is the smallest circumscribed convex polygon surrounding the targeted region (as shown in Figure 2-5) (<https://scikit-image.org/>).

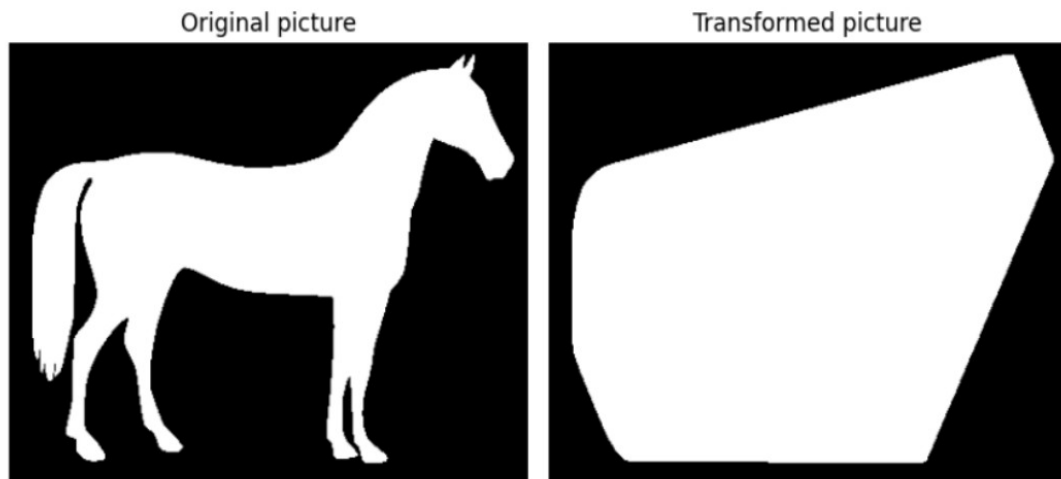


Figure 2-5. Convex hull (right) and original image (left) (<https://scikit-image.org/>)

14. Perimeter is calculated as in method number 2, but corrected by solidity factor.
15. Perimeter is obtained by function *bwperim* from *Python* module *Mahotas*. Input data for the function is a binary matrix (the image) and number of connections, which represents the number or directions in which the nearest voxels are considered elements of target voxel's neighbourhood (4 if only orthogonal axes are considered or 8 if all 8

directions are considered). Every voxel that has a zero-value voxel in its neighbourhood is then considered an element of object perimeter and is added to total perimeter (<https://mahotas.readthedocs.io/en/latest/labeled.html>).

16. Perimeter is calculated as average value of perimeters obtained by methods 7 and 9.

17. Perimeter is obtained by function *measure.perimeter* from *Python* module *scikit-image*.

First step in perimeter calculation via this function is object edge detection. This is achieved by subtracting the matrix obtained by performing binary erosion on the original image from the original image matrix. Remaining voxels are then categorized in three categories based on directions in which they are connected to neighbouring voxels of the object edges. For each category a weight is assigned, which represents voxel's individual contribution to the total perimeter. Categorization mechanism is shown in Figure 2-6 (Benkrid et al., 2000). Total perimeter is then equal to sum of voxel in each category multiplied by corresponding voxel weight.

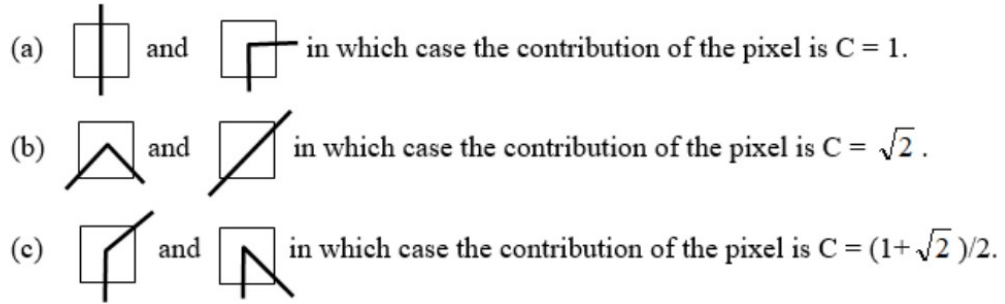


Figure 2-6. Voxel categorization mechanism and assigned weights (Benkrid et al., 2000)

18. Perimeter is calculated by method number 17 and corrected by solidity coefficient.

19. Perimeter is calculated by method number 15 and corrected by solidity coefficient.

20. Perimeter is calculated by method number 16 and corrected by solidity coefficient.

21. Perimeter is calculated by method number 9 and corrected by circularity coefficient.

Circularity coefficient is defined as follows:

$$cl = \frac{1}{4\pi} \cdot \frac{(pore\ area)}{(pore\ perimeter)^2} \quad (2-8)$$

and represents degree of similarity of object shape to a circle. Value of 1 would mean the object is completely circular in shape.

22. Perimeter is calculated by method number 1 and corrected by circularity coefficient.

Perimeters acquired by described methods are dimensionless (they only represent the number of voxels that make up object perimeter) and it is necessary to multiply them by voxel resolution.

2.2.5. Pore radius calculation

When pore perimeters and areas are known, pore radii are calculated using the expression for radius of a circle. If pore perimeter was calculated, the pore radius is calculated as follows:

$$r = \frac{(\text{pore perimeter})}{2\pi} \quad (2-9)$$

Similarly, if pore area was calculated, the pore radius is calculated as follows:

$$r = \sqrt{\frac{(\text{pore area})}{\pi}} \quad (2-10)$$

When calculating pore radius from area, the value of pore area can be corrected by solidity or circularity factors beforehand. This will result in smaller pore radii.

Pore radius can also be calculated as hydraulic radius which is defined by following expression:

$$HR = \frac{2(\text{pore area})}{(\text{pore perimeter})} \quad (2-11)$$

The purpose of hydraulic radius is to take pore shape into consideration when calculating pore radius as pore cross-sections are obviously not circular and it is questionable if using the formula for circle area/perimeter is valid when calculating pore radius. Similarly, another type of radius estimation can be used when calculating pore radius, and that is geometric mean radius, which is defined as:

$$GMR = \sqrt{HR \cdot \sqrt{\frac{(\text{pore area})}{\pi}}} \quad (2-12)$$

2.3. Absolute permeability calculation

A new algorithm for sample's average permeability estimation based on digital tomographic images of rock samples was developed and is described in this thesis. The algorithm analyses each individual flow path throughout the sample and detects restrictions

along these paths. As these restrictions are what determines sample's permeability, it was thought that analysing them at pore-scale level would give valuable insight into sample's hydraulic properties. No numerical flow simulations were performed while determining permeability, as it was determined by exclusively analysing geometrical properties of pores.

After radius of each individual pore that contributes to fluid flow (interconnected pores) has been determined via pore perimeter and/or area, pairs of two thin sections of three-dimensional binary image are used to calculate permeability between them. If the same pore exists in both slices, a pair of permeabilities is calculated using Hagen-Poiseuille's law for circular cross-sections and Darcy's law for linear flow. Hagen-Poiseuille's law for circular cross-sections is defined as:

$$Q = \frac{\Delta p A_{pores}^2}{8\pi\mu L} \quad (2-13)$$

where:

- Q is fluid flow rate through the pore, m^3/s ,
- A_{pore} is area of pores, assuming circular cross-section can be expressed as r^2/π , m^2 ,
- Δp is pressure drop along the conduit, Pa,
- μ is fluid viscosity and, $mPa \cdot s$,
- L is conduit length, m

Darcy's law for linear flow through porous media is defined as:

$$Q = \frac{kA\Delta p}{\mu L} \quad (2-14)$$

where:

- A is area of sample cross-section and
- k is absolute permeability.

Area of sample cross-section can also be expressed as:

$$A = \frac{A_{pores}}{\phi} \quad (2-15)$$

where Φ is sample porosity. By rearranging these equations (2-13, 2-14 and 2-15), it is possible to express absolute permeability as:

$$k = \frac{r^4 \pi}{8A} \quad (2-16)$$

After permeability of a pore is calculated using equation 2-16 for both slices of a pair, average pore permeability for the slice pair is calculated using the expression for permeability for flow through series beds:

$$\bar{k} = \frac{L}{\sum_{i=1}^n \frac{L_i}{k_i}} \quad (2-17)$$

where:

- $\bar{k}$ is average permeability,
- L is sample length,
- n is a number of series for which average permeability is calculated,
- k_i is current slice's permeability and
- L_i is current slices length.

This process is repeated for each pore in slice pair. After every pore's average permeability in slice pair is known, the average permeability for the whole slice is calculated by expression for permeability for flow through layered (parallel) beds:

$$\bar{k} = \frac{\sum_{i=1}^n k_i h_i}{h} \quad (2-18)$$

where:

- h_i is slice thickness,
- h is total sample height.

This whole process is repeated for every slice pair. The number of obtained average permeabilities at this point is equal to one half of number of slices. Since this method is not applicable to samples with odd number of slices, for those cases last slice is duplicated. The last step is to calculate total average sample permeability using the expression for permeability for flow through series beds once again.

3. DEVELOPMENT OF APPLICATION FOR FAST PERMEABILITY CALCULATION

The algorithm described in section 2.3 was implemented into computer code. *Python* programming language runs the backend of the application, while frontend is handled by *EEL* module (<https://github.com/ChrisKnott/Eel>) for creating *Electron*-like, html-based applications. Application folder structure and scheme is shown in Figure 3-1.

