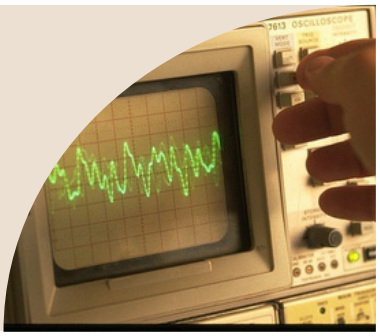
TNS Project Report : Part 1

Filière : GSTR1

HAMMOUCHI LOUAY



Part 1: Code Documentation: Convolution

Objectif :

The code is designed to simulate the output sequence y[n], which results from the convolution of two user-provided sequences, x[n] and h[n].

1. Import libraries:

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

- numpy: For numerical computations and array manipulations.
- matplotlib.pyplot: For plotting the resulting signals.
- re: For parsing user input strings (used to identify delta
- impulses and their parameters).

2. Define the delta function:

```
# Define the delta function
def delta(n):
    return np.where(n == 0, 1, 0)
```

Purpose: Defines a discrete delta function (δ[n]\delta[n]) for a given input array n.

- For n=0: Returns 1 (impulse).
- For n≠0: Returns 0.

Use Case: Conceptually represents impulses in the input

3. Parse Delta input

```
# Function to parse user input for delta impulses (handles positive and negative deltas)
def parse_delta_input(user_input, n_range):
    """
    Parses a string input (e.g., 'delta[n-3] + 2*delta[n+2] - delta[n]')
    and returns the signal array with proper handling of signs.
    """
    signal = np.zeros(len(n_range))
    matches = re.findall(r'([+-]?\d*)\*?delta\[n([+-]\d+)?\]', user_input)
```

Initialize the signal: A zero array (signal) of the same size as the time range (n_range).
Extract matches: The regular expression parses the input for terms like:

- Amplitudes: 2, -1, etc.
- Shifts: n-3, n+2, etc.
- Matches are tuples like ('2', '-3'), where 2 is the amplitude and -3 is the shift.

Handle each match

```
    for amp, shift in matches:
        # Handle amplitude
        if amp in ["", "+", "-"]:
            amplitude = 1 if amp == "" or amp == "+" else -1
        else:
            amplitude = int(amp)

        # Handle shift
        shift_value = int(shift) if shift else 0
        index = np.where(n_range == -shift_value)[0]  # Correct shift direction
        if len(index) > 0:
            signal[index] += amplitude  # Sum amplitudes if overlaps occur
    return signal
```

- **Amplitude Parsing:** Handles positive, negative, or omitted amplitudes (delta[n], -delta[n]).
- **Shift Parsing:** Translates shifts (n-3) into indices in the n_range.
- **Update Signal:** Adds the impulse amplitude at the computed index, summing amplitudes if overlapping impulses exist.

Return the Parsed Signal: The final signal array, with delta impulses represented by non-zero values at appropriate positions.

4. Find Delta Position

```
# Function to find positions of delta impulses
def find_delta_positions(signal, n_range):
    return n_range[np.where(signal != 0)]
```

- **Purpose**: Identifies positions in n_range where the signal has non-zero values (i.e., delta impulses).
- **Use Case**: Helps plot and label the impulses on the graph.

5. Main Program Flow

Define time range

```
# Range of time indices
n = np.arange(-10, 11)
```

Defines a range of discrete time indices (n) from -10 to 10.

User Input

```
# User input for x[n] and h[n]
print("Enter the input signal x[n] using Dirac impulses (e.g., delta[n-3] + 2*delta[n+2] - delta[n]):")
x_input = input("x[n]: ")
print("Enter the impulse response h[n] using Dirac impulses (e.g., delta[n] - delta[n-1]):")
h_input = input("h[n]: ")
```

Asks the user to insert the input signals x[n] and h[n], with examples provided for clarification.
Users describe signals using Dirac delta notations (e.g., 2*delta[n-3] - delta[n+2]).

Parse Signals

```
# Parse user inputs to generate signals
x = parse_delta_input(x_input, n)
h = parse_delta_input(h_input, n)
```

Converts user inputs into discrete arrays representing the signals x[n] and h[n].

