BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai – 400058-India
Department of Computer Engineering

Name	Rishabh Santosh Shenoy
UID no.	2023300222
Experiment No.	7

AIM:	To implement Auto Associative and Bidirectional Associative Memory Network
Program	
PROBLEM STATEMENT :	To implement Auto Associative and Bidirectional Associative Memory Network
PROGRAM:	see below

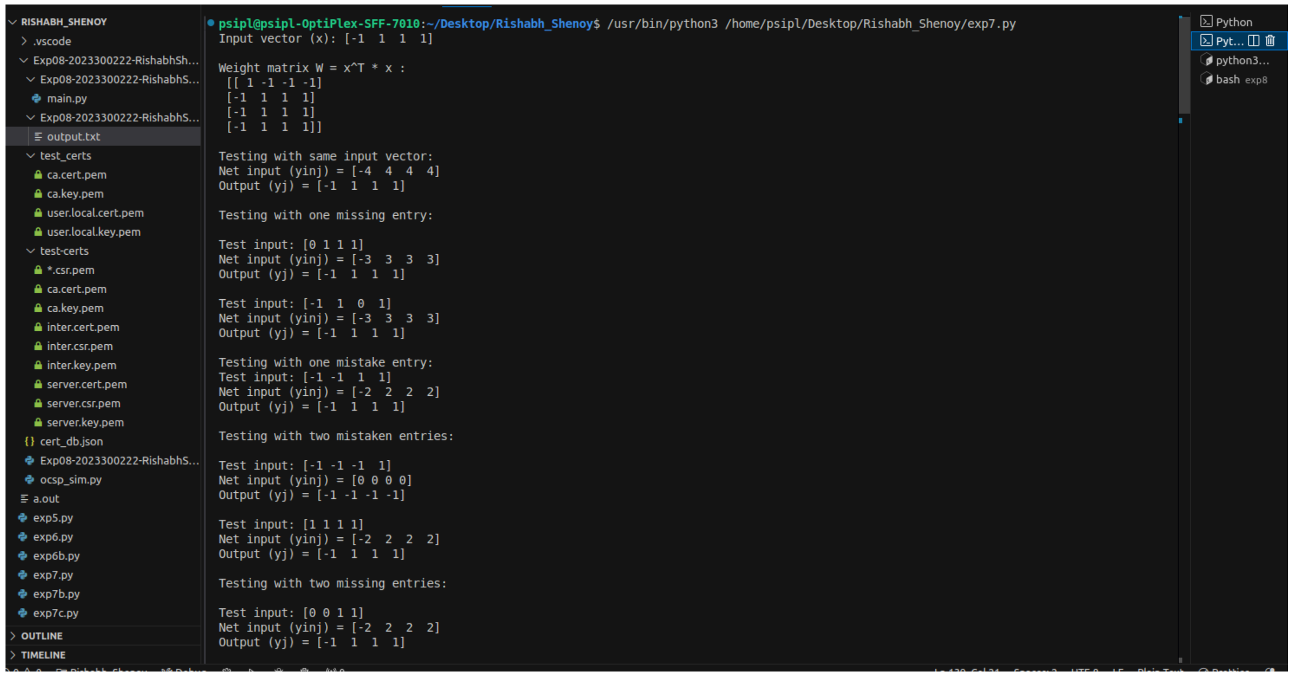
```
import numpy as np

def activation(y_in):
    return np.where(y_in > 0, 1, -1)

x = np.array([-1, 1, 1, 1])
print("Input vector (x):", x)

W = np.outer(x, x)
np.fill_diagonal(W, 1)
print("\nWeight matrix W = x^T * x :\n", W)

print("\nTesting with same input vector:")
test = np.array([-1, 1, 1, 1])
yin = np.dot(test, W)
print("Net input (yinj) =", yin)
y = activation(yin)
print("Output (yj) =", y)

print("\nTesting with one missing entry:")
test_cases = [
    np.array([0, 1, 1, 1]),
    np.array([-1, 1, 0, 1])
]
```

```
for test in test_cases:
    print("\nTest input:", test)
    yin = np.dot(test, W)
    print("Net input (yinj) =", yin)
    y = activation(yin)
    print("Output (yj) =", y)

print("\nTesting with one mistake entry:")
test = np.array([-1, -1, 1, 1])
print("Test input:", test)
yin = np.dot(test, W)
print("Net input (yinj) =", yin)
y = activation(yin)
print("Output (yj) =", y)

print("\nTesting with two mistaken entries:")
test_cases = [
    np.array([-1, -1, -1, 1]),
    np.array([1, 1, 1, 1])
]
for test in test_cases:
    print("\nTest input:", test)
    yin = np.dot(test, W)
    print("Net input (yinj) =", yin)
    y = activation(yin)
    print("Output (yj) =", y)

print("\nTesting with two missing entries:")
test_cases = [
    np.array([0, 0, 1, 1]),
    np.array([-1, 0, 0, 1])
]
for test in test_cases:
    print("\nTest input:", test)
