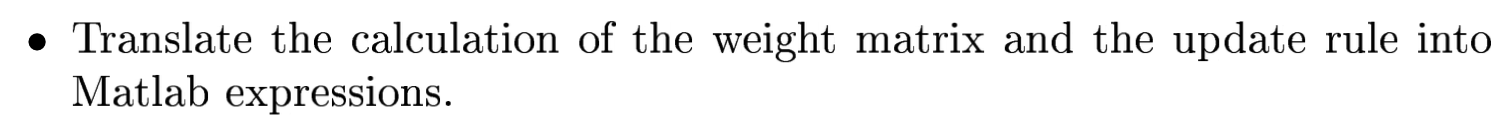
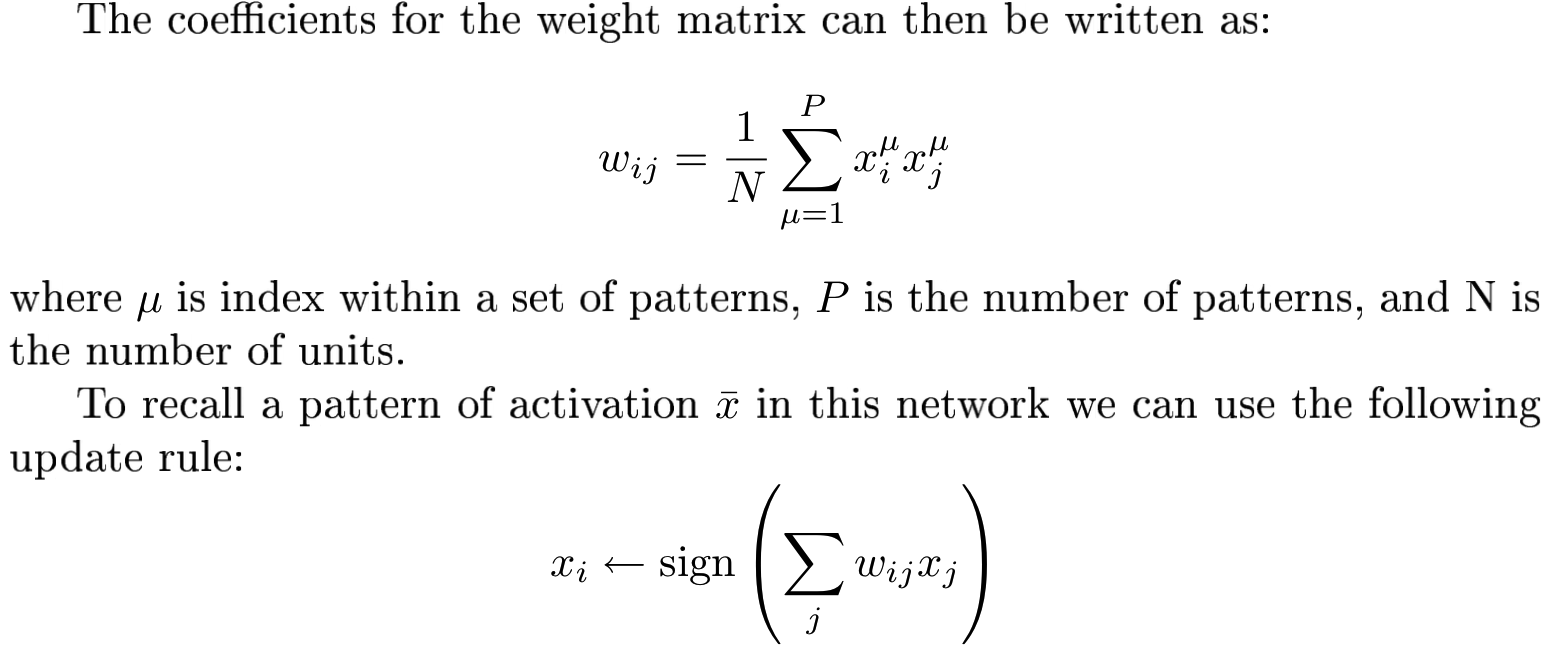
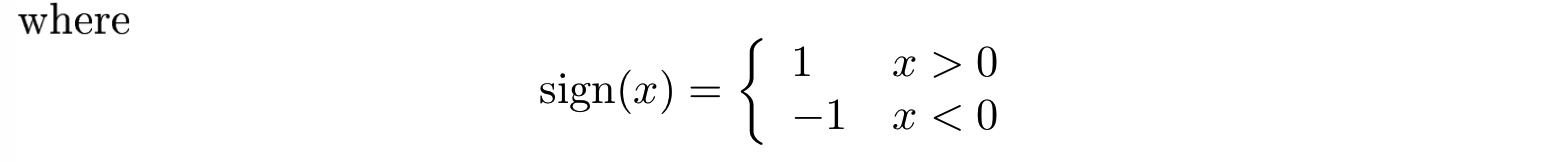
**Lab4 Hopfield Network**



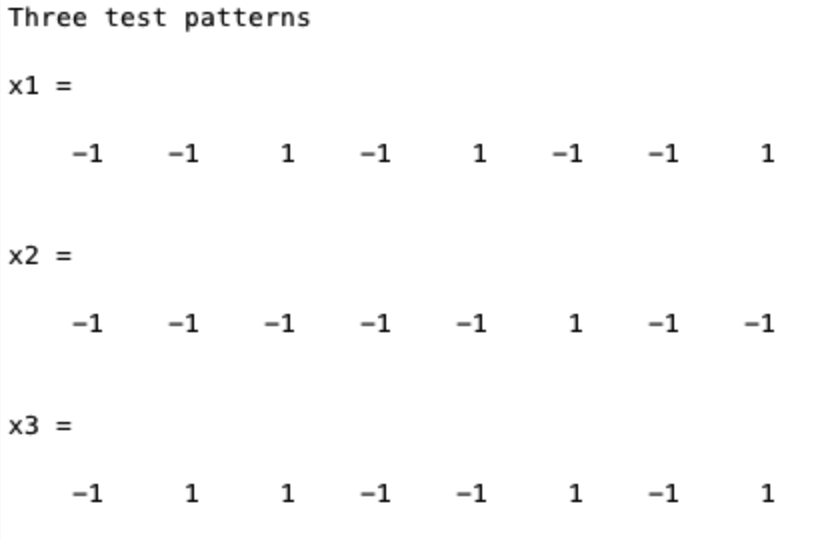
**Question1**



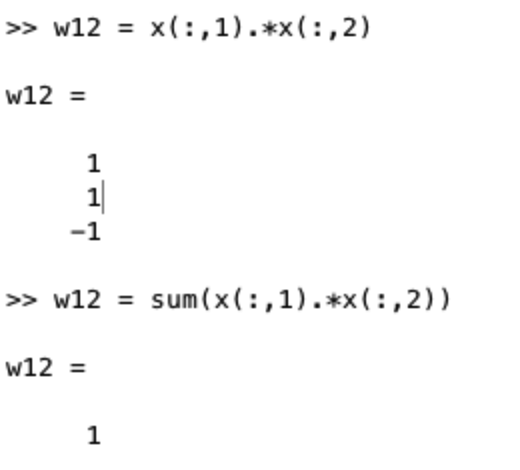




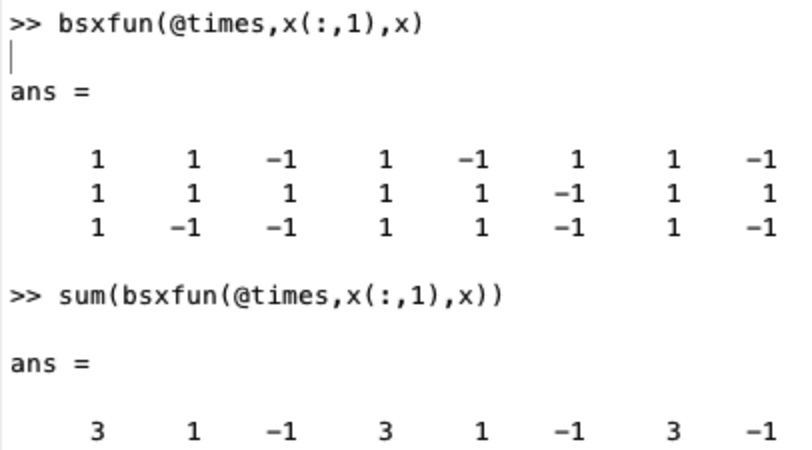
**Translate the calculation of the weight matrix into Matlab expressions:**



