

## **Module NED2: Non Equilibrium Dynamics in Growth and Surface Processes**

Prerequisite: Module C2 (note that NED1 is not a prerequisite for this module)

References: Mostly you will need a good MATLAB reference. You may also wish to read up on fractals and diffusion limited aggregation. The library has several introductory books on fractals.

MODULE NED2

Name:

Term:

Problem	Max	1 <sup>st</sup>	2 <sup>nd</sup>	Final
1	5			
2	5			
3	5			
4	5			
5	5			
6	10			
7	10			
8	5			
9	10			
10	5			
11	10			
12	10			
TOTAL	85			

## ***Part 1: The growth of a fractal***

A solid grows by aggregation of smaller particles (sometimes molecular or atomic in size). A crystalline solid is one in which the atoms form a periodic array, so that once you know the location of a few atoms, you can predict (based on the pattern) the location of all of the other atoms in the crystal. Crystal formation is generally a slow process in that when atoms arrive at the surface of the crystal, they need time to migrate around along the surface to fill in voids. Growth, in fact, often occurs at steps formed on the surface, allowing a single layer to be completed before a new layer commences. The migration of atoms on the surface during crystal growth can be viewed as a process in which the atoms are seeking to minimize their potential energy. That is, the atoms are seeking their equilibrium positions.

Other forms of solid growth happen too rapidly for atoms to find the potential energy minima. This type of growth often results in more irregular and therefore more complex atomic arrangements. One such growth process is called “diffusion limited aggregation” (DLA). In this process, particles move randomly (due to Brownian motion) until they encounter a particle of the same type. For instance, a molecule in the gas phase may wonder around until it happens to hit the surface of a solid composed of similar molecules. When the encounter occurs, the molecule sticks at that site and does not migrate any further. In this sense, the aggregation of particles is limited by the diffusion process.

You will build a program to model a very simple DLA process. This program may be the longest MATLAB program you have written so far. I'll give you a major chunk of it, and you will be responsible for understanding the code and modifying and extending it as needed. Here is a simple overview of the code: We will begin with a two dimensional array ( $A$ ) representing allowed locations for the particles in our model. A zero in an array entry indicates no particle is there; a one indicates the site is occupied. Double occupancy is not allowed. We will start with a smooth solid surface,

indicated by setting the bottom row of the array equal to one. A new particle is introduced at a random location and allowed to wander around randomly until it is adjacent to an occupied space. When that happens, the particle becomes attached to the solid and that array entry is set equal to one. A new particle is allowed to enter and attach, and so on, until we have grown a rather odd looking solid.

First, create an array of all zeroes:

```
>>A=zeros(20,20,'int8');
```

This command creates a 20x20 array. The last argument, 'int8', is optional. It stores this array as an integer, rather than a real number. In principle this should reduce the storage space and speed up calculations a little, an aspect that would be useful if we were to work with substantially larger arrays.

Next, create the smooth solid at the base of the array, on which we will grow our new solid:

```
>>A(20,:)=1
```

This sets all columns (as indicated by “:”) of row 20 equal to 1. I have left the semicolon off the end so that you can verify that your array is all zeros except for a single line of ones at the bottom.

The next step is to allow a particle to execute a random walk (that is to wander around randomly) until it encounters the solid and attaches itself. We will do this by picking a starting point near the top of our array (row 3) in a random column. The particle checks to see if it is next to an occupied site. If it is, then it attaches itself there. If not, it takes a random step and repeats the process. What if the particle tries to go outside the array? It cannot go past row 20 since we have a line of particles in that bottom row to stop it. If it tries to go above row 1, we will count that as an escaped particle and start over with a new particle. If it tries to go to column 21, we will allow it to wrap around and enter at the same row on column 1. Likewise, if it tries to go to column zero, we will direct it to column 20 of the same row. These conditions on the vertical boundary are

known as “periodic boundary conditions”. The conditions are equivalent to saying that each column,  $j$ , is equivalent to column  $j+20$ . Another way of looking at this is that it is like the array is wrapped around a cylinder, causing columns 1 and 20 to be adjacent to each other.

