LAB 1: Discrete-Time Signals & Random Sequences

1. OBJECTIVES

After completing this lab, students will be able to:

Generate and represent discrete signals

Perform operations with δ(n) and u(n)

Plot signals using STEM

Observe and analyze waveforms

Generate random signals (Uniform, Gaussian)

Plot histograms and interpret distribution patterns

Understand Central Limit Theorem (CLT)

2. THEORY BACKGROUND

2.1 Basic Sequences

- **Impulse Function**

$$\delta(n) = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases}$$

- **Unit Step Function**

$$u(n) = \begin{cases} 1, & n \geq 0 \\ 0, & n < 0 \end{cases}$$

2.2 Basic Signal Operations

Time shifting:

$$x[n - n_0]$$

Time reversal:

$$x[-n]$$

Amplitude scaling:

$$Ax[n]$$

Time scaling (discrete case only integer scaling allowed)

2.3 Random Sequences

- **Uniform Distribution**
  - All values are equally likely within interval $[a, b]$.
- **Gaussian (Normal) Distribution**
  - Characterized by:
    - Mean μ
    - Variance σ^2

A random sequence is a sequence of random variables:

$$X[n]$$

Important statistical properties:

- Mean:

$$\mu = E[X]$$

- Variance:

$$\sigma^2 = E[(X - \mu)^2]$$

- Autocorrelation:

$$R_X[k] = E[X[n]X[n-k]]$$

2.4 Central Limit Theorem

The sum of many independent random variables tends toward a Gaussian distribution, regardless of the original distribution.

3. CODING IMPLEMENTATION

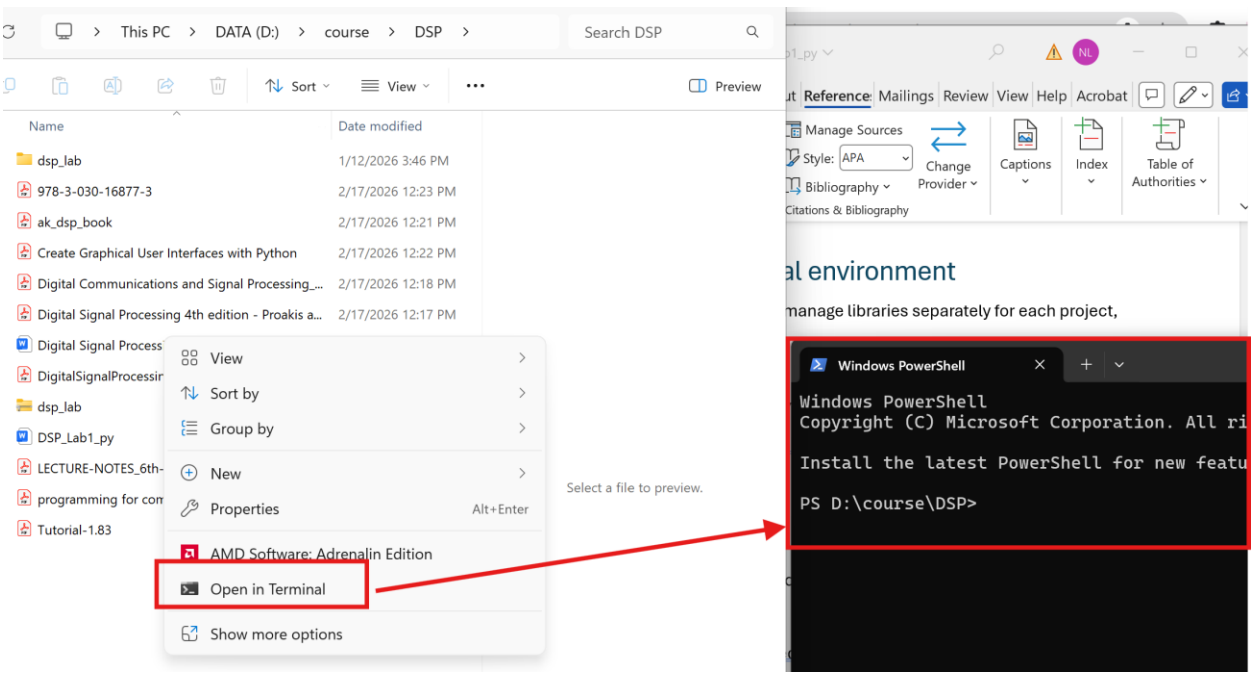
3.1 Sample Coding

3.1.1 How to create a Python virtual environment

Creating a Python virtual environment helps manage libraries separately for each project, avoiding version conflicts.

Detailed steps:

- Open the terminal and navigate to your project directory.



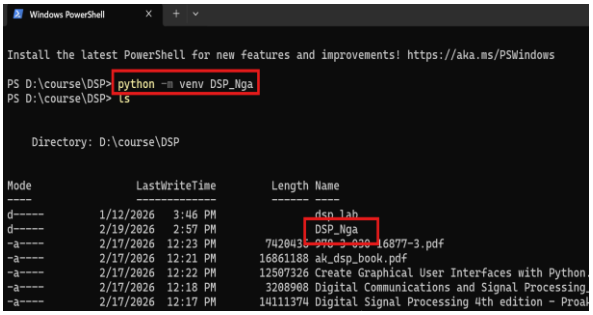
Creating a virtual environment:

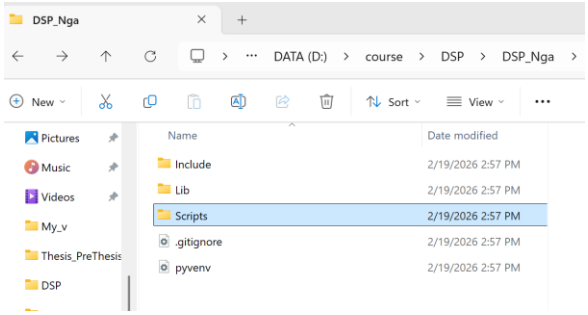
Windows/macOS/Linux: *python -m venv DSP_Nga*

This command:

- Uses Python's built-in venv module
- Creates a virtual environment folder named **DSP_Nga**
- Inside that folder, Python creates:
  - `Scripts/`
  - `Lib/`
  - `Include/`
  - a local Python executable

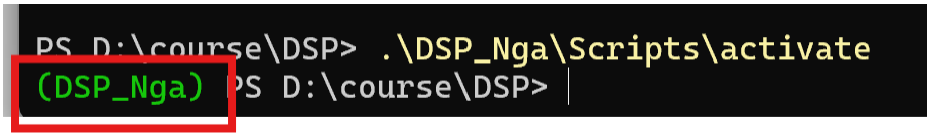
Then the command: ls → Shows that the folder DSP_Nga now exists in:





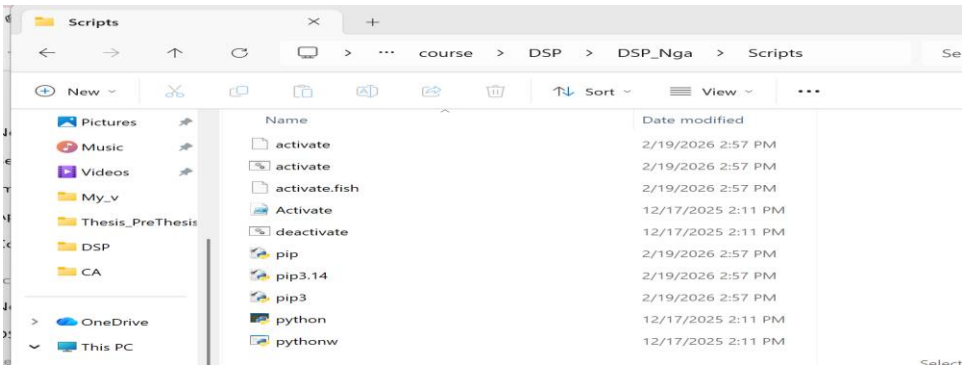
Activating the virtual environment:

Windows: .\DSP_Nga\Scripts\activate



After running the command, you see: the text **(DSP_Nga)** in green means the virtual environment is now activated.
This indicates that:

- Python and pip will now use the isolated environment DSP_Nga
- Installed packages will be stored inside this environment
- It prevents conflicts with global Python packages



macOS/Linux: source .venv/bin/activate

Usage: After activation, the terminal will display (.venv), at which point pip install <package_name> will install the library only in this environment.

Exiting the environment: deactivate.

Alternative methods:

Using Conda: conda create -n myenv python=3.x.

Using Visual Studio Code: Open Command Palette (Ctrl+Shift+P) -> Python: Create Environment.

