Práctica 6-
Diffusion Limited Aggregatiion

Asignatura: *Computación Avanzada*

Profesor: *David Martín y Marero*

Luis Sánchez Izquierdo | viernes 13 de abril de 2018

Contents

Introducción

Teoría

En esta práctica vamos a estudiar un cúmulo creado mediante el método limitado por difusión, o en inglés *diffusion limited aggregation* al que nos referiremos como *DAL*. Este agregado produce una figura que sigue una estructura fractal.

La teoría que sostiene el método de creación de este agregado es mediante los pasos aleatorios.

Objetivos

La práctica consiste en:

- Desarrollar el método de crecimiento del fractal y representarlo gráficamente. Este método desarrollado individualmente será explicado paso a paso
- Calcular la dimensión fractal del cúmulo. Esta dimensión fractal será caracterizada y explicada.

Además, realizar un apartado extra en el que caracterizaré la propia entropía del sistema y por último he realizado un programa en el que se ve la evolución visual en un sistema que es ligeramente diferente.

El método del crecimiento del cúmulo

- ## Definiciones iniciales

La idea es crear un mallado del espacio sobre la que se van a situar las partículas. Este mallado es una matriz llamada "$posiciones$" de dimensión $L \cdot L$ (en donde L es la longitud de la caja) en la que guardaré variables de tipo *bool*. Si la variable es "$True$" hay una partícula en esa posición y si la variable es "*False*" entonces no hay partícula en esa posición. El centro de la caja estará en la posición $C = \frac{L}{2}$

Sobre el espacio ponemos una semilla inicial de tamaño $2x2$ en las posiciones $(0,0)$ $(0,1)$ $(1,0)$ y $(1,1)$, que corresponden en la matriz "$posiciones$" a las posiciones

$$(C,C) \quad (C+1,C) \quad (C,C+1) \quad (C+1,C+1)$$

```
m = 400          # numero de pasos de cada paseantedr = 10
C=m+1            # posicion del centro
L=m*2+1          # longitud de las posiciones
posiciones = full((L,L), False)

posiciones[C,C] = True   #semilla inicial
posiciones[C+1,C] = True   #semilla inicial
posiciones[C,C+1] = True   #semilla inicial
posiciones[C+1,C+1] = True   #semilla inicial
```

Además, establezco también un número total de partículas que llamaré N. Cada vez que se agregue una partícula al cúmulo una variable que actúa como contador i crecerá en un valor 1 de modo que $i += 1$

Para agregar a todas las partículas haré un bucle *while i<N*

- ## El método del crecimiento

El método de crecimiento consiste en:

1- Crear una partícula a una distancia r del centro.
2- Dejar que la partícula se mueva siguiendo los pasos de un camino aleatorio (comúnmente llamado como camino del borracho)
3- Cada vez que camine un paso comprobar si a su alrededor hay alguna partícula alrededor:
   - De ser cierto que hay una partícula en su vecindad entonces la partícula se queda pegada y actualizamos su posición en la matriz de "$posiciones$"
   - Si no hay una partícula alrededor dejamos que continúe caminando hasta que agote los pasos.

La explicación de cada paso la realiza a continuación:

a. Crear una partícula a una distancia r del centro

La distancia r la defino como

$$r = R + dr$$

En donde

$$R = radio\ del\ cúmulo$$

$$dr = una\ distancia\ constante$$

El radio del cúmulo es la distancia al centro de la partícula del cúmulo que está más alejada. A medida que se van agregando partículas el radio del cúmulo es posible que vaya aumentando, de modo que hay que ir actualizando r cada vez que se agrega una partícula mediante el algoritmo:

```
68                  r = max(sqrt(x*x + y*y) + dr, r)
```

En donde (x, y) son las posiciones de la nueva partícula agregada. Esto significa que agregaremos partículas siempre a una separación constante dr del radio del cúmulo.

