GMDL, HW #1

Oren Yacouel - 208727164
Gal Alon - 318512910
Jonathan Lewittes - 336132014

Exercise 3:

Temp	ZTemp
1	121.23293134406595
1.5	40.922799092745386
2	27.048782764334526

Exercise 4:

Temp	ZTemp
1	365645.7491357704
1.5	10565.42198351426
2	2674.518123060087

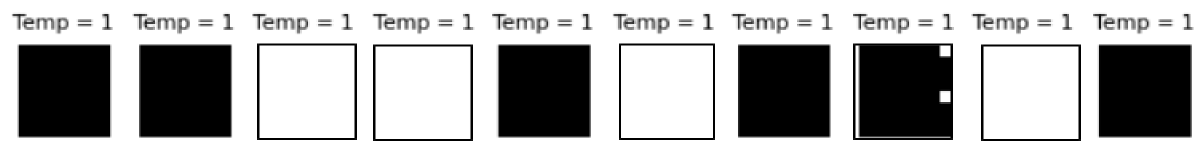
Exercise 5: for 2X2 Lattice

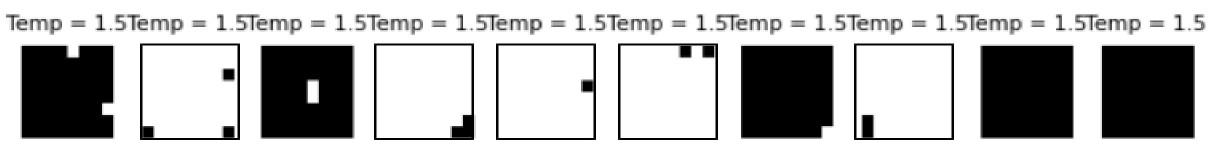
Temp	ZTemp
1	121.23293134406595
1.5	40.922799092745386
2	27.048782764334526

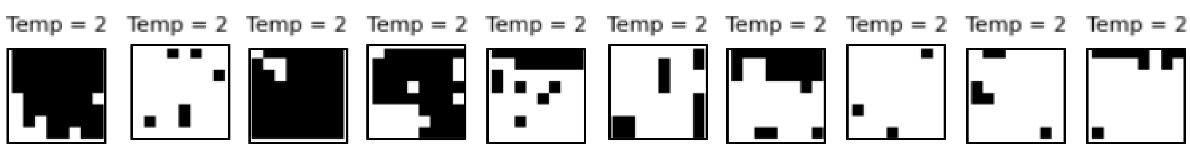
Exercise 6: for 3X3 Lattice

Temp	ZTemp
1	365645.7491357704
1.5	10565.421983514265
2	2674.518123060087

Exercise 7:







<u>Problem 1:</u>

The most likely programming bug is that the student encountered is related to integer division in Python 2. When dividing two integers in Python 2, the result is an integer, which means that any fractional part of the result is truncated. For example, 3/2 in Python 2 would evaluate to 1, whereas in Python 3 it would evaluate to 1.5.

When Temp is set to 2, this ratio would be less than 1.0 for many pairs, which would cause 1/Temp = ½, which is interpreted as 0 due to the integer division. The effect of this bug is that all the probabilities are the same in the samples of each pixel in the image and therefore as we learned in class we obtain the uniform distribution.

Exercise 8:

Temp	$E(X_{(1,1)}X_{(2,2)})$	$E(X_{(1,1)}X_{(8,8)})$
1	0.958	0.904
1.5	0.77	0.542
2	0.494	0.058

<u>Problem 2:</u>
It can be seen that when the temperature is lower, then the chance that $X_{(1,1)} = X_{(2,2)}$ increases (and similarly for $X_{(1,1)} = X_{(8,8)}$). This is because, as we learned in class, as the temperature -> 0, all values of points in the lattice will be the same.

