Image Analysis using Percolation theory

Abhinav M

November 24, 2021

Table of Contents

Abstract

The primary objective of this project is to develop an algorithm that can analyse scientific images to investigate relationships between associated variables in a system by comparing the image data before and after a change has taken place. The algorithm(s) used here is based on the concept of percolation in a lattice. It will focus on identifying clusters in the image dataset and how they evolve as changes take place in the system.

Chapter 1: **Introduction**

Images have always been at the forefront of scientific discoveries from Astrophysics to Quantum mechanics. Imaging is crucial not only because they help us comprehend the world through our senses but also due to the huge advances that have been made in scientific imaging technologies. These technologies help us document our observations and extract vast amounts of data.

Imaging is of paramount importance in fields such as material science, medicine, astrophysics, metallurgy, geology and so on. With such a vast domain of application we are met with the need to process this image data to extract useful information. Simply looking at an image with the naked eye and drawing conclusions might not be accurate or feasible in most cases especially when the data in question is more or less abstract. Therefore, we need to develop computer programs or algorithms to take in this image data and present us with useful metrics so that we can find the underlying relationships between variables in a system.

This project will be focused on comparing two images before and after a change/process has taken place in the system by looking at how the clusters in the image evolve and how the layout and arrangements of the clusters change.

1.1 Introduction to percolation

Percolation in general is the phenomenon by which a substance seeps into another substance. A classic example of this would be water soaking into soil. During this process, water slowly travels down the soil layers by filling empty air pockets in between soil particles. At any instant in time, the fraction of water-filled air pockets (occupied sites) to that of total available air pockets is known as the site occupation probability. During percolation, the invading material, in this case, the water, travels through random paths in the soil, forming complex networks, patterns and clusters. These patterns and cluster formation phenomenon are not unique to percolation phenomenon and may find applications elsewhere. This is what we are interested in.

1.2 Programming approach

In order to study cluster formation and evolution in scientific images we first need to convert our image data into a lattice such that each pixel in the image is in one of two states – occupied or empty. We do this by binarizing the image with a calculated threshold value. Now we have a matrix with cells that are either filled or empty. We can now start identifying the clusters and calculate different metrics relating to the shape, size and arrangement of the clusters and study how they change after the system has gone through a change.

1.3 MATLAB

This project has been entirely developed on MATLAB. MATLAB is a popular programming language and scientific computing environment specifically designed for scientific and engineering applications. Convenience of debugging and the ease of translating ideas into code were some of the main factors that made MATLAB the preferred language for this project.

1.4 Useful metrics

As our program is concerned with analysing the change in cluster formations in a lattice before and after a change has taken place, in order for us to quantify these changes in shape and arrangement of clusters, we have included the following metrics that the program can measure and track –

(i) Site occupation probability

(ii) Number of clusters

(iii) Size of clusters

(iv) Distance between centre of two clusters

(v) Minimum distance between two clusters

Chapter 2: **Code review**

The program takes two images as input, one of the images is of the original system and the other is taken after a change has happened or the system has undergone a process such as doping, annealing etc. Using these two images, it calculates an optimum threshold value to binarize the images and gives us insights into the cluster formation and how these evolve from the first image to the second.

A couple of key steps in this process are elaborated below.

2.1 Binarizing RGB images

Binarizing the input images is one of the most important steps in our analysis. To binarize the RGB image the **greythresh** function is used separately in R,G and B layers of both the images and weighted average is used to calculate the threshold level separately for both images. Mean of these thresholds are used to binarize the image. **Im2bw** function is used to obtain the binary image.

2.2 Labelling Clusters

Labelling clusters forms the core of our program’s logic. We use MATLAB’s built in **bwlabel** function to label the clusters in the binary image. Bwlabel is based on the HK Algorithm. After the clusters are labelled, it is just a matter of making the algorithm aware of the location of every element under each cluster label and then we can extract all the metrics that we require for our analysis.

