User Guide

pfem4ec.jl

Version: 1.0.0

September, 2020

License: GPL 3.0

Authors:

- Victor Wriedt Sapucaia [1]
- Pedro Cortez Fetter Lopes [2]
- Andre Maues Brabo Pereira [2]
- Ricardo Leiderman [2]
- Rodrigo Bagueira Azeredo [3]

[1] School of Engineering, Fluminense Federal University

[2] Institute of Computing, Fluminense Federal University

[3] Institute of Chemistry, Fluminense Federal University

Introduction

This is a simple user guide to pfem4ec.jl, a script-like Julia program for the computation of the effective electrical conductivity of heterogeneous materials from binary RAW files that represent the microscale with 8-bit information. A few brief considerations, examples of usage and coupling of this tool with the pytomoviewer application are presented in this document.

Some brief considerations

pfem4ec.jl is a Julia in-core solver to compute effective electrical conductivity of heterogeneous materials from raw images using Pixel-Based Finite Element Method (PFEM). This is achieved through the analysis of electric conduction phenomena. The implemented algorithm follows an assembly-free strategy, also known as Element-by-Element, allowing the study of large models without the need for out-of-core solutions.

Two input files are needed for the analysis to be run. One is a binary RAW file that represents an image of the microscale of a certain body, with an 8-bit grayscale. That is, each pixel is associated with an integer between 0 and 255, indicating which material it has. The other is a JSON header file, homonymous to the first one, containing some specific parameters to guide the analysis (this is further detailed on the example of usage presented in this document). Both files can be consistently generated, from a TIFF file that represents the original image, via the pytomoviewer application.

Julia was chosen because the code can be as simple as in a scripting language (such as Python or Matlab/Octave), and the program can also be as efficient as in compiled languages, as it takes advantage of a just-in-time (JIT) technology. In light of this, the code of pfem4ec.jl was designed to be familiar to those who work with script-interpreted languages, without giving up performance.

To run pfem4ec.jl, users need to have downloaded and set up the Julia environment on their computer. If you don't have access to Julia yet, please refer to their official website to learn more about setting it up, <https://julialang.org/>.

Example of usage

In this section, an example of how to run pfem4ec.jl is presented, by detailing the input files, showing how to run Julia code and exposing the output. An artificially constructed model, shown in Figure 1, with dimensions 5x5 and two different materials (identified by 0 and 255 – black and white), is used to depict the process.

1) Input files

Two files are needed for the analysis to be run. One describes the geometry of the model (a binary RAW file), the other sets specific analysis parameters (a JSON file). Both files must have the same name, just different extensions.

1.1) RAW file

The first input is a binary RAW file, that represents images of the microscale of a heterogeneous material with 8-bit data (integer values within the range 0 to 255). As the model is characterized as pixel-based, an image input is necessary to define the Finite Element mesh, admitting that each pixel is a quadrilateral element. The contents of the file are depicted graphically in Figure 1, where the numbers in red indicate the index of each pixel on an array that stores the 8-bit data.

