Effective Diffusivity Algorithm Documentation

Andre Adam

Last Updated: June 7, 2024

Contents

1 Introduction

This is documentation on how to use the algorithm for predicting the effective diffusivity of a system. It assumes a steady-state solutions exists, and a trace amount of the diffusing species. If there are any questions at any moment, just contact one of the main contributors. It takes one 2D grayscale image as the input.

The top and bottom boundaries are impenetrable, while the left and right boundaries of the 2D domain are set concentrations. There are several different uses for this code, and we will look at all of them. After reading this document, you will be ready to draft your own input file and use this code.

2 Requirements

To run the code, the following files are necessary in the same folder:

```
Deff2D.cu
Deff2D.cuh
stb_image.h
input.txt
AnyImage.jpg
```

Currently the code only runs on NVIDIA GPUs (no CPU based computation). The code has been tested on CUDA Capability 8.6 or newer. That doesn't mean it won't work for older Compute Capabilities, it just means this is all we had available to test.

The input file must be named input.txt, and in this document we will go over a sample input file, specifications, intricacies, and how to best use it.

2.1 Compiling

Compilation of the code is only done once, but it must be done. If the code is modified, then the user must compile again. From the command line the following command will compile the .cu file into a executable:

```
nvcc Perm2D.cu
```

It assumes the command is running in the same folder as the other files. A simple double click on the .exe file should run the code with the specified inputs. Other compiler options work, this is just the simplest working case.

2.2 Image Requirements

The image must be a single channel (**NOT** RGB) grayscale .jpg image for the algorithm to correctly run. If the image has any number of channels other than 1, the code returns indicating an error message telling the user to set the image to 1 channel. The file *stb_image.h*, which was sourced from open source project, is responsible for reading the specified image.

The code can currently handle 2 phases and 3 phases. For encoding image in 2 phases, simply binarize it, and the code will assume white is solid and black is void space. For the 3-phase code, we maintain solid as white and void (gas) as black, and add fluid (generally opaque) as gray. When you process images before feeding them into the algorithm, please set white and black to 255 and 0, and keep gray close to 150, all in the grayscale.

3 Input.txt

In this section we take a look at a sample .txt file, and how to use it. The first line of the file is optional, but the name of the file must be "input.txt", verbatim.

It must also be noted that the names have to be exactly as shown below. The code does differentiate from lower and uppercase letters. The order in which the arguments are entered is not important.

As a final note, there must always be a space after the ":" and the numbers or filenames.

```
Input File:
Phases: 3
Ds: 0
Df: 1
Dg: 1237500
MeshAmpX: 1
MeshAmpY: 1
InputName: 00042.jpg
CR: 1
CL: 0
OutputName: singleTest.csv
printCMap: 1
CMapName: CMAP_00042.csv
Convergence: 1e-5
MaxIter: 5e6
Verbose: 1
RunBatch: 0
NumImages: 500
```

3.1 Phases

This input lets the user decide if they are running the code on 2 or 3 phases. Any other options will return an error.

3.2 Diffusivity

The first three options are the diffusion coefficients of the trace species in solid, fluid, and gas, respectively. They are named Ds, Df, and Dg. If there is an impermeable phase, usually Ds is set to 0. Also whether you are running a 2 phase model or 3 phase model, the value of Df should be 1 as a best practice. This comes from the non-dimensionalization of the mass diffusion equation. If we have real diffusion coefficients with units of cm^2/s, labelled d_s^*, d_f^* and d_g^*, we should extract our Ds, Df, and Dg as follows:

$$\begin{gathered} Ds = \frac{d_s^*}{d_f^*} \\ Df = \frac{d_f^*}{d_f^*} = 1 \\ Dg = \frac{d_g^*}{d_f^*} \end{gathered} \tag{1}$$

3.3 MeshAmp

These numbers will be interpreted as integers and cannot be negative. They determine how much the mesh will be increased from the pixels of the original image.