2.3 Source code

The complete source code with detailed comments is given below.

testim.m

```
%#ok<*NOPTS>
%------------------------------------------------------------------------
                          %USEFUL VARIABLE NAMES
%------------------------------------------------------------------------
%I -> image that is to be processed
%I2 -> the second image (for finding average threshold)
%num -> number of clusters
%lvl -> average threshold for both images
%Iclean -> binary matrix of I
%occupied -> number of occupied sites
%count_array -> number of elements in each cluster (size of clusters)
%imhei,imwdh -> pixel dimensions of the input image
%sorted_cords -> coordinates of elements sorted according to cluster label
%adjusted_coc -> rounded off coordinates of center of clusters
%red,green,blue -> 0-255 values to find colour corresponding to threshold
%Cluster_Label_Im -> binary matrix after labelling
%center_of_clusters -> coordinates of center of each cluster
%dist_bw_centers(M,N,adjusted_coc)-> distance b/w centers of clusters M,N
%min_distance_bw_clusters(M,N,sorted_cords,count_array)-> min. distance b/w
clusters M,N
%------------------------------------------------------------------------

clear vars;

%--!--!--!--!--!--!--!--!--!--!--!--!--!--!--!--!--!--!--!--!--!--!--!--
                   % NEED TO ADJUST BELOW LINES BEFORE RUNNING
%--!--!--!--!--!--!--!--!--!--!--!--!--!--!--!--!--!--!---!--!--!--!--!--

%SELECT ONE OF THE 2 LINES BELOW,(IMAGE TO BE PROCESSED)
I = imread('pure00.jpg');
%I = imread('annealed00.jpg');


%SELECT ANNEALED/PURE ACCORDING TO WHAT YOU SELECTED ABOVE
imwdh = 512; imhei = 510; %pure
%imwdh = 530; imhei = 511; %annealed


%SELECT I2 AS THE ONE YOU DID NOT SELECT ABOVE
%I2 = imread('pure00.jpg');
I2 = imread('annealed00.jpg');

%--!--!--!--!--!--!--!--!--!--!--!--!--!--!--!--!--!--!---!--!--!--!--!--






%------------------------------------------------------------------------
% NOTE: UNCOMMENT LINE 104 IF YOU WANT TO SEE THE BINARY IMAGE AS OUTPUT
(slower run-time)
%------------------------------------------------------------------------
```

```
%___________________________________________________________________________%
                      %FINDING THE THRESHOLD WITH GREYTHRESH
%---------------------------------------------------------------------------%
%Below code is for finding the average threshold 'lvl' for both images
redChannel = I(:, :, 1);
greenChannel = I(:, :, 2);
blueChannel = I(:, :, 3);
RL= graythresh(redChannel);
GL= graythresh(greenChannel);
BL= graythresh(blueChannel);
redChannel2 = I2(:, :, 1);
greenChannel2 = I2(:, :, 2);
blueChannel2 = I2(:, :, 3);
RL2= graythresh(redChannel2);
GL2= graythresh(greenChannel2);
BL2= graythresh(blueChannel2);

lvl=(RL*RL + BL*BL + GL*GL)/(RL+GL+BL); %Weighted average of RGB layers for
first image
lvl2=(RL2*RL2 + BL2*BL2 + GL2*GL2)/(RL2+GL2+BL2); %Weighted average of RGB
layers for second image


lvl=(lvl+lvl2)/2; %Average threshold for both images
thresh_lvl=lvl

%__________________________________________________________________________%
                         %FINDING THE THRESHOLD 'COLOUR'
%--------------------------------------------------------------------------%
%This is converting layer thresholds into 0-255 values
R1=255*RL;
G1=255*GL;
B1=255*BL;
R2=255*RL2;
G2=255*GL2;
B2=255*BL2;

%Everething below this shade of rgb is black in both images
red=round((R1+R2)/2);
green=round((G1+G2)/2);
blue=round((B1+B2)/2);

RGB_thresh=[red,green,blue]

%___________________________________________________________________________%
                 %BINARY IMAGE OBTAINED AS PER THRESHOLD 'lvl'
%---------------------------------------------------------------------------%
%binary image obtained
Iclean = im2bw(I,lvl);
%imshow(Iclean); (Line 104)
%___________________________________________________________________________%
```

```
%________________________________________________________________________________%
                    %FINDING THE SITE OCCUPATION PROBABILITY
%--------------------------------------------------------------------------------%

%To find SOP
occupied=0;
for x=1:1:imhei
for y=1:1:imwdh
if(Iclean(x,y)==1)
occupied=occupied+1;
end