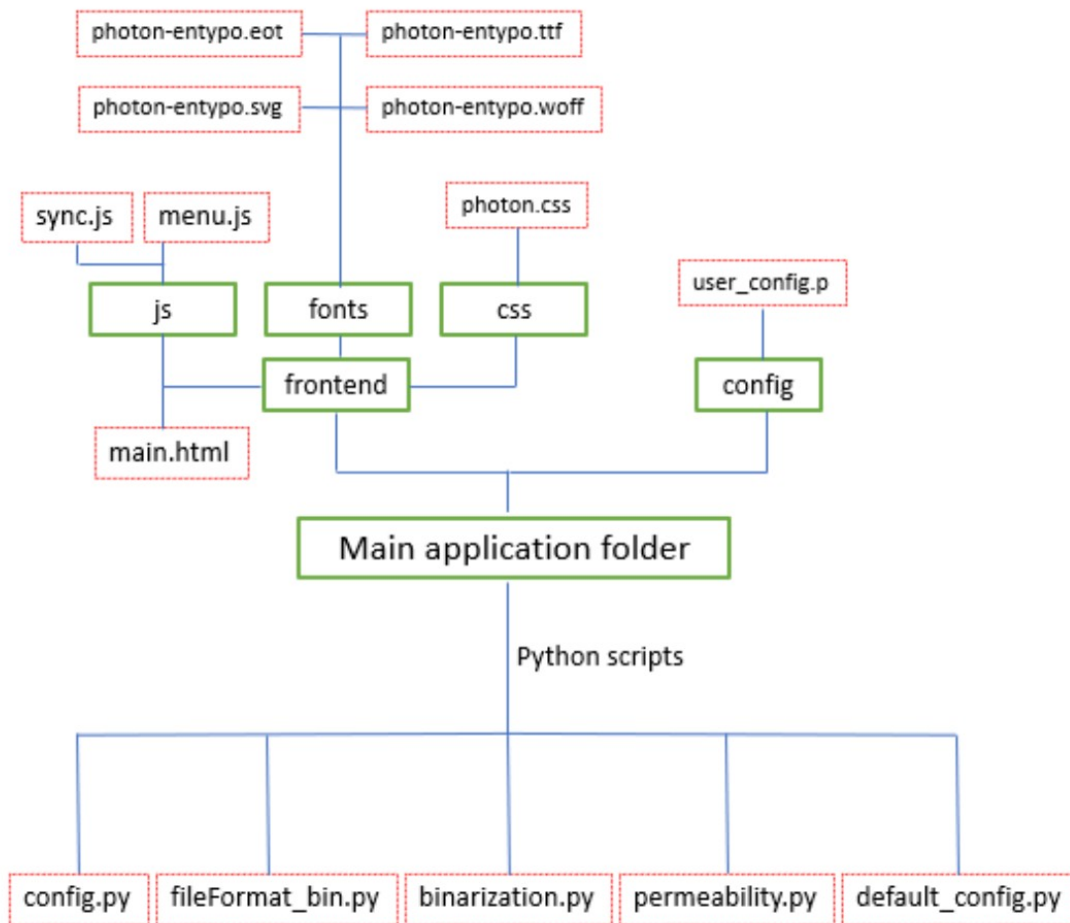


Figure 3-1. Application folder structure, green boxes represent folders and red boxes represent files

3.1. Backend

As mentioned before, the application backend (which can be considered as the *engine* of the application) is handled by *Python* programming language. Backend consists of several scripts, each one handling another task. *Config.py* script's main purpose is communication with application's frontend and running other scripts. If user did not specify certain input parameters,

it will load default values of these parameters which are specified in *default_config.py* script. It also saves last used input parameters in a file inside config directory called *user_config.p* and loads them when application is next started which is accomplished by using *pickle* module. That way, every time the application is opened, the input parameters will automatically be set to last used values.

Binarization.py (appendix 7.1) script is used if samples that are to be analysed are not yet binary, but in grayscale. The script first loads the image stack in from user specified path. The script gives several image processing options before thresholding. User can choose to denoise images using median filter and/or non-local means filter or enhance contrast by applying gamma correction. It is also possible to apply morphological operations like morphological opening and closing. Image is finally run through thresholding process. Global threshold value is determined by calculating the Otsu's threshold value for each two-dimensional image slice and then minimal value is selected as the threshold value for the whole stack of images. Images are saved as text files in the user specified folder. Absolute porosity is also calculated after thresholding, as a ratio between the number of zero voxels and total number of voxels. The script uses *numpy* module (Harris et al., 2020) to represent images as three-dimensional matrices. *OpenCV* and *scikit-image* modules are used for image pre-processing and binarization.

Several other utility scripts are used for applying in-house made and imported modules into uniform format for further digital rock sample analysis. *FileFormat_bin.py* script uses *numpy* module to convert text images (binary images saved as text files) to compressed numpy format (*npz*). This code also processes images before converting and prepares them for permeability analysis. When images are binarized, output images contain only two voxel values, 0 and 255 (if used with 8-bit images). More appropriate for digital rock analysis is to replace values of 255 to values of 1. At this point, pore space is represented by zeros, and rock grains are represented as ones. This script can also invert the image (switch zeros and ones), if necessary.

The purpose of *permeability.py* script is determining the absolute permeability of the sample. This script heavily relies on *numpy* and *scipy* modules. The script first loads the image as a three-dimensional matrix (*numpy* array) of user-specified dimensions from compressed *npz* files created by *fileFormat_bin.py* script. If the number of slices is odd, the last slice of the image

is duplicated, as the number of slices needs be even for the script to successfully calculate permeability between pairs of neighbouring slices. Then, pore connectivity is checked, and all isolated pores are removed from the image. After pore connectivity, semantic analysis is performed to give each pore a unique value. If that unique value is present in both first and the last slice, this pore is accepted as the pore which can contribute the fluid flow from the first to the last slice. All the other pores are removed by setting their voxel value to zero. At this point, effective porosity of the sample is calculated as a ratio of number of non-zero voxels and total number of voxels. The next step is to determine pore 2D areas. Pairs of slices are analysed, and for each pore that is present in both slices an array of areas is created. That way it is possible to track the changes in 2D area for each individual pore. This is again achieved by labelling the pores with unique voxel values. Pore radii are then calculated as described in section 2.2.5. As there are three different radii values when calculated from pore area (non-corrected, corrected by solidity factor and corrected by circularity factor), three different permeability values are calculated from pore areas (as described in section 2.3). Pore perimeters are then calculated using user chosen perimeter calculation methods. The same principle is valid for perimeter calculation, pairs of slices are analysed, and variation of every pore's perimeter is tracked from slice to slice. At this point it is possible to calculate pore radii from pore perimeters and also hydraulic and geometric mean pore radii. That means that the number of permeabilities that are now calculated equals to a number of chosen methods multiplied by three. The results are then organized into a data frame, using *pandas* (McKinney, 2010) module for *Python*. This allows for the results to be saved in the *Microsoft Excel* document at the user specified location on the computer. The results spreadsheet is automatically opened when the script finishes running.

3.2. Frontend (Grafical User Interface)

The application uses *EEL* module to connect the calculation algorithms (i.e. backend) with graphical user interface (GUI) as a kind of module based on *Electron* software framework (ChrisKnott/Eel, 2021). *EEL* module allows to program the visual interactive part (frontend GUI) as *HTML/JavaScript* page and to connect GUI and the backend engine in both directions. Many modern applications are built with use of similar frameworks, namely MS Teams, Twitch, Visual Studio Code, WhatsApp etc. *EEL* hosts a local webserver, then lets the user annotate functions in *Python* so that they can be called from *JavaScript* and vice versa (ChrisKnott/Eel,

2021]). Most of the frontend-backed communication is happening between config.py script and sync.js which is a *JavaScript* file. Input parameters defined by user in the GUI are passed by sync.js to *Python*. After buttons in the GUI are pressed, sync.js triggers the execution of the corresponding *Python* script. Asynchronous functions inside this script are also defined and the role of these functions is to await feedback from *Python* scripts. If the script executed correctly, a success message is being passed to sync.js, which finally transcribes this message into the GUI.

The GUI is divided into panes. The first pane, named ‘actions’, contains buttons which trigger the execution of the corresponding *Python* scripts (Figure 3-2). The ‘settings’ pane queries the user about input parameters for calculations as well as file paths to data upon which calculations are carried out.

property	value	description
input directory	Enter path	slices stored as 2D ASCII arrays of 0 and 1
output directory	Enter path	slices stored in binary numpy format

dimensions of file that is going to be converted

rx px ny px nz px

dimensions for the analysis (cropped sample)

rx px ny px nz px

pixel size micrometers

File format info

file extension file prefix

number of digits voxel separator default: space

start from slice no. Replace value 255 for 1 ☐ Replace colors

☐ Invert images (default: pore=1, grain=0)

Permeability calculation config

Pore perimeter calculation type: enter comma separated list of numbers 0<=n<=21

☐ Calculate permeability from pore area

Results directory: C:/Users/results/

Automatic binarization config

Grayscale image path: enter path

Denoising filters to apply ☐ Non-local means filter ☐ Median filter

Figure 3-2. 'settings' pane of the GUI

Another *JavaScript* file, menu.js, allows the user to toggle between different panes.

