

Report 3 : 95%

Prelab 3 : 98%

Prelab 4 : 98%

Constructor University Bremen

Natural Science Laboratory

Signals And Systems

Fall Semester 2024

Lab Experiment 3- Fourier Series and Fourier Transform

Author: Wanzia Nambule

Instructor: Pagel Uwe

Experiment conducted by: Wanzia Nambule, Dahyun Ko

Place of execution: Teaching Lab EE

Bench 12

Date of execution: 09 October, 2024

INTRODUCTION

Objective

The experiment focused on analyzing various signals through their Fourier coefficients to gain a deeper understanding of the Fourier transform. The experiment emphasized the simulation and implementation of the Fast Fourier Transform (FFT), with comparisons made between the experimental and simulation results, highlighting any differences observed.

Theory

MATLAB was utilized to examine the Fourier transform of signals through the Fast Fourier Transform (FFT). The primary objective was to efficiently calculate the discrete Fourier transform of a sequence or dataset. Additionally, the focus was placed on understanding the concept of dBVrms.

Prelab Fourier Series and Fourier Transform

Problem 1 : Decibels

1.

$$x(t) = 5\cos(2\pi 1000t)$$

- a) Signal Amplitude=5V and Vpp is twice the peak value= 10V
- b) The V_{RMS} value of the signal is $\frac{10}{2\sqrt{2}} = 3.53V$
- c) The amplitude of the spectra peak in dBV_{RMS}

$$20\log_{10} V_{RMS} = dBV_{RMS}$$

$$20\log_{10} \frac{5}{\sqrt{2}} = 10.9691dB$$

2.

Square wave of 1 Vpp the voltage changes between -0.5 V and 0.5 V

- a) For square wave $V_{RMS} = V_{peak}$

$$V_{peak} = \frac{V_{pp}}{2} = \frac{1}{2} = 0.5V$$

$$V_{RMS} = 0.5V$$

- b) The amplitude in dBV_{RMS}

$$20\log_{10} V_{RMS} = dBV_{RMS}$$

$$20\log_{10} 0.5 = -6.021dB$$

Problem 2 : Determination of Fourier series coefficients

- 1) Determining the Fourier coefficients up to the 5th harmonic

$$f(t) = 4t^2, \quad -0.5 < t < 0.5$$

Since this is an even function, $b_n = 0$

$$a_0 = \frac{2}{T} \int_{-0.5}^{0.5} 4t^2 dt$$

$$a_0 = 2 \times \left[\frac{4t^3}{3} \right]_{-0.5}^{0.5}$$

$$a_0 = \frac{2}{3}$$

To get the coefficients of a up to the 5th harmonic, i used the matlab code below

```
>> % MATLAB code to compute Fourier coefficients a_0 and a_n up to the 5th harmonic

% Define the function f(t) = 4t^2
f = @(t) 4 * t.^2;

% Set the number of harmonics n (up to 5th harmonic) and the period limits
n_values = 0:5; % including a_0, up to the 5th harmonic
a_n = zeros(size(n_values)); % initialize a_n array
T = 1; % period
t1 = -0.5; % lower bound of the interval
t2 = 0.5; % upper bound of the interval

% Calculate a_0
a_n(1) = integral(f, t1, t2); % a_0 = integral(f(t), from -0.5 to 0.5)

% Compute a_n for n = 1 to 5
for k = 2:length(n_values) % Start from 2 since a_n(1) is a_0
    n = n_values(k);
    integrand = @(t) f(t) .* cos(2 * pi * n * t / T); % function to integrate
    a_n(k) = 2 * integral(integrand, t1, t2); % numerical integration
end

% Display the results
disp('Fourier coefficients a_n (a_0 to a_5):')
disp(a_n)

% Optional: plot the Fourier coefficients
figure;
stem(n_values, a_n, 'filled');
title('Fourier Coefficients a_n (up to 5th harmonic)');
xlabel('n');
ylabel('a_n');
Fourier coefficients a_n (a_0 to a_5):
    0.3333    -0.4053     0.1013    -0.0450     0.0253    -0.0162
```