Example: If a 100×100 pixels image is used as input, and MeshAmpX is set to 3, and MeshAmpY is set to 1, then the domain being evaluated will be 300 cells wide by 100 cells high.

3.4 InputName

Verbatim name for the input image. Must be single channel grayscale .jpg image.

3.5 CL and CR

Sets the concentration on the left and right boundary. Best practice is to always have these as dimensionless and between 0 and 1. If we have real concentrations of trace species "A" at the left and right boundaries, $c_{A,L}$ and $c_{A,R}$, then we can achieve CL and CR by defining a C_x as follows:

$$C_x = \frac{c_{A,x} - c_{A,L}}{c_{A,R} - c_{A,L}} \tag{2}$$

From equation (2), it is easy to see how CL and CR are 0 and 1 respectively.

3.6 OutputName

String with the name of the file to save the output of the code. If the code is in batch mode, the output file will save all of the output. In batch mode, all of the output is saved at the end, so if the code is interrupted, all progress is lost. I am working on a better option.

3.7 Print Concentration Map

We are solving for the concentration map of trace species "A" throughout the domain, as a function of the normalized dimensionless diffusion coefficients. If you want to print the map of what the diffusion concentration map looks like, set this flag equal to 1. Setting it equal to 0 will not print the map. Since this is a boolean, I am not sure what happens if you enter a value that is not 1 or 0.

If the flag is set to 1, then it prints the concentration map with the name provided by the user in the option CMapName. It has to be a .csv or .txt filename.

3.8 Convergence Criteria

Currently there is only one convergence criteria. If there are better ideas for convergence, I can always add something different and add the option to choose what the convergence criteria is.

The current convergence criteria calculates the d_{eff} every 10,000 iterations. The change in d_{eff} is calculated as follows:

$$\text{Change} = \frac{|d_{eff,old} - d_{eff,new}|}{d_{eff,old}} \tag{3}$$

3.9 MaxIter

Interpreted as a integer, this is the maximum number of iterations before the code exits without reaching the convergence criteria.

3.10 Verbose

The option verbose is a flag that controls printing to the console. With verbose set to 1, several things are printed to the command line/console while the code is running. If verbose is set to 0, then the code runs without printing anything. The output files are separate from this, and are not affected by this flag.

While for a single image this has a negligible impact on speed, for a batch of images on HPC or when saving the command line output, it can be non-negligible in terms of speed and storage.

3.11 RunBatch

This is a flag (0 for false, 1 for true) to run a batch of images. If this flag is set to 1, the program will take input from the NumImages option to seek the number of images. Naming and finding the appropriate files can be difficult, therefore there is a standard naming required for using this option.

Whatever NumImages is, the images in the folder are to be named using consecutive 5 digit numbers starting at "00000.jpg". There are always leading zeroes for small numbers. If, for example, there are 500 images in the folder, they will start at "00000.jpg" and be consecutively labelled up to "00499.jpg". This way the program will be able to find and evaluate all the structures.

This method also limits the number of images for a single batch to 100,000 images. If there are more than 100,000 images, however, it is strongly suggested that the user split them into separate folders and run different batches. Let it be known that it would be an easy change to the code to incorporate 6 digits or more, so if that is a need, consider directly changing the code and recompiling.

3.12 NumImages

This simply tells the program the total number of images present in a batch.

4 Additional Details

Below are some implementation details. They are not necessary to run the code but give more context to the simulation.

4.1 Initial Guess at Solution

The solution is initialized as a straight line between CL and CR, assumes a linear concentration gradient.

4.2 Solvers

Currently, there are two solver options: the Jacobi Iteration and a standard over-relaxed Jacobi-Iteration (SOR Jacobi Iteration).

More detail at `https://en.wikipedia.org/wiki/Successive_over-relaxation`.

5 Computational Model and Validation

There is currently a need for better validation. The theoretical/analytical results (parallel and series arrangement) are too easy to achieve, hence there is the need to cross-validate this algorithm with other methods (such as FEA or finite difference), or directly compare readouts to commercial software simulations.

