Two-Point Correlation Microstructure Characterization and Reconstruction of Two-Phase Material

ME 341 Final Project Report

Sourav Saha
Sam Sorkin

Date Submitted: December 13, 2018

Table of Contents

List of Figures

ABSTRACT

Microstructure characterization and reconstruction (MCR) is an essential step of building a process-structure-property (PSP) relationship for materials. There are many different methods for MCR. Among those methods, *n*-point correlation functions are popular for microstructure characterization and Monte-Carlo methods like simulated annealing (SA) are widely used for reconstruction. The aim of the project is to identify a proper 2-point correlation function that can statistically describe a two-phase microstructure and use a SA algorithm to reconstruct that microstructure. The project involves developing MATLAB codes that can perform the characterization and reconstruction processes. Additionally, the authors have tried to explore the different tuning parameters of SA algorithm and window size of images to find the capability and efficiency of their code.

KEYWORDS Two-point correlation, MCR, Simulated Annealing, MATLAB

INTRODUCTION

Multi-scale modeling of materials has recently gained popularity in computational materials science and computational mechanics [1]. Through multi-scale modeling, engineers and computational scientists are trying to build PSP relationships for materials. These relationships can be used to design materials with desired properties by adopting a bottom-up approach [2]. Integrated computational materials science and engineering (ICME) [3], is a key part of this approach which uses microstructural information to identify relevant process parameters. There are a multitude of methods to describe microstructures, some examples are linear path functions, the YT method, and advanced machine learning informed methods like spectral density function [2]. Liu et al. [4] proposed a two-point correlation function approach for microstructure characterization. This method has been in use extensively for image processing for a long time. The method provides a very simple, yet accurate, tool to characterize a two-phase microstructure. This correlation function is being used extensively to characterize microstructures in a data-driven simulation framework [5].

After the characterization of the microstructure, the next step involved reconstruction. In order to reconstruct the microstructure in computationally efficient manner, many algorithms like genetic algorithm, discrete minimization based algorithm, gradient based algorithm [2], and simulated annealing algorithms are in use. SA is Monte-Carlo based technique that always aim for the global minima when minimizing a function. Hence, the algorithm is appropriate for reconstruction work as it involves reconstructing a characteristic function that matches the most with the function constructed from target binary image (more in methods section). However, SA demands significant computational time due to its inherent nature. In this work, the authors tried to tune the SA parameters in such a way that minimum time is required to reach a sufficiently accurate reconstructed image. The whole point of such characterization and reconstruction is to realize statistically similar microstructure so that we can analyze it with multiple instances to see how the different realizations of the structure with the same general material properties may lead to variation in mechanical properties. In the subsequent sections of this report, the general methodology for microstructure reconstruction and results of the analysis are mentioned. In addition, based on the results of the analysis some insights on our developed code and possible future modifications are provided.

METHODOLOGY

Two-point Correlation Function

The two-point correlation function strategy to identify the microstructure works well when there are two phases in the material. In order to get a proper statistical description of the image, the experimental image needs to be transformed into a binary image at first where black and white pixels represent different phases [4]. For example a nano-composite with 20% volume fraction will have 20% of the binary representation of image as 1s. After constructing the binary image the next step is to build two-point correlation function to describe the statistical features of the image. Definition of two-point correlation function in the context of image characterization can be described as the probability of tossing a line on to the image and having both ends of the line be on the same phase of the material. Mathematically, if Φ_1 and Φ_2 are the solid volume fractions of the two phases of the material, then $\Phi_1 + \Phi_2 = 1$. For a given microstructure, the indicator function $I^1(\mathbf{x})$ at the coordinate $\mathbf{x}$ ε V (probability space) can be expressed as [6],

$$I^1(\boldsymbol{x}) = \begin{cases} 1, & \boldsymbol{x} \in V_i, i = 1,2 \\ 0, & otherwise \end{cases}$$

The two-point correlation function is the expectation value of the product $I^1(\mathbf{x})\ I^2(\mathbf{x})$ written as,

$$S_2^1(x_1, x_2) \equiv \langle\, I^1(x_1) I^1(x_2)\, \rangle$$

For the analysis in present work a statistically isotropic and homogeneous image is chosen. For such media, the two-point correlation function is related to the scalar distance between any two points and can be written as $S_2^1(r)$ where r is the scalar distance between x_1 and x_2. The properties of $S_2^1(r)$ are mentioned below

- If r = 0, $S_2^1(r)$ equals to the volume fraction of phase 1.
- If r is sufficiently large, $S_2^1(r)$ equals to the square of volume fraction of phase 1.

