Faculty of Engineering
Cairo University

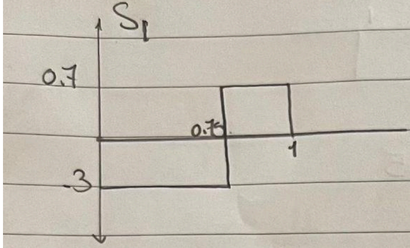
EECS316: Communications-2
Gram-Schmidt Assignment

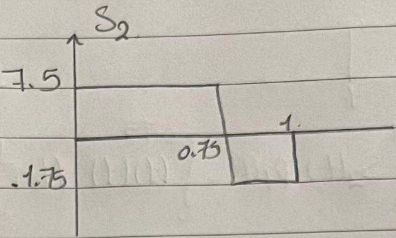
Prepared by:

- Nada Abdallah Mahmoud El Said 1210381
- Maged Amgad Rasmy 1210375
- Mostafa Osama Nassar 1210377
- Karim Mohamed Sayed 1210111

Part 1: Hand Analysis

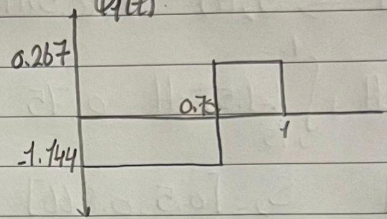
Problem 1:-





$E_1 = \int_0^1 S_1^2\,dt = \int_0^{0.75} (-3)^2\,dt + \int_{0.75}^1 (0.7)^2\,dt = 6.87$, $S_{11} = \sqrt{E_1} = 2.621$

$\phi_1(t) = \frac{S_1(t)}{\sqrt{E}}$

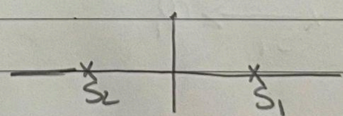


$S_{21} = \int_0^1 S_2(t)\,\phi_1(t)\,dt$

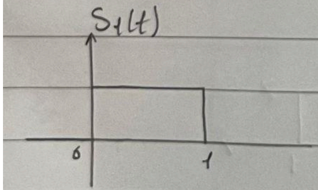
$= \int_0^{0.75} (7.5)(-1.144) + \int_{0.75}^1 (-1.75)(0.267) = -6.55$, $g_2(t) = S_2(t) - S_{21}\phi_1(t) = 0$

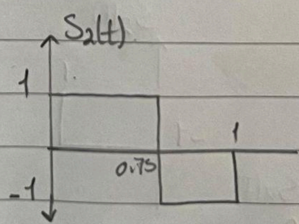
$S_1 = [2.621]$, $S_2 = [-6.55]$

Constellation Diagram $\Rightarrow$

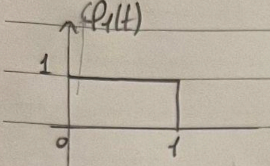


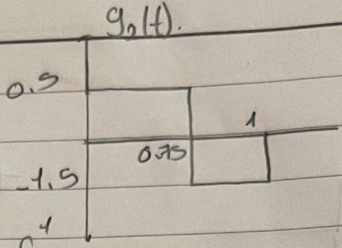
Problem 2:





$E_1 = \int_0^1 (1)^2\,dt = 1$, $\phi_1(t) = \frac{S_1(t)}{\sqrt{E}} = S_1(t)$





$$S_{21} = \int_0^1 S_2(t)\varphi_1(t) = \int_0^{0.75}(1)(1)\,dt + \int_{0.75}^{1}(-1)(1) = 0.5.$$

$g_2(t) = S_2 - S_{21}\varphi_1$

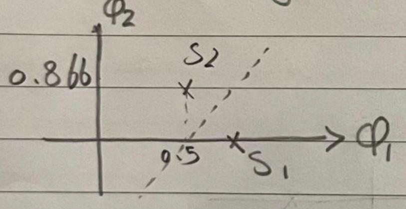
$0<t<0.75 \Rightarrow g_2(t) = 1 - 0.5(1) = 0.5$

$0.75<t<1 \Rightarrow g_2(t) = -1-(0.5)(1) = -1.5.$

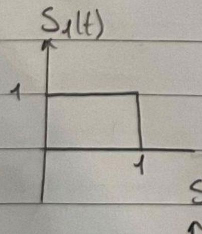
$$E_{g_2} = \int_0^{0.75}(0.5)^2\,dt + \int_{0.75}^{1}(-1.5)^2\,dt = 0.75. \quad , \varphi_2(t) = \frac{g_2(t)}{\sqrt{E}} = \frac{g_2(t)}{0.866}$$

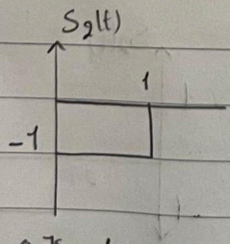
$S_1 = [1, 0]$, $S_2 = [0.5, 0.866]$.

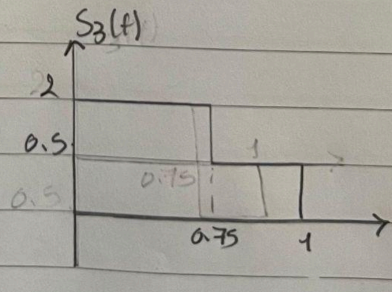
Constellation Diagram

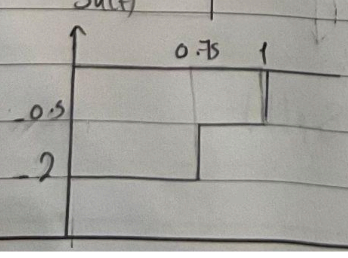


Problem 3



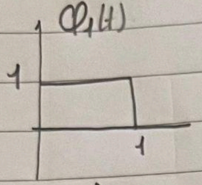


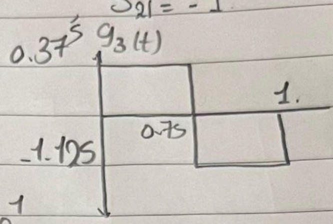




الموضوع: ____________ التاريخ: / /

$S_{11} = \sqrt{E_1} = 1$, $\phi_1 = \frac{S_1}{\sqrt{E}} = 1.$ $S_{21} = -1$





$$S_{31} = \int s_3 \phi_1 \, dt = \int_0^{0.75} (2)(1)\, dt + \int_{0.75}^{1} (0.5)(1)\, dt = 1.5 + 0.125 = 1.625.$$

$g_3 = S_3 - S_{31}\phi_1 \Rightarrow 0 < t < 0.75 \Rightarrow 2 - 1.625 = 0.375.$

$0.75 < t < 1 \Rightarrow 0.5 - 1.625 = -1.125.$

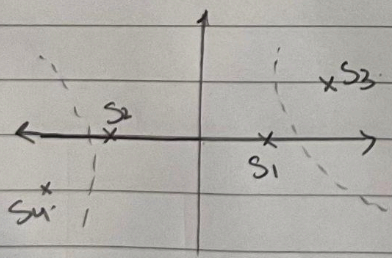
$E_{g_3} = \int_0^1 (g_3(t))^2 = 0.4218$, $S_{32} = \sqrt{E_{g_2}} = 0.6495$

