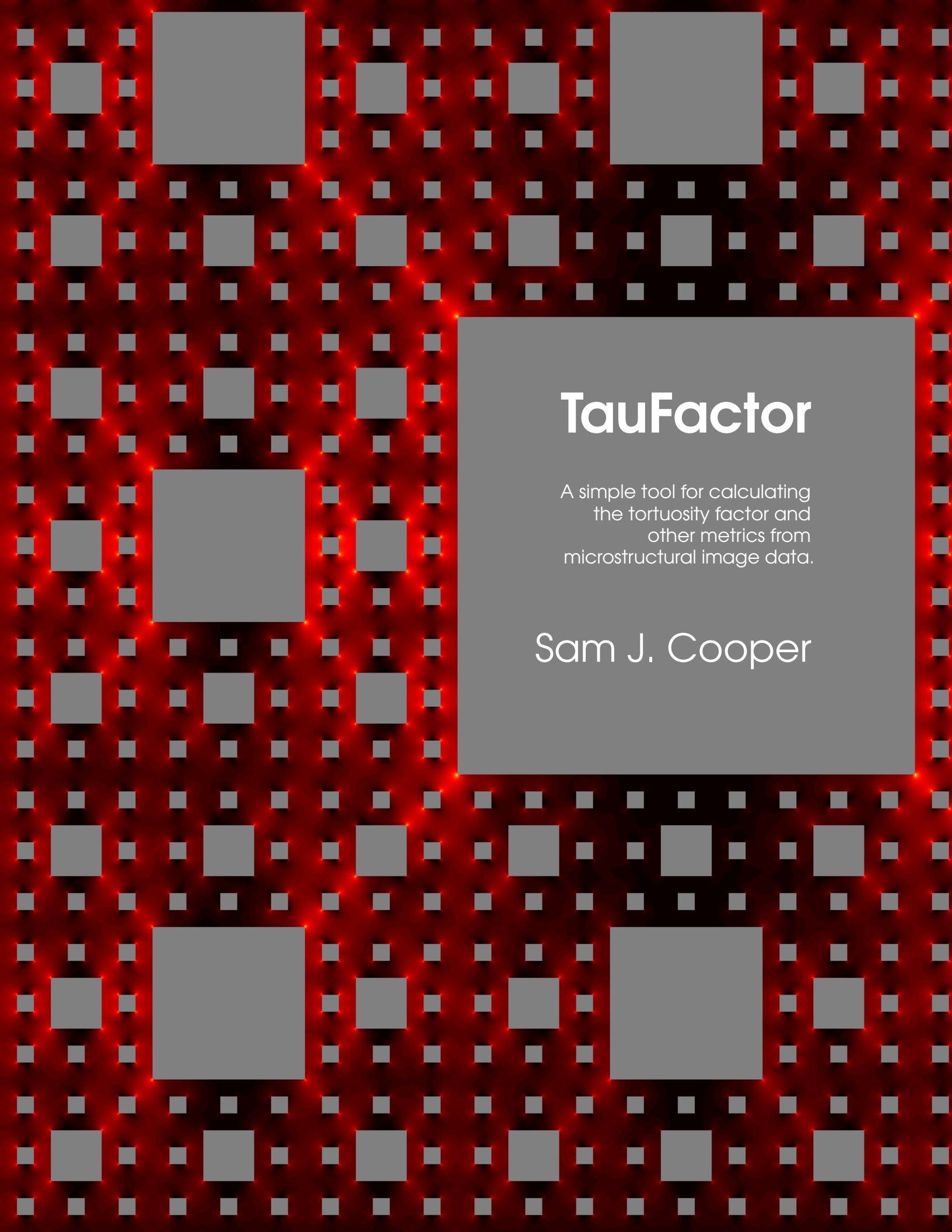