Implementation of Two-point Correlation Function

The characterization of a binary microstructural image was implemented in MATLAB® by directly calculating the two-point correlation function. A binary matrix representing a given two-phase microstructural image is parsed through, considering every possible pairing of two cells, without repetition. The radius, r, between each pairing of cells is calculated as $r = \sqrt{\Delta x^2 + \Delta y^2}$, where Δx and Δy are the differences between the values of row and column indices of the two cells, respectively. This radius is stored in the first column of an n-by-2 "storage" array (n being the number of total cell-pairings), while the second column is populated with a 1 if both cells contain a 1, otherwise with a 0. After being fully populated, the "storage" array is used to calculate both the total number, $count(r)$, of each unique radius value and the sum, $sum(r)$, of all the values associated with each unique radius value. The value at each radius of the two-point correlation function, $S_2^1(r)$, is calculated as

$$S_2^1(r) = \frac{sum(\mathrm{r})}{count(\mathrm{r})}\,.$$

Due to the finite size of our matrix, there will be fewer instances of a radius value the larger that value is. This is due to not being able to look beyond the boundaries of the array. The lower sampling of radius values will result in higher variability between successive values of r as r gets larger. To reduce this variability (as well as to give lower values of r more importance in reconstruction, which will be discussed in a later section) binning is performed by averaging the $S_2^1(r)$ of progressively higher amounts of unique radius values as the radii grow larger. We consider this binned $S_2^1(r)$ to be the two-point correlation function of the image.

We made an improvement to the basic design by incorporating a modification in the code which first identifies which phase has the lower fraction inside the image. If it is the 1s, the code runs as normal. If the 0s have the phase fraction, the code first inverts the 1s and 0s, then performs the algorithm as normal, inverting the 1s and 0s again before returning the reconstructed image. In this way, calculations are always performed on the smaller of the two sets of elements.

Simulated Annealing

The simulated annealing algorithm is a stochastic algorithm widely used for optimization. This is a Monte-Carlo based optimization method which takes randomly generated value for design parameter and tries to optimize the function based on some probability. The key feature of the SA algorithm is that it does not discard the worse design points directly rather associates these points with certain probability of acceptance. Within the framework of this project, the SA algorithm can be explained in the following steps:

Step 1. Take an input of randomly generated binarized microstructural image.

Step 2. Compute the two-point correlation function for the input.

Step 3. Compare the correlation function with that of the target image (original microstructure) and calculate the difference (error in the correlation function).

Step 4. If the error is less than the previous iteration then the new image is accepted as an improvement. If the error does not decrease then a random number, *r*, between 0 and 1 is generated and compared with the value

$$P = e^{\frac{-\Delta f}{kT}}.$$

Here P is the probability of acceptance, Δf is the difference of errors between new and previous correlation functions, T is Temperature, and k is a constant used for weighting (generally the Boltzmann constant, but it can be modified to suit the need). If $P > r$, accept the new design (in this case the image of the microstructure) and reject it otherwise. The temperature is an analogous parameter to the physical annealing process, as at higher temperatures greater diffusion takes place. In the case of our algorithm, instead of diffusion the temperature dictates how lenient we will be to accept designs with greater error. In the initial stage (when temperature is high), probability of acceptance is higher, but as the simulation runs on (or, as the metal cools) temperature should be slowly dropped to accept higher errors less frequently.

Step 5. If the new image is not within tolerance of the target distribution, randomly swap pixels from two different phases and test the newly generated design (starting from step 2).

Step 6. If at any point the error is less than the set value of tolerance terminate simulation with the optimum design.

Implementation

A brief discussion on the implementation of these steps are as follows:

Step 1. The random binary image is generated using MATLAB® by keeping the phase fraction similar to our target image.

Step 2. The two-point correlation generator function is implemented in this step in order to generate new value of correlation function.

Step 3. This is an important step and requires some discussion by its own merit. The correlation function generated in step 2 contains some variation, which is natural. In order to consider this function as a basis of comparison of error in the subsequent steps we need to choose some definite points on the function value. Before we can do that we need to smooth the graph out. To accomplish this without losing data of higher importance, we put more weight at the front part of the correlation function since we are interested primarily in this part (as shown in Fig 1.). In our work, we implemented this by binning the radius values and finding the average value of the function for that bin. The binning was done a way so that we take a few points at the front as they are more important and more and more points as we go further and further.

By binning and getting the values of function in this way we could retrieve the points so that there is no general loss of data in the process.

