Random Field Ising Model

Timothé ALEZRAA & Sankarshan SAHU

15/03/2023 -21/03/2023

Abstract

In this report our primary aim is to investigate the behaviour of the Ising model in presence of a random field. We consider the regime of zero temperature and compute the histograms and the hysteresis loops of the Random Field Ising Model, a signature of which are the avalanches which are present in case of RFIM but are absent for an Ising model without a Random Field. We also investigate this in case of Erdös-Rényi random graphs.

1 Introduction

The Random Field Ising Model is one of the most important and studied systems in condensed matter physics.The Random Field Ising model is often used to describe diluted anti-ferromagnets in presence of a uniform magnetic field like $Fe_2Zn_{1-x}F_2$ and $Fe_2Mg_{1-x}Cl_2$.

In recent years, the physics of many complex systems has been captured by the Ising model with different lattices and interactions. In our model/simulation we will be considering a periodic lattice, where lattice sites are occupied by spins with $s_i = \pm 1$. The Hamiltonian describing a system with $s = s_i$ spins, is thus given by:-

$$\mathcal{H}(s) = -\sum_{i,j} J_{ij} s_i s_j - \sum_i h_i s_i - H \sum_i s_i \qquad (1)$$

This suggests that any pair of spins contribute with a random interaction strength J_{ij} and each spin also has a bias represented by a local magnetic field intensity h_i which is chosen randomly from a Gaussian distribution. In our specific case, we also consider the ferromagnetic case $J_{ij} = J > 0$.Also we consider cases without frustation i.e. (A system is said to be frustrated whenever we cannot minimize its total classical energy by minimizing each group of interacting degrees of freedom.) Thus we consider the Hamiltonian [1] :-

$$\mathcal{H}(s) = -J \sum_{i,j} s_i s_j - \sum_i h_i s_i - H \sum_i s_i \qquad (2)$$

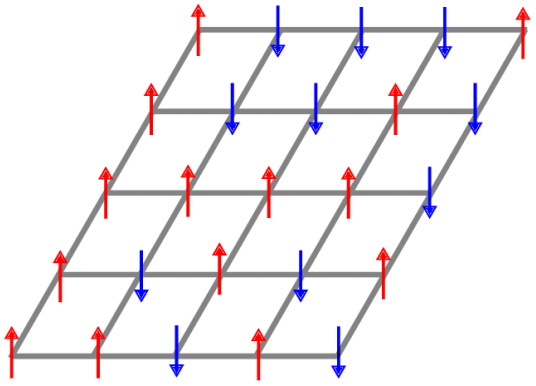


Figure 1: Schematic Diagram of a 2D Ising Model on square lattice [5]

Here we consider a square lattice with side L and $N = L^d$, where d indicates the number of space dimensions and N, the number of spins. In our report, we mainly work with $d = 2$ and $d = 3$.

Our main goal in the report is to study the behaviour of the hystersis loop at $T = 0$ and then at $T \neq 0$. As H is slowly increased from $-\infty$ to $+\infty$, and then again decreased back to $-\infty$, each spin flips deterministically when the quantity $B_i = sign(J \sum_{j \in \partial i} s_j + H + h_i)$ changes .

1.1 Spin-Flip Dynamics and Avalanches

Initially we start with a state where all spins are spin down -1 and $H = -\infty$, and we keep on increasing H. As we keep on increasing H, for some sites with a very large random field, the quantity B_i becomes positive, i.e. it changes sign and hence the spin at that corresponding lattice site flips. Flipping a spin, increases the local field at a neighbouring site, which in turn might cause the spin at the neighbouring site to flip up. If increasing the field H, by a small amount flips t spins, then we are said to have an avalanche of size t. As the applied field H, increases more and more spin flips up, until the point when increasing H has no effect on the configuration of spins, since we have a system with only spin ups. Then again we start decreasing the applied field H, from ∞ to

$-\infty$, and in doing so plot a hysteresis loop, with many avalanches where the number of avalanches depend on the variance of the Gaussian distribution from where the random field h_{ij} was selected.

1.2 Detail Balance and the Metropolis Algorithm

When a spin system is in contact with a heat bath at temperature T, the stochastic changes induced in s, is given by [4]-

$$\frac{\partial P}{\partial t} = \sum_{s^j} [c(s^j; j)P(s^j) - c(s; j)P(s)] \tag{3}$$

where, $P(s)$, represents the configuration of the system s, at any time t and $c(s; j)$ is the probability of flipping the spin from s to s^j per unit time. This is obtained by assuming Boltzman distribution for the spins, i.e. the detail balance condition is given by:-

$$c(s; j) = c(s^j; j)e^{-\beta\Delta\mathcal{H}} \tag{4}$$

where:- $\Delta\mathcal{H} = \mathcal{H}(s) - \mathcal{H}(s^j)$, with $\beta = (k_B T)^{-1}$, where k_B is the Boltzman constant.This is indeed satisfied by the Metroplois Algorithm given as $c(s; j) = min\{1, \exp(-\beta\Delta\mathcal{H})\}$.