In addition, it can be seen that the chance that X (2,2) = X (1,1) is greater than the chance that X (8,8) = X (1,1) because as the distance between 2 points in the lattice increases, the dependence between them decreases. Therefore we can conclude, accordingly:
$E(X_{(1,1)}X_{(2,2)}) \geq E(X_{(1,1)}X_{(8,8)})$

Exercise 9:

Method 1 Results:

Temp	$E(X_{(1,1)}X_{(2,2)})$	$E(X_{(1,1)}X_{(8,8)})$
1	0.936	0.538
1.5	0.75	0.356
2	0.516	0.108

Method 2 Results:

Temp	$E(X_{(1,1)}X_{(2,2)})$	$E(X_{(1,1)}X_{(8,8)})$
1	0.951	0.902
1.5	0.758	0.544
2	0.497	0.108

Problem 3:

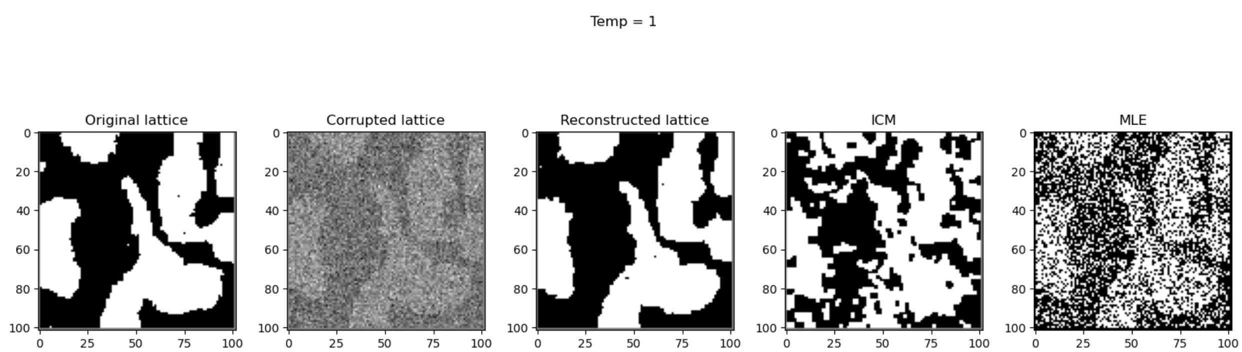
As we can see in our results, method 2 better resembles the distribution which we portray in exercise 8.

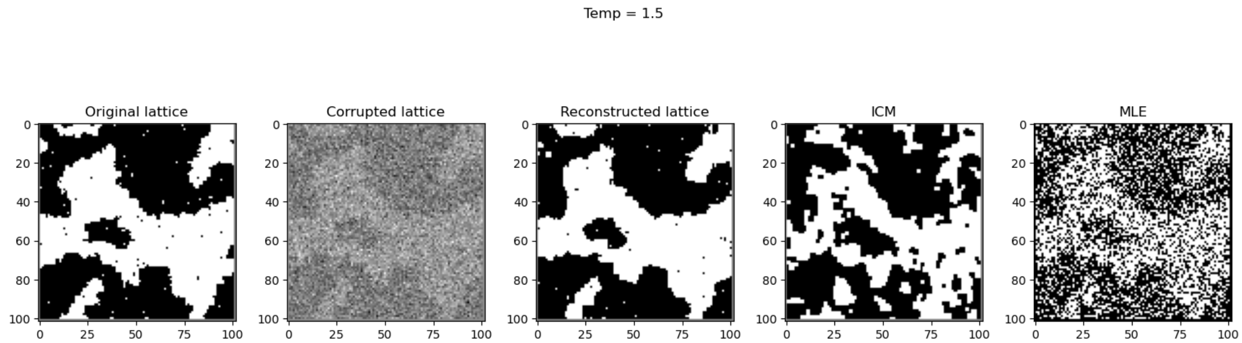
That is what we expected, because in the class we saw that in Gibbs Sampling:

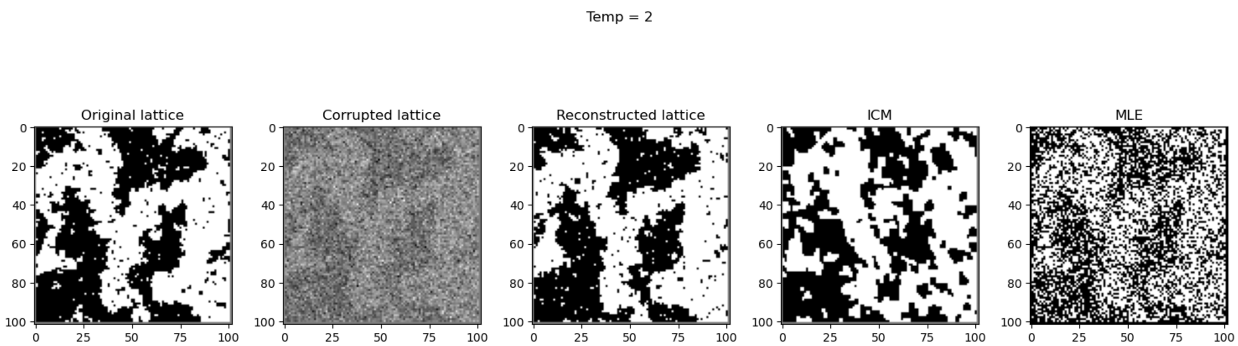
$p(x^{[t]}, y^{[t]}, z^{[t]}) \rightarrow (as\ t \rightarrow \infty)\ p(x, y, z)$

In method 2 we make only one random initialization and iterate 25,000 sweeps, which is a much larger number of iterations (or sweeps) than the 25 sweeps we made in method 1 for each random initialization of the 10,000.

Exercise 10:




Temp = 1.5
Original lattice
Corrupted lattice
Reconstructed lattice
ICM
MLE
0 20 40 60 80 100
0 25 50 75 100


Temp = 2
Original lattice
Corrupted lattice
Reconstructed lattice
ICM
MLE
0 20 40 60 80 100
0 25 50 75 100

1.1 The Clique Functions in the Ising Model

```
#Computer Exercise 1
import numpy as np
def G(row_s: np.ndarray, Temp: float):
    return np.exp(np.sum(row_s[:-1] * row_s[1:]) / Temp)


#Computer Exercise 2

def F(row_s: np.ndarray, row_t: np.ndarray, Temp: float):
    return np.exp(np.sum(row_s * row_t) / Temp)
```

1.2 Brute Force on Small Lattices Some of the computer exercises in this section require you to use an absurd number (from a programmer's perspective) of nested loops. Thus, you can (but do not have to) exploit Python's itertools module to make this a bit more elegant. compute ZTemp (for three different values of Temp where Temp ∈ {1, 1.5, 2}) using brute force (use 4 nested For loops, one for each of the xs's, the looping is done over the values that xs can take: {−1, 1}). To help you debug: For Temp = 1, your result should be ZTemp = 121.23 . . ..

```
#Computer Exercise 3
from itertools import product

def compute_ZTemp1(Temp: float):
    Z = 0
    vals = [-1, 1]

    for x11 in vals:
        for x12 in vals:
            for x21 in vals:
                for x22 in vals:
                    Z += np.exp((x11*x21+ x12*x22 + x11*x12 +
x21*x22)/Temp)

    return Z

temp = [1,1.5,2]
for i in temp:
    z = compute_ZTemp1(i)
    print(z)


121.23293134406595
40.922799092745386
27.048782764334526

#Computer Exercise 4
```

```
import numpy as np

import numpy as np

def compute_ZTemp2(Temp: float):
    Z = 0
    vals = [-1, 1]

    for x11 in vals:
        for x12 in vals:
            for x13 in vals:
                for x21 in vals:
                    for x22 in vals:
                        for x23 in vals:
                            for x31 in vals:
                                for x32 in vals:
                                    for x33 in vals:
                                        energy = x11*x21 + x12*x22 +
x13*x23 + x21*x31 + x22*x32 + x23*x33 + x11*x12 + x12*x13 + x21*x22 +
x22*x23 + x31*x32 + x32*x33
                                        Z += np.exp(energy/Temp)

    return Z




temp = [1,1.5,2]
for i in temp:
    z = compute_ZTemp2(i)
    print(z)
```

```
365645.74913577037
10565.421983514265
2674.518123060087
```

```
# computer exercise 5

def y2row(y, width=2):
    """
    y: an integer in (0,...,(2**width)-1)
    """
    if not 0 <= y <= (2 ** width) - 1:
        raise ValueError(y)
    my_str=np.binary_repr(y,width=width)
    # my_list = map(int,my_str) # Python 2
    my_list = list(map(int,my_str)) # Python 3
    my_array = np.asarray(my_list)
    my_array[my_array==0]=-1
    row = my_array
```

```
    return row

def compute_ZTemp3(Temp: float):
    Z = 0
    ys = [0,1,2,3]
    rows = []
    for y in ys:
        rows.append(y2row(y, width=2))                       # convert
to row vectors

    rows = np.asarray(rows)                                  # convert
to numpy array

    for y1 in rows:
        for y2 in rows:
            Z += G(y1, Temp) * G(y2, Temp) * F(y1, y2, Temp) #
required logic

    return Z

temp = [1,1.5,2]
for i in temp:
    z = compute_ZTemp3(i)
    print(z)
```

```
121.23293134406595
40.922799092745386
27.048782764334526
```

```
# computer exercise 6
def compute_ZTemp4(Temp: float):
    Z = 0
    ys = [0,1,2,3,4,5,6,7]
    rows = []
    for y in ys:
        rows.append(y2row(y, width=3))                       # convert
to row vectors

    rows = np.asarray(rows)                                  # convert
to numpy array

    for y1 in rows:
        for y2 in rows:
            for y3 in rows:
                Z += G(y1, Temp) * G(y2, Temp) * G(y3, Temp) * F(y1,
y2, Temp) * F(y2, y3, Temp)
    return Z

temp = [1,1.5,2]
for i in temp:
```

```
    z = compute_ZTemp4(i)
    print(z)
```

```
365645.7491357704
10565.421983514265
2674.518123060087
```