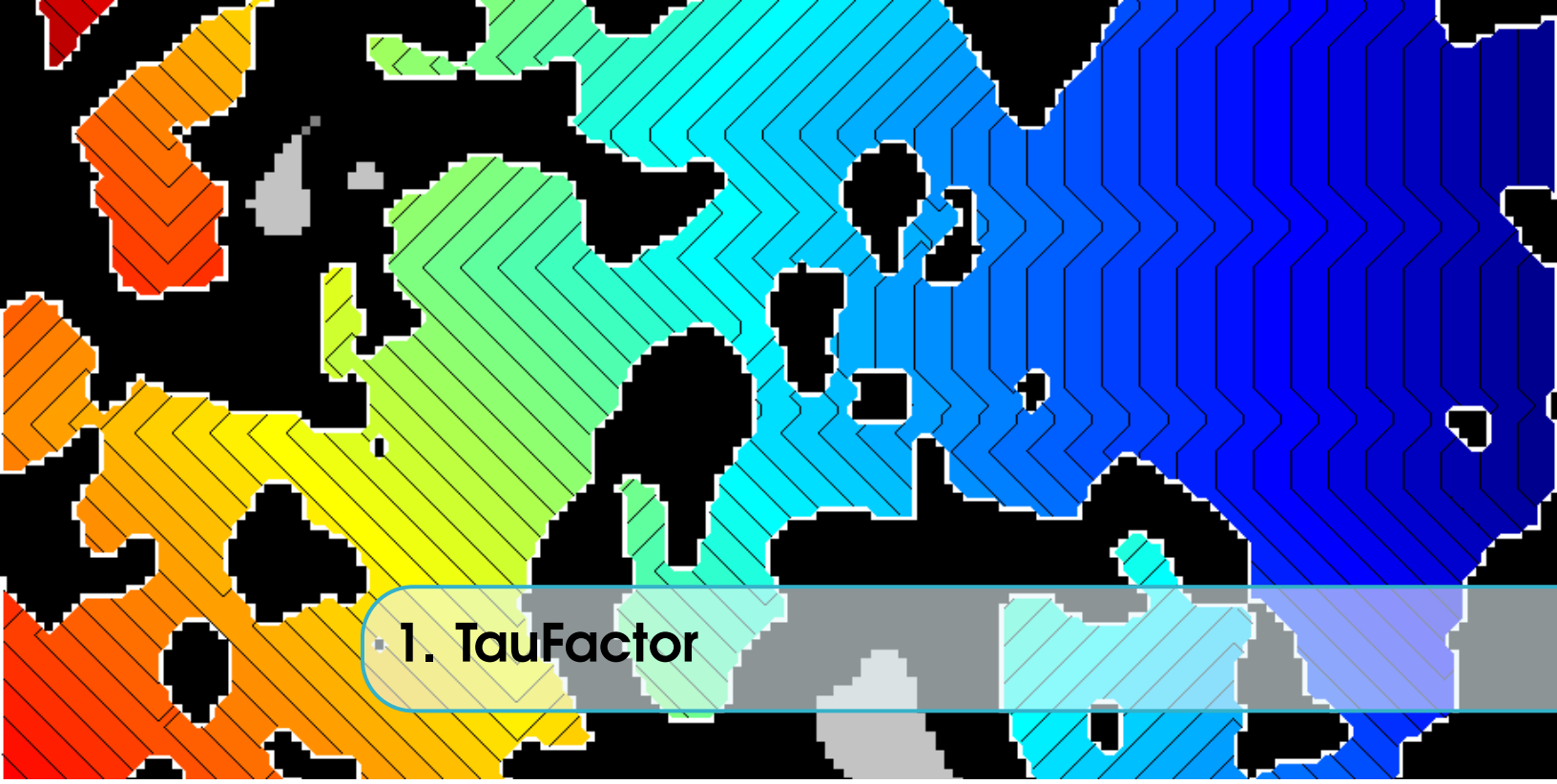