GUI’s visual style is designed by using *Photon CSS (Cascading Style Sheets)* library. *Photon* library is specifically designed for building *Electron*-style applications (<http://photonkit.com/>).

4. RESULTS AND DISCUSSION

Berea sandstone sample was used for the permeability analysis for the purpose of this thesis. This particular sample was analysed by (5) Neumann et al. (2021). Micro-CT 8-bit grayscale images of the sample were obtained with voxel resolution of $2.25\text{ }\mu\text{m/voxel}$. Uncropped and unprocessed sample slice is shown in Figure 4-1. Cylindrical plug sample (height = 38 mm; radius = 19 mm) was characterized experimentally with porosity equalling 18.96% and permeability equalling 121 mD (Neumann et al., 2021).

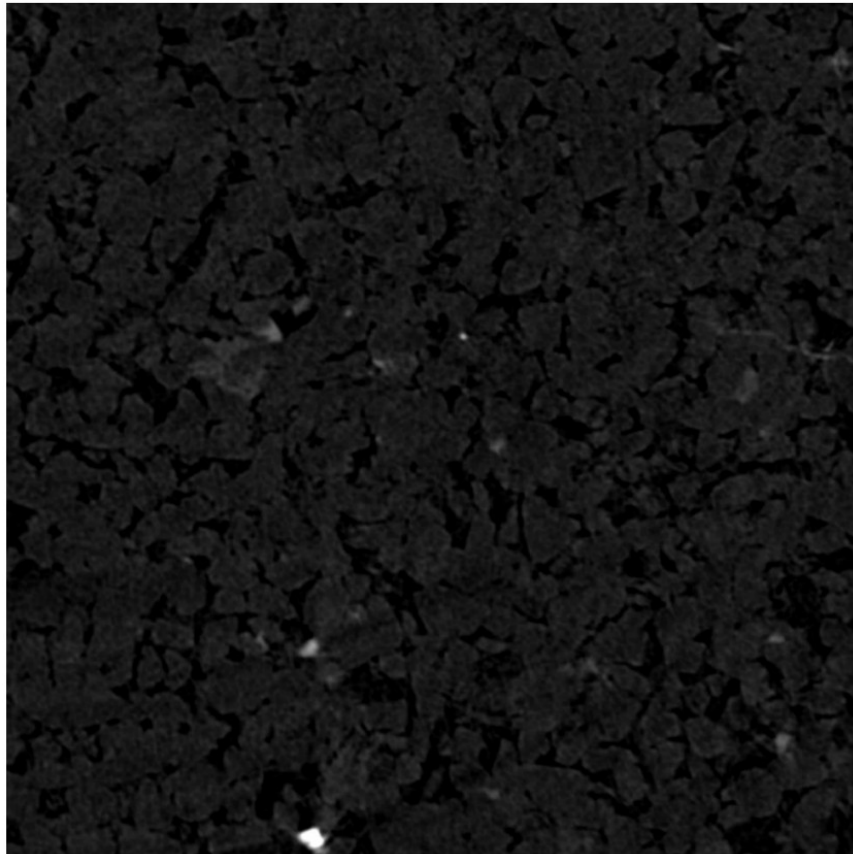


Figure 4-1. Unprocessed and uncropped sample slice

The subsample of the plug (height = 30 mm; radius = 5 mm) was scanned using high-resolution 3D X-ray microtomography. Each slice was 4904×3280 voxels in size. For the purpose of this thesis, the sample was cropped to $500 \times 500 \times 500$ voxel 3D subsample for further processing and analysis (as shown in Figure 4-2).

The next step was to set a global threshold for the image stack in order to binarize it. Two methods for this were applied and compared: manual and automatic thresholding. Manal

threshold was simply applied using *ImageJ* software for scientific image processing and analysis (Schindelin et al., 2012). Automatic thresholding was achieved using the *Python* script described in section 3-1. As significant artefacts were present in the image stack, more specifically, due to minerals glaring, white or very bright regions occurred (Figure 4-3). Passing these images directly to the automatic thresholding algorithm would lead to highly imprecise result with almost no value. Image processing was therefore a necessary step, before a threshold was applied.

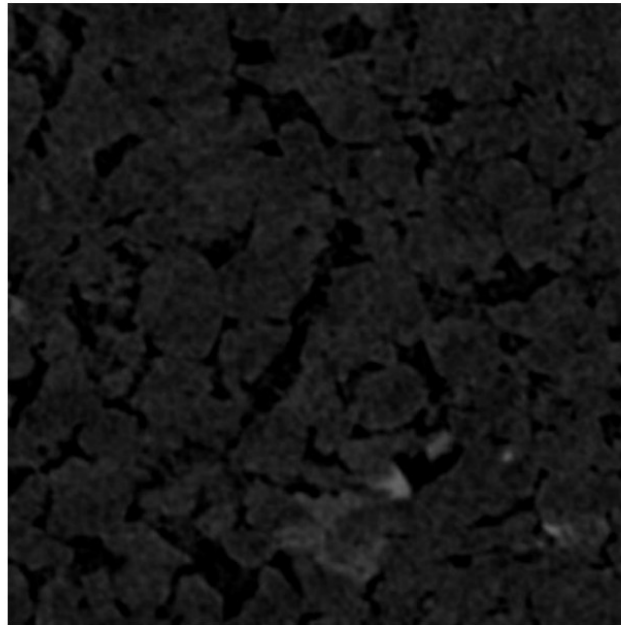


Figure 4-2. Cropped subsample used for the analysis

Extremely bright regions, in mostly dark grey images, would shift the automatically calculated threshold value towards higher values, which would result in significantly overestimated porosity. To overcome this effect, gamma corrected image was used for threshold calculation. Gamma correction with value of gamma equal to 1.5 was applied. That way, shades of grey near each extreme value remained almost untouched by the operation, while in-between values were distributed along the spectre. A threshold value for each slice was calculated using Otsu's method. As a list of threshold values was obtained, the minimal value from the list was chosen as a global threshold value. It is important to mention that it was not the gamma corrected image that was binarized, but the original image. Calculated threshold value was 13, meaning every voxel with intensity below that was set to zero-intensity, while all other voxels were set

to a maximum intensity of 255. Comparison of the first slice binarized manually and automatically is shown in Figure 4-3. Absolute porosity in both cases is 19.59 %.

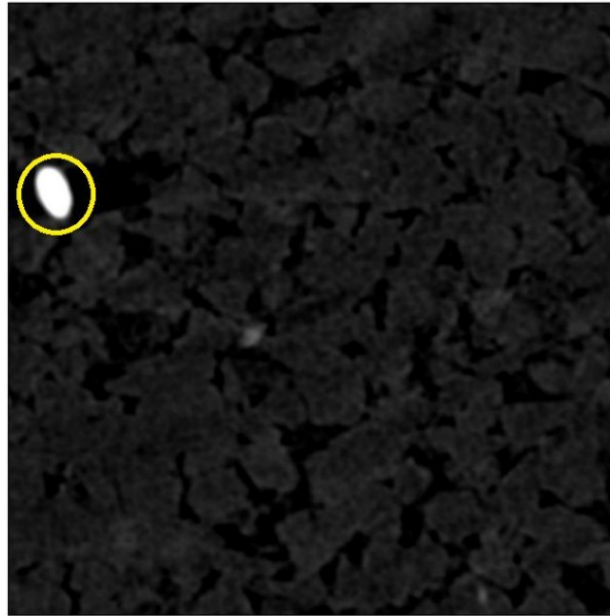


Figure 4-3. Mineral glaring effect

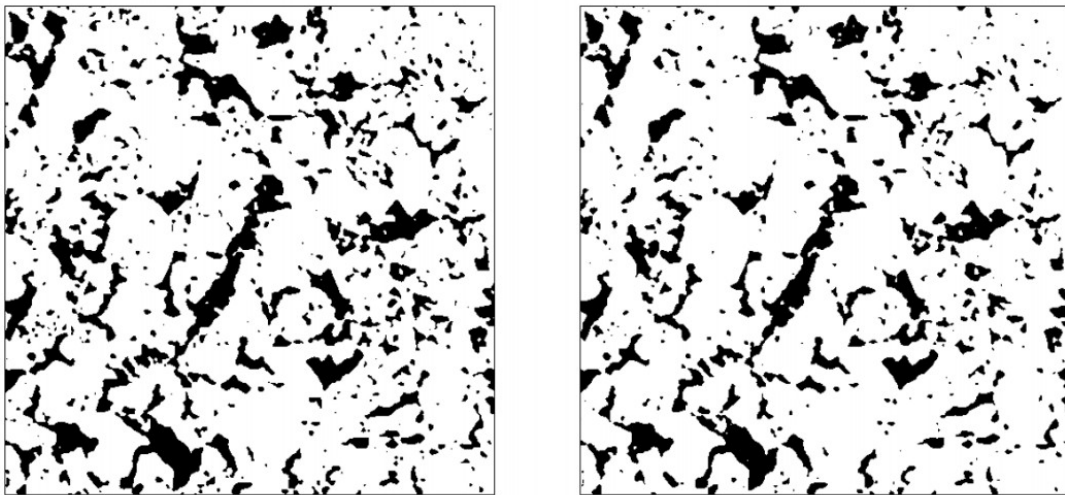


Figure 4-4. Manually binarized sample (left) vs. automatically binarized sample (right)

Autocorrelation function was calculated for the sample in order to check if the chosen subsample is larger than REV. The results are shown in Figure 4-5.

