Homework 6

December 9, 2022

1 Spatial Correlations and Dynamical exponent

1.0.1 Submitted by : Anushka and Yashasvee

1.0.2 Answer 1 : Is C biased?

According to me, C is not biased. Intuitively, we talked about m not being zero and how to resolve the issue. If we pick up any spin, it is possible it will be surrounded by similar spin configurations, so flipping it does not make much sense but that is when we introduced this two point correlation function which is translationally invariant. So even if m is biased, C is not biased.

1.0.3 Answer 2 : C for r = 0

According to the convolution formula, if r=0 then the overlap is 100 percent, which means that we just sum over the same site and get Λ , which in turn cancels from the one in denominator and hence, C is 1 for r = 0

1.0.4 Answer 3 : Implementing C via convolution

[1]:
```
# What is convolution? A convolution is an integral that expresses the amount␣
 ↪of overlap of one function as it is
# shifted over another function.  It therefore "blends" one function with␣
 ↪another.
```

[127]:
```
# Necessary modules

%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import random as r
from random import choice

# We now work to generalize the Ising model to 2-Dimensions and or that the␣
 ↪lattice size becomes (N_x) X (N_y) where we
# assume periodic boundary conditions in both, x and y, directions. It is given␣
 ↪that we use the same coupling constant in
# both directions.

# Number of sweeps for thermalization for every value of J,h
```

```
n_therm = 10000

# Number of measurements for each J,h
n_meas = 10000

# External magnetic field
h = 0
```

[ ]:

[128]:
```
# add the sum thing, check notes. the above thing is just the fft of the integral, you gotta sum it !
# equation 6 in homework sheet!

# wrong probably
# n=5
# C_r = (1/n**2)*np.sum(C)
```

[129]:
```
J_c = 0.44068 # Critical J
```

[130]:
```
# To calculate change in energy after one spin flips at position (x,y)

# This is the general formula and we have already defined h=0 above
def energy_flip(s,J,h,x,y,n):
#     Here, the s is the spin, x and y are th positions of the 2 D lattice, J,n, and h are the same variables as defined
#     before.
#     The function calculates the energy after the spin at one site is flipped.
    return 2 * s[x][y]*(J*(s[((x+1)%n)][y] + s[((x-1)%n)][y] + s[x][((y+1)%n)] + s[x][((y-1)%n)]) + h)


def energy(s,J,h,x,y,n):
    return -J*((s[((x+1)%n)][y] + s[((x-1)%n)][y] + s[x][((y+1)%n)] + s[x][((y-1)%n)])*s[x][y])-h*s[x][y]
```

[131]:
```
#  Beginning by defining the two point correlation function

# Taken from the discussion in the tutorial

# This function returns the fft values and takes care of k and -k indices in the summation

# skp = +k and skm = -k
n=5
s = [[choice((+1,-1)) for x in range(n)] for y in range(n)]
def C(s,n):
```

```
#     this is the equation 5 of homework sheet

    skp = np.fft.fft2(s, norm = 'forward')    # 2 D fft
    # print(skp)
    # print(skp.shape)
    skm = np.fft.ifft2(s, norm = 'backward')  # 2 D ifft
    return np.fft.fft2(skm*skp, norm = 'backward')/n**2
```

[132]:
```
C(s,n)
```

[132]:
```
array([[ 0.04  +0.j,  0.0016+0.j,  0.008 +0.j,  0.008 +0.j,  0.0016+0.j],
       [ 0.008 +0.j, -0.0048+0.j, -0.0048+0.j, -0.0048+0.j, -0.0112+0.j],
       [-0.0048+0.j,  0.008 +0.j,  0.0016+0.j,  0.0016+0.j,  0.0016+0.j],
       [-0.0048+0.j,  0.0016+0.j,  0.0016+0.j,  0.0016+0.j,  0.008 +0.j],
       [ 0.008 +0.j, -0.0112+0.j, -0.0048+0.j, -0.0048+0.j, -0.0048+0.j]])
```