**Problem 1:** In preparation for the next programming phase, explore the “mod” function. Write out in your own words what it does and show some sample results that illustrate its properties.

**Problem 2:** This is not a programming problem in the sense that you need to create something. Rather, you need to enter the following code, verify that it runs properly, and explain what each section of the code does. Although retyping the code is tedious, I think that is a useful way for you to pay attention to and understand the syntax of each statement. You should turn in a written explanation of the function (you can, for instance, print out a copy and annotate it by hand) that makes it clear you understand what it is doing and why. You should also verify that your function actually works (I’ll take your word for it). Run it several times. The screen will display the coordinates of the wandering particle. Sometimes the function will end with the particle wandering out of bounds (i.e., floating up too high). Other times the function ends with an attachment. The display will also tell you how many moves the particle made in an attempt to attach. Here is the function:

```
function attach=addone(A)
width=20;
startrow=3;
startcol=int8(rand(1)*width+0.5);
%is it occupied?
if A(startrow,startcol)==1
    full='full'
    return
end
%set the starting point
atomc=startcol;
atomr=startrow;
%keep track of number of tries and set landing indicator
tries=0;
```

```

land=0;
%begin movement process and attachment check
while (tries<2000)&(land==0)
    %find neighbors
    atomcp1=mod(atomc,width)+1;
    atomcm1=mod(atomc-2,width)+1;
    atomrm1=atomr-1;
    atomrp1=atomr+1;
    %are any of its neighbors occupied?
    if A(atomr,atomcp1)==1
        land=1;
    end
    if A(atomr,atomcm1)==1
        land=1;
    end
    if A(atomrp1,atomc)==1
        land=1;
    end
    %atomrm1 is a special case--if atomr=1, there is no array at
atomr-1
    if atomrm1>0
        if A(atomrm1,atomc)==1
            land=1;
        end
    end
    %if an occupied neighbor was found, report it and leave
    if land==1
        attach=[atomr,atomc]
        tries
        return
    end
    %if no landing took place, let the atom move to a new spot
    delrow=0;
    delcol=0;
    if(rand(1)<0.5)
        delrow=2*int8(rand(1))-1;
    else
        delcol=2*int8(rand(1))-1;
    end
    atomc=mod(atomc+delcol-1,width)+1;

```

```

atomr=atomr+delrow;
[atomr,atomc]
if atomr==0
    message='out of bounds; no attachment'
    attach=[0,0];
    tries
    return
end
tries=tries+1;
end
message='maxed out on tries'
end

```

For this problem, submit a copy of your m-file that shows additional comment statements (they begin with “%”) that you have added to help remind yourself how the program works.

**Problem 3:** First, in your addone function, delete the line that displays the location of the wondering particle. Next, reinitialize your array (all zeroes except the bottom row). Then enter the following into your command window:

```

>> for counter=1:200
hold=addone(A);
if hold(1)*hold(2)>0
A(hold(1),hold(2))=1;
end
end

```

This will result in 200 attempts at adding a particle being made. Have the computer display the array A, and study it to see if you can understand how all the ones are connected to the base. Remember that the array is wrapped around a cylinder (column 1 and 20 are neighbors). To get a feel for the efficiency of the growth process, have the computer count the number of ones in the array, excluding the bottom row, and compare this to the 200 attempts at adding a particle. Repeat this process 4 more times so that you have a total of 5 efficiencies calculated. For this problem, turn in the programming line or lines you used to count ones and your 5

values for efficiency. Occasionally you may get an “array full” message before the 200 tries are complete. This will occur when one of the columns in the uppermost rows is occupied (not when the entire array is really full). If you look at the variable list in your workspace menu, you should see the value of “counter” when the array reached capacity. Use that in the denominator of your efficiency calculation.