Example for calculating one weight (w12):



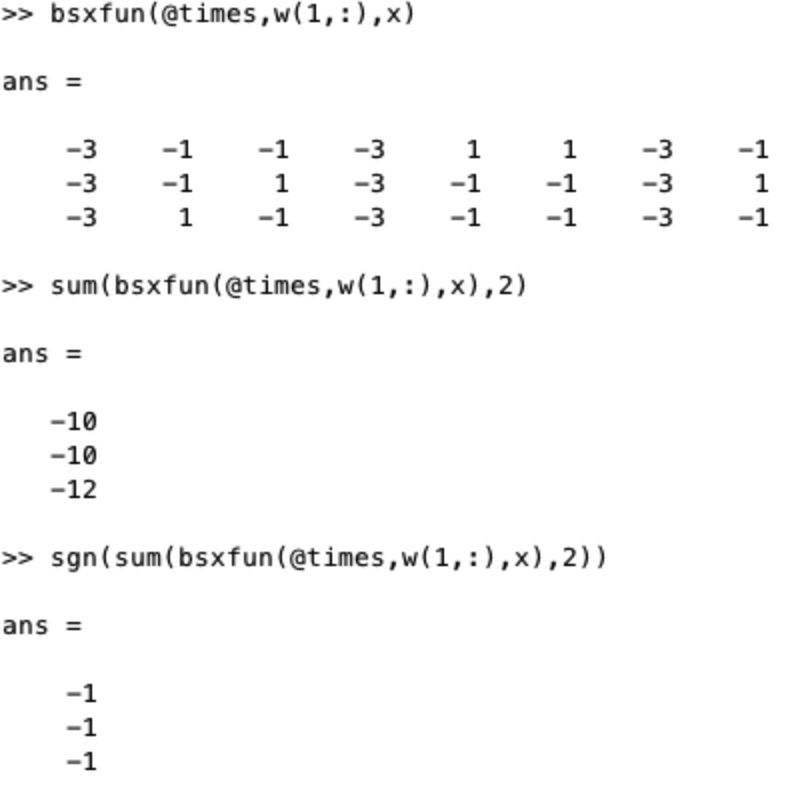
Example for calculating weights between first unit and other units:



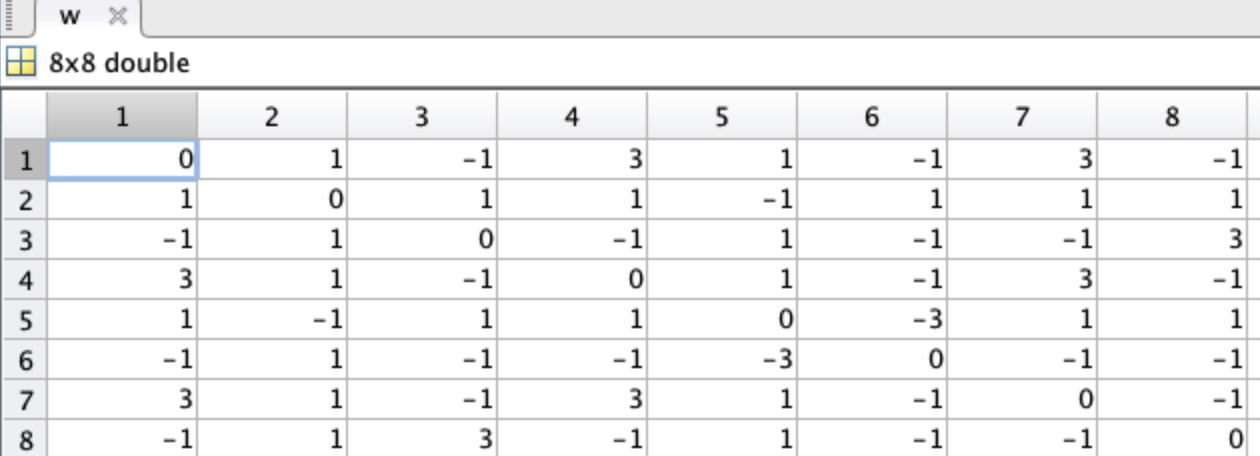
*Note: Weight will be positive if the values of i and j will tend to become equal, but weight will be negative if the bits corresponding to neurons i and j are different.*

**Translate the update rule into Matlab expressions:**

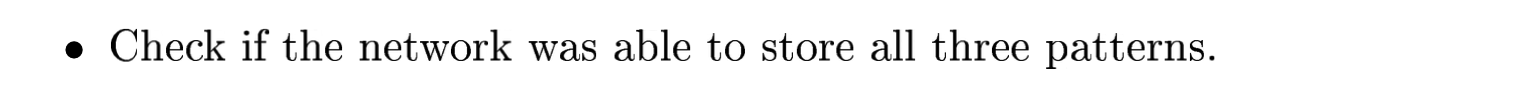
Example for calculating one update unit/node, x1:



A final weight matrix is symmetric. The diagonal elements are zeros.

****

**Question2**

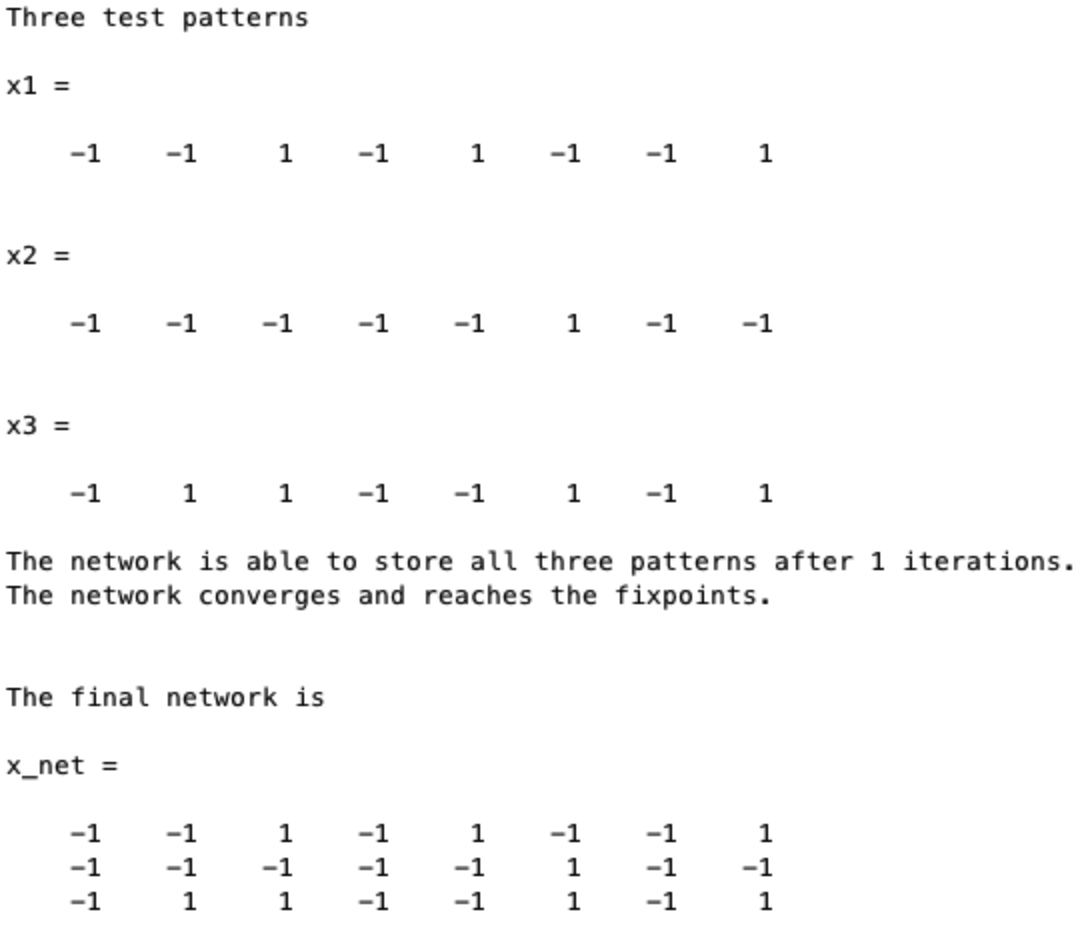


Run hopfield.m & **Result**:

1) The final network is able to store all three patterns, i.e. the final network is the same as the input.