1️⃣ Basic Variables (Matrix / Array)

Matlab

a = [3] % scalar

x = [1,2,3] % row vector

y = [1;2;3] % column vector

A = [1,2,3;4,5,6] % matrix

```
import numpy as np

a = 3  # scalar

x = np.array([1, 2, 3])          # 1D array (row-like)
y = np.array([[1], [2], [3]])    # column vector (2D)

A = np.array([[1, 2, 3],
              [4, 5, 6]])        # matrix
```

Step 1: install numpy

```
(DSP_Nga) PS D:\course\DSP\DSP_Nga> python DSP_Lab1.py
Traceback (most recent call last):
  File "D:\course\DSP\DSP_Nga\DSP_Lab1.py", line 1, in <module>
    import numpy as np
ModuleNotFoundError: No module named 'numpy'
(DSP_Nga) PS D:\course\DSP\DSP_Nga> pip install numpy
Collecting numpy
  Downloading numpy-2.4.2-cp314-cp314-win_amd64.whl.metadata (6.6 kB)
Downloading numpy-2.4.2-cp314-cp314-win_amd64.whl (12.4 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.4/12.4 MB 63.8 MB/s  0:00:00
Installing collected packages: numpy
Successfully installed numpy-2.4.2

[notice] A new release of pip is available: 25.3 -> 26.0.1
[notice] To update, run: python.exe -m pip install --upgrade pip
(DSP_Nga) PS D:\course\DSP\DSP_Nga>
```

Step 2:

```
(DSP_Nga) PS D:\course\DSP\DSP_Nga> pip install --upgrable pip

Usage:
  pip install [options] <requirement specifier> [package-index-options] ...
  pip install [options] -r <requirements file> [package-index-options] ...
  pip install [options] [-e] <vcs project url> ...
  pip install [options] [-e] <local project path> ...
  pip install [options] <archive url/path> ...

no such option: --upgrable
(DSP_Nga) PS D:\course\DSP\DSP_Nga> python DSP_Lab1.py
(DSP_Nga) PS D:\course\DSP\DSP_Nga>
```

Step 3:

```
import numpy as np

a = 3  # scalar

x = np.array([1, 2, 3])          # 1D array (row-like)
y = np.array([[1], [2], [3]])    # column vector (2D)

A = np.array([[1, 2, 3],
              [4, 5, 6]])        # matrix

print ("x row vector is",x, "\n y column vector is \n",y, "\nA matrix is \n", A )
```

```
(DSP_Nga) PS D:\course\DSP\DSP_Nga> python DSP_Lab1.py
x row vector is [1 2 3]
 y column vector is
 [[1]
 [2]
 [3]]
A matrix is
 [[1 2 3]
 [4 5 6]]
(DSP_Nga) PS D:\course\DSP\DSP_Nga>
```

2 Operators Translation

MATLAB	Meaning	Python Equivalent
=	assignment	=
==	equality	==
+	addition	+
-	subtraction	-
*	matrix multiplication	@ or np.matmul()
.*	element-wise multiply	*
^	matrix power	np.linalg.matrix_power(A, n)
.^	element-wise power	**
/	matrix division	np.linalg.solve() (preferred)
./	element-wise division	/
< > <= >=	relational	same

MATLAB	Meaning	Python Equivalent
&	logical AND	and (scalar) / & (array)
`	`	logical OR
~	logical NOT	not (scalar) / ~ (array)
'	transpose	.T
.'	array transpose	.T

Examples

```
# Matrix multiplication
C = A @ A

# Element-wise multiplication
C = A * A

# Element-wise power
C = A ** 2

# Matrix power
C = np.linalg.matrix_power(A, 2)

# Transpose
AT = A.T
```

3️⃣ Indexing Operations

Matlab

x = a:b:c

B = A(2:4,3:6)

Python

range with step

x = np.arange(a, c + 1, b) # careful: Python excludes end

```
# slicing matrix
B = A[1:4, 2:6]   # Python is 0-indexed!
```

⚠️ Important difference:

- MATLAB indexing starts at **1**
- Python indexing starts at **0**

4️⃣ if – elif – else Structure

Matlab

```
if condition1
  command1
elseif condition2
  command2
else
  command3
end
```

Python

```
if condition1:
  command1
elif condition2:
  command2
else:
  command3
```

5️⃣ for Loop

Matlab

```
for index = values
  statements
```

end

Python

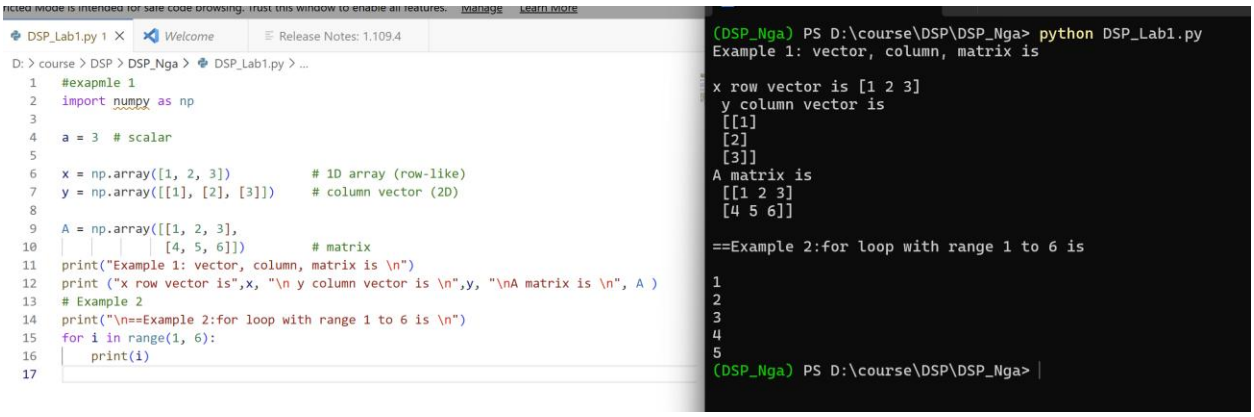
for index in values:

  statements

Example2

for i in range(1, 6):

  print(i)



Example 3

Consider the following sum of sinusoidal functions

$$x(t) = \sin(2\pi t) + \frac{1}{2}\sin(4\pi t) + \frac{1}{3}\sin(6\pi t)$$

It is a **finite Fourier series approximation**.

Using Python to solve it in 3 cases

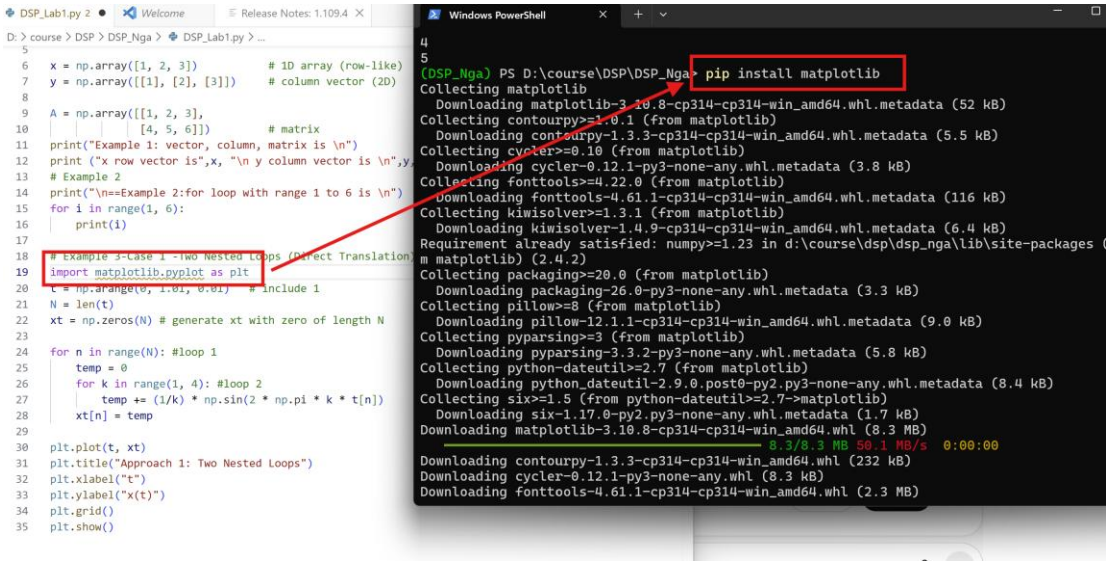
CASE 1: **Two Nested Loops (Direct Translation)**

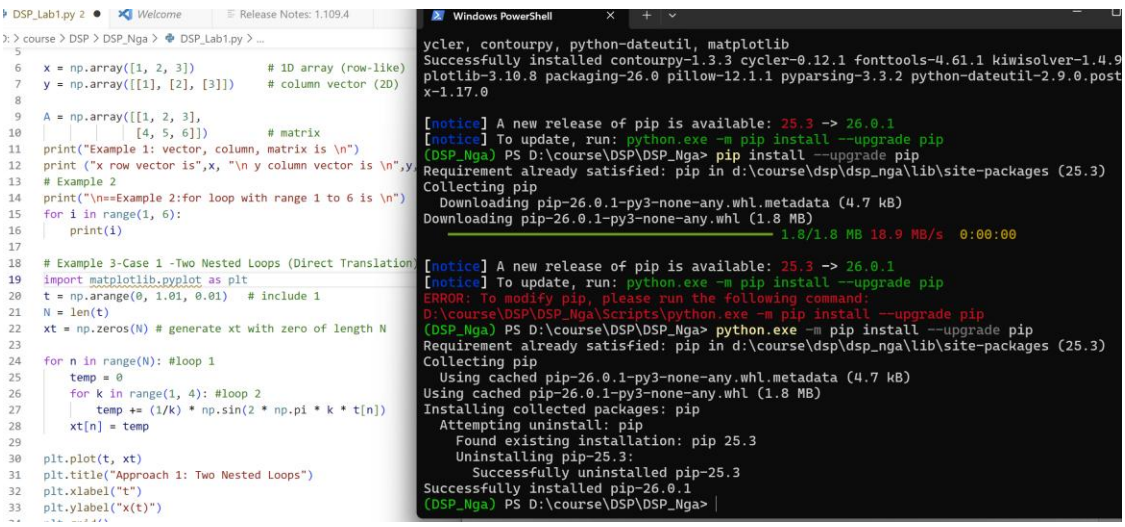
CASE 2: **One Loop (Vectorized Over Time)**