$S_{41} = -1.625$, $S_{42} = -0.6495.$

$S_1 = [1, 0]$, $S_2 = [-1, 0]$, $S_3 = [1.1625, 0.6495]$

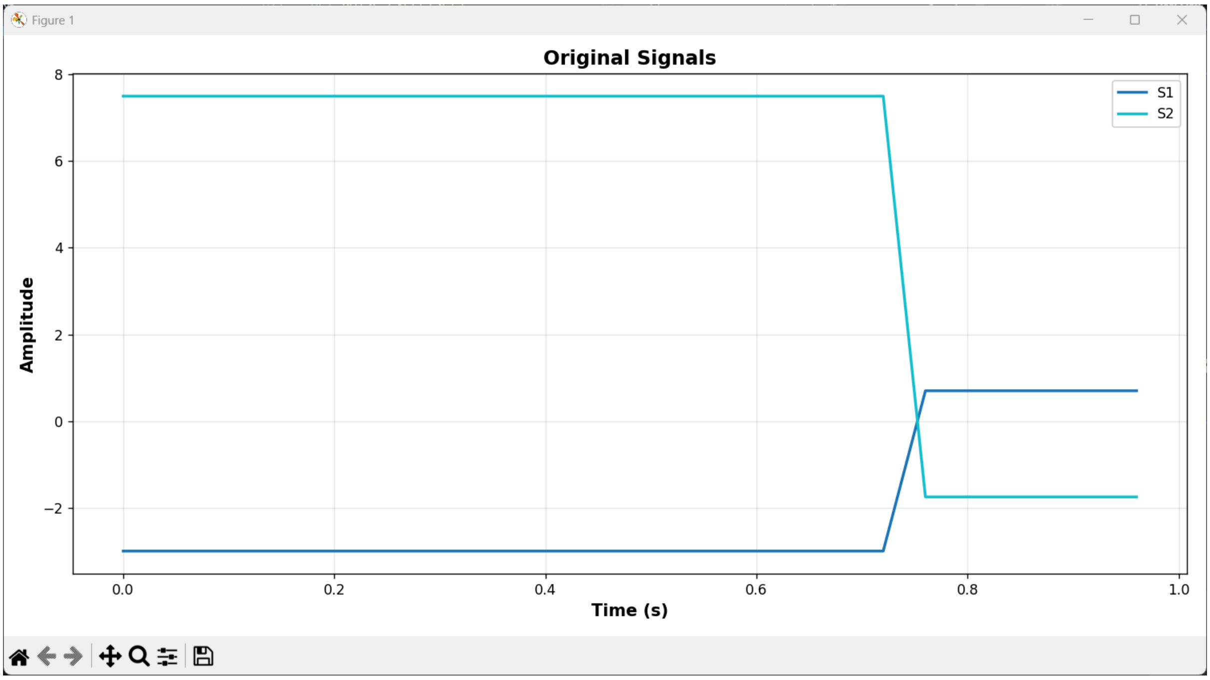
$S_4 = [-1.625, -0.6495]$

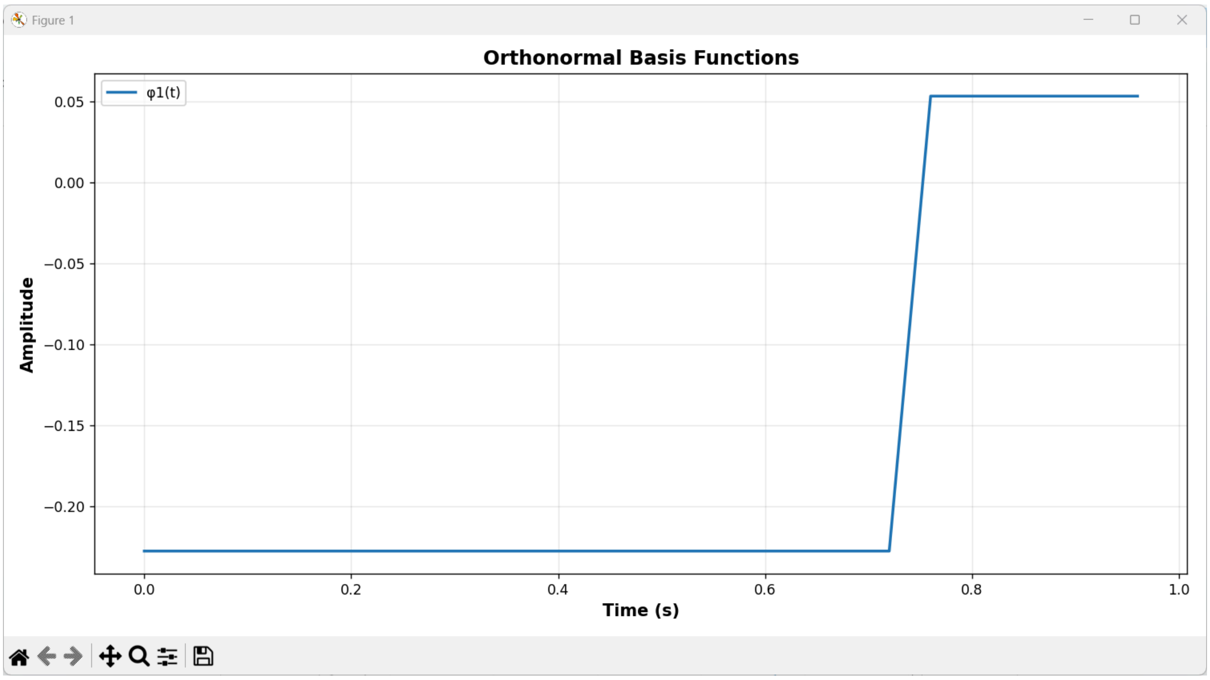
Constellation Diagram.

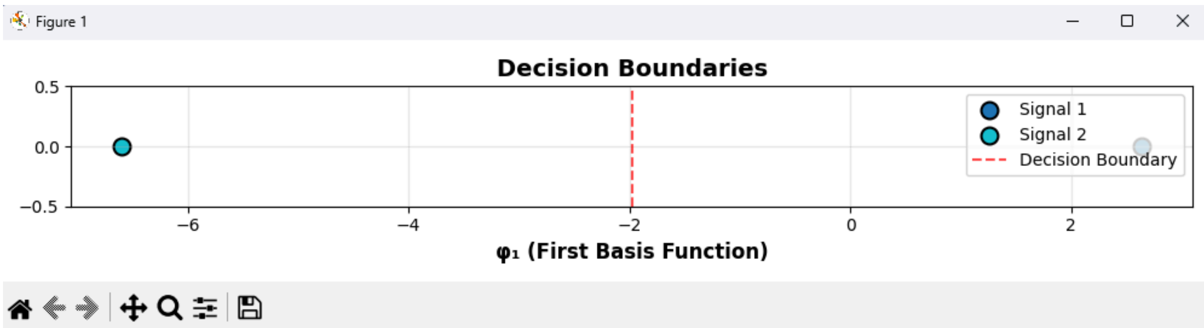


Part 2:

Problem 1:







```
======================================================================
PROBLEM 1
======================================================================

Number of basis functions: m = 1
Basis functions shape: (1, 25)

Signal Space Coefficients:
Signal 1 (S1): [2.63772629]
Signal 2 (S2): [-6.59431573]

======================================================================
SIGNAL SPACE ANALYSIS
======================================================================
Number of signals: 2
Number of basis functions: 1
----------------------------------------------------------------------
Euclidean Distances and Cross Correlations:
----------------------------------------------------------------------
Euclidean Distance Matrix:
       S1      S2
S1   0.0000  9.2320
S2   9.2320  0.0000

Cross Correlation Matrix:
       S1      S2
S1   1.0000 -1.0000
S2  -1.0000  1.0000
----------------------------------------------------------------------
Minimum Distance Analysis:
----------------------------------------------------------------------
Minimum Euclidean Distance: 9.232042

Signal pairs with minimum distance:
  Signal 1 <-> Signal 2:
    Distance: 9.232042
    Cross Correlation: -1.000000
  Signal 2 <-> Signal 1:
    Distance: 9.232042
    Cross Correlation: -1.000000
======================================================================
```

Problem 2:

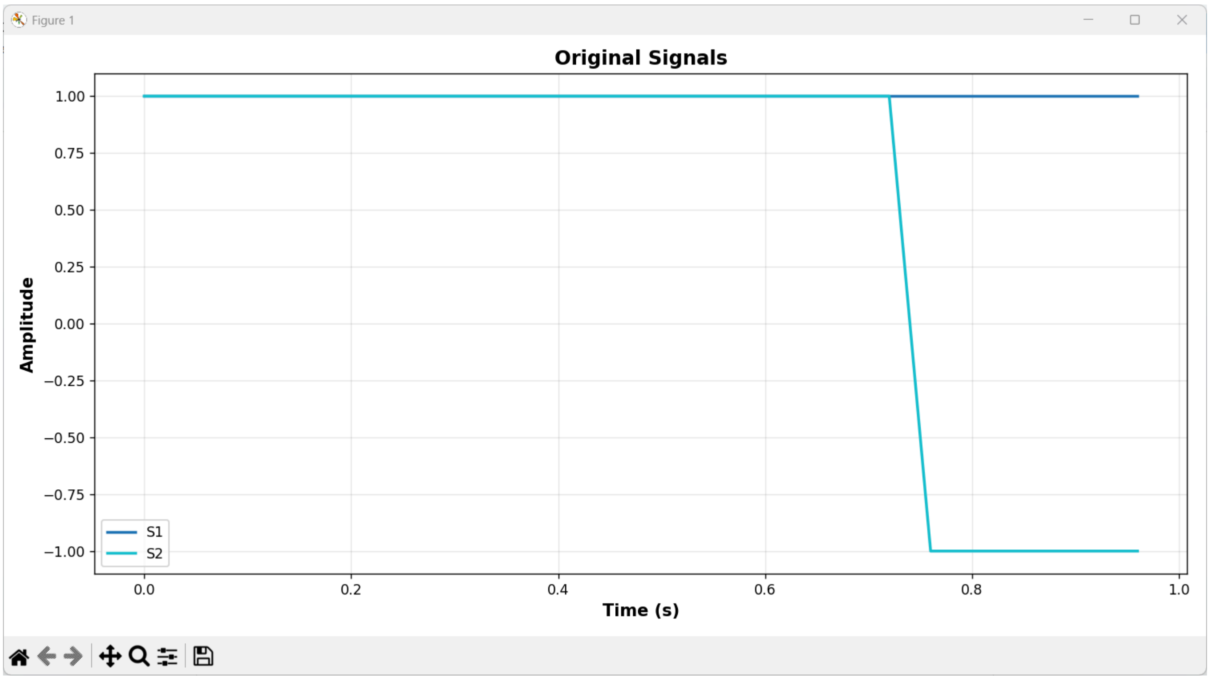


Figure 1
Original Signals
Amplitude
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
S1
S2
0.0
0.2
0.4
0.6
0.8
1.0
Time (s)

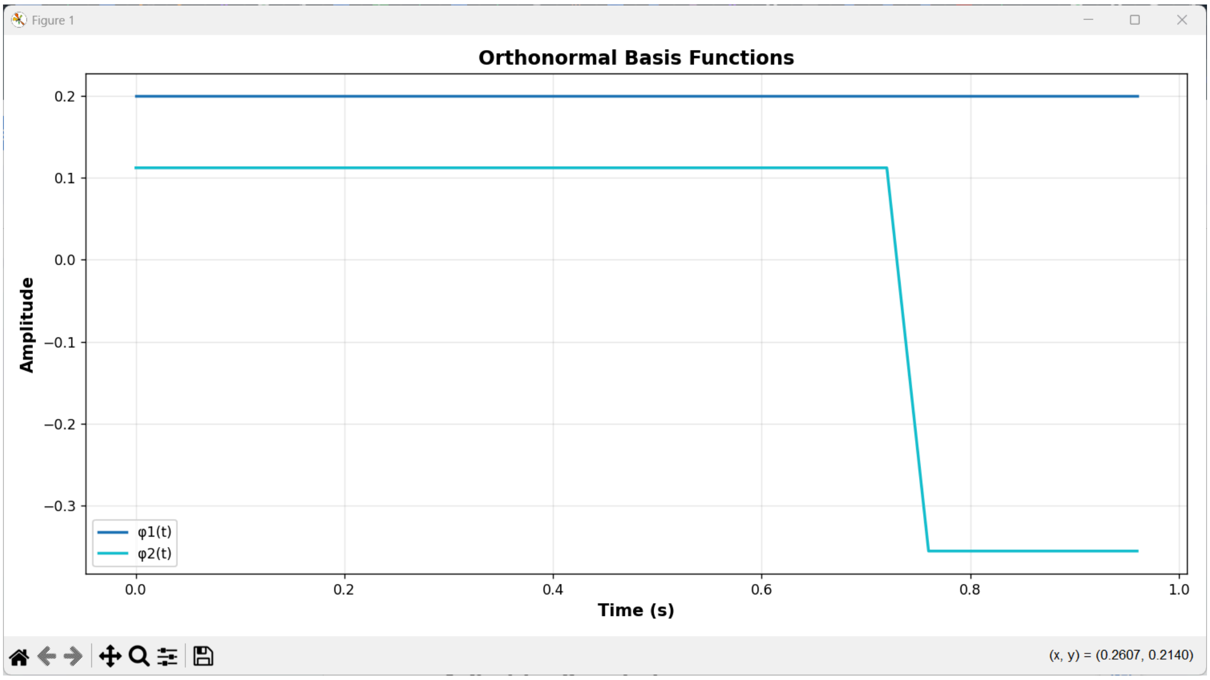
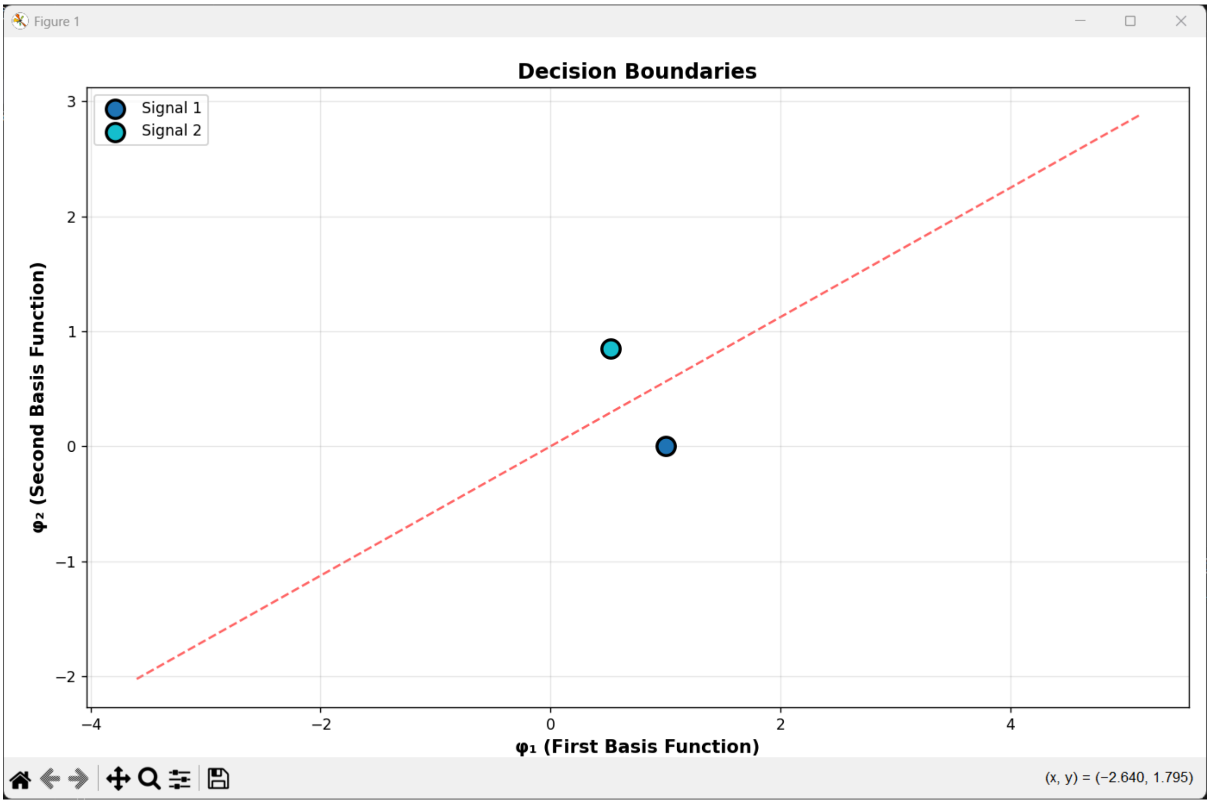