TauFactor

A simple tool for calculating the tortuosity factor and other metrics from microstructural image data.

Sam J. Cooper



1. TauFactor

1.1 Getting Started

TauFactor (available from sourceforge.net/projects/taufactor) is a tool developed for the analysis of microstructural image data, including fast numerical approximation of the tortuosity factor. It was written in the MatLab language and is suitable to run on any computer with access to this platform (*i.e.* it does not required a large multi-core workstation to run).

To install the app, simply double click on the `TauFactor.mlappinstall` file. This should cause MatLab to open and then prompt the user with the following message:

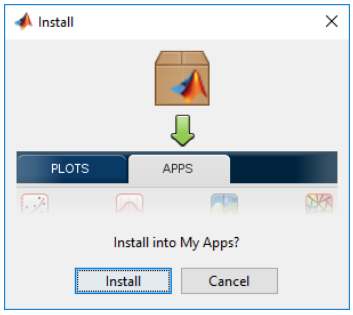


Figure 1.1: MatLab install pop-up.

After pressing *Install*, the app can then be opened by clicking the relevant icon in the app tab at the top of the MatLab window shown in fig. 1.2.

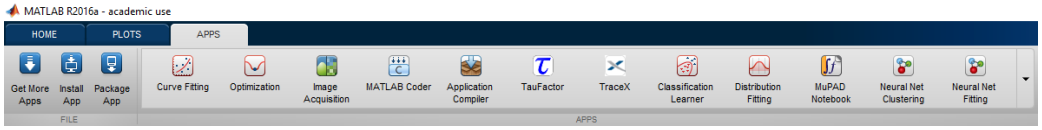
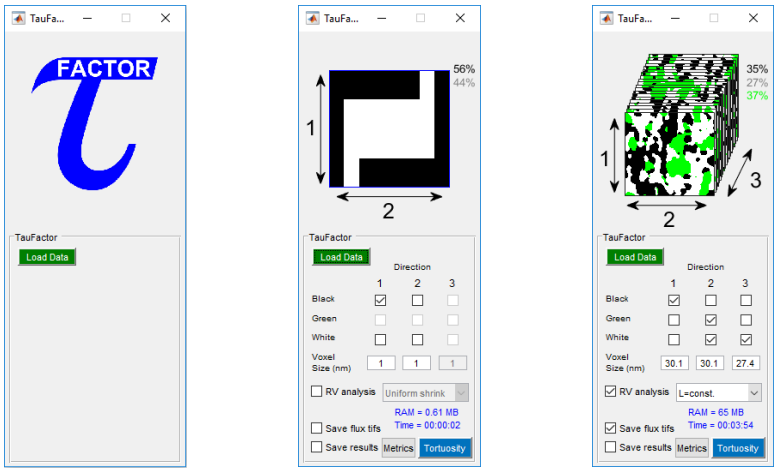


Figure 1.2: TauFactor icon in the MatLab app docking tab.

1.2 Using the app

When the app is first opened, it should look like fig. 1.3(a), with only the green *Load Data* button visible.



(a) (b) (c)

Figure 1.3: The window view when the *TauFactor* application is (a) first opened, (b) after a simple 2D dataset with 2 phases has been loaded and (c) after a 3D dataset with 3 phases has been loaded.

Pressing this button will open a file selection dialogue box for the user to find their data. For 3D datasets, the file must be a 3D `tif` file; however, it is also possible to load 2D image files as either `tif` or `png`. This flexibility was added so that the user can rapidly draw microstructures in simple programs such as MS Paint and then load them directly into *TauFactor*, as it is felt that this will encourage users to play around and get a better intuitive understanding of the tortuosity factor.

Importing multiple datasets is also permitted, in which case identical operations will be performed on each as a batch job.

Once the data has been loaded, it will be displayed at the top of the window (see fig. 1.3(b) & (c)) and more options will appear in the lower half of the window. The grid of 9 check boxes are used to select phases and directions for simulation. The three rows "Black", "Green" and "White" refer to the colour scheme in the image and the three columns "1", "2" and "3" refer to the directions. Check boxes will only be active for directions/phases that are available in the currently loaded data (*e.g.* a 2D, two phase data set will only have four boxes active).

Below this grid there are three edit boxes aligned with the three direction columns. These allow the user to input the size (in nm) of the voxels, which feeds in to the simulation.

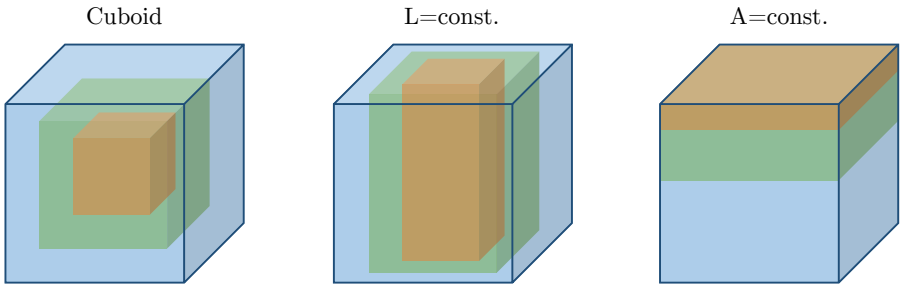


Figure 1.4: Illustration of the three methods of representative volume analysis. In each case, three volume steps are show, whereas in reality 20 steps are taken of constant volume increment.