end
end
sop=occupied/(imhei*imwdh)
%________________________________________________________________________________%




%________________________________________________________________________________%
                              %LABELLING THE CLUSTER
%--------------------------------------------------------------------------------%

%cluster labelling
[Cluster_Label_Im,num] = bwlabel(Iclean,4);
%________________________________________________________________________________%




%________________________________________________________________________________%
    %FINDING X,Y CO-ORDINATES OF ALL CLUSTER ELEMENTS(SORTED BY CLUSTER NAME)
%--------------------------------------------------------------------------------%

fileId = fopen('cord_out.txt', 'w'); %WILL CONTAIN COORDINATES OF ALL CLUSTER
ELEMENTS
for x=1:1:imhei
for y=1:1:imwdh
k=Cluster_Label_Im(x,y);
if(k~=0)
fprintf(fileId, '%d %d %d\n', k,x,y);
end
end
end
fclose(fileId);

%SORTING BY CLUSTER NUMBER/LABEL BELOW
cords = readmatrix('cord_out.txt');
sorted_cords = sortrows(cords,1); %<<- sorted_cords contains (cluster_label, x,
y) values, open to see
fileId1 = fopen('sorted_cords.txt', 'w');
writematrix(sorted_cords,'sorted_cords.txt')
%________________________________________________________________________________%
```

```
%_____________________________________________________________________________%
        %FINDING THE CENTER OF EACH CLUSTER AND THE SIZE OF EACH CLUSTER
%-----------------------------------------------------------------------------%

count_array=zeros(num); %<<- will contain size of all clusters
%Finding center of each cluster and size of each cluster below
CC=0;
i=1;
x=1;
y=1;
count=1;
loopvar=1;
i=1;
av=0;
sumx=0;
sumy=0;
fileId4 = fopen('xd_out.txt', 'w');
fileId5 = fopen('yd_out.txt', 'w');
fileId6 = fopen('mdst_clstrs_out.txt', 'w');
while(count<occupied)

if(sorted_cords(loopvar,1)==i&& i<occupied)
av=av+1;
sumx=sumx+sorted_cords(loopvar,2);
sumy=sumy+sorted_cords(loopvar,3);
loopvar=loopvar+1;
CC=CC+1;
end
if(sorted_cords(loopvar,1)~=i&& i<occupied)
fprintf(fileId4, '%d\n', sumx/av);
fprintf(fileId5, '%d\n', sumy/av);
sumx=0;
sumy=0;
av=0;
i=i+1;
count_array(i)=CC; %<<- size of each cluster array, (don't take the first zero
value...offset by +1, open the array to see what I mean)
CC=0;
end
count=count+1;
end

%Coordinates of center of all clusters present in center_of_clusters
a=dlmread('xd_out.txt');
b=dlmread('yd_out.txt');
center_of_clusters=[a,b];
adjusted_coc=round(center_of_clusters);
%_____________________________________________________________________________%
```

```
%_____________________________________________________________________________%
  %FINDING SIGNIFICANT CLUSTERS BY EXCLUDING CLUSTERS WITH SIZE LESS THAN THE
AVERAGE CLUSTER SIZE
%-----------------------------------------------------------------------------%

%List of clusters with more than average cluster size
%i.e, significant clusters are listed inside sig_clusters
sig_clusters=zeros(num);
av_clstr_size=round(mean(count_array(:,1)));
j=1;
for i=1:1:num
if(count_array(i,1)>av_clstr_size)
sig_clusters(j)=i;
j=j+1;
end
end
%_____________________________________________________________________________%




%_____________________________________________________________________________%
                               %SOME USEFUL FUNCTIONS
%-----------------------------------------------------------------------------%
M=2408;
N=272;

%_____________________________________________________________________________%
% BELOW LIES THE FUNCTION THAT FINDS MIN DISTANCE BETWEEN 2 CLUSTERS M,N
%-----------------------------------------------------------------------------%
[min_dist,cl1_xy,cl2_xy] =
min_distance_bw_clusters(M,N,sorted_cords,count_array);

min_dist %->> MINIMUM DISTANCE


%_____________________________________________________________________________%
% BELOW LIES THE FUNCTION THAT FINDS DISTANCE BW CENTERS OF CLUSTERS M,N
%-----------------------------------------------------------------------------%
ctr_dist = dist_bw_centers(M,N,adjusted_coc);

ctr_dist %->> DISTANCE B/W CENTERS




%___________________________________:p_______________________________________%
```