1.3 Erdös-Rényi graphs

In the field of graph theory,the Erdös-Rényi model refers to two closely related models used for generation of Random graphs [2][3]. An Erdös-Rényi random graph is a set of N vertices connected by B bonds, where the propability p of two sites being connected by a bond is given by $p = C/N$ where C is the average number of neighbours a spin has in the graph. The connectivity of a site is given by the total number of bonds connected to that site, i.e. $k_i = \sum_j l_{ij}$, where $l_{ij} = 1$, if two sites are connected and 0 if they are disconnected [3]. In this report we study the Random Field Ising model on Erdös-Rényi graph, and plot the hysteresis loop at temperature T=0 and compare it with the normal case where the spins are kept on a square lattice (for two dimensions) or a cubic lattice (for three dimensions) with equal average coordinality. Our interest mainly lies in how the avalanches and the hysteresis loop itself change, due to the introduction of the Erdös-Rényi graphs(2).

2 Results

Figure:-3 shows the plot of magnetisation vs the external Magnetic field H (i.e. the Hysteresis loop) at T=0 (for two dimensions). Here, we just see a square loop without any avalanches. We actually find exactly the same plot for three dimensions i.e.

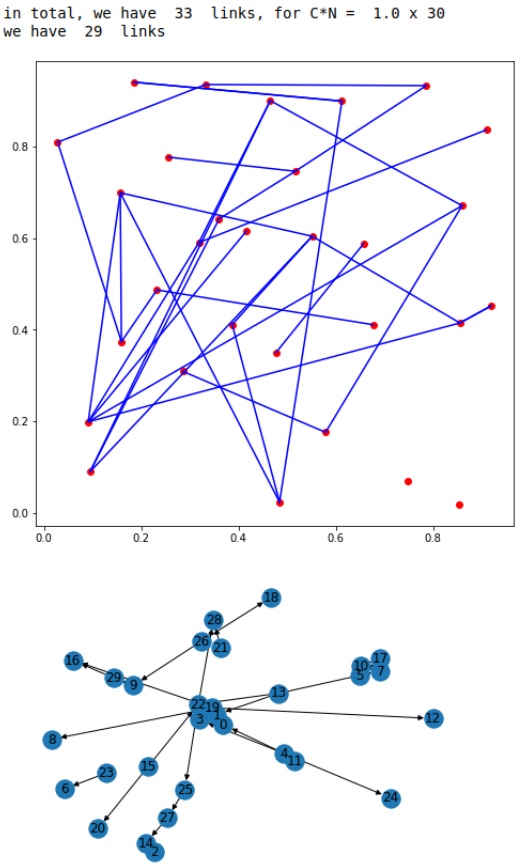


Figure 2: representation of an Erdos-Renyi graph links

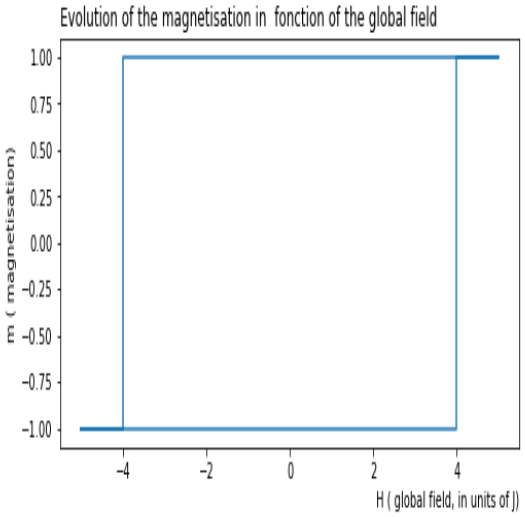


Figure 3: Hysteresis plot without any random field at zero temperature(T=0)

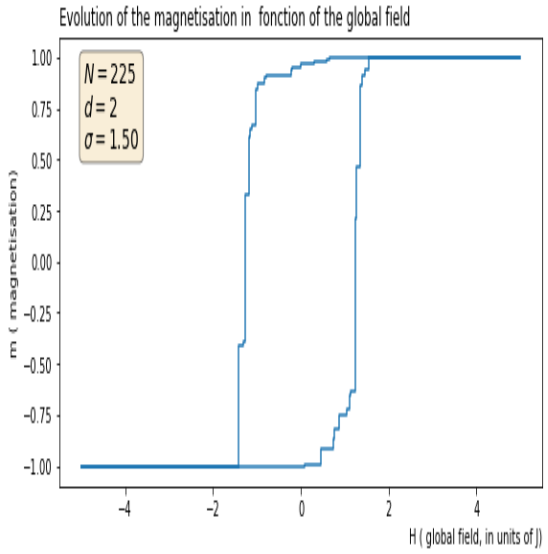


Figure 4: Hysteresis plot without a random field at low bias at zero temperature(T=0) for a square lattice (2-Dimensions)