Step 4. Step 4 is the convergence determining step for the entire algorithm. The choice of *T* and cooling schedule (the rate at which *T* decreases) are very critical for the convergence of the algorithm. Depending on the cooling schedule the computational time may increase or decrease. In our code, we implemented the temperature scheduling to be an exponentially decaying function. We tweaked the exact nature of how the temperature is reduced, but in general the idea was to run the optimization algorithm for a certain number of iterations and then reduce the temperature. In exponential function, the temperature is reduced at a quicker rate at first knowing that these are mostly wrong designs and gradually the temperature reduction is slowed down as we get closer to an optimum.

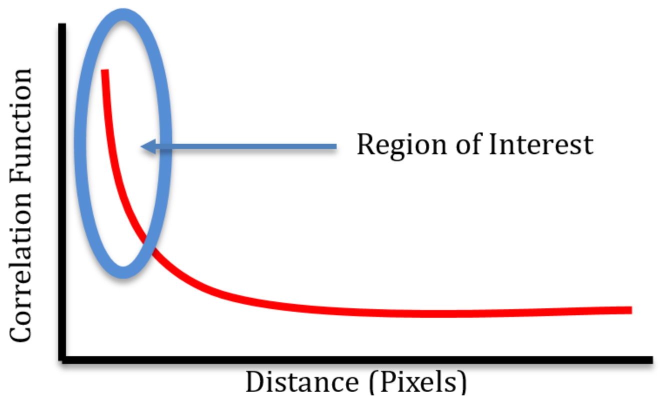


Fig 1. A general representation of the Two-point correlation function.

Step 5. In step 5, the swapping of pixel is the controlling part of how the fast we are approaching the desired microstructure. From our literature survey, we found that it is the general norm to swap one pixel from each phase per iteration. That is the method we chose to implement in our code. However, there are some improvements that can be made. For example, the pixels with the highest number of neighbors with dissimilar phase could be chosen to swap. One improvement we did make was that our code has the provision to swap any number of pixels per iteration. In our example, we specifically chose to swap 10 pixels at the start and relatively quickly decreased to swapping 1 pixel at the later parts of the iterations assuming that initially the generated microstructural images will be more erroneous.

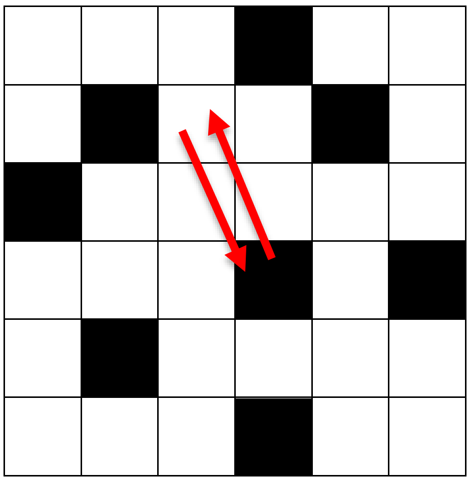
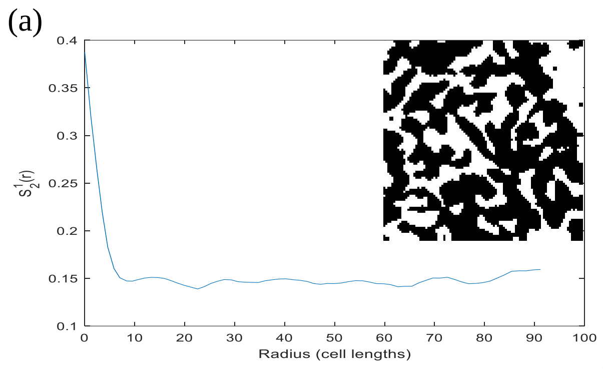
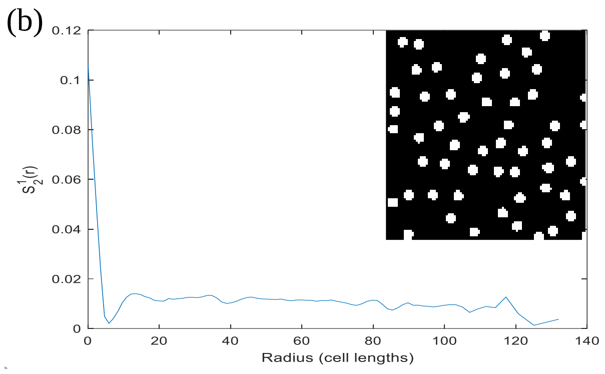


Fig 2. Swapping Pixels during Iteration.