Una vez definida la distancia r la posición inicial de la partícula se calcula con

```
42 def rand_move(raux):
43     phi = random()*2*pi
44     return int(cos( phi ) * raux), int(sin( phi ) * raux)
```

b. El camino aleatorio

La partícula va a realizar un camino aleatorio en el que se puede mover en todas las direcciones por igual. Es decir, tiene la misma probabilidad de ir a

$(1,-1)$	$(1,0)$	$(1,1)$
$(0,-1)$		$(0,1)$
$(-1,-1)$	$(-1,0)$	$(-1,1)$

Para simular eso basta con crear un número aleatorio entre 0 y 7 que viene dado por la función *randint()* importada de numpy.random y con ese número aleatorio se mueve a la partícula de la forma

```
51 while i < N:
52     print("paso ",i)
53     #posicion inicial
54     rx, ry = rand_move(r)
55     x = rx
56     y = ry
57     j = 0       #indice del paso
58     while j < m:    #realizar el camino aleatorio
59         ran = randint(0,7)
60         x += nx[ran]
61         y += ny[ran]
```

En donde $nx[\,]$ y $ny[\,]$ son las matrices de las posiciones vecinas

```
38 #vecinos
39 nx = [-1, -1, -1, 0,  0, 1, 1,  1]
40 ny = [ 1,  0, -1, 1, -1, 1, 0, -1]
```

c. Ver si la partícula tiene un vecino

Y recorro las posiciones vecinas en un bucle *for* y compruebo si la partícula existe (condiciones que la posición sea *True*):

```
63          for k in range(7):
64              if posiciones[x + C + nx[k], y + C + ny[k]] == True:
65                  posiciones[x + C, y + C] = True
66                  i += 1
67                  j = m
68                  r = max(sqrt(x*x + y*y) + dr, r)
69                  break
```

d. Comprobar que la partícula no se aleje demasiado

Para comprobar que la partícula no se aleja demasiado lo que haré no será poner la condición de que $r > 1.5 \cdot r_0$ sino que he preferido poner la condición $r > r_0 + dr$ ya que si r_0 aumenta mucho entonces $1.5 \cdot r_0$ será cada vez más inalcanzable y por tanto llegará un momento en que la condición no valga.

```
62          if x*x + y*y > (r + drmargin)*(r + drmargin):
63              j=m
```

La condición $j = m$ hace que en el siguiente paso del $while\ j < m$ no se cumpla y por tanto nos hayamos salido del bucle

Todo este proceso es lo que denominamos como la eliminación de una partícula.

Representación del cúmulo

- ### Código computacional para la representación del cúmulo

Para representar el cúmulo solo basta con ver cuales posiciones de la matriz "$posiciones$" son *True* y representarlas con un *plot:*

```
74 # Dibujar el cúmulo
75 xplot=[]
76 yplot=[]
77 for i in range(L):
78     for j in range(L):
79         if posiciones[i,j] == True:
80             xplot.append(i-C)
81             yplot.append(j-C)
82 figure()
83 title("Cumulo DAL")
84 xlabel("Posicion X")
85 ylabel("Posicion Y")
86 plot(xplot,yplot,'s')
```

- ### Resultados obtenidos

Para un sistema con partículas generadas a una distancia $dr = 10$ del radio del cúmulo el resultado que se obtiene es el mostrado a continuación