**Problem 4:** Write a function, `create(width, maxcnt)`, which creates an array of dimensions `width x width` and tries to attach “maxcnt” particles to it. That is, this function will allow you to execute the procedures outlined in Problem 3 in a single line. Note that you will want to modify the “addone” function also to allow for variable width. You might also want to suppress the screen output in that function, allowing it to run a bit faster. For this problem, submit a copy of your modified “addone” function and your new “create” function, as well as a printout of the array for the case of `width=30` and `maxcnt=500`. If you are having trouble printing out the array in a manageable form, try this:

```
>> dlmwrite('savedarray.doc',A, ' ')
```

This creates a file called `savedarray.doc` consisting of the elements of array `A` separated by a space. If I had wanted a comma separating the elements, the last bit would have been `','` rather than `' '`. You may have to search to find where MATLAB has saved the file, but once you find it, you can open it as a word document. You can also try the `xlswrite` command to export `A` to an Excel file, but I did not have any luck with that on my Mac. Before you leave this MATLAB session, try to complete Problem 5.

**Problem 5:** It would be nice to print out the array in a more representational format. Create the following new function:

```
function done=Aplot(A,width)
```

```
done=0;
```

```
points=0;
```

```
for i=1:width
```

```
    for j=1:width
```

```
        if A(i,j)==1
```

```
            points=points+1;
```

```
            xsave(points)=j;
```

```
            ysave(points)=i;
```

```

    end
  end
end
plot(xsave,ysave,'rs','MarkerSize',10)
axis([0 width+1 0 width+1])
done=1
end

```

The first part of the function identifies the coordinates of all occupied sites in your array. The second part plots each of these occupied sites as a square ('rs') of marker size 10. Depending on the size of your array, you may want to adjust the marker size. Ideally adjacent squares will touch but not overlap.

Test this function out by entering into your command window:

```
>> done=Aplot(A,30)
```

Submit a printout of the plot.

**Problem 6:** Run the “create” function for a width of 40 and maxcnt=2000. Be patient—this may take a few minutes. If you get a message indicating the array is full, try again with the same parameters until an array is successfully completed. Plot the result using Aplot. You should observe a highly segmented tree-like structure. What is the dimension of this object? It lies on a two dimensional surface, but clearly it does not efficiently fill up an area. This structure is an example of an object with a fractional dimension, or a “fractal”. There are a variety of ways to calculate a fractal dimension, and they do not necessarily give the same result. That is, the answer to the question, “What is its dimension?” is not necessarily unique. In this problem, you will learn how to define the “box counting” dimension. Suppose you have a line segment of length  $L$ . If we try to cover this line segment as efficiently as possible with squares (boxes) of side  $\delta$ , then we would require a total of  $L/\delta$  boxes. If instead we have an area of dimension  $L \times L$  to cover, then we would require  $(L/\delta)^2$  boxes to cover it. Generalizing this to  $D$  dimensions, if we have a  $D$ -dimensional hypercube, we would require  $(L/\delta)^D$  boxes to cover it. In this context, “hypercube” is just a generalized way of referring to a line segment, a square, a 3D cube, a 4D cube, etc. Letting  $N$  represent the number of boxes required to cover our object,

$$N = \left(\frac{L}{\delta}\right)^D$$

$$\log N = D \log L + D \log\left(\frac{1}{\delta}\right)$$

This suggests a plot of  $\log(N)$  vs  $\log(1/\delta)$  will produce a straight line whose slope is the dimension,  $D$ . Now for a random structure, as we have created in the array  $A$ , the plot will not necessarily be linear. The box counting dimension is defined as the slope of the plot in the limit  $\delta$  approaches zero:

$$D_B = \lim_{\delta \rightarrow 0} \frac{d(\log N)}{d(\log \frac{1}{\delta})}$$

Print out a plot of your array (created from Aplot). There will likely be one, dominant tree in the plot. You will estimate the box counting dimension for that tree. Let's take the unit of measure to be the size of one entry in the array. That is, if we ask how many boxes of size 40x40 are required to cover the tree, the answer is one since a box that size covers the whole array. In this scheme we do not use fractional boxes. The question is what is the minimum number of boxes of a given size required to cover the whole tree. Now estimate how many boxes of size 10x10 are required to cover your tree. Repeat with boxes of size 8x8, 6x6, 4x4, 2x2, and 1x1. The last one is easy since it is just the number of squares in your tree. Plan on turning in your printed plot with the 6x6 boxes sketched in so that I can see how you carry out this procedure. Fill out a table of the following form:

Box size ( $\delta$ )	Boxes to cover (N)
40	
10	
8	
6	
4	
2	
1	

Now use Excel to plot  $\log(N)$  on the vertical axis against  $\log(1/\delta)$  on the horizontal axis, and extract a slope. Repeat this process for two

more arrays. You should submit the following:

- a. The “Aplot” version of your first array with sketches showing how you cover the main tree with 6x6 boxes.
- b. The Aplot version of the other two arrays, with the main tree identified.
- c. The table for each of the three arrays.
- d. The Excel plots you used to find  $D_B$  for each array and the value of  $D_B$ .
- e. A discussion of your results for the box counting dimension.