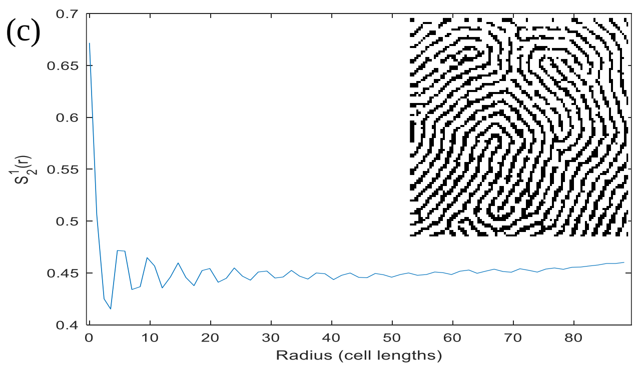


Fig 3. Sample two phase microstructures and their two-point correlation functions. The corresponding microstructures are shown in the inset.

CHARACTERIZATION OF SAMPLE MICROSTRUCTURES

Since SA algorithm is computationally expensive, before going for the reconstruction part we wanted to know which microstructure will be feasible for analysis. For that reason, we chose three representative microstructures to analyze. The results are presented in Figs. 3(a), (b), and (c). For each case 100×100 image size was chosen to analyze. As we can see, our code was able to capture the inherent statistical nature of the microstructure. It is perhaps most apparent ini Fig. 3(c) where the periodic nature of different phases of the materials is captured by the zig-zag in the two-point correlation function. From the result of this analysis we can see that Fig. 3(b) shows the lowest phase fraction for the microstructure. We finally chose this structure because we have the chance of converging quickly with this structure. This microstructure resembles the microstructure of any material with voids inside it and also could represent any nano-composite.

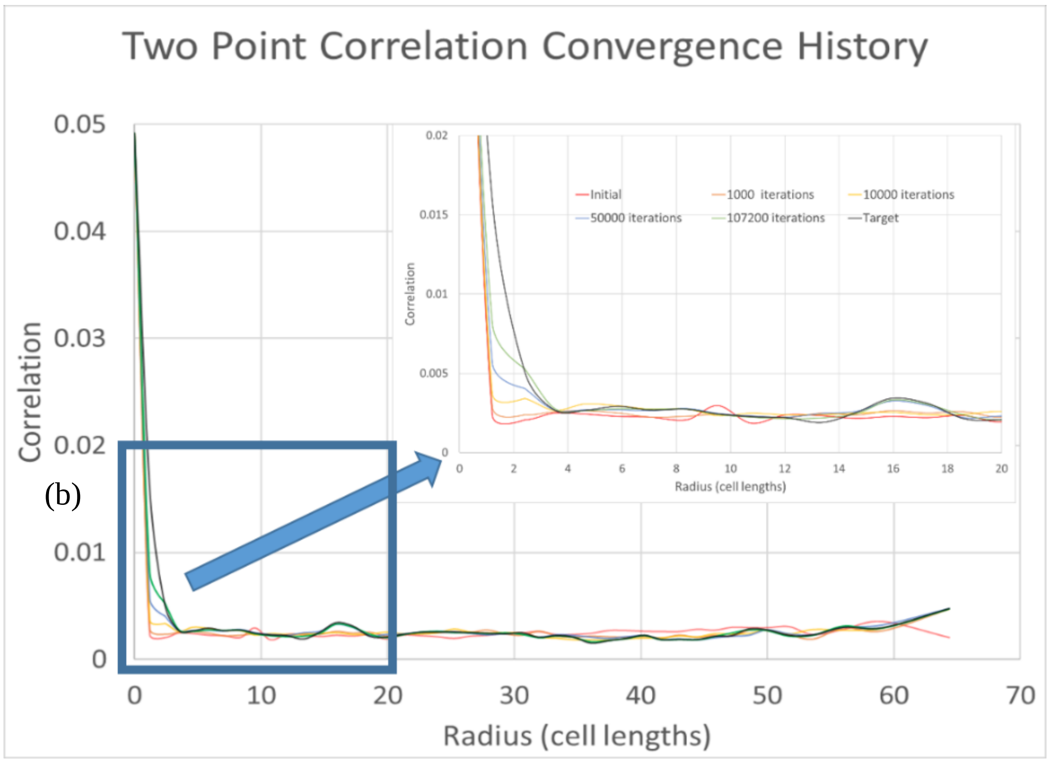
RECONSTRUCTION

After characterization of a 100×100 pixel image, we chose the one with the least phase fraction to continue the reconstruction part. In order to be computationally efficient we chose 50×50 image for analysis. Our results are shown in Figs. 4(a) and (b). As we can see from different stages of the simulation that as our code ran its course we could observe that the generated microstructures are successively going towards the target image. Especially, it could capture the trend at the front-side of the image quite easily. We also present a sample image of the microstructure that we generated for our work in Fig. 4(b). One can observe that the generated microstructure after 107200 iterations is some statistically similar to the target image.