Ticking the *RV analysis* check box activates the adjacent mode selector. This allows the user to select between four different representative volume analysis (RVA) study options, as illustrated in fig. 1.4 (the fourth is the same as the third, but starting from the base (*i.e.* from the highest index face)). The RVA will be performed separately for all phases/directions currently selected. The four RVA methods change orientation according to the direction specified for the calculation.

Pressing the *Tortuosity* button will calculate the tortuosity factor in the directions and phases specified using the check boxes. Pressing the *Metrics* button will perform a representative volume analysis on the volume fractions, surface areas and triple phase boundary densities (if the volume contains 3 phases). The results of these two studies are then returned in the reports shown in fig. 1.6. If the *Save Results* check box is ticked, the report will be saved in the same folder as the dataset, with an appropriate name.

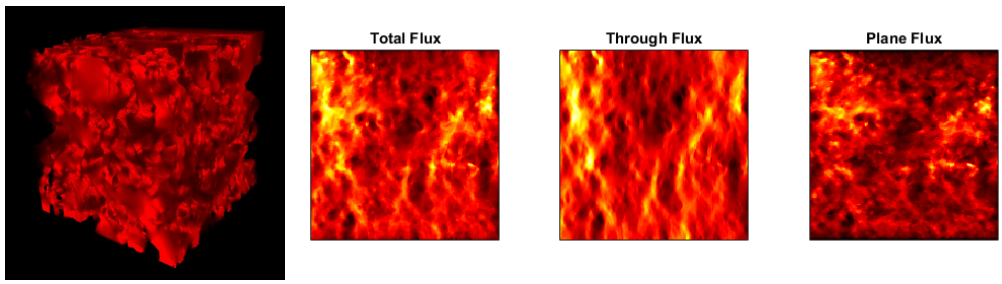


Figure 1.5: Map of the normalised flux resulting from a tortuosity factor simulation, as well as 3 projections through the various flux maps export options.

The *Save Flux tifs* check box will save maps of the flux (based on local gradients) in the same folder as the dataset, with an appropriate name, as well as two decompositions of the flux into the "in-plane" and "through-plane" directions. These flux maps can be visualised in various packages (*e.g.* ImageJ, see fig. 1.5) and used to find bottle necks.

Just above the *Tortuosity* button, the approximate memory and time requirements are given for the tortuosity simulations currently specified, based on a single i7 @2.9 GHz processor core.

1.2.1 Inline Calling

TauFactor can also be called as an inline function within another script or through the command window.

```
Results = TauFactor('InLine',FindTau,FindMetrics,RVAmode,Net,PhaDir,VoxDims)
```

The `'InLine'` command prepares the script for an inline operation. `FindTau` and `FindMetrics` should both receive a binary input (0 or 1) to indicate the user's preference for these options. `Net` should be an array containing the 3D (or 2D) voxelised geometry data, where the phases should ideally be labelled as 0, 1 or 2 (although if this is not done, *TauFactor* will attempt to normalise). `PhaDir` is a 3×3 array of binaries that correspond to the checkbox grid shown in the GUI. The rows are the phases, "Black", "Green" and "White" (for 2 phase images, the "Green" row should be all zeros); and the columns are the directions 1, 2 and 3. `VoxDims` are the dimensions, in nanometres, of each voxels in directions 1, 2 and 3 respectively. `RVAmode` should be feed an integer from 0 to 4, where 0 does not call an RVA, but 1 to 4 correspond to the methods listed in the RVA dropdown box in the GUI (1='`Uniform shrink`', 2='`L=const.`', 3='`A=const.(from top)`', 4='`A=const.(from base)`').