2) The network reach attractor states at 1 itertation. (*Note: The number of iterations needed to reach an attractor scales roughly as log(N) with the network size, which means there are few steps for a network this small.* )

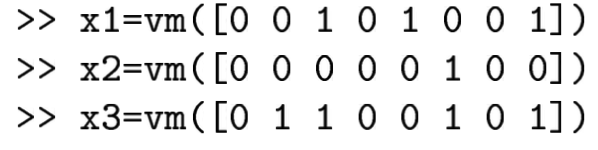
3) When two updates give the same network, which is also the same as the input pattern, the network reaches the fixpoints.



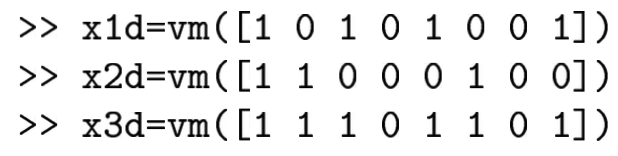


**Can the memory recall the stored patterns from distorted inputs patterns? Define a few new patterns which are distorted versions of the original ones.**

original ones：

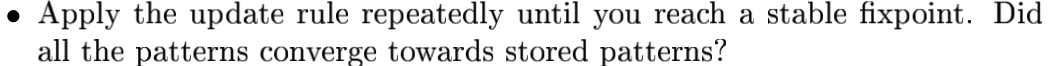


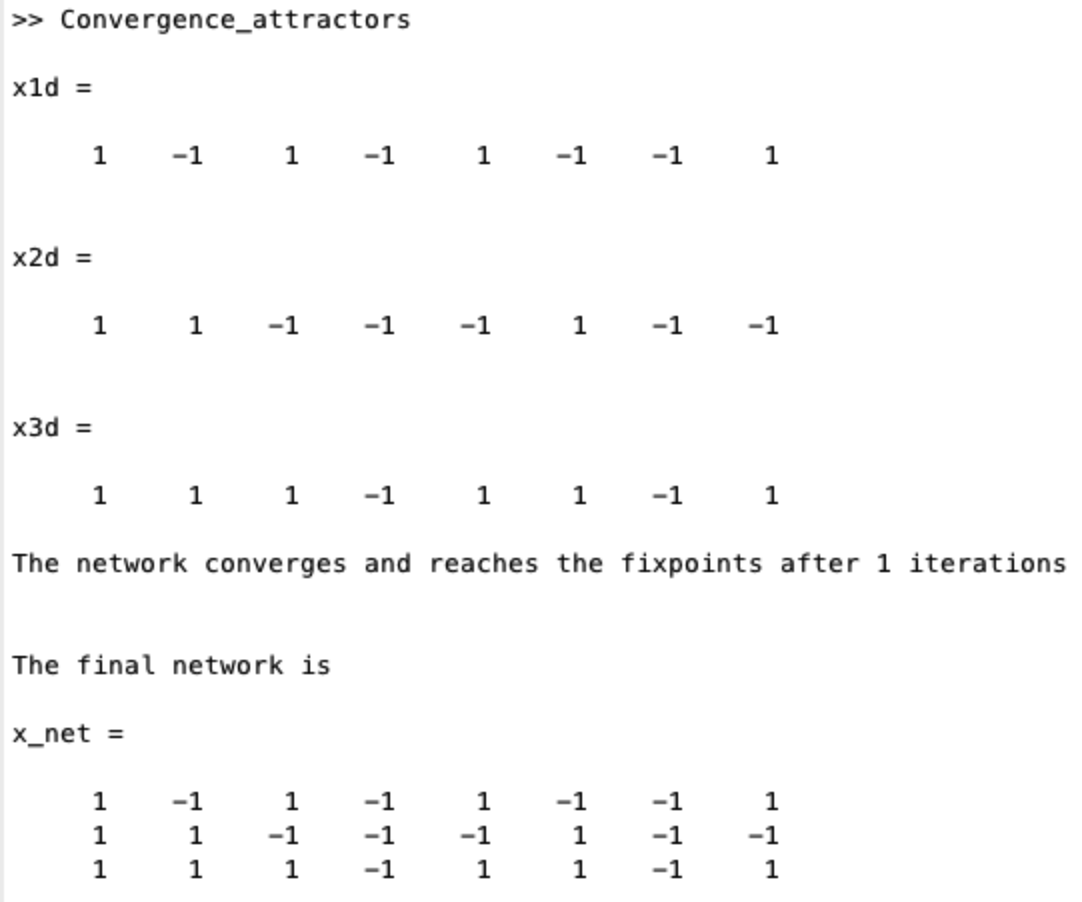
distorted ones：



The first, second and fifth patterns are distorted [1,2,5].

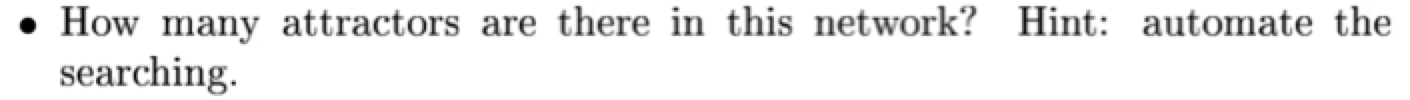
**Question1**

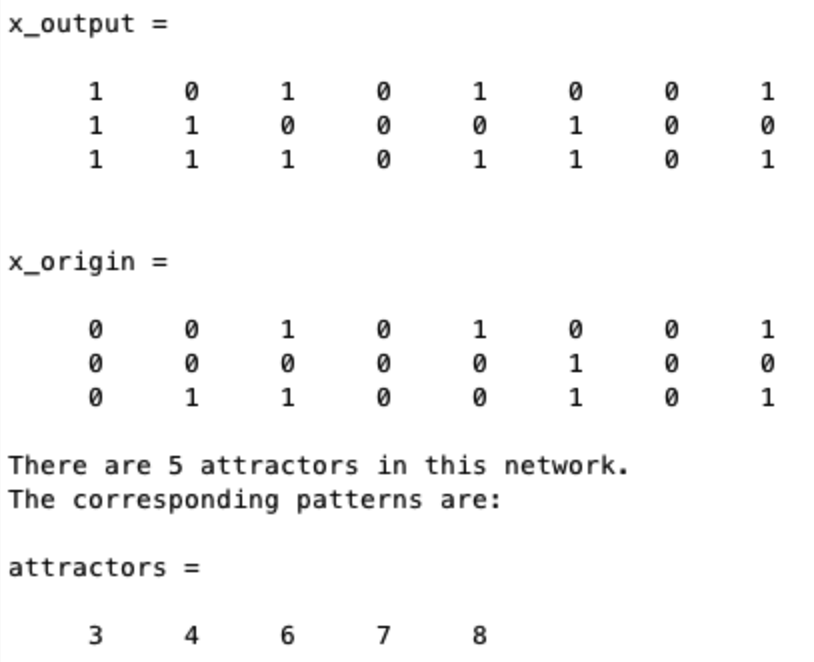




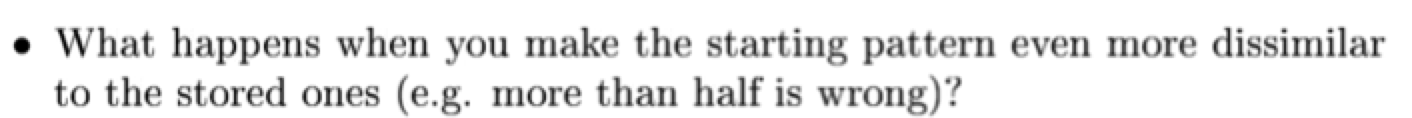
The network stores the distorted versions; but cannot recall all stored patterns from distorted inputs patterns.