IMPACT OF SA PARAMETERS ON STOCHASTICITY

One very important aspect of the simulated annealing algorithm is the probability of acceptance. It largely depends on the choice of the parameter *T*. Several trials are to be made before choosing the appropriate *T*. Here, to explore how the choice of this parameter affects the convergence, we considered two different case with initial *T* as 1000 and 3000. The results are shown in the Fig 5. From the Fig 5 (a) and (b) we can see that when *T* is very high the error is reducing at a much quicker rate. This is expected as at this stage, the probability of acceptance is very high. As the iteration runs on, according to the cooling schedule the temperature will go down and the corresponding errors go down as well. However, as we can see that at the later parts the error does not necessarily reduces in the same way as it did in the earlier parts because at this stage the probability of acceptance is low. Another interesting aspect of the Figure is that, since for *T* = 3000 the rate of cooling is higher and hence the simulation does not reach the tolerance value with the same number of iterations as for *T*= 1000. So this is another important parameter of simulated annealing algorithm. Combined with these two parameters and choice of tolerance values the algorithm takes varying amount of time to reach a converged value.

(a)



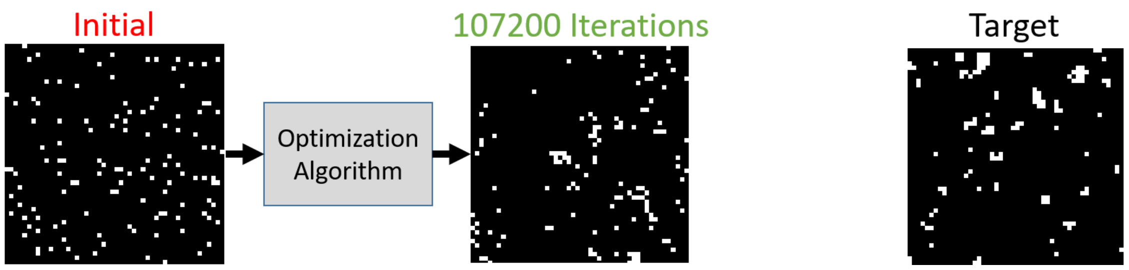


Fig 4. (a) Different stages of sample two phase microstructure reconstruction. (b) Representations of the initial and our output microstructure in reference to the target on.

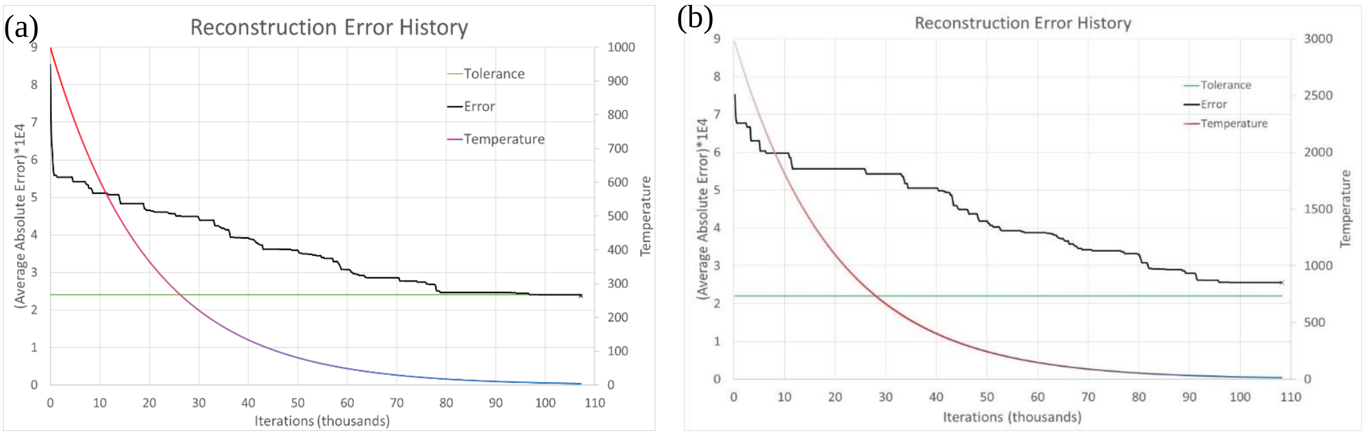


Fig 5. A comparison between the reconstruction error histories for (a) T=1000, and (b) T = 3000.