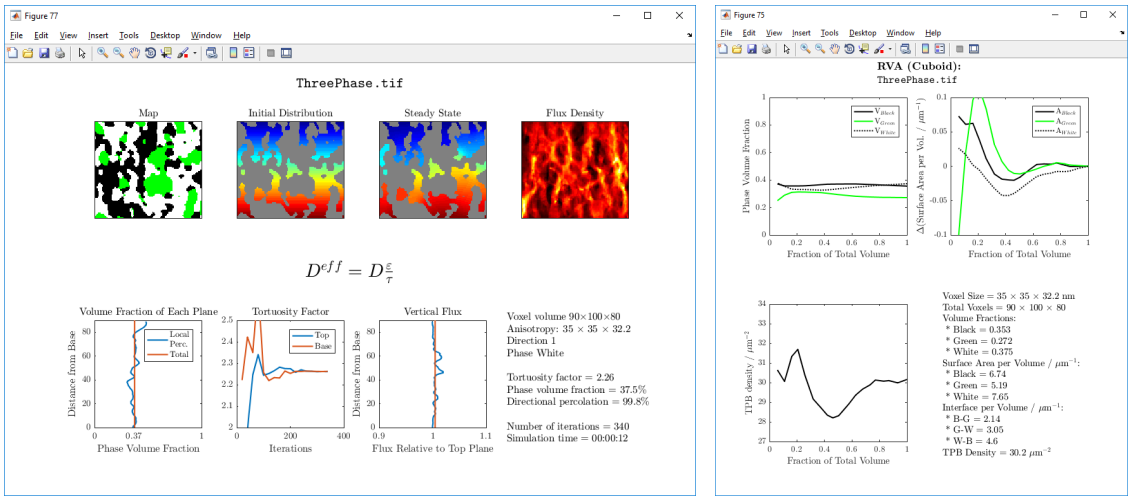
MatLab stores applications in a folder called "Add-Ons" within the main MATLAB folder. It is necessary to add this folder to your main MatLab path. This can be done by browsing to the "Add-Ons" folder within the Matlab environment, right-clicking and then selecting "Add to Path" followed by "Selected Folders and Subfolders". It should now be possible to run the example below by directly copying and pasting it into your command window and pressing return:

```
for i= 1:3
     Results(i)=TauFactor('InLine', 1, 1, 0,rand(50,50,50)>(0.4+i*0.05),...
     [1 0 0;0 0 0;0 0 0], [1 1 1]);
end
disp(['Tau factor at 50% porosity = ', num2str(Results(2).Tau_B1.Tau)])
```

This should begin a three step study on the tortuosity factor and other metrics of randomly generate 2 phase microstructures with different volume fractions. The analysis is being performed in direction 1 of the black ('0') phase. This will result in the creation of a structured variable in the workspace (`Results`). Once this study is complete, the tortuosity factor of the second volume generated will be displayed in a message in the command window.

It is good practice to use structured variables to store this kind of multifaceted data, but they can appear mysterious to the uninitiated. However, they are simply variables within variables (possibly within variables) and to navigate this structure, simple place a "`.`" between the address at each level. For example, to access the surface area density of the with 45% porosity, call "`Results(1).Metrics.SurfAreaDens_over_um`" (copy and paste into your command window to test).

The naming structure of these variables is intended to follow a logical pattern, but this would be tedious to explain and difficult to remember. The best way is to have a go yourself before setting a large study to run. Remember that so see a list of the next level of addresses stored in a structured variable, simply type the path to the level above.



(a) (b)

Figure 1.6: Reports generated by *TauFactor* for (a) a tortuosity factor calculation and (b) a representative volume study of various other metrics including volume fractions, surface area and triple phase boundary densities on some example three phase microstructural data.

1.3 Notes on implementation

1.3.1 Tortuosity Factor

The main journal article describing the function of *TauFactor* offers a description of the checkerboarding, over-relaxation and vectorisation techniques used to accelerate the convergence of the simulation. The use of the *ghost node* concept is also described. However, the measurement of convergence was only mentioned briefly.

When the data is initially loaded, the maximum number of permissible iterations is calculated based on the longest dimension of the volume and the degree of anisotropy. This value is then used to calculate an appropriate iteration interval at which convergence is checked. When the simulation starts, 5 of these intervals are allowed to pass before the first check.

The current approximation for the tortuosity factor of the system is easy to check only at the two faces where the Dirichlet boundary conditions are applied. As such, it is these two values that are used to asses convergence. The current values are compared to themselves in previous iterations (to assess stability) as well as to each other (to assess consistency). It was found that neither of these two approaches were able to reliably detect convergences alone, but that the combination was effective. Although the system does have an exact solution, finite convergence tolerances are used to avoid the unnecessary (and meaningless) pursuit of high accuracy values. It can be clearly seen from the convergence graph generated in the live report that the final value has converged to at least 1 decimal place. The significant and compounded uncertainties of the tomography and segmentation processes should discourage authors from reporting tortuosity factors to beyond 2 significant figures.