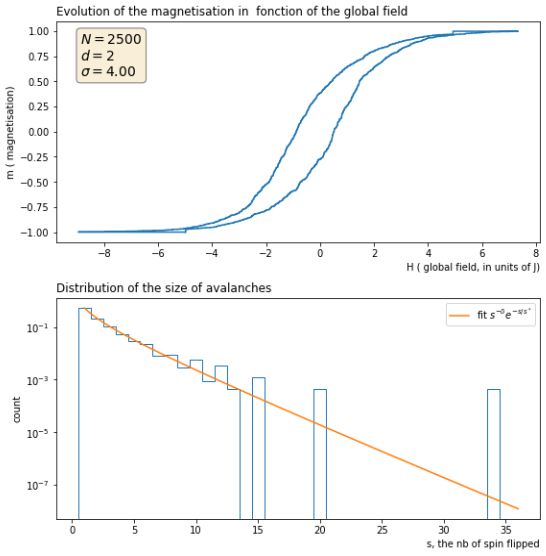


Figure 5: (a)Hysteresis plot and (b)distribution of size of avalanches with a random field at high bias at zero temperature(T=0) for a square lattice (2-Dimensions)

the one without any avalanches for the same reason. Figure:-4 shows the Hysteresis plot for a random field with low bias (with variance=1.5) at T=0 in two dimensions for $L = 15$, i.e. $N = 225$ spins and . Here as we have explained in section 1.1, we expect to see avalanches due to the presence of a random field. Figure:-5(a) shows the hysteresis plot for a random field with a high bias (with variance=4.0) at T=0 for a similar configuration, and we see much more avalanches as we would expect. Figure:-5(b) shows the distribution of size of avalanches fitted with the curve $s^{-\tau}e^{-s/s*}$, where τ and $s*$ are the parameters of the fit. Next, we plot the same for 3 dimensions with $L = 10$ i.e. $N = 1000$ spins in Figure:-6(a) and Figure:-6(b). The results are similar as one would expect from the spin flip dynamics.Figure:-7(a) gives the hysteresis plot for the Erdös-Rényi graph in 2 dimensions at zero temperature. Here we see an unique behaviour for the Hysteresis loop, i.e. there is no proper hysteresis.

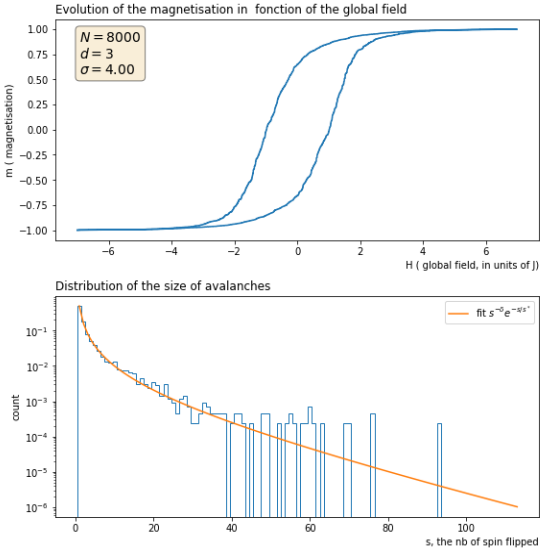


Figure 6: (a)Hysteresis plot and (b)distribution of size of avalanches with a random field at high bias at zero temperature(T=0) for a cubic lattice (3-Dimensions)

3 Conclusions

In this section we summarize the report and discuss the future directions in which this work could have been extended. In the report we first start with a rather theoretical introduction to the Random Field Ising Model. We then discuss the spin-flip dynamics and the avalanches in the hysteresis loop that results out of it. Then we discuss the detail balance condition and the metropolis algorithm which we intended to implement for $T > 0$ regime, however due to time constraints we were unable to do so. But in any case

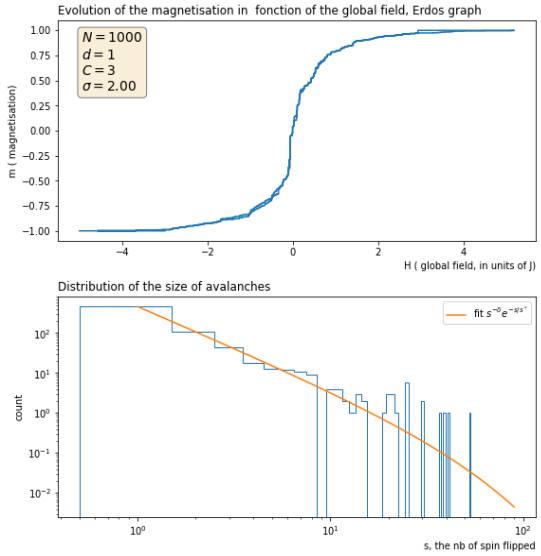


Figure 7: (a)Hysteresis plot and (b)distribution of size of avalanches with a random field at high bias at zero temperature(T=0) for Erdös-Rényi graphs in 2-Dimensions

we have tried to put together an overall theory underlying this algorithm. We then give a brief preview of the Erdös-Rényi random graphs. In the Results section, we produce several plots and compare the hysteresis loops and distribution of avalanches in case of bias, no bias and small bias in 2D and 3D Ising models and proceed to do the same for the Erdös-Rényi case.