Figure 1
Orthonormal Basis Functions
Amplitude
0.2
0.1
0.0
−0.1
−0.2
−0.3
φ1(t)
φ2(t)
0.0
0.2
0.4
0.6
0.8
1.0
Time (s)
(x, y) = (0.2607, 0.2140)



==

PROBLEM 2

==

Number of basis functions: m = 2

Basis functions shape: (2, 25)

Signal Space Coefficients:

Signal 1 (S1): [1. 0.]

Signal 2 (S2): [0.52 0.85416626]

==

SIGNAL SPACE ANALYSIS

==

Number of signals: 2

Number of basis functions: 2

Euclidean Distances and Cross Correlations:

Euclidean Distance Matrix:

	S1	S2
S1	0.0000	0.9798
S2	0.9798	0.0000

```
Cross Correlation Matrix:
       S1     S2
S1   1.0000  0.5200
S2   0.5200  1.0000
----------------------------------------------------------------------
Minimum Distance Analysis:
----------------------------------------------------------------------
Minimum Euclidean Distance: 0.979796
Signal pairs with minimum distance:
  Signal 1 <-> Signal 2:
    Distance: 0.979796
    Cross Correlation: 0.520000
  Signal 2 <-> Signal 1:
    Distance: 0.979796
    Cross Correlation: 0.520000
======================================================================
```

Problem 3:

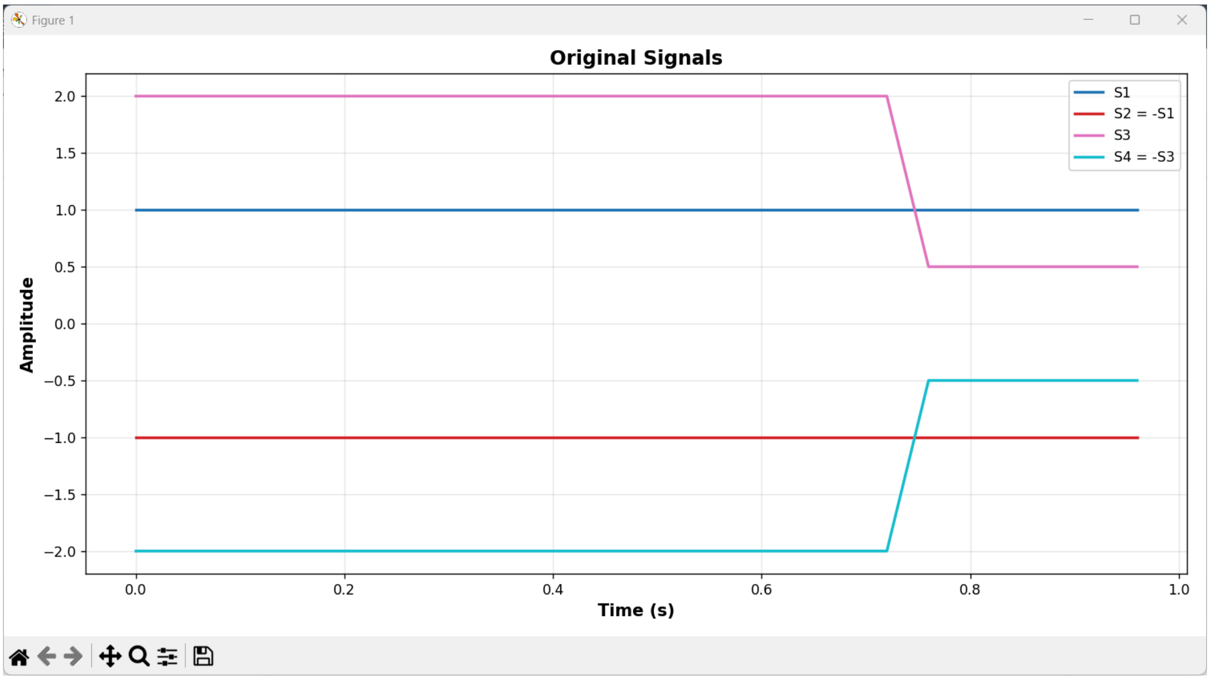


Figure 1
Original Signals
S1
S2 = -S1
S3
S4 = -S3
Amplitude
Time (s)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0.0
0.2
0.4
0.6
0.8
1.0

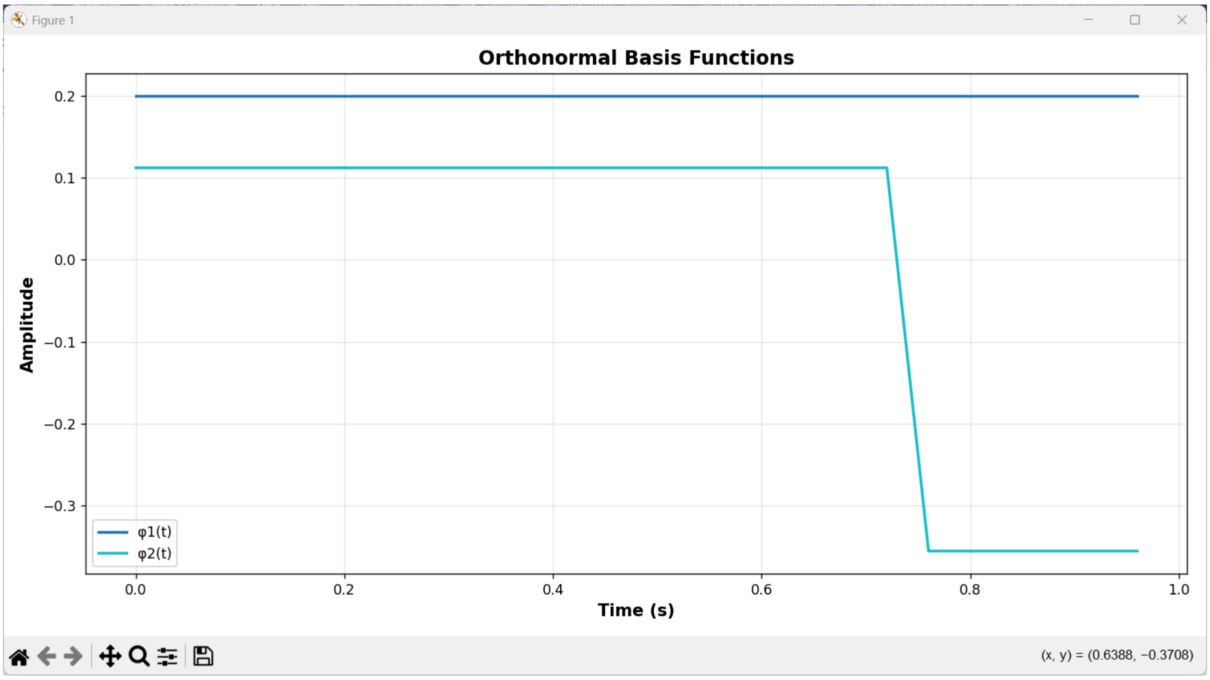
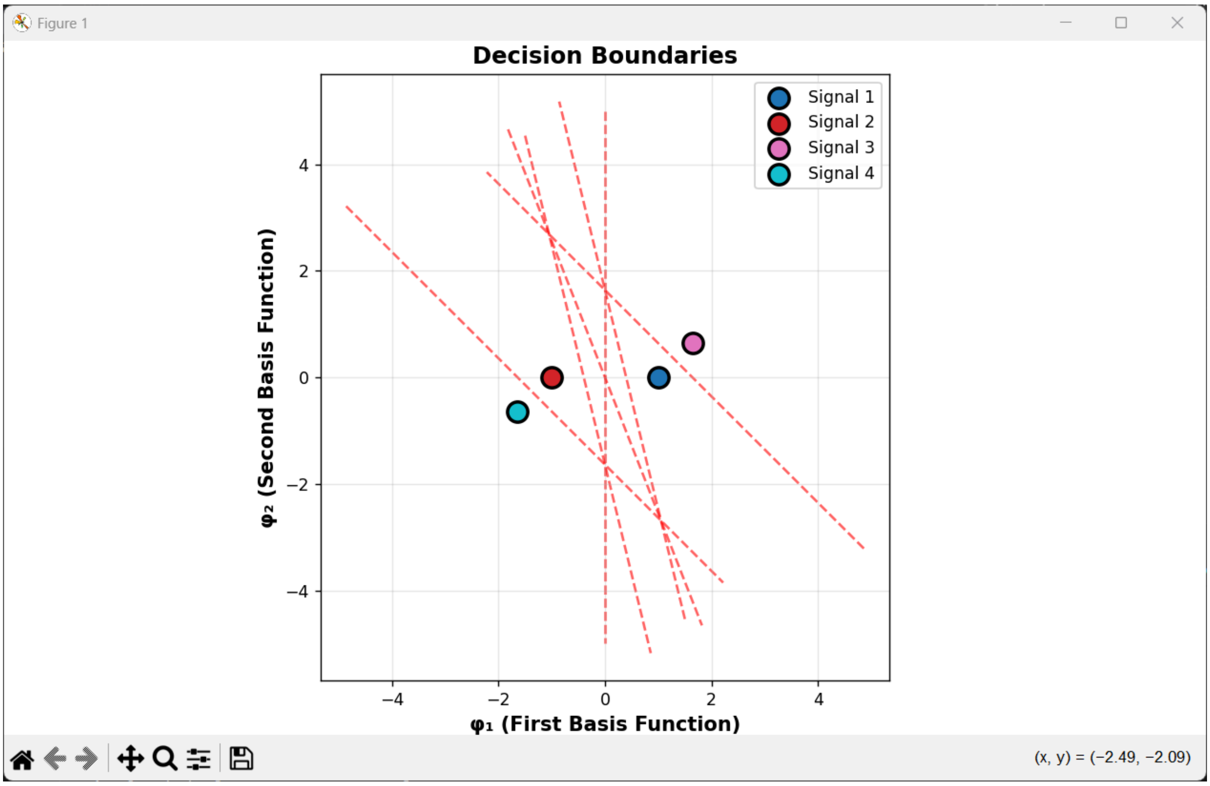


