EE3700/CC5901

Practical 1 – Crash Course on Matlab/Python and Pulse Signal Transmission.

Dr Tao (Kevin) Huang email: tao.huang1@jcu.edu.au

Contents

Part A – Crash course on Matlab/Python

In this practical session, you will be introduced to essential programming concepts in Matlab and Python, two powerful tools widely used in engineering and scientific research. This session is divided into several parts, each focusing on different aspects of programming and signal processing. You will learn how to navigate the Matlab and Python environments, work with arrays and matrices, define functions and control flow, generate and analyze signals, and finally, generate Additive White Gaussian Noise (AWGN) with both nominal and specified power levels. This comprehensive introduction will equip you with the skills needed for more advanced topics in communications and signal processing.

1. Introduction to Matlab/Python

Objective: Familiarize with the Matlab/Python environment and basic syntax.

Matlab

Overview of Matlab interface:

- Command Window: Where you enter commands.
- Workspace: Displays variables created during the session.
- Command History: Shows a history of commands you have entered.
- Editor: Used to create and edit scripts and functions.

Basic Commands:

```
clc;        % Clears the command window
clear;      % Clears all variables from the workspace
disp('Hello, Matlab!'); % Displays a message
```

Basic Arithmetic Operations and Functions:

```
a = 2 + 3;   % Addition
b = 5 - 2;   % Subtraction
c = 4 * 3;   % Multiplication
d = 8 / 2;   % Division
e = sqrt(16); % Square root
f = log(10); % Natural logarithm
```

Python

Overview of Python environment:

- Jupyter Notebook, Spyder, or any preferred IDE.
- Installation of libraries: pip install numpy matplotlib

Basic Commands:

```
print('Hello, Python!')  # Prints a message
# This is a comment
```

Variable Declaration and Types:

```
x = 5       # Integer
```

```
y = 3.14    # Floating-point
z = 'Hello' # String
```

Basic Arithmetic Operations and Functions:

```
a = 2 + 3   # Addition
b = 5 - 2   # Subtraction
c = 4 * 3   # Multiplication
d = 8 / 2   # Division
import math
e = math.sqrt(16) # Square root
f = math.log(10)  # Natural logarithm
```

2. Working with Arrays and Matrices

Objective: Learn how to manipulate arrays and matrices.

Matlab

Creating Arrays and Matrices:

```
A = [1, 2, 3; 4, 5, 6];     % Matrix
B = zeros(3,3);             % 3x3 matrix of zeros
C = ones(2,4);              % 2x4 matrix of ones
D = eye(3);                 % 3x3 identity matrix
```

Accessing and Modifying Elements:

```
A(1,2) = 10;                % Modify element
row = A(2, :);              % Access second row
col = A(:, 3);              % Access third column
```

Basic Matrix Operations:

```
A_trans = A';               % Transpose
A_inv = inv(A);             % Inverse
A_mult = A * B;             % Multiplication
```

Plotting:

```
x = 0:0.1:10;
y = sin(x);
plot(x, y);
xlabel('x');
ylabel('sin(x)');
title('Sine Wave');
```

Python

Importing Necessary Libraries:

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

Creating Arrays and Matrices:

```
A = np.array([[1, 2, 3], [4, 5, 6]])   # Matrix
B = np.zeros((3, 3))                   # 3x3 matrix of zeros
```

```
C = np.ones((2, 4))              # 2x4 matrix of ones
D = np.eye(3)                    # 3x3 identity matrix
```

Accessing and Modifying Elements:

```
A[0, 1] = 10                     # Modify element
row = A[1, :]                    # Access second row
col = A[:, 2]                    # Access third column
```

Basic Matrix Operations:

```
A_trans = A.T                    # Transpose
A_inv = np.linalg.inv(A)         # Inverse
A_mult = np.dot(A, B)            # Multiplication
```

Plotting

```
x = np.arange(0, 10, 0.1)
y = np.sin(x)
plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('sin(x)')
plt.title('Sine Wave')
plt.show()
```

3. Functions and Loops

Objective: Introduce functions and control flow.