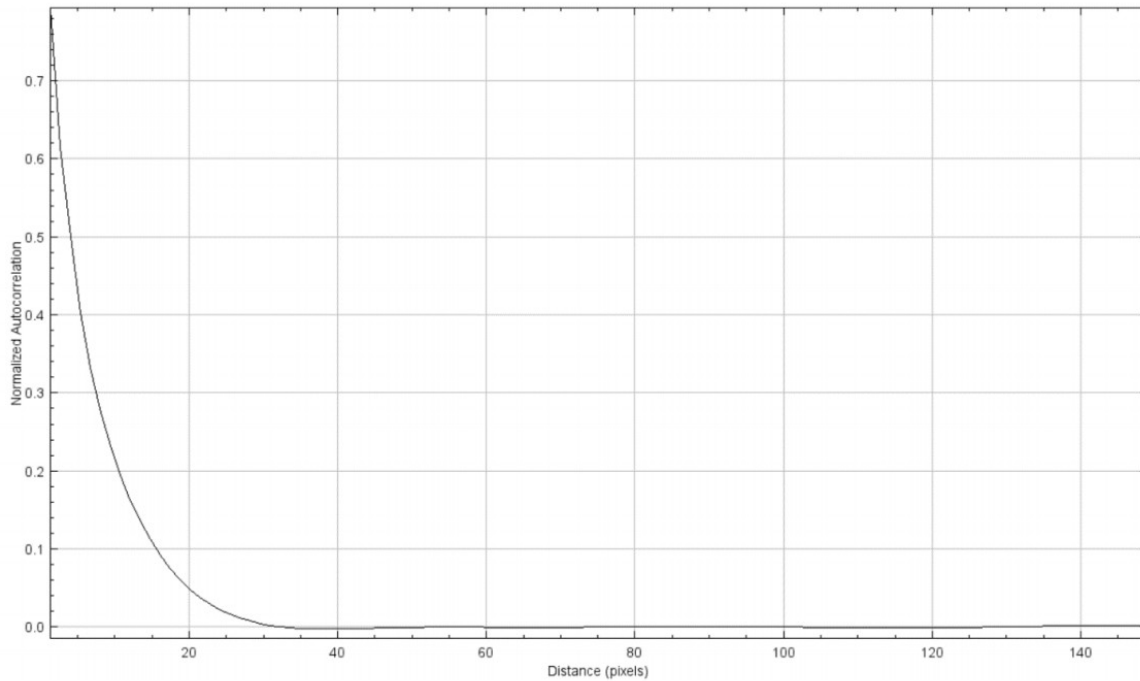


Figure 4-5. Autocorrelation function for the subsample

As the normalized autocorrelation function values start to oscillate around zero-value at around 30 voxels sample-length, and the subsample is 500 voxels large in each orthogonal axis, so it is evident that the subsample is much larger than the calculated REV. This means that the subsample can be treated as representative, and by taking larger subsamples, porosity should not change significantly.

Isolated pores have been filtered out of the sample and effective porosity was calculated. The calculated effective porosity equals to 18.95% so there is almost perfect matching to experimentally obtained effective porosity which equals to 18.96%. After filtering out isolated pores, absolute permeability was calculated using several different methods. Results of permeability calculation using pore areas are shown in Table 4-1.

Table 4-1. Permeability from pore area

radius obtained from:	pore area	pore area · solidity	pore area · circularity
permeability, k (mD)	216.815	49.4	49.4

Three different perimeter types were chosen for pore radii calculation, types 18, 19 and 20. Other perimeter types were massively overestimated sample permeability to the point where

the results had no physical interpretation. Results of permeability calculation using pore perimeter are shown in Table 4-2.

Table 4-2. Permeability from pore perimeter

perimeter method ↓	radius type: →	from pore perimeter, mD	hydraulic radius, mD	geometric mean radius, mD
18		832.86	232.43	372.24
19		1667.534	39.06	87.2
20		440.52	139.49	166.7

Although permeability calculated with hydraulic radius obtained by pore perimeter type 20 best matches the experimental data, due to the inconsistency of perimeter-based methods on other tested samples, permeability calculated from pore area is considered to be the most precise and trustworthy.

Permeability was calculated for nine more samples and compared to results obtained by CFD (Computational Fluid Dynamics) simulation results. The comparison is shown in Table 5-3. All of the samples had $300 \times 300 \times 300$ voxel shape. Permeabilities obtained by pore area show the most consistency of all methods. The problem with pore perimeter method is that very small pores, that have almost no or very little contribution to fluid flow through the sample can still have a significant perimeter (calculated on a sample of limited resolution, and with the assumption of circular shape of pore). This leads to overestimating permeability calculated by using pore perimeters. Although in some cases, results calculated by using pore perimeter (samples S1, S2, S5 and S6; perimeter type 20) matches expected values accurately, it is important to notice that these are extremely permeable samples, that are almost never encountered in oil and gas reservoirs. The pores in these samples are large, so the effect of permeability overestimation is mostly annulled. Still, these methods do not show consistency, as calculated permeability for the most permeable sample (S8) shows the biggest deviation from the expected value. Permeability estimation using pore area shows promising results in high permeability range, but significant deviations from the expected values are present when estimating samples with lower permeability. Taking parameters like pore tortuosity into account could lead to much more consistent and precise results as flow paths with higher tortuosity are

longer and therefore produce greater pressure drop through the sample. Lastly, CFD obtained reference permeabilities are not necessarily very precise as well and unlike with experimental method, many parts of data processing workflow like pore network extraction and image segmentation are critical for proper sample characterization.

Table 4-3. Calculated permeabilities for samples previously analysed by CFD methods

Sample	Voxel resolution, μm	CFD permeability, mD	permeability from area, mD	perimeter type	permeability from perimeter, mD	permeability from hydraulic radius, mD	permeability from geometric mean radius, mD
S1	8.638	1678	2027	19	5569	837	1276
				18	2941	1583	1812
				20	1588	2809	2358
S2	4.956	3898	3019	19	14439	673	1393
				18	7574	1406	2189
				20	4004	2339	2621
S3	9.1	224	1100	19	3488	419	659
				18	1799	1412	1628
				20	1105	1249	1150
S4	8.96	259	865	19	2675	341	528
				18	1369	983	1123
				20	833	1013	921
S5	3.997	4651	3144	19	12069	852	1617
				18	6966	1478	2192
				20	3455	2863	2995
S6	5.1	10974	8919	19	44817	1669	3816
				18	27866	2803	5128
				20	13181	5468	6988
S7	4.803	6966	5201	19	26455	874	2194
				18	14121	1791	3451
				20	7249	3124	4163
S8	4.892	13169	20990	19	463821	613	3565
				18	264438	1050	4868
				20	130326	2124	6780
S9	3.4	3640	1640	19	5673	504	893
				18	3287	871	1223
				20	1645	1667	1641

5. CONCLUSION

The developed method for fast permeability calculation shows promising results with many advantages over numerical flow simulation methods, but also some drawbacks. The most important advantages are:

- Significantly shorter runtime. While numerical flow simulations can take more than a day to complete, methods like this can be performed within hour.
- Significantly less computing power and RAM memory required. These methods can be successfully run on an average PC, while CFD methods often require high performance server (cloud) computers to run.
- These methods provide insight into phenomena that occur at the pore scale, which is beneficial for understanding flow mechanisms that occur in porous media. Permeability for each pore is available and bottlenecks in flow paths can be identified.
- These methods are much easier to use because there is no need to define boundary conditions or fluid/rock interaction data.

Some of the disadvantages of using these methods are:

- Limited data that can be extracted from the sample. Only absolute permeability can be estimated by analysing pore network geometry; flow simulation must be performed if effective permeability or multiphase flow is being examined.
- These methods are less trustworthy than CFD analysis.

More research needs to be done to validate algorithms for calculating permeability based on restrictions. The inclusion of other pore network geometry parameters, such as tortuosity, could lead to a significant improvement in these methods.

However, by examining various methods for perimeter calculation, which also take into account pores that cannot be assumed to have circular shapes, the perimeter method 20, with the hydraulic radius as an intermediate parameter, resulted with good agreement. It should be noted that such results depend on the distribution of pore sizes, and most publicly available CT images are for samples with much higher permeability (i.e., they were not reservoir rocks). Under such conditions, verification of the methods is not consistent and more images of reservoir rocks would confirm or deny the algorithms tested in this work. In addition, based on good data on

pore size distribution (from a high-resolution CT scan or very precise capillary pressure measurements by mercury injection) and data on rock chemistry, some correction factors could be derived, or other algorithms for perimeter calculation would work. If radii are calculated only from the perimeter or only from the 2D pore area, information about the pressure drop due to friction will be biased (or oversimplified). For this reason, the hydraulic radius (which essentially evaluates the difference between the radius from the 2D pore area and the radius calculated from the pore perimeter) is the most promising parameter for such calculations.