Figure 1
Orthonormal Basis Functions
φ1(t)
φ2(t)
Amplitude
Time (s)
0.2
0.1
0.0
−0.1
−0.2
−0.3
0.0
0.2
0.4
0.6
0.8
1.0
(x, y) = (0.6388, −0.3708)



==

PROBLEM 3

==

Number of basis functions: m = 2
Basis functions shape: (2, 25)

Signal Space Coefficients:
Signal 1 (S1): [1. 0.]
Signal 2 (S2): [-1. 0.]
Signal 3 (S3): [1.64   0.6406247]
Signal 4 (S4): [-1.64   -0.6406247]

==

SIGNAL SPACE ANALYSIS

==

Number of signals: 4
Number of basis functions: 2

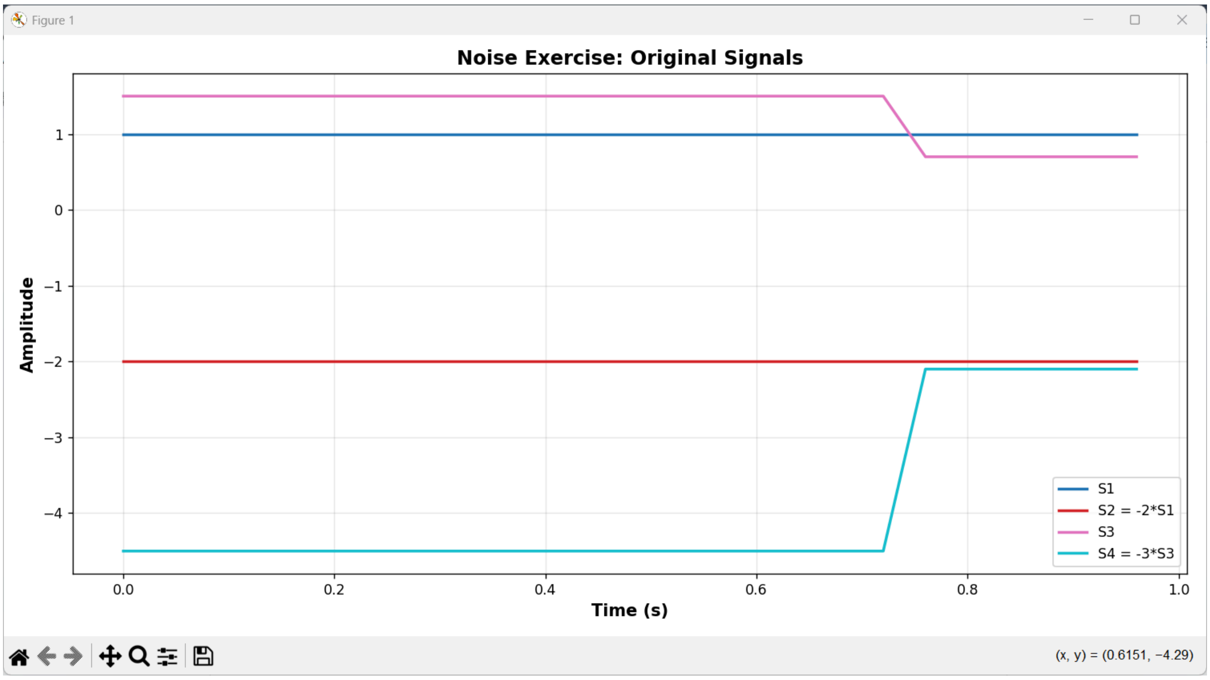
Euclidean Distances and Cross Correlations:

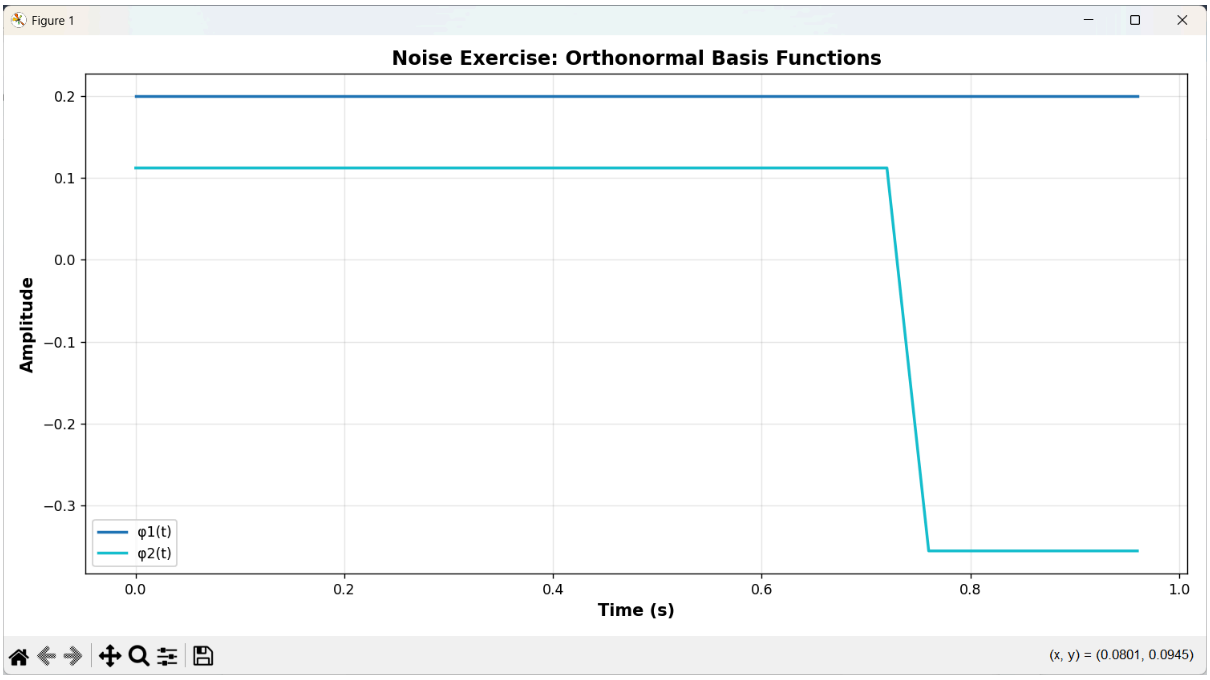
Euclidean Distance Matrix:

	S1	S2	S3	S4
S1	0.0000	2.0000	0.9055	2.7166
S2	2.0000	0.0000	2.7166	0.9055

```
S3   0.9055  2.7166  0.0000  3.5214
S4   2.7166  0.9055  3.5214  0.0000
Cross Correlation Matrix:
       S1     S2     S3     S4
S1   1.0000 -1.0000  0.9315 -0.9315
S2  -1.0000  1.0000 -0.9315  0.9315
S3   0.9315 -0.9315  1.0000 -1.0000
S4  -0.9315  0.9315 -1.0000  1.0000
----------------------------------------------------------------------
Minimum Distance Analysis:
----------------------------------------------------------------------
Minimum Euclidean Distance: 0.905539
Signal pairs with minimum distance:
  Signal 1 <-> Signal 3:
    Distance: 0.905539
    Cross Correlation: 0.931457
  Signal 2 <-> Signal 4:
    Distance: 0.905539
    Cross Correlation: 0.931457
  Signal 3 <-> Signal 1:
    Distance: 0.905539
    Cross Correlation: 0.931457
  Signal 4 <-> Signal 2:
    Distance: 0.905539
    Cross Correlation: 0.931457
======================================================================
```

Noise Exercise:





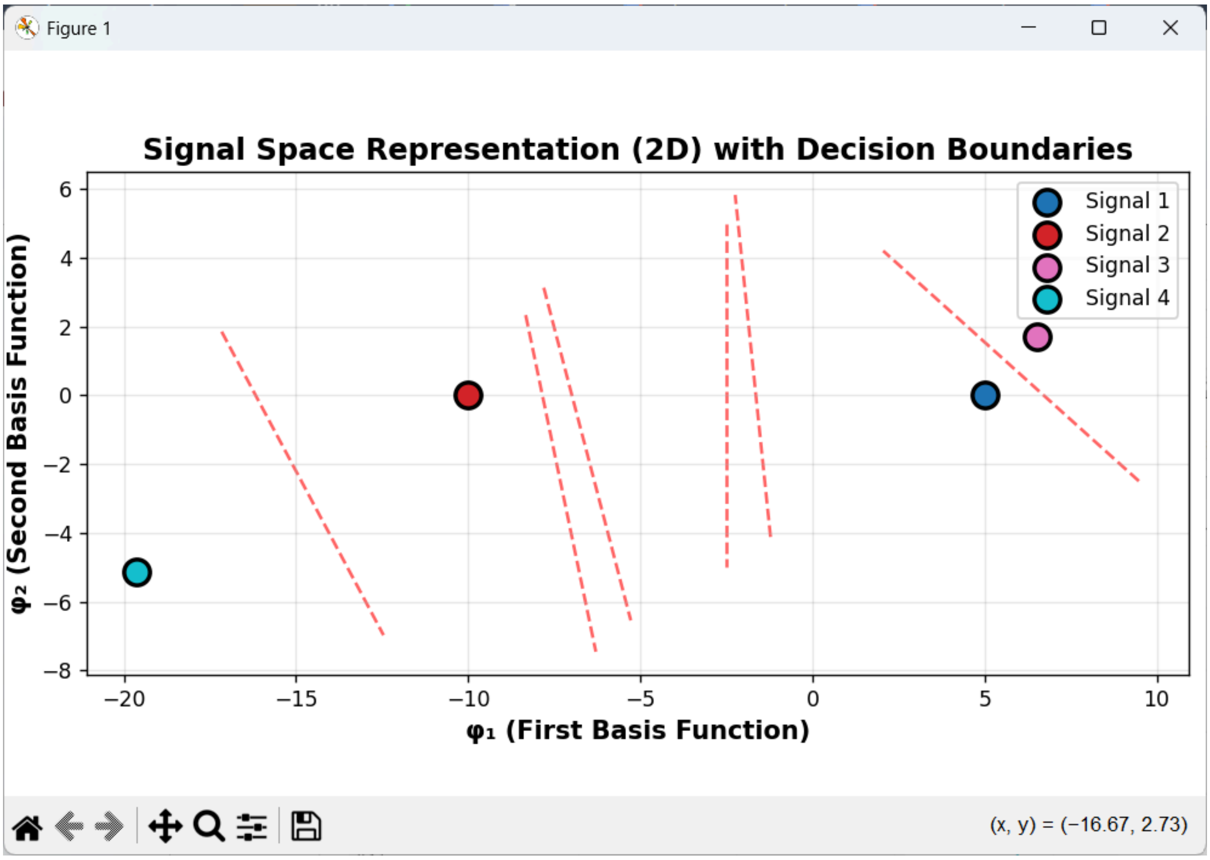
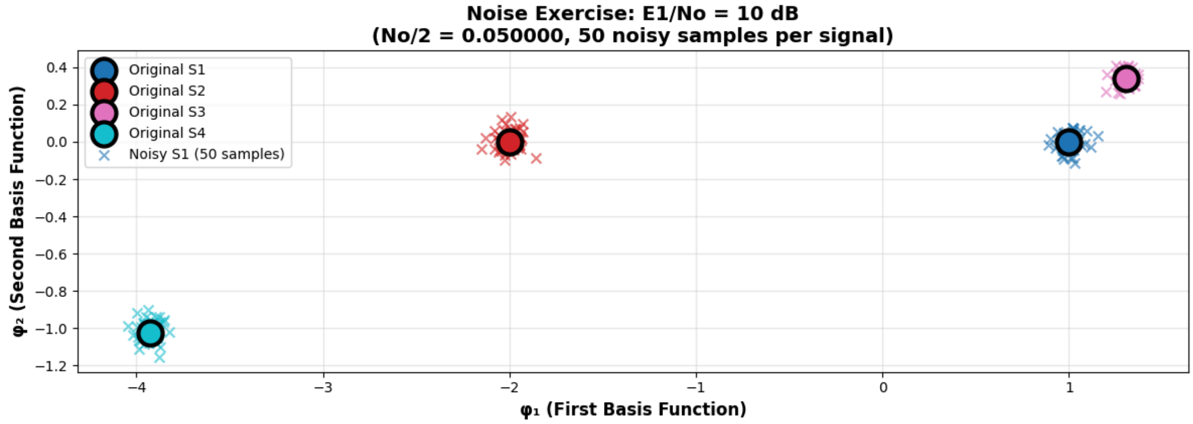


Figure 1
Signal Space Representation (2D) with Decision Boundaries
Signal 1
Signal 2
Signal 3
Signal 4
φ₂ (Second Basis Function)
φ₁ (First Basis Function)
(x, y) = (−16.67, 2.73)

E1/N0 = 10db

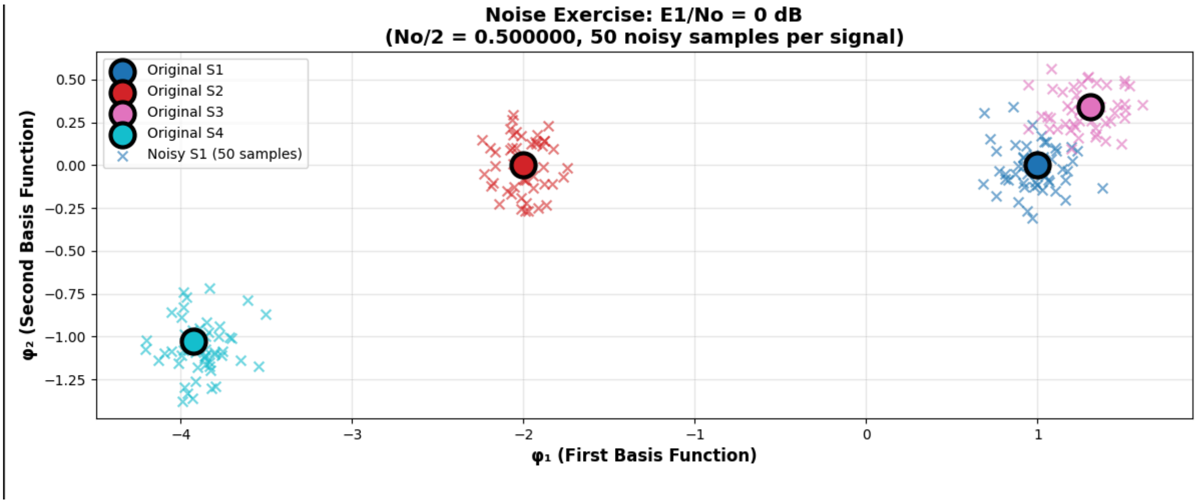


Comment: At 10 dB, the signal is much stronger than the noise. We observed 0 errors out of 200 samples. The noise level (N0 = 0.1) is very low, so the signal samples are far from the decision boundary, resulting in perfect detection.

E1/N0 = 5db

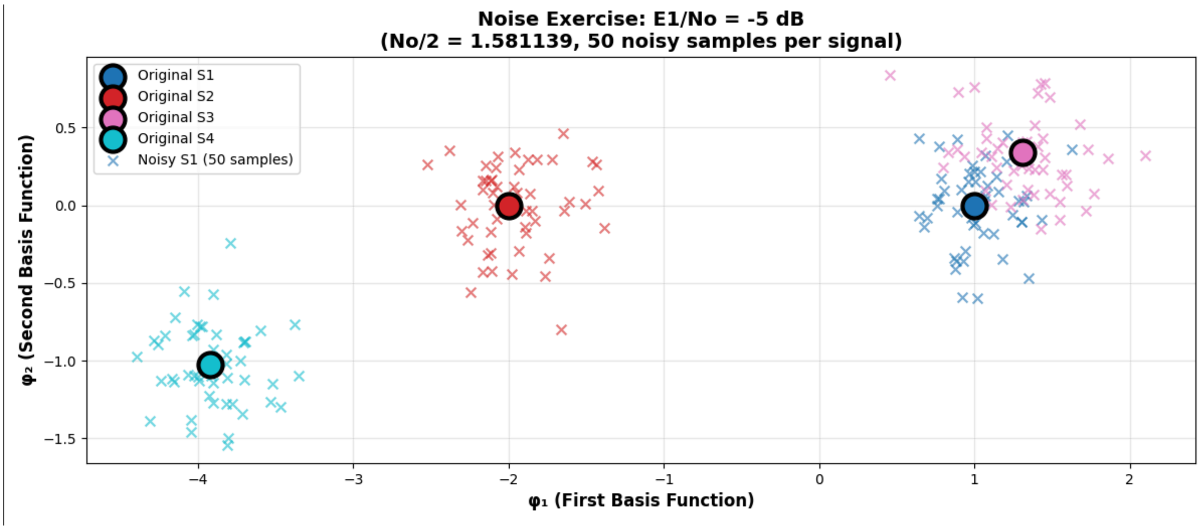
Comment: The error rate is still very low at 0.50% (only 1 error). Even though the noise power increased to roughly 0.32, the signal is still clear. This shows that the system is reliable when the Signal-to-Noise Ratio (SNR) is moderately high.

E1/N0 = 0db



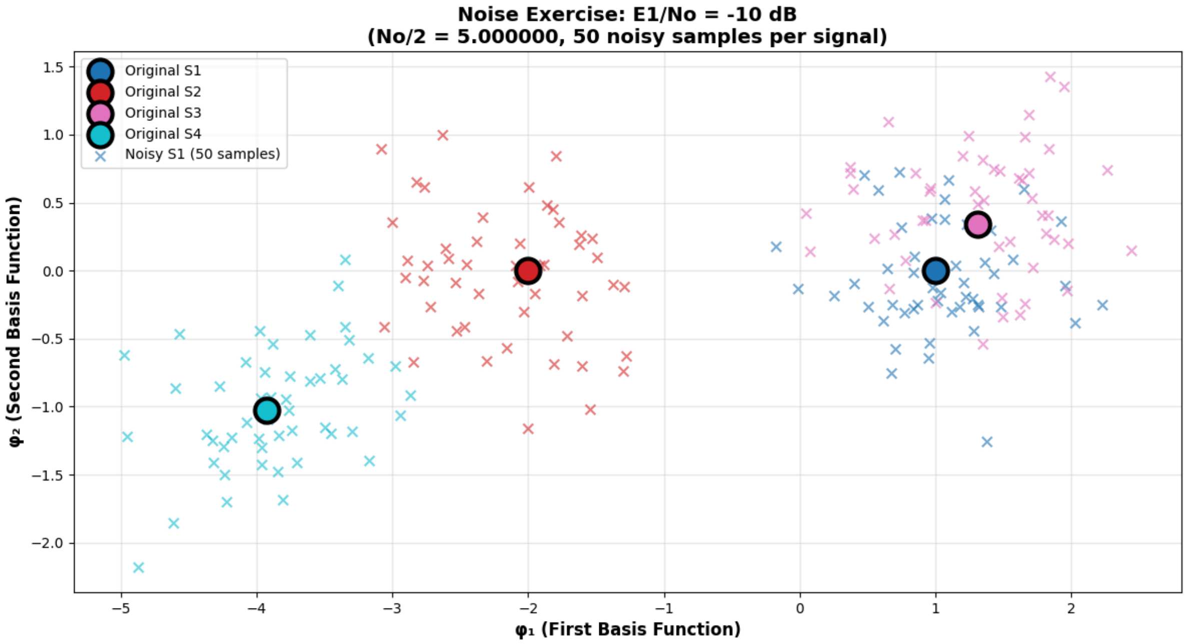
Comment: In this case, the signal energy is equal to the noise power (E1 = N0 = 1). We found 2 errors (1% rate). The noise is strong enough now to cause a small amount of signal overlap, but the system performance is still acceptable.