FUTURE RECOMMENDATIONS

There are some aspects that could be improved in implementing our algorithm. The key factors are discussed in following points

- The most time-consuming part of the simulation is finding the two-point correlation function. As our code does not consider the images as isotropic ones, it actually sees all the pixels in the image and calculate based on that. The two-point correlation function generally works well with isotropic images. So we can take advantage of that and only see in one direction to figure what the correlation will be in all other directions. This is a logical improvement that increase the computational efficiency of our code.
- Swapping of pixel randomly is not a robust method that guarantees you a feasible solution at the end of the iterations. We can swap pixels in an informed manner, like swapping the pixels with the most dissimilar neighbors, or choose to swap pixels at any particular direction only. The problem of random swapping is not restricted to only reduce computational efficiency but it also clusters the similar pixels at some locations which is not something one can expect in an isotropic materials. So the idea of informed swapping of pixel can also be implemented.
- Choice of SA parameters play a key role in increasing the computational efficiency of the process. Unfortunately, this section requires trial error to find out the suitable parameters. We could not perform this task due to lack of time but this is a point that can surely be improved.
- The code that we wrote in MATLAB® do not have the provision for parallel simulation or GPU implementation. This could be a big improvement considering how computationally expensive SA algorithms are.

CONCLUSIONS

The current project has been a worthwhile experience for both the members. Through this project we have been able to work on developing our own code to characterize and reconstruct microstructure which is an essential part of ICME. While developing codes, we ran into some issues that were previously unknown and unexpected and we think it is those experience are even more valuable. As for the project itself, our analysis show some important aspects of both two-point correlation function and SA algorithm. First of all, we found using two-point correlation makes much more sense computationally if the original microstructure that we are working on is an isotropic one. The two-point correlation function can be smoothened by the use of binning of radii which we did. Additionally, during the reconstruction process more weight should be put on the front-end of the correlation function. Through proper binning we achieved that. Although our code worked satisfactorily for identifying the structures, it took a long time which eventually made the entire process computationally expensive. We observed that considering the isotropy and also tuning the SA parameters can make the process significantly faster. Among the tuning parameters, the temperature *T* has a significant impact on the efficiency of the algorithm.

REFERENCES

1. Chen W, Yin X, Lee S, & Liu WK. (2010) A multi-scale design methodology for hierarchical systems with random field uncertainty. *ASME Journal of Mechanical Design*,132 (4):041006.
2. Bostanabad, R., Zhang, Y., Li, X., Kearney, T., Brinson, L. C., Apley, D. W., & Chen, W. (2018). Computational microstructure characterization and reconstruction: Review of the state-of-the-art techniques. *Progress in Materials Science*.
3. Committee on Integrated Computational Materials Engineering NRC. Integrated computational materials engineering: a transformational discipline for improved competitiveness and national security. National Academies Press; 2008.
4. Liu, Y., Greene, M. S., Chen, W., Dikin, D. A., & Liu, W. K. (2013). Computational microstructure characterization and reconstruction for stochastic multi-scale material design. *Computer-Aided Design*, *45*(1), 65-76.
5. Rozman MG, & Utz M. (2002). Uniqueness of reconstruction of multiphase morphologies from two-point correlation functions. *Physical Review Letters* 89(13):135501
6. Torquato S. (2002) Random heterogeneous materials: microstructure and macroscopic properties. New York: Springer.

Appendix A: MATLAB code for control function

```
% master script for two-point correlation MCR of binary image
% assumes
function [optImg,History] = MCR2ptcor(IMG,tolerance,rate,maxIter,cycle, Tinit)
binImg = IMG;
Star = TwoPIso(binImg);
phsfrac = Star(1,2);

% if there are more zeros than ones, then it will be faster to characterize
%   with respect to the zeros, so invert and then invert back at the end.
% invertFlag tracks if there are more zeros or ones
invertFlag = 1;
if phsfrac > 0.5
    binImg = mod(binImg+1,2);
    Star = TwoPIso(binImg);
    invertFlag = 0;
end

% create a randomized binary matrix with the same phase fraction as IMG
Ainit = zeros(length(binImg));
Ainit( randperm(numel(Ainit),ceil(phsfrac*numel(Ainit)))) = 1;
Sinit = TwoPIso(Ainit);
[Sopt,optImg,History] = simAF2(Star,Sinit,Ainit,tolerance,rate,maxIter,cycle,Tinit);

if invertFlag == 0
    optImg = mod(binImg+1,2);
end

plot(Star(:,1),Star(:,2),'-r');
hold on
plot(Sopt(:,1),Sopt(:,2),'-b');

end
```