The work took heavy computing permeability problem with runtime optimization as an objective. Many other core properties have been tested and obtained, such as absolute (sum of ones in the 3D array) and effective porosity (sum of ones in the 3D array of connected pores) or pore size distribution (corresponding to capillary pressure measurement), which opens a tremendous possibility for rock analysis even if no standard size of the core is available or if the core sample is damaged.

In this sense, the development of a code for digital core analysis is a valuable effort that opens up many possibilities for the future of petrophysics in smaller laboratories, or when core preparation, preservation, cleaning, and analysis are not economically feasible.

6. REFERENCES

1. Andrä, H., Combaret, N., Dvorkin, J., Glatt, E., Han, J., Kabel, M., ... & Zhan, X., 2013. Digital rock physics benchmarks—Part I: Imaging and segmentation. *Computers & Geosciences*, 50, 25-32.
2. Andrä, H., Combaret, N., Dvorkin, J., Glatt, E., Han, J., Kabel, M., Keehm, Y., Krzikalla, F., Lee, M., Madonna, C., Marsh, M., Mikerji, T., Saenger, E. H., Sain, R., Saxena, N., Ricker, S., Wiegmann, A., Zhan, X., 2013. Digital rock physics benchmarkspart ii: computing effective properties, *Computers&Geosciences*, 50.
3. Benkrid, K., Crookes, D., & Benkrid, A., 2000. Design and fpga implementation of a perimeter estimator. In *Proceedings of the Irish Machine Vision and Image Processing Conference* (pp. 51-57).
4. Blunt, M. J., Bijeljic, B., Dong, H., Gharbi, O., Iglauer, S., Mostaghimi, P., Paluszny, A., Pentland, C., 2013. Pore-scale imaging and modelling, *Advances in Water Resources*,str. 197-216.
5. Bradley, D., & Roth, G., 2007. Adaptive thresholding using the integral image. *Journal of graphics tools*, 12(2), 13-21.
6. Buades, A., Coll, B., Morel, J. M., 2005. A non-local algorithm for image denoising, *Computer vision and Pattern Recognition, IEEE Computer Society Conference on*, vol. 2.
7. Dong, H., Blunt, M. J., 2009. Pore-network extraction from micro-computerized-tomography images, *Phys. Rev.*, E80 (3).
8. Eibenberger, E., Borsdorf, A., Wimmer, A., Hornegger, J., 2008. Edge-perserving denoising for segmentation in ct-images, *Bildverarbeitung für die Medizin 2008*. Springer.
9. Fredrich, J. T., Lakshatanov, D., Lane, N., Liu, E. B., Natarajan, C., Ni, D. M., Toms, J., 2014. Digital rocks: developing an emerging technology through to a proven capatibility deployed in the business, *SPE Annual Technical Conference and Exhibition, Society of Petroleum Engineers*.
10. Harris, C.R., Millman, K.J., van der Walt, S.J. et al., Array programming with NumPy. *Nature* 585, 357–362 2020. DOI: 0.1038/s41586-020-2649-2. (Publisher link).

11. Huang, Z. K., Chau, K. W., 2008. A new image thresholding method based on gaussian mixture model, *Applied Mathematics and Computation*, 205.
12. Iassonov, P., Gebrenegus, T., Tuller, M., 2009. Segmentation of x-ray computed tomography images of porous materials: a crucial step for characterization and quantitative analysis of pore structures, *Water Resources Research* 45 (W09415).
13. Leu, L., Berg, S., Enzmann, F., Armstrong, R., Kersten, M., 2014. Fast x-ray tomography of multiphase flow in Berea sandstone: a sensitivity analysis on image processing, *Transport in Porous Media*, ISSN: 0169-3913 105(2).
14. Mandzhieva, R., 2017. Introduction to digital core analysis: 3D reconstruction, numerical flow simulations and pore network modeling, Norwegian University of Science and Technology.
15. McKinney, W., 2010. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference* (Vol. 445, pp. 51-56).
16. Mukunoki, T., Miyata, Y., Mikami, K., Shiota, E., 2016. X-ray CT analysis of pore structure in sand, *Solid Earth*, 7.
17. Neumann, R. F., Barsi-Andreetta, M., Lucas-Oliveira, E., Barbalho, H., Trevizan, W. A., Bonagamba, T. J., & Steiner, M. B., 2021. High accuracy capillary network representation in digital rock reveals permeability scaling functions. *Scientific reports*, 11(1), 1-8.
18. Otsu, N., 1979. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1), 62-66.
19. Rabbani, A., Jamshidi, S., & Salehi, S., 2014. An automated simple algorithm for realistic pore network extraction from micro-tomography images. *Journal of Petroleum Science and Engineering*, 123, 164-171.
20. Schindelin, J., Arganda-Carreras, I., Frise, E., Kaynig, V., Longair, M., Pietzsch, T., ... Cardona, A., 2012. Fiji: an open-source platform for biological-image analysis. *Nature Methods*, 9(7), 676–682. doi:10.1038/nmeth.2019.
21. Schlüter, S., Sheppard, A., Brown, K., Wildenschild, D., 2014. Image processing of multiphase images obtained via x-ray microtomography; a review, *Water Resources Research* 50 (4).

22. Silin, D., Patzek, T., 2006. Pore space morphology analysis using maximal inscribed spheres, *Physica A: Statistical Mechanics and its Applications*, ISSN: 038-4371 371 (2).
23. Verri, I., Della Torre, A., Montenegro, G., Onorati, A., Duca, S., Mora, C. A., Radaelli, F., Trombin, G., 2017. Development of a Digital Rock Physics workflow for the analysis of sandstones and tight rocks, *Journal of Petroleum Science and Engineering* 156.
24. Yeoung, C. L. Y., Torquato, S., 1998. Reconstructing random media, *Physical review E* 57(1).
25. Yousefi, J., 2011. Image binarization using Otsu thresholding algorithm. Ontario, Canada: University of Guelph.

Web references:

26. Scikit image; Image processing in Python: <https://scikit-image.org/> (accessed on 13.6.2021.)
27. Mahotas: Computer vision in Python: <https://mahotas.readthedocs.io/en/latest/labeled.html> (accessed on 13.6.2021.)
28. ImageJ: <https://imagej.net/> (accessed on 19.6.2021.)
29. ChrisKnott/Eel, A little Python library for making simple Electron-like HTML/JS GUI apps: <https://github.com/ChrisKnott/Eel> (accessed on 19.6.2021.)

7. APPENDIX – PYTHON CODE

7.1. binarization.py

```
import cv2 as cv
import numpy as np
from tqdm import tqdm
from skimage import img_as_float, exposure, io

def loadImage(path):
    img = np.uint8((cv.imreadmulti(path)[1]))
    return(img)

def loadImage(path):
    #loads greyscale image
    img = io.imread(path)
    img = exposure.adjust_gamma(img, 1.5, 2)
    return(img)

def denoise(img, type=["nlm"], i=0):
    if "nlm" in type:
        clean = cv.fastNlMeansDenoisingMulti(img, i, 1, None, 4, 7, 35)
    if 'median' in type:
        clean = cv.medianBlur(img, 3)
    return(clean)

def thresh(img):
    gamma = exposure.adjust_gamma(img, 1.5, 2)
    ret, th = cv.threshold(gamma, 0, 255, cv.THRESH_BINARY+cv.THRESH_OTSU)
    return(ret)

def morpho(img, it=1, structure=1, operation='open'):
    """
    open = erosion + dilation
    close = dilation + erosion
    """
    if structure == 1:
        kernel = np.ones((3, 3), np.uint8)
    if structure == 2:
        kernel = np.array([[0, 1, 0],
                           [1, 1, 1],
                           [0, 1, 0]], np.uint8)
    if operation == 'open':
        opened = cv.morphologyEx(img, cv.MORPH_OPEN, kernel, iterations=it)
        return(opened)
    if operation == 'close':
        closed = cv.morphologyEx(img, cv.MORPH_CLOSE, kernel, iterations=it)
        return(closed)

def main():
    import pickle
    configDir = 'config\\'
    userConfigFile = 'user_config.p'
    ucf = configDir + userConfigFile          # full path to user cofig file

    with open(ucf, 'rb') as f:
```

```

    userConfig = pickle.load(f)
    filters=[]

    path=userConfig['imgpath']
    savepath=userConfig['sourceDir']

    if int(userConfig['NLM']) == 1:
        denois = True
        filters+=['nlm']
    elif int(userConfig['MedianFilter']) == 1:
        denois = True
        filters+=['median']
    else:
        denois = False

    numberDigits = userConfig['numberDigits']

    img = loadImage(path)

    if denois:
        clean=np.zeros(np.shape(img), np.uint8)
        for i in tqdm(range(len(clean))):
            clean[i] = denoise(img, filters)
    else:
        clean = img

    numberFormat = '0'+ str(numberDigits) + 'd'
    binary = np.zeros(np.shape(img), np.uint8)
    ret = np.zeros(len(clean))
    for i in tqdm(range(len(binary))):
        ret[i] = thresh(clean[i])
    thr = np.min(ret)
    for i in tqdm(range(len(binary))):
        binary[i] = np.array(clean[i] > thr) * 255
        np.savetxt(savepath+str(format(i, numberFormat))+'.txt', binary[i], fmt='%i')

    porosity = np.sum((binary==0)*1)/np.size(binary)

    import eel
    @eel.expose
    def sendporo():
        return(porosity)

    sendporo()
    eel.refresh()

    return(img, binary, ret, thr)