5.1 Boundary Conditions and System Setup

The boundary conditions can be seen in figure 1 below.

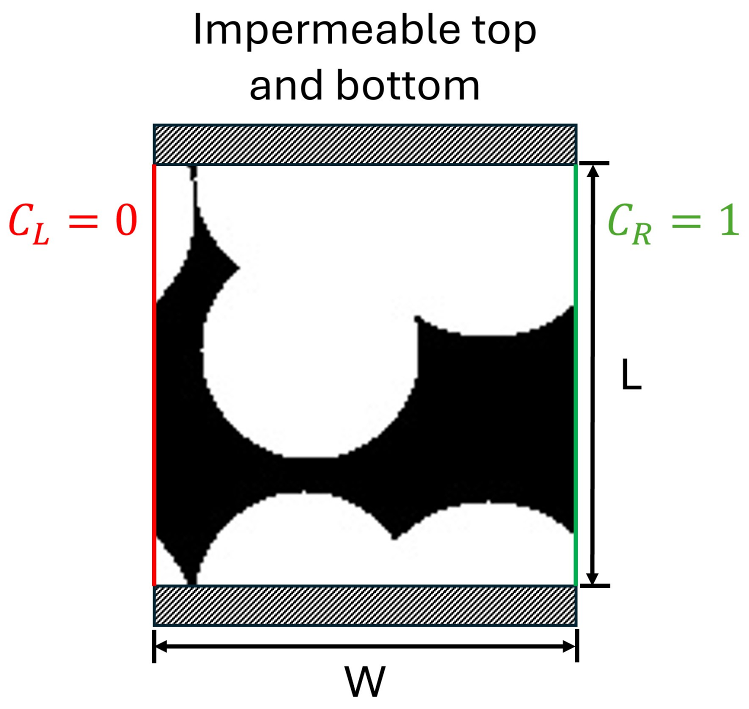

Figure 1: Simulation domain with boundary conditions.

5.2 Computational Model

The computational model is very similar to the computational model from this publication `https://doi.org/10.1016/j.egyai.2023.100310`, except we are dealing with mass diffusion as opposed to thermal conductivity. We are solving the steady-state diffusion equation under trace species assumption, with no mass sources or sinks. The governing equation is written below in equation (4):

$$\frac{\partial}{\partial x}\left(D_{AB}\frac{\partial c_A}{\partial x}\right)+\frac{\partial}{\partial y}\left(D_{AB}\frac{\partial c_A}{\partial y}\right)=0 \tag{4}$$

where D_{AB} is the diffusivity of trace species A in medium B, and c_A is the concentration of A in medium B. From this setup, we can make the model more general by making it non-dimensional. If we assume a system with a width (w) and a length (l), we can make those non-dimensional and normalized by applying the following transformation5.

$$W = \frac{x}{w}$$
$$L = \frac{y}{l} \tag{5}$$

The diffusion coefficients are made non-dimensional by dividing them by the diffusion coefficient of the trace species in the fluid phase, as shown in equation (1). The concentrations are made dimensionless and normalized by applying the transformation from equation (2).

The resulting model is then solved using a simple central differencing scheme from the finite volume method. From this model, the diffusion coefficients are evaluated at the boundaries between cells, and at those locations we use the harmonic mean between the diffusion coefficients of the two cells. A linear set of equations is built and solved using the standard over-relaxed Jacobi Iteration.

From the concentrations gathered, we can calculated the average dimensionless mass flux through the domain, and we call the average mass flux per unit area and per unit time J_A^*. The effective diffusivity of the system emerges from rearranging Fick's Law, so we estimate D_{eff} to be as follows (6).

$$D_{eff} = \frac{J_A^* W}{C_R - C_L} \tag{6}$$

5.3 Validation

In terms of validation, there are two theoretical test cases which we can use to validate our calculation of effective diffusivity, and these are the parallel and series arranged structures. The structures are shown below in figure 2 side by side.