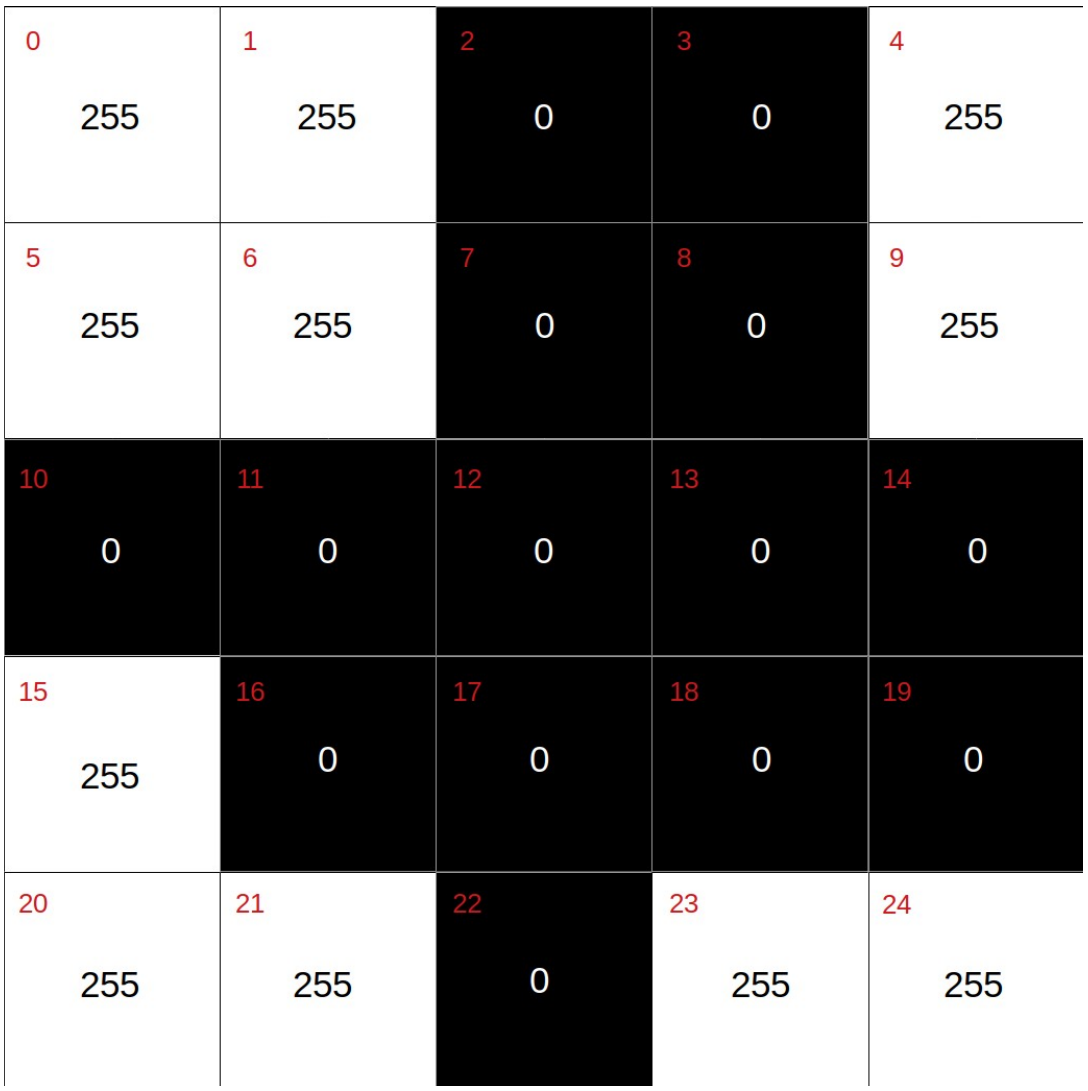
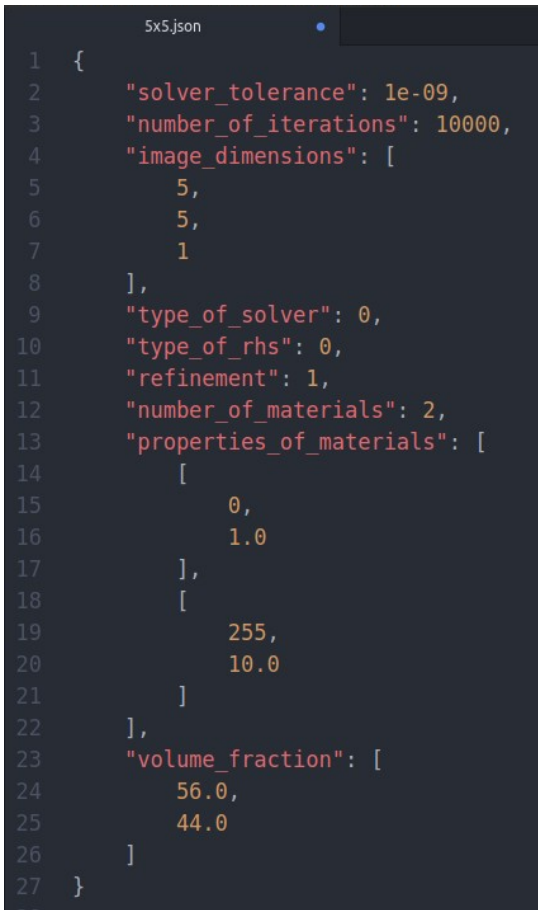


