Code Snippets

generate_reverb_data.sce

```
clear; close; clc;

//input wav file
[inp, fs_inp] = wavread("D:/LEVEL UP/IITB Info Related Docs/ACADEMICS/YEAR 2/SEM-3/EE229 - Signal Processing 1/Assignments/CA1/CA1_wav_sce/BheegiRegular.wav");
[rir, fs_rir] = wavread("D:/LEVEL UP/IITB Info Related Docs/ACADEMICS/YEAR 2/SEM-3/EE229 - Signal Processing 1/Assignments/CA1/CA1_wav_sce/long_echo_hall_16k.wav");

//SINGLE CHANNEL CONVOLUTION FUNCTION
function [output]=fun_conv(inp1, rir1)
  //PASTE YOUR CODE FOR CONVOLUTION OF TWO SIGNALS
  disp(">>> EXECUTING THE MANUAL FOR-LOOP FUNCTION NOW <<<");
  N = length(inp1);
  M = length(rir1);
  output_len = N + M - 1;
  output = zeros(1, output_len);


  for i = 1:N
    for j = 1:M
      output(i + j - 1) = output(i + j - 1) + inp1(i) * rir1(j);
      disp(i);disp(j);
    end
  end
  //output = conv(inp1,rir1);
endfunction

//obtain RIR for left channel
rir_left = rir(1,:);
//obtain RIR for right channel
rir_right = rir(2,:);

//obtain convolved signal for left channel
out_left = fun_conv(inp,rir_left);
//obtain convolved signal for right channel
out_right = fun_conv(inp,rir_right);

//obtaining stereo sound by combaining two channels
out = [out_left;out_right];
out = out/max(abs(out));

//playing convolved signal
playsnd(out,fs_inp);

//writing convolved signal
wavwrite(out,fs_inp,"D:/LEVEL UP/")
```

toy_problem.sce

```
// Clear console and variables
clc;
clear;

// --- Part 1: Define Signals ---
// Given parameters
alpha = 0.4;
n_vec = 0:10; // Time vector for n = 0, 1, ..., 10

// Define x[n] = alpha^n * u[n]
// Since n >= 0, u[n] is 1, so x[n] = alpha^n
x = alpha.^n_vec;

// Define h[n] = u[n]
// Since n >= 0, u[n] is 1
h = ones(1, length(n_vec));

// --- Part 2: Manual Convolution Code ---
// Initialize the output vector y with zeros
y_code = zeros(1, length(n_vec));

// Outer loop iterates through each output sample 'n'
for n = 0:10
  // Initialize the sum for the current y[n]
  y_n_sum = 0;

  // Inner loop calculates the sum for k from 0 to n
  // y[n] = sum_{k=0 to n} x[k]h[n-k]
  for k = 0:n
    // Scilab uses 1-based indexing, so we use k+1 and (n-k)+1
    term = x(k + 1) * h(n - k + 1);
    y_n_sum = y_n_sum + term;
  end

  // Store the final sum in our output vector y
  // Again, use n+1 for the index
  y_code(n + 1) = y_n_sum;
end

disp("Output from manual convolution code (y_code):");
disp(y_code'); // Transpose (') for a clean column display

// --- Part 3: Compute using Closed-Form Expression and Verify ---
// y[n] = (1 - alpha^(n+1)) / (1 - alpha)
y_formula = (1 - alpha.^(n_vec + 1)) / (1 - alpha);

disp("Output from closed-form formula (y_formula):");
disp(y_formula'); // Transpose for clean display

// --- Verification ---
// Calculate the difference between the two methods
difference = y_code - y_formula;

disp("Maximum absolute difference between the two methods:");
disp(max(abs(difference))); // Should be 0 or a very small floating point error
```

toy.py

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

# --- 1. Define Parameters and Signals ---
alpha = 0.4
# Define the time vector for n = 0, 1, ..., 10
n = np.arange(11)

# Generate the signal x[n] = alpha^n * u[n]
# Since n is non-negative, u[n] = 1
x_n = alpha**n

# Generate the signal h[n] = u[n]
# Since n is non-negative, h[n] = 1
h_n = np.ones_like(n)

# --- 2. Calculate the Output y[n] in Two Ways ---

# Method A: Numerical Convolution
# Use NumPy's convolve function. We only need the first 11 points of the
output.
y_numerical = np.convolve(x_n, h_n)[:len(n)]

# Method B: Closed-Form Analytical Solution
# From the geometric series formula, y[n] = (1 - alpha^(n+1)) / (1 -
alpha)
y_analytical = (1 - alpha**(n + 1)) / (1 - alpha)


# --- 3. Graphically Verify the Results ---
print("Verification:")
print(f"Max difference between numerical and analytical results:
{np.max(np.abs(y_numerical - y_analytical))}")

# Create a plot to display the results
plt.figure(figsize=(12, 7))
plt.style.use('seaborn-v0_8-notebook')

# Plot the numerical result with blue filled circles
```

```
plt.stem(n, y_numerical, linefmt='b-', markerfmt='bo', basefmt=" ",
label='Numerical Result (np.convolve)')

# Plot the analytical result with red open circles on top to show they
match
plt.stem(n, y_analytical, linefmt='r:', markerfmt='ro', basefmt=" ",
label='Analytical Result (Formula)')

# Add plot titles, labels, and legend for clarity
plt.title(r'Final Output Signal $y[n] = x[n] * h[n]$', fontsize=15)
plt.xlabel('Sample (n)', fontsize=12)
plt.ylabel('Amplitude', fontsize=12)
plt.xticks(n)
plt.legend(fontsize=11)
plt.grid(True)

# Show the plot
plt.show()
```