Cody Costa

EE263

8/4/2025

Homework 3

PYTHON IMAGE RESULTS:

Please see attached code files to assignment submission for step-by-step procedure

PROBLEM 1)

```
14
15  ''' PROBLEM 1:  DCT/IDCT 4X4 '''
16
17  # starting matrix
18  matrix = np.array([[1, 0, 1, 0],
19                     [2, 0, 2, 0],
20                     [0, 1, 0, 1],
21                     [-1, 0, -1, 0]])
22
23  # compute DCT
24  dct_matrix = dctn(matrix, type=2, norm='ortho')
25
26  # compute IDCT
27  idct_matrix = idctn(matrix, type=2, norm='ortho')
28
29  # print results
30  print(f'DCT:\n{dct_matrix}\n\nIDCT:\n{idct_matrix}')
31
```

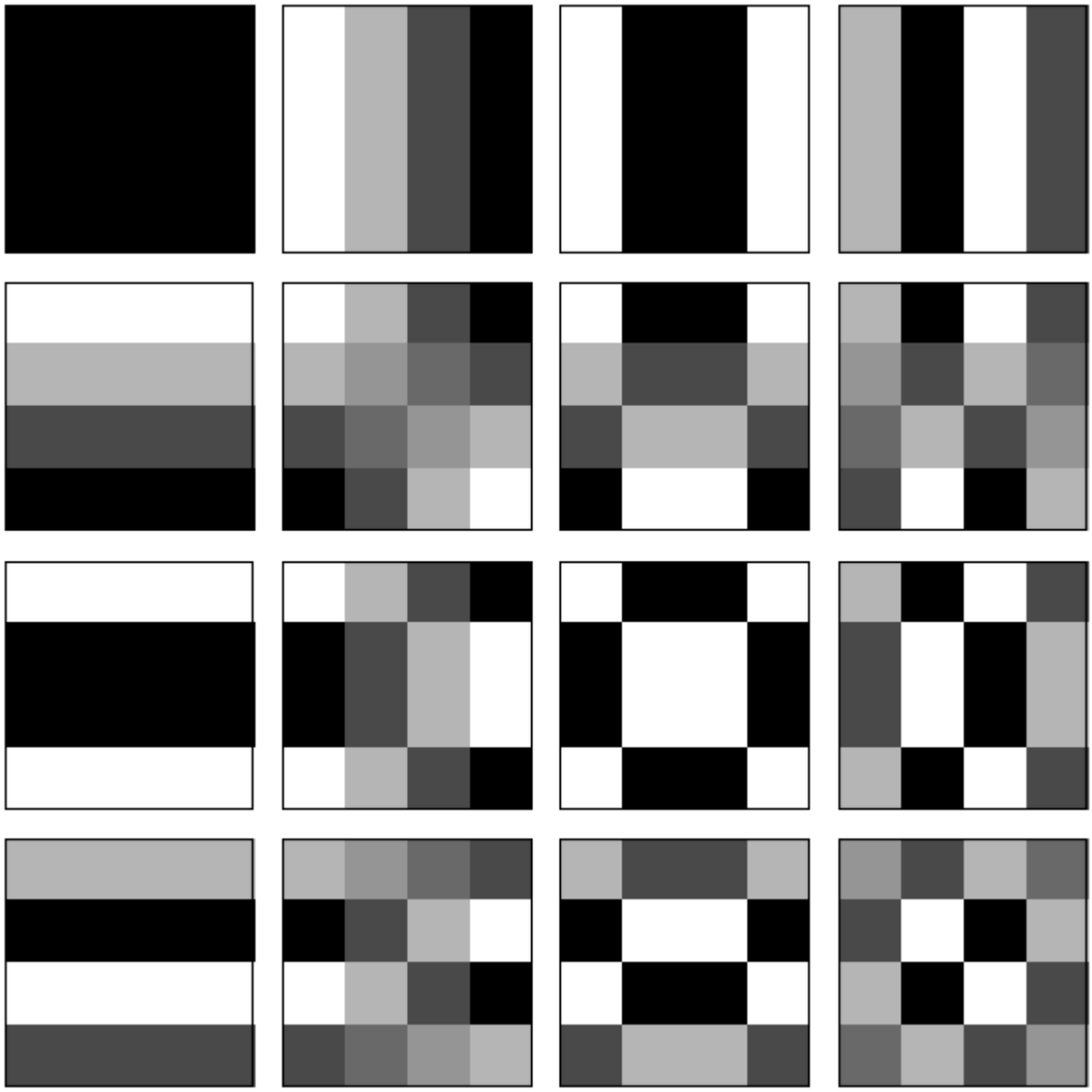
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
→ py .\main.py
DCT:
[[ 1.5         0.19134172  0.          0.46193977]
 [ 1.57716101  0.81066017  0.          1.95710678]
 [-1.5        -0.19134172  0.         -0.46193977]
 [-0.11208538 -0.54289322  0.         -1.31066017]]