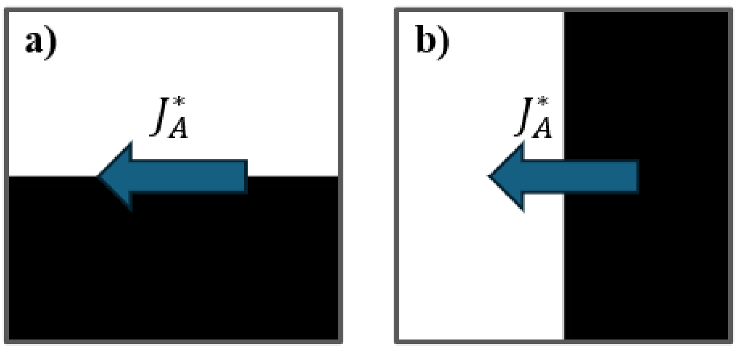

Figure 2: Figure shows the structures for which there are analytical solutions for the effective diffusivity. In **a)** we see the parallel structure arrangement, and in **b)** we see the series structure arrangement. The mass flux is indicated.

For the test cases shown in figure 2, the equations (7) and (8) below allow us to analytically calculate the effective diffusivity.

$$parallel : \bar{D}_{\mathrm{eff}} = \epsilon \cdot \bar{D}_{\mathrm{f}} + (1-\epsilon) \cdot \bar{D}_{\mathrm{s}} \tag{7}$$

$$series : \bar{D}_{\mathrm{eff}} = \left(\frac{\epsilon}{\bar{D}_{\mathrm{f}}} + \frac{1-\epsilon}{\bar{D}_{\mathrm{s}}} \right)^{-1} \tag{8}$$

The bar in $\bar{D}_{\mathrm{eff}}$ and the other diffusion coefficients means they are non-dimensional. $\bar{D}_{\mathrm{s}}$ and $\bar{D}_{\mathrm{f}}$ are the dimensionless solid diffusion coefficient (the white portion) and the dimensionless fluid diffusion coefficient (the black portion). The coefficients are non-dimensionalized by dividing them by D_{f}, similar to what is shown in Adam et al. [1], but instead of thermal conductivity we here use diffusivity coefficients.

For the validation, we consider 2D images of 100 × 100 pixels, and use the following porosities $\epsilon = 0.1, 0.3, 0.5, 0.7, and 0.9$. For each porosity, we have the parallel and series arrangement. D_{f} is always equal to 1 because of the non-dimensionalization, and we consider 5 values of D_{s}, which are all exponents from 10^{-1} to 10^{-6}. The results are shown below in figure 3.

In summary, this simulations setup eliminates some of the error that was present in Adam et al. [1], which occurred when the ratios of the constants were too large. We see from the figure that even when $D_f/D_s = 10^6$, both the series and parallel arrangements retain high accuracy and consistency with the analytical results.

A brief note on validation: The validation using analytical test results is what has been used in literature for other works involving effective thermal conductivity and effective diffusivity. However, those analytical results are very simplistic, and in the future we recommend further work comparing the simulation results to commercial software and other methods. More specifically, we are interested in how these readouts (second order accurate FVM scheme) compare with a higher order FVM scheme, or a finite-difference formulation, or a finite element formulation. This is a note that applies to all work published in computational mechanics in general, not just this work.