CASE 3: **Fully Vectorized (Matrix Multiplication Style)**

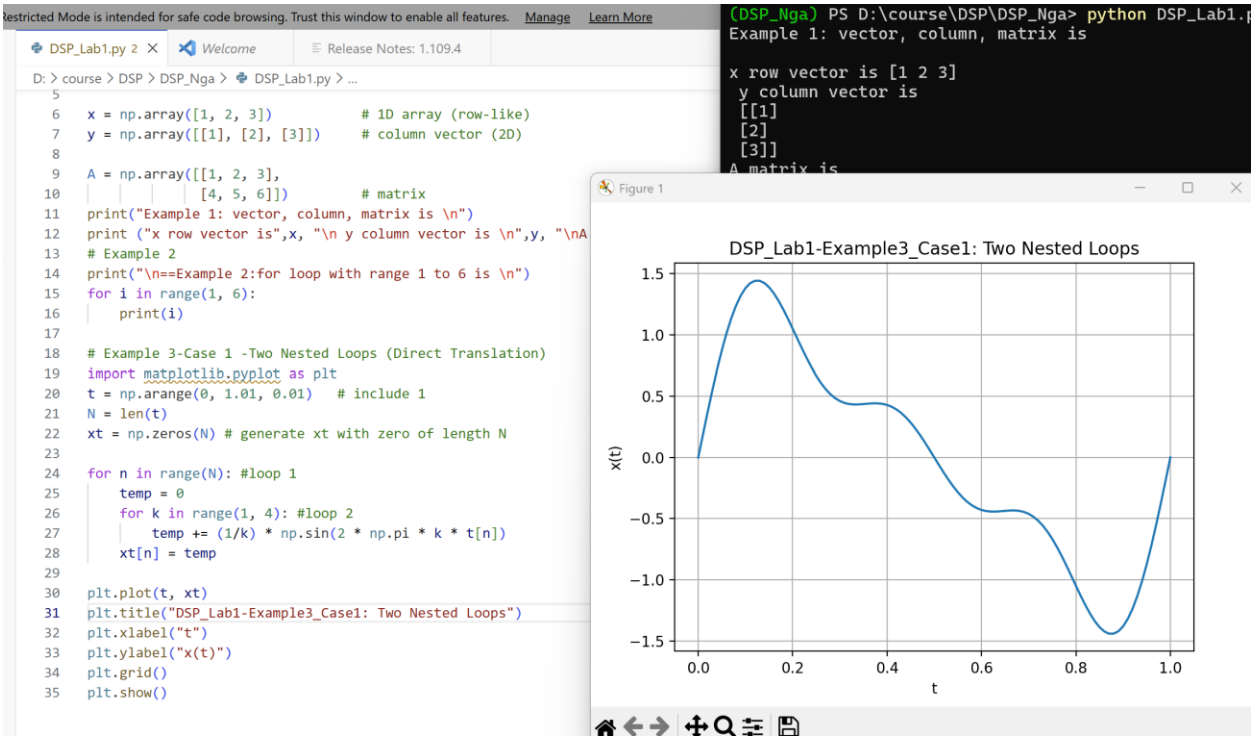
SOL:

Case 1: **Two Nested Loops (Direct Translation)**

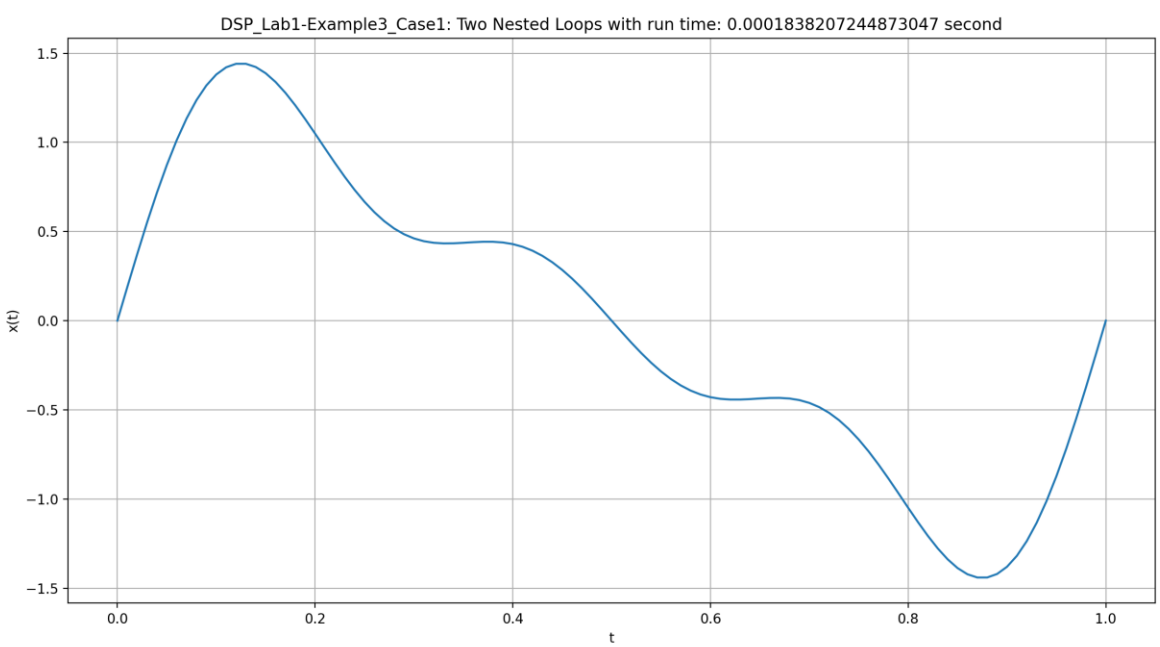

DSP_Lab1.py 2
Welcome
Release Notes: 1.109.4
D: > course > DSP > DSP_Nga > DSP_Lab1.py > ...
x = np.array([1, 2, 3]) # 1D array (row-like)
y = np.array([[1], [2], [3]]) # column vector (2D)
A = np.array([[1, 2, 3],
[4, 5, 6]]) # matrix
print("Example 1: vector, column, matrix is \n")
print ("x row vector is",x, "\n y column vector is \n",y,
# Example 2
print("\n==Example 2:for loop with range 1 to 6 is \n")
for i in range(1, 6):
print(i)
# Example 3-Case 1 -Two Nested Loops (Direct Translation)
import matplotlib.pyplot as plt
t = np.arange(0, 1.01, 0.01) # include 1
N = len(t)
xt = np.zeros(N) # generate xt with zero of length N
for n in range(N): #loop 1
temp = 0
for k in range(1, 4): #loop 2
temp += (1/k) * np.sin(2 * np.pi * k * t[n])
xt[n] = temp
plt.plot(t, xt)
plt.title("Approach 1: Two Nested Loops")
plt.xlabel("t")
plt.ylabel("x(t)")
plt.grid()
plt.show()
Windows PowerShell
4
5
(DSP_Nga) PS D:\course\DSP\DSP_Nga> pip install matplotlib
Collecting matplotlib
Downloading matplotlib-3.10.8-cp314-cp314-win_amd64.whl.metadata (52 kB)
Collecting contourpy>=1.0.1 (from matplotlib)
Downloading contourpy-1.3.3-cp314-cp314-win_amd64.whl.metadata (5.5 kB)
Collecting cycler>=0.10 (from matplotlib)
Downloading cycler-0.12.1-py3-none-any.whl.metadata (3.8 kB)
Collecting fonttools>=4.22.0 (from matplotlib)
Downloading fonttools-4.61.1-cp314-cp314-win_amd64.whl.metadata (116 kB)
Collecting kiwisolver>=1.3.1 (from matplotlib)
Downloading kiwisolver-1.4.9-cp314-cp314-win_amd64.whl.metadata (6.4 kB)
Requirement already satisfied: numpy>=1.23 in d:\course\dsp\dsp_nga\lib\site-packages (
m matplotlib) (2.4.2)
Collecting packaging>=20.0 (from matplotlib)
Downloading packaging-26.0-py3-none-any.whl.metadata (3.3 kB)
Collecting pillow>=8 (from matplotlib)
Downloading pillow-12.1.1-cp314-cp314-win_amd64.whl.metadata (9.0 kB)
Collecting pyparsing>=3 (from matplotlib)
Downloading pyparsing-3.3.2-py3-none-any.whl.metadata (5.8 kB)
Collecting python-dateutil>=2.7 (from matplotlib)
Downloading python_dateutil-2.9.0.post0-py2.py3-none-any.whl.metadata (8.4 kB)
Collecting six>=1.5 (from python-dateutil>=2.7->matplotlib)
Downloading six-1.17.0-py2.py3-none-any.whl.metadata (1.7 kB)
Downloading matplotlib-3.10.8-cp314-cp314-win_amd64.whl (8.3 MB)
8.3/8.3 MB 50.1 MB/s 0:00:00
Downloading contourpy-1.3.3-cp314-cp314-win_amd64.whl (232 kB)
Downloading cycler-0.12.1-py3-none-any.whl (8.3 kB)
Downloading fonttools-4.61.1-cp314-cp314-win_amd64.whl (2.3 MB)