    yin = np.dot(test, W)
    print("Net input (yinj) =", yin)
    y = activation(yin)
    print("Output (yj) =", y)
```

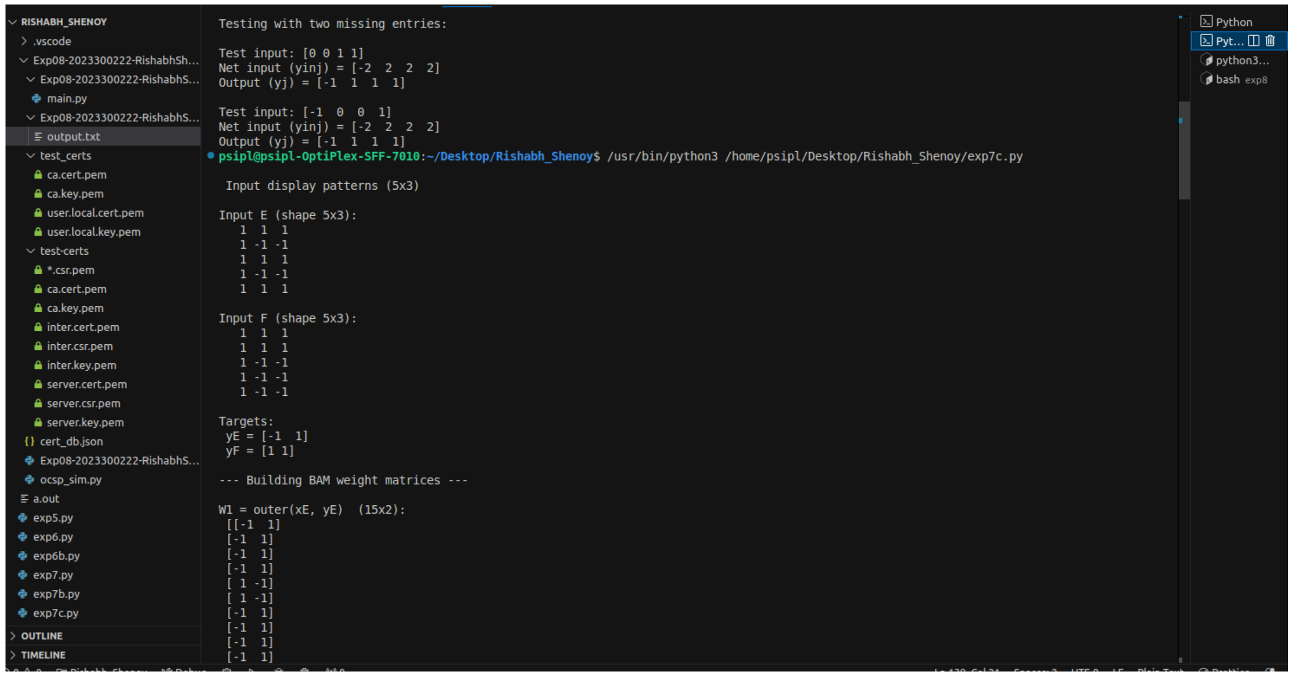
Program 2:

```
import numpy as np

def sign_bipolar(v):
    """Bipolar sign: >0 -> 1, <0 -> -1, ==0 -> 0 (keeps zeros explicit)."""
    return np.where(v > 0, 1, np.where(v < 0, -1, 0))

def print_matrix_as_display(v, rows=5, cols=3, name="pattern"):
    """Pretty print a flattened vector as rows x cols matrix."""
    mat = v.reshape((rows, cols))
    print(f"{name} (shape {rows}x{cols}):")
    for r in mat:
        print(" ", " ".join(f"{int(x):2d}" for x in r))
    print()

xE = np.array([ 1,  1,  1,
                1, -1, -1,
                1,  1,  1,
                1, -1, -1,
                1,  1,  1], dtype=int)

xF = np.array([ 1,  1,  1,
                1,  1,  1,
                1, -1, -1,
                1, -1, -1,
                1, -1, -1], dtype=int)

yE = np.array([-1,  1], dtype=int)
yF = np.array([ 1,  1], dtype=int)

print("\n Input display patterns (5x3) \n")
print_matrix_as_display(xE, 5, 3, name="Input E")
print_matrix_as_display(xF, 5, 3, name="Input F")

print("Targets:")
print(" yE =", yE)
print(" yF =", yF)
print("\nBuilding BAM weight matrices\n")

W1 = np.outer(xE, yE)
W2 = np.outer(xF, yF)
W = W1 + W2

np.set_printoptions(linewidth=120, formatter={'int': '{:2d}'.format})
print("W1 = outer(xE, yE)  (15x2):\n", W1)
print("\nW2 = outer(xF, yF)  (15x2):\n", W2)
print("\nW = W1 + W2  (15x2):\n", W)
print("\nEnd of weight construction\n")

def bam_recall(W, x_init=None, y_init=None, max_iters=10,
verbose=True):
    """
    Run iterative BAM recall. Provide either x_init (input->output) or y_init
(output->input).
    Returns final (x, y) bipolar vectors and steps.