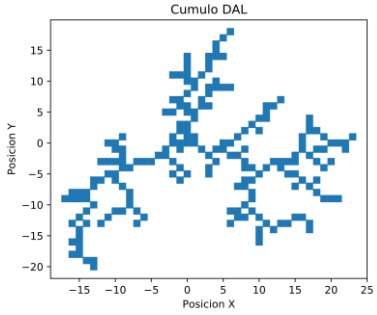


Figure 1- Numero de paseantes N = 200

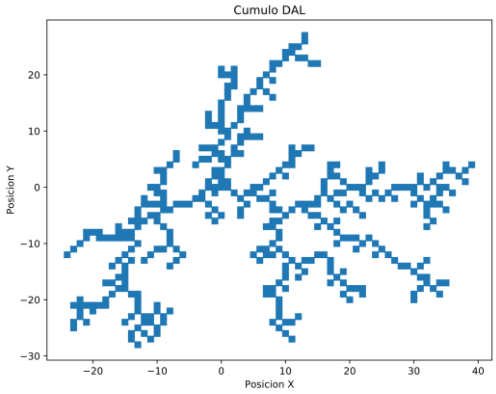


Figure 2- Numero de paseantes N = 400

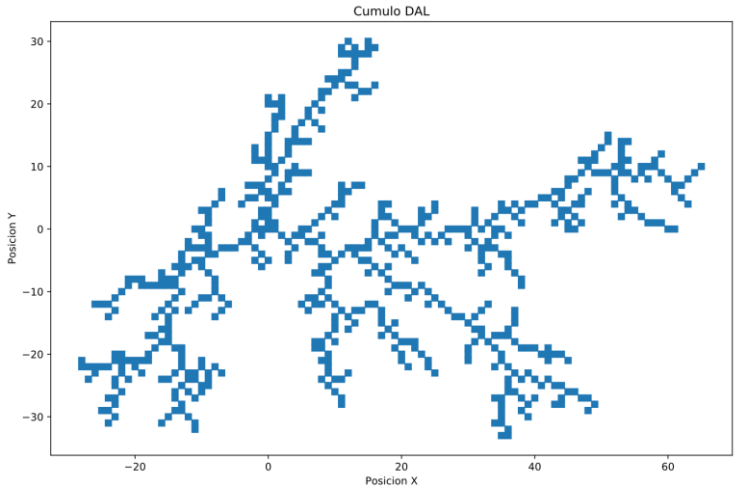


Figure 3- Numero de paseantes N = 600

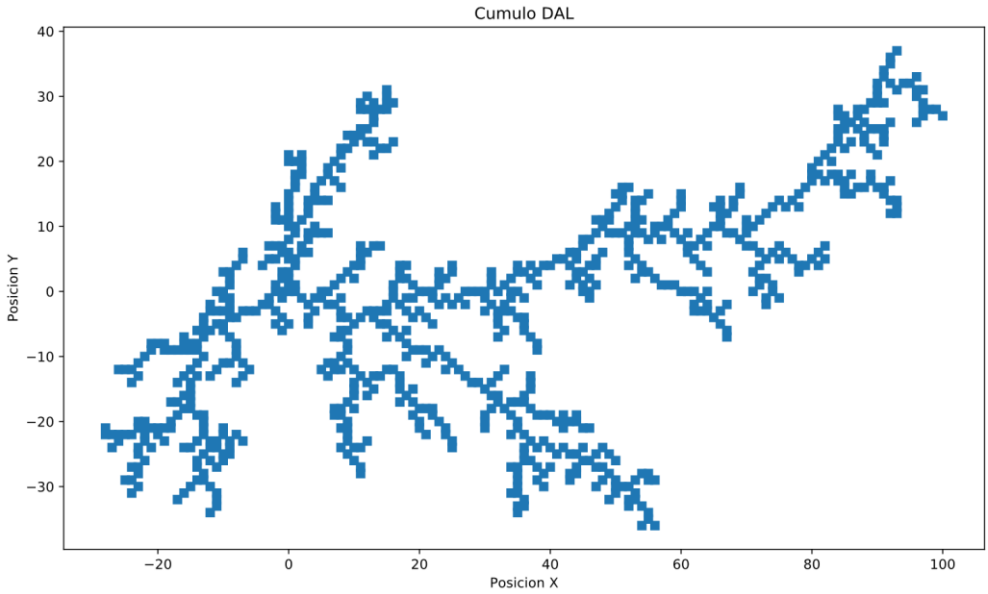


Figure 4- Numero de paseantes N = 800

Dimensión fractal del cúmulo

• Definición de la dimensión fractal

La dimensión fractal mide la cantidad de masa que hay a una distancia r_0 del centro. Este índice nos indica que el cúmulo es un fractal ya que cumple una de las propiedades más importantes: la repetición de patrones a medidas que haces *zoom in* o bien si haces *zoom out*.