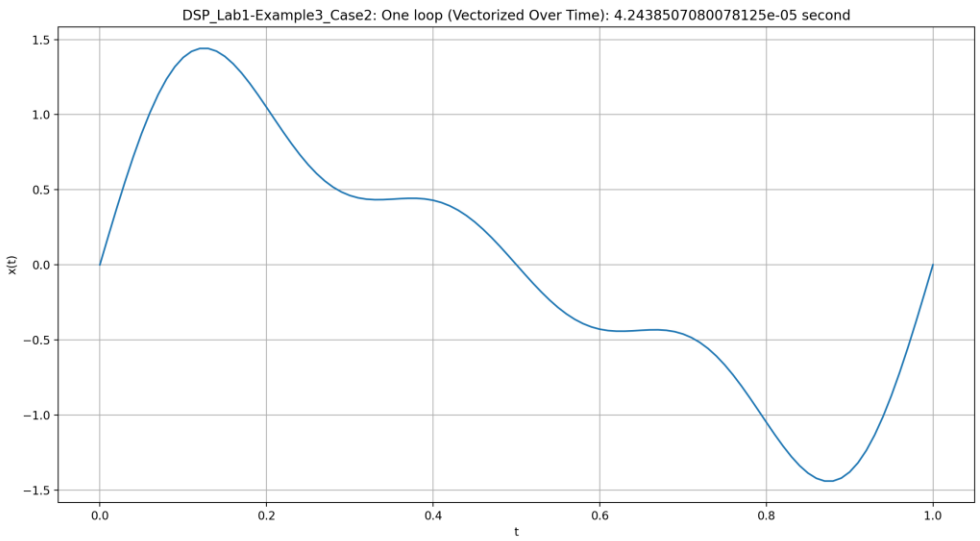

DSP_Lab1.py 2
Welcome
Release Notes: 1.109.4
> course > DSP > DSP_Nga > DSP_Lab1.py > ...
x = np.array([1, 2, 3]) # 1D array (row-like)
y = np.array([[1], [2], [3]]) # column vector (2D)
A = np.array([[1, 2, 3],
[4, 5, 6]]) # matrix
print("Example 1: vector, column, matrix is \n")
print ("x row vector is",x, "\n y column vector is \n",y,
# Example 2
print("\n==Example 2:for loop with range 1 to 6 is \n")
for i in range(1, 6):
print(i)
# Example 3-Case 1 -Two Nested Loops (Direct Translation)
import matplotlib.pyplot as plt
t = np.arange(0, 1.01, 0.01) # include 1
N = len(t)
xt = np.zeros(N) # generate xt with zero of length N
for n in range(N): #loop 1
temp = 0
for k in range(1, 4): #loop 2
temp += (1/k) * np.sin(2 * np.pi * k * t[n])
xt[n] = temp
plt.plot(t, xt)
plt.title("Approach 1: Two Nested Loops")
plt.xlabel("t")
plt.ylabel("x(t)")
Windows PowerShell
ycler, contourpy, python-dateutil, matplotlib
Successfully installed contourpy-1.3.3 cycler-0.12.1 fonttools-4.61.1 kiwisolver-1.4.9
plotlib-3.10.8 packaging-26.0 pillow-12.1.1 pyparsing-3.3.2 python-dateutil-2.9.0.post
x-1.17.0
[notice] A new release of pip is available: 25.3 -> 26.0.1
[notice] To update, run: python.exe -m pip install --upgrade pip
(DSP_Nga) PS D:\course\DSP\DSP_Nga> pip install --upgrade pip
Requirement already satisfied: pip in d:\course\dsp\dsp_nga\lib\site-packages (25.3)
Collecting pip
Downloading pip-26.0.1-py3-none-any.whl.metadata (4.7 kB)
Downloading pip-26.0.1-py3-none-any.whl (1.8 MB)
1.8/1.8 MB 18.9 MB/s 0:00:00
[notice] A new release of pip is available: 25.3 -> 26.0.1
[notice] To update, run: python.exe -m pip install --upgrade pip
ERROR: To modify pip, please run the following command:
D:\course\DSP\DSP_Nga\Scripts\python.exe -m pip install --upgrade pip
(DSP_Nga) PS D:\course\DSP\DSP_Nga> python.exe -m pip install --upgrade pip
Requirement already satisfied: pip in d:\course\dsp\dsp_nga\lib\site-packages (25.3)
Collecting pip
Using cached pip-26.0.1-py3-none-any.whl.metadata (4.7 kB)
Using cached pip-26.0.1-py3-none-any.whl (1.8 MB)
Installing collected packages: pip
Attempting uninstall: pip
Found existing installation: pip 25.3
Uninstalling pip-25.3:
Successfully uninstalled pip-25.3
Successfully installed pip-26.0.1
(DSP_Nga) PS D:\course\DSP\DSP_Nga>

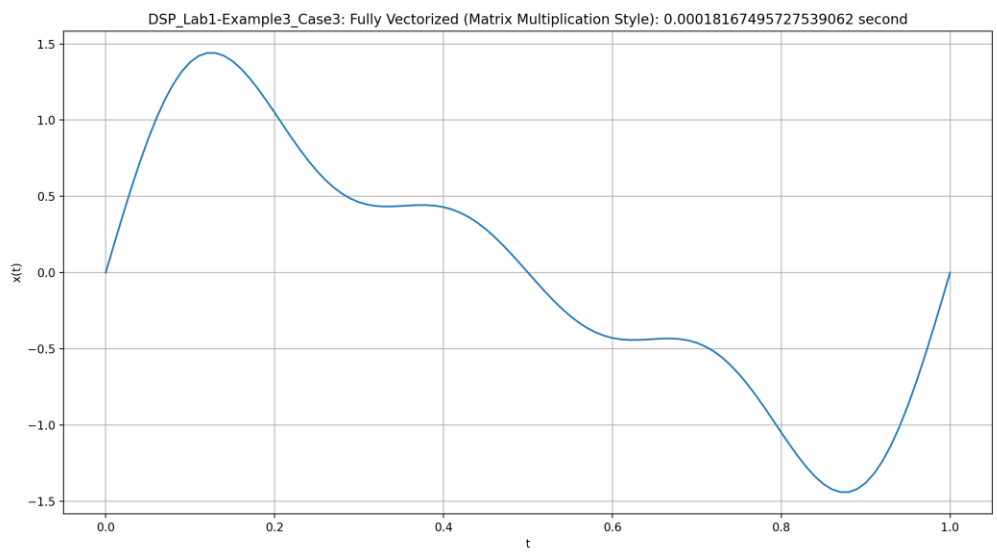
Run



Faster because NumPy computes the sine for the whole vector at once.








```python
start_time3=time.time()
# Broadcasting
xt = np.sum((1/k) * np.sin(2 * np.pi * k * t), axis=0)

end_time3=time.time()
exectime3=end_time3-start_time3
plt.plot(t, xt)
plt.title(f'DSP_Lab1-Example3_Case3: Fully Vectorized (Matrix Multiplication St
plt.xlabel("t")
plt.ylabel("x(t)")
plt.grid()
plt.show()
minEx=0.0
case=3
if exectime3>exectime1:
    case=1
if exectime3 > exectime2:
    case=2
minEx=min(exectime1,exectime2,exectime3)
print(f'exec time is {minEx} with case {case}')
```

```
A matrix is
 [[1 2 3]
 [4 5 6]]

==Example 2:for loop with range 1 to 6 is

1
2
3
4
5
exec time is 4.57763671875e-05 with case 2
(DSP_Nga) PS D:\course\DSP\DSP_Nga>
```

3.2 Practice

Import libraries:

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

PART A – PROBLEM 1

Plot 8 signals (subplot 4×2)

Define impulse & step

```
def delta(n):
    return (n == 0).astype(int)

def step(n):
    return (n >= 0).astype(int)
```

def delta(n):

return (n == 0).astype(int)

def step(n):

return (n >= 0).astype(int)

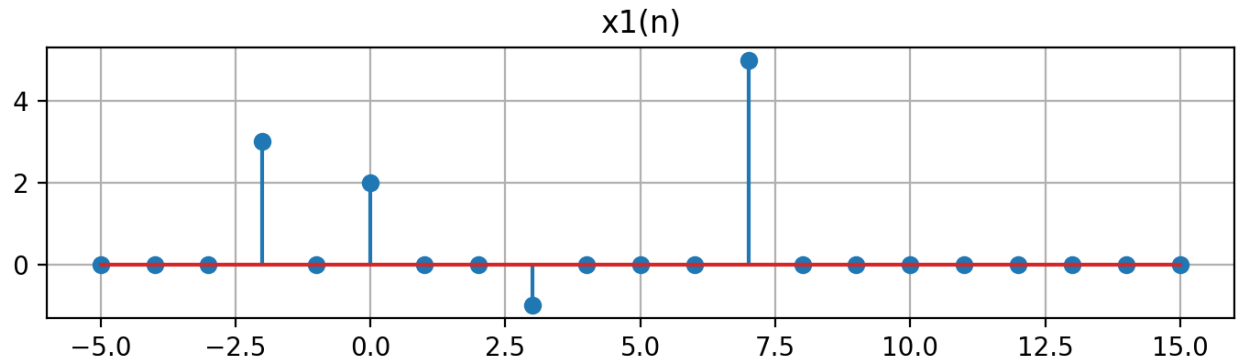
Signal 1

$$x_1(n) = 3\delta(n+2) + 2\delta(n) - \delta(n-3) + 5\delta(n-7)$$

Coding:

n1 = np.arange(-5,16)

x1 = 3*delta(n1+2) + 2*delta(n1) - delta(n1-3) + 5*delta(n1-7)



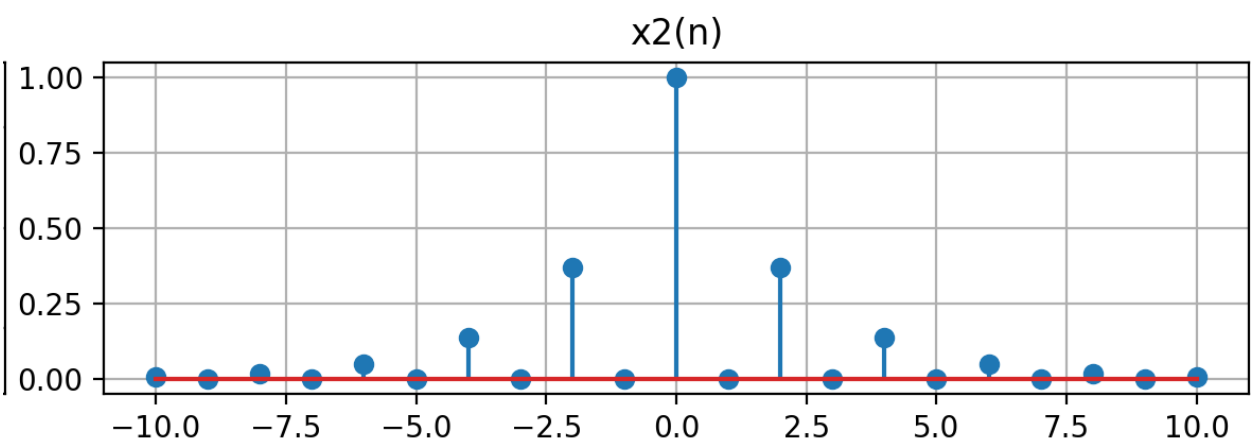
- Observation x1(n): Mostly zero values, a few isolated nonzero samples at specific indices, contains both positive and negative values. **Interpretation:** This is a **sparse discrete signal**, possibly a combination of shifted impulses. **Characteristics:** Finite duration, Energy signal, and Non-periodic