**Problem 7:** This problem will be an exploration on your part. There are other rules we could have used in modeling this growth process. For instance, we could bias the random walk of the particle so that it is more likely to step down towards the solid. Or we could allow an attached particle to hop along the solid once or twice (as long as it stays attached. Another possibility is to allow a wandering particle to bounce off a site with a certain probability before coming to a final resting spot. Create your own new rule, modify your program and then repeat the procedure you followed in Problem 6. Submit the same type of documentation you turned in for Problem 6, along with a description of your modified growth rules and a copy of your addone function. Does your modified growth rule change the nature of the fractal?

## **Part 2: Surface Processes**

The previous set of problems investigated the growth of a solid when the only relevant process is attachment of new particles. In many situations, both with and without growth, other processes play an important role in surface dynamics. Among these processes are the migration of particles along the surface, sometimes modeled as atoms hopping from one site to a neighboring site, and desorption, the departure of particles on the surface. We will focus on a simple model for the desorption process.

First we consider a model for the potential energy experienced by a particle near the surface of a solid. In this context, “particle” will be a generic term for either an atom or a molecule. There are a variety of forms one can choose from. Since force is the derivative of the potential, we need a function that is flat far away from the surface, indicating that the particle experiences no binding force at that distance. On the other hand, the potential should have a hard core repulsion near the surface, under the assumption that the particle cannot penetrate the surface. Finally, the potential should have a local minimum, representing the equilibrium position of the particle when it is bound to the surface. To keep things manageable, we will consider just a one dimensional potential. The Lennard-Jones potential, discussed in Module C2, satisfies these conditions. For variety, we will use a different one, the Morse potential. You will see this written different ways in the literature. One form is  $V(x) = 2ma^2\omega^2(e^{-2x/a} - e^{-x/a})$  where  $m$  is the mass of the particle and  $\omega$  and “ $a$ ” represent parameters that can be adjusted to help the model fit a particular system.

**Problem 8:** Study the Morse potential properties. Specifically,

- Select appropriate dimensionless representations for the position and the potential energy and plot the potential energy as a function of position. Show your transformation into dimensionless variables and submit the plot.
- Determine the minimum in the potential energy (using calculus-based techniques). Find both  $x$  and  $V(x)$  at the minimum.
- Determine the frequency of small amplitude oscillations about this minimum.

**Problem 9:** Motion in the presence of a Morse potential.

- a. Calculate the force function associated with the Morse potential.
- b. Write an m-file for this force function and use it to solve for the position as a function of time. Choose suitable dimensionless variables to replace  $x$  and  $t$ . Take as your initial condition the mass being released from rest at some point away from the equilibrium position. Review the second half of Module C2 if necessary. Submit a table of values for the period of oscillation as a function of the starting position, showing that small amplitude oscillations have a period as predicted by Problem 8 and that large amplitude oscillations deviate from that value. Note that this work mirrors some of that found in Module C2. If you saved your work from that, you may re-use it.

**Problem 10.** The Fluctuation-Dissipation Theorem. An atom absorbed on a surface can be modeled as experiencing two other forces beyond that which represents its interaction with a static surface. The first force is a dissipation force. If you drop a rock onto a soft mattress, the fact that the mattress has some flexibility allows it to absorb some of the rock's kinetic energy. The energy initially at least gets transformed into vibrational energy in the mattress. A common form for this dissipation term is

$$\vec{F}_{diss} = -\beta\vec{v}. \quad (10.1)$$

If this is the only additional force an absorbed atom experienced, then it would shortly lose all of its energy to the solid. In fact, the thermal energy in the solid causes random movement in surface atoms on the solid. This random movement gives rise to the adsorbed atom experiencing a random force,  $\vec{F}_{rand}(t)$ . Consider the value of the random force at two different times,  $t_1$  and  $t_2$ . The product of these two forces will itself be a random number, sometimes positive and sometimes negative. If we were to average this product over a wide range of values of the two times, we would expect to get zero, unless of course if we set the two times equal to each other. Mathematically we express this as

$$\langle F_{rand}(t_1)F_{rand}(t_2) \rangle_{ave} = q\delta(t_1 - t_2). \quad (10.2)$$