    """
    Nx, Ny = W.shape
    if x_init is None and y_init is None:
        raise ValueError("Provide x_init or y_init to recall.")
    x = None
    y = None
    if x_init is not None:
        x = x_init.copy().astype(int)
        x = sign_bipolar(x)
    if y_init is not None:
        y = y_init.copy().astype(int)
        y = sign_bipolar(y)

    steps = []
    for it in range(max_iters):
        changed = False
        if x is not None:
            y_in = x.dot(W)
            y_new = sign_bipolar(y_in)
            steps.append(("x->y", it+1, x.copy(), y_in.copy(), y_new.copy()))
            if y is None or not np.array_equal(y_new, y):
                changed = True
            y = y_new
        if y is not None:
            x_in = W.dot(y)
            x_new = sign_bipolar(x_in)
            steps.append(("y->x", it+1, y.copy(), x_in.copy(), x_new.copy()))
            if x is None or not np.array_equal(x_new, x):
                changed = True
            x = x_new
        if verbose:
            print(f"Iteration {it+1}:")
            for step in steps[-2:]:
                direction, pass_no, src_vec, net_in, out_vec = step
                if direction == "x->y":
                    print(f"  x->y (pass {pass_no}):")
                    print("    x (flattened 15):", src_vec.tolist())
                    print("    net input y_in     :", net_in.tolist())
                    print("    y (after sign)     :", out_vec.tolist())
                else:
                    print(f"  y->x (pass {pass_no}):")