To get the coefficients of c up to the 5th harmonic, i used the matlab code below

```

>> % MATLAB code to compute Fourier coefficients a_n up to the 5th harmonic

% Define the function f(t) = 4t^2
f = @(t) 4 * t.^2;

% Set the number of harmonics n (up to 5th harmonic) and the period limits
n_values = 1:5; % only up to the 5th harmonic
a_n = zeros(size(n_values)); % initialize a_n array
T = 1; % period
t1 = -0.5; % lower bound of the interval
t2 = 0.5; % upper bound of the interval

% Compute a_n for each n
for k = 1:length(n_values)
    n = n_values(k);
    integrand = @(t) f(t) .* cos(2 * pi * n * t / T); % function to integrate
    a_n(k) = 2 * integral(integrand, t1, t2); % numerical integration
end

% Display the results
disp('Fourier coefficients a_n up to the 5th harmonic:')
disp(a_n)

% Optional: plot the Fourier coefficients
figure;
stem(n_values, a_n, 'filled');
title('Fourier Coefficients a_n (up to 5th harmonic)');
xlabel('n');
ylabel('a_n');

```

Fourier coefficients a_n up to the 5th harmonic:

-0.4053	0.1013	-0.0450	0.0253	-0.0162
---------	--------	---------	--------	---------

- 2) Using MatLab to plot the original function and the inverse Fourier transform and Putting both graphs into the same diagram.
For the matlab code below

```

>> % Define the function f(t) = 4t^2
f = @(t) 4 * t.^2;

% Set the period and bounds
T = 1; % Period
t1 = -0.5; % Lower bound
t2 = 0.5; % Upper bound

% Number of harmonics
n_harmonics = 5; % Change this for more harmonics
n_values = 0:n_harmonics; % Including a_0
a_n = zeros(size(n_values)); % Initialize a_n array

% Calculate a_0
a_n(1) = integral(f, t1, t2); % a_0 = integral(f(t), from -0.5 to 0.5)

% Compute a_n for n = 1 to n_harmonics
for k = 2:length(n_values) % Start from 2 since a_n(1) is a_0
    n = n_values(k);
    integrand = @(t) f(t) .* cos(2 * pi * n * t / T); % Function to integrate
    a_n(k) = 2 * integral(integrand, t1, t2); % Numerical integration
end

% Define time vector for plotting
t_plot = linspace(-0.5, 0.5, 1000); % 1000 points for smooth plot

% Original function values
f_values = f(t_plot);

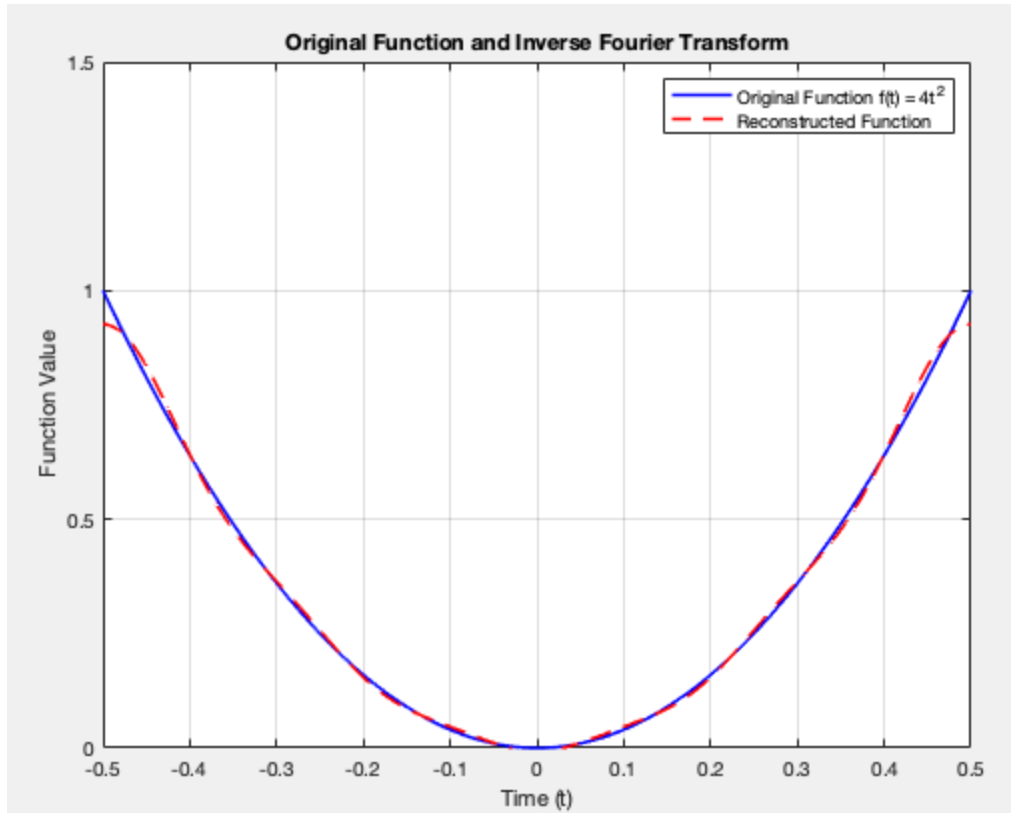
% Reconstructed function using inverse Fourier series
f_reconstructed = zeros(size(t_plot));
for n = 0:n_harmonics
    f_reconstructed = f_reconstructed + a_n(n+1) * cos(2 * pi * n * t_plot / T);
end

% Plotting the original function and the reconstructed function
figure;
plot(t_plot, f_values, 'b', 'LineWidth', 1.5); % Original function in blue
hold on;
plot(t_plot, f_reconstructed, 'r--', 'LineWidth', 1.5); % Reconstructed function in red dash
hold off;

% Add titles and labels
title('Original Function and Inverse Fourier Transform');
xlabel('Time (t)');
ylabel('Function Value');
legend('Original Function f(t) = 4t^2', 'Reconstructed Function');
grid on;
axis([-0.5 0.5 0 1.5]); % Adjust axis limits as needed

