

# **Signals and Systems (EE2001E)**



**NATIONAL INSTITUTE OF TECHNOLOGY CALICUT**

## **Department of Electrical and Electronics Engineering**

**Faculty Guide:** Dr. Mithun M S

**Project Title:**

- 1.convolution of two functions theoretically in MATLAB.
- 2.Demonstration of Frequency Spectrum of a Square Wave using Arduino and 555 Timer.

**Semester:** S3

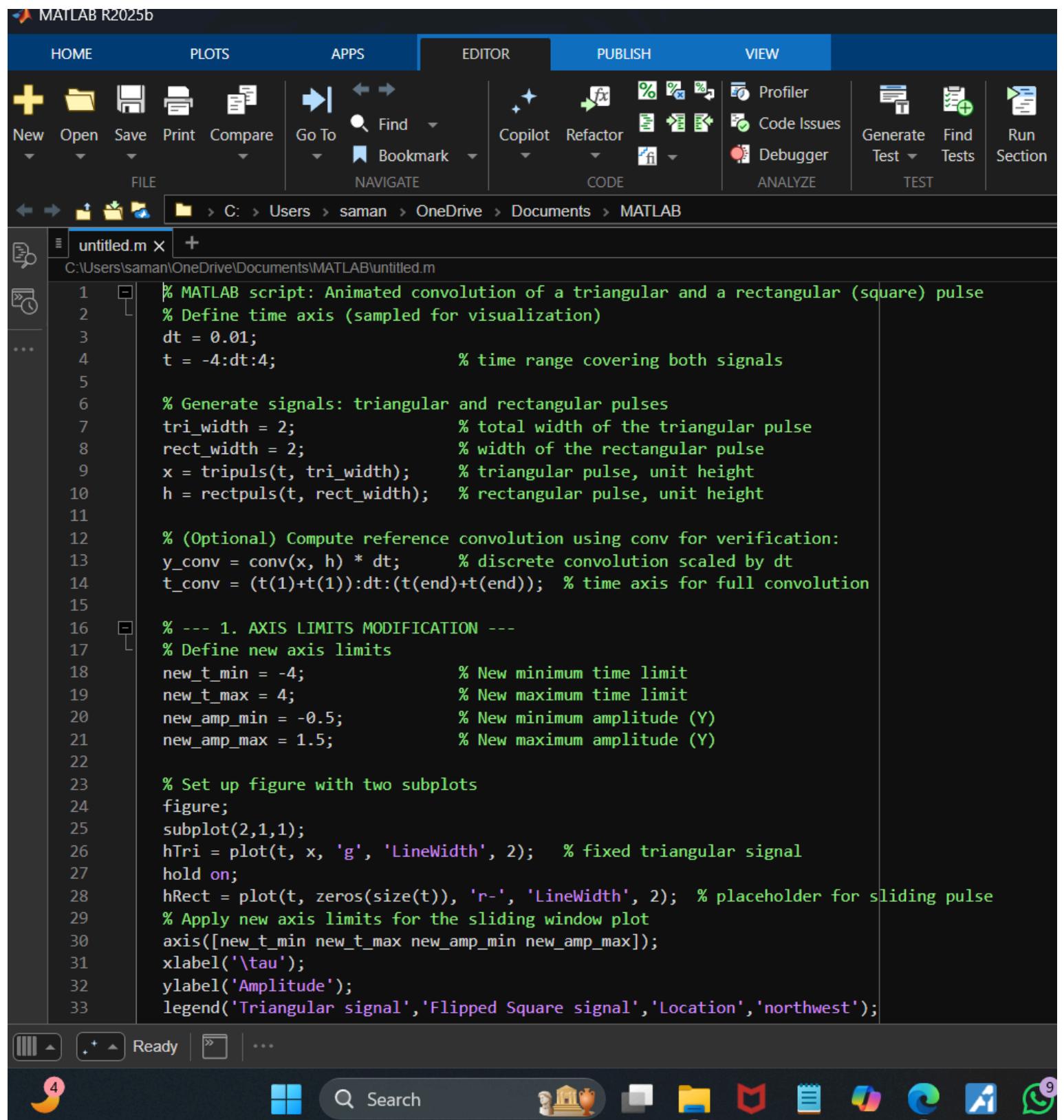
**Department:** Electrical and Electronics Engineering