Appendix B: MATLAB code for calculating two-point correlation function

```
function [S] = TwoPointCor(A)
[m,n] = size(A);

% rad is list of all radii
rad = zeros((m*n)*(m*n-1)/2,1);

% values is list for all these radii which are between phase of interest, 1
values = zeros((m*n)*(m*n-1)/2,1);

ind = 1;
for i = 1:m*n
    for j = i:m*n
        rad(ind) = sqrt( (mod((i-1),n) - mod((j-1),n))^2 ...
                         + (ceil(i/n) - ceil(j/n))^2 );
        values(ind) = A(i)*A(j);

        ind = ind+1;
    end
end

% Fin stores all the radii and "comparison" values, sorted by radius value
Fin = sortrows([rad values]);
p = size(Fin,1);

% unqR is the number of unqiue radii
unqR = size(unique(Fin),1);

% radii is the list of all the radius values
% count is the number of each radius in radii
% sum is the sum of the "comparison" values for each radius in radii
radii = zeros(unqR,1);
count = zeros(unqR,1);
sum = zeros(unqR,1);

count(1) = 1;
radii(1) = Fin(1,1);
sum(1) = Fin(1,2);
ind = 1;

for i = 1:p-1
    if Fin(i,1) == Fin(i+1,1)
        sum(ind) = sum(ind) + Fin(i+1,2);
        count(ind) = count(ind) + 1;
    else
        ind = ind+1;
        radii(ind) = Fin(i+1,1);
        radii(ind-1) = (radii(ind-1) + Fin(i,1))/2;
        sum(ind) = Fin(i+1,2);
        count(ind) = 1;
        sum(ind) = sum(ind) + Fin(i+1,2);
        count(ind) = count(ind) + 1;
    end
end
radii(ind-1) = (radii(ind-1) + Fin(p,1))/2;

prob = sum./count;

% phsfrac2 is one criteria for the max radius
phsfrac2 = (sum(1)/count(1))^2;

% find radius that has been above phsfrac2 for 50 radii
maxRind = 0;
low = 0;
higher = 0;
for i = 1:1:unqR
    if prob(i) <= phsfrac2
        low = 1;
        maxRind = i;
        higher = 0;
    else
        higher = higher + 1;
        if (low == 1 && higher >= 50)
            break;
        end
    end
    %disp([i,low,higher,maxRind])
end

% radBin is average of radii in a single bin
% probBin is averaged probability over all radii in a single bin
radBin = zeros(ceil(0.5*(1+sqrt(1+8*maxRind)))-2,1);
sumBin = 0*radBin;
countBin = 0*radBin;

% binSize is number of radii values to be averaged in current bin
% binNum is number of radii stored in current bin
j = 1;
for binNum = 1:1:( floor(0.5*(1+sqrt(1+8*maxRind))) - 1)
    radBin(binNum) = radii(j);
    for binSize = 1:1:binNum
        sumBin(binNum) = sumBin(binNum) + sum(j);
        countBin(binNum) = countBin(binNum) + count(j);
        j = j + 1;
    end
    radBin(binNum) = (radBin(binNum)+radii(j-1))/2;
end
probBin = sumBin./countBin;
S = [radBin,probBin];
end
```