```

7.2. permeability.py

```
import numpy as np
from scipy.ndimage import label
from scipy import ndimage
from numba import jit
from tqdm import tqdm
import mahotas
import cv2 as cv
from skimage import measure
import pickle

def load3DMatrix (nz, nx, ny, imageDir = '\\img'):
    print ("loading 3D matrix...")
    m = np.zeros([nz, ny, nx])
    for i in tqdm(range(1, nz+1)):
        slice = np.load(imageDir + str(i-1)+'.input.npz')['arr_0']
        m[i-1, :, :] = slice[:ny, :nx]
    return (m)

@jit(nopython=True)
def binaryToImage(im):
    img = im.astype(np.uint8) * 255
    return (img)

def labelPores(myImage, pattern = 2):
    if pattern == 1:
        pattern_3D = np.array([
            [[1, 1, 1],
             [1, 1, 1],
             [1, 1, 1]],
            [[1, 1, 1],
             [1, 1, 1],
             [1, 1, 1]],
            [[1, 1, 1],
             [1, 1, 1],
             [1, 1, 1]]])
    if pattern == 2:
        pattern_3D = np.array([
            [[0, 1, 0],
             [1, 1, 1],
             [0, 1, 0]],
            [[0, 1, 0],
             [1, 1, 1],
             [0, 1, 0]],
            [[0, 1, 0],
             [1, 1, 1],
             [0, 1, 0]]])

    labeledArray, numFeatures = label(myImage>0, structure = pattern_3D)
    return (labeledArray, numFeatures)

def rotatedArray(array2D):
    newArray = ndimage.rotate(array2D, 45, reshape = False)
    return (newArray)
```

```

def connectedPores(myArray):
    connected_pores = myArray.copy()
    test_elements = np.unique(connected_pores[-1].flatten()).copy()
    connected_pores = connected_pores*(1*np.isin(connected_pores, test_elements))
    test_elements = np.unique(connected_pores[0].flatten()).copy()
    connected_pores = connected_pores*(1*np.isin(connected_pores, test_elements))
    return (connected_pores)

@jit(nopython=True)
def perimeter(slice, voxelResolution, boundary = True):
    if boundary:
        p = np.sum(slice[:,1:] != slice[:, :-1]) + np.sum(slice[1:,:] != slice[:-1,:])
        \
        + np.sum(slice[0,:]) + np.sum(slice[:,0]) + np.sum(slice[-1,:]) +
np.sum(slice[:, -1])
    else:
        p = np.sum(slice[:,1:] != slice[:, :-1]) + np.sum(slice[1:,:] != slice[:-1,:])
    return (p * voxelResolution)

@jit(nopython=True)
def perimeterCV(slice, voxelResolution):
    img = binaryToImage(slice)
    ret, thresh = cv.threshold(img, 127, 255, 0)
    contours, hierarchy = cv.findContours(thresh, 1, 2)
    cnt = contours[0]
    p = cv.arcLength(cnt, True)
    return (p * voxelResolution)

def perimeterScikit(slice, voxelResolution = 1):
    pr = measure.perimeter(slice)
    if pr == 0: pr = 1e-30
    return (pr*voxelResolution)

def perimeterE(slice, voxelResolution):
    eroded = ndimage.binary_erosion(slice).astype(slice.dtype)
    a1 = np.sum(eroded)
    a2 = np.sum(slice)
    p = a2-a1
    return (p * voxelResolution)

def perimeterD(slice, voxelResolution):
    dilated = ndimage.binary_dilation(slice).astype(slice.dtype)
    a1 = np.sum(dilated)
    a2 = np.sum(slice)
    p = a1-a2
    return (p * voxelResolution)

def circularity(area, perimeter):
    cl = 4*3.14159265359*area/(perimeter**2)
    # if area is 1 pixel:
    #     radius from perimeter will be smaller than radius from area
    if cl>1: cl=1 # this would be the radius of 2 pixel object
    return (cl)

```

```

perimeterTypeDesc=[
    'perimeter with boundaries',
    'perimeter without boundaries',
    'perimeter with boundaries, averaged value with rotated',
    'perimeter without boundaries, averaged value with rotated',
    'perimeter with boundaries, min value compared with rotated',
    'perimeter without boundaries, min value compared with rotated',
    'perimeter eroded = perimeter as pixels in pore - pixels in eroded
pore',
    'perimeter eroded averaged with rotated 45 degrees',
    'perimeter dilation',
    'perimeter dilation averaged with rotated 45 degrees',
    'minimum of eroded and rotated 45 degrees',
    'minimum of dilated and rotated 45 degrees',
    'perimeter with boundaries * solidity',
    'perimeter without boundaries * solidity',
    'mahotas perimeter',
    '(dilated + eroded)/2',
    'scikit image perimeter',
    'scikit image perimeter * solidity',
    'mahotas perimeter * solidity',
    '((dilated + eroded)/2)*solidity',
    'dilated * circularity',
    'perimeter with boundaries * circularity'
]

def perimeterTypes(slice, voxelResolution, perType=1):
    if perType == 0:
        # perimeter with boundaries
        pr = perimeter(slice, voxelResolution, True)
    if perType == 1:
        # perimeter without boundaries
        pr = perimeter(slice, voxelResolution, False)
    if perType == 2:
        # perimeter with boundaries, averaged value with rotated
        pr = perimeter(slice, voxelResolution, True)
        pr = 0.5*(pr + perimeter(rotatedArray(slice), voxelResolution, True))
    if perType == 3:
        # perimeter without boundaries, averaged value with rotated
        pr = perimeter(slice, voxelResolution, False)
        pr = 0.5*(pr + perimeter(rotatedArray(slice), voxelResolution, False))
    if perType == 4:
        # perimeter with boundaries, min value compared with rotated
        pr = min(perimeter(slice, voxelResolution, True),
                  perimeter(rotatedArray(slice), voxelResolution, True))
    if perType == 5:
        # perimeter without boundaries, min value compared with rotated
        pr = min(perimeter(slice, voxelResolution, False),
                  perimeter(rotatedArray(slice), voxelResolution, False))
    if perType == 6:
        # perimeter eroded = perimeter as pixels in pore - pixels in eroded pore
        pr = perimeterE(slice, voxelResolution)
    if perType == 7:
        # perimeter eroded averaged with rotated 45 degrees
        pr = 0.5*(perimeterE(slice, voxelResolution) +

```

```

        perimeterE(rotatedArray(slice), voxelResolution))
if perType == 8:
    # perimeter dilation
    pr = perimeterD(slice, voxelResolution)
if perType == 9:
    # perimeter dilation averaged with rotated 45 degrees
    pr = 0.5*(perimeterD(slice, voxelResolution) +
        perimeterD(rotatedArray(slice), voxelResolution))
if perType == 10:
    # minimum of eroded and rotated 45 degrees
    pr = min(perimeterE(slice, voxelResolution),
        perimeterE(rotatedArray(slice), voxelResolution))
if perType == 11:
    # minimum of dilated and rotated 45 degrees
    pr = min(perimeterD(slice, voxelResolution),
        perimeterD(rotatedArray(slice), voxelResolution))
if perType == 12:
    # perimeter with boundaries * solidity
    s = measure.regionprops(slice)[0].solidity
    pr = perimeter(slice, voxelResolution, True)*s
if perType == 13:
    # perimeter without boundaries * solidity
    s = measure.regionprops(slice)[0].solidity
    pr = perimeter(slice, voxelResolution, False)*s
if perType == 14:
    # mahotas perimeter
    pr = np.sum(mahotas.bwperim(slice, n=8)*1)*voxelResolution
if perType == 15:
    # (dilated + eroded)/2
    pr = 0.5*(perimeterD(slice, voxelResolution)+
        perimeterE(slice, voxelResolution))
if perType == 16:
    # Scikit
    pr = perimeterScikit(slice, voxelResolution)
if perType == 17:
    # Scikit * solidity
    s = measure.regionprops(slice)[0].solidity
    pr = perimeterScikit(slice, voxelResolution)*s

if perType == 18:
    # mahotas perimeter * solidity
    s = measure.regionprops(slice)[0].solidity
    pr = np.sum(mahotas.bwperim(slice, n=8)*1)*voxelResolution*s

if perType == 19:
    # ((dilated + eroded)/2)*solidity
    s = measure.regionprops(slice)[0].solidity
    pr = s*0.5*(perimeterD(slice, voxelResolution)+
        perimeterE(slice, voxelResolution))

if perType == 20:
    # dilated * circularity
    o = perimeterD(slice, 1)
    p = np.sum(slice)
    cl = circularity(p, o)