Si decimos que cada partícula tiene una masa $m = 1$, entonces queda definida la masa del cúmulo a una distancia r_0, es decir $m(r_0)$, como la cantidad de partículas que hay en el interior de la circunferencia de radio r

$$m(r_0) = \#partículas\ en\ la\ zona\ r < r_0$$

Dicha masa está relacionada con r_0 a través de la dimensión fractal mediante

$$m(r) \propto r^{d_f}$$

Y tomando logaritmos podemos obtener una regresión lineal para hallar d_f

$$\log\big(m(r)\big) = a + d_f \cdot \log(r)$$

Para un cúmulo denso en el que todos los huecos se llenan tendremos que la masa será proporcional al área

$$m(r) = \pi r^2$$

De modo que la dimensión fractal será $d_f = 2$

Nosotros para nuestro cúmulo esperamos encontrar una dimensión fractal menor ya que hay huecos entre medias y la densidad de huecos aumenta a medida que aumenta el cúmulo

• Obtención de la dimensión fractal computacionalmente

Para obtener la dimensión fractal computacionalmente lo que hago es que, a partir de las posiciones de las partículas del cúmulo, obtengo su posición r al centro.

```
# obtener la dimension fractal
x = array(xplot)
y = array(yplot)
r = sqrt( x*x + y*y )       #vector de distancias al centro
```

Después hallo la mayor distancia r_{max} y hago un vector equiespaciado desde 0 hasta r_{max} y otro vector de conteo (que será el vector de masas)

```
rmax =  max(r)
Ndiv = 30
rplot = linspace(0,rmax, Ndiv)
count = zeros(Ndiv)
```

Tras esto, recorro el vector de distancias al centro que había calculado y relleno el vector de masas.

```
for i in r:
    # posicion de la particula i con respecto a rmax
    nn = int(i / rmax * Ndiv)
    # dicha partícula contribuye a la masa a partir de nn hasta Ndiv
    for j in range(nn, Ndiv):
        count[j] += 1
```

Después represento los datos y realizo un ajuste de mínimos cuadrados para hallar la dimensión fractal. Esto lo hago usando la función *polyfit* de grado 1.

```
#representación de los datos
figure()
auxx=log(rplot)
auxy=log(count)
plot(auxx, auxy,'.-')
title("Obtención de la dimensión fractal")
xlabel("log(r)")
ylabel("log(m)")

p,r=polyfit(auxx[2:-3],auxy[2:-3],1)
xfit = linspace(0,log(rmax),1000)
yfit = p * xfit + r
plot(xfit,yfit,'-r')
```

