

Exercise Lecture XIII

Diffusion Limited Aggregates, fractal models of surface growth, percolation

1. Diffusion Limited Aggregates (DLA)

- (a) Write a program to generate DLA on a square lattice. See for instance the code `dla2d.f90`. Choose each walker starting randomly at a distance $R = R_{max} + 2$ from the center, where R_{max} is the maximum distance of the particles already aggregated in the cluster from its origin. To save time, eliminate the walker that go too much far away, e.g. that reach a distance equal to $2R_{max}$ from the center (“killing circle”). Choose $L=31$. Try to color in a different way the sites according to the order of aggregation (e.g. after 20 particles aggregated change color). Which are the last aggregated? Which are the former?
- (b) At $t = 0$ we have 4 perimetral sites with a probability $p_i=1/4$ of being occupied. After having occupied one of them, we have 6 perimetral sites which have different occupancy possibility: two of them have $p_i=2/9$ and the other 4 have $p_i = 5/36$. Verify with a Monte carlo simulation.
- (c) The efficiency of the algorithm can be improved considering displacements with variable length, the longer the distance from the center, the longer is the step length. For instance, if the walker is at distance $R > R_{max}$, consider a length displacement $R - R_{max} - 1$ (if it is > 1), whereas consider a unitary displacement if the walker is close to the cluster already grown.
- (d) Generate some DLA clusters and calculate their fractal dimension, which should be $d = 1.66$ (see: Witter et al., Phys. Rev. Lett. 47, 1400 (1983)).

2. Fractal growth of surfaces

Consider the *Eden model* to generate a corrugated surface. The algorithm is:

- (a) choose randomly a lattice site and occupy it. The *nearest neighbor sites* of the occupied site (i.e. 4 sites in case of a square lattice) are the *perimetral sites*.
- (b) choose randomly a *perimetral site* and occupy it. When occupied, it is no longer a *perimetral site*: update the list of *perimetral sites* with the new ones. Repeat from (1).

The code `eden.f90` is proposed as a draft here (the suggestion is to modify it, it has several “print” for checks...)

Consider a simple model where the surface is initially (at time $t = 0$) an horizontal line of L occupied sites. The growth is along the vertical direction.

According to the Eden model, choose a *perimetral site* randomly and occupy it. For the initial configuration of our surface, at time $t = 0$ the *perimetral sites* are the horizontal line of empty sites adjacent to the line of occupied sites. The average height of the cluster is:

$$\bar{h} = \frac{1}{N_s} \sum_{i=1}^{N_s} h_i$$

where h_i is the distance of the i surface site from the initial line, and the sum is over all the surface sites N_s .

The deposition of a particle corresponds to the increment of time t by one. Study how the *roughness* w of the surface change with time, where w is defined as:

$$w^2 = \frac{1}{N_s} \sum_{i=1}^{N_s} (h_i - \bar{h})^2,$$

($w=0$ for a planar surface). w depends on L and t . Initially w increases with time:

$$w(L, t) \sim t^\beta$$

β measures the increasing in time of the correlations in the vertical direction. Given a characteristic time, the length for the correlation of the fluctuations is comparable with L , and the roughness w reaches a limiting value depending only on L . We can write:

$$w(L, t \gg 1) \sim L^\alpha,$$

where α measure the corrugation.

- (a) Consider a 1D surface growing over a line of $L=100$ sites and apply the Eden model. Consider x the horizontal index, i.e. the label of the columns, and h_x the height (max. distance of a perimetral site from the substrate). Use PBC in the horizontal direction. We call *surface sites* those *perimetral sites* with maximum h for a given x . Try to visualize the growth in time, with the evidence of the occupied, perimetral and surface sites.
 - a) is the surface well defined?
 - b) where are most of the perimetral sites?
 - c) if we choose *all perimetral sites* as *surface sites*, is something changing?