[137]:
```
# fft of sx and sy

def variables(n,J,h,r):

# Defining local arrays

    # m = np.array([])  # Magnetization
    # E = np.array([])  # Energy
    # m_absolute = np.array([]) # Absolute value of magnetization
    prob = np.array([]) # Probability information
    Conv = np.array([])
#   Now we need to assign spins to the site and because this is a 2D lattice, we have to keep in mind the dimensions
#   x and y and do that. This has been done as follows:
#   The variable (s) is the spin here and n is some random integer

    s = [[choice((+1,-1)) for x in range(n)] for y in range(n)]

    for j in range(n_therm):
#         Assigning a random integer value to x and y
        x = np.random.randint(n)
        # r = np.random.randint(n)
        y= (x+r) % n  # Implementing the Translational Property

#   We need to define the change of energy after flipping the lattice site (x,y) picked randomly. We call the defined
#   function energy_flip to calculate that.
        # print(x,y)
        delta_energy = energy_flip(s,J,h,x,y,n)
        if delta_energy < 0:
```

```
            s[x][y] *= -1          # Condition to accept the spin flip
        else:
            if np.random.uniform(0,1) <= np.exp(-delta_energy):
                s[x][y] *= -1

    for i in range(n_meas):
        newarray = np.zeros([n,n])

        for k in range(n):              #sweeping the lattice
            # x = j % n
            # y = j // n
            x = k
            y = (k + r) % n
            delta_energy = energy_flip(s,J,h,x,y,n)
            if delta_energy < 0:
                s[x][y] *= -1          # Condition to accept the spin flip
            else:
                if np.random.uniform(0,1) <= np.exp(-delta_energy):
                    s[x][y] *= -1
                    prob = np.append(prob,1.)    # Accept.
                else:
                    prob = np.append(prob,0.)     # We reject the other values.
        newarray = C(s,n)
            # Cxy = C(s[x][y],n)
        C_r=0
        for k in range(n):
            x = k
            y = (k+r) % n
            C_r += newarray[x][y]
        # if obs == "Conv"
        Conv = np.append(Conv,C_r)
    # print(Conv)
    obs=np.mean(Conv)


    return obs
```

[138]:
```
# Q : also, for 3, i have the function for the convoluted c but how do i call␣
 ↪it? i have to show it is 1 for r=0 but here
# technically y = x + r for understanding
# so do something like x+y where y is r but then operator assignment error.
# how do i take out the value of r.

# Q : so i have my function, here i just do that with metropolis hastings so␣
 ↪how do i measure it? just take the c
```

```
# function as the obs and append in the array and then i put different n for␣
 ↪answer 4.
# My understanding is then i will c as my obs and i can plot like i did before,␣
 ↪is that it?
# DO after confirmation
```

[139]:
```
h = 0
c_3  = variables(3,J_c,h,0)
print(c_3)
```

```
(0.07387654320987654+0j)
```

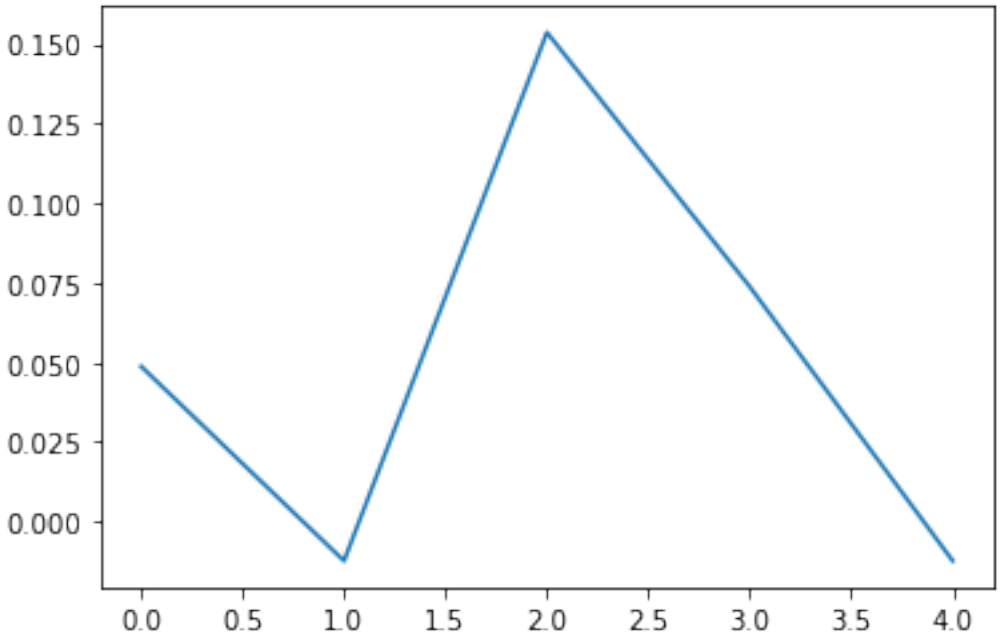
1.0.5 Answer 4 : Behaviour of C as a function of r for different N