References

[1] Nattermann, T., and J. Villain. "Random-field ising systems: A survey of current theoretical views." Phase transitions 11.1-4 (1988): 5-51.

[2] Erdős, Paul, and Alfréd Rényi. "On the evolution of random graphs." Publ. Math. Inst. Hung. Acad. Sci 5.1 (1960): 17-60.

[3] Lima, F. W. S., and Muneer A. Sumour. "Ising model with spins S= 1/2 and 1 on directed and undirected Erdös–Rényi random graphs." Physica A: Statistical Mechanics and its Applications 391.4 (2012): 948-953.

[4] Crokidakis, Nuno. "First-order phase transition in a 2D random-field Ising model with conflicting dynamics." Journal of Statistical Mechanics: Theory and Experiment 2009.02 (2009): P02058.

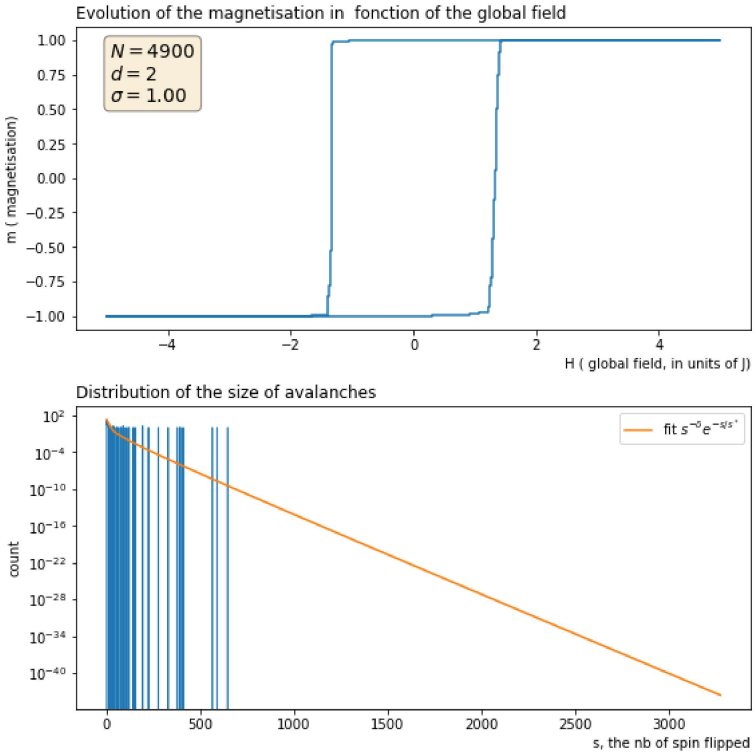
[5] Wald, Sascha Sebastian. Thermalisation and Relaxation of Quantum Systems. Diss. Université de Lorraine, 2017.

```
import numpy as np
import matplotlib.pyplot as plt
import math
import random
from scipy.optimize import curve_fit
import networkx as nx
import sys
sys.setrecursionlimit(10000)

sign = lambda x : math.copysign(1,x)
from IPython.core.display import display, HTML
display(HTML("<style>div.output_scroll { height: 44em; }</style>"))
```

```
def create_graph_lattice(L, d=2):  # creates a graph with L*L spins, here a lattice of dim d
    shape = [L for k in range(d)]
    res = - np.ones(shape)
    return res

def random_field(L,sigma=0.3,d=2): # for each spin its bias
    shape = [L for k in range(d)]
    rng = np.random.default_rng()
    return rng.normal(0,sigma,size=shape)


def decrease_energy(config,position,neighbours_mat, bias,global_field):  # in place function that minimize energy locally
    d=len(position)
    S=global_field+ bias[position]
    list_neighbours = neighbours_mat[position]
    for position_of_neighbours in list_neighbours :
        S+= config[tuple(position_of_neighbours)]
    config[position]=sign(S)



def compute_B_i(config,bias): # Computes every Bi of every point, square lattice only (linear algebra) !
    L,d =config.shape[0],len(config.shape)
    T=np.eye(L+2,k=1)+np.eye(L+2,k=-1) # 2d matrix to sum neighbours
    if d==1:
        S=np.zeros(L+2)
        S[1:L+1],S[0],S[-1]=config,config[-1],config[0]
        return bias + (T@S)[1:L+1]
    if d==2:
        temp = np.concatenate([ [config[-1]],config,[config[0] ]  ])
        S = np.concatenate( (np.array([temp[:,-1]]).T,temp,np.array([temp[:,0]]).T) ,axis=1)
        #print('this is S \n',S)
        #print('this is the spin configuration \n',config)
        return bias + (T@ S + S@T)[1:L+1,1:L+1]
    if d==3 :
        S = np.zeros((L+2,)*d)
        S[1:L+1,1:L+1,1:L+1] = config
        S[0,1:L+1,1:L+1], S[-1,1:L+1,1:L+1]= config[-1,:,:],config[0,:,:]
        S[1:L+1,0,1:L+1],S[1:L+1,-1,1:L+1]=config[:,-1,:],config[:,0,:]
        S[1:L+1,1:L+1,0],S[1:L+1,1:L+1,-1]=config[:,:,-1],config[:,:,0]
        #print('this is S \n',S)
        #print('this is the spin configuration \n',config)
        big =np.einsum('il,ljk->ijk',T,S)+np.einsum('jl,ilk->ijk',T,S)+np.einsum('kl,ijl->ijk',T,S)
        return bias + big[1:L+1,1:L+1,1:L+1]
    else :
        print('Careful, the dimension seems to be too big')



def spin_flips(config,bias): #in place, flips the necessary spins at constant global field  ( bc we shifted it)
    B=compute_B_i(config,bias)
    print('we get in the loop of flips')
    compteur = 0
    while np.any(B*config<= 0):  #condition, important !
        ind = np.unravel_index(np.argmin(B*config, axis=None), B.shape)
        config[ind] *= -1
        B=compute_B_i(config,bias)
        compteur +=1
        #print(B)
    #print('out of the loop of flips,we flipped '+str(compteur)+' spin(s)')


def spin_flips_recursive(config,bias,compteur=0):
    B=compute_B_i(config,bias)
```

```
    if np.any(B*config < 0) :
        ind = np.unravel_index(np.argmin(B*config, axis=None), B.shape)
        #print(B[ind],ind)
        config[ind] *= -1
        spin_flips_recursive(config,bias,compteur+1)
    #else :
        #print('out of the loop of flips,we flipped '+str(compteur)+' spin(s)')



d=2
J=1
L=70
N=L**d
sigma = 1


def loop_finite(d,J,L,N,sigma): # loop for when we wait for thermalisation, not efficient but random processes