Matlab

Defining Functions:

```
function result = factorial(n)
    result = 1;
    for i = 1:n
        result = result * i;
    end
end
```

For Loops:

```
for i = 1:5
    disp(i);
end
```

If-Else Statements:

```
x = 10;
if x > 0
    disp('Positive');
elseif x < 0
    disp('Negative');
```

```
else
   disp('Zero');
end
```

Python

Defining Functions:

```
def factorial(n):
    result = 1
    for i in range(1, n + 1):
       result *= i
    return result
```

For Loops:

```
for i in range(1, 6):
    print(i)
```

If-Else Statements:

```
x = 10
if x > 0:
    print('Positive')
elif x < 0:
    print('Negative')
else:
    print('Zero')
```

4. Signal Generation and Basic Analysis

Objective: Apply programming skills to generate and analyze signals.

Matlab

Generating Signals:

```
t = 0:0.01:1;
signal = sin(2 * pi * 10 * t);
```

Sampling and Quantization:

```
fs = 100; % Sampling frequency
ts = 1/fs;
n = 0:ts:1;
sampled_signal = sin(2 * pi * 10 * n);
```

Fourier Transform:

```
fft_signal = fft(signal);
f = (0:length(fft_signal)-1)*fs/length(fft_signal);
plot(f, abs(fft_signal));
```

Python

Generating Signals:

```
t = np.arange(0, 1, 0.01)
signal = np.sin(2 * np.pi * 10 * t)
```

Sampling and Quantization:

```
fs = 100  # Sampling frequency
ts = 1/fs
n = np.arange(0, 1, ts)
sampled_signal = np.sin(2 * np.pi * 10 * n)
```

Fourier Transform:

```
fft_signal = np.fft.fft(signal)
f = np.fft.fftfreq(len(fft_signal), ts)
plt.plot(f, np.abs(fft_signal))
plt.show()
```

5. Generating AWGN Noises

Objective: Teach students how to generate Additive White Gaussian Noise (AWGN) with nominal and specified power levels.

Matlab

Generating AWGN with Nominal Power:

```
N = 1000;                   % Number of samples
noise = randn(1, N);        % Generate AWGN
```

Generating AWGN with Specified Power:

```
specified_power = 0.5;      % Specified power – Can be any number
noise_power = var(noise);   % Compute the current power
scaling_factor = sqrt(specified_power / noise_power);
scaled_noise = noise * scaling_factor; % Scale the noise
```

Visualizing the Noise:

```
figure;
subplot(2,1,1);
plot(noise);
title('AWGN with Nominal Power');
xlabel('Sample');
ylabel('Amplitude');

subplot(2,1,2);
```

```
plot(scaled_noise);
title('AWGN with Specified Power');
xlabel('Sample');
ylabel('Amplitude');
```

Python

Generating AWGN with Nominal Power:

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

N = 1000                    # Number of samples
noise = np.random.randn(N)      # Generate AWGN
```

Generating AWGN with Specified Power:

```
specified_power = 0.5          # Specified power – Can be any number
noise_power = np.var(noise)      # Compute the current power
scaling_factor = np.sqrt(specified_power / noise_power)
scaled_noise = noise * scaling_factor  # Scale the noise
```

Visualizing the Noise:

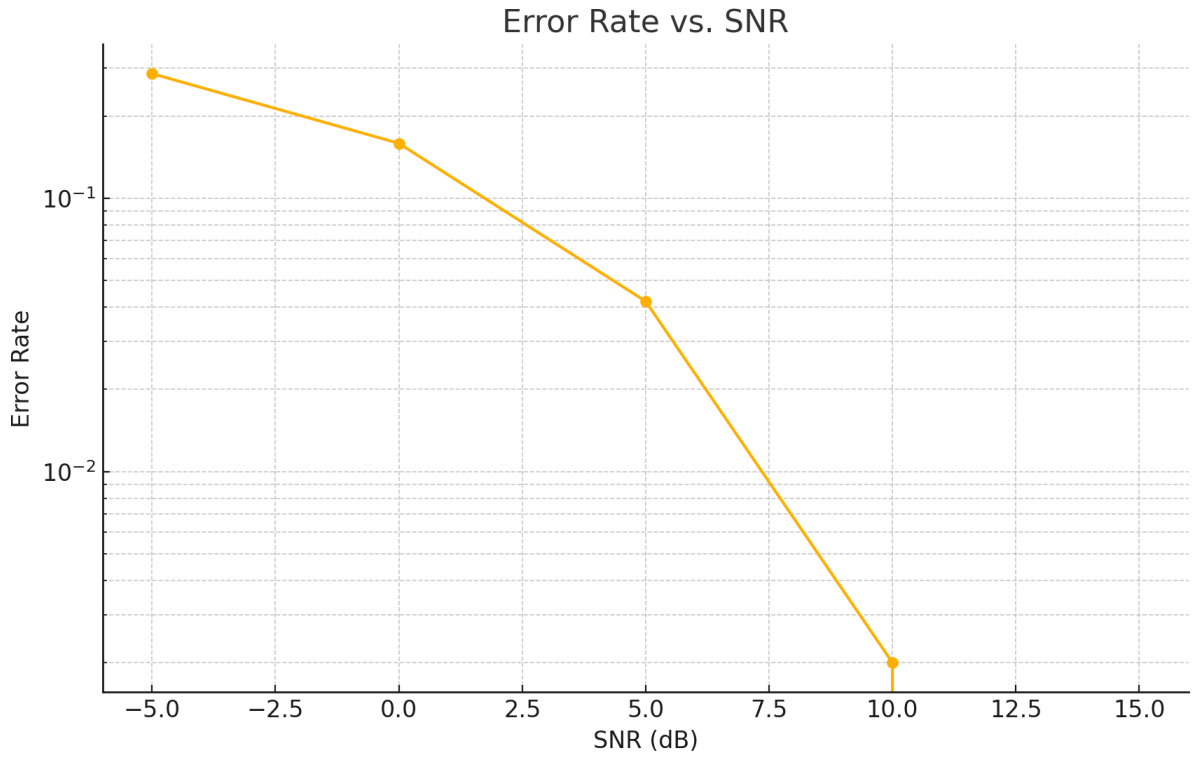
```
plt.figure()

plt.subplot(2,1,1)
plt.plot(noise)
plt.title('AWGN with Nominal Power')
plt.xlabel('Sample')
plt.ylabel('Amplitude')

plt.subplot(2,1,2)
plt.plot(scaled_noise)
plt.title('AWGN with Specified Power')
plt.xlabel('Sample')
plt.ylabel('Amplitude')

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

Part B – Simulate pulse signal transmission in AWGN channel.

Task

In this section, we will simulate the transmission of random pulse signals {−1,1} over an AWGN channel. The probability of transmitting −1 and 1 are equal. The channel adds AWGN noise to the transmitted signals. At the receiver, we implement a simple detection rule: any received value above 0 is detected as 1, and any value below 0 is detected as −1. We will then compare the detected message with the transmitted message **to evaluate the performance of the system under different noise power conditions.** The Signal-to-Noise Ratio (SNR) in dB will be calculated and used to analyze the error rate on a logarithmic scale.

An example result plot:



The example code is attached at the end in case help is required. However, you are encouraged to think through the task and try to program it.

Tips for Students

Understanding the Simulation

- **Signal and Noise:** Remember that the transmitted signal consists of pulses {−1,1} and that the noise added by the channel is modelled as Additive White Gaussian Noise (AWGN).
- **SNR (Signal-to-Noise Ratio):** This is a measure of the signal power relative to the noise power. Higher SNR values mean a clearer signal with less noise interference.
- **Detection Rule:** The receiver decides the transmitted signal based on the received value. If the received value is greater than 0, it detects a 1; if less than 0, it detects a -1.