[140]:
```
# something like this

J_c = 0.44068
h = 0
r_range = np.arange(0,5,1)

c_3  = [variables(3,J_c,h,r) for r in r_range]
c_7  = [variables(7,J_c,h,r) for r in r_range]
c_11 = [variables(11,J_c,h,r) for r in r_range]
c_15 = [variables(15,J_c,h,r) for r in r_range]
c_19 = [variables(19,J_c,h,r) for r in r_range]
c_23 = [variables(23,J_c,h,r) for r in r_range]
```

[141]:
```
# print()
```

[148]:
```
plt.figure()
plt.plot(c_3)
```

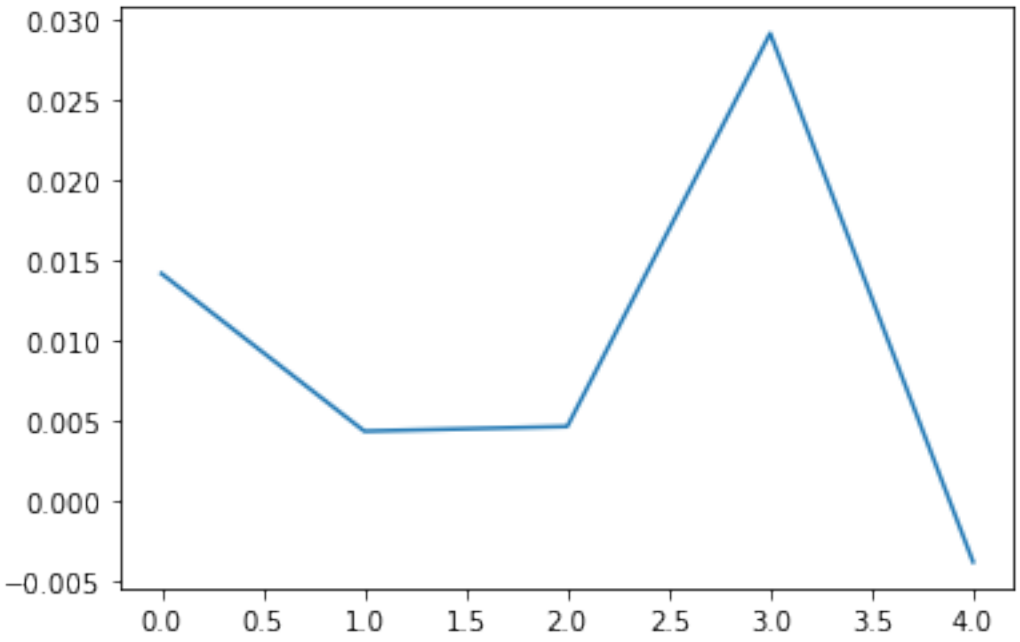
[148]:
```
[<matplotlib.lines.Line2D at 0x7f98d57c4670>]
```