dist_bw_centers.m

```
function [ctr_dist] = dist_bw_centers(cl1_lb,cl2_lb,adjusted_coc)

cl_1_xy(1,1)=adjusted_coc(cl1_lb, 1);
cl_1_xy(1,2)=adjusted_coc(cl1_lb, 2);

cl_2_xy(1,1)=adjusted_coc(cl2_lb, 1);
cl_2_xy(1,2)=adjusted_coc(cl2_lb, 2);

ctr_dist = pdist2(cl_1_xy,cl_2_xy);
end
```

min_dist_bw_clusters.m

```
function [min_dist,cl1_xy,cl2_xy] =
min_distance_bw_clusters(cl1_lb,cl2_lb,sorted_cords,count_array)
w=1;
hh=sum(count_array(2:cl1_lb))+1;
if(cl1_lb==1)
hh=1;
end
for i=hh:1:count_array(cl1_lb+1)+sum(count_array(2:cl1_lb))
cl1_xy(w,1)=sorted_cords(i, 2);
cl1_xy(w,2)=sorted_cords(i, 3);
w=w+1;
end
w=1;
gg=sum(count_array(2:cl2_lb))+1;
if(cl2_lb==1)
gg=1;
end
for i=gg:1:count_array(cl2_lb+1)+sum(count_array(2:cl2_lb))
cl2_xy(w,1)=sorted_cords(i, 2);
cl2_xy(w,2)=sorted_cords(i, 3);
w=w+1;
end
min_dist = min(min(pdist2(cl1_xy,cl2_xy)));
end
```

Chapter 3 : Test data

To test the program two AFM images that studied the current flow through a metal alloy before and after annealing were used. These before and after images are given below for reference. In these images the brighter shades of yellow signify a higher current flow. We can determine the current flow from the shade of yellow in the images.

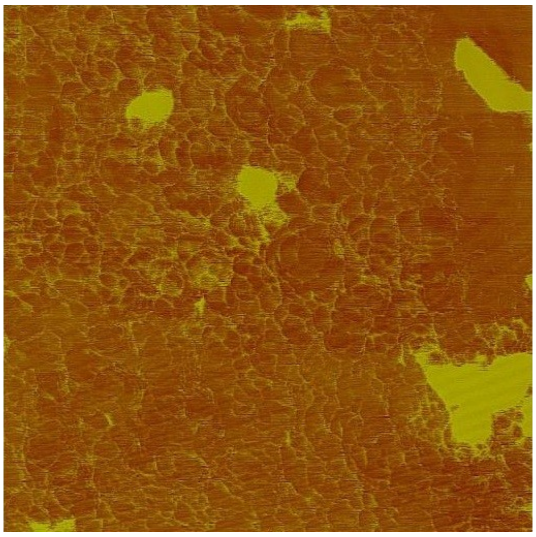


Image 1. Pure sample (before)