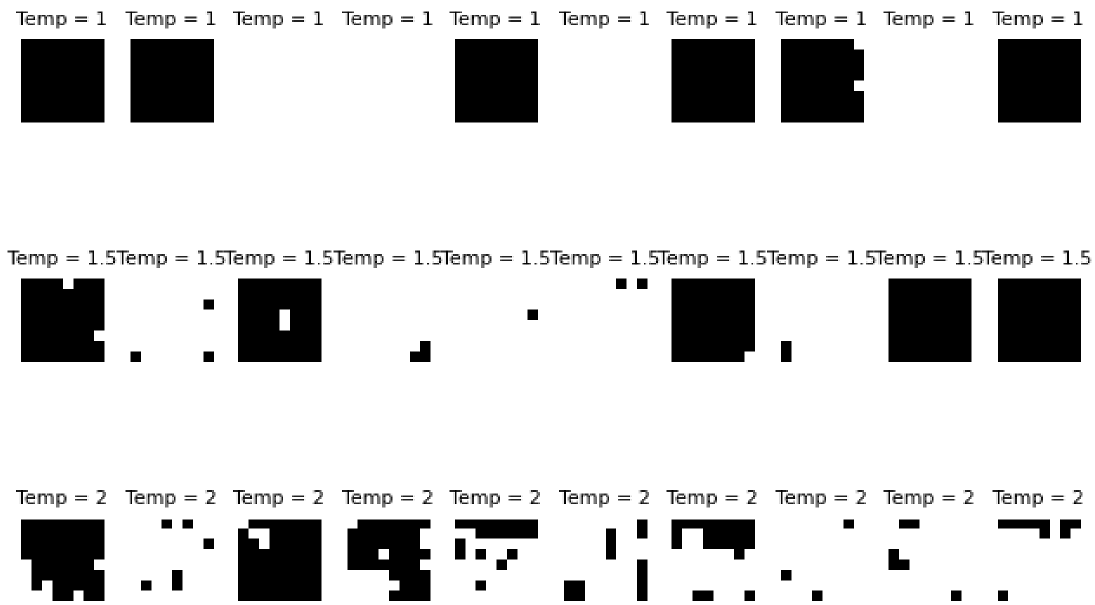
1.3 Dynamic Programming on an 8×8 Lattice

```
#computer exercise 7

import numpy as np


def get_Gs(size: int, Temp: float):
    """
    size: size of the lattice
    Temp: temperature
    """
    G_MAT = np.asarray([G(y2row(i, width=size), Temp) for i in range(2
** size)])
    return G_MAT

def get_Fs(size: int, Temp: float):
    """
    size: size of the lattice
    Temp: temperature
    """
    F_MAT = np.asarray([[F(y2row(i, width=size), y2row(j, width=size),
Temp) for j in range(2 ** size)] for i in
                            range(2 ** size)])
    return F_MAT


def get_Ts(size,temp): #forward pass
    T = np.zeros((size - 1, 2 ** size))
    G_MAT = get_Gs(size, temp)
    F_MAT = get_Fs(size, temp)

    for ti in range(size - 1):
        for yi in range(2 ** size):
            if ti == 0:
                T[ti, yi] = np.sum([G_MAT[i] * F_MAT[i, yi] for i in
range(2 ** size)]) # equation - (18)
            else:
                T[ti, yi] = np.sum(
                    [T[ti - 1, i] * G_MAT[i] * F_MAT[i, yi] for i in
range(2 ** size)]) # equation - (19)

    ZTemp = np.sum(T[-1, :] * G_MAT) # equation (20)
```

```
    return T, ZTemp




def get_Ps(size, temp): #reference solution
    T,ztemp = get_Ts(size, temp)
    G_MAT = get_Gs(size, temp)
    F_MAT = get_Fs(size, temp)
    P = []
    for row in range(size - 1, -1, -1): #row in {0...7}
        if row == size - 1: #(21) first row when starting from the
last
            P.insert(0, [(T[row - 1, yk] * G_MAT[yk]) / ztemp for yk
in range(2 ** size)])
        elif row == 0:      #(23) last row
            P.insert(0, [[(G_MAT[y1] * F_MAT[y1, y2]) / T[row, y2] for
y2 in range(2 ** size)] for y1 in range(2 ** size)])
        else:               #(22) - general case for k in {1...6}
            P.insert(0, [[(T[row - 1, yi] * G_MAT[yi] * F_MAT[yi, yj])
/ T[row, yj] for yj in range(2 ** size)] for yi in
                          range(2 ** size)])
    return P

def sample_ys(size, P): #backward sampling the whole lattice
    Y = np.zeros(size)
    for row in range(size - 1, -1, -1):
        if row == size - 1:
            Y[row] = int(np.random.choice(np.arange(2 ** size),
p=P[row]))
        else:
            Y[row] = np.random.choice(range(2 ** size),
p=np.asarray(P[row])[:, int(Y[row + 1])])
    return Y


#debbug the calculation using forward pass
Ts, ZTemp = get_Ts(2, 1)
print("T1= :", Ts[0, :])
print("T2= Z:", ZTemp)

Ts, ZTemp = get_Ts(3, 1)
print("T1= :", Ts[0, :])
print("T2= :", Ts[1, :])
print("T3= Z:", ZTemp)
```

```
#brute force results of ztemp

print(compute_ZTemp1(1))
print(compute_ZTemp2(1))


T1= : [21.18917525  8.20463255  8.20463255 21.18917525]
T2= Z: 121.23293134406596
T1= : [155.37102759  46.44297052  31.70116107  46.44297052
46.44297052
  31.70116107  46.44297052 155.37102759]
T2= : [23416.16435187  4634.76802124  3916.10003703  4634.76802124
  4634.76802124  3916.10003703  4634.76802124 23416.16435187]
T3= Z: 365645.7491357699
121.23293134406595
365645.74913577037

y = 7
print(y2row(y, width=3))

[1 1 1]

from matplotlib import pyplot as plt

temps = [1, 1.5, 2]
size = 8
images_num = 10

for row,temp in enumerate(temps):
    P = get_Ps(size, temp)
    for num in range(images_num):
        Y = sample_ys(size, P).astype(int)

        image = []
        # image is 1D array of size =size, need to convert back to 2D
image using y2row
        for i in range(size):
            x = y2row(Y[i], width=size)
            image.append(x)
        image = np.asarray(image)
        # print(image)

        #plot all the images in one figure, all the same temps in one
row, set border to image to be on, -1 as white and 1 as black
        plt.subplot(len(temps), images_num, row * images_num + num +
1)
        plt.imshow(image, cmap='gray', vmin=-1,
vmax=1,interpolation='None')
        plt.axis('off')
        plt.title('Temp = {}'.format(temp), fontsize=8)
```

```
plt.tight_layout()
plt.show()
```