    initial_state=create_graph_lattice(L,d)
    neighbours_mat=create_neighbours_mat(initial_state)
    bias = random_field(L,sigma,d)
    list_H = np.concatenate( [np.arange(-2*d-1,2*d+1,0.1),np.arange(2*d+1,-2*d-1,-0.1)])
    list_m=[]
    for H in list_H:
        for k in range(10*N):
            position = tuple(random.randint(0,L-1) for k in range(d)) #chooses which spin to flip
            decrease_energy(initial_state,position,neighbours_mat,bias,H)
        m = initial_state.sum()/initial_state.size
        list_m.append(m)

    fig=plt.figure(figsize=(8,8))
    plt.plot(list_H,list_m)
    plt.title('Evolution of the magnetisation in  fonction of the global field')
    plt.xlabel('H ( global field, in units of J)')
    plt.ylabel('m ( magnetisation)')


def next_event_loop(d,J,L,N,sigma): # next event simulation of H compared to the Bi's
    initial_state=create_graph_lattice(L,d)
    bias = random_field(L,sigma,d)
    list_H=[-2*d-1]
    list_m=[]
    list_s=[]  #list of size of avalanches
    m = initial_state.sum()/initial_state.size
    list_m.append(m)
    while m < 1- 1e-3:
        B=compute_B_i(initial_state,bias+list_H[-1])
        #print('This is the matrix of B_i \n',B)
        if np.any(B<0):
            list_H.append(list_H[-1]-B[B<0].max()+1e-8)  #### minimum value of H to cause a single flip, important !
            #print('we flip spins at H=',list_H[-1])
            #spin_flips(initial_state,bias+list_H[-1])
            size_avalanche=initial_state.sum()
            spin_flips_recursive(initial_state,bias+list_H[-1])
            size_avalanche=abs(initial_state.sum()-size_avalanche)/2
            list_s.append(size_avalanche)
            list_H.append(list_H[-1])
            list_m.append(m)
            m = initial_state.sum()/initial_state.size
            list_m.append(m)
    m = initial_state.sum()/initial_state.size
    list_m.append(m)
    list_H.append(2*d+1)
    while m > -1+ 1e-3:
        B=compute_B_i(initial_state,bias+list_H[-1])
        #print('This is the matrix of B_i \n',B)
        if np.any(B>0):
            list_H.append(list_H[-1]-B[B>0].min()-1e-8)  #### minimum value of H to cause a single flip, important !
            #print('we flip spins at H=',list_H[-1])
            #spin_flips(initial_state,bias+list_H[-1])
            size_avalanche=initial_state.sum()
            spin_flips_recursive(initial_state,bias+list_H[-1])
            size_avalanche=abs(initial_state.sum()-size_avalanche)/2
            list_s.append(size_avalanche)
            list_H.append(list_H[-1])
            list_m.append(m)
            m = initial_state.sum()/initial_state.size
            list_m.append(m)
            #condition = list_H[-1]!=list_H[-2]
    list_m.append(-1)
    list_H.append(-2*d-1)
```

```
    fig,(ax1,ax2)=plt.subplots(2,figsize=(8,8))
    ax1.plot(list_H,list_m)
    ax1.set_title('Evolution of the magnetisation in  fonction of the global field',loc='left')
    ax1.set_xlabel('H ( global field, in units of J)',loc='right')
    ax1.set_ylabel('m ( magnetisation)')
    s=np.array(list_s)
    bins=np.arange(s.min()-0.5,s.max()+0.5,1)
    counts,bins=np.histogram(s,bins=bins)
    ax2.stairs(counts,bins)
    f=lambda s,A,tau,s_lifetime :A* s**(-tau)*np.exp(-(s-1)/s_lifetime)
    bins_centers=(bins[0:bins.shape[0]-1]+bins[1:bins.shape[0]])/2
    popt,pcov=curve_fit(f,bins_centers,counts,p0=(counts[0],1,1))
    X=np.linspace(s.min(),s.max(),100)
    ax2.plot(X,f(X,popt[0],popt[1],popt[2]),label='fit $s^{-\delta}e^{-s/s^*}$')
    ax2.set_title('Distribution of the size of avalanches',loc='left')
    ax2.set_ylabel('count')
    ax2.set_xlabel('s, the nb of spin flipped',loc='right')
    textstr = '\n'.join((
    r'$N=%.0f$' % (N, ),
    r'$d=%.0f$' % (d, ),
    r'$\sigma=%.2f$' % (sigma, )))
    # these are matplotlib.patch.Patch properties
    props = dict(boxstyle='round', facecolor='wheat', alpha=0.5)

    # place a text box in upper left in axes coords
    ax1.text(0.05, 0.95, textstr, transform=ax1.transAxes, fontsize=14,
            verticalalignment='top', bbox=props)
    fig.tight_layout(pad=1.0)
    plt.legend()
    plt.yscale('log')
    plt.show()
%matplotlib inline
next_event_loop(d,J,L,N,sigma)
```