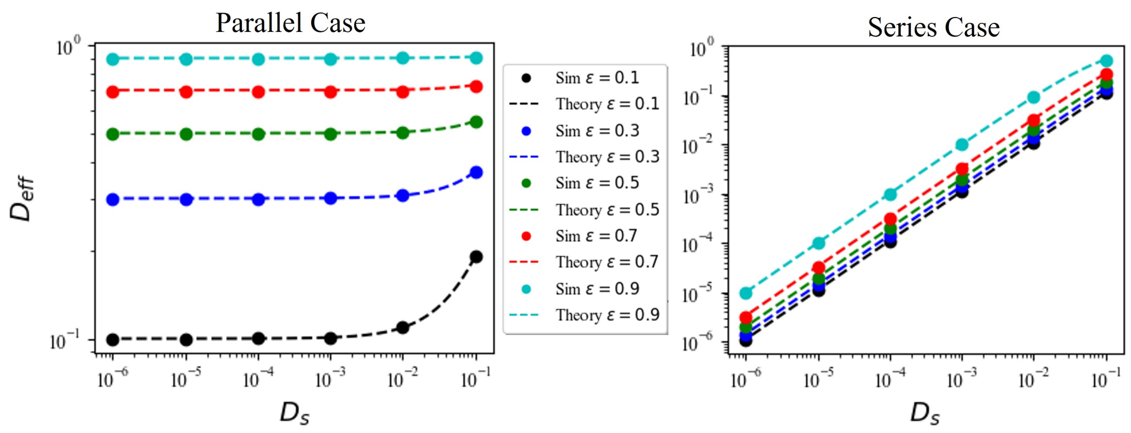

Figure 3: The dots are simulated results, and the lines are the analytical solutions from equation (7) and (8).

5.3.1 Special Case: Thin Phase

There is a special case of figure 3 b), in which we set one of the phases to be really thin. If the phase that is thin is the phase with much lower diffusivity, then we will see some sharp gradients in the overall concentration near where the transition. Furthermore, we should still be able to use the equation (8) to find an analytical answer. Below in figure 4 we show an example of such a setup.

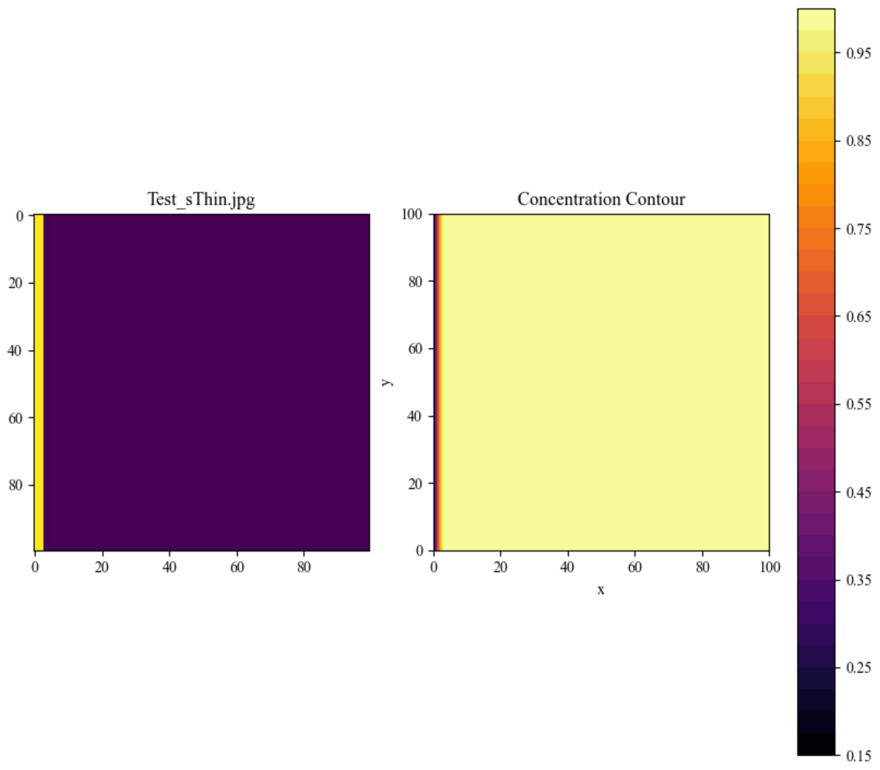

Figure 4: A case where the ratio of $D_f/D_s > 10^6$. On the left we see the actual binary image, whereas on the right we see the concentration distribution. Note that on the concentration distribution we see a sharp gradient where the two phases meet.