Convolution

```
# Perform convolution
y = np.convolve(x, h, mode='full')
n_y = np.arange(2 * n[0], 2 * n[-1] + 1)  # Time indices for y
```

- **np.convolve: Computes the convolution y[n]=x[n]⋆h[n].**
- **mode='full': Returns the entire convolution result.**
- **n_y: Defines the range of indices for the output signal y[n].**

Find impulse locations and amplitudes

```
# Find positions and amplitudes of impulses in y[n]
delta_positions = find_delta_positions(y, n_y)
amplitudes = y[np.where(y != 0)]
```

- **delta_positions:** Time indices of non-zero values in y[n].
- **amplitudes:** Corresponding non-zero values (amplitudes of impulses).

6. Plotting

Styling and labeling

```
# Check if there are any impulses to plot
if len(delta_positions) > 0 and len(amplitudes) > 0:
    # Plot the convolution result
    markerline, stemlines, baseline = plt.stem(delta_positions, amplitudes, basefmt=" ")
    plt.setp(stemlines, 'color', 'blue', 'linewidth', 1.5)  # Styling
    plt.setp(markerline, 'color', 'red', 'markersize', 5)   # Styling
    plt.setp(baseline, 'color', 'black', 'linewidth', 1.0)  # Styling
    plt.title("Representation of Convolution Result y[n]")
    plt.xlabel("n (Time Index)")
    plt.ylabel("Amplitude")
    plt.grid()
```

Ensure that only integer numbers are represented on the x-axis and y-axis

```
# Force integer ticks on both axes
plt.xticks(np.arange(min(delta_positions)-1, max(delta_positions)+2, 1))
plt.yticks(np.arange(int(np.floor(min(amplitudes)) - 1), int(np.ceil(max(amplitudes)) + 2), 1))
```

7. Handle no impulse case

```
    plt.show()
else:
    print("No impulses to plot in the convolution result.")
```

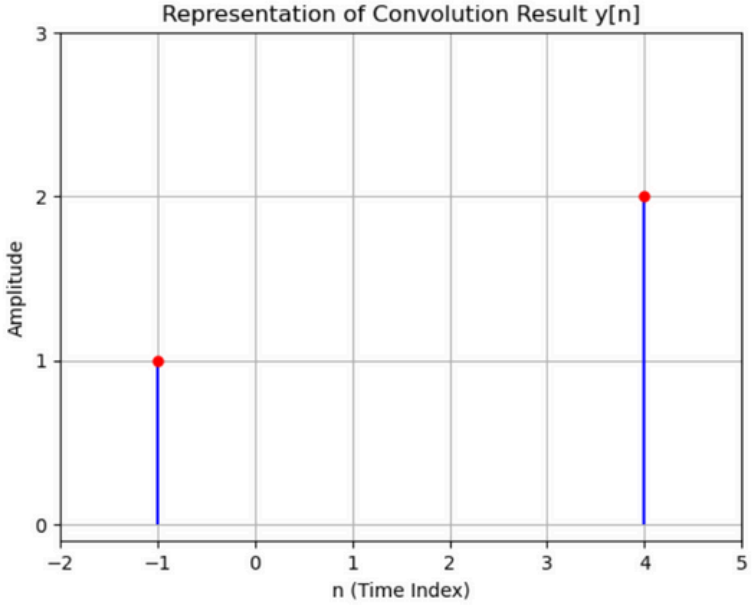
- **plt.show(): Displays the output signal y[n].**
- **else:**
- **print("No impulses to plot in the convolution result."): If there are no impulses in y[n], skips plotting and prints a message instead.**

8. Displays the convolution result array and the positions of the delta impulses in y[n]

```
# Output results with delta positions
print("y[n] (Result of x[n] * h[n]):", y)
print("Delta positions in y[n]:", delta_positions)
```

Example of an output signal:

```
Enter the input signal x[n] using Dirac impulses (e.g., delta[n-3] + 2*delta[n+2] - delta[n]):
x[n]:  2*delta[n-3] + delta[n+2]
Enter the impulse response h[n] using Dirac impulses (e.g., delta[n] - delta[n-1]):
h[n]:  delta[n-1]
```



FIN