IDCT:
[[ 1.99790468 -0.19134172  0.19134172  1.07402515]
 [ 1.23253782  0.19134172 -0.19134172  2.15641735]
 [-1.15641735  0.19134172 -0.19134172 -0.23253782]
 [-0.07402515 -0.19134172  0.19134172 -0.99790468]]
```

PROBLEM 2)

```
''' PROBLEM 2:  BASIS FUNCTIONS OF DCT '''

def dct_basis(N):
    basis = []
    for u in range(N):
        for v in range(N):
            coeff = np.zeros((N, N))
            coeff[u, v] = 1  # Impulse at (u, v)

            # 2D inverse DCT with orthogonal normalization
            basis_fn = idctn(coeff, norm='ortho')
            basis.append(basis_fn)
    return basis


N = 4
basis_funcs = dct_basis(N)

# # Plot 16 basis functions
fig, axes = plt.subplots(N, N, figsize=(6, 6))
for i, ax in enumerate(axes.flat):
    ax.imshow(basis_funcs[i], cmap='gray')
    ax.axis('off')

    # Add border (rectangle)
    rect = patches.Rectangle(
        (0, 0), 1, 1, transform=ax.transAxes,
        linewidth=1.5, edgecolor='black', facecolor='none'
    )
    ax.add_patch(rect)