- Signal 2

$$x_2(n) = \sum_{k=-5}^{5} e^{-|k|}\,\delta(n-2k)$$

Coding:

```
n2 = np.arange(-10,11)
x2 = np.zeros_like(n2,dtype=float)
for k in range(-5,6):
  x2 += np.exp(-abs(k)) * delta(n2-2*k)
```



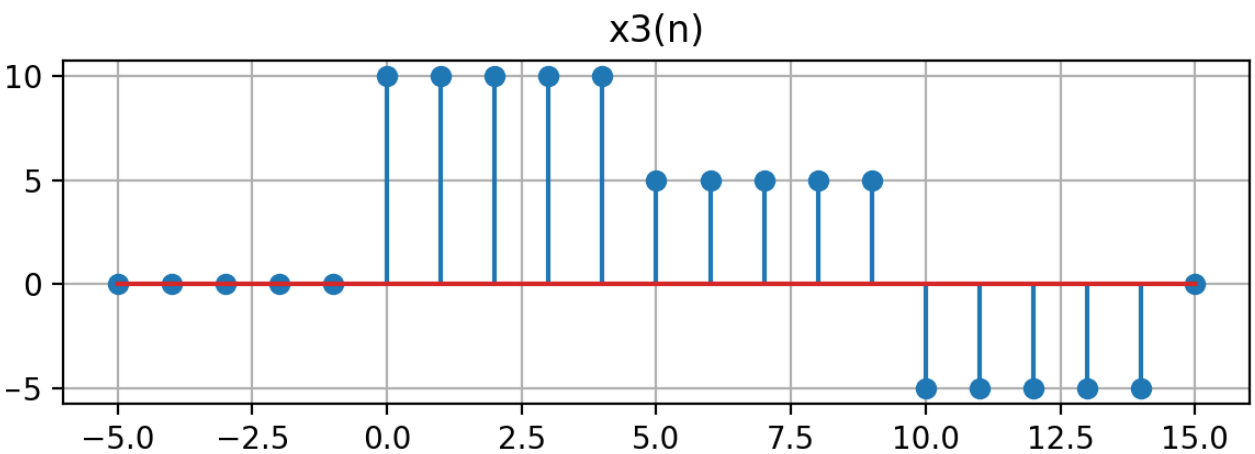
Observation x2(n): Symmetric around n = 0, Maximum at n = 0, Exponentially decaying as |n| increases. **Interpretation:** This looks like a **two-sided decaying exponential**: $x[n] = a^{|n|}$. **Characteristics:** Even signal (symmetric), energy signal, and stable.

- Signal 3: **$x_3(n)$**

$$x_3(n) = 10u(n) - 5u(n-5) - 10u(n-10) + 5u(n-15)$$

Coding:

```
x3 = 10*step(n1) - 5*step(n1-5)  -10*step(n1-10) + 5*step(n1-15)
```



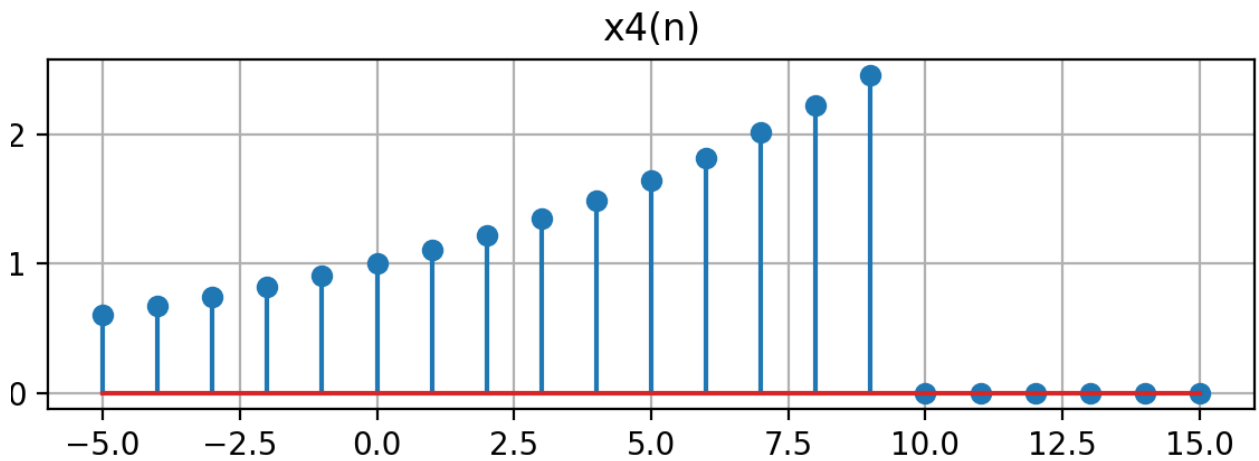
- **Observation x3(n):** Piecewise constant values, positive block from n=0 to 4, smaller positive block from n=5 to 9, negative block from n=10 onward. **Interpretation:** This is a **piecewise discrete signal**, likely constructed using shifted unit step functions. **Characteristics:** Finite support, discontinuous, deterministic signal

- Signal 4: **$x_4(n)$**

$$x_4(n) = e^{0.1n}[u(n+20) - u(n-10)]$$

coding:

```
x4 = np.exp(0.1*n1)*(step(n1+20)-step(n1-10))
```



- Observation x4(n): exponentially increasing values, defined only for n between -5 and about 9, zero afterward. **Interpretation:** This appears to be a **truncated exponential growth signal**. **Characteristics:** Finite duration**, n**ot stable if extended infinitely**,** Energy signal (since truncated).

- Signal 5: **$x_5(n)$**

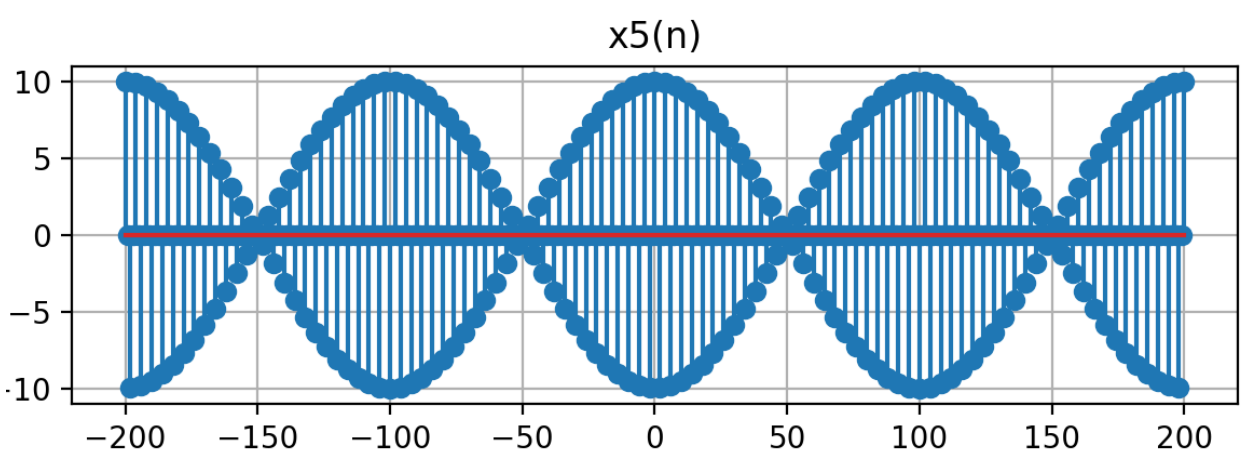
$$x_5(n) = 5[\cos(0.49\pi n) + \cos(0.51\pi n)]$$

Coding:

```
n5 = np.arange(-200,201)

x5 = 5*(np.cos(0.49*np.pi*n5)+   np.cos(0.51*np.pi*n5))
```

Comment: The phenomenon of beating effect (amplitude interference)



Observation: Pure sinusoidal waveform, constant amplitude, symmetric oscillation, and long time range (-200 to 200)

Interpretation:

This is a **pure discrete-time sinusoidal signal**:

$$x[n] = A\cos(\omega n)$$

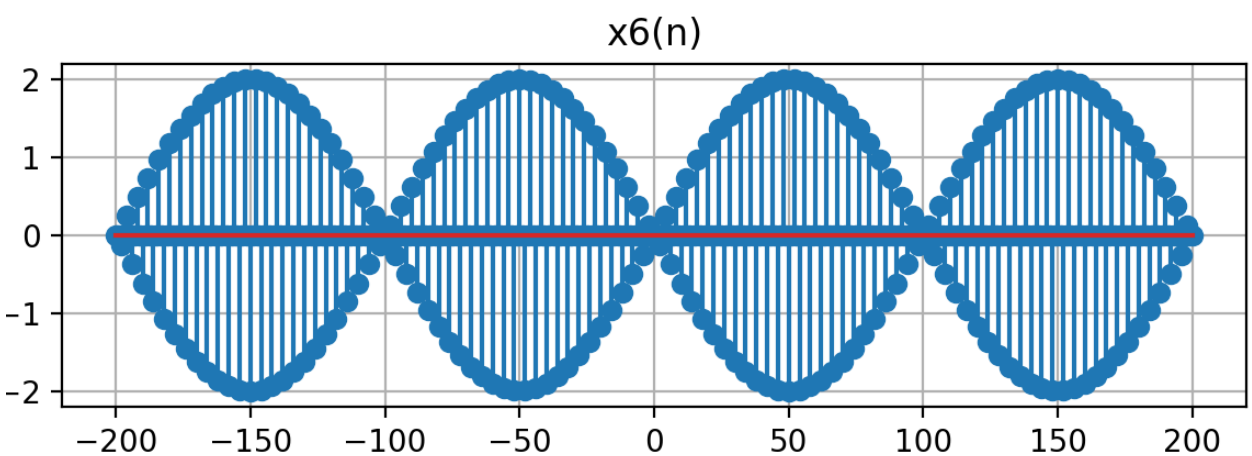
Characteristics: Periodic, power signal (not energy signal), and deterministic

- **Signal 6: $x_6(n)$**

$$x_6(n) = 2\sin(0.01\pi n)\cos(0.5\pi n)$$

```
x6 = 2*np.sin(0.01*np.pi*n5)* np.cos(0.5*np.pi*n5)
```

Comment: Amplitude modulation



Observation: Sinusoidal waveform, smaller amplitude than x5(n), same periodic structure

Interpretation: Another sinusoidal signal with different amplitude or frequency.