Steps to Implement the Simulation

1. **Generate the Transmitted Signal:** Use random integers to create a sequence of {−1,1}.
2. **Calculate Noise Power:** For a given SNR, calculate the noise power.
3. **Generate AWGN:** Use the calculated noise power to generate the noise.
4. **Simulate Reception:** Add the noise to the transmitted signal to simulate the received signal.
5. **Apply Detection Rule:** Compare the received signal to 0 to detect the transmitted pulses.
6. **Calculate Error Rate:** Compare the detected signal to the original transmitted signal to find the error rate.

Key Concepts to Remember

- **AWGN:** This type of noise has a constant power spectral density and is a good model for many types of communication channels.
- **Error Rate:** This is a critical metric that shows how well your communication system performs. Lower error rates mean better performance.
- **Logarithmic Scale for Error Rate:** When plotting error rates, a logarithmic scale helps visualize the changes more clearly, especially when error rates vary by orders of magnitude.

Analyzing Results

- **High Error Rate at Low SNR:** Expect high error rates when SNR is low because noise significantly distorts the signal.
- **Decreasing Error Rate with Increasing SNR:** As SNR improves, the error rate should decrease exponentially, indicating better signal clarity and detection accuracy.
- **Error-Free Transmission:** At very high SNR values, the error rate may drop to zero, indicating nearly perfect signal detection.

Report Requirement:

Title Page

- Title of the Report
- Course Name and Code
- Your Name and Student ID
- Date of Submission

Table of Contents

- List of Sections and Subsections with page numbers

1. **Introduction**

- Objective: Briefly state the objectives of the practical session.
- Overview: Provide a brief overview of the topics covered in the practical session.

2. **Part A: Crash Course on Matlab/Python**
   - Show your code examples and screenshots of your results. You can show any code you have tried, including your own practice.

3. **Part B: Simulate Pulse Signal Transmission in AWGN Channel**
   - Provide a brief overview of the simulation process and detection rule.
   - List the parameters used in the simulation (e.g., number of samples, SNR values).
   - Describe the steps involved in the simulation.
   - Present the error rates for each SNR level in a table.
   - Include the following plots:
     - Transmitted signal and received signal with noise for each SNR level.
     - Error rate vs. SNR plot with a logarithmic scale.
   - Analyze the results, discussing how noise affects the signal and the performance of the detection rule at different SNR levels. Provide insights into the significance of SNR in communication systems and how improving SNR can reduce error rates.

4. **Conclusion**

- Summarize the key findings from both parts of the practical session.
- Reflect on the skills and knowledge gained through the session.

5. **References**
   - List any references or resources used to complete the practical session and report. Use IEEE style. Please search online and find out what is the IEEE reference style.

6. **Appendices**

- **Code Listings:** Include complete code listings for Matlab and Python used in the practical session. Please include detailed comments.
- **Additional Figures:** Include any additional figures or plots that support your analysis.

Appendix

Matlab Implementation

```
% Parameters
N = 1000;                    % Number of samples
transmitted_signal = 2*randi([0 1], 1, N) - 1; % Generate random pulse signals {-1, 1}
signal_power = 1;            % Signal power (Ps)

% SNR values in dB to simulate
snr_db = [-5, 0, 5, 10, 15];

% Simulation
error_rates = [];
for i = 1:length(snr_db)
   snr = 10^(snr_db(i)/10);      % Convert SNR from dB to linear scale
   noise_power = signal_power / snr; % Calculate noise power
   noise = sqrt(noise_power) * randn(1, N); % Generate AWGN noise
   received_signal = transmitted_signal + noise; % Received signal
   detected_signal = received_signal > 0;  % Detection rule
   detected_signal(detected_signal == 0) = -1; % Convert 0 to -1 for comparison

   % Calculate error rate
   error_rate = sum(detected_signal ~= transmitted_signal) / N;
   error_rates = [error_rates, error_rate];
   fprintf('SNR (dB): %d, Error Rate: %.4f\n', snr_db(i), error_rate);

   % Plot results
   figure;
   subplot(2, 1, 1);
   plot(transmitted_signal);
   title(sprintf('Transmitted Signal (SNR = %d dB)', snr_db(i)));
   xlabel('Sample');
   ylabel('Amplitude');

   subplot(2, 1, 2);
   plot(received_signal);
   title('Received Signal with Noise');
   xlabel('Sample');
   ylabel('Amplitude');
end

% Plot Error Rate vs. SNR
figure;
semilogy(snr_db, error_rates, 'o-');
title('Error Rate vs. SNR');
xlabel('SNR (dB)');
ylabel('Error Rate');
grid on;
```