**Institute:** NIT Calicut

**Submitted by:**

D SAI ROHAN  
CH SURYA SASHANK  
J SAMANVITHA

# 1.CONVOLUTION OF TWO FUNCTIONS THEORETICALLY IN MATLAB:



The screenshot shows the MATLAB R2025b interface with the 'EDITOR' tab selected. A script named 'untitled.m' is open in the editor. The code performs the following steps:

- Defines a time axis from -4 to 4 with a step of 0.01.
- Generates triangular and rectangular pulses.
- Computes reference convolution using `conv`.
- Modifies axis limits for a plot.
- Creates a figure with two subplots: one for the triangular signal and one for the convolution result.

```
% MATLAB script: Animated convolution of a triangular and a rectangular (square) pulse
% Define time axis (sampled for visualization)
dt = 0.01;
t = -4:dt:4; % time range covering both signals

% Generate signals: triangular and rectangular pulses
tri_width = 2; % total width of the triangular pulse
rect_width = 2; % width of the rectangular pulse
x = tripuls(t, tri_width); % triangular pulse, unit height
h = rectpuls(t, rect_width); % rectangular pulse, unit height

% (Optional) Compute reference convolution using conv for verification:
y_conv = conv(x, h) * dt; % discrete convolution scaled by dt
t_conv = (t(1)+t(1)):dt:(t(end)+t(end)); % time axis for full convolution

% --- 1. AXIS LIMITS MODIFICATION ---
% Define new axis limits
new_t_min = -4; % New minimum time limit
new_t_max = 4; % New maximum time limit
new_amp_min = -0.5; % New minimum amplitude (Y)
new_amp_max = 1.5; % New maximum amplitude (Y)

% Set up figure with two subplots
figure;
subplot(2,1,1);
hTri = plot(t, x, 'g', 'LineWidth', 2); % fixed triangular signal
hold on;
hRect = plot(t, zeros(size(t)), 'r-', 'LineWidth', 2); % placeholder for sliding pulse
% Apply new axis limits for the sliding window plot
axis([new_t_min new_t_max new_amp_min new_amp_max]);
xlabel('\tau');
ylabel('Amplitude');
legend('Triangular signal', 'Flipped Square signal', 'Location', 'northwest');
```

HOME PLOTS APPS EDITOR PUBLISH VIEW

New Open Save Print Compare Go To Find Copilot Refactor Profiler

Bookmark ANALYZE Debugger Generate Test Find TEST Run Section

FILE NAVIGATE CODE

C:\Users\saman\OneDrive\Documents\MATLAB\untitled.m

```
33 legend('Triangular signal','Flipped Square signal','Location','northwest');
34 title('Step-by-Step Convolution: Sliding Window');
35
36 subplot(2,1,2);
37 hConv = plot(NaN, NaN, 'b', 'LineWidth', 2); % placeholder for convolution output
38 hold on;
39 % Apply new axis limits for the convolution output plot
40 axis([t_conv(1) t_conv(end) new_amp_min new_amp_max]);
41 xlabel('t');
42 ylabel('Convolution y(t)');
43 title('Manual Convolution Output');
44
45 % --- 2. REPEAT ANIMATION MODIFICATION ---
46 % Animation loop: slide the square signal across the triangular and accumulate overlap
47 % We will use an outer loop to repeat the entire process indefinitely (or N times)
48 num_repetitions = 5; % Set how many times you want the animation to repeat
49
50 for rep = 1:num_repetitions
51     y_anim = zeros(size(t)); % Reset convolution output for each repetition
52
53     % Clear the previous convolution plot for a fresh start
54     if rep > 1
55         delete(hConv); % Remove the old line handle
56         hConv = plot(NaN, NaN, 'b', 'LineWidth', 2); % Create a new placeholder
57     end
58
59     for w = 1:length(t)
60         shift = t(w);
61         % Compute the flipped and shifted version of h: h(shift - tau)
62         h_shift = rectpuls(shift - t, rect_width); % unit-height rectangle flipped & shifted
63
64         % Compute convolution integral at this shift (area under product x.*h_shift)
65         y_anim(w) = trapz(t, x .* h_shift); % numeric integration (continuous conv)
```

Ready ...