                    print("    y (2)              :", src_vec.tolist())
                    print("    net input x_in     :", net_in.tolist())
                    print("    x (after sign)     :", out_vec.tolist())
            print()
        if not changed:
            if verbose:
                print(f"Converged after {it+1} iterations.\n")
            break
    return x, y, steps


print("\n Test recall: Present stored input E and read output y \n")
x_test = xE.copy()
x_rec, y_rec, steps = bam_recall(W, x_init=x_test, max_iters=10,
verbose=True)
print("Final reconstructed y:", y_rec)
print_matrix_as_display(x_rec, 5, 3, name="Final reconstructed x (from
y->x)")

print("\n Test recall: Present stored input F and read output y \n")
x_test = xF.copy()
x_rec, y_rec, steps = bam_recall(W, x_init=x_test, max_iters=10,
verbose=True)
print("Final reconstructed y:", y_rec)
print_matrix_as_display(x_rec, 5, 3, name="Final reconstructed x (from
y->x)")

print("\n Test recall: Present target yE and reconstruct x \n")
x_from_y, y_from_y, _ = bam_recall(W, y_init=yE, max_iters=10,
verbose=True)
print("Final reconstructed x from yE:")
print_matrix_as_display(x_from_y, 5, 3, name="x (reconstructed from
yE)")

print("\n Test recall: Present target yF and reconstruct x \n")
x_from_y, y_from_y, _ = bam_recall(W, y_init=yF, max_iters=10,
verbose=True)
print("Final reconstructed x from yF:")
print_matrix_as_display(x_from_y, 5, 3, name="x (reconstructed from
yF)")

print("Test with noisy input (flip a few bits of E) \n")
noisyE = xE.copy()
noisy_idxs = [0, 7, 14]