- Resultados obtenidos

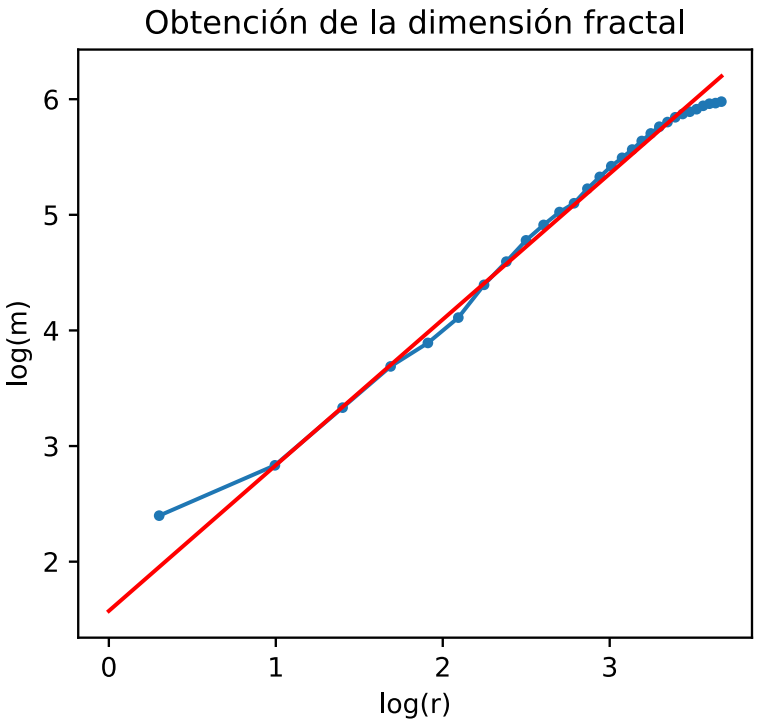


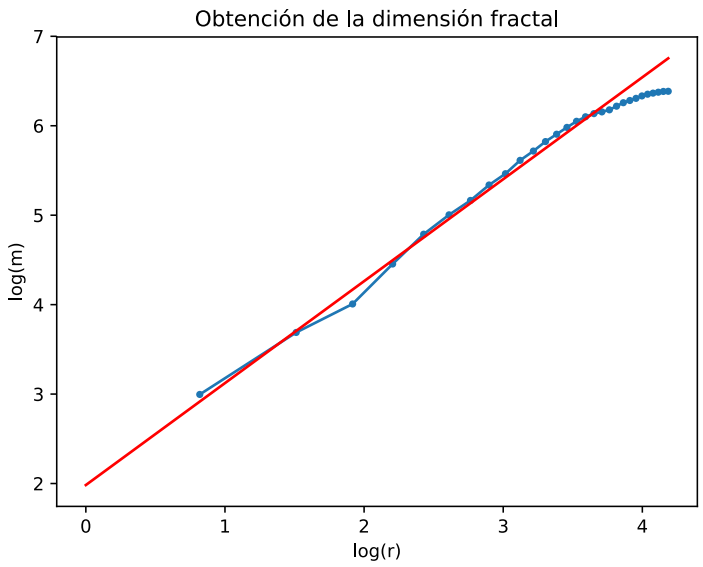
Figure 5- Azul = puntos experimentales. Rojo = ajuste lineal

Esta dimensión fractal la he calculado para un cúmulo de $N = 400$ partículas. La gráfica con el ajuste lineal nos da un valor de ajuste:

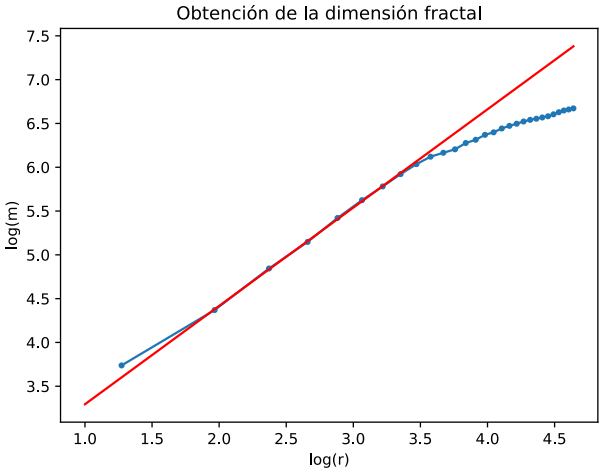
$$\log(m) = 1.26\log(r) + 1.57$$

De modo que el valor de la dimensión fractal es $d_f = 1.26$

Para un valor de partículas mayor, $N = 600$, obtengo un ajuste parecido en el que la dimensión fractal vale $d_f = 1.15$



Para un valor de partículas mayor aún, $N = 800$, obtengo un ajuste parecido en el que la dimensión fractal vale $d_f = 1.12$



- Interpretación de los resultados

Primeramente, se puede ver que tanto el valor es menor de 2 como habíamos anticipado en la sección de la definición.

$$d_f \approx 1.2 < 2$$

Después se puede ver que a medida que aumentamos en número de partículas va disminuyendo la pendiente (se va haciendo más horizontal). La explicación a este fenómeno es que a medida que aumentan el número de partículas se va generando preferencia por una rama, de modo que el crecimiento pasa de ser muy circular (ver Figure-1) a ser muy lineal (ver Figure-4)

Extra 1: Obtención de la Entropía

- Definición de la entropía

Este apartado tiene como objetivo calcular el valor de la entropía del sistema. La entropía viene definida como

$$S = \sum_i P_i \log(P_i)$$

En donde P_i es la probabilidad de estar en el estado i. Para calcularlo lo que se hace es dividir el espacio en cuadrantes de tamaño $NN \cdot NN$ y cuento en número de partículas que hay en dicho cuadrante (el lado tiene longitud NN). Ese será el valor de P_i.