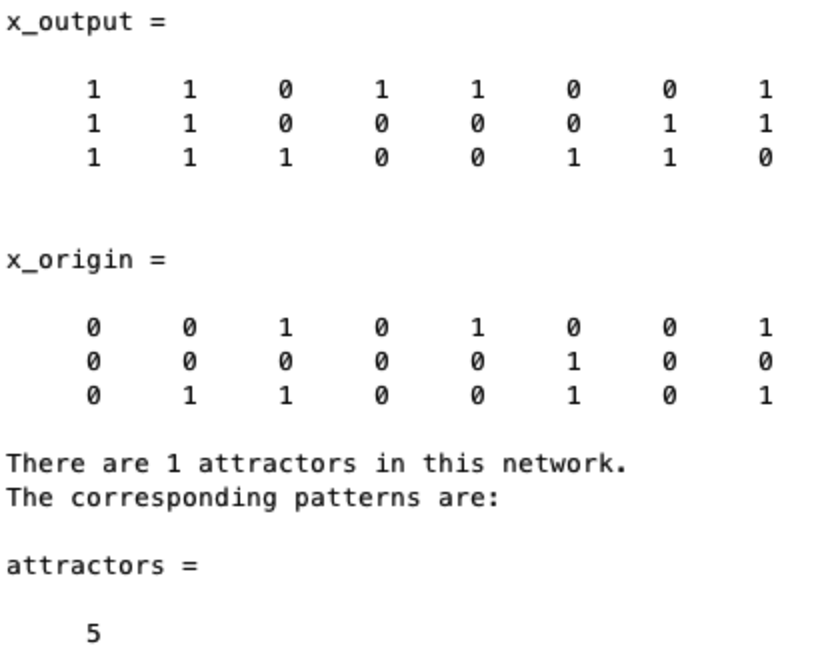
**Question2**





**Question3**

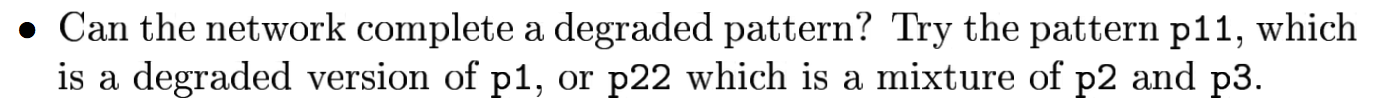


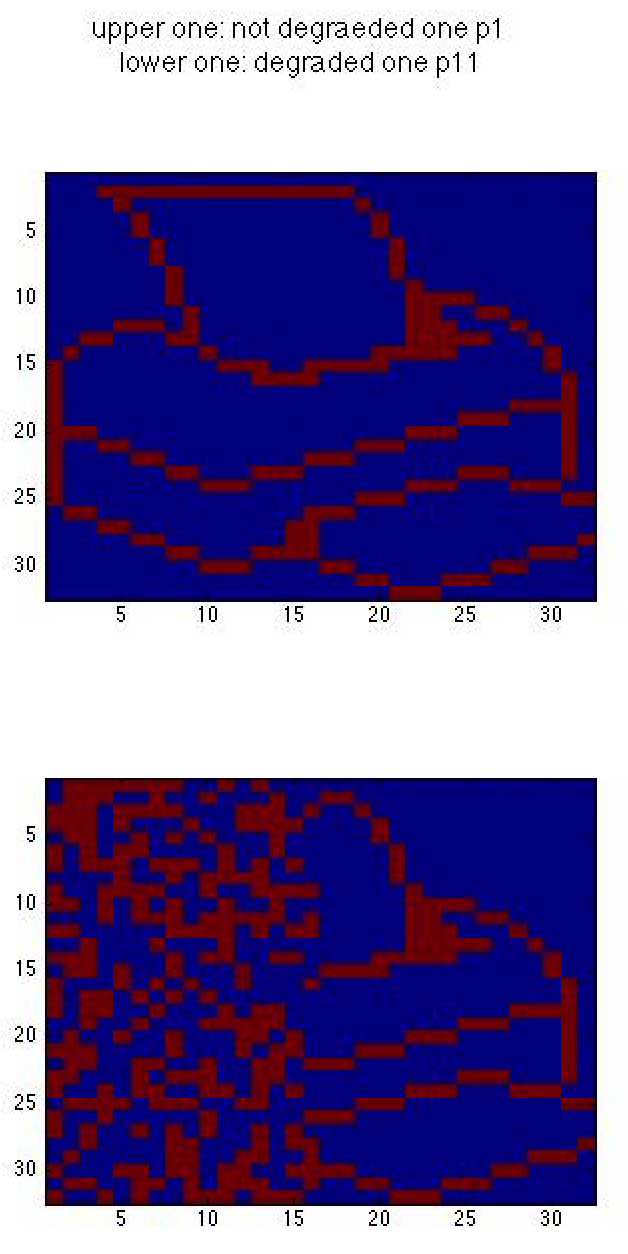


If there are more patterns that are distorted, the network will not be able to recall stored patterns from distorted inputs patterns.



**Question1**

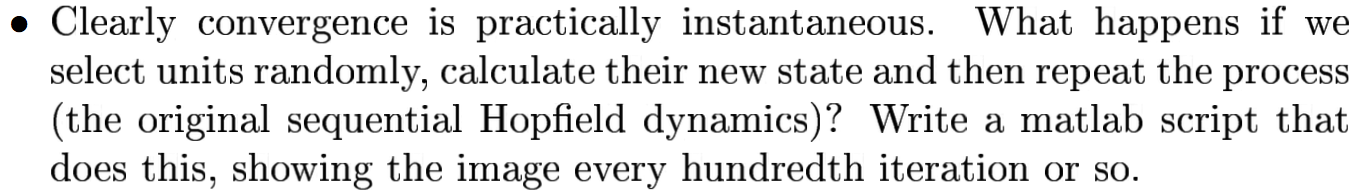




It takes only 1 iteration for convergence; convergence is practically instantaneous.

The network can complete a degraded pattern.

**Question2**



If we select units randomly to calculate their new state with ***patterns chosen to be learned.***

**Example: Try with the degraded version:**

**1000 units**

|  |  |
| --- | --- |
|  |  |
|  |  |

With 1000 units, it converges after less then 100 iterations.

The network can complete a degraded pattern.

**10 units**

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

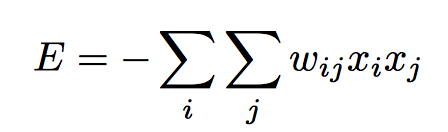
With 10 units, it still does not converge after 1000 iterations.

The network cannot complete a degraded pattern.

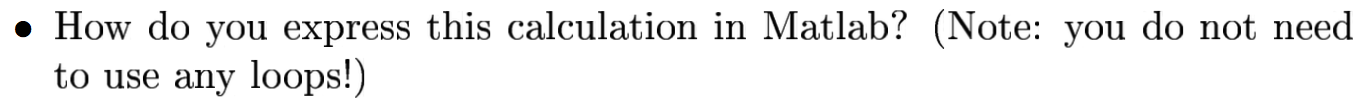
**Conclusion: With more units, the network takes less iteration to converge (converges faster) and have better performance.**



Energy function:



**Question1:**



eg. We do not use loops.

x = [p1;p2;p3];

w = x’\*x; % calculate weight matrix

w = w -diag(diag(w)); % diagonal elements are zeros

E = -sum(sum(w.\*(**x\_net’\*x\_net**),1),2); %x\_net means the current states of the network

**Question2:**



This means energy for different pattern.

Eg. The attractor for p1, energy is

w = x(1,:)’\* x(1,:); % calculate weight matrix

w = w -diag(diag(w)); % diagonal elements are zeros

E = -sum(sum(w.\*(**x\_net’\*x\_net**),1),2); %x\_net means the current states of the network

**Question3:**



This means energy for distorted patterns.