Characteristics: Periodic, power signal, stable

- **Singal 7: $x_7(n)$**

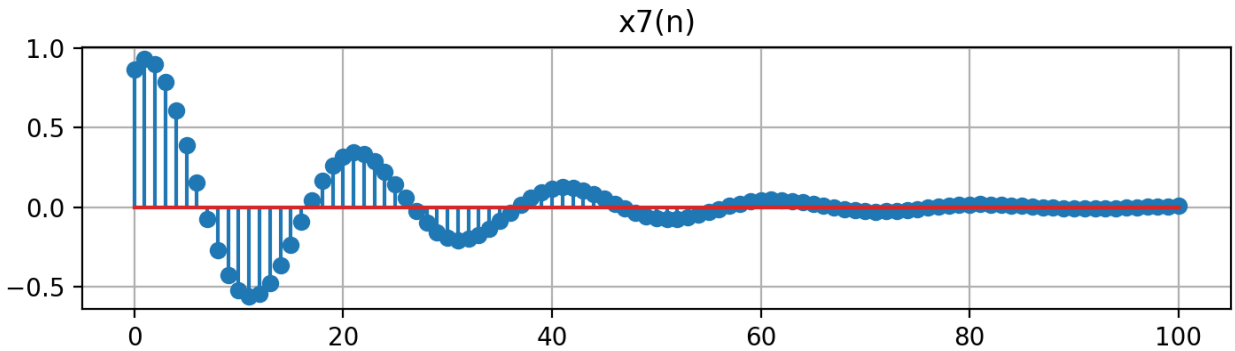
$$x_7(n) = e^{-0.05n}\sin(0.1\pi n + \pi/3)$$

Coding

```
n7 = np.arange(0,101)

x7 = np.exp(-0.05*n7)*  np.sin(0.1*np.pi*n7 + np.pi/3)
```

Comment: Damped sinusoid



Observation: Oscillatory behavior, amplitude decreases over time, approaches zero as n increases.

Interpretation: This is a **damped sinusoid**:

$$x[n] = a^n \cos(\omega n) \text{with } |a| < 1$$

Characteristics: Exponentially decaying, energy signal, stable

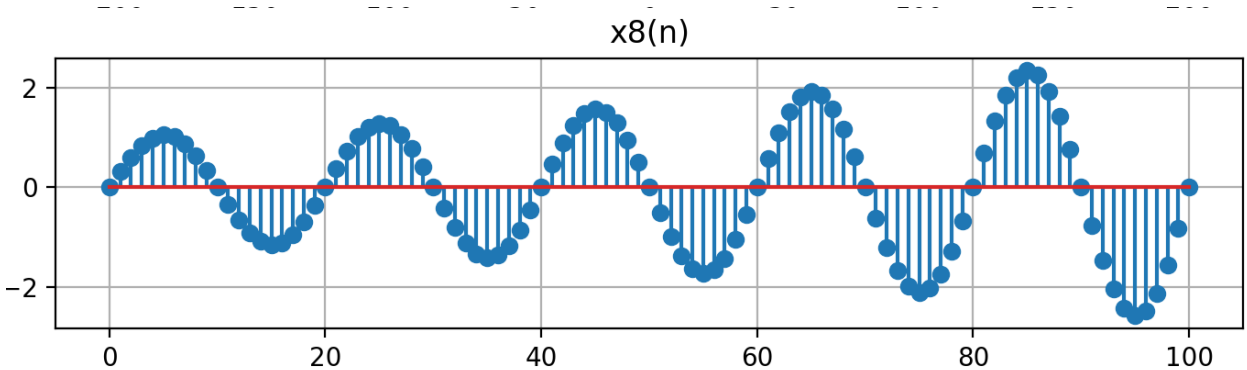
- **Signal 8: $x_8(n)$**

$$x_8(n) = e^{0.01n} \sin(0.1\pi n)$$

Coding:

```
x8 = np.exp(0.01*n7)*    np.sin(0.1*np.pi*n7)
```

Comment: Growing sinusoid



Observation: Oscillatory signal, amplitude increases over time, eventually becomes large

Interpretation: This is a **growing sinusoid**:

$$x[n] = a^n \cos(\omega n) \text{with } |a| > 1$$

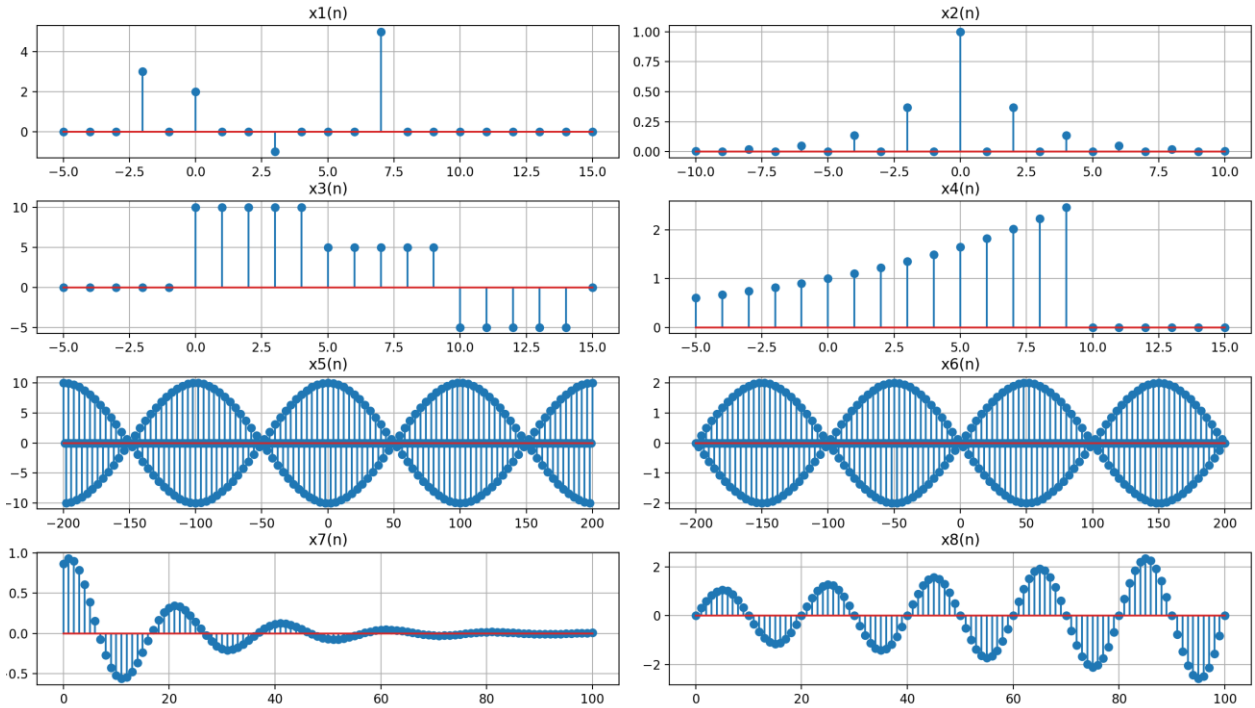
Characteristics: Unstable, infinite energy, non-stationary amplitude

- Plot (4×2)

```
plt.figure(figsize=(12,10))
signals = [x1,x2,x3,x4,x5,x6,x7,x8]
indices = [n1,n2,n1,n1,n5,n5,n7,n7]

for i in range(8):
  plt.subplot(4,2,i+1)
  plt.stem(indices[i],signals[i])
  plt.title(f"x{i+1}(n)")
  plt.grid()
plt.tight_layout()
plt.show()
```

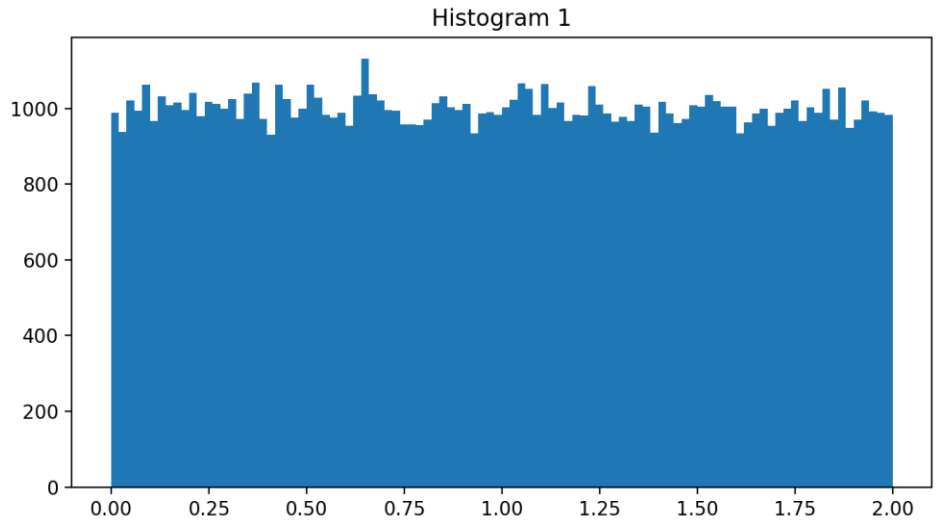
Results:



The figure contains **eight discrete-time signals** labeled from **x1(n) to x8(n)**. They represent different types of signals commonly studied in **Discrete-Time Signal Processing**, including:

- Impulse-like sequences
- Decaying signals
- Piecewise signals
- Exponentials
- Sinusoids
- Damped oscillations
- Amplitude-modulated signals

	Signal Type	**Periodic**	**Energy/Power**	**Stable**
x1(n)	Sparse finite	No	Energy	Yes
x2(n)	Decaying exponential	No	Energy	Yes



Signal	Type	Periodic	Energy/Power	Stable
x3(n)	Piecewise	No	Energy	Yes
x4(n)	Truncated exponential	No	Energy	Yes
x5(n)	Sinusoidal	Yes	Power	Yes
x6(n)	Sinusoidal	Yes	Power	Yes
x7(n)	Damped sinusoid	No	Energy	Yes
x8(n)	Growing sinusoid	No	Infinite	No

PART B – PROBLEM 2

Random Sequences (subplot 2×2)

1) $x_1(n)$ Uniform [0,2]

Coding: x1 = np.random.uniform(0,2,100000)

Histogram 1 – Uniform Distribution on [0, 2]

Observation: The histogram is **flat and approximately constant** across the interval. All bins have roughly equal height. No central peak or skewness is observed.

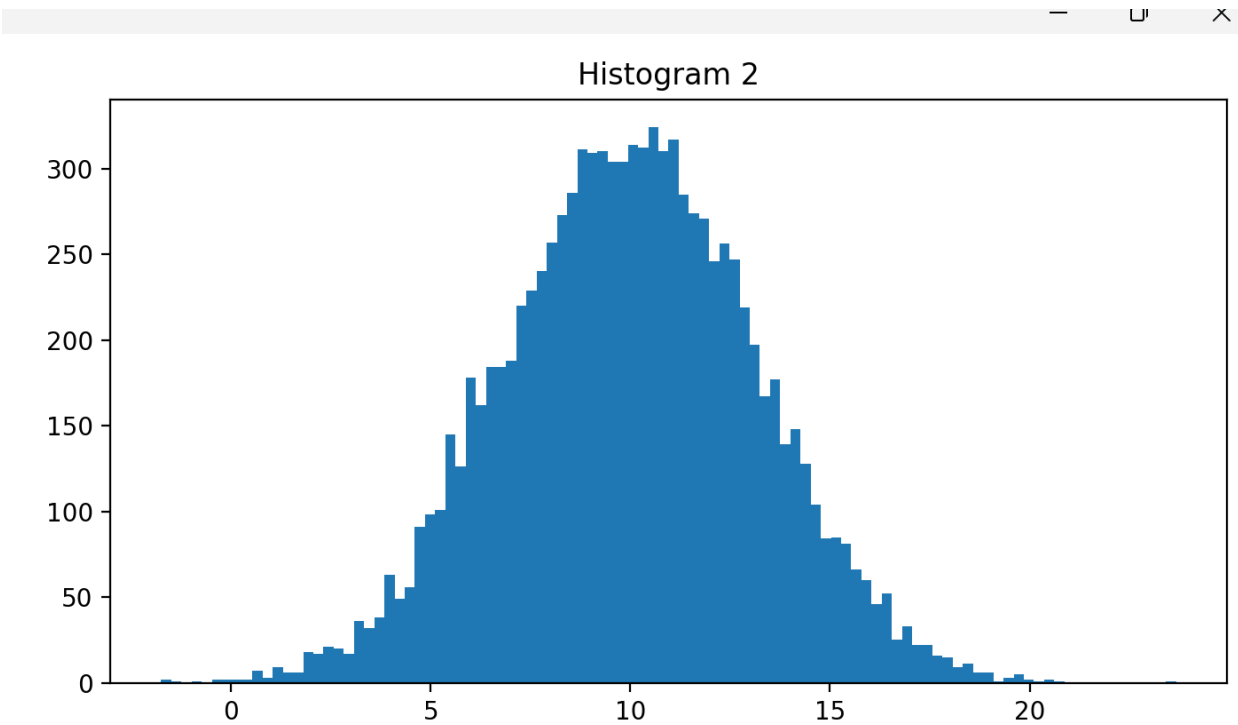
Interpretation: This confirms that the samples follow a **Uniform distribution** over [0, 2].

For a continuous uniform random variable: $f_X(x) = \begin{cases} \frac{1}{b-a}, & a \le x \le b \\ 0, & \text{otherwise} \end{cases}$

Here: $a = 0, b = 2, f_X(x) = 1/2$. The flat shape matches **the theoretical PDF**.

2) $x_2(n)$ Gaussian (mean=10, var=10)

Coding: x2 = np.random.normal(10,np.sqrt(10),10000)



Histogram 2 – Gaussian Distribution (Mean = 10, Variance = 10)

Observation: Bell-shaped curve. Symmetric around approximately **x = 10**. Smooth tapering tails.

Interpretation: This matches a **Normal (Gaussian) distribution**: $X \sim \mathcal{N}(10,10)$

The shape follows: $f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$

- Mean $\mu = 10$, Variance $\sigma^2 = 10$

The smooth bell curve confirms independence and Gaussian sampling.

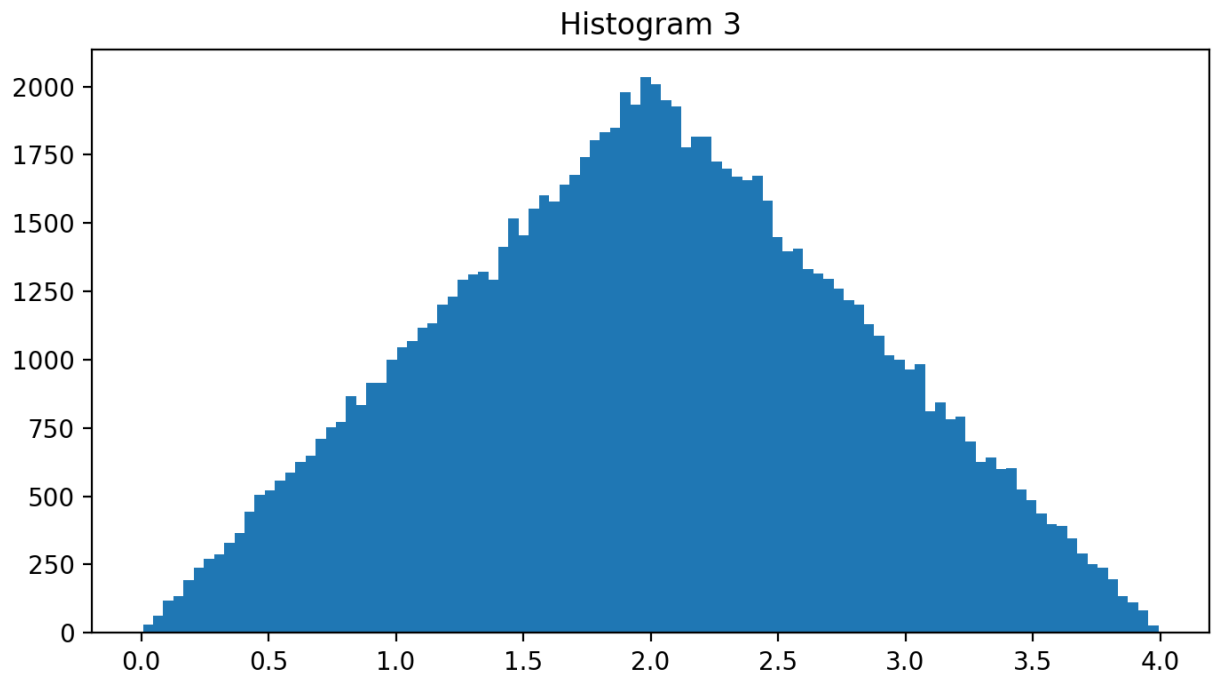
3) Sum of Adjacent Samples

x3(n)=x1(n)+x1(n−1)

Coding

x3 = x1[:-1] + x1[1:]

Comment: Histogram approaches Gaussian distribution (CLT effect).



Histogram 3 – Sum of Two Independent Uniform Variables

Observation: The shape is **triangular**. Maximum probability around the center (~2). Linearly decreasing toward both ends.

Interpretation: If:

$$x_3(n) = x_1(n) + x_1(n-1)$$

where both are uniform on [0,2], then: the resulting distribution is the **convolution of two uniform distributions**. The sum of two independent uniform variables produces a **triangular distribution**.

Range:

$$0 \leq X \leq 4$$

This matches the observed histogram shape. This is a classical convolution result in probability theory.