Como en el crecimiento del DAL vamos añadiendo partículas, la magnitud relevante no va a ser valor total de la entropía sino el valor de la entropía por partícula.

$$s_j = \frac{S_j}{n_j}$$

En donde

$$s_j = entropía\ por\ unidad\ de\ partícula\ en\ el\ instante\ j$$

$$n_j = número\ de\ partículas\ en\ el\ instante\ j$$

- ## Código computacional

Computacionalmente lo hago definiendo el vector de entropía.

```
# Entropia
Entropy = [0]
lenentropy = 30
divS = int(N/lenentropy)
NN = 4 #Longitud del tamaño donde calcularemos la entropia
indmax = L//NN
```

Y definiendo la función de entropía:

```
def Entropia(Npart):
    entropia = zeros(indmax**2)
    for i in range(L):
        for j in range(L):
            if posiciones[i,j] == True:
                entropia[i//NN + indmax*(j//NN)] += 1
    S = 0
    for k in entropia:
        if k != 0:
            S += (k/Npart) * log(k/Npart)
    return S
```

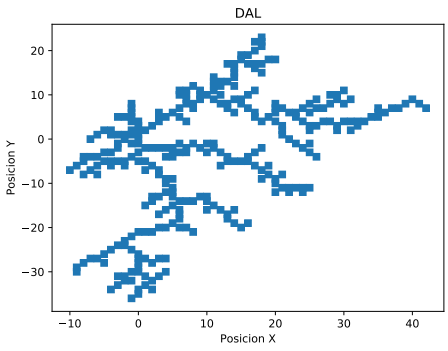
Y lo único que tengo que hacer el que cada vez que el número de partículas $i = \#partículas$ aumente en una unidad $divS$, entonces calcule la entropía por partícula $s_j = S_j/n_j(i)$

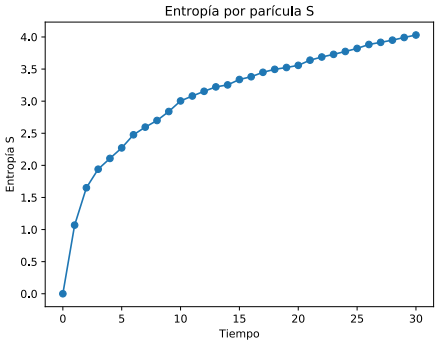
```
                if(i%divS == 0):
                    Entropy.append(-Entropia(i))
```

- ## Resultados e interpretación

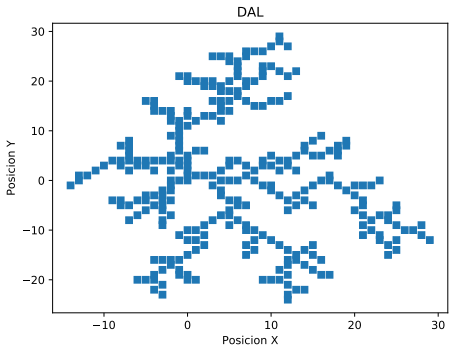
Dependiendo de la semilla inicial obtendremos un cúmulo u otro. He realizado lo mismo para varios cúmulos. En todas estas simulaciones he tomado cajas de lado $NN = 4$