noisyE[noisy_idxs] *= -1
print("Noisy E (flattened):", noisyE.tolist())
print_matrix_as_display(noisyE, 5, 3, name="Noisy E")
x_rec, y_rec, _ = bam_recall(W, x_init=noisyE, max_iters=10,
verbose=True)
print("Reconstructed y from noisy E:", y_rec)
print_matrix_as_display(x_rec, 5, 3, name="Final reconstructed x from
noisy E")
```

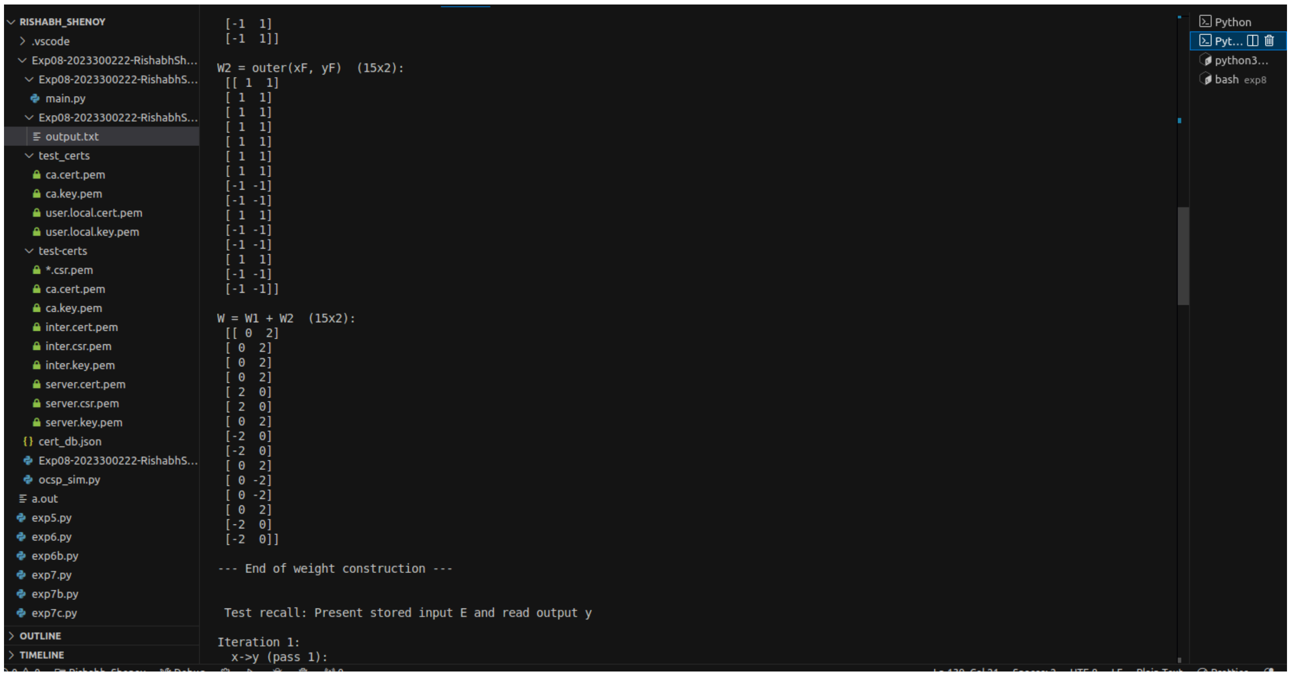
RESULT:



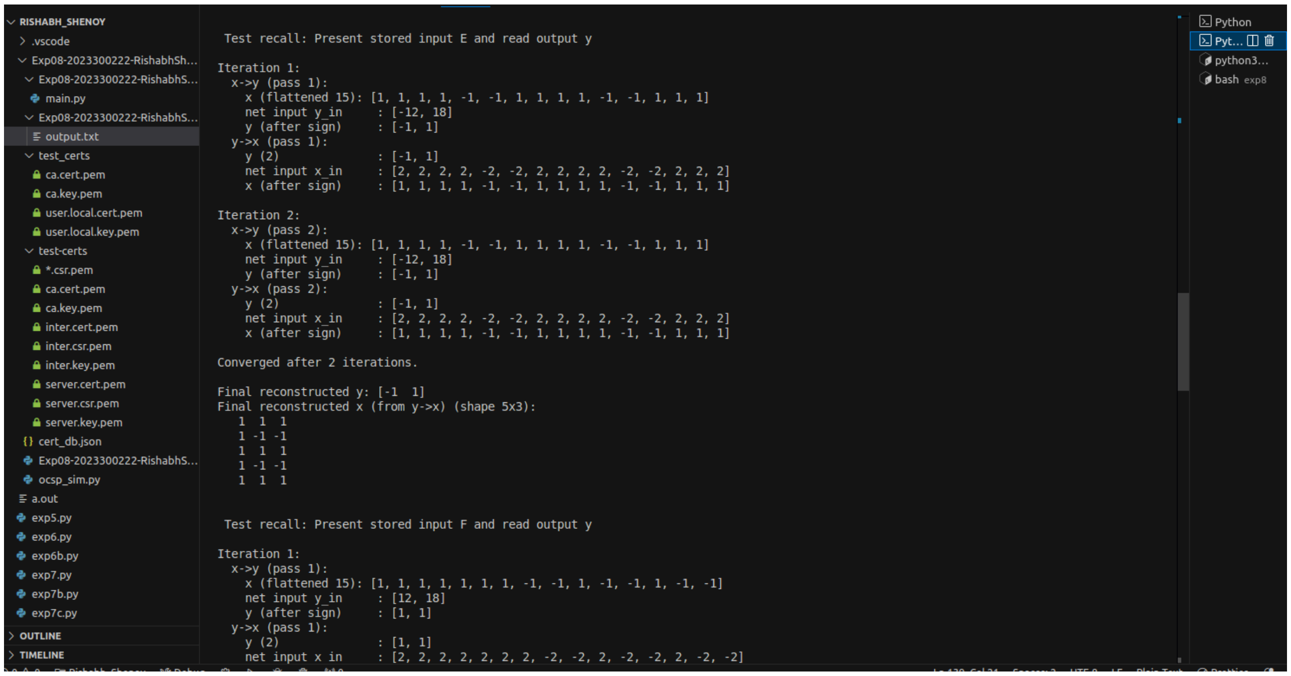
```
psipl@psipl-OptiPlex-SFF-7010:~/Desktop/Rishabh_Shenoy$ /usr/bin/python3 /home/psipl/Desktop/Rishabh_Shenoy/exp7.py
Input vector (x): [-1  1  1  1]

Weight matrix W = x^T * x :
 [[ 1 -1 -1 -1]
 [-1  1  1  1]
 [-1  1  1  1]
 [-1  1  1  1]]

Testing with same input vector:
Net input (yinj) = [-4  4  4  4]
Output (yj) = [-1  1  1  1]

Testing with one missing entry:

Test input: [0 1 1 1]
Net input (yinj) = [-3  3  3  3]
Output (yj) = [-1  1  1  1]

Test input: [-1  1  0  1]
Net input (yinj) = [-3  3  3  3]
Output (yj) = [-1  1  1  1]

Testing with one mistake entry:
Test input: [-1 -1  1  1]
Net input (yinj) = [-2  2  2  2]
Output (yj) = [-1  1  1  1]

Testing with two mistaken entries:

Test input: [-1 -1 -1  1]
Net input (yinj) = [0 0 0 0]
Output (yj) = [-1 -1 -1 -1]

Test input: [1 1 1 1]
Net input (yinj) = [-2  2  2  2]
Output (yj) = [-1  1  1  1]

Testing with two missing entries:

Test input: [0 0 1 1]
Net input (yinj) = [-2  2  2  2]
Output (yj) = [-1  1  1  1]
```



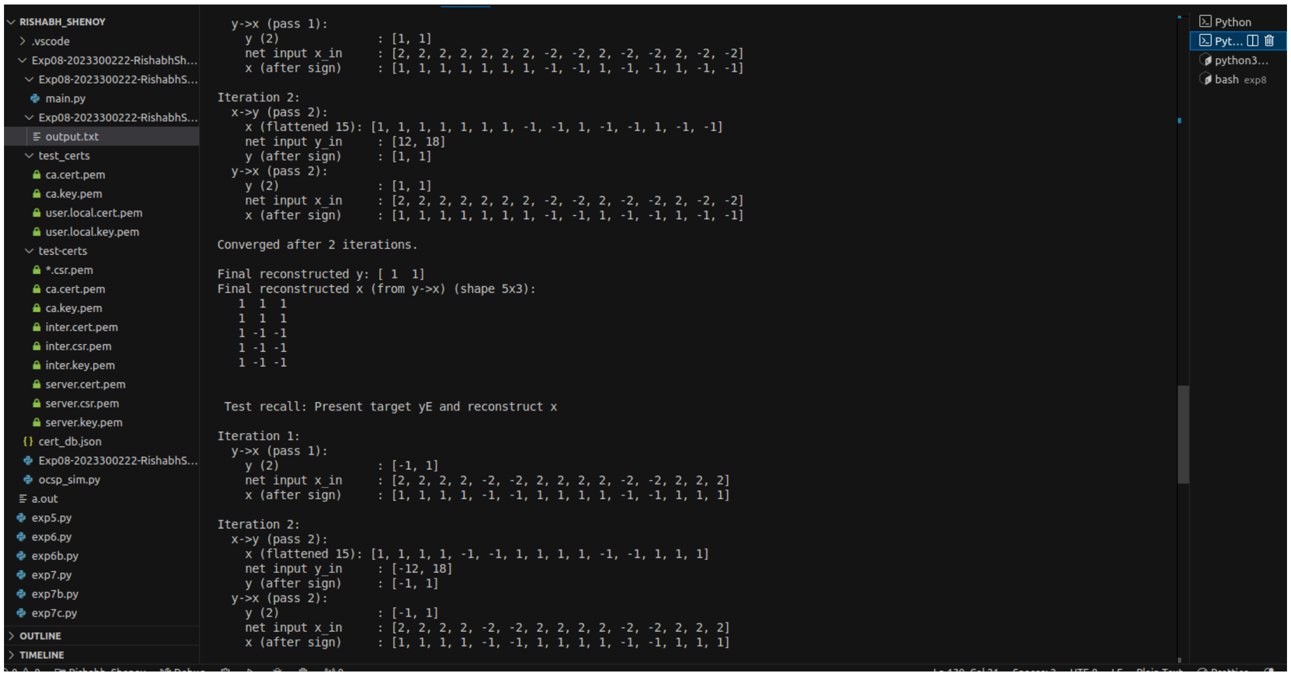
```
Testing with two missing entries:

Test input: [0 0 1 1]
Net input (yinj) = [-2  2  2  2]