Figure 1 – Visual representation of the contents of a RAW input file for pfem4ec.jl

1.2) JSON header file

A second input file is necessary for pfem4ec.jl to be run, containing specific parameters to guide the FEM analysis. Figure 2 exposes the data that needs to be provided on a JSON file, that must have the same name as the RAW file that describes the geometry of the model.



```
{
    "solver_tolerance": 1e-09,
    "number_of_iterations": 10000,
    "image_dimensions": [
        5,
        5,
        1
    ],
    "type_of_solver": 0,
    "type_of_rhs": 0,
    "refinement": 1,
    "number_of_materials": 2,
    "properties_of_materials": [
        [
            0,
            1.0
        ],
        [
            255,
            10.0
        ]
    ],
    "volume_fraction": [
        56.0,
        44.0
    ]
}
```

Figure 2 – Contents of a JSON input file for pfem4ec.jl

From top to bottom, the input parameters are:

- **Solver tolerance (**solver_tolerance**):** The maximum accepted residual value for the Preconditioned Conjugate Gradient (PCG) Method, adopted to solve the large system of algebraic equations associated with the Finite Element Method.

- **Number of iterations (**number_of_iterations**):** The maximum accepted number of iterations to be run on the PCG method.

- **Image dimensions (**image_dimensions**):** An array of three positive integer values, associated with the dimensions of the model in pixels, on x, y, and z (left to right, up to down, and near to far), respectively. As pfem4ec.jl currently only works for 2D models, the dimension z is unused.

- **Type of solver (**type_of_solver**):** A flag to indicate if the system of equations must be solved with the PCG method or a direct method (matrix factorization). **0 = PCG, 1 = Direct method**. It is advised that the PCG solver is adopted, especially for large models.

- **Type of right-hand side (**type_of_rhs**):** A flag to switch between conditions being set on domain or boundary of the model. **0 = Domain, 1 = Boundary**.

- **Refinement (**refinement**):** A positive integer value to indicate the mesh refinement from the provided image. By setting this as 1, each pixel becomes a single quadrilateral finite element. If it is set as 2, for example, each pixel will be equally subdivided into 4 quadrilateral finite elements (1x1 becomes 2x2, in this case). In an analogous fashion, if set as 3, each pixel becomes 9 equal quadrilateral finite elements, and so on.
- **Number of materials (**number_of_materials**):** A positive integer value to specify how many different materials are present on the heterogeneous model.
- **Properties of materials (**properties_of_materials**):** An array of two values for each material. The first value is an integer between 0 and 255, which correlates the materials with a grayscale color read from the RAW file. The second value is a floating-point number that provides the electrical conductivity of this specific material, assuming that it is isotropic.
- **Volume fraction (**volume_fraction**):** This is not an input parameter to pfem4ec.jl. It contains an array of percentages that indicate the proportion of occurrence of each material on the respective model. This is generated by the pytomoviewer application and is interesting for users to quickly identify any predominant materials on a determined model.

2) Running the code

Firstly, it is important to state that, even though two input files are needed, when running the code, pfem4ec.jl takes a single input, a string containing the name of the model (without file extensions), hence the need to give the same name to correlated RAW and JSON files.