Computer Exercise 8 Using the three samplers you implemented above, at each of the three temperatures, draw 10,000 samples, x(1), . . . , x(10000) (each such sample is an 8 × 8 binary image) and compute two empirical expectations: EbTemp(X(1,1)X(2,2)) , 1 10000 10000 X n=1 x(1,1)(n)x(2,2)(n) (13) EbTemp(X(1,1)X(8,8)) , 1 10000 10000 X n=1 x(1,1)(n)x(8,8)(n) (14) where Temp = 1, 1.5, and 2 and where x(i,j)(n) is the value at the (i, j)-th lattice site of sample n. To help you debug here are the values you should get for Temp = 1: EbTemp(X(1,1)X(2,2)) ≈ 0.95; EbTemp(X(1,1)X(8,8)) ≈ 0.9.

```
# computer exercise 8
#need to run with 10000

from tqdm import tqdm

num_samples = 1000
size = 8
temps = [1, 1.5, 2]

expectation_dict = {}

for row,temp in enumerate(temps):
    data = []
    P = get_Ps(size, temp)
    for num in tqdm(range(num_samples)):
```

```
        Y = sample_ys(size, P).astype(int)
        image = []
        # image is 1D array of size =size, need to convert back to 2D image using y2row
        for i in range(size):
            x = y2row(Y[i], width=size)
            image.append(x)
        image = np.asarray(image).astype(np.int8)
        # print(image)
        data.append(image)

    #calculate the expectation
    E12 = np.sum([data[i][0,0]*data[i][1,1] for i in range(num_samples)]) / num_samples
    E18 = np.sum([data[i][0,0]*data[i][7,7] for i in range(num_samples)]) / num_samples
    print("E12 = ", E12)
    print("E18 = ", E18)

    expectation_dict[temp] = [E12, E18]


print(expectation_dict)
```

```
100%|██████████| 1000/1000 [01:54<00:00,  8.77it/s]

E12 =  0.958
E18 =  0.904

100%|██████████| 1000/1000 [01:38<00:00, 10.19it/s]

E12 =  0.77
E18 =  0.542

100%|██████████| 1000/1000 [01:44<00:00,  9.55it/s]

E12 =  0.494
E18 =  0.058
{1: [0.958, 0.904], 1.5: [0.77, 0.542], 2: [0.494, 0.058]}
```

```
#imports....
import numpy as np
import matplotlib.pyplot as plt
```