Eg. If the index of distorted patterns is 1, the energy is

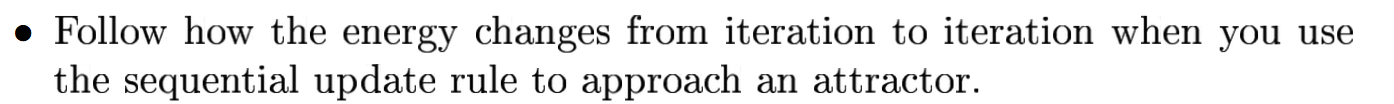
x = [p11;p2;p3];

w = x(1,:)’\* x(1,:); % calculate weight matrix

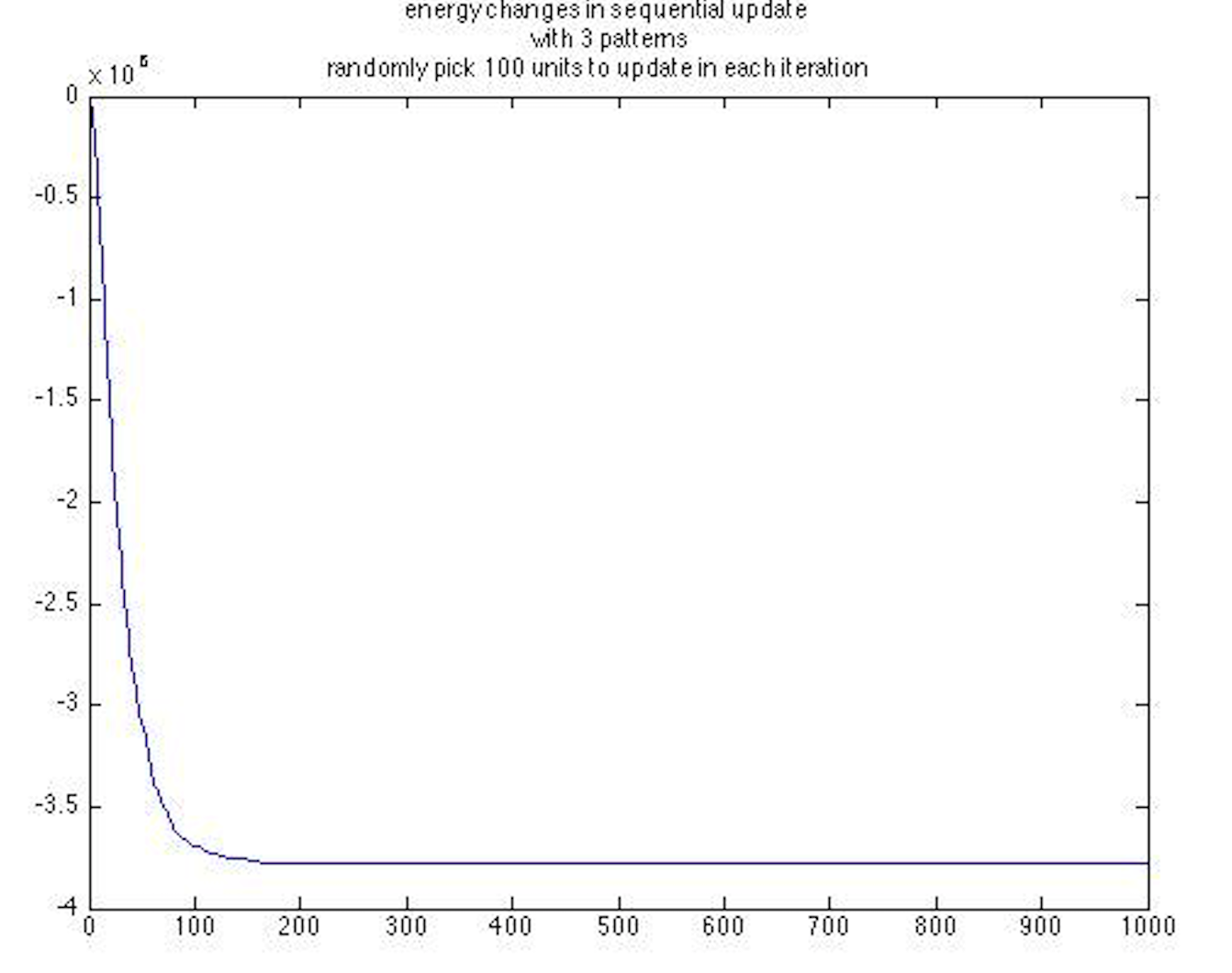
w = w -diag(diag(w)); % diagonal elements are zeros

E = -sum(sum(w.\*(**x\_net’\*x\_net**),1),2); %x\_net means the current states of the network

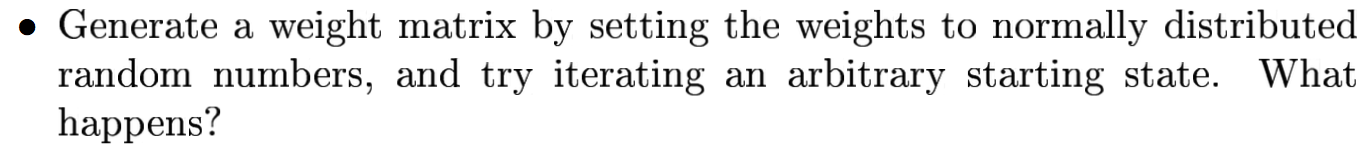
**Question4:**

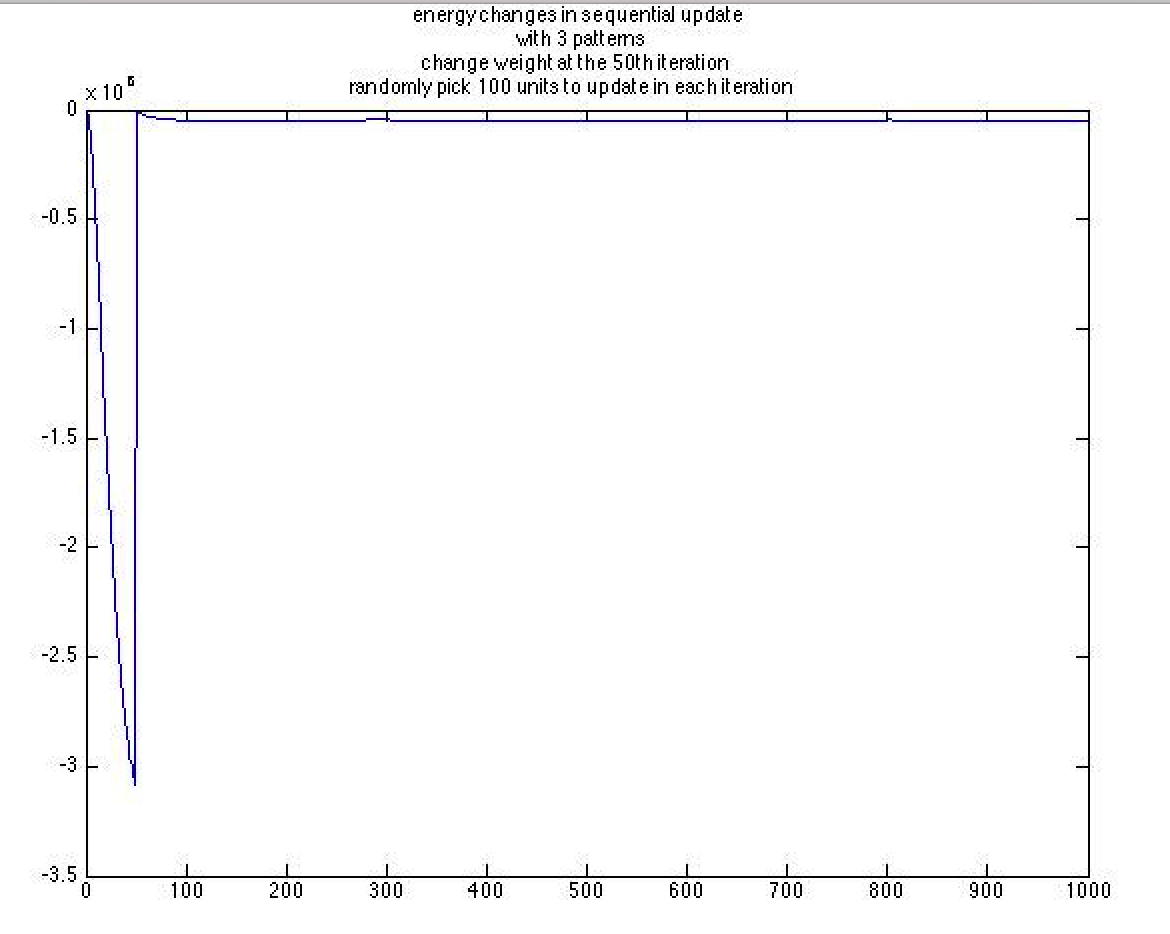


The energy decreases from iteration to iteration. It finally converges to the local minima.