2/5

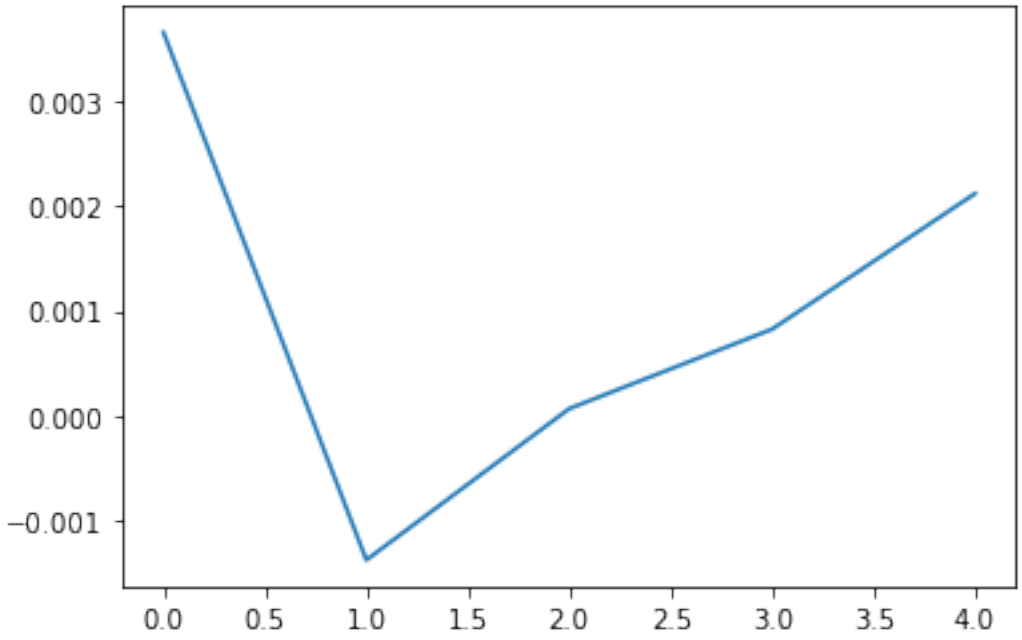
[143]:
```
plt.figure()
plt.plot(c_7)
```

[143]: [<matplotlib.lines.Line2D at 0x7f98cd4108e0>]



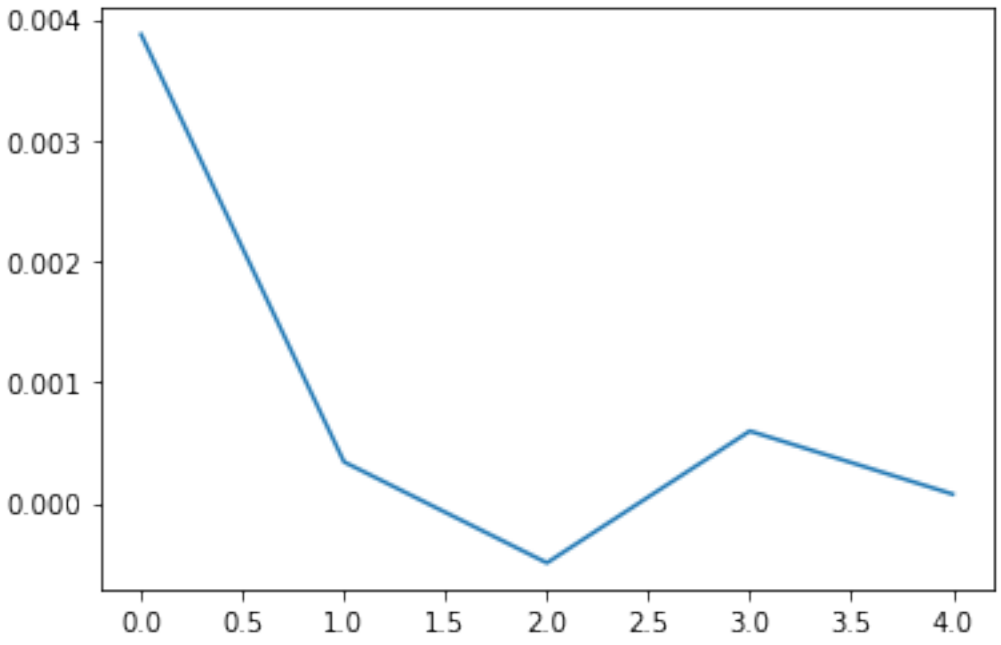
[144]:
```
plt.figure()
plt.plot(c_11)
```

[144]: [<matplotlib.lines.Line2D at 0x7f98cd37bbb0>]