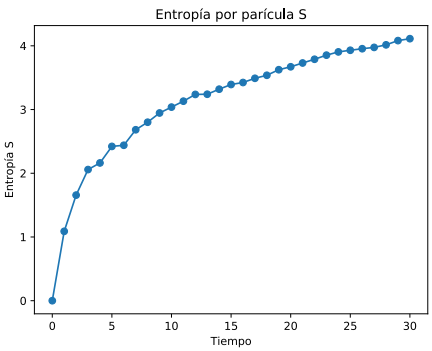
Seed = 11



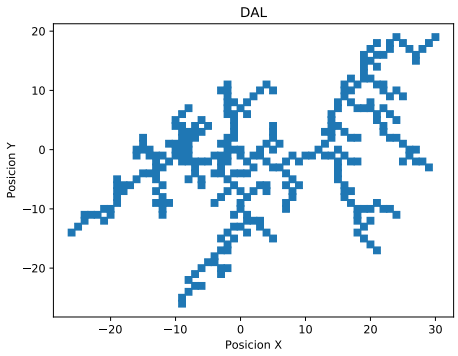


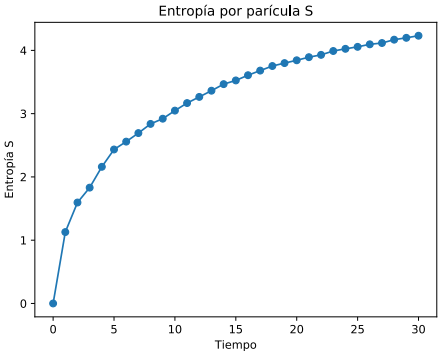
Seed = 10





Seed = 1





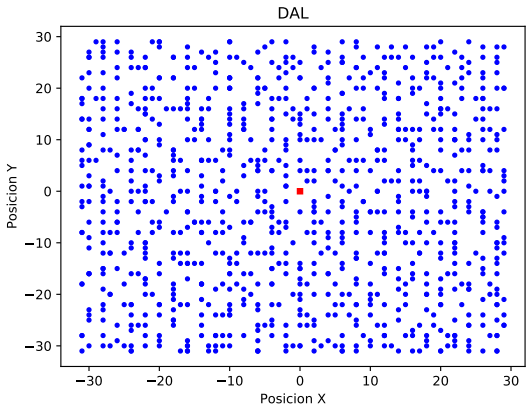
Se observa claramente que la entropía por partícula va aumentando como una raíz cuadrada. Esto tiene una explicación bastante sencilla y es que todas las partículas están igual de desordenadas. Además, se cumple la propiedad más importante del fractal: a medida que aumentamos o disminuimos el tamaño el patrón es igual, de modo que eso significa que el desorden tiene que tender a un valor asintóticamente, como se ven en las gráficas.

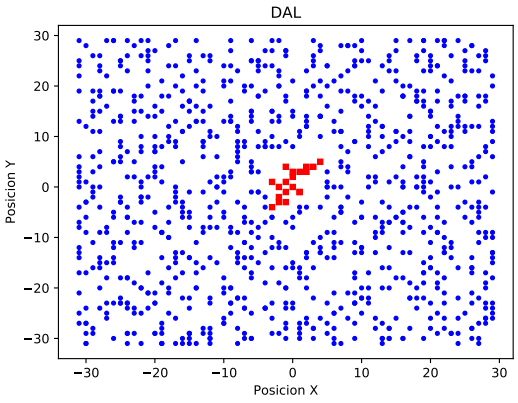
Extra 2: Otro sistema DAL

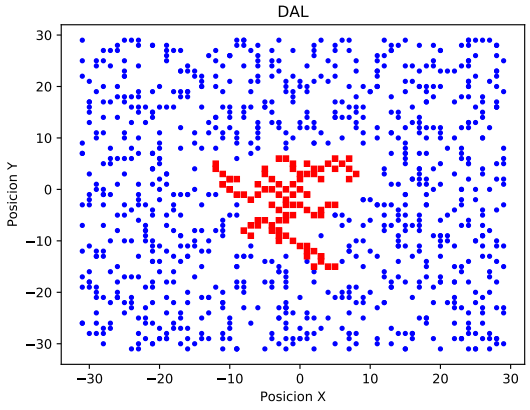
Lo que hago en este caso es poner las partículas equiespaciadamente en el en tablero y dejo que todas y cada una de las partículas se pongan a andar siguiendo camino aleatorios. Dichas partículas serán representadas en color azul.

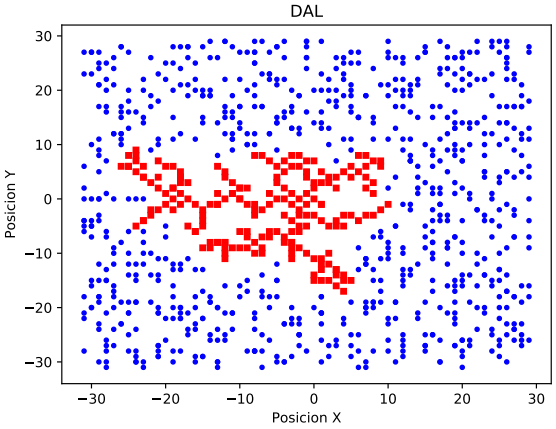
De nuevo cada vez que una partícula se choque con alguna que esté quieta (es decir, que ya esté formando parte del cúmulo) se quedará también quieta y pasarán a formar parte del cúmulo. Estas partículas las pintaré de color rojo.