```

The plot below



Problem 3 : FFT of a Square/Rectangular Wave

- 1) A square wave of 1 ms period, 2 Vpp amplitude, no offset, and duty cycle 50% was generated on matlab using the square function . The frequency of 200 kHz was used as the sampling frequency for the square wave.

Matlab code use for the square wave below

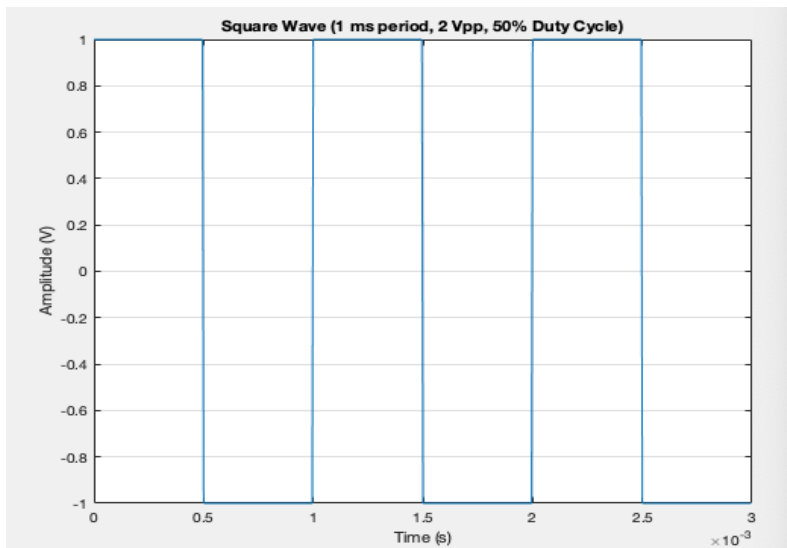
```
>> % Parameters
fs = 200e3; % Sampling frequency (200 kHz)
T = 1e-3; % Period (1 ms)
A = 1; % Amplitude (1 V peak, 2 Vpp means peak-to-peak = 2 V)
duty_cycle = 50; % Duty cycle 50%

% Time vector for 3 cycles (3 ms)
t = 0:1/fs:3*T; % Time from 0 to 3*T with step size of 1/fs
t(end) = []; % Remove the last sample to keep the length consistent

% Generate square wave
square_wave = A * square(2 * pi * t / T, duty_cycle); % Scale the am

% Plot square wave in time domain
figure;
plot(t, square_wave);
title('Square Wave (1 ms period, 2 Vpp, 50% Duty Cycle)');
xlabel('Time (s)');
ylabel('Amplitude (V)');
grid on;
```