Python Implementation

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

# Parameters
N = 1000  # Number of samples
transmitted_signal = 2 * np.random.randint(0, 2, N) - 1  # Generate random pulse signals {-1,
1}
signal_power = 1  # Signal power (Ps)

# SNR values in dB to simulate
snr_db = [-5, 0, 5, 10, 15]

# Simulation
error_rates = []

for snr in snr_db:
   snr_linear = 10**(snr / 10)  # Convert SNR from dB to linear scale
   noise_power = signal_power / snr_linear  # Calculate noise power
   noise = np.sqrt(noise_power) * np.random.randn(N)  # Generate AWGN noise
   received_signal = transmitted_signal + noise  # Received signal
   detected_signal = (received_signal > 0).astype(int)  # Detection rule
   detected_signal[detected_signal == 0] = -1  # Convert 0 to -1 for comparison

   # Calculate error rate
   error_rate = np.sum(detected_signal != transmitted_signal) / N
   error_rates.append(error_rate)
   print(f'SNR (dB): {snr}, Error Rate: {error_rate:.4f}')

   # Plot results
   plt.figure(figsize=(10, 6))
   plt.subplot(2, 1, 1)
   plt.plot(transmitted_signal)
   plt.title(f'Transmitted Signal (SNR = {snr} dB)')
   plt.xlabel('Sample')
   plt.ylabel('Amplitude')

   plt.subplot(2, 1, 2)
   plt.plot(received_signal)
   plt.title('Received Signal with Noise')
   plt.xlabel('Sample')
   plt.ylabel('Amplitude')
   plt.tight_layout()
   plt.show()

# Plot Error Rate vs. SNR
plt.figure(figsize=(10, 6))
plt.plot(snr_db, error_rates, marker='o')
plt.title('Error Rate vs. SNR')
plt.xlabel('SNR (dB)')
```

```
plt.ylabel('Error Rate')
plt.yscale('log')  # Set y-axis to log scale
plt.grid(True, which="both", ls="--")
plt.show()
```

Explanation

Transmitted Signal Generation:

- The transmitted signal is generated as random pulse signals {−1,1} with equal probability.
- Matlab: transmitted_signal = 2*randi([0 1], 1, N) - 1;
- Python: transmitted_signal = 2 * np.random.randint(0, 2, N) – 1

AWGN Noise Generation:

- AWGN noise is generated with specified power levels based on the SNR.
- Matlab: noise = sqrt(noise_power) * randn(1, N);
- Python: noise = np.sqrt(noise_power) * np.random.randn(N)

Received Signal:

- The received signal is the sum of the transmitted signal and the noise.
- Matlab: received_signal = transmitted_signal + noise;
- Python: received_signal = transmitted_signal + noise

Detection Rule:

- At the receiver, a simple detection rule is applied: any value above 0 is detected as 1, and any value below 0 is detected as −1.
- Matlab: detected_signal = received_signal > 0; detected_signal(detected_signal == 0) = -1;
- Python: detected_signal = (received_signal > 0).astype(int); detected_signal[detected_signal == 0] = -1

Error Rate Calculation:

- The error rate is calculated as the ratio of incorrectly detected signals to the total number of signals.
- Matlab: error_rate = sum(detected_signal ~= transmitted_signal) / N;
- Python: error_rate = np.sum(detected_signal != transmitted_signal) / N

Signal-to-Noise Ratio (SNR) Calculation:

- SNR is specified in dB and converted to a linear scale to compute the noise power.
- Matlab and Python: snr_linear = 10^(snr / 10)

Visualization:

- The error rate versus SNR is plotted on a logarithmic scale to better visualize the variations.
- Matlab: semilogy(snr_db, error_rates, 'o-');
- Python: plt.yscale('log')