4 Search

9

MATLAB R2025b

HOME PLOTS APPS EDITOR PUBLISH VIEW

New Open Save Print Compare Go To Find Copilot Refactor Profiler Code Issues Debugger Generate Test Find Run Section Run Run Section

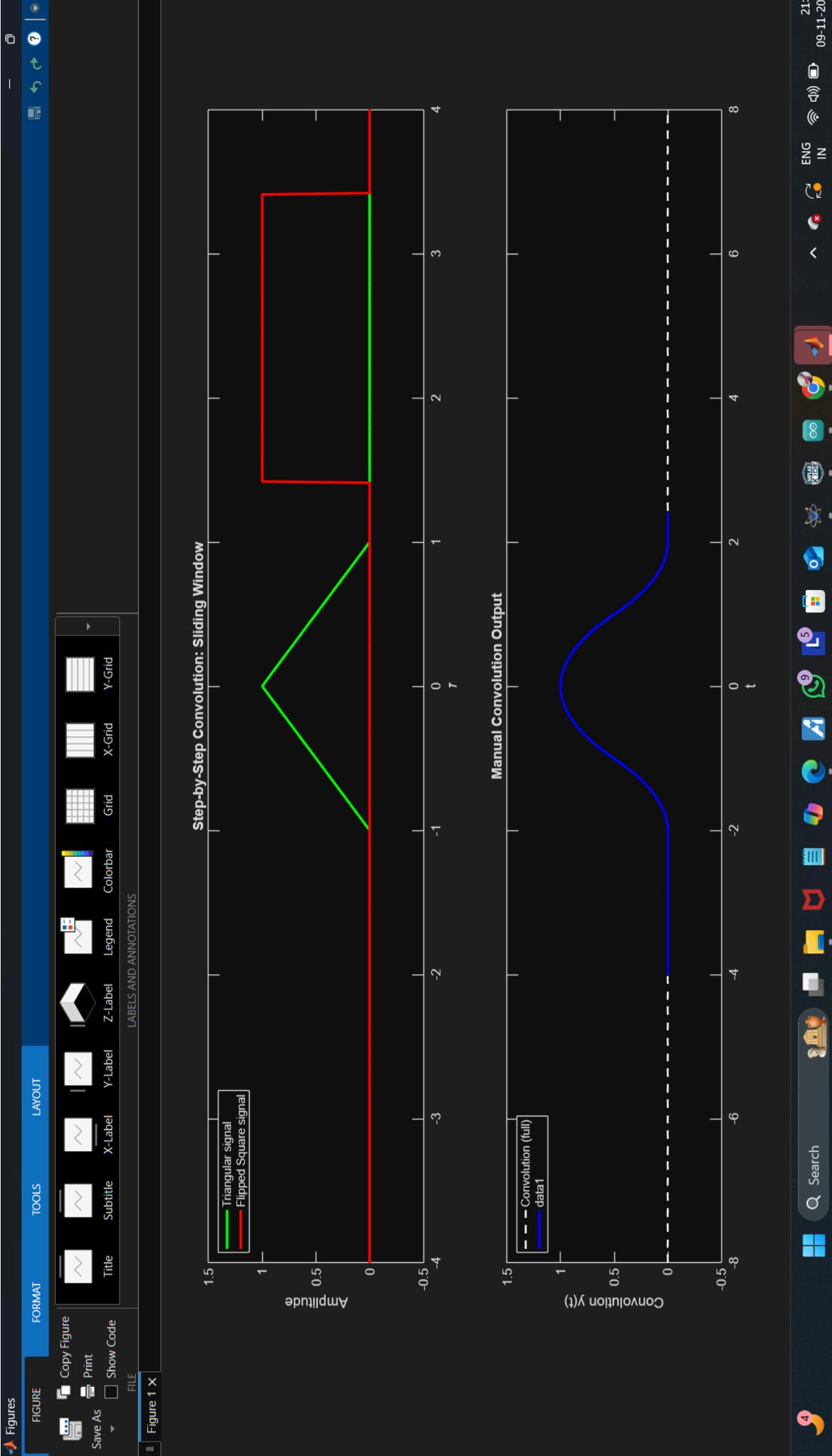
FILE NAVIGATE CODE ANALYZE TEST SECTION

C:\Users\saman\OneDrive\Documents\MATLAB\untitled.m

```
51 y_anim = zeros(size(t)); % Reset convolution output for each repetition
52
53 % Clear the previous convolution plot for a fresh start
54 if rep > 1
55     delete(hConv); % Remove the old line handle
56     hConv = plot(NaN, NaN, 'b', 'LineWidth', 2); % Create a new placeholder
57 end
58
59 for w = 1:length(t)
60     shift = t(w);
61     % Compute the flipped and shifted version of h: h(shift - tau)
62     h_shift = rectpuls(shift - t, rect_width); % unit-height rectangle flipped & shifted
63
64     % Compute convolution integral at this shift (area under product x.*h_shift)
65     y_anim(w) = trapz(t, x .* h_shift); % numeric integration (continuous conv)
66
67     % Update the sliding signal (red) plot
68     set(hRect, 'YData', h_shift);
69     % Update convolution output plot up to current shift
70     set(hConv, 'XData', t(1:w), 'YData', y_anim(1:w));
71
72     drawnow; % update figure
73     % pause(0.01); % optional slow-down for visualization
74 end
75
76 % Optional: Overlay final convolution result for reference after each cycle
77 plot(t_conv, y_conv, 'w--', 'LineWidth', 1.5); % Changed to blue dashed line for better visibility
78 legend('Convolution (animation)', 'Convolution (full)', 'Location', 'northwest');
79
80 % Pause before starting the next cycle
81 pause(1);
82 end
```

Ready ...

4 5