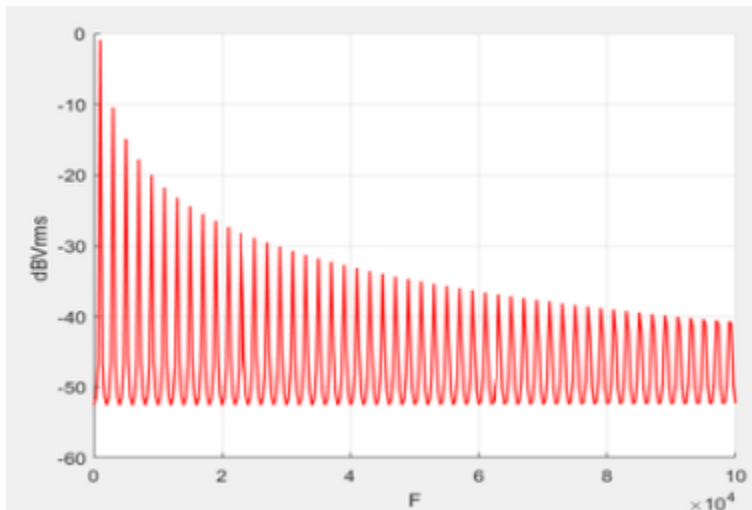
2) Square wave produced by the code above



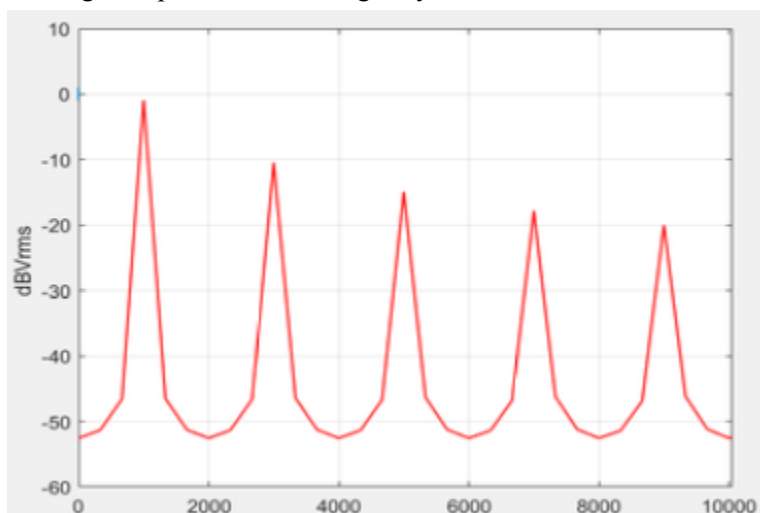
3) Obtaining the FFT spectrum using Matlab FFT function. Making the FFT length to be the length of the square wave data vector.

```
1      sf = 200000;  
2      fn = sf / 2;  
3      T = 0.001;  
4  
5      t = 0:1/ sf :3* T ;  
6  
7      x = square (2* pi *1000* t ,50) ;  
8      n = length ( x ) ;  
9  
10     plot (t,x, 'LineWidth',1);  
11     hold on  
12     Y = 2 * (( abs ( fft ( x ) ) ) / n ) ;  
13     Y = Y (1:(( n +1) /2) ) ;  
14     d = 20* log10 ( Y / sqrt (2) ) ;  
15     f = linspace (0 , fn , length ( d ) ) ;  
16  
17     plot (f ,d , 'r','LineWidth',1) ;  
18     xlim ([0 (3200* pi)])  
19  
20     xlabel ('F')  
21     ylabel ('dBVrms ')  
22     grid on
```