- (b) Plot $w(t)$ as a function of t for $L=32, 64, 128$ and estimate the exponents α e β .
 - a) Which kind of plot is it convenient to do?
 - b) Does w increase initially with a power law? If yes, estimate β .
 - c) Is there a characteristic time (depending on L) for w to reach an asymptotic value?
 - d) Can you estimate α ? (you should find $\beta = 1/3$ and $\alpha = 1/2$).
- (c) The dependence of w on L and t can be summarized with the law:

$$w(L, t) \approx L^\alpha f(t/L^{\alpha/\beta})$$

where : $f(x) \approx x^\beta$ for $x \ll 1$ and $f(x) = \text{constant}$ for $x \gg 1$

- a) Using for α and β the best estimates obtained in the previous point, verify the law plotting $w(L, t)/L^\alpha$ as a function of $t/L^{\alpha/\beta}$ for the different values of L considered. b) Repeat using instead the exact result, $\beta = 1/3$ and $\alpha = 1/2$. You should find a universal curve (i.e. the *same* curve using the scaled variables for different values of L)
- (d) *Random Deposition* In the Eden model each perimetral site can be part of the cluster. In the random deposition model, instead, a column is chosen randomly and a particle is deposited on top of it. No horizontal correlations are therefore present.
 - a) Make a simulation with this model and visualize the surface.
 - b) Verify that the height of the columns follow a Poisson distribution and that $\bar{h} \sim t$ and $w \sim t^{1/2}$. This structure does not depend on L and therefore $\alpha = 0$.
- (e) *Balistic Deposition* In this model the horizontal coordinate is chosen randomly and a particle falls down up to reach the first available perimetral site which is a nearest neighbor of an occupied site. This algorithm allows also a horizontal growth. Consider one particle falling down for each unit time. Discuss the differences -in terms of algorithm and results- with respect to the previous models.

3. Site percolation on the square lattice

- (a) Use the code `perc.f90` or write your own code to generate random site configurations on a square lattice, filling each site with a certain given probability p . Estimate the critical percolation threshold $p_c(L)$ by finding the average value of p at which a spanning cluster is first attained. Choose $L = 8$ and begin at a value of p for which a spanning cluster is unlikely to be present. Then increase p in increments of $\delta p = 0.01$ until you find a spanning cluster. Record the value of p at which spanning first occurs for each spanning criteria. Repeat this process for a total of ten configurations and find the average value of $p_c(L)$. (Remember that each configuration corresponds to a different set of random numbers.) Are your results for $p_c(L)$ using the three spanning criteria consistent with your expectations?
- (b) Repeat part (a) for $L = 16$ and 32 or even larger. Is $p_c(L)$ better defined for larger L , that is, are the values of $p_c(L)$ spread over a smaller range of values? How quickly can you visually determine the existence of a spanning cluster? For a quantitative answer, you can consider the multiple cluster labeling method of Hoshen and Kopelman (see slides, or Gould-Tobochnich, Ch. 13)


```

        end select
    if(any(abs(newpos) > Nw)) exit ! Walker stepped out of the box. Start a new one
    if(occupied(newpos(1),newpos(2))) then ! walker gets stuck to the crystal
        occupied(prevpos(1),prevpos(2)) = .true. ! Add the walker to the crystal
        distance = sqrt(real(dot_product(prevpos,prevpos)))
        if(distance > (radius-5)) radius = distance + 5 ! Make the starting circle larger if necessary
        exit ! terminates the loop immediately
    endif
    prevpos = newpos ! Walker made a valid move (didn't get stuck or wander away). Update and keep it m
enddo
    if(radius > Nw) exit ! terminates the loop immediately
enddo

! Write occupied sites to disk
open(10,file="dla2d.data",status="replace")
do i = -Nw, Nw
    do j = -Nw, Nw
        if(occupied(i,j)) write(10,*) i,j
    enddo
enddo
close(10)

! Do the m(r) analysis and write results to disk
open(11,file="dlamass.data")
do idist = 2, int(0.75*distance)
    mass = 0
    do i = -Nw, Nw
        do j = -Nw, Nw
            if(occupied(i,j)) then
                if(idist**2 >= i**2 + j**2) mass = mass + 1
            endif
        enddo
    enddo
    write(11,'(2i10)') idist, mass
enddo
close(11)

end program dla2d