Build a Gibbs sampler for the 8 × 8 Ising model at temperatures Temp ∈ {1, 1.5, 2}. Compute the empirical expectations of ETemp(X(1,1)X(2,2)) and ETemp(X(1,1)X(8,8)) (45) using two different methods; see below. Compare the estimates to the nearlyexact values computed in $ 1; see Table 1. Programming Note: A convenient way to handle boundaries is to embed the n×n lattice in the interior of an (n+ 2)×(n+ 2) lattice whose boundary values are set to zero, and then visit only the n × n interior sites of the larger lattice.

```
# Computer Exercise 9

def rand_init(size = 8, p = 0.5):
    #embed the n x n lattice in a (n+2) x (n+2) lattice with 0s on the
boundary
    lattice = np.random.randint(low =0, high = 2, size = (size+2,
size+2)) * 2 - 1
    lattice[0,:] = 0
    lattice[:,0] = 0
    lattice[size+1,:] = 0
    lattice[:,size+1] = 0
    return lattice

# init Gibbs sampler at random configuration
def sweep(lattice, temp,size = 8):
    for i in range(1,size+1):
        for j in range(1,size+1):
            lattice[i,j] = pixel_sample(lattice,i,j,temp)
    return lattice

# Expression (40): from Ising prior, sampling from this cond. dist.
p(Xs|sX) is affected by the states of the neighbors
def prior(lattice,curr,i,j,temp):
    return np.exp(curr*(1/temp)*(lattice[i-1,j]+lattice[i+1,j]
+lattice[i,j-1]+lattice[i,j+1]))

def pixel_sample(lattice,i,j,temp):
    p =
prior(lattice,lattice[i,j],i,j,temp)/(prior(lattice,1,i,j,temp)
+prior(lattice,-1,i,j,temp))
    # draw xs according to p
    if lattice[i,j] > 0:
        return np.random.choice([1,-1],p=[p,1-p])
    else:
        return np.random.choice([-1,1],p=[p,1-p])

# Gibbs sampler
def lattice_sampler(temp,size = 8, num_sweeps = 25):
    lattice = rand_init(size)
```

```
    for k in range(num_sweeps):
        sweep(lattice,temp,size)
    return lattice

#method 1

def method1(size = 8, num_samples = 10000):
    from tqdm import tqdm

    expectation_dict  = {}
    temps = [1,1.5,2]
    for temp in temps:
        data = []
        for i in tqdm(range(num_samples)):
            lattice = lattice_sampler(temp,size)
            data.append(lattice)

        # calculate the expectation - Expression (47)
        E12 = np.sum([data[i][1,1]* data[i][2,2] for i in range(num_samples)]) / num_samples
        E18 = np.sum([data[i][1,1]* data[i][8,8] for i in range(num_samples)]) / num_samples
        print("Temp = ", temp)
        print("E12 = ", E12)
        print("E18 = ", E18)
        expectation_dict[temp] = [E12,E18]
    return expectation_dict

method1()
```

```
100%|██████████| 10000/10000 [11:20<00:00, 14.69it/s]

Temp =  1
E12 =  0.9366
E18 =  0.5384

100%|██████████| 10000/10000 [11:39<00:00, 14.29it/s]

Temp =  1.5
E12 =  0.7506
E18 =  0.3562

100%|██████████| 10000/10000 [10:56<00:00, 15.23it/s]

Temp =  2
E12 =  0.5164
E18 =  0.1086



{1: [0.9366, 0.5384], 1.5: [0.7506, 0.3562], 2: [0.5164, 0.1086]}
```