- 4) Plotting the single-sided amplitude spectrum in dBVrms.



- 5) Plotting the spectrum including only the first four harmonics in dBVrms



- 6) Repeating the previous steps using 20% and 33% duty cycles, respectively and keeping period and amplitude constant.

Plotting 20% duty cycle yields we get the following time domain and frequency plots:

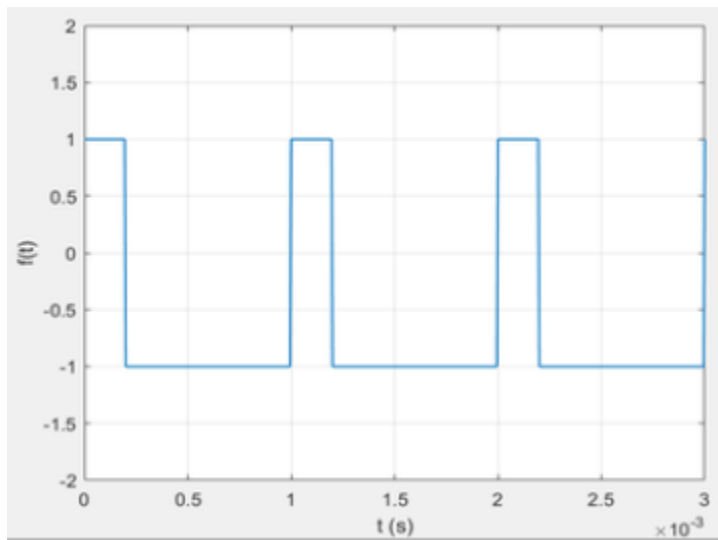
The matlab code of the square code is:

```

1      samplfreq =200000;
2      T = 0.001;
3
4      t = 0:1/ samplfreq :3* T ;
5
6      x = square (2* pi *1000* t ,20) ;
7      plot (t ,x , 'LineWidth' ,1) ;
8      ylim ([ -2 2])
9
10     xlabel ('t (s)')
11     ylabel ('f(t)')
12     grid on