E1/N0 = -5db



Comment: **There is a significant increase in errors here, reaching 7.50% (15 errors). Since the SNR is negative, the noise power (N0 = 3.16) is stronger than the signal. This causes more samples to cross the threshold and be classified incorrectly.**

E1/N0 = -10db



<u>Comment</u>: This is the worst scenario with 25 errors (12.5% rate). The noise power is ten times larger than the signal (N0 = 10). Because the noise is so high, the signals are heavily distorted, making it difficult to distinguish them correctly.

Code:

The code is able to run on different signals inputted from the user through the terminal, and it also creates a log.txt that stores all the logs.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
from mpl_toolkits.mplot3d import Axes3D
import warnings
warnings.filterwarnings('ignore')
# Set random seed for reproducibility
np.random.seed(42)
ts = 0.04  # 40 ms

def Basis_Cal(Signals, n):
    """
    Part 1.1: Gram-Schmidt Orthogonalization
    """
    n_signals, N = Signals.shape

    basis_functions = []

    s1 = Signals[0, :]
    norm_s1 = np.sqrt(np.sum(s1**2) * ts)

    if norm_s1 < 1e-10:
        pass
    else:
        phi1 = s1 / norm_s1
        basis_functions.append(phi1)

    for i in range(1, n_signals):
        si = Signals[i, :]

        si_orthogonal = si.copy()
        for phi in basis_functions:
            proj_coeff = np.sum(si * phi) * ts
            si_orthogonal = si_orthogonal - proj_coeff * phi

        # Normalize to get new basis function
        norm_si_orth = np.sqrt(np.sum(si_orthogonal**2) * ts)
```

```
        if norm_si_orth > 1e-10:
            phi_new = si_orthogonal / norm_si_orth
            basis_functions.append(phi_new)

    if len(basis_functions) == 0:
        m = 0
        Phis = np.array([]).reshape(0, N)
    else:
        m = len(basis_functions)
        Phis = np.array(basis_functions)

    return Phis, m


def Signal_Rep(Phis, signal):
    """
    Part 1.2: Signal Space Representation

    """
    m, N = Phis.shape

    # Calculate coefficients by projecting signal onto each basis
function
    signal_vector = np.zeros(m)
    for i in range(m):
        signal_vector[i] = np.sum(signal * Phis[i, :]) * ts
    signal_vector[np.abs(signal_vector) < 1e-10] = 0
    return signal_vector


def Decision_boundaries(Phis, Signals):
    """
    Part 1.3: Decision Boundaries

    """
    m, N = Phis.shape
    n, _ = Signals.shape

    if m > 2:
        handle_logs(f"Decision boundaries can only be drawn for m=1 or
m=2. Current m={m}")
        return
```

```
    # Convert signals to signal space representation
    signal_vectors = []
    for i in range(n):
        vec = Signal_Rep(Phis, Signals[i, :])
        signal_vectors.append(vec)
    signal_vectors = np.array(signal_vectors)

    if m == 1:
        fig, ax = plt.subplots(figsize=(10, 2))

        colors = plt.cm.tab10(np.linspace(0, 1, n))
        for i in range(n):
            ax.scatter(signal_vectors[i, 0], 0, s=100, c=[colors[i]],
                      label=f'Signal {i+1}', marker='o',
edgecolors='black', linewidths=1.5)

        if n > 1:
            sorted_indices = np.argsort(signal_vectors[:, 0])
            sorted_vectors = signal_vectors[sorted_indices, 0]

            for i in range(len(sorted_vectors) - 1):
                midpoint = (sorted_vectors[i] + sorted_vectors[i+1]) /
2
                ax.axvline(x=midpoint, color='red', linestyle='--',
linewidth=1.5,
                          alpha=0.7, label='Decision Boundary' if i ==
0 else '')

        ax.set_xlabel('φ₁ (First Basis Function)', fontsize=12,
fontweight='bold')
        ax.set_title('Decision Boundaries',
                    fontsize=14, fontweight='bold')
        ax.legend(fontsize=10)
        ax.grid(True, alpha=0.3)
        ax.set_ylim(-0.5, 0.5)
        plt.tight_layout()
        plt.show()

    elif m == 2:
        fig, ax = plt.subplots(figsize=(10, 8))

        colors = plt.cm.tab10(np.linspace(0, 1, n))
```

```
        for i in range(n):
            ax.scatter(signal_vectors[i, 0], signal_vectors[i, 1],
s=150,
                      c=[colors[i]], label=f'Signal {i+1}', marker='o',
                      edgecolors='black', linewidths=2, zorder=5)

        if n > 1:
            for i in range(n):
                for j in range(i+1, n):
                    mid_x = (signal_vectors[i, 0] + signal_vectors[j,
0]) / 2
                    mid_y = (signal_vectors[i, 1] + signal_vectors[j,
1]) / 2

                    dx = signal_vectors[j, 0] - signal_vectors[i, 0]
                    dy = signal_vectors[j, 1] - signal_vectors[i, 1]

                    perp_dx = -dy
                    perp_dy = dx

                    norm = np.sqrt(perp_dx**2 + perp_dy**2)
                    if norm > 1e-10:
                        perp_dx /= norm
                        perp_dy /= norm

                        scale = 5
                        ax.plot([mid_x - scale*perp_dx, mid_x +
scale*perp_dx],
                               [mid_y - scale*perp_dy, mid_y +
scale*perp_dy],
                               'r--', linewidth=1.5, alpha=0.6,
zorder=1)

        ax.set_xlabel('φ₁ (First Basis Function)', fontsize=12,
fontweight='bold')
        ax.set_ylabel('φ₂ (Second Basis Function)', fontsize=12,
fontweight='bold')
        ax.set_title('Decision Boundaries',
                    fontsize=14, fontweight='bold')
        ax.legend(fontsize=10, loc='best')
        ax.grid(True, alpha=0.3)
        ax.set_aspect('equal', adjustable='box')
        plt.tight_layout()
```

```
        plt.show()

def Signal_Space_Analysis(Phis, Signals):
    """
    Part 1.4: Signal Space Analysis
    """
    n, N = Signals.shape

    # Convert all signals to signal space representation
    signal_vectors = []
    for i in range(n):
        vec = Signal_Rep(Phis, Signals[i, :])
        signal_vectors.append(vec)
    signal_vectors = np.array(signal_vectors)

    # Calculate Euclidean distances and cross correlations
    distances = np.zeros((n, n))
    cross_correlations = np.zeros((n, n))

    min_distance = float('inf')
    min_pairs = []


    handle_logs("="*70)
    handle_logs(f"\nNumber of signals: {n}")
    handle_logs(f"Number of basis functions: {Phis.shape[0]}")
    handle_logs("\n" + "-"*70)
    handle_logs("Euclidean Distances and Cross Correlations:")
    handle_logs("-"*70)

    for i in range(n):
        for j in range(n):
            if i == j:
                distances[i, j] = 0
                cross_correlations[i, j] = 1.0
            else:
                # Euclidean distance in signal space
                dist = np.linalg.norm(signal_vectors[i, :] -
signal_vectors[j, :])
                distances[i, j] = dist

                # Cross correlation (normalized)
                sig_i = Signals[i, :]
```

```
                sig_j = Signals[j, :]
                cross_corr = np.dot(sig_i, sig_j) /
(np.linalg.norm(sig_i) * np.linalg.norm(sig_j))
                cross_correlations[i, j] = cross_corr

                # Track minimum distance
                if dist < min_distance:
                    min_distance = dist
                    min_pairs = [(i, j)]
                elif abs(dist - min_distance) < 1e-10:
                    min_pairs.append((i, j))

    handle_logs("\nEuclidean Distance Matrix:")
    handle_logs("     ", end="")
    for j in range(n):
        handle_logs(f"  S{j+1}  ", end="")
    handle_logs()
    for i in range(n):
        handle_logs(f"S{i+1}  ", end="")
        for j in range(n):
            handle_logs(f"{distances[i, j]:7.4f} ", end="")
        handle_logs()

    handle_logs("\nCross Correlation Matrix:")
    handle_logs("     ", end="")
    for j in range(n):
        handle_logs(f"  S{j+1}  ", end="")
    handle_logs()
    for i in range(n):
        handle_logs(f"S{i+1}  ", end="")
        for j in range(n):
            handle_logs(f"{cross_correlations[i, j]:7.4f} ", end="")
        handle_logs()

    handle_logs("\n" + "-"*70)
    handle_logs("Minimum Distance Analysis:")
    handle_logs("-"*70)
    handle_logs(f"Minimum Euclidean Distance: {min_distance:.6f}")
    handle_logs(f"\nSignal pairs with minimum distance:")
    for pair in min_pairs:
        i, j = pair
        handle_logs(f"  Signal {i+1} <-> Signal {j+1}:")
        handle_logs(f"    Distance: {distances[i, j]:.6f}")
```

```
        handle_logs(f"    Cross Correlation: {cross_correlations[i,
j]:.6f}")

    handle_logs("="*70 + "\n")

    return distances, cross_correlations


def AWGN_Signal_Space(Phis, Signals, No_over_2, plot_title="AWGN in
Signal Space"):
    """
    Part 1.5: AWGN in Signal Space

    """
    m, N = Phis.shape
    n, _ = Signals.shape

    if m > 3:
        handle_logs(f"3D plotting not supported for m > 3. Current
m={m}")
        return

    # Convert original signals to signal space
    original_vectors = []
    for i in range(n):
        vec = Signal_Rep(Phis, Signals[i, :])
        original_vectors.append(vec)
    original_vectors = np.array(original_vectors)

    # Add AWGN noise to signals
    noisy_signals = Signals.copy()
    for i in range(n):
        noise = np.random.normal(0, np.sqrt(No_over_2), N)
        noisy_signals[i, :] = Signals[i, :] + noise

    # Convert noisy signals to signal space
    noisy_vectors = []
    for i in range(n):
        vec = Signal_Rep(Phis, noisy_signals[i, :])
        noisy_vectors.append(vec)
    noisy_vectors = np.array(noisy_vectors)

    colors = plt.cm.tab10(np.linspace(0, 1, n))
```

```
    if m == 1:
        fig, ax = plt.subplots(figsize=(10, 6))

        for i in range(n):
            ax.scatter(original_vectors[i, 0], 0, s=200, c=[colors[i]],
                      label=f'Original Signal {i+1}', marker='o',
                      edgecolors='black', linewidths=2, zorder=5)

        for i in range(n):
            ax.scatter(noisy_vectors[i, 0], 0, s=100, c=[colors[i]],
                      marker='x', linewidths=2, alpha=0.7, zorder=3,
                      label=f'Noisy Signal {i+1}' if i == 0 else '')

        ax.set_xlabel('φ₁ (First Basis Function)', fontsize=12,
fontweight='bold')
        ax.set_title(f'{plot_title}\n(Noise Variance: No/2 =
{No_over_2:.4f})',
                    fontsize=14, fontweight='bold')
        ax.legend(fontsize=10)
        ax.grid(True, alpha=0.3)
        ax.set_ylim(-0.5, 0.5)

    elif m == 2:
        fig, ax = plt.subplots(figsize=(10, 8))

        for i in range(n):
            ax.scatter(original_vectors[i, 0], original_vectors[i, 1],
s=200,
                      c=[colors[i]], label=f'Original Signal {i+1}',
marker='o',
                      edgecolors='black', linewidths=2, zorder=5)

        for i in range(n):
            ax.scatter(noisy_vectors[i, 0], noisy_vectors[i, 1], s=100,
                      c=[colors[i]], marker='x', linewidths=2,
alpha=0.7, zorder=3,
                      label=f'Noisy Signal {i+1}' if i == 0 else '')

        ax.set_xlabel('φ₁ (First Basis Function)', fontsize=12,
fontweight='bold')
        ax.set_ylabel('φ₂ (Second Basis Function)', fontsize=12,
fontweight='bold')
```