$\delta(x)$  is known as the Dirac Delta Function. It has the following properties:

$$\delta(x) = 0 \quad \text{if } x \neq 0$$

$$\delta(x) = \infty \quad \text{if } x = 0$$

$$\int_{-\infty}^{\infty} f(x)\delta(x-a) = f(a)$$

For simplicity, I am assuming a one dimensional problem so that the vector notation may be dropped. The random force tends to pump energy into the adsorbed atom while the dissipation term takes it away. The ratio of these forces gives a feel for how much energy the adsorbed atom can keep and thus is connected to its temperature. Mathematically, this relationship is known as the Fluctuation-Dissipation Theorem:

$$q = 2k_B T \beta \quad (10.3)$$

where  $k_B$  is Boltzmann's constant and  $T$  is the temperature in Kelvin.

To program in these thermal effects, you will need to update the force function in your differential equation m-file:

$$F_{new} = F_{interaction} - \beta v + F_{rand}(t) \quad (10.4)$$

where  $F_{interaction}$  is the force function you derived in the previous problem. Take the random force to have two possible values, "+kick" or "-kick", with the value being re-selected every  $\Delta t = \text{"kicktime"}$ .

- a. Provide a convincing argument that for the above-defined random force, it is reasonable to take  $q = \text{kick}^2 * \text{kicktime}$ , where  $q$  is defined in equation (10.2).
- b. The programming is a little tricky, since we need to allow the force in the m-file to vary (due to the random kick). I solved this problem by creating a second m-file that calls the original one. The second file generates the random force and passes it to the first one, with the value of the random force and beta passed using the same array that stores the position and the velocity. You are free to use my code or to write your own code, just make sure you understand what MATLAB is doing.

If you understand these files well enough, you should know how to call the function from your command window and plot the results. In particular, choose, the temperature equal to 0.05 and the

maximum time equal to 100, and plot the position as a function of time. Be patient; once your program is working properly it will take a while to run this long an integration.

Repeat the integration for a temperature of 3.0. Submit printouts of your m-files, your plots at the two different temperatures, and a discussion of the difference between those two plots and how that relates to the temperature difference.

.....  
My Code:

```
function data=thermal(temp, maxtime)
%need to scale the kick with kicktime
ystart=1;
beta=2;
time=0;
kicktime=0.1;
kick=(2*temp*beta/kicktime)^0.5;
index=1;
data(1,index)=0;
data(2,index)=ystart;
data(3,index)=0;
while time<maxtime
    frand=kick;
    if(rand(1)<0.5)
        frand=-kick;
    end
    [t,z]=ode45(@newexp,[0,kicktime],[data(2,index),data(3,index),beta,
    frand]);
    index=index+1;
    mlength=length(t);
    time=time+kicktime;
    data(1,index)=time;
    data(2,index)=z(mlength,1);
    data(3,index)=z(mlength,2);
end

function dydt=newexp(t,y)
dydt(1,1)=y(2);
```

```
dydt(2,1)=2*(2*exp(-2*y(1))-exp(-y(1)))-y(3)*y(2)+y(4);  
dydt(3,1)=y(3);  
dydt(4,1)=y(4);
```

.....