**Question5:**

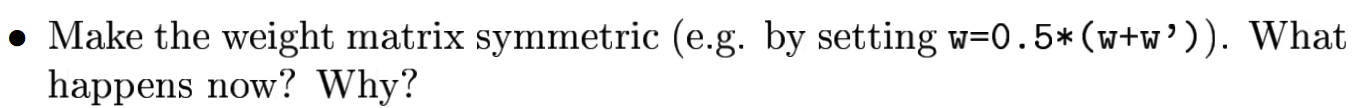


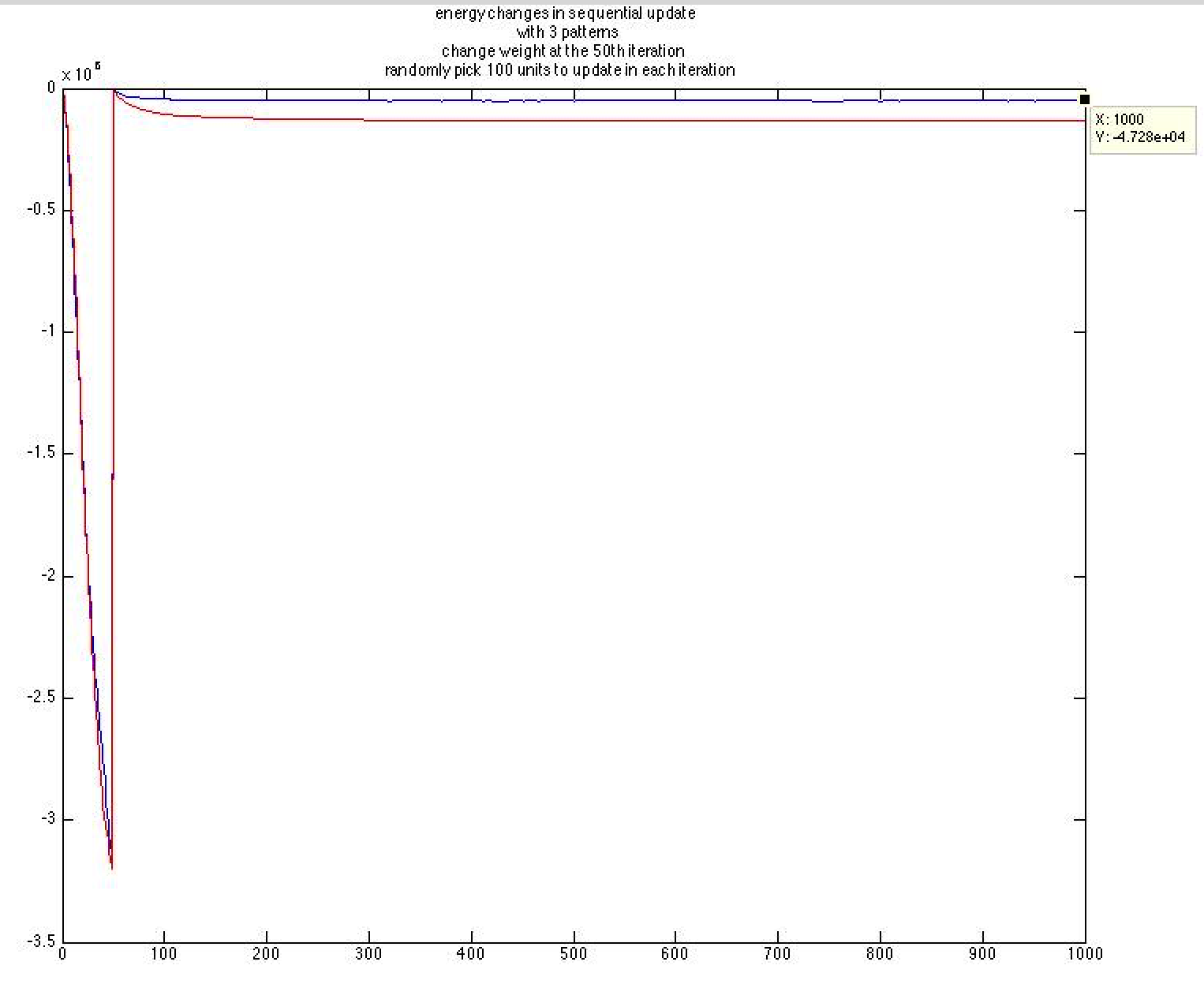


When try iterating an arbitrary starting state(at the 50th iteration), the energy starts to grow from iteration to iteration. It finally converges to the initial level.

The reason is because this weight matrix is totally different from the desired weight matrix. This wrong matrix destroys the desired update and therefore making the energy of the network high.

**Question6:**

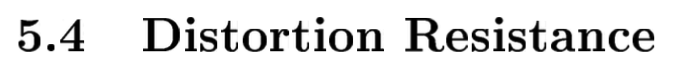


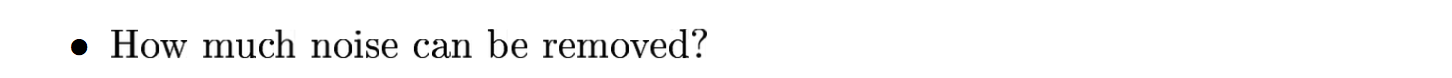


The red line is the one with symmetric weight (normally distributed random numbers).

The result is similar to the one with non-symmetric weight. But it finally converges to a relative lower right level.

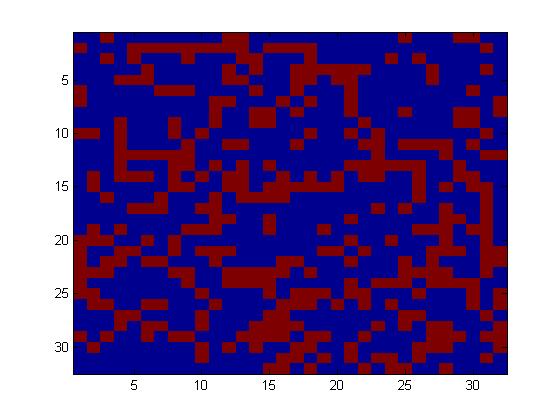
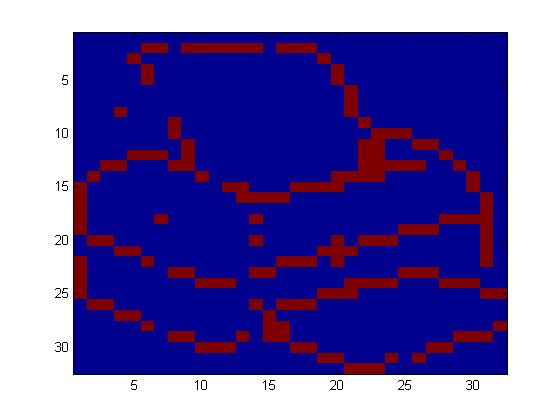
The reason is because the network is internally symmetric. A weight of a mutual connection between two nodes is same: wij=wji.

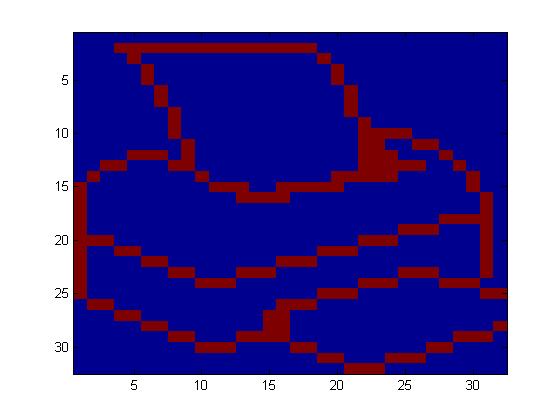




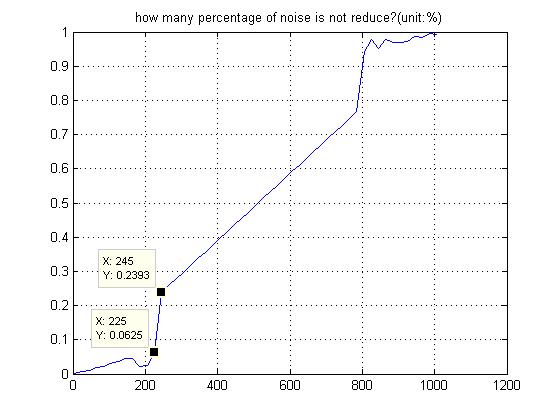
**Answer1:**

Let the script run across 0 to 100% noise and plot the result. Correspondingly, the number of noise can be set from 5 to 1024. When the noise points change from 225(left) to 245(right), the network cannot reconstruct the data anymore obviously. The reference is original p1.