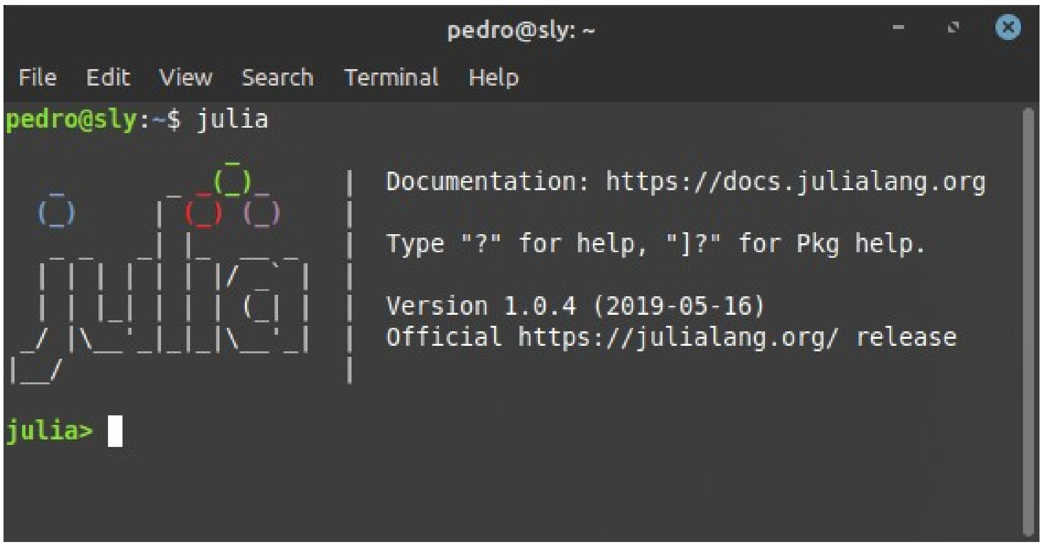
Julia disposes of a few different ways to run code. Due to the script-like nature of the language, it is natural for developers and users to simply call a command line from a terminal like this:

```
your/current/directory $ julia pfem4ec.jl model_filename
```

This works just fine to run pfem4ec.jl, but users need to bear in mind that Julia compiles the code prior to running, meaning that each call like the one presented above will take some extra time to compile (for relatively small models, this usually takes more time than the analysis itself). Certainly, if no changes are made to pfem4ec.jl, there's no need to compile the same code over and over again. In light of this, the authors suggest that users run pfem4ec.jl from within the Julia environment, as follows.

a) Start the Julia environment. Your terminal should then display something similar to Figure 3.
P.S.: Make sure that you're on the directory where pfem4ec.jl and the input files are placed.

```
your/current/directory $ julia
```



```
pedro@sly:~$ julia
               _
   _       _ _(_)_     |  Documentation: https://docs.julialang.org
  (_)     | (_) (_)    |
   _ _   _| |_  __ _   |  Type "?" for help, "]?" for Pkg help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 1.0.4 (2019-05-16)
 _/ |\__'_|_|_|\__'_|  |  Official https://julialang.org/ release
|__/                   |

julia>
```

Figure 3 – The Julia environment

b) Include pfem4ec.jl.

```
julia> include("pfem4ec.jl")
```

c) Run the pfem4ec function.

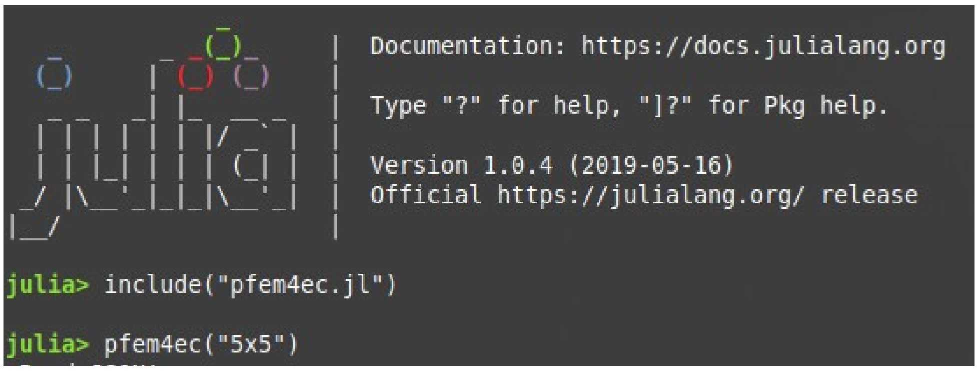
Obs.: The first call will compile and run, all subsequent calls will just run, considerably reducing overall elapsed time to obtain results.

```
julia> pfem4ec("model_filename")
```