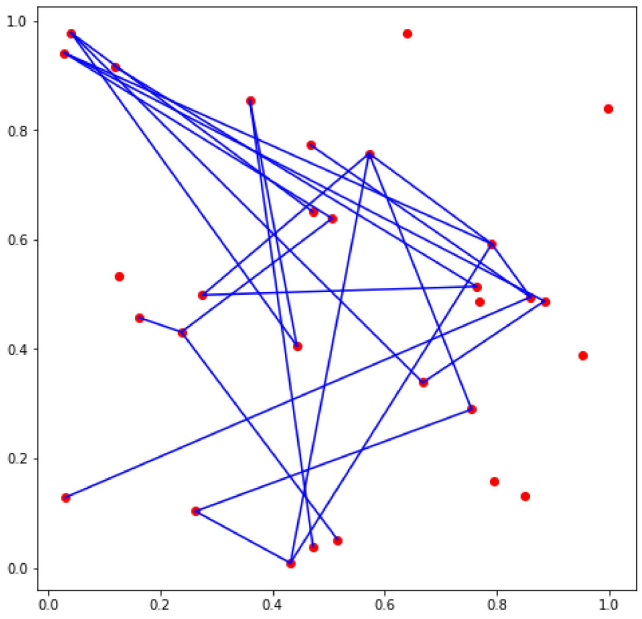


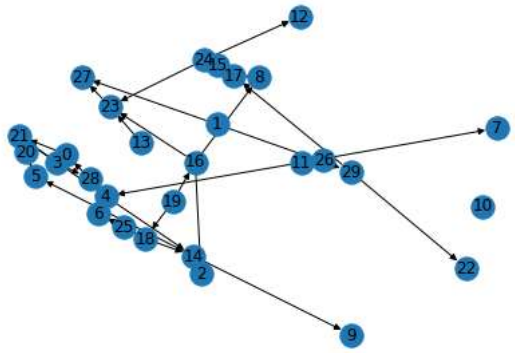

```
def create_neighbours_erdos_naive(M,C=3):
    N=M.shape[0]
    res =np.zeros(N,dtype=object)
    for k in range(N):
        neig_k=[]
        for q in range(N):
            if np.random.uniform() < C/N and q!=k:
                neig_k.append(q)
        res[k]=np.array(neig_k)
    return res

def create_neighbours_erdos_mat(M,C=3):  #using linear algebra to count neighbours
    N=M.shape[0]
    rng = np.random.default_rng()
```

```
    return ((rng.uniform(size=(N,N))+np.eye(N))<C/N).astype(int)


def compute_erdos_naive_B_i(config,bias,neighbours): # knowing the neighbours, we compute Bi, config is 1D!
    N=config.shape[0]
    B=np.zeros(N)
    for k in range(N):
        neigh_k = neighbours[k]
        for q in neigh_k:
            B[k]+=config[q]
    return B +bias


def compute_erdos_mat_B_i(config,bias,neighbours): # knowing the neighbours, we compute Bi, config is 1D!
    return neighbours@config + bias


def spin_flips_recursive_erdos_naive(config,bias,neighbours,compteur=0):
    B=compute_erdos_naive_B_i(config,bias,neighbours)
    if np.any(B*config < 0) :
        ind = np.unravel_index(np.argmin(B*config, axis=None), B.shape)
        #print(B[ind],ind)
        config[ind] *= -1
        compteur+=1
        spin_flips_recursive_erdos_naive(config,bias,neighbours,compteur)
    #else :
     #   print('out of the loop of flips,we flipped '+str(compteur)+' spin(s)')


def spin_flips_recursive_erdos_mat(config,bias,neighbours,compteur=0):
    B=compute_erdos_mat_B_i(config,bias,neighbours)
    if np.any(B*config < 0) :
        ind = np.unravel_index(np.argmin(B*config, axis=None), B.shape)
        #print(B[ind],ind)
        config[ind] *= -1
        compteur+=1
        spin_flips_recursive_erdos_mat(config,bias,neighbours,compteur)