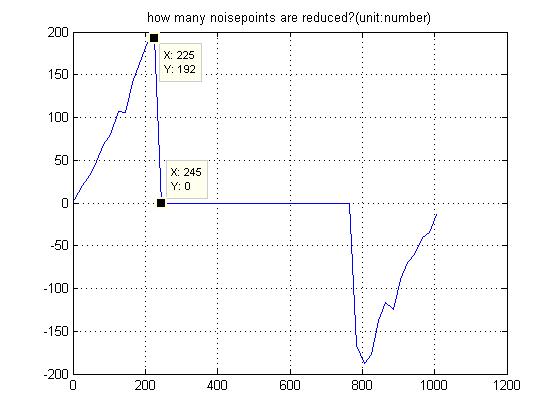
(Original p1)

1) How many percentage of noise can be removed?



So, the probability of noise which can be moved is around 225/1024=21.97%

2) How many noise points can be removed?



When we change the noise points from 225 to 245, the noise points will not be moved by the network.

**Answer2:**

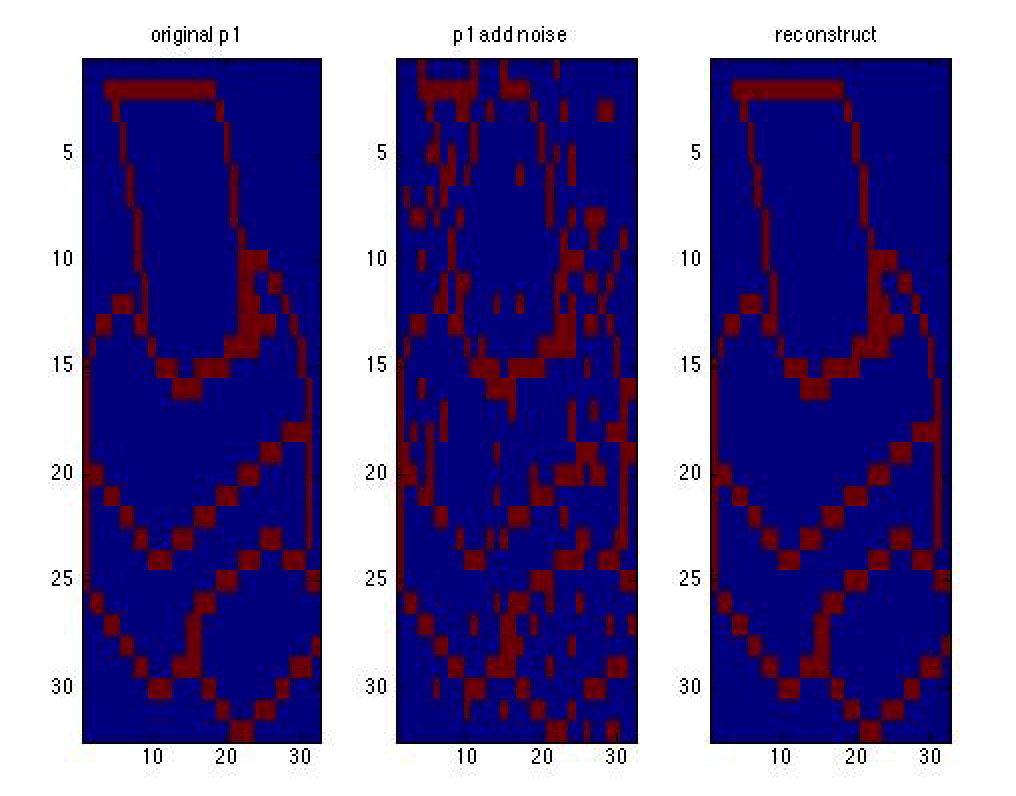
Step1: train a network with p1, p2, p3;

Step2: add noise to a pattern;

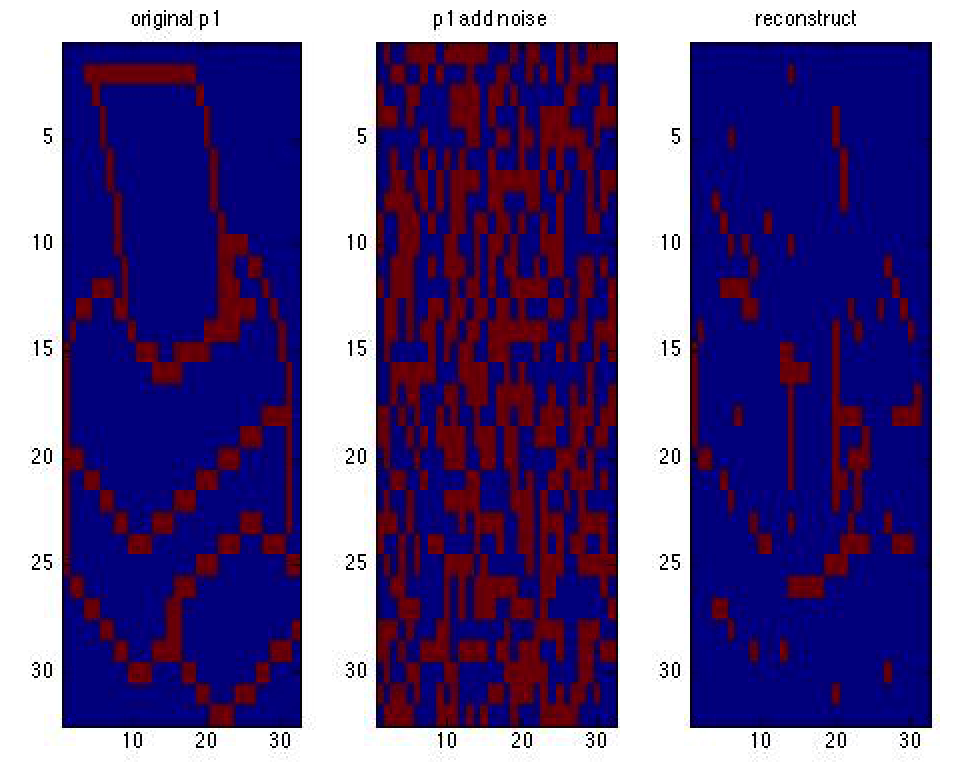
Step3: iterate it a number of times and check whether it has been successfully restored.

**Result:**

1) When adding 100 noise points, the network is able to restore the correct pattern.



2) When adding 500 noise points, the network is not able to restore the correct pattern.