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

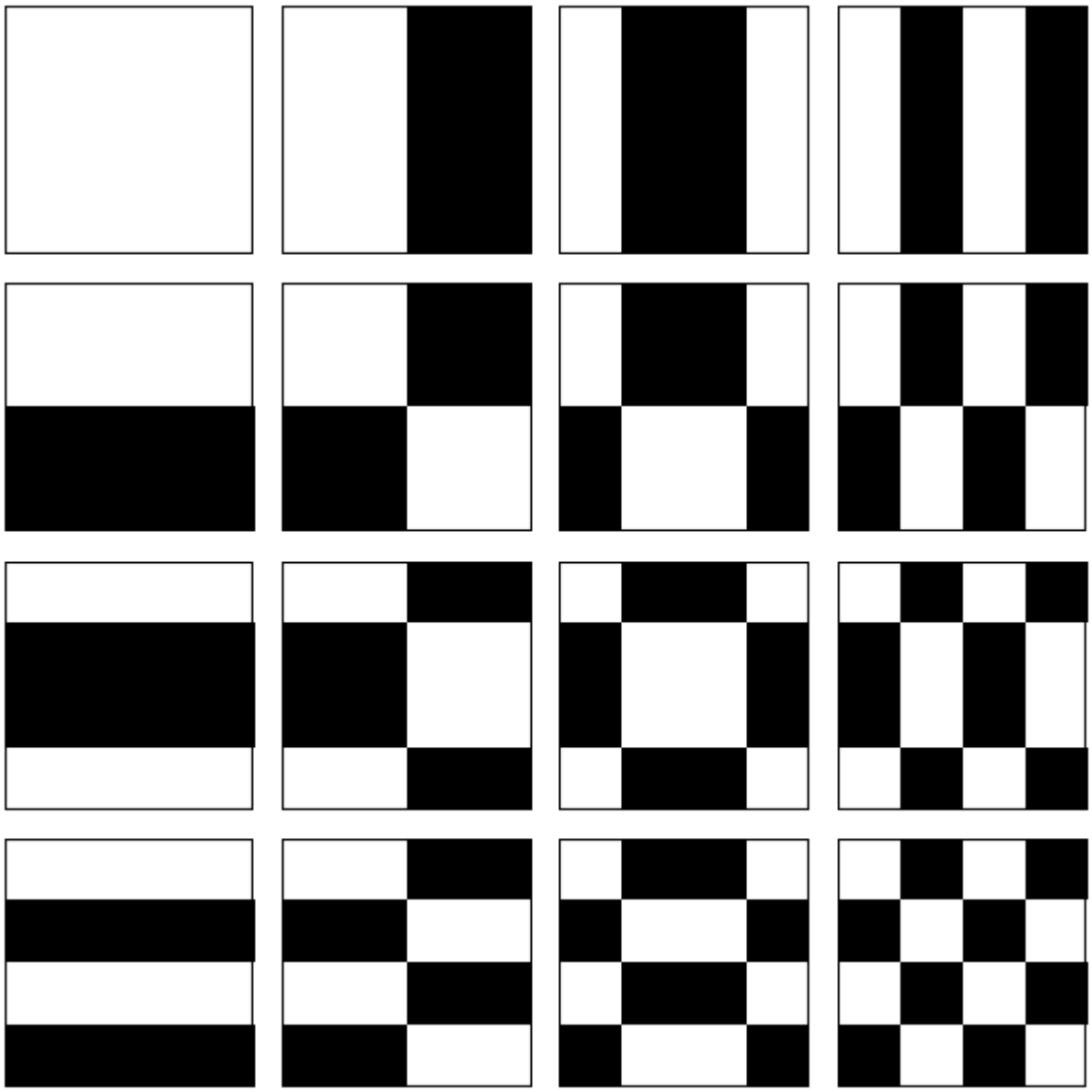
Output:



PROBLEM 3)

```
''' PROBLEM 3:  WALSH HADAMARD TRANSFORM '''

def sequency_order(H):
    """Reorder Hadamard matrix rows to sequency order."""
    n = H.shape[0]
    # Count sign changes in each row
    sign_changes = [np.sum(H[i, :-1] ≠ H[i, 1:]) for i in range(n)]
    order = np.argsort(sign_changes)  # Sort by sign changes
    return H[order]

# Generate 4×4 Hadamard matrix
H = hadamard(4)

# Convert to sequency order
H_seq = sequency_order(H)

# Generate 2D Walsh-Hadamard basis
def walsh_hadamard_basis(H):
    n = H.shape[0]
    basis = []
    for u in range(n):
        for v in range(n):
            basis.append(np.outer(H[u], H[v]))
    return basis

basis = walsh_hadamard_basis(H_seq)

# Plot 16 basis functions (4×4 grid)
fig, axes = plt.subplots(4, 4, figsize=(6, 6))
for i, ax in enumerate(axes.flat):
    ax.imshow(basis[i], cmap='gray', vmin=-1, vmax=1)
    ax.axis('off')

    # Add border (rectangle)
    rect = patches.Rectangle(
        (0, 0), 1, 1, transform=ax.transAxes,
        linewidth=1.5, edgecolor='black', facecolor='none'
    )
    ax.add_patch(rect)

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

Output:

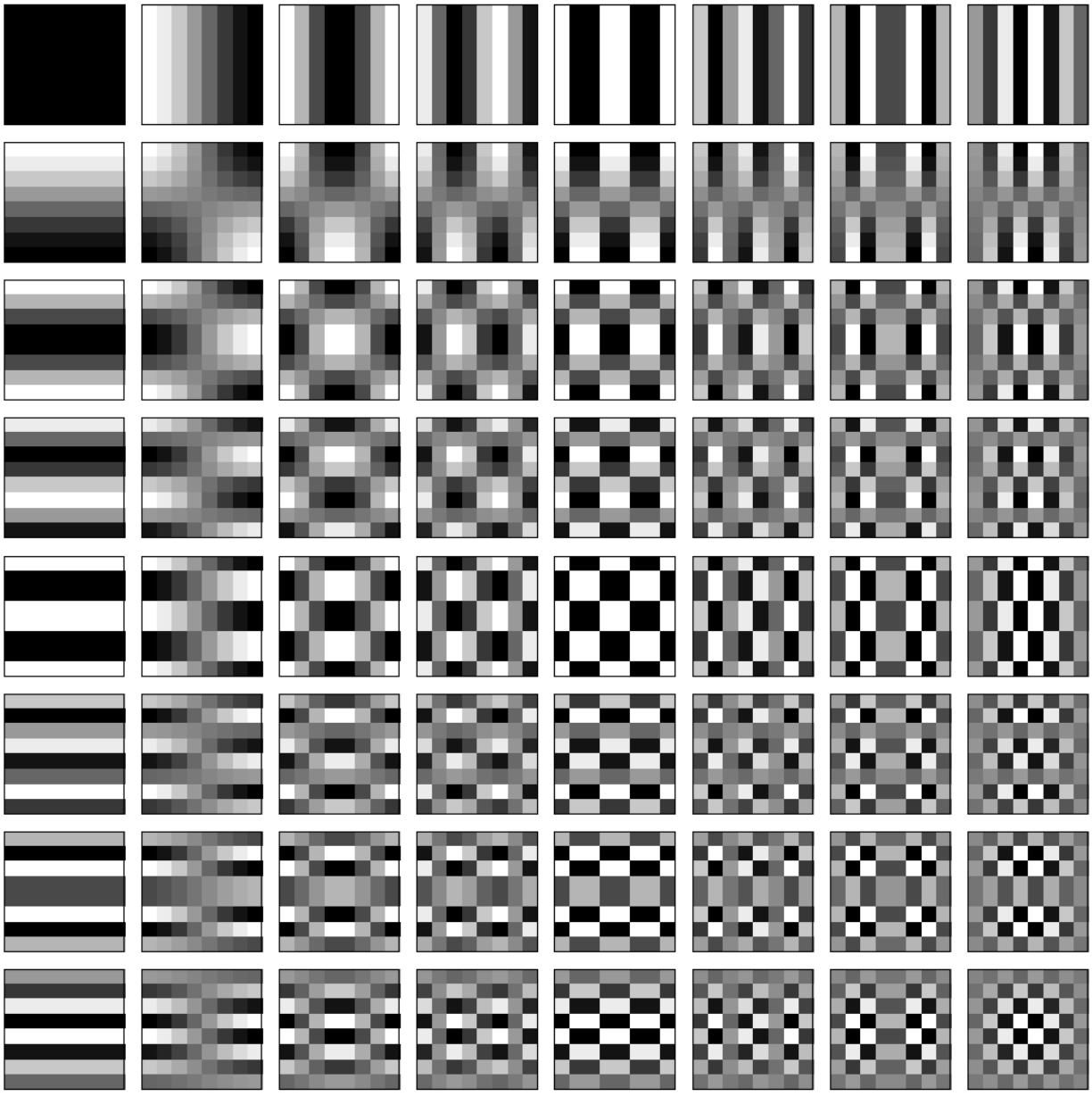


PROBLEM 4)

Reused earlier user defined function dct_basis with N = 8

```
''' PROBLEM 4:  8X8 DCT BASIS '''

N = 8
basis_functions = dct_basis(N)

# Plot 64 basis functions (8×8 grid)
fig, axes = plt.subplots(N, N, figsize=(10, 10))
for i, ax in enumerate(axes.flat):
    ax.imshow(basis_functions[i], cmap='gray')
    ax.axis('off')