```

```

        pr = o*cl*voxelResolution

    if perType == 21:
        # perimeter with boundaries
        o = perimeter(slice, 1, False)
        p = np.sum(slice)
        cl = circularity(p, o)
        pr = o*cl*voxelResolution
    return (pr)

def printSlices(myArray, showImage=False):
    if len(myArray)>5: showImage = False
    if showImage: import matplotlib.pyplot as plt
    for slice in myArray:
        print(slice)
        if showImage:
            plt.figure()
            plt.imshow(slice, cmap='Greys', interpolation='none')
            plt.show()
    return (True)

def radiusFromArea(area):
    return ((area/3.14159265359)**0.5)

def perimeterGroups(labConPores, voxelResolution, perimeterType=1):
    pg = np.zeros(len(labConPores), dtype=object)
    i = 1
    while i < len(labConPores):
        conPores, numPores = labelPores(labConPores[i-1:i+1], 2)
        perimeters1, perimeters2 = [], []
        for pore in np.intersect1d(conPores[0], conPores[1])[1:]:
            perimeters1.append(perimeterTypes((conPores[0]==pore)*1, voxelResolution,
perimeterType))
            perimeters2.append(perimeterTypes((conPores[1]==pore)*1, voxelResolution,
perimeterType))
        pg[i-1], pg[i] = perimeters1.copy(), perimeters2.copy()
        i += 2
    return (pg)

def weightedAreas(areaGroups, voxelResolution):
    ai = (areaGroups/np.sum(areaGroups))*voxelResolution**2
    return (ai)

def areaGroups(labConPores, voxelResolution):
    ag = np.zeros(len(labConPores), dtype=object)
    i = 1
    while i < len(labConPores):
        conPores, numPores = labelPores(labConPores[i-1:i+1], 2)
        areas1, areas2 = [], []
        for pore in np.intersect1d(conPores[0], conPores[1])[1:]:
            areas1.append(np.sum(conPores[0]==pore)*1 * voxelResolution**2)
            areas2.append(np.sum(conPores[1]==pore)*1 * voxelResolution**2)
        ag[i-1], ag[i] = np.array(areas1.copy()), np.array(areas2.copy())
        i += 2
    return (ag)

```

```

def solidityGroups(labConPores):
    ag = np.zeros(len(labConPores), dtype=object)
    i = 1
    while i < len(labConPores):
        conPores, numPores = labelPores(labConPores[i-1:i+1], 2)
        sols1, sols2 = [], []
        for pore in np.intersect1d(conPores[0], conPores[1])[1:]:
            s1 = measure.regionprops((conPores[0]==pore)*1)[0].solidity
            sols1.append(s1)
            s2 = measure.regionprops((conPores[1]==pore)*1)[0].solidity
            sols2.append(s2)
        ag[i-1], ag[i] = np.array(sols1.copy()), np.array(sols2.copy())
        i += 2
    return (ag)

def circularityGroups(labConPores, perimeterType=20):
    """
    Measure of similarity of the pore area and the equivalent circle area
     $cl = 4 * \pi * \text{area} / (\text{perimeter}^2)$ 
    Parameters
    -----
    labConPores : numpy array
        3D array of connected pores
    perimeterType : integer
        Algorithm for calculating perimeter.
        Note: not recommended for changing as the area is calculated as
        sum of pixels, so it could result with  $c > 1$ 
    Returns
    -----
    cl : numpy array of arrays
        circularities for all pores connected to next slice
        In each slice, pores are detected and analyzed for circularity.
    """
    cl = np.zeros(len(labConPores), dtype=object)
    i = 1
    while i < len(labConPores):
        conPores, numPores = labelPores(labConPores[i-1:i+1], 2)
        circls1, circls2 = [], []
        for pore in np.intersect1d(conPores[0], conPores[1])[1:]:
            region1 = (conPores[0]==pore)*1
            region2 = (conPores[1]==pore)*1
            perimeter1 = perimeterTypes(region1, 1, perimeterType)
            perimeter2 = perimeterTypes(region2, 1, perimeterType)
            circls1.append(circularity(np.sum(region1), perimeter1))
            circls2.append(circularity(np.sum(region2), perimeter2))
        cl[i-1], cl[i] = np.array(circls1.copy()), np.array(circls2.copy())
        i += 2
    return (cl)

def permeabilityFromPG(perGroups, A_slice, Li, hTotal):
    i, pi = 1, 3.14159265359
    nslices = len(perGroups)
    c = pi/(8*A_slice)
    sliceGroupPerm = np.ones(np.int(nslices/2))

```

```

i_sgp = 0
seriesk = []
while i < len(perGroups):
    npores = len(perGroups[i])
    k_avg_series = np.ones(npores)
    for pn in np.arange(npores):
        try:
            r1 = perGroups[i-1][pn]/(2*pi)
            r2 = perGroups[i][pn]/(2*pi)
            k1 = (r1**4)*c
            k2 = (r2**4)*c
            k_avg_series[pn] = np.nan_to_num((2*Li) / (Li/k1 + Li/k2))
        except:
            k_avg_series[pn] = 0
    seriesk.append(k_avg_series)
    sliceGroupPerm[i_sgp] = Li*np.sum(k_avg_series)/hTotal
    i_sgp += 1
    i += 2
k_avg = np.nan_to_num(nsllices*Li/(np.sum(Li/sliceGroupPerm)))
return (k_avg, sliceGroupPerm, seriesk)

def permeabilityFromHR(perGroups, aGroups, A_slice, Li, hTotal):
    """
    Parameters
    -----
    perGroups : TYPE
        groups of pore perimeters
    aGroups : TYPE
        groups of pore areas
    A_slice : TYPE
        area of one slice
    Li : TYPE
        width of one slice (voxel dimension)
    hTotal : TYPE
        total dimension in Z direction

    Returns
    -----
    k_avg : real
        average permeability
    sliceGroupPerm : array
        permeability for each slice group
    seriesk : list of arrays
        DESCRIPTION.
    """
    # hydraulic radius : the ratio of the cross-sectional area of a channel or pipe
    in which a fluid is flowing to the wetted perimeter of the conduit

    i, pi = 1, 3.14159265359
    nsllices = len(perGroups)
    c = pi/(8*A_slice)
    sliceGroupPerm = np.ones(np.int(nsllices/2))
    i_sgp = 0
    seriesk = []

```

```

while i < len(perGroups):
    npores = len(perGroups[i])
    k_avg_series = np.ones(npores)
    for pn in np.arange(npores):
        try:
            # try is needed for more complex perimeter types (rotation etc)
            r1 = 2*aGroups[i-1][pn]/perGroups[i-1][pn]
            r2 = 2*aGroups[i][pn]/perGroups[i][pn]
            k1 = (r1**4)*c
            k2 = (r2**4)*c
            k_avg_series[pn] = np.nan_to_num((2*Li) / (Li/k1 + Li/k2))
        except:
            k_avg_series[pn] = 0
    seriesk.append(k_avg_series)
    sliceGroupPerm[i_sgp] = Li*np.sum(k_avg_series)/hTotal
    i_sgp += 1
    i += 2
k_avg = np.nan_to_num(nsllices*Li/(np.sum(Li/sliceGroupPerm)))
return (k_avg, sliceGroupPerm, seriesk)

def permeabilityFromGMR(perGroups, aGroups, A_slice, Li, hTotal):
    """
    Parameters
    -----
    perGroups : TYPE
        groups of pore perimeters
    aGroups : TYPE
        groups of pore areas
    A_slice : TYPE
        area of one slice
    Li : TYPE
        width of one slice (voxel dimension)
    hTotal : TYPE
        total dimension in Z direction

    Returns
    -----
    k_avg : real
        average permeability
    sliceGroupPerm : array
        permeability for each slice group
    seriesk : list of arrays
        permeabilities of each pore in each slice-pair
    """
    i, pi = 1, 3.14159265359
    nsllices = len(perGroups)
    c = pi/(8*A_slice)
    sliceGroupPerm = np.ones(np.int(nsllices/2))
    i_sgp = 0
    seriesk = []
    while i < nsllices:
        npores = len(perGroups[i])
        k_avg_series = np.ones(npores)
        for pn in np.arange(npores):
            try:

```

```

        # try is needed for more complex perimeter types (rotation etc)
        hr1 = 2*aGroups[i-1][pn]/perGroups[i-1][pn]
        r1 = pow(hr1*(pow((np.abs(aGroups[i-1][pn]))/pi,0.5)),0.5)
        hr2 = 2*aGroups[i][pn]/perGroups[i][pn]
        r2 = pow(hr2*(pow((np.abs(aGroups[i][pn]))/pi,0.5)),0.5)
        k1 = (r1**4)*c
        k2 = (r2**4)*c
        # import pdb ; pdb.set_trace()
        k_avg_series[pn] = np.nan_to_num((2*Li) / (Li/k1 + Li/k2))
    except:
        k_avg_series[pn] = 0
    seriesk.append(k_avg_series)
    sliceGroupPerm[i_sgp] = Li*np.sum(k_avg_series)/hTotal
    i_sgp += 1
    i += 2
k_avg = nslices*Li/(np.sum(Li/sliceGroupPerm))
return (k_avg, sliceGroupPerm, seriesk)

def permeabilityFromArea(aGroups, A_slice, Li, hTotal):
    i, pi = 1, 3.14159265359
    nslices = len(aGroups)
    c = pi/(8*A_slice)
    sliceGroupPerm = np.ones(np.int(nslices/2))
    i_sgp = 0
    seriesk = []
    while i < len(aGroups):
        npores = len(aGroups[i])
        k_avg_series = np.ones(npores)
        for pn in np.arange(npores):
            r1 = radiusFromArea(aGroups[i-1][pn])
            r2 = radiusFromArea(aGroups[i][pn])
            k1 = (r1**4)*c
            k2 = (r2**4)*c
            if k1==0 or k2==0:
                k_avg_series[pn] = 0
            else:
                k_avg_series[pn] = np.nan_to_num((2*Li) / (Li/k1 + Li/k2))
        seriesk.append(k_avg_series)
        sliceGroupPerm[i_sgp] = Li*np.sum(k_avg_series)/hTotal
        i_sgp += 1
        i += 2
    k_avg = np.nan_to_num(nslices*Li/(np.sum(Li/sliceGroupPerm)))
    return (k_avg, sliceGroupPerm, seriesk)

def permeabilityFromAreaSolidity(aGroups, A_slice, Li, hTotal):
    i, pi = 1, 3.14159265359
    nslices = len(aGroups)
    c = pi/(8*A_slice)
    sliceGroupPerm = np.ones(np.int(nslices/2))
    i_sgp = 0
    seriesk = []
    while i < len(aGroups):
        npores = len(aGroups[i])
        k_avg_series = np.ones(npores)
        for pn in np.arange(npores):