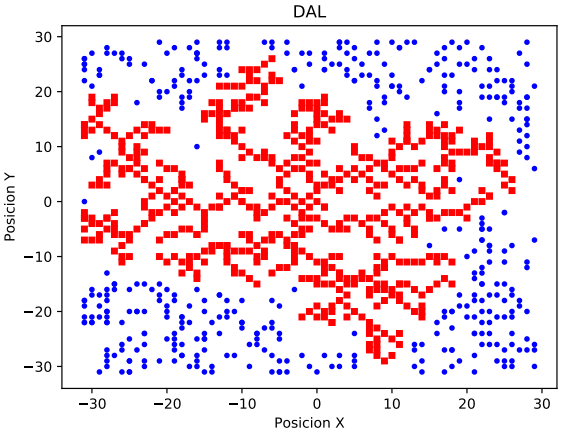
El resultado es el que se muestra a continuación:

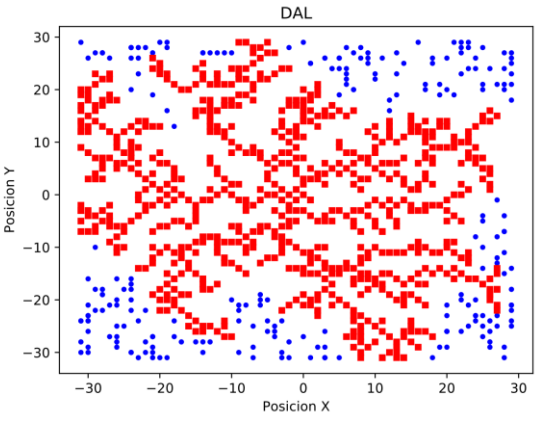


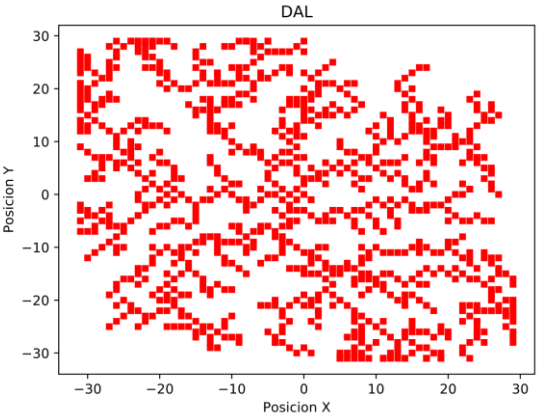





DAL
Posicion Y
Posicion X
30
20
10
0
−10
−20
−30
−30
−20
−10
0
10
20
30


DAL
Posicion Y
Posicion X
30
20
10
0
−10
−20
−30
−30
−20
−10
0
10
20
30


DAL
Posicion Y
Posicion X
30
20
10
0
−10
−20
−30
−30
−20
−10
0
10
20
30


DAL
Posicion Y
Posicion X
30
20
10
0
−10
−20
−30
−30
−20
−10
0
10
20
30

Conclusiones

Tras analizar el método de creación de un DAL he explicado cómo funciona el método que he desarrollado. Este método es el más eficiente computacionalmente ya que lo que usa es valores booleanos para guardar las posiciones de las partículas.

Además, calculo el valor de la dimensión fractal obteniendo un valor aproximadamente de

$$d_f \approx 1.2$$

Habiendo realizado esto, he pasado a hacer un análisis avanzado midiendo el valor de la entropía. Se obtienen resultados acordes a lo que se observa en el libro Giordano.

Por último, he desarrollado otro tipo de cúmulo creado por el límite de difusión. En este nuevo algoritmo, más eficiente aún, se puede ver la evolución del cúmulo en visualmente en forma de vídeo si se corre el programa