```

[illegible]

```
implicit none
public::load, init, edengen
integer, parameter, public :: d=2
integer, public :: Lx, Ly, nmcs, posx, c, xmax
real, public :: rnd
integer, public, dimension(1)::seed
```

contains

```
!grid parameters
```

```
subroutine load()
```

```
print*, "L>"
read*, Lx
```

```
print*, "nmcs>"
read*, nmcs
Ly=nmcs
```

```
print*, "seed>"
read*, seed
```

```
end subroutine load
```

```
!Initialize the lattice
```

```

subroutine init(grid, Lx,Ly,s, v)
  integer,intent(inout) :: Lx,Ly,v
  integer :: i
  integer, dimension(Lx,Ly), intent(inout) :: grid
  integer, dimension(2,v), intent(inout) :: s

```

```
grid = 0
s = 0
```

```
do i=1,Lx
  grid(i,1) = 1
  s(1,i)=i
  s(2,i)=2
end do
```

```

!do i= 1, nmcs
!   w(i) = 0.0_d
!   hmed(i) = 0.0_d
!end do

end subroutine init

!eden model

subroutine edengen(grid,Lx,Ly,v,s)

  integer,intent(inout) :: v,Lx,Ly
  integer :: i,ccp,j
  integer, dimension(Lx,Ly), intent(inout) :: grid
  integer, dimension(2,v), intent(inout) :: s
  integer, dimension(2) :: loc

  call random_seed (put = seed)

  loc = minloc(s)
  xmax = loc(2) - 1
  print*,"xmax = ",xmax

  call random_number(rnd)
  posx=int(rnd*xmax)+1
  c=0
  print*,"posx=",posx, "s(1:2,pox)=",s(:,posx)

  grid(s(1,posx),s(2,posx))=1

  if (s(1,posx)==1) then

    ccp=Lx
  else
    ccp=s(1,posx)-1
  end if

  if (grid(ccp,s(2,posx))==0) then

    do i=1,xmax
      if ((s(1,i)/=ccp) .and. (s(2,i)/=s(2,posx))) then
        s(1,posx)=ccp
        s(2,posx)=s(2,posx)
        c=1
      end if
    end do
  end if
end subroutine edengen

```

```

        end if
    end do
end if

if (s(1,posx)==Lx) then
    ccp=1
else
    ccp=s(1,posx)+1
end if

if (grid(ccp,s(2,posx))==0) then

    do i=1,xmax
        if ((s(1,i)/=ccp) .and. (s(2,i)/=s(2,posx))) then
            if (c==0) then
                s(1,posx)=ccp
                s(2,posx)=s(2,posx)
                c=1
            else
                s(1,xmax+1)=ccp
                s(2,xmax+1)=s(2,posx)
                xmax=xmax+1
            end if
        end if
    end do
end if

if (s(2,posx)==Ly) then
    ccp=1
else
    ccp=s(2,posx)+1
end if

if (grid(s(1,posx),ccp)==0) then
    print*,"s(1,posx)=",s(1,posx)
    print*,"ccp=",ccp
    do i=1,xmax
        if ((s(1,i)/=s(1,posx)) .and. (s(2,i)/=ccp)) then
            print*,"si"
            if (c==0) then
                s(1,posx)=s(1,posx)
                s(2,posx)=ccp
                c=1
            else
                s(1,xmax+1)=s(1,posx)
                s(2,xmax+1)=ccp
            end if
        end if
    end do
end if

```



```

        xmax=xmax+1
    end if
end if
end do
end if

!   do i=1,2
!   print*,(s(i,j), j=1,v)
!   end do
!   do i=1,Ly
!   print*,(grid(j,i), j=1,Lx)
!   end do

if (grid(s(1,posx),s(2,posx)-1)==0) then
    print*,"s4"
    do i=1,xmax
        if ((s(1,i)/=s(1,posx)) .and. (s(2,i)/=s(2,posx)-1)) then
            if (c==0) then
                s(1,posx)=s(1,i)
                s(2,posx)=s(2,i)-1
                c=1
            else
                s(1,xmax+1)=s(1,i)
                s(2,xmax+1)=s(2,i)-1
                xmax=xmax+1
            end if
        end if
    end do
end if

end subroutine edengen

end module common

program eden

    use common
    implicit none
    integer::i,v,j
    integer, dimension(:,:), allocatable :: grid
    real, dimension (:), allocatable :: w, hmed
    integer, dimension(:,:), allocatable :: s

    open (unit=1, file="eden1.dat", status= "replace", action="write")

    call load ()