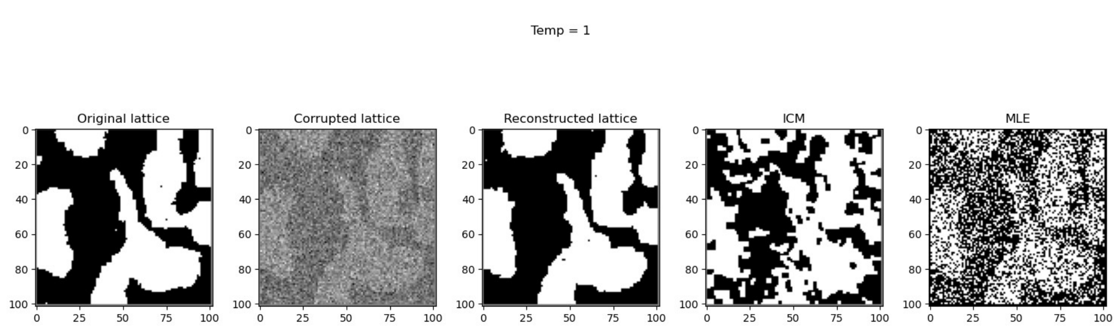
```
def method2(size = 8, num_sweeps = 25000):
    from tqdm import tqdm

    temps = [1,1.5,2]
    expectation_dict  = {}
    for temp in temps:
        # initialize the lattice
        lattice = rand_init(size)
        E12 = 0
        E18 = 0
        counter = 0
        for k in tqdm(range(num_sweeps)):
            lattice = sweep(lattice,temp,size)
            # following burn-in period, we start collecting data
            if k > 100:
                counter += 1
                E12 += lattice[1,1]* lattice[2,2]
                E18 += lattice[1,1]* lattice[8,8]
        # final estimate of the expectation - Expression (47)
        expectation_dict[temp] = [E12/counter,E18/counter]

    return expectation_dict


method2()
```

```
100%|██████████| 25000/25000 [02:08<00:00, 194.51it/s]
100%|██████████| 25000/25000 [01:46<00:00, 233.95it/s]
100%|██████████| 25000/25000 [01:22<00:00, 303.62it/s]

{1: [0.9514036708301539, 0.9028073416603076],
 1.5: [0.7584641953492108, 0.5443190489577895],
 2: [0.49773083256355677, 0.10807662958351741]}
```