This case, the phase with the lower diffusivity (thin phase) occupies only 3% of the domain (3 pixels wide in this case). With the diffusivity of the trace species held at 1 for the thin phase, the diffusivity of the other phase is held at 1,237,500 (it has physical meaning, it is roughly the ratio of diffusivity of oxygen in air over the diffusivity of oxygen in some electrolytes. the order of magnitude is more important than the actual value). The code correctly predicts the effective diffusivity predicted by the equation 8 as being 33.33.

5.3.2 Special Case: 3 Phases

On a setup with solid, fluid, and gas, generally the trace species will diffuse much faster in the gas when compared to the liquid, and the solid is pretty much impermeable in this scenario. For this reason, we can repeat the test in figure 3 for the parallel case, and equation (7) can be extended into:

$$\bar{D}_{\text{eff}} = \text{LVF} \cdot \bar{D}_{\text{f}} + \text{SVF} \cdot \bar{D}_{\text{s}} + \text{GVF} \cdot \bar{D}_{\text{g}} \tag{9}$$

Where LVF, SVF, and GVF are the liquid, solid, and gas volume fractions, respectively. We treat $\bar{D}_{\text{s}} = 0$, since it is virtually impermeably. To $\bar{D}_{\text{f}}$ we maintain the value as 1 (which comes from the non-dimensionalization, so this value is always held at 1), and $\bar{D}_{\text{g}} = 1237500$.

We create a simulation setup where the solid is 30% of the domain, the fluid is 40%, and the gas is 30%, and it can be seen below:

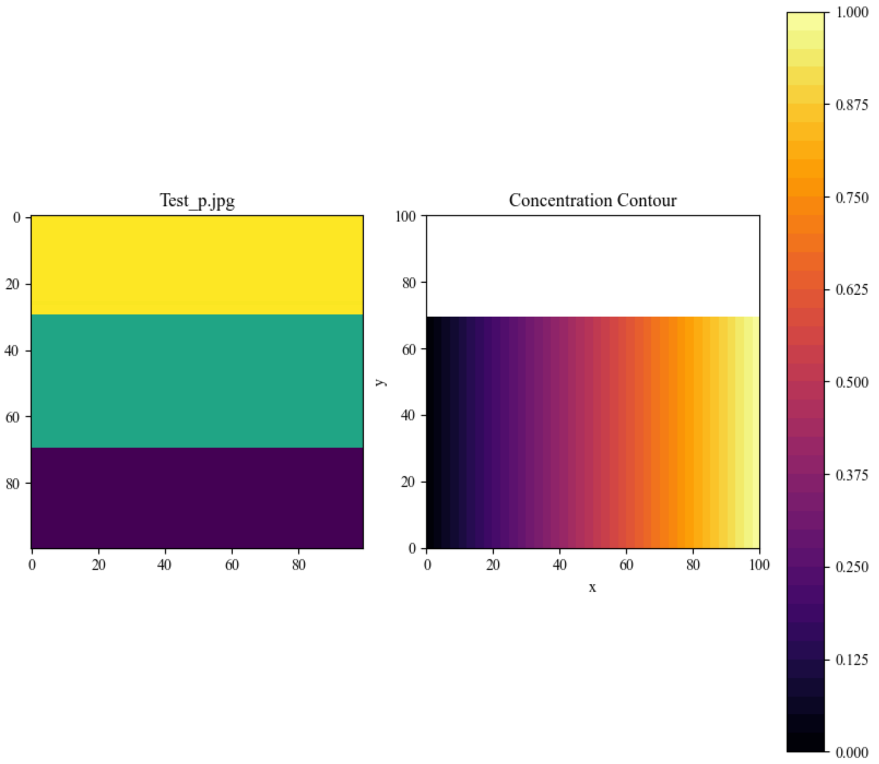

Figure 5: The parallel orientation of the three phases yields a 1D solution, even for a 2D simulation domain. It is easy to see how the concentration gradient remains the same at steady state for the diffusing phases, and the concentration is zeros for the impermeable solid.