```

```

v=lx*ly

allocate(grid(Lx,Ly))
allocate(w(nmcs))
allocate(hmed(nmcs))
allocate(s(2,v))

call init(grid,Lx,Ly, s, v)
print*,"v=lx*ly=",v
print*,"nmcs=",nmcs
do i=1, nmcs
!   print*," imcs=",i,"    grid:"
!   do j=ly,1,-1
!       print*,grid(1:lx,j)
!   end do
!       call edengen(grid,Lx,Ly,v,s)
end do

write(unit=1,fmt=*) s

close(unit=1)

deallocate(grid,w,hmed,s)

end program eden

```

```
! ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! perc.f90
! ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! from:
! http://www.physics.nau.edu/~hart/classes/550_spring_2004/hw_schedule/solutions/perc.f90
```

```

        label = min(grid(i,j),grid(i-1,j)) ! select the smaller cluster number
        where(grid == max(grid(i,j),grid(i-1,j))) ! relabel all the sites with the
            grid = label ! higher cluster number
        endwhere
    endif
    if(grid(i,j+1)/=0)then ! neighboring site is occupied
        label = min(grid(i,j),grid(i,j+1)) ! select the smaller cluster number
        where(grid == max(grid(i,j),grid(i,j+1))) ! relabel all the sites with the
            grid = label ! higher cluster number
        endwhere
    endif
    if(grid(i,j-1)/=0)then ! neighboring site is occupied
        label = min(grid(i,j),grid(i,j-1)) ! select the smaller cluster number
        where(grid == max(grid(i,j),grid(i,j-1))) ! relabel all the sites with the
            grid = label ! higher cluster number
        endwhere
    endif

end subroutine cluster_update
end module perc

program percolation
use perc
use random_stuff
implicit none

integer, parameter :: L=22 ! size of grid
integer, dimension(0:L+1,0:L+1) :: site = 0 ! grid of sites
real :: p ! occupation ratio
real :: ratio ! ratio of sites in percolating cluster vs. number of total occupied sites
real :: rnd(2) ! 2 random numbers for generating random x, y positions
real :: invL ! inverse box edge length

integer :: i, j, loop ! general loop counters
integer :: x, y ! x, y positions (integer)
integer :: nsites = 0 ! number of occupied sites
integer :: iclust = 1 ! current cluster number
integer :: PercClusLab ! Cluster number (label) of the percolated cluster

call set_random_seed(0) ! "Randomly" set the random seed

invL = 1/real(L)
open(11, file="percavg.data", status = "old", position = "append")
do loop = 1, 20
    site = 0
    nsites = 0

```

```

iclust = 1
do ! Loop until percolation happens
  call random_number(rnd) ! Select two random numbers
  x = int(rnd(1)*L)+1      ! Change to random numbers between 1 and L (x, y positions)
  y = int(rnd(2)*L)+1
  if(site(x,y)==0) then    ! it's unoccupied
    site(x,y) = iclust    ! mark the site as occupied
    nsites = nsites + 1    ! keep track of the number of occupied sites
    ! check to see if this new site is already part of a cluster
    call cluster_update(x,y,site) ! relabel any clusters as necessary
    iclust = iclust + 1
    if(perccheck(site)/=0) exit ! Stop adding sites if a cluster has percolated
  endif
enddo
p = real(nsites)/L**2
ratio = count(site==perccheck(site))/real(nsites)
print *, "occupancy: ", p
print *, "ratio in percolating cluster: ", ratio
write(11,*) invL, p, ratio
enddo
close(11)

! Write cluster data to file
PercClusLab = site(x,y)
open(10,file="perc.data"); open(11,file="spanningcluster.data")
do i=1,L; do j=1,L
  if(site(i,j)/=0) write(10,'2i10') j,i
  if(site(i,j)==site(x,y)) write(11,'2i10') j,i ! write sites for spanning cluster only
enddo;enddo
close(10); close(11)

end program percolation

```