    #else :
     #   print('out of the loop of flips,we flipped '+str(compteur)+' spin(s)')

def draw_erdos_naive(N=30,C=1.):
    points = np.array([ (np.random.uniform(),np.random.uniform()) for k in range(N)]) # create N points in a 2D plane
    neigh=create_neighbours_erdos_naive(points,C)
    fig=plt.figure(figsize=(8,8))
    nb_of_links=0
    for k in range(N):
        neigh_k = neigh[k]
        for q in neigh_k:
            plt.plot([points[k][0],points[q][0]],[points[k][1],points[q][1]],'-',color='blue')
        plt.scatter(points[k][0],points[k][1],color='r')
        nb_of_links+=neigh[k].shape[0]
    print('in total, we have ',nb_of_links,' links, for C*N = ',C,'x',N)


def draw_erdos_nice(config,neighbours):
    fig = plt.figure()
    N=config.shape[0]
    tdict = {str(k):{str(q):{'information'} for q in neighbours[k]} for k in range(N)}
    print(tdict)
    G = nx.DiGraph()
    def create_graph(d, g, p = None):
        for a, b in d.items():
            g.add_node(a)
            if p is not None:
                g.add_edge(p, a)
            if not isinstance(b, set):
                create_graph(b, g, a)
    create_graph(tdict, G)
    G=G.reverse()
    print(len(G.edges))
    nx.draw(G, with_labels = True)
    plt.show()

def from_mat_to_list(mat):
    n,p=mat.shape
    neigh = np.zeros(n,dtype='object')
    for i in range(n):
        for j in range(p):
            neigh_i=[]
            if mat[i,j]==1:
                neigh_i.append(j)
            neigh[i]=np.array(neigh_i)
```

```
    return neigh

%matplotlib inline
draw_erdos_naive()
draw_erdos_nice(np.zeros(30),create_neighbours_erdos_naive(np.zeros(30),1))
```

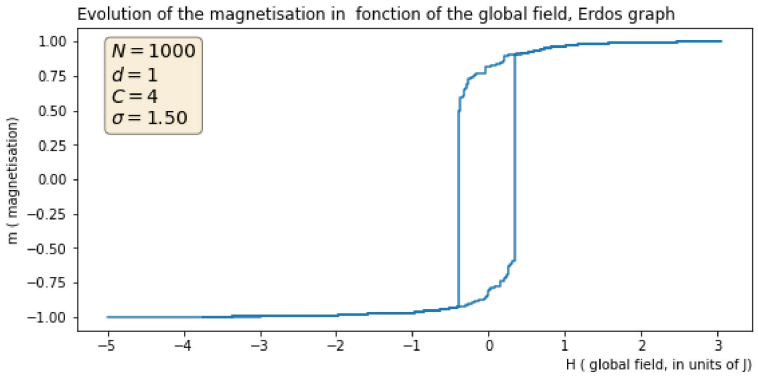
```
in total, we have  24  links, for C*N =  1.0 x 30
{'0': {'28': {'information'}}, '1': {}, '2': {'16': {'information'}}, '3': {}, '4': {
34
```

