2D Thermal Conductivity CFD Algorithm Documentation

Andre Adam

Last Updated: June 13, 2023

This document is documentation on how to use the 2D CFD algorithm to predicting the thermal conductivity of a two-phase system. It assumes a steady-state solutions exists, there is no fluid flow (no convection) and no radiation.

The top and bottom boundaries are adiabatic, while the left and right boundaries of the 2D domain are set temperatures. The current code operates in two modes: it can either do a mesh convergence test on single images, or it can run and generate a reading for the effective thermal conductivity (k_{eff}). In the former option, the user can also choose to print temperature maps and heat flux maps with predefined names.

Requirements

To run the code, the following files are necessary:

```
keff2D.cpp
keff2D.h
stb_image.h
stb_image_write.h
input.txt
AnyImage.jpg
```

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.

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* and *stb_image_write.h*, which were not written by me and are part of an open source project, are responsible for reading the specified image.

This image is always under the assumption that the white pixels (or 255 in grayscale) are the solid portion, while the black pixels (or 0 in grayscale) are the fluid or lower thermal conductivity ones. In the input file, the user can specify thermal conductivities of solid and fluid, but note that if in a given case black is assumed to be solid and white to be fluid, then the user will set k_f as the thermal conductivity of solid, and k_s as fluid.

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. It is also important to note that the entire code development was using the gcc family of compilers, so it may or may not work with other compilers.

From the command line, assuming in the same folder as the other files, the following command will compile the .cpp file into a executable:

```
g++ -fopenmp keff2D.cpp -lm -o keff2D.exe
```

A simple double click on the .exe file should run the code with the specified inputs.

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:
ks: 10
kf: 1
MeshAmpX: 1
MeshAmpY: 1
InputName: 00001.jpg
TR: 1
```

```
TL: 0
OutputName: batchTest.csv
printTMap: 1
TMapName: TMAP_00001.csv
printQMap: 1
QMapName: QMAP_00001.csv
Convergence: 0.00001
MaxIter: 500000
Verbose: 1
NumCores: 8
MeshFlag: 0
MaxMesh: 10
RunBatch: 0
NumImages: 500
```

Thermal Conductivity

The first two options of the input file are thermal conductivities, ks and kf. Both of them are interpreted as the datatype double. ks is the thermal conductivity of the white pixels and kf is the thermal conductivity of the black pixels.

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.

InputName

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

TR and TL

Sets the temperatures at the left and right boundary, inputs are interpreted as datatype double.

OutputName

String with the name of the file to save the output of the code. The following is what the code records and saves, in order:

```
keff,QL,QR,Iter,ConvergeCriteria,inputName,nElements,MeshIncreaseX,MeshIncreaseY,
porosity,ks,kf,tl,tr,time,nCores
```

Most of the outputs are self-explanatory, but it is worth mentioning QL and QR are measures of the heat flux through the left and right boundaries, respectively. The time printed is always in units of seconds.

Print Temperature Map

The flag printTMap is a boolean, 1 means true (print temperature map) and 0 means false (no temperature map).

The input TMapName is the name of the file to save the temperature map, can be a .txt file or a .csv. In this file, the x and y coordinates of every pixel are printed along with the temperature at the center of the pixel.

Note that, since this was programmed in C/C++, the origin $(0, 0)$ is the top left corner, and **NOT** the bottom left, as most would expect. Have that in mind when creating contour plots of temperature and heat flux.

Heat Flux Map

The flag printQMap is a boolean, 1 means true (print heat-flux map) and 0 means false (no heat-flux map).

The input QMapName is the name of the file to save the heat-flux map, can be a .txt file or a .csv. In this file, the x and y coordinates of every pixel are printed along with the heat-flux in the x-direction at the center of the pixel.

Note that, since this was programmed in C/C++, the origin $(0, 0)$ is the top left corner, and **NOT** the bottom left, as most would expect. Have that in mind when creating contour plots.

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 k_{eff} every 10 iterations. The change in k_{eff} is calculated as follows:

$$\text{Change} = \frac{|k_{eff,old} - k_{eff,new}|}{k_{eff,old}} \tag{1}$$

From the example input file shown above, the code will terminate when the change reaches the number input. In this example, Convergence is set to 0.00001, which means when the change in k_{eff} is less that 0.01% in the last 10 iterations.

MaxIter

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

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.

NumCores

Parallel computing is currently implemented using simple OpenMP constructs. The additional number of cores is applied to speed up operations by the solver. If a number is entered that is larger than the current available number of cores, the code does not return an error, and that is due to the way OpenMP works.

It is on the user to know how many cores are available.

Mesh Convergence

The option MeshFlag is a flag to perform a convergence test. Setting this flag to 1 (true) performs a mesh convergence test, and the output is saved to the OutputName file. No temperature maps or heat flux maps are printed no matter what flags they have, the MeshFlag overrides everything.

If MeshFlag is set to 1, the options AmpMeshX and AmpMeshY also don't do anything. Instead, the MaxMesh option will take an integer. That integer stands for the maximum MeshAmp factor.

In the example given above, MaxMesh is set to 10. Assuming the original image is 100×100, this example will run the original image, a 2-fold increase in the number of cells in the x and y directions (4-fold increase in the number of elements), 3-fold, 4-fold, all the way up to MaxMesh, 10-fold increase in the number of elements in each direction (100-fold increase in the number of elements).

0.1 MaxMesh

When the MeshFlag is set to true, this option will limit what the biggest mesh considered in the mesh convergence will be. If MeshFlag is set to false, this option is completely ignored.

0.2 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.

Important note on RunBatch & MeshFlag: Only one of these can be active at a time, they are mutually exclusive modes in the code.

0.3 NumImages

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

Additional Details

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

Initial Guess at Solution

The solution is initialized as a straight line between TR and TL, assumes a linear temperature gradient.

Solvers

There are three solvers currently implemented: Jacobi Iteration, Gauss-Seidel Method, and the Iterative TriDiagonal Matrix Algorithm (TDMA). TDMA and Gauss-Seidel perform quickly in serial, with comparable time. However, Gauss-Seidel is very easily implemented in parallel, while TDMA is not, hence Gauss-Seidel is used. The user doesn"t currently have an option to chose which solver they want without editing the code, and the reason for this is that Gauss-Seidel is just better than the others.

Future Work

- Test and validate this code for effective diffusivity as well.
- More flexible options for input images, both in terms of format as well as reading RGB images and converting to grayscale, and ultimately binary.
- Add the option to consider more than two phases (systems of 3 or more phases), but this change is honestly more of a matter of properly interpreting input images rather than adapting the current algorithm, as it would already be able to handle 3 phases.