Output (yj) = [-1  1  1  1]

Test input: [-1  0  0  1]
Net input (yinj) = [-2  2  2  2]
Output (yj) = [-1  1  1  1]
psipl@psipl-OptiPlex-SFF-7010:~/Desktop/Rishabh_Shenoy$ /usr/bin/python3 /home/psipl/Desktop/Rishabh_Shenoy/exp7c.py

 Input display patterns (5x3)

Input E (shape 5x3):
   1  1  1
   1 -1 -1
   1  1  1
   1 -1 -1
   1  1  1

Input F (shape 5x3):
   1  1  1
   1  1  1
   1 -1 -1
   1 -1 -1
   1 -1 -1

Targets:
 yE = [-1  1]
 yF = [1 1]

--- Building BAM weight matrices ---

W1 = outer(xE, yE)  (15x2):
 [[-1  1]
 [-1  1]
 [-1  1]
 [-1  1]
 [ 1 -1]
 [ 1 -1]
 [-1  1]
 [-1  1]
 [-1  1]
 [-1  1]
```



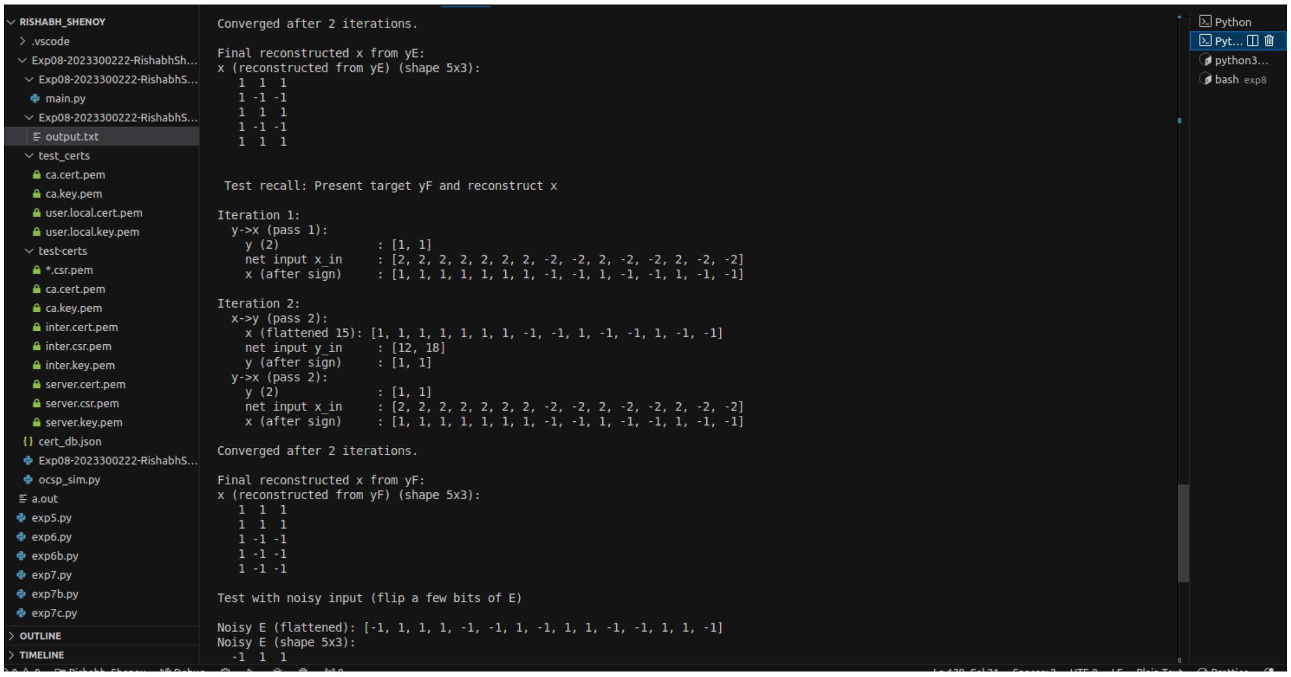
```
 [-1  1]
 [-1  1]]

W2 = outer(xF, yF)  (15x2):
 [[ 1  1]
 [ 1  1]
 [ 1  1]
 [ 1  1]
 [ 1  1]
 [ 1  1]
 [ 1  1]
 [-1 -1]
 [-1 -1]
 [ 1  1]
 [-1 -1]
 [-1 -1]
 [ 1  1]
 [-1 -1]
 [-1 -1]]

W = W1 + W2  (15x2):
 [[ 0  2]
 [ 0  2]
 [ 0  2]
 [ 0  2]
 [ 2  0]
 [ 2  0]
 [ 0  2]
 [-2  0]
 [-2  0]
 [ 0  2]
 [ 0 -2]
 [ 0 -2]
 [ 0  2]
 [-2  0]
 [-2  0]]

--- End of weight construction ---


 Test recall: Present stored input E and read output y

Iteration 1:
  x->y (pass 1):
```



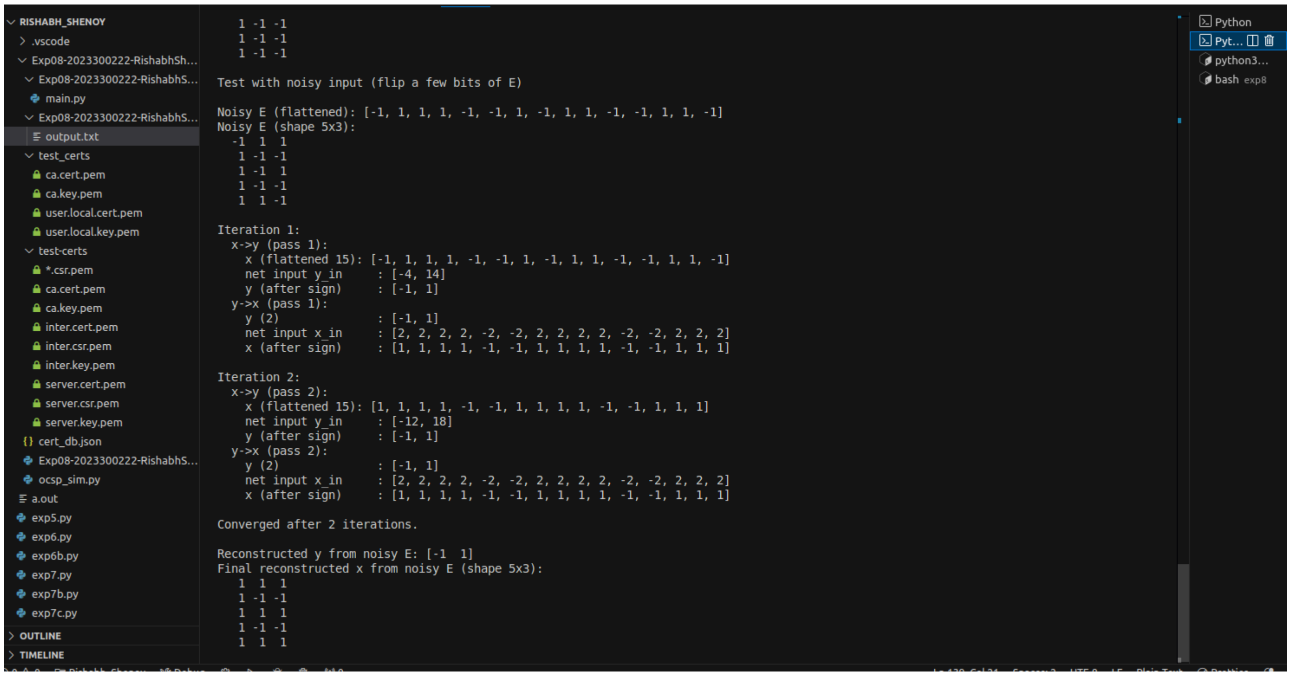
```
 Test recall: Present stored input E and read output y

Iteration 1:
  x->y (pass 1):
    x (flattened 15): [1, 1, 1, 1, -1, -1, 1, 1, 1, 1, -1, -1, 1, 1, 1]
    net input y_in     : [-12, 18]
    y (after sign)     : [-1, 1]
  y->x (pass 1):
    y (2)              : [-1, 1]
    net input x_in     : [2, 2, 2, 2, -2, -2, 2, 2, 2, 2, -2, -2, 2, 2, 2]