**Problem 11. Energy Analysis.**

- a. You generate output in the form of a time, position and velocity array. Use this information to write a new m-file that will compute the average kinetic energy and the average potential energy of the molecule during your integration time period. That is, average over the whole array. For this part, submit a copy of your new m-file.
- b. Your m-file presumably returns numerical values for the average energy. Discuss what parameters you need to multiply these values by to get quantities with appropriate energy units. You may wish to return to the problem in which you reduced the equations to dimensionless form.
- c. In your program, you enter a numerical value for temperature. Discuss how that can be converted into a value with appropriate temperature units.
- d. For each of the following temperatures (in dimensionless form): 0.01, 0.05, 0.10, 0.15, run your program out to an integration time of 500 (have something else to work on while these are running). In each case, calculate the average potential energy and the average kinetic energy. Submit a table with your results.
- e. The equipartition theorem states that for a system in equilibrium with a term in its energy equation of the form  $\frac{1}{2}\Omega\chi^2$  where  $\Omega$  is a constant and  $\chi$  is a position or velocity coordinate, then  $\left\langle \frac{1}{2}\Omega\chi^2 \right\rangle = \frac{1}{2}k_B T$ . Does this result hold for your kinetic energy calculation?
- f. What would the equipartition theorem predict for low temperatures, when your molecule experiences mostly small amplitude oscillations about the minimum in the potential? Compare this prediction to your data.

**Problem 12.** I modified my “thermal” function as follows:

```
function data=thermal(temp, maxtime)
ystart=0.69;
beta=2;
time=0;
kicktime=0.1;
kick=(2*temp*beta/kicktime)^0.5;
index=1;
data(1,index)=0;
data(2,index)=ystart;
data(3,index)=(temp)^0.5*0;
%modifications for desorption calculation
xlimit=8;
while time<maxtime
    frand=kick;
    if(rand(1)<0.5)
        frand=-kick;
    end
    [t,z]=ode45(@newexp,[0,kicktime],[data(2,index),data(3,index),beta,
frand]);
    index=index+1;
    mlength=length(t);
    time=time+kicktime;
    data(1,index)=time;
    data(2,index)=z(mlength,1);
    data(3,index)=z(mlength,2);
    %modifications for desorption calculation
    if data(2,index)>xlimit
        data(1,1)=time;
        data(2,1)=data(2,index);
        data(3,1)=data(3,index);
        time=maxtime;
    end
end
end
```

a. Discuss how this works to follow the particle until it reaches  $x=8$ , at which point it is so far away from the potential minimum (i.e.

from the surface) that I treat it as desorbed. You should submit a discussion of all of the changes in this version as compared to your previous version. Where is the desorption time stored?

b. Test your program by running at  $T=2.0$  with  $\text{maxtime}=2000$ . The execution of this program should only take a minute or two. Once you are convinced it is operating properly, write a “for” loop in your command window that calls “thermal” 50 times at  $T=2.0$  and stores the desorption times in an array called “events”. Calculate the average desorption time for all 50 events. You will notice considerable variation in your data. Run your loop again and calculate the average desorption time again. This will give you a feel for the amount of uncertainty in your average time. For this part, submit a copy of the program loop that you wrote in your command window and the results for the desorption time averages for your two 50-event runs.

c. The following table of values shows what I obtained for average desorption times. My goal is to have students like you over the years fill in additional data on this table so that we can collectively produce a good plot of the desorption time vs. temperature.

Temperature	Time
2.0	47.9
1.5	59.9
1.0	78.1
0.5	162
0.4	297

In this part, you will perform 3 more 50-event runs, using temperatures other than those listed above. One temperature should be in the interval (1.5,2.0), one in the interval (1.0,1.5) and one in the interval (0.5,1.0). Obviously, your lowest temperature run will take longest since that is when the particle is slowest to desorb. Depending on the temperature and the computer, it may take an hour or more, so have something else with you to work on while it is running. For this part, submit a table showing the above values as well as your new values, and then submit a plot of the average desorption time (vertical axis) vs. temperature (horizontal axis).

d. The desorption rate,  $R$ , is the inverse of the average desorption

time. One theory suggests that the desorption rate should obey the Arrhenius Law  $R = Ae^{-B/k_B T}$  where A and B are constants. This equation suggests that a plot of  $\ln(R)$  vs.  $(1/T)$  should produce a straight line of slope  $(-B/k_B)$  and intercept  $\ln(A)$ . Using all of the data from part (c), construct this plot (known as an Arrhenius plot) and determine the slope and the intercept.