1.3.2 Metrics

When performing RVA on the Metrics, if the RVA method specified is effected by orientation, this can be selected by ticking a corresponding checkbox (of any phase) in the GUI checkbox matrix. Also, for both the surface area and TPB density studies, the outer most layer of voxels was not

included in the calculation, although their values were used to label the next layer in. This is because it is not possible to determine for all of the faces/edges in the outer layer should be labelled as interfaces/TPBs without making some additional assumptions.

Volume Fractions

The phase volume fractions are calculated simply by summing the voxels containing each phase separately and then dividing this number by the size of the volume considered.

Surface Areas

In the case of 2 phase data, the surface area density is calculated by comparing each voxel to its 6 face-adjacent neighbours to see if they are of the same phase. For anisotropic voxels, it is necessary to consider the three directions separately, as the faces will have differing areas. The sum of these 6 potential interfacial areas on each voxel is then divided by 2. This is due to the fact that each interface is common to two voxels, which would otherwise result in double counting. This approach has the additional benefit that areas on the outer surface of the labelled volume effectively have a weighting of 0.5 applied to them. This is important to ensure that the density of faces considered in the density calculation scales linearly with volume. Consider that the total number of faces, F_{tot}, in a cubic volume, $V = n^3$ is

$$F_{\text{tot}} = 3n^2(n+1) \tag{1.1}$$

However, including all of these in the area density calculation would cause this value to have a misleading scale factor of $\frac{3}{n}$, which would obscure the user from determining whether or not a volume should be considered representative. Weighting the faces on the outer surface of the region by a factor of 0.5 yields an effective face number of

$$F_{\text{eff}} = F_{\text{tot}} - \left[0.5 \times (6n^2)\right] = 3n^3 \tag{1.2}$$

In the case of three phase data, this operation needs simply to be repeated for each phase separately, by first creating new binary volumes labelling one phase at a time. Although other approaches are possible for making this correction (such as calculating an effective volume for the number of faces considered), the method described above is highly conducive to a rapid RVA study.

Triple Phase Boundaries

TauFactor is able to count TPBs very quickly by first relabelling the three phases with the prime numbers 2, 3 and 5, stored as an unsigned 8-bit integer array. Next, matrix operations are used to find the product of the four voxels contacting each edge (the three orientations are again considered separately to account for anisotropy). Finally, only if this product is a multiple of 30 is the edge considered to be a TPB (N.B. $5 \times 5 \times 5 \times 3 = 375$ and $5 \times 5 \times 5 \times 5 = 625$ are both greater than the maximum value allowed by the class (255); however, MatLab defaults to this maximum value, which itself is not a multiple of 30, so no problems arise).

As each voxel has 12 edges and each of these edges is in contact with 4 voxels, so the sum of the TPB length found in each voxel is divided by 4 to avoid quadruple counting. Once again, this allows us to compensate for the over counting of TPBs in the density calculation, by acting as an effective

weighting factor (0.5 at the sides and 0.25 at the corners). Consider that the total number of edges, E_{tot}, in a cubic volume, $V = n^3$ is

$$E_{\text{tot}} = 3n(n+1)^2 \tag{1.3}$$

However, including all of these in the TPB density calculation would cause this value to have a misleading scale factor of $\frac{6n+3}{n^2}$, which would obscure the user from determining whether or not a volume should be considered representative. Weighting the edges on the outer surface of the region by a factor of 0.5 and those at the corners by 0.25 yields an effective edge number of

$$E_{\text{eff}} = E_{\text{tot}} - [0.5 \times 12n(n-1) + (1-0.25) \times 12n] = 3n^3 \tag{1.4}$$

As with the area analysis, this TPB approach is well suited to a rapid representative volume study, which is why it was chosen over other equally valid methods.

The effectiveness of this scaling approach can be seen by calculating the *Metrics* of a large randomly generated 3 phase volume, as the resulting RVA graphs should all contain flat lines. *TauFactor* has a random microstructure generator built in to it, which is activated in the following way: When the app is first opened (*i.e.* before any data have been loaded), click 4 times on the word `TauFactor` that appears halfway down the window. The volume generated should contain an equal proportion of each phase (*i.e.* 33% of the black phase, 33% of the green phase and 33% of the white phase), for which it is possible to analytically determine the expected densities of interfacial area and TPBs.