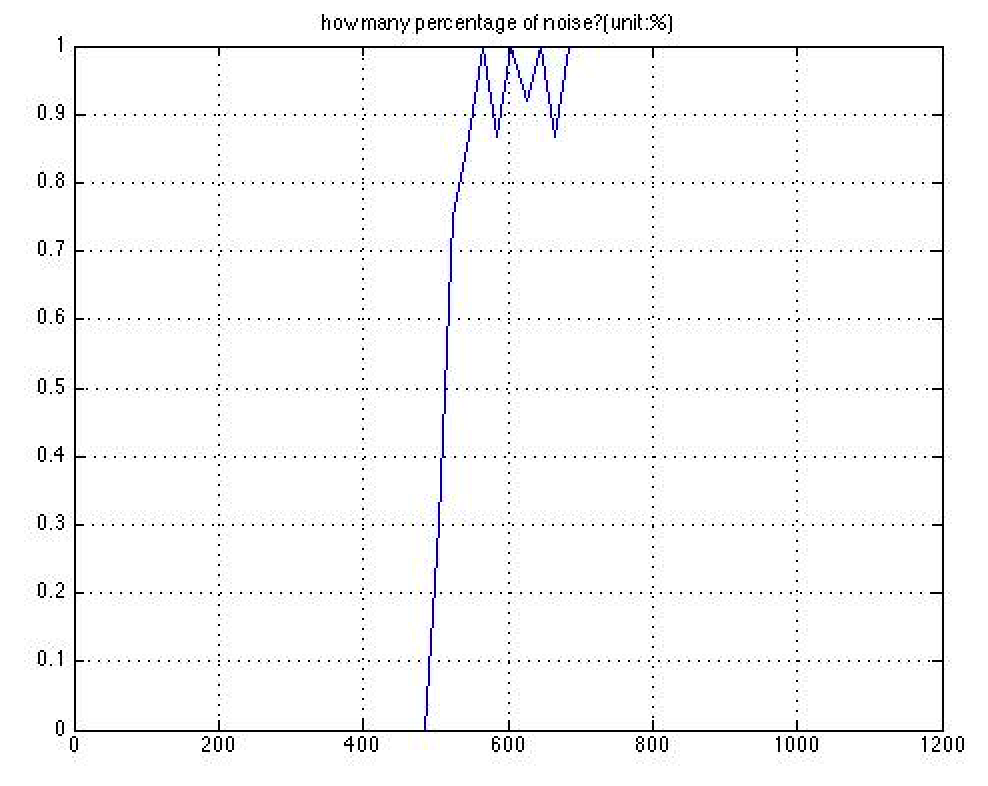
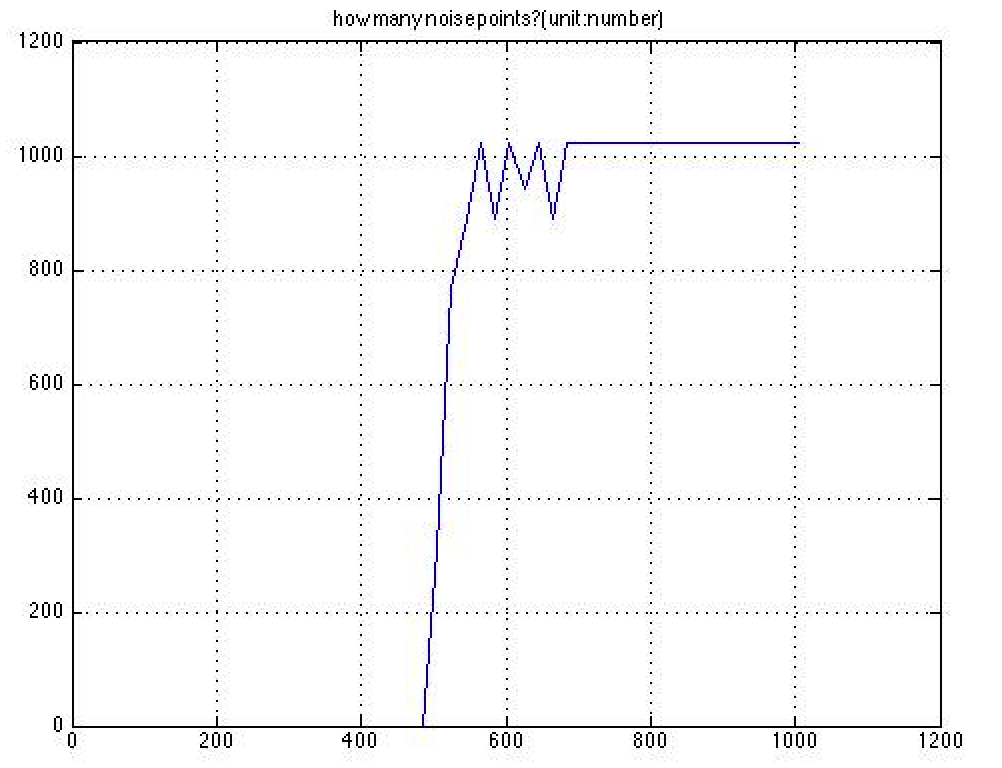
3) Let the script run across 0 to 100% noise and plot the result.

When adding less than around 500 noise points, the network is able to restore the correct pattern. All noise points are compressed.

When adding more than around 500 noise points, the network is not able to restore the correct pattern. There are a lot of noise points.

**Left: how many noise points in the reconstructed noisy pattern?**

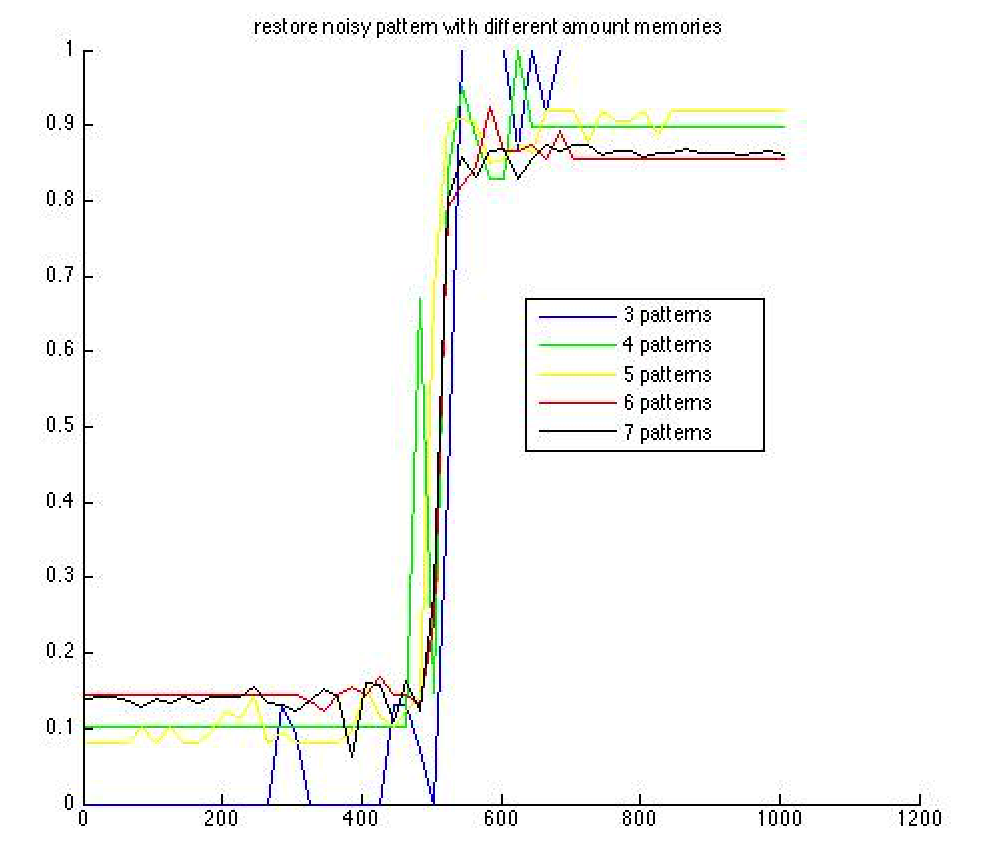
**Right: how many percentage of noise in the reconstructed noisy pattern?**





Start by adding p4 into the weight matrix and check if moderately distorted patters can still be recognized. Then continue by adding others such as p5, p6 and p7 in some order and checking the performance after each addition.

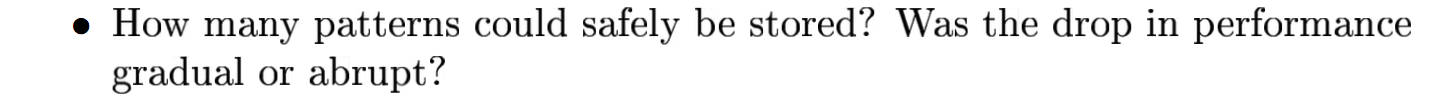
Add p4, p5, p6 and p7 to check the restoring performance:



**Result:**

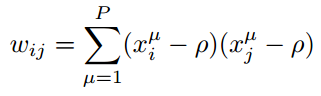
It becomes harder for the network to restore a distorted pattern and compress noise for a pattern.

**Question1**

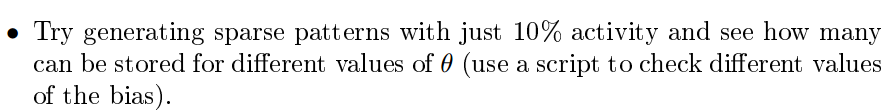




Patterns x has less activity, we can set as 10%. Then generate the weights connected to units.



**Question1**



**Question2**