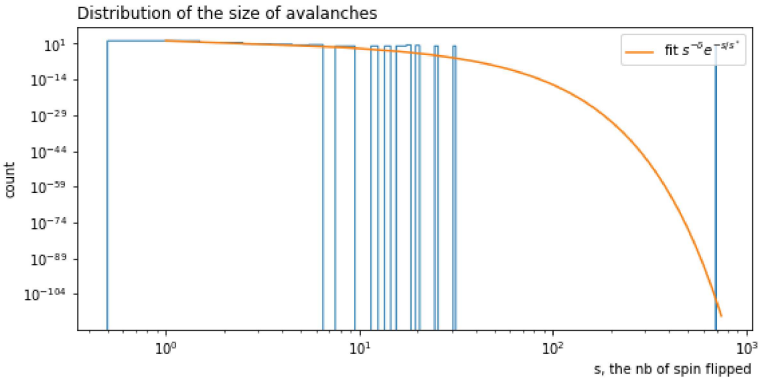





```
d=1
J=1
L=1000
N=L**d
sigma = 1.5
C=4

def next_event_erdos_mat(d,J,L,N,sigma,C): # next event simulation of H compared to the Bi's
    initial_state=create_graph_lattice(L,d)
    bias = random_field(L,sigma,d)
    neighbours = create_neighbours_erdos_mat(initial_state,C)
    #plt.imshow(neighbours)
    #draw_erdos_nice(initial_state,from_mat_to_list(neighbours))
    list_H=[-2*d-3]
    list_m=[]
    list_s=[]  #list of size of avalanches
    m = initial_state.sum()/initial_state.size
    list_m.append(m)
    while m < 1- 1e-3:
        B=compute_erdos_mat_B_i(initial_state,bias+list_H[-1],neighbours)
        #print('This is the matrix of B_i \n',B)
        if np.any(B<0):
            list_H.append(list_H[-1]-B[B<0].max()+1e-8)  #### minimum value of H to cause a single flip, important !
            #print('we flip spins at H=',list_H[-1])
            #spin_flips(initial_state,bias+list_H[-1])
            size_avalanche=initial_state.sum()
            spin_flips_recursive_erdos_mat(initial_state,bias+list_H[-1],neighbours)
            size_avalanche=abs(initial_state.sum()-size_avalanche)/2
            list_s.append(size_avalanche)
            list_H.append(list_H[-1])
```

```
            list_m.append(m)
            m = initial_state.sum()/initial_state.size
            list_m.append(m)

    m = initial_state.sum()/initial_state.size
    list_m.append(m)
    list_H.append(2*d+1)
    while m > -1+ 1e-3:
        B=compute_erdos_mat_B_i(initial_state,bias+list_H[-1],neighbours)
        #print('This is the matrix of B_i \n',B)
        if np.any(B>0):
            list_H.append(list_H[-1]-B[B>0].min()-1e-8)  #### minimum value of H to cause a single flip, important !
            #print('we flip spins at H=',list_H[-1])
            #spin_flips(initial_state,bias+list_H[-1])
            size_avalanche=initial_state.sum()
            spin_flips_recursive_erdos_mat(initial_state,bias+list_H[-1],neighbours)
            size_avalanche=abs(initial_state.sum()-size_avalanche)/2
            list_s.append(size_avalanche)
            list_H.append(list_H[-1])
            list_m.append(m)
            m = initial_state.sum()/initial_state.size
            list_m.append(m)
            #condition = list_H[-1]!=list_H[-2]
    list_m.append(-1)
    list_H.append(-2*d-1)
    fig,(ax1,ax2)=plt.subplots(2,1,figsize=(8,8))
    ax1.plot(list_H,list_m)
    ax1.set_title('Evolution of the magnetisation in  fonction of the global field, Erdos graph',loc='left')
    ax1.set_xlabel('H ( global field, in units of J)',loc='right')
    ax1.set_ylabel('m ( magnetisation)')
    s=np.array(list_s)
    bins=np.arange(s.min()-0.5,s.max()+0.5,1)
    counts,bins=np.histogram(s,bins=bins)
    ax2.stairs(counts,bins)
    f=lambda s,A,tau,s_lifetime :A* s**(-tau)*np.exp(-(s-1)/s_lifetime)
    bins_centers=(bins[0:bins.shape[0]-1]+bins[1:bins.shape[0]])/2
    popt,pcov=curve_fit(f,bins_centers,counts)
    X=np.linspace(s.min(),s.max(),100)
    ax2.plot(X,f(X,popt[0],popt[1],popt[2]),label='fit $s^{-\delta}e^{-s/s^*}$')
    ax2.set_title('Distribution of the size of avalanches',loc='left')
    ax2.set_ylabel('count')
    ax2.set_xlabel('s, the nb of spin flipped',loc='right')
    ax2.legend(fontsize=22)
    textstr = '\n'.join((
    r'$N=%.0f$' % (N, ),
    r'$d=%.0f$' % (d, ),
    r'$C=%.0f$' % (C, ),
    r'$\sigma=%.2f$' % (sigma, )))
    # these are matplotlib.patch.Patch properties
    props = dict(boxstyle='round', facecolor='wheat', alpha=0.5)

    # place a text box in upper left in axes coords
    ax1.text(0.05, 0.95, textstr, transform=ax1.transAxes, fontsize=14,
            verticalalignment='top', bbox=props)
    fig.tight_layout(pad=1.0)
    plt.legend()
    plt.yscale('log')
    plt.xscale('log')
    plt.show()

%matplotlib inline
next_event_erdos_mat(d,J,L,N,sigma,C)
```


Evolution of the magnetisation in fonction of the global field, Erdos graph
$N = 1000$
$d = 1$
$C = 4$
$\sigma = 1.50$
m (magnetisation)
H (global field, in units of J)


Distribution of the size of avalanches
fit $s^{-\delta}e^{-s/s^*}$
count
s, the nb of spin flipped