    # Add border (rectangle)
    rect = patches.Rectangle(
        (0, 0), 1, 1, transform=ax.transAxes,
        linewidth=1.5, edgecolor='black', facecolor='none'
    )
    ax.add_patch(rect)

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

Output:



PROBLEM 5)

```
''' PROBLEM 5:  AREA OF CHAIN CLOSURE '''

# 8-connectivity direction vectors
directions = {
    0: (1, 0),   1: (1, -1),  2: (0, -1),  3: (-1, -1),
    4: (-1, 0),  5: (-1, 1),  6: (0, 1),   7: (1, 1)
}

def chain_code_area(chain_code):
    # Convert string to list of integers
    chain = [int(c) for c in str(chain_code)]

    # Starting point
    x, y = 0, 0
    points = [(x, y)]

    # Build boundary points
    for d in chain:
        dx, dy = directions[d]
        x += dx
        y += dy
        points.append((x, y))

    # Shoelace formula
    points = np.array(points)
    x = points[:, 0]
    y = points[:, 1]
    area = 0.5 * abs(np.dot(x, np.roll(y, -1)) - np.dot(y, np.roll(x, -1)))

    return int(area)

print(chain_code_area(217644))
```

Output area for ‘217644’ = 3

PROBLEM 6)

```
''' PROBLEM 6:  HAND COMPUTE DCT/IDCT '''

mat = np.array([[9, 10, 5, 6],
                [10, 12, 8, 8],
                [12, 11, 7, 10],
                [14, 13, 8, 5]])

# DCT compute (no scipy)
def manual_dct2(matrix):
    N = matrix.shape[0]
    dct = np.zeros([N, N])

    for u in range(N):
        for v in range(N):
            alpha_u = np.sqrt(1 / N) if u == 0 else np.sqrt(2 / N)
            alpha_v = np.sqrt(1 / N) if v == 0 else np.sqrt(2 / N)

            sum_val = 0
            for x in range(N):
                for y in range(N):

                    sum_val += matrix[x, y] * \
                        np.cos((np.pi * (2 * x + 1) * u) / (2 * N)) * \
                        np.cos((np.pi * (2 * y + 1) * v) / (2 * N))

            dct[u, v] = alpha_u * alpha_v * sum_val

    return dct


# IDCT compute (no scipy)
def manual_idct2(dct_matrix):
    N = dct_matrix.shape[0]
    img = np.zeros([N, N])

    for x in range(N):
        for y in range(N):

            sum_val = 0

            for u in range(N):
                for v in range(N):
                    alpha_u = np.sqrt(1 / N) if u == 0 else np.sqrt(2 / N)
                    alpha_v = np.sqrt(1 / N) if v == 0 else np.sqrt(2 / N)

                    sum_val += alpha_u * alpha_v * dct_matrix[u, v] * \
                                np.cos((np.pi * (2 * x + 1) * u) / (2 * N)) * \
                                np.cos((np.pi * (2 * y + 1) * v) / (2 * N))

            img[x, y] = sum_val
    return img



DCT_RES = manual_dct2(mat)
print(np.round(DCT_RES, 2))

IMG = manual_idct2(DCT_RES)
print(np.round(IMG, 2))
```

Input array:

```
mat = np.array([[9, 10, 5, 6],
                [10, 12, 8, 8],
                [12, 11, 7, 10],
                [14, 13, 8, 5]])
```

Output:

DCT

```
[[37.    7.66 -0.   -3.71]
 [-3.54 -2.56 -0.16 -1.06]
 [-2.    2.88 -1.    0.43]
 [-0.7  -1.06  2.23 -0.44]]
```

IDCT of DCT

```
[[ 9. 10.  5.  6.]
 [10. 12.  8.  8.]
 [12. 11.  7. 10.]
 [14. 13.  8.  5.]]
```

Idct(dct(mat)) == mat

Results check out