To obtain time metrics for each time pfem4ec is run, simply add the Julia macro @time before the call presented above.

```
julia> @time pfem4ec("model_filename")
```

Figure 4 shows steps b) and c), calling an analysis for the 5x5 model presented in Figure 1.

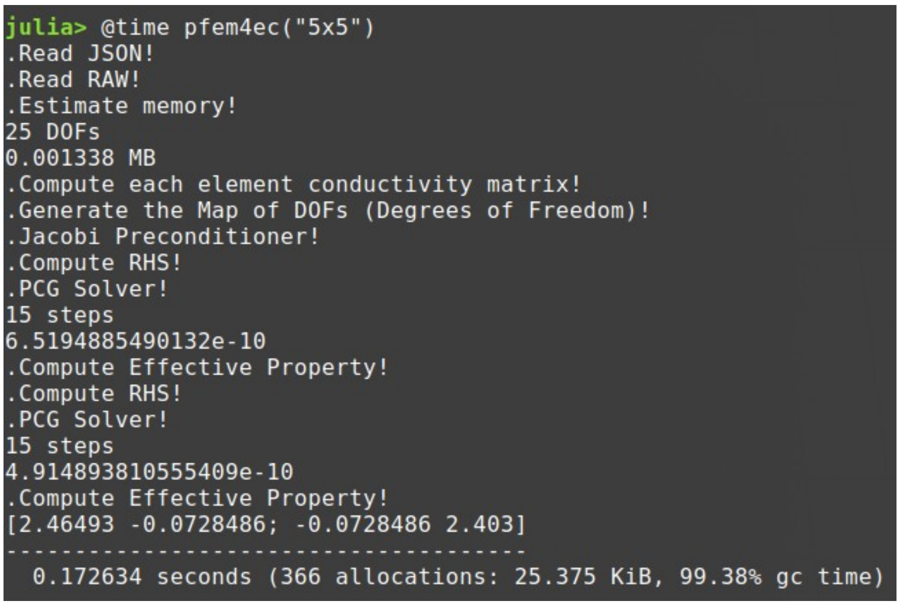


```
               _
   _       _ _(_)_     |  Documentation: https://docs.julialang.org
  (_)     | (_) (_)    |
   _ _   _| |_  __ _   |  Type "?" for help, "]?" for Pkg help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 1.0.4 (2019-05-16)
 _/ |\__'_|_|_|\__'_|  |  Official https://julialang.org/ release
|__/                   |

julia> include("pfem4ec.jl")

julia> pfem4ec("5x5")
```

Figure 4 – Including and running pfem4ec in the Julia environment

3) Output

For now, pfem4ec.jl output consists on feedback of the analysis that is printed on the terminal that called the program. Users are provided with an estimation of allocated memory, convergence metrics for the PCG numerical method, a stamp for each function called by pfem4ec along the process and, finally, the resulting effective homogenized conductivity matrix. Figure 5 shows the output for the 5x5 example model adopted in this guide.



```
julia> @time pfem4ec("5x5")
.Read JSON!
.Read RAW!
.Estimate memory!
25 DOFs
0.001338 MB
.Compute each element conductivity matrix!
.Generate the Map of DOFs (Degrees of Freedom)!
.Jacobi Preconditioner!
.Compute RHS!
.PCG Solver!
15 steps
6.5194885490132e-10
.Compute Effective Property!
.Compute RHS!
.PCG Solver!
15 steps
4.914893810555409e-10
.Compute Effective Property!
[2.46493 -0.0728486; -0.0728486 2.403]
--------------------------------------
  0.172634 seconds (366 allocations: 25.375 KiB, 99.38% gc time)
```