## 2.Demonstration of Frequency Spectrum of a Square Wave using Arduino and 555 Timer:

### Abstract

This project focuses on analyzing the frequency spectrum of a periodic non-sinusoidal signal using an Arduino microcontroller. A square wave signal is generated with a 555 timer IC circuit and interfaced with Arduino to visualize and analyze its frequency components. The experiment provides practical insight into Fourier analysis concepts discussed in the Signals and Systems course.

### Components Used

Component	Specification / Description
Arduino Uno	Microcontroller board used for signal analysis
555 Timer IC	Used to generate the square wave signal
Resistors	For setting the time constant in the 555 timer circuit
Capacitors	Used with 555 timer for frequency adjustment
Breadboard	For circuit prototyping
Connecting Wires	To connect the circuit components
Oscilloscope / Serial Plotter	To visualize the waveform and spectrum

### Circuit Description:

The circuit consists of a 555 timer configured in astable mode to produce a continuous square wave signal. The frequency of oscillation depends on the resistors and capacitor connected to the timer. The output of the 555 timer is fed to the Arduino analog input pin, where it is sampled and analyzed. The Arduino then computes the frequency components using discrete Fourier transform algorithms or displays the waveform using the Serial Plotter for visual observation.

### Working Principle :

The square wave generated by the 555 timer contains multiple harmonics, which form its frequency spectrum. The Arduino reads the signal and visualizes it, showing fundamental and harmonic frequencies. The experiment demonstrates how a non-sinusoidal periodic waveform can be decomposed into a series of sinusoidal components, validating Fourier theorem principles.