Each face in a volume is in contact with two voxels. In a 3 phase volume, there are 9 possible arrangements of the phases in these two voxels. If the two phases are not the same, the face is considered to be an interface and this is the case in 6 of the 9 arrangements. Therefore, the expected density of interfaces in a random 3 phase volume is

$$\begin{aligned}\rho_{\text{Interface}}^{\text{Random}} &= \frac{\#\text{Faces} \times \text{P}(\text{Faces} = \text{Interface})}{\text{Volume}} \\ &= \frac{3n^3 \times \frac{6}{9}}{n^3} = 2\end{aligned}$$

Similarly, each edge in a volume is in contact with four voxels. In a 3 phase volume, there are 81 possible arrangements of the phases in these four voxels. If all three phases are present in these four voxels, it is considered a TPB and this is the case in 36 of the 81 arrangements. Therefore, the expected density of TPBs in a random 3 phase volume is

$$\begin{aligned}\rho_{\text{TPB}}^{\text{Random}} &= \frac{\#\text{Edges} \times \text{P}(\text{Edge} = \text{TPB})}{\text{Volume}} \\ &= \frac{3n^3 \times \frac{36}{81}}{n^3} = 1.\dot{3}\end{aligned}$$

1.4 Further thoughts

1.4.1 Intuitive tortuosity factors

It is hoped that the speed and simplicity of *TauFactor* will allow users to develop better intuition about the tortuosity factor. Simple 2D flow networks can be drawn (in MS Paint for example) and used to test hypotheses about idealised geometry.

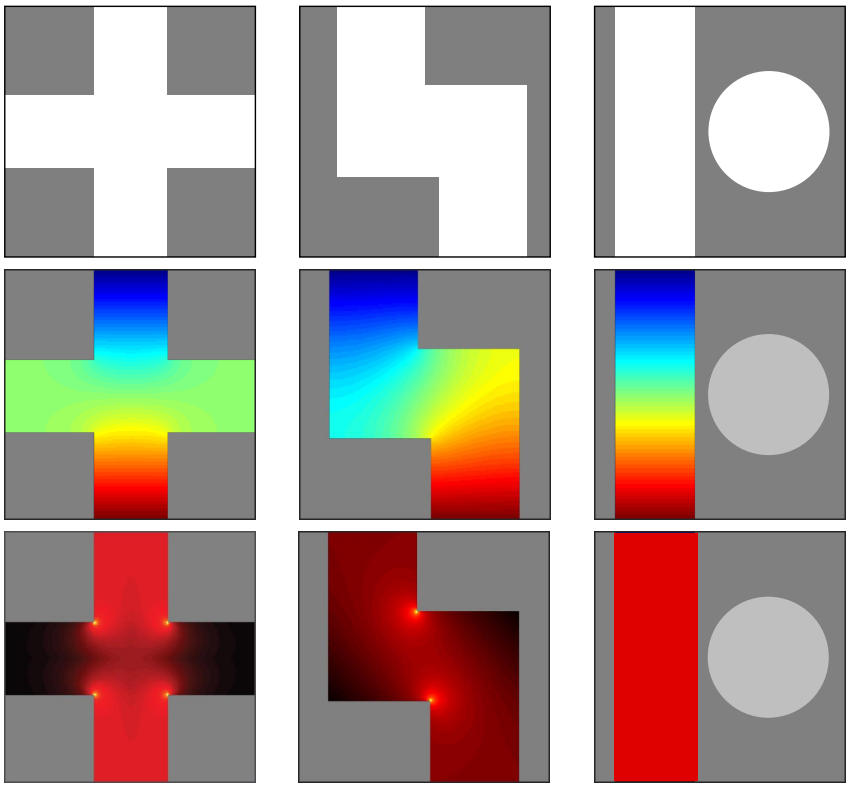


Figure 1.7: Three simple 2D microstructures each with the same conductive volume fraction (50%) and the same tortuosity factor (≈ 1.57). The top row shows the distribution of the conductive phase; the second row shows the distribution of the potential between the two Dirichlet boundaries; and the last row shows the normalised local flux through each voxel.

