Effective Diffusivity Algorithm Documentation

Andre Adam

Last Updated: May 9, 2024

Contents

1 Introduction **1**

2 Requirements **1**
2.1 Compiling . . . 2
2.2 Image Requirements . . . 2

3 Input.txt **2**
3.1 Phases . . . 2
3.2 Diffusivity . . . 2
3.3 MeshAmp . . . 3
3.4 InputName . . . 3
3.5 CL and CR . . . 3
3.6 OutputName . . . 3
3.7 Print Concentration Map . . . 3
3.8 Convergence Criteria . . . 3
3.9 MaxIter . . . 3
3.10 Verbose . . . 3
3.11 RunBatch . . . 3
3.12 NumImages . . . 3

4 Additional Details **4**
4.1 Initial Guess at Solution . . . 4
4.2 Solvers . . . 4

5 Computational Model and Validation **4**
5.1 Boundary Conditions and System Setup . . . 4
5.2 Computational Model . . . 4
5.3 Validation . . . 5

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.

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 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
```

There is only a GPU option for the code (no CPU based computation), and only for NVIDIA GPUs. 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, assuming in the same folder as the other files, the following command will compile the .cu file into a executable:

```
nvcc Perm2D.cu
```

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 should return an error. The files *stb_image.h*, which were not written by me and are part of an open source project, are 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 as close to 255 and 0 as possible, and keep gray as close to 150 as possible, 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:

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

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 right and left 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.

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.

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

These are some implementation details, they are not necessary to run the code but give insightful information.

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 is only a Jacobi-Iteration and a standard over-relaxed Jacobi-Iteration.

5 Computational Model and Validation

There is currently a need for better validation. The theoretical/analytical results 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.

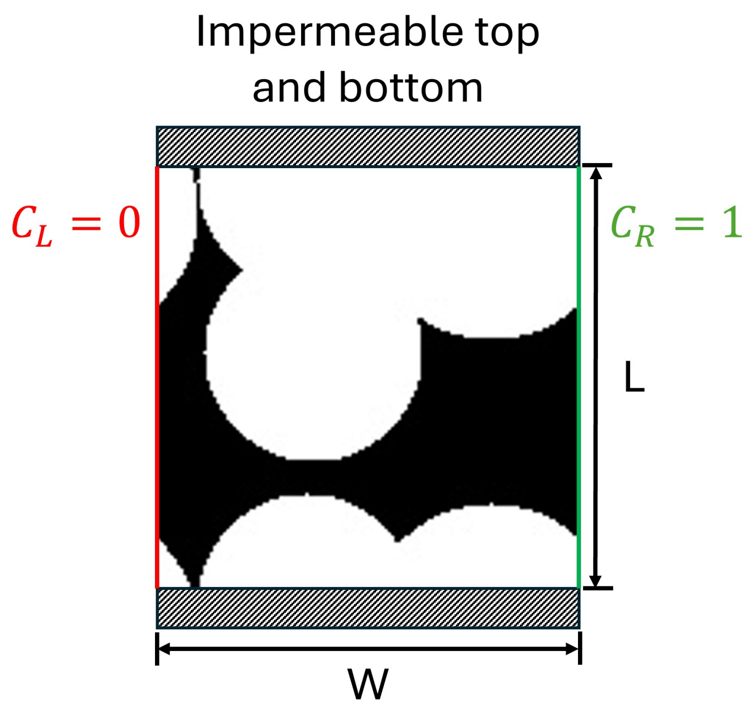

Figure 1: Simulation domain with boundary conditions.

5.2 Computational Model

The computational model is very similar to the computational model from this publications `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.

$$\begin{aligned} W &= \frac{x}{w} \\ L &= \frac{y}{l} \end{aligned} \tag{5}$$

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

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