## Code :

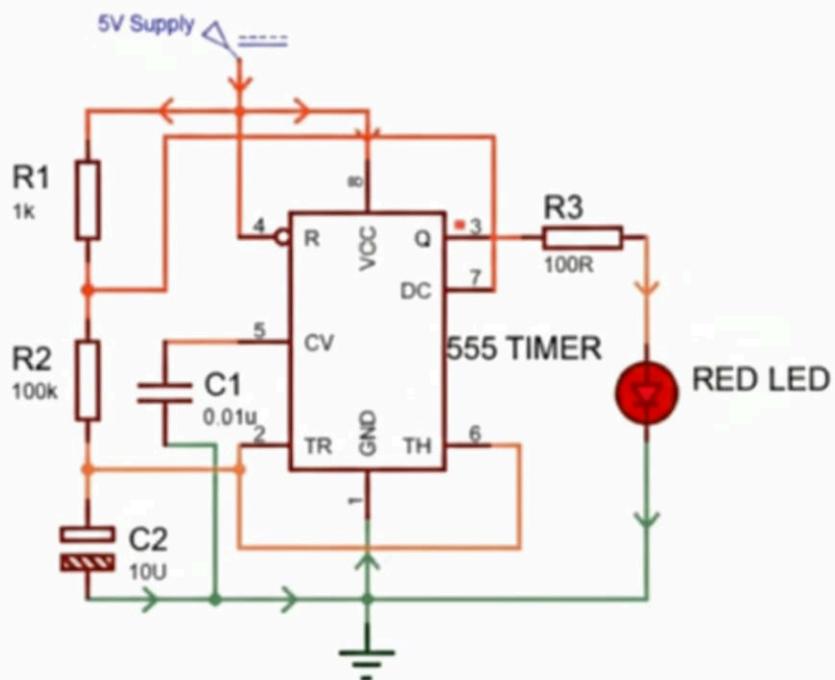
### Arduino code :

```
1 // const int signalPin = A0;           // Analog pin to read from
2 // const int sampleCount = 500;         // Number of samples to collect
3 // int samples[sampleCount];          // Array to store samples
4 // unsigned long samplingInterval = 100; // Microseconds between samples (10 kHz)
5
6 // void setup() {
7 //   Serial.begin(115200);             // Start serial communication
8 // }
9
10 // void loop() {
11 //   // Step 1: Sample the signal and store in array
12 //   for (int i = 0; i < sampleCount; i++) {
13 //     samples[i] = analogRead(signalPin); // Read and store
14 //     delayMicroseconds(samplingInterval); // Wait before next sample
15 //   }
16
17 //   // Step 2: Print the array to Serial Monitor
18 //   for (int i = 0; i < sampleCount; i++) {
19 //     Serial.println(samples[i]);       // Print each sample
20 //   }
21
22 //   delay(1000); // Optional pause before next batch
23 // }
24
25
26
27 const int signalPin = A0;           // Analog pin connected to 555 output
28 const int sampleCount = 500;         // Number of samples
29 int samples[sampleCount];          // Array to store samples
30 unsigned long samplingInterval = 50; // Microseconds between samples (~20 kHz)
31
32 void setup() {
33   Serial.begin(115200);             // Start serial communication
34 }
35
36 void loop() {
37   // Step 1: Sample the signal
38   for (int i = 0; i < sampleCount; i++) {
39     samples[i] = analogRead(signalPin); // Read analog value (0-1023)
40     delayMicroseconds(samplingInterval); // Wait before next sample
41   }
42
43   // Step 2: Stream data for Serial Plotter
44   for (int i = 0; i < sampleCount; i++) {
45     Serial.println(samples[i]);       // Print each sample
46   }
47
48   delay(500); // Optional pause before next batch
49 }
```

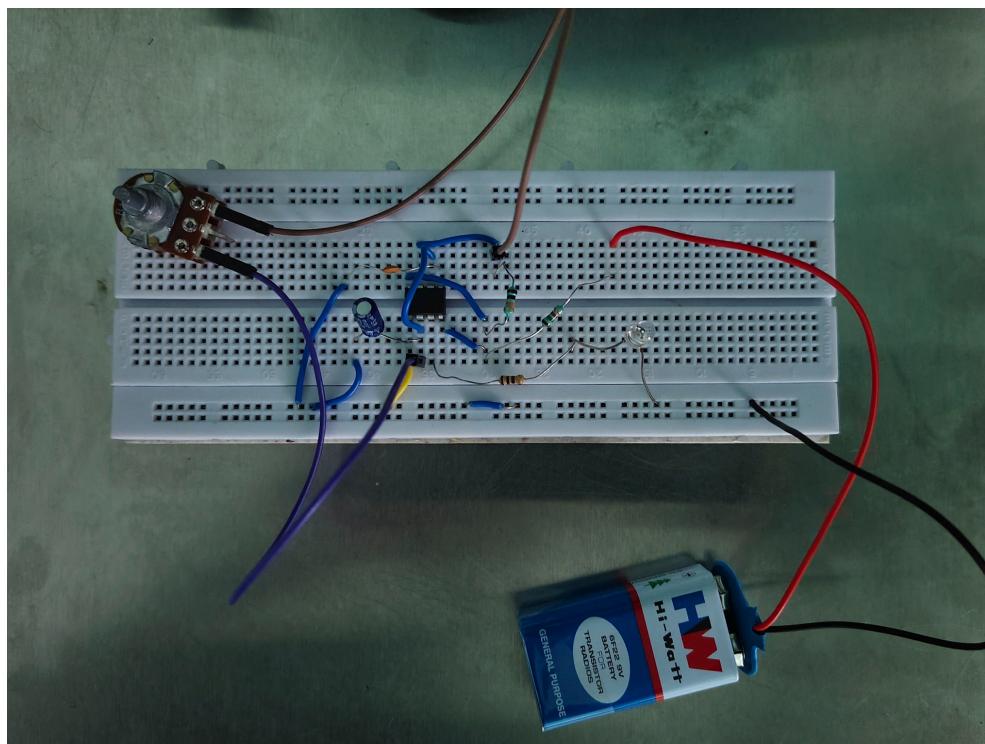
Python code:

```
1 import serial
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 # Adjust COM port and baud rate
6 ser = serial.Serial('COM3', 115200) # Replace 'COM3' with your port
7 samples = []
8 sample_count = 500 # Match with Arduino
9
10 # Read samples from Serial
11 while len(samples) < sample_count:
12     line = ser.readline().decode().strip()
13     if line.isdigit():
14         samples.append(int(line))
15
16 ser.close()
17
18 voltages = [val * (5.0 / 1023.0) for val in samples]
19
20
21 sampling_rate = 10000 # Hz
22 fft_result = np.fft.fft(samples)
23 frequencies = np.fft.fftfreq(len(samples), 1/sampling_rate)
24
25 # Only take the positive half
26 half = len(samples) // 2
27 fft_magnitude = np.abs(fft_result[:half])
28 freqs = frequencies[:half]
29
30 plt.plot(freqs, fft_magnitude)
31 plt.title("Frequency Spectrum")
32 plt.xlabel("Frequency (Hz)")
33 plt.ylabel("Amplitude")
34 plt.grid(True)
35 plt.show()
```