Appendix D: MATLAB code for simulated annealing optimization

```
% simAF2 is a simmulated annealing algorithm that attempts to reconstruct
%   a binary image with the same two-point-correlation as a specified target
function [Sopt,Aopt,History] = simAF2(Star,Sinit,Ainit,tolerance,rate,maxIter,cycle,Tinit)
tic
% simAF2 takes as input the elements below
%   STar is the target correlation of image
%   A is the initial random binary image matrix
%   S is the initial correlation of A
%   tolerance a window of optimality for the solution
%   rate is the rate of decrease of T with respect to iterations
%       should be a value beween 0 and 1, higher is faster decrease.
%   cycle is the number of iterations between checking for optimality
%   maxIter is the maximum number of iterations, it is adjusted to the
%       nearest multiple of cycle
maxIter = cycle*floor(maxIter/cycle);

% Sopt is the best correlation binning found
% Aopt is the binary matrix corresponding to Sopt
% Scur is the current correlation binning
% Acur is the binary matrix corresponding to Scur
Sopt = Sinit;
Aopt = Ainit;
Scur = Sinit;
Acur = Ainit;

History = Sinit;

% Errcur is the sum of squared error (SSE) between Scur and Star
Errcur = 0;
SC = 10^8;

% calculating the SSE between Star and S, initial error
for i = 1:(min(length(Star),length(Sinit)))
    Errcur = Errcur + abs(Star(i,2) - Sinit(i,2)) ;
end
Errcur = SC*Errcur/(min(length(Star),length(Sinit)));

% Erropt is the SSE between Sopt and Star, want to minimize to zero
Erropt = Errcur;

% k and T help define the probability of accepting a less optimal solution
%k = 1.3806e-23;
T = Tinit;

% flag monitors if an optimum within tolerance is found
flag = 1;
numSwap = 10;
for j = 1:maxIter
    tic
```

```
for i = 1:cycle
    disp('error')
    disp(Errcur*1e-4)

    % swaPixel swaps numSwap pixels in Acur to make a new matrix, Atemp
    Atemp = swaPixel(Acur,numSwap);
    % TwoPIso calculates the correlation for Atemp
    Stemp = TwoPIso(Atemp);

    % Errtemp is the SSE between Scur and Stemp
    Errtemp = 0;
    for b = 1:(min(length(Star),length(Stemp)))
        Errtemp = Errtemp + abs(Star(b,2) - Stemp(b,2));
    end
    Errtemp = SC*Errtemp/(min(length(Star),length(Stemp)));
    % if the SSE of Atemp is larger than the SSE of Aopt
    %   there is a probability that Atemp is only our new Acur
    % elseif the error of Atemp is only larger than the error of Aopt
    %   then Atemp is only our new Acur
    % otherwise (SEE of Atemp is less than or equal to Aopt SSE)
    %   Atemp is both our new Acur and Aopt
    if Errtemp > Errcur
        % calculate a probability that we still
        %   accept Atemp as our new Acur
        prob = exp( -(Errtemp-Errcur)/(T));
        disp('prob')
        disp(prob)
        r = rand();
        if prob > r
            Acur = Atemp;
            Scur = Stemp;
            Errcur = Errtemp;
        end
    elseif Errtemp > Erropt
        Acur = Atemp;
        Scur = Stemp;
        Errcur = Errtemp;
    else    %condition: Errtemp <= Erropt
        Aopt = Atemp;
        Acur = Atemp;
        Sopt = Stemp;
        Scur = Stemp;
        Errcur = Errtemp;
        Erropt = Errtemp;
    end
end
toc
% T decreases every cycle, making it less likely to accept succesively
%   worse Stemp/Atemp as our new Scur/Acur as iterations increase
```

```
    T = T*(1-rate);
    if numSwap > 1
      numSwap = numSwap - 1;
    end

    History = [History, Sopt(:,2)];

    % if the current Sopt is within tolerance of Star
    %   then accept current Aopt and Sopt as our optimum and stop looking
    % otherwise
    %   keep looking for another cycle
    if (Erropt*1e-4) < tolerance
        flag = 0;
        break;
    end
end

% determine whether the current Sopt/Aopt are an optimum within tolerance
%   or if no Sopt/Aopt within tolerance was found within the max iterations
if flag == 0
    disp('Solution within tolerance found');
end
if flag == 1
    disp('Max iterations reached before solution within tolerance found');
    disp('Best solution found');
end
end
```

Appendix C: MATLAB code for Pixel Swapping

```
function [In] = swaPixel(I,numswap)
[m,n]=size(I);
Zind = zeros(n,1); %creates index values storage
Vind = zeros(n,1);
ind1 = 1;
ind0 = 1;
for i=1:m
    for j=1:n
        if I(i,j) == 0
            Zind(ind0) = sub2ind([m,n],i,j);
            ind0=ind0+1;
        else
            Vind(ind1) = sub2ind([m,n],i,j);
            ind1 = ind1+1;
        end
    end
end

for i = 1:numswap
    tempz = randi(numel(Zind)); %gets random index of zero element
    temp1 = randi(numel(Vind));
    p = I(ind2sub([m,n],tempz));
    I(ind2sub([m,n],tempz))=I(ind2sub([m,n],temp1));
    I(ind2sub([m,n],temp1))= p;
end
In = I;
end
```