```
        ax.set_title(f'{plot_title}\n(Noise Variance: No/2 =
{No_over_2:.4f})',
                        fontsize=14, fontweight='bold')
        ax.legend(fontsize=10, loc='best')
        ax.grid(True, alpha=0.3)
        ax.set_aspect('equal', adjustable='box')

    elif m == 3:
        fig = plt.figure(figsize=(12, 10))
        ax = fig.add_subplot(111, projection='3d')

        for i in range(n):
            ax.scatter(original_vectors[i, 0], original_vectors[i, 1],
                        original_vectors[i, 2], s=200, c=[colors[i]],
                        label=f'Original Signal {i+1}', marker='o',
                        edgecolors='black', linewidths=2)

        for i in range(n):
            ax.scatter(noisy_vectors[i, 0], noisy_vectors[i, 1],
                        noisy_vectors[i, 2], s=100, c=[colors[i]],
                        marker='x', linewidths=2, alpha=0.7,
                        label=f'Noisy Signal {i+1}' if i == 0 else '')

        ax.set_xlabel('φ₁', fontsize=12, fontweight='bold')
        ax.set_ylabel('φ₂', fontsize=12, fontweight='bold')
        ax.set_zlabel('φ₃', fontsize=12, fontweight='bold')
        ax.set_title(f'{plot_title}\n(Noise Variance: No/2 =
{No_over_2:.4f})',
                        fontsize=14, fontweight='bold')
        ax.legend(fontsize=10)

    plt.tight_layout()
    plt.show()

    return noisy_signals, noisy_vectors


def create_signal_from_piecewise(t, time_ranges, values):
    """
    Function to create signals from piecewise definitions.
    """
    signal = np.zeros_like(t)
    for (t_start, t_end), value in zip(time_ranges, values):
```

```
        mask = (t >= t_start) & (t < t_end)
        signal[mask] = value
    return signal


def plot_signals(t, Signals, title="Signals", labels=None):
    """
    Function to plot signals in time domain.
    """
    n, _ = Signals.shape
    fig, ax = plt.subplots(figsize=(12, 6))

    colors = plt.cm.tab10(np.linspace(0, 1, n))
    for i in range(n):
        label = labels[i] if labels else f'Signal {i+1}'
        ax.plot(t, Signals[i, :], linewidth=2, color=colors[i],
label=label)

    ax.set_xlabel('Time (s)', fontsize=12, fontweight='bold')
    ax.set_ylabel('Amplitude', fontsize=12, fontweight='bold')
    ax.set_title(title, fontsize=14, fontweight='bold')
    ax.legend(fontsize=10)
    ax.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()


def plot_basis_functions(t, Phis, title="Basis Functions"):
    """
    Function to plot basis functions.
    """
    m, N = Phis.shape
    fig, ax = plt.subplots(figsize=(12, 6))

    colors = plt.cm.tab10(np.linspace(0, 1, m))
    for i in range(m):
        ax.plot(t, Phis[i, :], linewidth=2, color=colors[i],
                label=f'φ{i+1}(t)', linestyle='-')

    ax.set_xlabel('Time (s)', fontsize=12, fontweight='bold')
    ax.set_ylabel('Amplitude', fontsize=12, fontweight='bold')
    ax.set_title(title, fontsize=14, fontweight='bold')
    ax.legend(fontsize=10)
```

```python
    ax.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()

def handle_logs(*args, end='\n', sep=' '):
    message = sep.join(str(arg) for arg in args) if args else ''
    print(message, end=end)
    try:
        with open('logs.txt', 'a') as f:
            f.write(message + (end if end != '\n' else '\n'))
    except:
        pass


def solve_problem_1():
    handle_logs("\n" + "="*80)
    handle_logs("PROBLEM 1")
    handle_logs("="*80)

    t = np.arange(0, 1, ts)
    N = len(t)

    S1 = create_signal_from_piecewise(t, [(0, 0.75), (0.75, 1)], [-3,
0.7])
    S2 = create_signal_from_piecewise(t, [(0, 0.75), (0.75, 1)], [7.5,
-1.75])

    Signals = np.array([S1, S2])
    n = 2

    plot_signals(t, Signals, "Original Signals",
                 labels=['S1', 'S2'])

    Phis, m = Basis_Cal(Signals, n)
    handle_logs(f"\nNumber of basis functions: m = {m}")
    handle_logs(f"Basis functions shape: {Phis.shape}")

    plot_basis_functions(t, Phis, "Orthonormal Basis Functions")

    handle_logs("\nSignal Space Coefficients:")
    for i in range(n):
        vec = Signal_Rep(Phis, Signals[i, :])
        handle_logs(f"Signal {i+1} (S{i+1}): {vec}")
```

```
    if m <= 2:
        Decision_boundaries(Phis, Signals)

    distances, cross_correlations = Signal_Space_Analysis(Phis,
Signals)
    return Phis, Signals, t


def solve_problem_2():
    handle_logs("\n" + "="*80)
    handle_logs("PROBLEM 2")
    handle_logs("="*80)

    t = np.arange(0, 1, ts)
    N = len(t)

    S1 = create_signal_from_piecewise(t, [(0, 1)], [1])
    S2 = create_signal_from_piecewise(t, [(0, 0.75), (0.75, 1)], [1,
-1])

    Signals = np.array([S1, S2])
    n = 2

    plot_signals(t, Signals, "Original Signals",
                 labels=['S1', 'S2'])

    Phis, m = Basis_Cal(Signals, n)
    handle_logs(f"\nNumber of basis functions: m = {m}")
    handle_logs(f"Basis functions shape: {Phis.shape}")

    plot_basis_functions(t, Phis, "Orthonormal Basis Functions")
    handle_logs("\nSignal Space Coefficients:")
    for i in range(n):
        vec = Signal_Rep(Phis, Signals[i, :])
        handle_logs(f"Signal {i+1} (S{i+1}): {vec}")

    if m <= 2:
        Decision_boundaries(Phis, Signals)
    distances, cross_correlations = Signal_Space_Analysis(Phis,
Signals)
    return Phis, Signals, t
```

```python
def solve_problem_3():
    handle_logs("\n" + "="*80)
    handle_logs("PROBLEM 3")
    handle_logs("="*80)

    t = np.arange(0, 1, ts)
    N = len(t)
    S1 = create_signal_from_piecewise(t, [(0, 1)], [1])
    S2 = -S1
    S3 = create_signal_from_piecewise(t, [(0, 0.75), (0.75, 1)], [2,
0.5])
    S4 = -S3

    Signals = np.array([S1, S2, S3, S4])
    n = 4

    plot_signals(t, Signals, "Original Signals",
                 labels=['S1', 'S2 = -S1', 'S3', 'S4 = -S3'])

    Phis, m = Basis_Cal(Signals, n)
    handle_logs(f"\nNumber of basis functions: m = {m}")
    handle_logs(f"Basis functions shape: {Phis.shape}")

    plot_basis_functions(t, Phis, "Orthonormal Basis Functions")

    handle_logs("\nSignal Space Coefficients:")
    for i in range(n):
        vec = Signal_Rep(Phis, Signals[i, :])
        handle_logs(f"Signal {i+1} (S{i+1}): {vec}")

    if m <= 2:
        Decision_boundaries(Phis, Signals)
    distances, cross_correlations = Signal_Space_Analysis(Phis,
Signals)
    return Phis, Signals, t


def noise_exercise():
    handle_logs("\n" + "="*80)
    handle_logs("NOISE EXERCISE")
    handle_logs("="*80)
```

```
    t = np.arange(0, 1, ts)
    N = len(t)

    S1 = create_signal_from_piecewise(t, [(0, 1)], [1])
    S2 = -2 * S1
    S3 = create_signal_from_piecewise(t, [(0, 0.75), (0.75, 1)], [1.5,
0.7])
    S4 = -3 * S3

    Signals = np.array([S1, S2, S3, S4])
    n = 4

    plot_signals(t, Signals, "Noise Exercise: Original Signals",
                 labels=['S1', 'S2 = -2*S1', 'S3', 'S4 = -3*S3'])
    Phis, m = Basis_Cal(Signals, n)
    handle_logs(f"\nNumber of basis functions: m = {m}")
    handle_logs(f"Basis functions shape: {Phis.shape}")

    plot_basis_functions(t, Phis, "Noise Exercise: Orthonormal Basis
Functions")

    handle_logs("\nSignal Space Coefficients:")
    signal_vectors = []
    for i in range(n):
        vec = Signal_Rep(Phis, Signals[i, :])
        signal_vectors.append(vec)
        handle_logs(f"Signal {i+1} (S{i+1}): {vec}")
    signal_vectors = np.array(signal_vectors)

    if m <= 2:
        Decision_boundaries(Phis, Signals)

    distances, cross_correlations = Signal_Space_Analysis(Phis,
Signals)

    E1 = np.sum(S1**2) * ts

    E1_No_dB_values = [10, 5, 0, -5, -10]
    colors = plt.cm.tab10(np.linspace(0, 1, n))
    num_samples = 50

    for E1_No_dB in E1_No_dB_values:
        E1_No_linear = 10**(E1_No_dB / 10)
```

```
        No = E1 / E1_No_linear
        No_over_2 = No / 2

        handle_logs(f"\nE1/No = {E1_No_dB} dB")
        handle_logs(f"  E1 = {E1:.6f}")
        handle_logs(f"  E1/No (linear) = {E1_No_linear:.6f}")
        handle_logs(f"  No = {No:.6f}")
        handle_logs(f"  No/2 = {No_over_2:.6f}")

        # Generate 50 noisy samples for each signal
        all_noisy_vectors = []
        for i in range(n):
            signal_noisy_vectors = []
            for sample in range(num_samples):
                # Add AWGN noise
                noise = np.random.normal(0, np.sqrt(No_over_2), N)
                noisy_signal = Signals[i, :] + noise
                # Part 1.2: Convert to signal space
                noisy_vec = Signal_Rep(Phis, noisy_signal)
                signal_noisy_vectors.append(noisy_vec)
            all_noisy_vectors.append(np.array(signal_noisy_vectors))