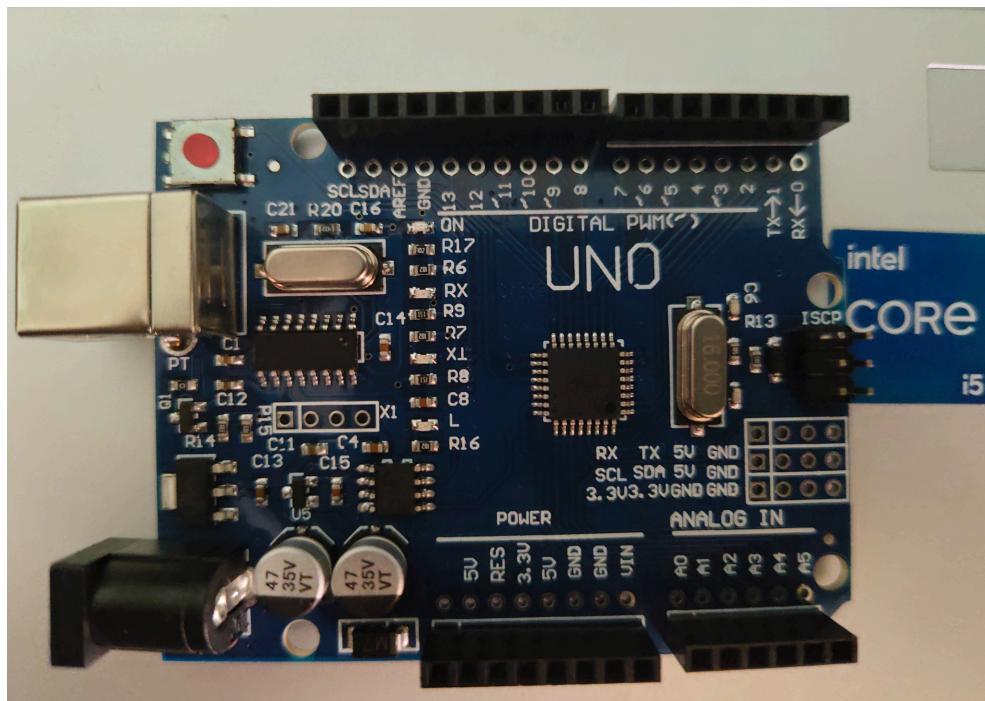
Circuit diagram:



## Circuit:



## Arduino UNO:

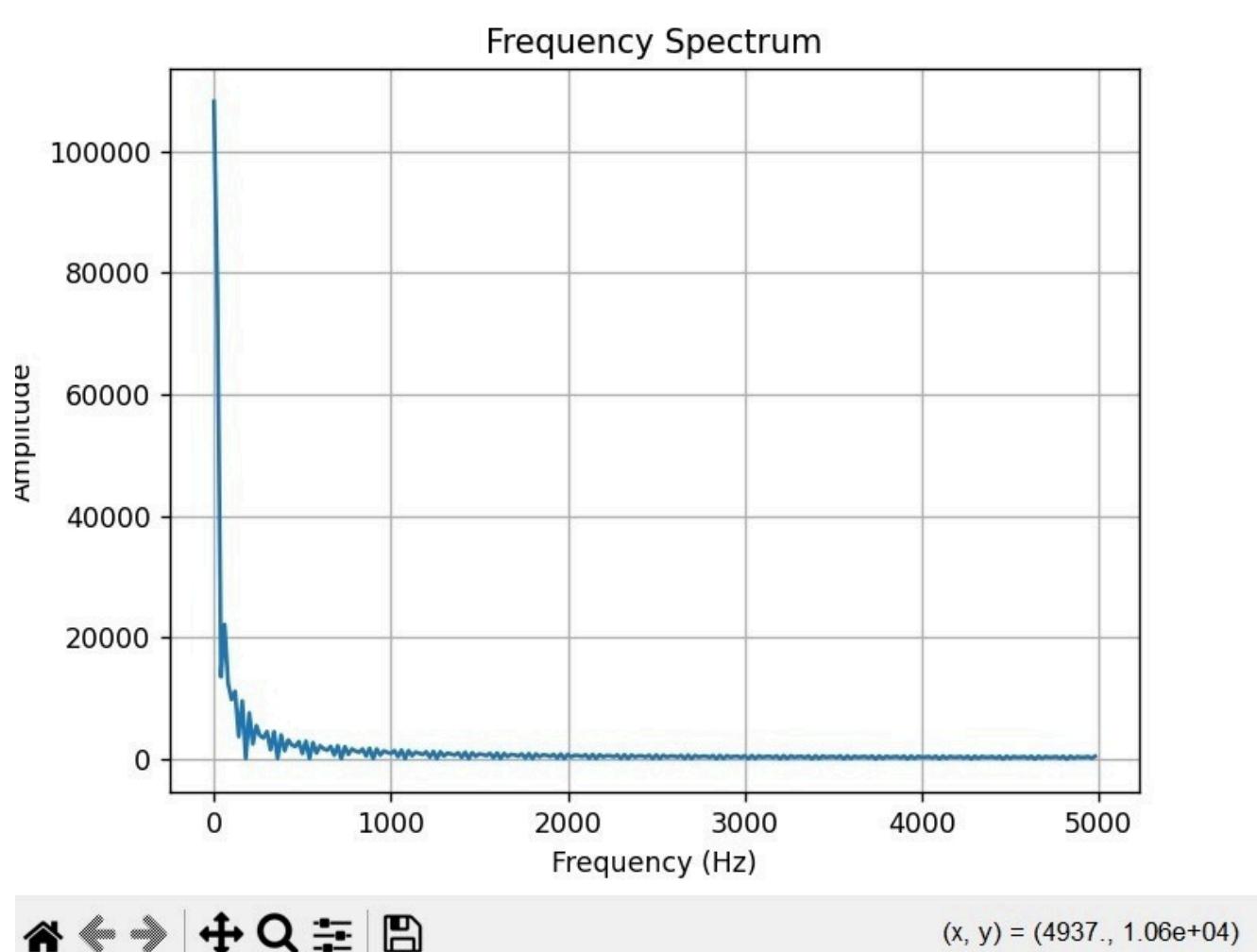


## Observations:

When the 555 timer was configured with appropriate resistor and capacitor values, a square wave of approximately 1 kHz was generated. The frequency spectrum obtained through Arduino analysis displayed odd harmonics of the fundamental frequency, confirming theoretical expectations for a square wave.

# Project Output

Figure 1



## **Conclusion:**

The project successfully demonstrated the frequency spectrum of a periodic non-sinusoidal signal using a 555 timer and Arduino. It provided hands-on experience in signal generation, sampling, and spectral analysis, enhancing understanding of core Signal and Systems concepts like harmonics, Fourier analysis, and time-frequency relationships.

## **Acknowledgment:**

We would like to express our sincere gratitude to our faculty guide, Dr. Mithun M S, for his constant support, guidance, and encouragement throughout this project. We also thank the Department of Electrical and Electronics Engineering, NIT Calicut, for providing resources and facilities to complete this work successfully.