Figure 5 – pfem4ec output for analysis on 5x5 model

Coupling pfem4ec.jl with pytomoviewer

pfem4ec.jl is a powerful tool to run homogenization analysis on the microscale of heterogeneous materials, from a binary image. In a practical sense, it would be good to be able to get proper input files for pfem4ec.jl from a generic image, or one obtained via micro-computed tomography, for example. A common file format for images of that sort is TIFF. With that in mind, the authors have also made available pytomoviewer, a tool for the visualization of TIFF images (and stacks of them, for 3D models) and the conversion into RAW and JSON files ready to be used as input for pfem4ec.jl. In this section, it will be briefly shown how to go about this process, using the same 5x5 model that presented as an example in the previous section of this guide. To learn more about pytomoviewer, please refer to https://github.com/LCC-UFF/pytomoviewer.

1) Launch pytomoviewer with Python3.

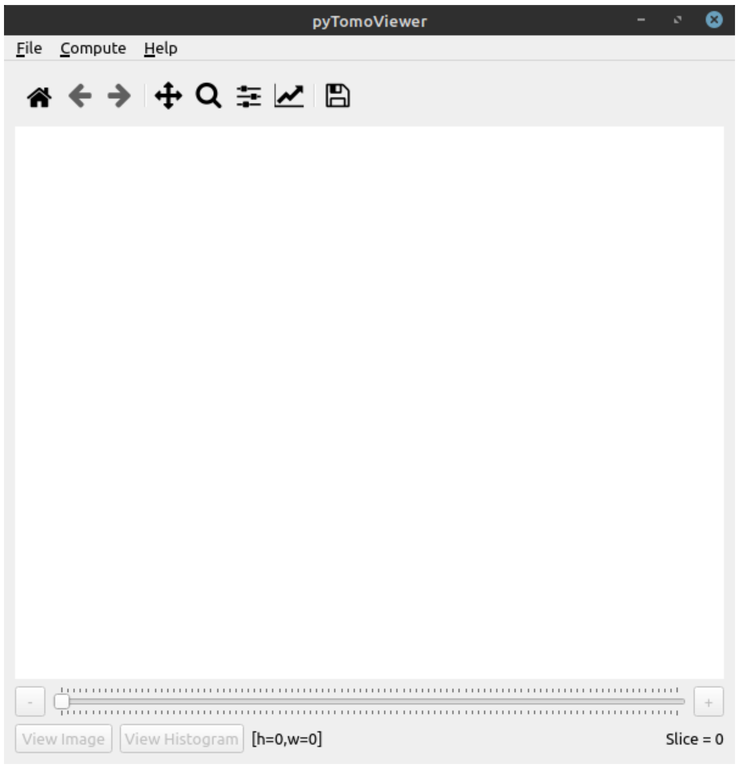


Figure 6 – pytomoviewer

2) Open the target TIFF file (or stack of files).

On pytomoviewer: File → Open.

On the file explorer: Select target TIFF file (“5x5.tif”, on this case) → Open.