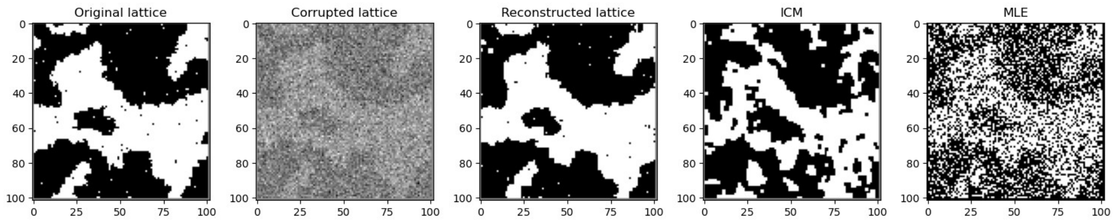
```
#Exercise 10

def noise_sample(size = 100):
    #embed the n x n lattice in a (n+2) x (n+2) lattice with 0s on the
boundary
    #sample the noise from a uniform distribution N(0,4)
    lattice = 2*np.random.standard_normal(size = (size+2, size+2))
    lattice[0,:] = 0
    lattice[:,0] = 0
    lattice[size+1,:] = 0
    lattice[:,size+1] = 0
    return lattice

def posterior(lattice,i,j,y,xs,temp,sig):
    return np.exp(xs*(1/temp)*(lattice[i-1,j]+lattice[i+1,j]
```

```
+lattice[i,j-1]+lattice[i,j+1]) - (1/(2*sig**2))*(y-xs)**2)

def corrupted_pixel_sample(lattice,i,j,temp,corrupted_lattice):
    sig = 2
    y = corrupted_lattice[i,j]
    p = 
posterior(lattice,i,j,y,lattice[i,j],temp,sig)/(posterior(lattice,i,j,
y,1,temp,sig)+posterior(lattice,i,j,y,-1,temp,sig))
    # draw xs according to p
    if lattice[i,j] > 0:
        return np.random.choice([1,-1],p=[p,1-p])
    else:
        return np.random.choice([-1,1],p=[p,1-p])


def ICM_pixel_sample(lattice,i,j,temp,corrupted_lattice):
    sig = 2
    y = corrupted_lattice[i,j]
    p = 
posterior(lattice,i,j,y,lattice[i,j],temp,sig)/(posterior(lattice,i,j,
y,1,temp,sig)+posterior(lattice,i,j,y,-1,temp,sig))
    # draw xs according to p
    if lattice[i,j] > 0:
        return 1 if p > 0.5 else -1
    else:
        return -1 if p > 0.5 else 1


def ICM_lattice_sampler(corrupted_lattice,temp,size = 100, num_sweeps
= 50):
    lattice = rand_init(size)
    for k in range(num_sweeps):
        for i in range(1,size+1):
            for j in range(1,size+1):
                lattice[i,j] = 
ICM_pixel_sample(lattice,i,j,temp,corrupted_lattice)
    return lattice


def corrupted_lattice_sampler(corrupted_lattice,temp,size = 100,
num_sweeps = 50):
    lattice = rand_init(size)
    for k in range(num_sweeps):
        for i in range(1,size+1):
            for j in range(1,size+1):
                lattice[i,j] = 
corrupted_pixel_sample(lattice,i,j,temp,corrupted_lattice)
    return lattice
```

```python
#10.4

temps = [1,1.5,2]
for temp in temps:

    lattice = lattice_sampler(temp = temp,size = 100, num_sweeps = 50)

    noise = noise_sample(size = 100)

    corrupted_lattice =  lattice + noise
    sampled_ICM = ICM_lattice_sampler(corrupted_lattice,temp,size =
100, num_sweeps = 50)
    sampled_lattice_from_corrupted =
corrupted_lattice_sampler(corrupted_lattice,temp,size = 100,
num_sweeps = 50)
    MLE = np.where(corrupted_lattice > 0, 1, -1)

    #plot the 5 lattices one next to the other
    fig, axs = plt.subplots(1, 5, figsize=(15, 5))
    axs[0].imshow(lattice, cmap='gray')
    axs[0].set_title('Original lattice')
    axs[1].imshow(corrupted_lattice, cmap='gray')
    axs[1].set_title('Corrupted lattice')
    axs[2].imshow(sampled_lattice_from_corrupted, cmap='gray')
    axs[2].set_title('Reconstructed lattice')
    axs[3].imshow(sampled_ICM, cmap='gray')
    axs[3].set_title('ICM')
    axs[4].imshow(MLE, cmap='gray')
    axs[4].set_title('MLE')
    fig.suptitle('Temp = ' + str(temp))
    plt.tight_layout()
    plt.show()
```



Temp = 1.5



Temp = 2