    x (after sign)     : [1, 1, 1, 1, -1, -1, 1, 1, 1, 1, -1, -1, 1, 1, 1]

Iteration 2:
  x->y (pass 2):
    x (flattened 15): [1, 1, 1, 1, -1, -1, 1, 1, 1, 1, -1, -1, 1, 1, 1]
    net input y_in     : [-12, 18]
    y (after sign)     : [-1, 1]
  y->x (pass 2):
    y (2)              : [-1, 1]
    net input x_in     : [2, 2, 2, 2, -2, -2, 2, 2, 2, 2, -2, -2, 2, 2, 2]
    x (after sign)     : [1, 1, 1, 1, -1, -1, 1, 1, 1, 1, -1, -1, 1, 1, 1]

Converged after 2 iterations.

Final reconstructed y: [-1  1]
Final reconstructed x (from y->x) (shape 5x3):
   1  1  1
   1 -1 -1
   1  1  1
   1 -1 -1
   1  1  1


 Test recall: Present stored input F and read output y

Iteration 1:
  x->y (pass 1):
    x (flattened 15): [1, 1, 1, 1, 1, 1, 1, -1, -1, 1, -1, -1, 1, -1, -1]
    net input y_in     : [12, 18]
    y (after sign)     : [1, 1]
  y->x (pass 1):
    y (2)              : [1, 1]
    net input x in     : [2, 2, 2, 2, 2, 2, 2, -2, -2, 2, -2, -2, 2, -2, -2]
```



```
  y->x (pass 1):
    y (2)             : [1, 1]
    net input x_in    : [2, 2, 2, 2, 2, 2, 2, -2, -2, 2, -2, -2, 2, -2, -2]
    x (after sign)    : [1, 1, 1, 1, 1, 1, 1, -1, -1, 1, -1, -1, 1, -1, -1]

Iteration 2:
  x->y (pass 2):
    x (flattened 15): [1, 1, 1, 1, 1, 1, 1, -1, -1, 1, -1, -1, 1, -1, -1]
    net input y_in    : [12, 18]
    y (after sign)    : [1, 1]
  y->x (pass 2):
    y (2)             : [1, 1]
    net input x_in    : [2, 2, 2, 2, 2, 2, 2, -2, -2, 2, -2, -2, 2, -2, -2]
    x (after sign)    : [1, 1, 1, 1, 1, 1, 1, -1, -1, 1, -1, -1, 1, -1, -1]

Converged after 2 iterations.