```

```

        r1 = radiusFromArea(aGroups[i-1][pn])
        r2 = radiusFromArea(aGroups[i][pn])
        k1 = (r1**4)*c
        k2 = (r2**4)*c
        if k1==0 or k2==0:
            k_avg_series[pn] = 0
        else:
            k_avg_series[pn] = np.nan_to_num((2*Li) / (Li/k1 + Li/k2))
        seriesk.append(k_avg_series)
        sliceGroupPerm[i_sgp] = Li*np.sum(k_avg_series)/hTotal
        i_sgp += 1
        i += 2
    k_avg = np.nan_to_num(nsllices*Li/(np.sum(Li/sliceGroupPerm)))
    return (k_avg, sliceGroupPerm, seriesk)

def main():
    #EDIT 9.6.2021.
    configDir = 'config\\'
    userConfigFile = 'user_config.p'
    ucf = configDir + userConfigFile          # full path to user cofig file

    with open(ucf, 'rb') as f:
        userConfig = pickle.load(f)

    npzImgDir = userConfig['targetDir']

    import time
    np.seterr(all='raise')
    t0 = time.time()
    nz, nx, ny = int(userConfig['nz']), int(userConfig['nx']), int(userConfig['ny'])
    a = load3DMatrix(nz, nx, ny, imageDir = npzImgDir)
    res = float(userConfig['voxelResolution']) * 1e-6
    print (f'time elapsed:{time.time()-t0} s')
    t0 = time.time()

    if len(a) & 0x1:
        print ('odd number of slices cannot be processed, adding copy of the last
slice...')
        a = np.concatenate((a, [a[-1]]), axis=0)
        # printSlices(a)
        labeledPores, poreNumber = labelPores(a, 1)          # pattern: 1 = 3x3 (full),
2 = cross
        labeledConnectedPores = connectedPores(labeledPores)
        areas = areaGroups(labeledConnectedPores, res)
        A = nx * ny * res**2
        h = res * nx
        kFromA = userConfig['kFromA']
        effective_porosity = (np.sum((labeledConnectedPores>0)*1) /
labeledConnectedPores.size)
        print(f'effective porosity: {effective_porosity}')

        resultsA=np.ones([1,3])
        if kFromA:

```

```

        k_ar, paralelk_ar, serialk_ar = permeabilityFromArea(areas, A_slice = A,
Li=res, hTotal = h)
        print (f'time elapsed{np.int(time.time()-t0)} s')
        print (f'k_ar = {np.int(k_ar* 1.01325 * 1e15)} mD')
        t0 = time.time()
        resultsA[0,0]=k_ar*resultsA[0,0]

        solidities = solidityGroups(labeledConnectedPores)
        k_car, paralelk_car, serialk_car = permeabilityFromArea(areas*solidities,
                                                                    A_slice = A, Li=res,
hTotal = h)
        resultsA[0, 1]=resultsA[0,1]*k_car
        print(f'time elapsed{np.int(time.time()-t0)} s')
        print (f'solidity corrected area k_car = {np.int(k_car* 1.01325 * 1e15)} mD')
        t0 = time.time()
        circularities = circularityGroups(labeledConnectedPores, perimeterType=20)
        k_car2, paralelk_car2, serialk_car2 =
permeabilityFromArea(areas*circularities,
                                                                    A_slice = A,
Li=res, hTotal = h)
        print(f'time elapsed:{np.int(time.time()-t0)} s')
        print (f'circularity corrected area k_car2 = {np.int(k_car2* 1.01325 * 1e15)}
mD')
        resultsA[0, 2]=resultsA[0,2]*k_car
        t0 = time.time()

        #calcRange=range(0, 22)          # range of perimeter types
        calcRange = userConfig['calcRange']
        results = np.ones([len(calcRange), 3])

        for i, pt in enumerate(calcRange):
            print ('\n perimeter type: %s (%s) ' %(pt, perimeterTypeDesc[pt]))
            perimeters = perimeterGroups(labeledConnectedPores,
                                                voxelResolution=res,
                                                perimeterType=pt)
            k_p, parallelk_p, serialk_p = permeabilityFromPG(perimeters, A_slice = A,
Li=res, hTotal = h)
            results[i,0] = results[i,0]*k_p
            k_hr, parallelk_hr, serialk_hr = permeabilityFromHR(perimeters, areas,
A_slice = A, Li=res, hTotal = h)
            results[i,1] = results[i,1]*k_hr
            k_gmr, parallelk_gmr, serialk_gmr = permeabilityFromGMR(perimeters, areas,
A_slice = A, Li=res, hTotal = h)
            results[i,2] = results[i,2]*k_gmr
            print(f'time elapsed:{np.int(time.time()-t0)} s')
            print (f'permeability from perimeter, k_p = {np.int(k_p * 1.01325 * 1e15)}
mD')
            print (f's permeability from hydraulic radius k_hr = {np.int(k_hr* 1.01325 *
1e15)} mD')
            print (f's GMR k_gmr = {np.int(k_gmr* 1.01325 * 1e15)} mD')
            t0 = time.time()

        import pandas as pd
        dfResults = pd.DataFrame(results* 1.01325 * 1e15, columns = ['k_p, mD', 'k_hr, mD',
'k_gmr'])

```

```

calcRange=np.array(calcRange)
dfResults['opis'] = [perimeterTypeDesc[i] for i in calcRange]
#dfResults.to_excel(userConfig['results']+"compared_results.xlsx")
if kFromA:
    dfResultsA = pd.DataFrame(resultsA * 1.01325 * 1e15, columns = ["k_ar, mD",
"k_car, mD", "k_car2, mD"])
    dfResultsA['opis'] = ['pore area']
    with pd.ExcelWriter(userConfig['results']+"compared_results.xlsx") as writer:
        dfResults.to_excel(writer, sheet_name='perimeter')
        if kFromA:
            dfResultsA.to_excel(writer, sheet_name='area')

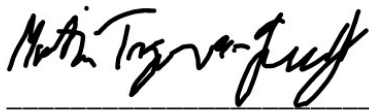
import os
os.system("start EXCEL.EXE " + userConfig['results'] + "compared_results.xlsx")

import eel
@eel.expose
def send():
    return('Permeability calculated successfully. Go to results directory to view
results.')
    send()
    eel.refresh()
    print(f'effective porosity: {effective_porosity}')
    return(effective_porosity)

```

IZJAVA:

Izjavljujem da sam ovaj rad izradio samostalno na temelju znanja stečenih na Rudarsko – geološko – naftnom fakultetu služeći se navedenom literaturom



Martin Trgovec-Greif



KLASA: 602-04/21-01/93
URBROJ: 251-70-12-21-2
U Zagrebu, 12.7.2021.

Martin Trgovec-Greif, student

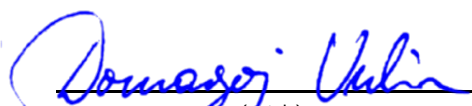
RJEŠENJE O ODOBRENJU TEME

Na temelju vašeg zahtjeva primljenog pod KLASOM 602-04/21-01/93, URBROJ: 251-70-12-21-1 od 21.4.2021. priopćujemo vam temu diplomskog rada koja glasi:

APPLICATION FOR ABSOLUTE PERMEABILITY ANALYSIS BASED ON DIGITAL ROCK SAMPLE

Za voditelja ovog diplomskog rada imenuje se u smislu Pravilnika o izradi i obrani diplomskog rada Izv.prof.dr.sc. Domagoj Vulin nastavnik Rudarsko-geološko-naftnog-fakulteta Sveučilišta u Zagrebu

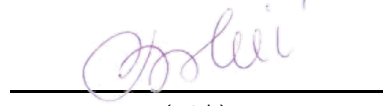
Voditelj:


(potpis)

Izv.prof.dr.sc. Domagoj Vulin

(titula, ime i prezime)

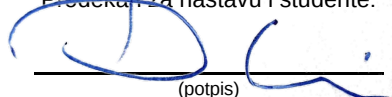
Predsjednik povjerenstva za
završne i diplomske ispite:


(potpis)

Izv.prof.dr.sc. Vladislav Brkić

(titula, ime i prezime)

Prodikan za nastavu i studente:


(potpis)

Izv.prof.dr.sc. Dalibor
Kuhinek

(titula, ime i prezime)