For example, it's clear from fig. 1.7 that although the three microstructures respond the same way in one direction at steady-state they are very different in other respects. Clearly, only the first structure would exhibit any lateral conduction (if it were periodic) and the last structure has a cut-off region and comparatively low surface area.

It may be possible to detect some differences of interest with an impedance type simulation, where the microstructure is exposed to an oscillating stimulation at one of its boundaries across a range of frequencies. This "tortuous impedance", defined in the frequency domain, has already been developed by the authors and will be presented in a future publication.

1.4.2 Voxelisation

TauFactor reports surface areas and TPB densities calculated directly from the features of the cuboid voxels. These values are likely to be different from the true values, but without some knowledge of the sample and imaging resolution, it is not even possible to say whether an under- or overestimation is expected.

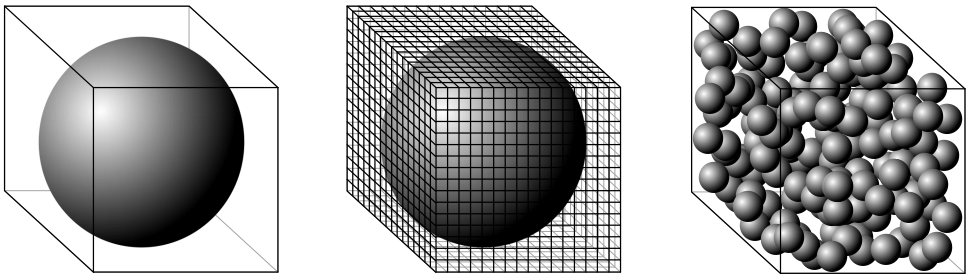


Figure 1.8: Illustration of three limiting cases of voxelisation induce surface area error.

The three limiting cases shown in fig. 1.8 are all generated under the assumption of "perfect tomography", whereby a voxel containing ≥ 0.5 of a phase is modelled as entirely consisting of that phase.

The first example shows the limiting case for over estimation of surface area. The single voxel contains a sphere of phase 1, whilst the remained (as well as all adjacent voxels) is made up of phase 2. It can be shown that for this sphere to fill over half the voxel, it will require a minimum radius of

$$r_{min} \geq \sqrt[3]{\frac{3}{8\pi}}$$

This means that the limiting case of area *overestimation* is

$$E_{over} = \frac{6}{4\pi r^2} = \frac{6}{\sqrt[3]{9\pi}} \approx 1.97$$

In the second case of interest shown in fig. 1.8, a sphere is contained in a bounding cube, subdivided into n^3 voxels. It can be shown that in the limit, the error due to voxelisation is

$$\lim_{n\to\infty}(E_{voxelise}) = \frac{24r^2 \int_0^{\pi/2} \sin^2\theta \mathrm{d}\theta}{4\pi r^2} = 1.5$$

Lastly, for a voxel containing a large number, m, of small spheres (or any alternative complex geometry), the voxelised rendering will greatly *underestimate* the true surface area. This case would represent the scenario of inappropriately coarse resolution of the imaging technique used and is felt by the author to be the more common scenario in real tomography.

$$\lim_{m\to\infty}(E_{under}) = \lim_{m\to\infty}\left(\frac{6}{m4\pi r_{small}^2}\right) = \lim_{m\to\infty}\left(\sqrt[3]{\frac{24}{m\pi}}\right) = 0$$

where,

$$r_{small} \geq \sqrt[3]{\frac{1}{m}\frac{3}{8\pi}}$$

Clearly, similar arguments can be made about the uncertainty in TPB measurements, although the limiting case of *overestimation* becomes infinite by imagining that, although present, the three phases to simply never make a triple point contact in reality.

It is therefore crucial that users of this app carefully consider to what extent the voxelised values represent the reality. To this end it should be considered bad practice to quote these values to a high level of precision (2 significant figures is plenty) or to place significance on minor differences observed between samples.

1.4.3 Citation

If you would like to cite this work in an academic study, please contact the author at sjc08@ic.ac.uk or camsooper@gmail.com.

The main article supporting this work was published in the Elsevier journal *SoftwareX* in 2016:

S.J. Cooper, A. Bertei, P.R. Shearing, J.A. Kilner, N.P. Brandon. TauFactor: An open-source application for calculating tortuosity factors from tomographic data. SoftwareX (2016)