        # Plot
        if m == 1:
            fig, ax = plt.subplots(figsize=(12, 6))

            # Plot original signals (circles)
            for i in range(n):
                ax.scatter(signal_vectors[i, 0], 0, s=300,
c=[colors[i]],
                          label=f'Original S{i+1}', marker='o',
                          edgecolors='black', linewidths=3, zorder=10)

            # Plot noisy samples (crosses)
            for i in range(n):
                ax.scatter(all_noisy_vectors[i][:, 0],
                          np.zeros(num_samples), s=50, c=[colors[i]],
                          marker='x', linewidths=1.5, alpha=0.6,
zorder=5,
                          label=f'Noisy S{i+1} (50 samples)' if i == 0
else '')
```

```
            ax.set_xlabel('φ₁ (First Basis Function)', fontsize=12, fontweight='bold')
            ax.set_title(f'Noise Exercise: E1/No = {E1_No_dB} dB\n' +
                        f'(No/2 = {No_over_2:.6f}, 50 noisy samples per signal)',
                        fontsize=14, fontweight='bold')
            ax.legend(fontsize=10)
            ax.grid(True, alpha=0.3)
            ax.set_ylim(-0.5, 0.5)

        elif m == 2:
            fig, ax = plt.subplots(figsize=(12, 10))

            # Plot original signals (circles)
            for i in range(n):
                ax.scatter(signal_vectors[i, 0], signal_vectors[i, 1],
                           s=300, c=[colors[i]], label=f'Original S{i+1}',
                           marker='o', edgecolors='black', linewidths=3, zorder=10)

            # Plot noisy samples (crosses)
            for i in range(n):
                ax.scatter(all_noisy_vectors[i][:, 0],
                           all_noisy_vectors[i][:, 1], s=50, c=[colors[i]],
                           marker='x', linewidths=1.5, alpha=0.6, zorder=5,
                           label=f'Noisy S{i+1} (50 samples)' if i == 0 else '')

            ax.set_xlabel('φ₁ (First Basis Function)', fontsize=12, fontweight='bold')
            ax.set_ylabel('φ₂ (Second Basis Function)', fontsize=12, fontweight='bold')
            ax.set_title(f'Noise Exercise: E1/No = {E1_No_dB} dB\n' +
                        f'(No/2 = {No_over_2:.6f}, 50 noisy samples per signal)',
                        fontsize=14, fontweight='bold')
            ax.legend(fontsize=10, loc='best')
            ax.grid(True, alpha=0.3)
            ax.set_aspect('equal', adjustable='box')
```

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


        error_count = 0
        for i in range(n):
            for noisy_vec in all_noisy_vectors[i]:
                # Find closest original signal
                distances_to_originals = [np.linalg.norm(noisy_vec -
signal_vectors[j, :])
                                          for j in range(n)]
                closest_idx = np.argmin(distances_to_originals)
                if closest_idx != i:
                    error_count += 1

        total_samples = n * num_samples
        error_rate = error_count / total_samples * 100

        handle_logs(f"  Error Analysis:")
        handle_logs(f"    Total samples: {total_samples}")
        handle_logs(f"    Errors (misclassified): {error_count}")
        handle_logs(f"    Error rate: {error_rate:.2f}%")

        # Add comment about noise effect
        if E1_No_dB >= 10:
            comment = "High SNR: Noise has minimal effect. Very few
errors occur. Signals are clearly distinguishable."
        elif E1_No_dB >= 5:
            comment = "Moderate-high SNR: Some noise spread visible.
Low error rate. Signals mostly distinguishable."
        elif E1_No_dB >= 0:
            comment = "Moderate SNR: Noticeable noise spread. Moderate
error rate. Some signal overlap."
        elif E1_No_dB >= -5:
            comment = "Low SNR: Significant noise spread. High error
rate. Considerable signal overlap."
        else:
            comment = "Very low SNR: Severe noise spread. Very high
error rate. Signals are barely distinguishable."

        handle_logs(f"  Comment: {comment}")

    return Phis, Signals, t
```

```python
def run_custom_signals():
    """
    Interactive function to create and analyze custom signals.
    Users can define signals piecewise with time intervals and values.
    """
    handle_logs("\n" + "="*80)
    handle_logs("CUSTOM SIGNAL ANALYSIS")
    handle_logs("="*80)

    handle_logs("\n" + "-"*80)
    handle_logs("INSTRUCTIONS:")
    handle_logs("  - Define signals using piecewise constant
functions")
    handle_logs("  - Each signal can have multiple time segments with
constant values")
    handle_logs("  - You can also define signals as multiples/negatives
of previous signals")
    handle_logs("  - Example: Signal 1: value=1 from t=0 to t=0.5,
value=-1 from t=0.5 to t=1")
    handle_logs("-"*80)

    try:
        # Get time range
        handle_logs("\nEnter time parameters:")
        t_start = float(input("  Start time (default 0): ") or "0")
        t_end = float(input("  End time (default 1): ") or "1")
        t = np.arange(t_start, t_end, ts)
        N = len(t)

        # Get number of signals
        num_signals = int(input("\nEnter number of signals: "))
        if num_signals < 1:
            handle_logs("Error: Number of signals must be at least 1")
            return None, None, None

        Signals_list = []
        labels_list = []

        # Get each signal definition
        for i in range(num_signals):
            handle_logs(f"\n--- Signal {i+1} ---")
```

```
            signal_name = input(f"  Signal name (default S{i+1}): ") or f"S{i+1}"
            labels_list.append(signal_name)

            # Check if signal is a multiple/negative of another
            use_relation = 'n'
            if i > 0:  # Can only reference previous signals
                use_relation = input("  Is this signal related to another? (y/n, default n): ").lower()

            if use_relation == 'y':
                ref_idx = int(input(f"  Reference signal index (1-{i}): ")) - 1
                if ref_idx < 0 or ref_idx >= len(Signals_list):
                    handle_logs(f"Error: Invalid reference signal index. Must be between 1 and {i}")
                    return None, None, None

                relation = input("  Relation (e.g., '-2*' for -2*S1, or '-' for -S1, default '-'): ") or "-"

                if relation == "-":
                    signal = -Signals_list[ref_idx]
                elif relation.endswith("*"):
                    multiplier = float(relation[:-1])
                    signal = multiplier * Signals_list[ref_idx]
                else:
                    handle_logs(f"Error: Invalid relation format")
                    return None, None, None
            else:
                # Get piecewise definition
                num_segments = int(input("  Number of time segments: "))
                time_ranges = []
                values = []

                for seg in range(num_segments):
                    handle_logs(f"  Segment {seg+1}:")
                    seg_start = float(input(f"    Start time: "))
                    seg_end = float(input(f"    End time: "))
                    seg_value = float(input(f"    Value: "))
                    time_ranges.append((seg_start, seg_end))
                    values.append(seg_value)
```

```
                signal = create_signal_from_piecewise(t, time_ranges,
values)

            Signals_list.append(signal)

        Signals = np.array(Signals_list)
        n = num_signals

        # Plot original signals
        plot_signals(t, Signals, "Custom Signals: Original Signals",
labels=labels_list)

        # Part 1.1: Calculate basis functions
        Phis, m = Basis_Cal(Signals, n)
        handle_logs(f"\nNumber of basis functions: m = {m}")
        handle_logs(f"Basis functions shape: {Phis.shape}")

        # Plot basis functions
        plot_basis_functions(t, Phis, "Custom Signals: Orthonormal
Basis Functions")

        # Part 1.2: Signal space representation
        handle_logs("\nSignal Space Coefficients:")
        for i in range(n):
            vec = Signal_Rep(Phis, Signals[i, :])
            handle_logs(f"{labels_list[i]}: {vec}")

        # Part 1.3: Decision boundaries
        if m <= 2:
            Decision_boundaries(Phis, Signals)

        # Part 1.4: Signal space analysis
        distances, cross_correlations = Signal_Space_Analysis(Phis,
Signals)

        return Phis, Signals, t

    except ValueError as e:
        handle_logs(f"Error: Invalid input - {e}")
        return None, None, None
    except Exception as e:
        handle_logs(f"Error: {e}")
```

```
        return None, None, None


def show_menu():
    """
    Display the main menu and handle user selection.
    """
    handle_logs("\n" + "="*80)
    handle_logs("SIGNAL SPACE ANALYSIS - MAIN MENU")
    handle_logs("="*80)
    handle_logs("\nSelect an option:")
    handle_logs("  1. Run Problem 1 (Assignment Example)")
    handle_logs("  2. Run Problem 2 (Assignment Example)")
    handle_logs("  3. Run Problem 3 (Assignment Example)")
    handle_logs("  4. Run Noise Exercise (Assignment Example)")
    handle_logs("  5. Run All Assignment Examples")
    handle_logs("  6. Input Custom Signals")
    handle_logs("  7. Exit")
    handle_logs("\n" + "-"*80)

    choice = input("Enter your choice (1-7): ").strip()
    return choice


if __name__ == "__main__":
    # Clear logs file at start
    with open('logs.txt', 'w') as f:
        f.write("")

    while True:
        choice = show_menu()

        if choice == '1':
            Phis1, Signals1, t1 = solve_problem_1()
            input("\nPress Enter to continue...")

        elif choice == '2':
            Phis2, Signals2, t2 = solve_problem_2()
            input("\nPress Enter to continue...")

        elif choice == '3':
            Phis3, Signals3, t3 = solve_problem_3()
            input("\nPress Enter to continue...")
```

```
        elif choice == '4':
            Phis_noise, Signals_noise, t_noise = noise_exercise()
            input("\nPress Enter to continue...")

        elif choice == '5':
            # Run all assignment examples
            Phis1, Signals1, t1 = solve_problem_1()
            Phis2, Signals2, t2 = solve_problem_2()
            Phis3, Signals3, t3 = solve_problem_3()
            Phis_noise, Signals_noise, t_noise = noise_exercise()

            handle_logs("\n" + "="*80)
            handle_logs("ALL ASSIGNMENT EXAMPLES COMPLETED
SUCCESSFULLY!")
            handle_logs("="*80)
            input("\nPress Enter to continue...")

        elif choice == '6':
            Phis_custom, Signals_custom, t_custom =
run_custom_signals()
            if Phis_custom is not None:
                handle_logs("\nCustom signal analysis completed
successfully!")
            input("\nPress Enter to continue...")

        elif choice == '7':
            handle_logs("\n" + "="*80)
            handle_logs("BYE :)")
            handle_logs("="*80)
            break

        else:
            handle_logs("\nInvalid choice! Please enter a number
between 1 and 7.")
            input("Press Enter to continue...")
```