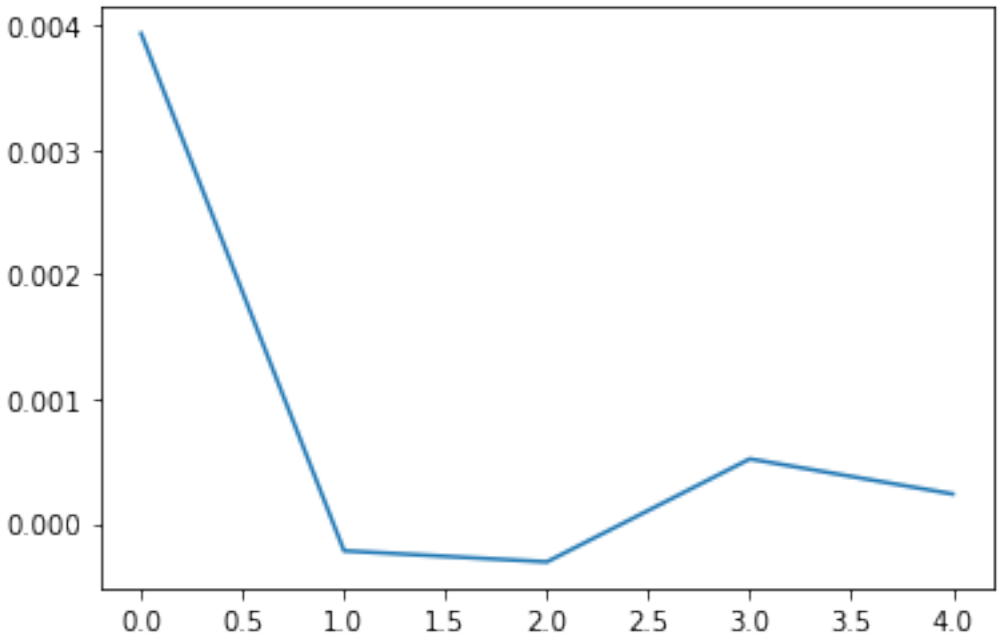
[145]:
```
plt.figure()
plt.plot(c_15)
```

[145]: [<matplotlib.lines.Line2D at 0x7f98cd2e49d0>]



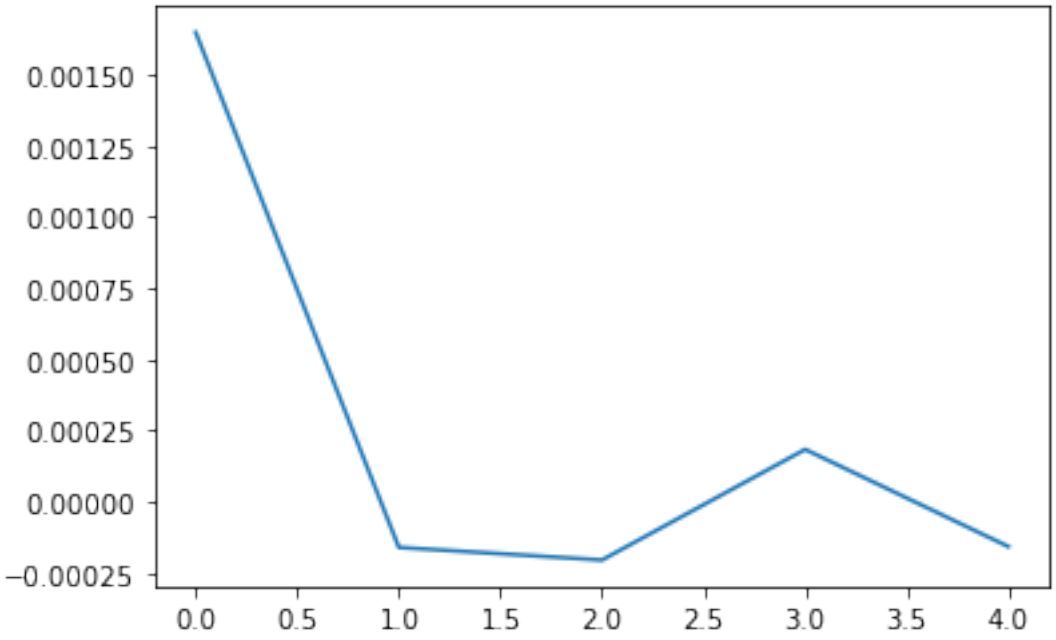
[146]:
```
plt.figure()
plt.plot(c_19)
```

[146]: [<matplotlib.lines.Line2D at 0x7f98cd2ce4c0>]



[147]:
```
plt.figure()
plt.plot(c_23)
```

[147]: [<matplotlib.lines.Line2D at 0x7f98cd2362b0>]



1.0.6 Answer 5 : Auto correlation for absolute magnetization

In the other notebook. Sorry again.