Final reconstructed y: [ 1  1]
Final reconstructed x (from y->x) (shape 5x3):
   1  1  1
   1  1  1
   1 -1 -1
   1 -1 -1
   1 -1 -1


 Test recall: Present target yE and reconstruct x

Iteration 1:
  y->x (pass 1):
    y (2)             : [-1, 1]
    net input x_in    : [2, 2, 2, 2, -2, -2, 2, 2, 2, 2, -2, -2, 2, 2, 2]
    x (after sign)    : [1, 1, 1, 1, -1, -1, 1, 1, 1, 1, -1, -1, 1, 1, 1]

Iteration 2:
  x->y (pass 2):
    x (flattened 15): [1, 1, 1, 1, -1, -1, 1, 1, 1, 1, -1, -1, 1, 1, 1]
    net input y_in    : [-12, 18]
    y (after sign)    : [-1, 1]
  y->x (pass 2):
    y (2)             : [-1, 1]
    net input x_in    : [2, 2, 2, 2, -2, -2, 2, 2, 2, 2, -2, -2, 2, 2, 2]
    x (after sign)    : [1, 1, 1, 1, -1, -1, 1, 1, 1, 1, -1, -1, 1, 1, 1]
```



```
Converged after 2 iterations.

Final reconstructed x from yE:
x (reconstructed from yE) (shape 5x3):
   1  1  1
   1 -1 -1
   1  1  1
   1 -1 -1
   1  1  1


 Test recall: Present target yF and reconstruct x

Iteration 1:
  y->x (pass 1):
    y (2)             : [1, 1]
    net input x_in    : [2, 2, 2, 2, 2, 2, 2, -2, -2, 2, -2, -2, 2, -2, -2]
    x (after sign)    : [1, 1, 1, 1, 1, 1, 1, -1, -1, 1, -1, -1, 1, -1, -1]

Iteration 2:
  x->y (pass 2):
    x (flattened 15): [1, 1, 1, 1, 1, 1, 1, -1, -1, 1, -1, -1, 1, -1, -1]
    net input y_in    : [12, 18]
    y (after sign)    : [1, 1]
  y->x (pass 2):
    y (2)             : [1, 1]
    net input x_in    : [2, 2, 2, 2, 2, 2, 2, -2, -2, 2, -2, -2, 2, -2, -2]
    x (after sign)    : [1, 1, 1, 1, 1, 1, 1, -1, -1, 1, -1, -1, 1, -1, -1]

Converged after 2 iterations.

Final reconstructed x from yF:
x (reconstructed from yF) (shape 5x3):
   1  1  1
   1  1  1
   1 -1 -1
   1 -1 -1
   1 -1 -1

Test with noisy input (flip a few bits of E)

Noisy E (flattened): [-1, 1, 1, 1, -1, -1, 1, -1, 1, 1, -1, -1, 1, 1, -1]
Noisy E (shape 5x3):
  -1  1  1
```



```
   1 -1 -1
   1 -1 -1
   1 -1 -1

Test with noisy input (flip a few bits of E)

Noisy E (flattened): [-1, 1, 1, 1, -1, -1, 1, -1, 1, 1, -1, -1, 1, 1, -1]
Noisy E (shape 5x3):
  -1  1  1
   1 -1 -1
   1 -1  1
   1 -1 -1
   1  1 -1

Iteration 1:
  x->y (pass 1):
    x (flattened 15): [-1, 1, 1, 1, -1, -1, 1, -1, 1, 1, -1, -1, 1, 1, -1]
    net input y_in     : [-4, 14]
    y (after sign)     : [-1, 1]
  y->x (pass 1):
    y (2)              : [-1, 1]
    net input x_in     : [2, 2, 2, 2, -2, -2, 2, 2, 2, 2, -2, -2, 2, 2, 2]
    x (after sign)     : [1, 1, 1, 1, -1, -1, 1, 1, 1, 1, -1, -1, 1, 1, 1]

Iteration 2:
  x->y (pass 2):
    x (flattened 15): [1, 1, 1, 1, -1, -1, 1, 1, 1, 1, -1, -1, 1, 1, 1]
    net input y_in     : [-12, 18]
    y (after sign)     : [-1, 1]
  y->x (pass 2):
    y (2)              : [-1, 1]
    net input x_in     : [2, 2, 2, 2, -2, -2, 2, 2, 2, 2, -2, -2, 2, 2, 2]
    x (after sign)     : [1, 1, 1, 1, -1, -1, 1, 1, 1, 1, -1, -1, 1, 1, 1]

Converged after 2 iterations.

Reconstructed y from noisy E: [-1  1]
Final reconstructed x from noisy E (shape 5x3):
   1  1  1
   1 -1 -1
   1  1  1
   1 -1 -1
   1  1  1
```

CONCLUSION:	From this experiment, I have understood how Auto Associative and Bidirectional Associative Memory (BAM) networks function to store and recall patterns through associative learning. In the Auto Associative Network, I learned to construct the weight matrix using the outer product of the input vector and observed how the network could recall the original pattern even with missing or incorrect entries, demonstrating its pattern completion capability. By performing matrix multiplication using the dot product, I could verify the network's ability to generate correct outputs after applying activation functions. In the BAM network implementation, I understood how the system establishes a two-way association between input and target patterns, allowing recall in both directions. This experiment helped me clearly understand how memory association, weight computation, and activation work together in neural networks and how these concepts can be practically implemented and tested using Python code.