Figure 5 correctly predicts the effective diffusivity from equation (9), that value being $\bar{D}_{\text{eff}} = 371250.4$, or roughly 30% of $\bar{D}_{\text{g}}$ plus 40% of $\bar{D}_{\text{f}}$. The impermeable solid does not participate in the simulation, and is accounted for automatically. It also does not break or influence the simulation in any way.

For a case with impermeable solid, we note that a validation using the series case is not possible, as it just yields a diffusivity of 0, as the solid essentially forms an impermeable wall. Varying the fractions of solid, liquid, and gas do not influence the accuracy of the computation, hence only this one case is shown.

5.3.3 Special Case: Wide Domain

In this case we show a similar validation case, but now the domain's width is twice it's height. Naturally, from equation (8) we should expect the same result, and indeed we arrive at the same result. We use a case where the domain is filled with 50% gas and 50% liquid in the series arrangement. Below the concentration map is shown.

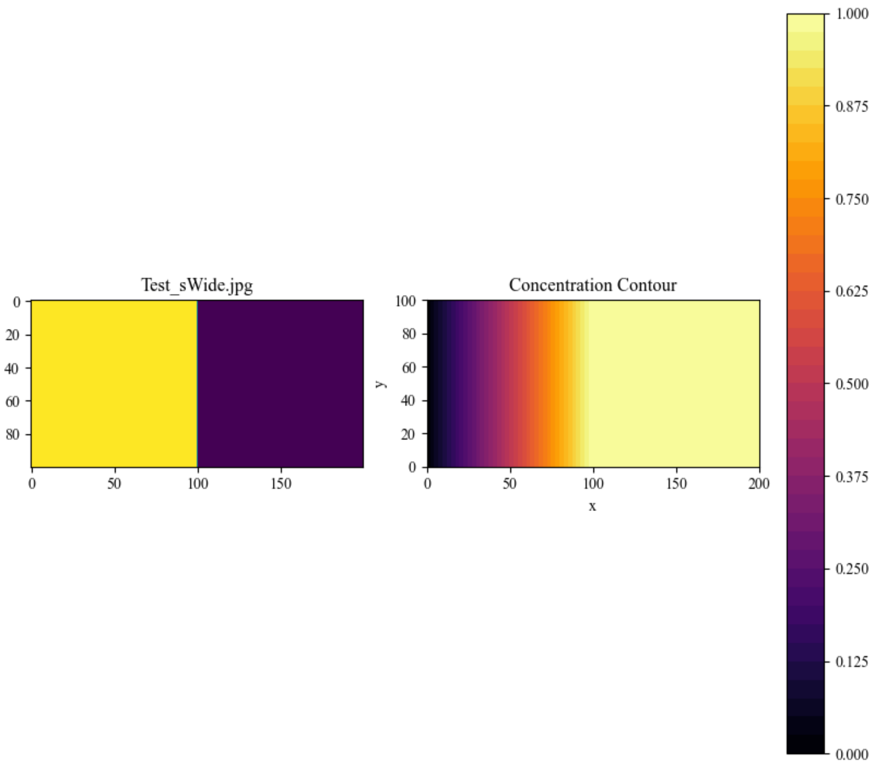

Figure 6: The series orientation on a wide domain yields the same result as expected from the theoretical equation (8) when the domain is 50% liquid and 50% gas.

References

[1] Adam, A., Fang, H. & Li, X. Effective thermal conductivity estimation using a convolutional neural network and its application in topology optimization. *Energy and AI* **15**, 2666–5468 (2024). URL `https://doi.org/10.1016/j.egyai.2023.100310`.