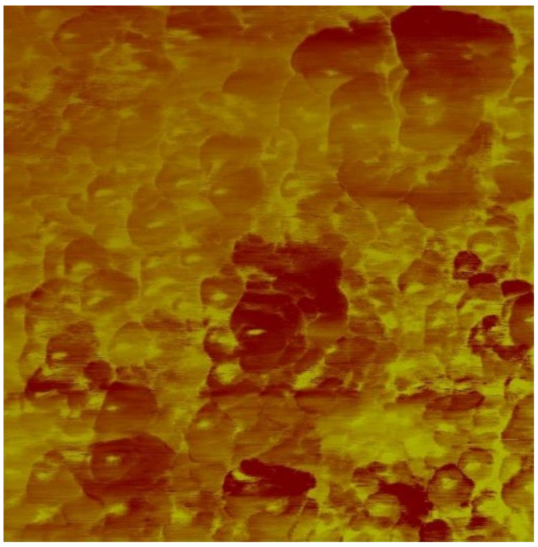
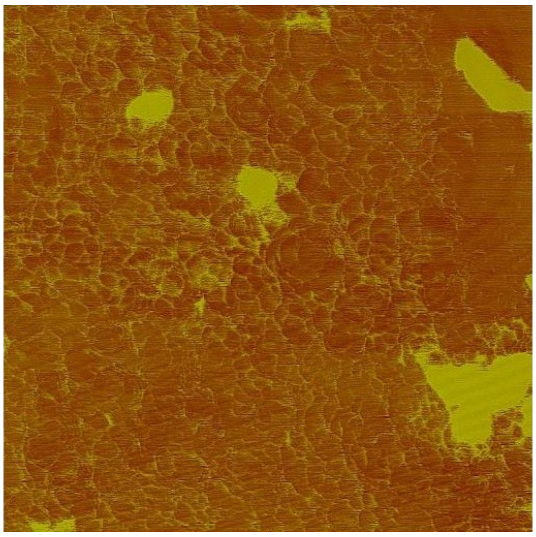
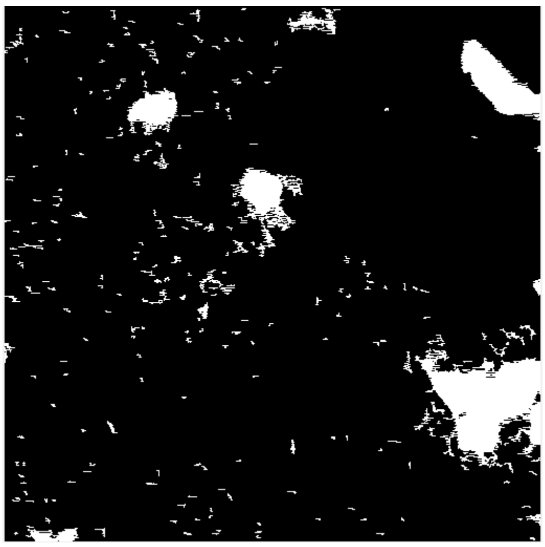
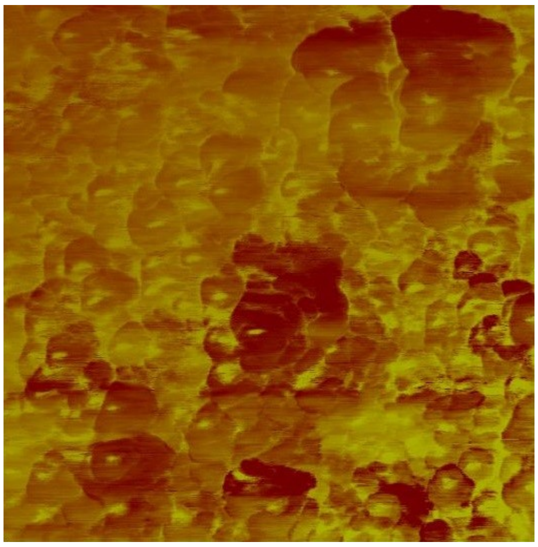


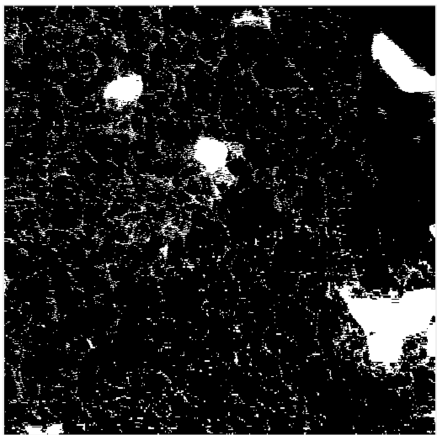
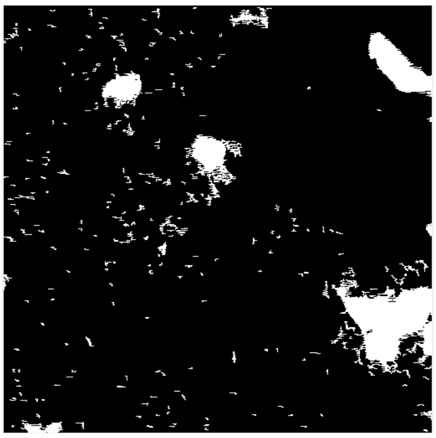
Image 2. Sample after Annealing

Chapter 4 : Results

The results of processing the images are shown below.

The two images below show filtering for significant clusters where only clusters with size above the average cluster size are displayed on the right.

Future enhancements

Need to link this method of image analysis with some experimental data to predict observations or relationships between variables in the system.

Conclusion

The program is able to process images and analyse cluster formation to extract useful metrics such as - Site occupation probability, Number of clusters, Size of clusters, Distance between centre of two clusters, Minimum distance between two clusters etc. Performance was good and output was obtained without errors.