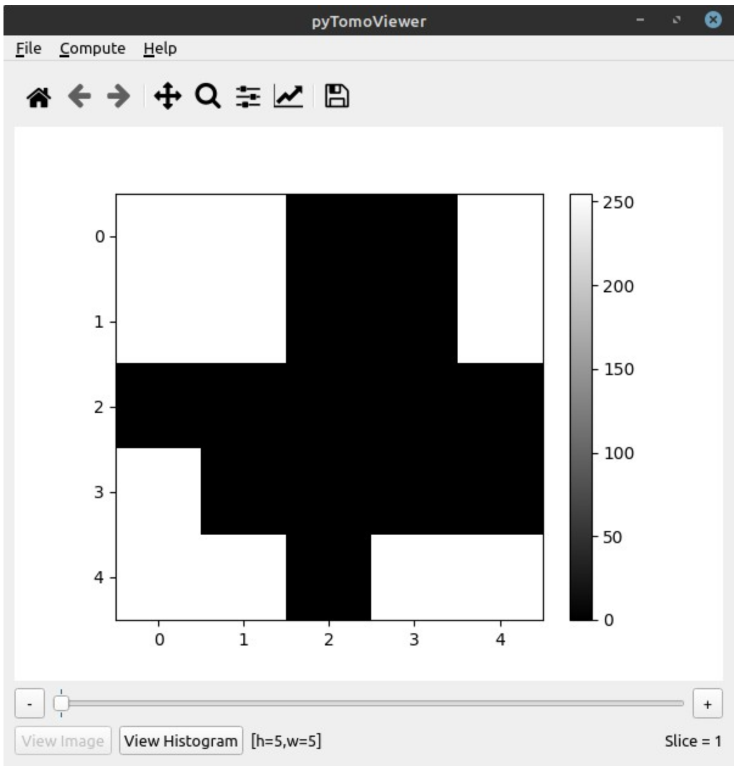


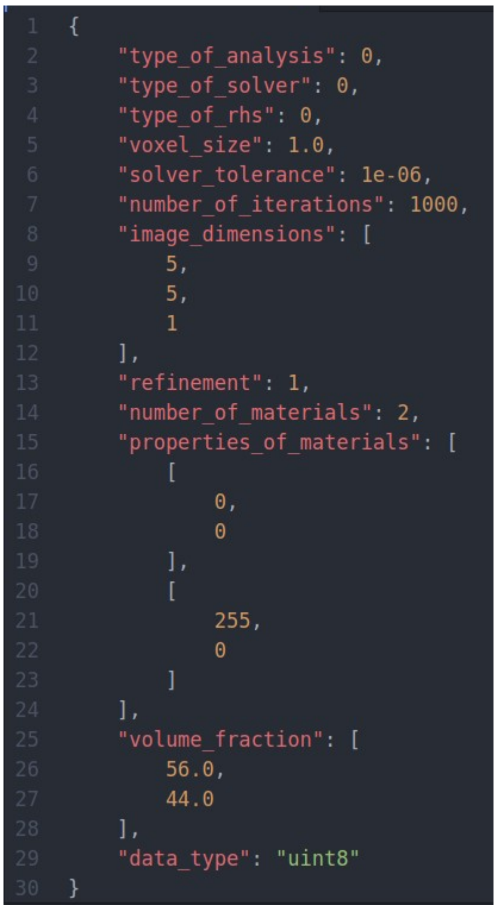
Figure 7 – 5x5 model visualization, from a TIFF file

3) Export RAW and JSON files.

On pytomoviewer: File → Export.
On the file explorer: Select target directory for RAW a JSON files → Save.
Obs.: It will also be created a Neutral file (.nf), with content analogous to the JSON file. This serves as an input for programs other than pfem4ec.jl.

4) Open the JSON file to specify analysis parameters and material properties.

The JSON file generated by pytomoviewer will have default input parameters. It is important that users verify the contents of the file (it can be viewed and modified via any text editor) to make sure that the analysis will be performed properly. <u>It is essential to set the value of conductivity for each material of the microscale, as pytomoviewer will only be able to tell materials apart by color, but will set their physical properties with a null value</u>, as it can be seen on Figure 8. So the value on the second position of each array on properties_of_materials must be updated accordingly.



```
{
    "type_of_analysis": 0,
    "type_of_solver": 0,
    "type_of_rhs": 0,
    "voxel_size": 1.0,
    "solver_tolerance": 1e-06,
    "number_of_iterations": 1000,
    "image_dimensions": [
        5,
        5,
        1
    ],
    "refinement": 1,
    "number_of_materials": 2,
    "properties_of_materials": [
        [
            0,
            0
        ],
        [
            255,
            0
        ]
    ],
    "volume_fraction": [
        56.0,
        44.0
    ],
    "data_type": "uint8"
}
```

Figure 8 – 5x5 model JSON file, provided by pytomoviewer

5) Run pfem4ec.jl using the RAW and JSON files as input, as shown in the previous section of this guide.