4) Sum of Four Independent Uniform Variables

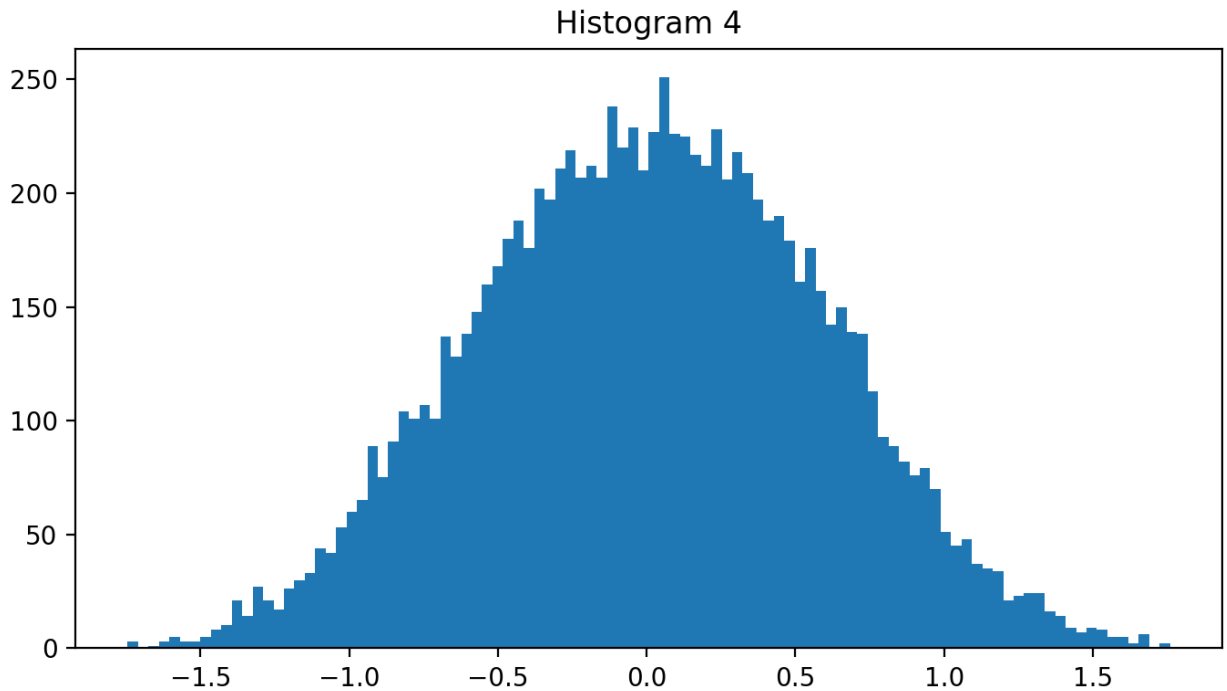
$$x_4(n) = \sum_{k=1}^{4} y_k(n)$$

Coding:

```
y = np.random.uniform(-0.5,0.5,(4,10000))

x4 = np.sum(y,axis=0)
```

Comment: Distribution becomes more Gaussian-like due to CLT.



Histogram 4 – Sum of 4 Independent Uniform Variables

Observation: The distribution appears approximately **Gaussian**. Smooth bell shape.

- Centered around 0. Symmetric.

Interpretation:

$$x_4(n) = \sum_{k=1}^{4} y_k\,(n)$$

where each $y_k(n) \sim U[-0.5,0.5]$

By the **Central Limit Theorem (CLT)**:

The sum of multiple independent random variables tends toward a Gaussian distribution, regardless of the original distribution. Even though each variable is uniform, their sum becomes approximately normal. This explains why Histogram 4 looks Gaussian.

Coding of Plot Histograms (2×2)

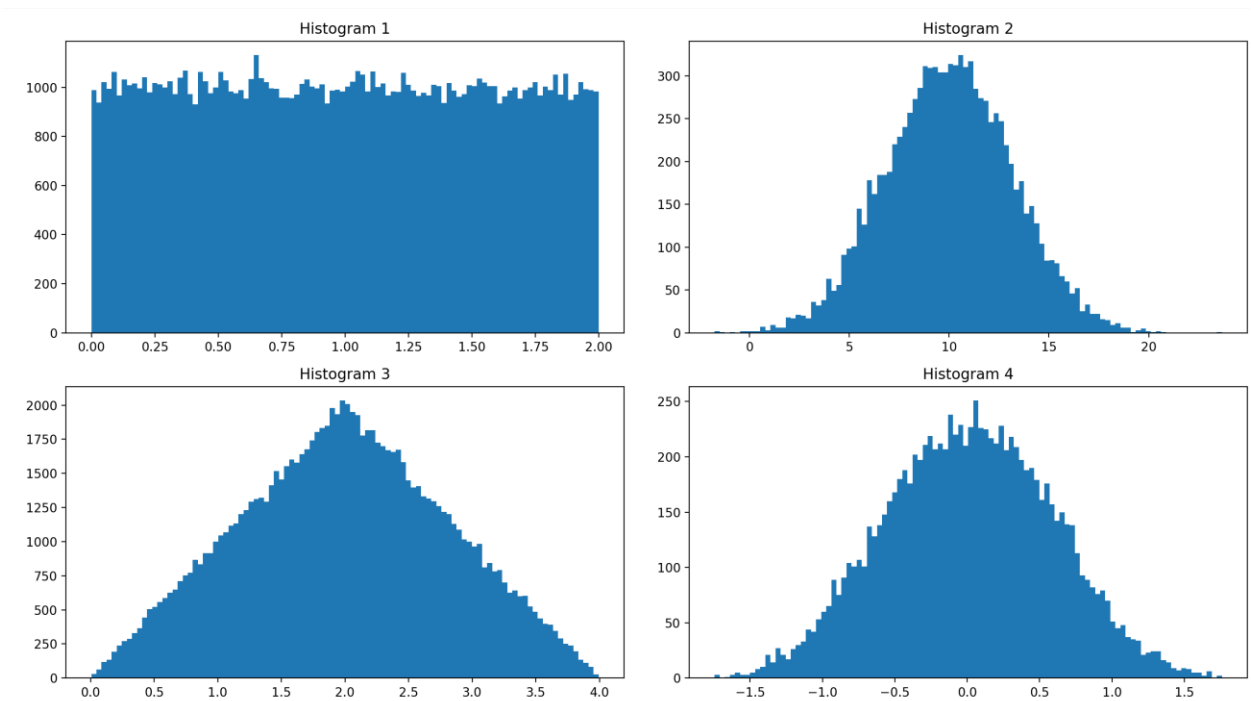
Coding:

plt.figure(figsize=(10,8))

```
data = [x1,x2,x3,x4]

for i in range(4):
    plt.subplot(2,2,i+1)
    plt.hist(data[i],bins=100)
    plt.title(f"Histogram {i+1}")

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



Histogram	Type	Shape	Theory
1	Uniform	Flat	Constant PDF
2	Gaussian	Bell-shaped	Normal distribution
3	Sum of 2 Uniform	Triangular	Convolution result

Histogram Type	Shape	Theory
4	Sum of 4 Uniform	Approx. Gaussian Central Limit Theorem

Key Learning Points

1. Uniform → Flat histogram
2. Gaussian → Bell-shaped histogram
3. Sum of 2 uniform → Triangular distribution
4. Sum of many independent variables → Gaussian (CLT)

4. ADVANCED CHALLENGING EXERCISES

4.1 Exercise 1: Energy and Power Classification

Given the signal:

$$x[n] = (0.9)^n u[n]$$

Tasks

1. Determine analytically whether it is:
   - Energy signal
   - Power signal
   - Neither
2. Compute analytically:

$$E = \sum_{n=-\infty}^{\infty} | x[n] |^2$$

3. Verify numerically using simulation.

Coding:

Results:

Concept Being Tested

- Convergence of geometric series
- Energy definition
- Infinite vs finite summation

4.2 Exercise 2: Discrete-Time Convolution (Manual + Code)

Let:

$$x[n] = u[n] - u[n-5]$$
$$h[n] = (0.8)^n u[n]$$

Tasks

1. Sketch both signals.
2. Compute convolution manually for first 10 samples.
3. Verify using:

y = conv(x,h);

4. Explain why output is smooth exponential rise.

Learning Goal

Understand convolution as:

- Weighted accumulation
- Memory effect
- System response

Coding:

Results:

4.3 Exercise 3: Stationarity Test of Random Sequence

Generate:

$$X[n] = randn(1,10000)$$

Tasks

1. Divide sequence into 5 equal segments.
2. Compute mean of each segment.
3. Compare results.
4. Repeat for variance.

Question:

Is the process approximately Wide-Sense Stationary?

4.4 Exercise 4: Law of Large Numbers

Generate Gaussian noise with increasing length:

- N = 100
- N = 1000
- N = 10000

Compute sample mean each time.

Plot convergence behavior.

Concept:

$$\lim_{N\to\infty} \bar{X}_N = \mu$$

4.5 Exercise 5: Correlated Random Process

Generate:

$$Y[n] = X[n] + 0.8X[n-1]$$

Where $X[n]$is white noise.

Tasks:

1. *Plot autocorrelation of X.*
2. *Plot autocorrelation of Y.*
3. *Compare shapes.*

Question:

Is Y white noise?

Explain mathematically.

Coding:

4.6 Exercise 6: Random Walk Process

Define:

$$S[n] = S[n-1] + W[n]$$

where $W[n]$is white noise.

Tasks:

1. Simulate random walk.
2. Plot trajectory.
3. Compute variance as function of n.

Theoretical Result:

$$Var(S[n]) = n\sigma^2$$

Verify experimentally.

4.7 Exercise 7: Signal Detection in Noise

Let:

$$x[n] = cos(0.2\pi n)$$

Add noise:

$$y[n] = x[n] + 0.5randn$$

Tasks:

1. Plot clean and noisy signal.
2. Estimate SNR.
3. Apply moving average filter.
4. Compare before and after.

4.8 Exercise 8: Monte Carlo Estimation of Probability

Generate Gaussian noise.

Estimate:

$$P(X > 1)$$

Compare with theoretical value from Gaussian table.

4.9 Exercise 9: Ergodicity Test

Check if time average ≈ ensemble average.

Procedure:

1. Generate 100 realizations.
2. Compute ensemble mean at n=50.
3. Compute time mean of one long realization.

4. Compare.

4.10 Exercise 10: Sampling Effect Study

Generate continuous-time sinusoid (conceptually):

$$x(t) = cos(2\pi f t)$$

Sample at:

- fs = 5f
- fs = 2f
- fs = 1.2f

Plot discrete signals.

Observe aliasing behavior.

5. What to submit

Please solve exercise 1 to 10 with more explanation and coding clearly.
Include all files and report in a folder: fullname_Lab1
Submit it via blackboard