```

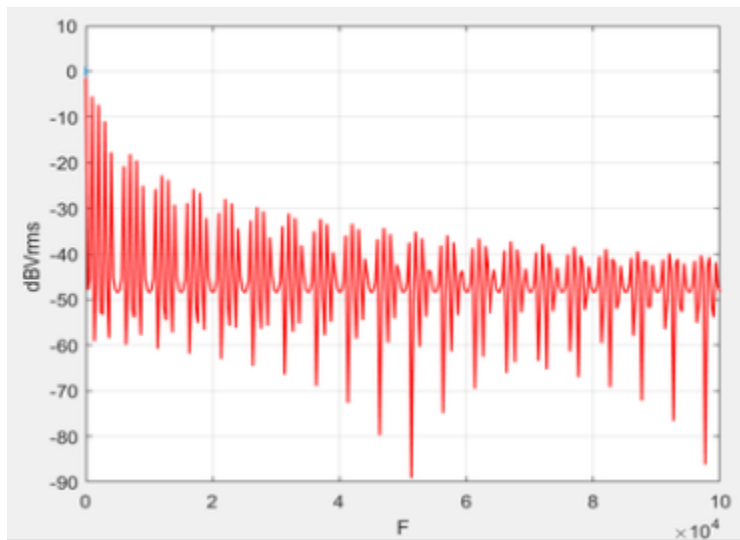

The square wave of the code above



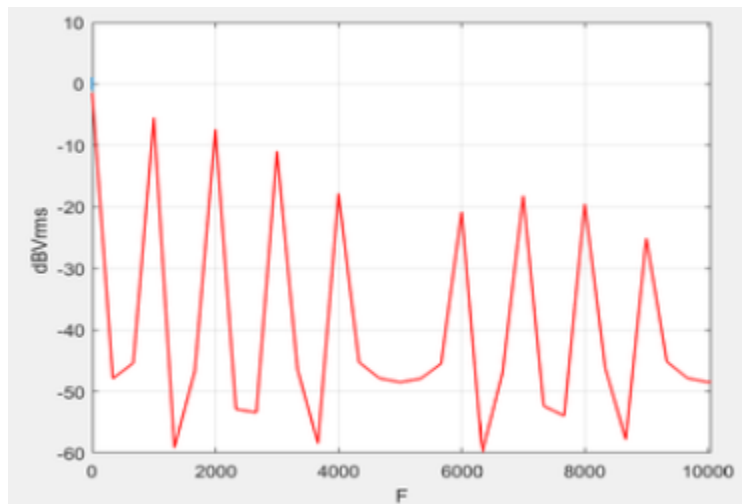
Matlab code of the frequency spectrum

```
sf = 200000;  
fn = sf / 2;  
T = 0.001;  
  
t = 0:1/ sf :3* T ;  
  
x = square (2* pi *1000* t ,20) ;  
n = length ( x ) ;  
  
plot (t,x, 'Linewidth',1);  
hold on  
Y = 2 * (( abs ( fft ( x ) ) ) / n ) ;  
Y = Y (1:(( n +1) /2) ) ;  
d = 20* log10 ( Y / sqrt (2) ) ;  
f = linspace (0 , fn , length ( d ) ) ;  
  
plot (f ,d , 'r', 'Linewidth',1) ;  
%xlim ([0 (3200* pi)])  
  
xlabel ('F')  
ylabel ('dBVrms ')  
grid on
```

20% square wave



The vector spectrum of the 20% duty cycle is below

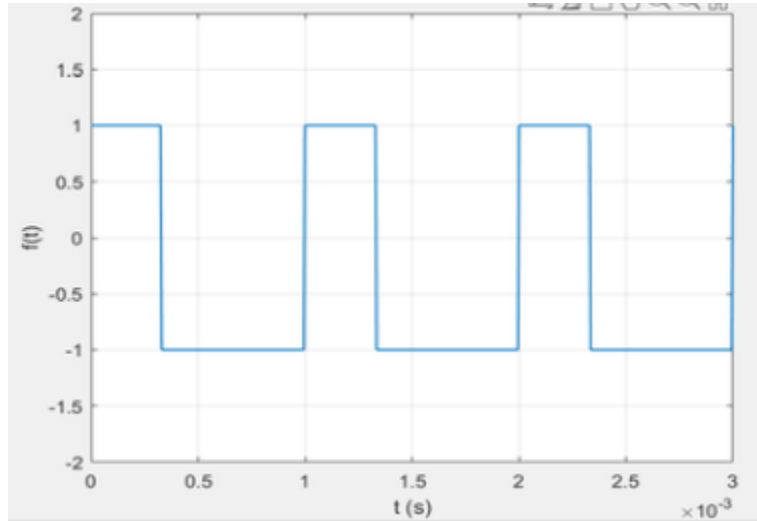


Plotting 33% duty cycle yields we get the following time domain and frequency plots:

The matlab code for the square wave of 33% duty cycle is below

```
samplfreq = 200000;  
T = 0.001;  
  
t = 0:1/ samplfreq :3* T ;  
  
x = square (2* pi *1000* t ,33) ;  
plot (t ,x , 'LineWidth' ,1) ;  
ylim ([ -2 2])  
  
xlabel ('t (s)')  
ylabel ('f(t)')  
grid on
```

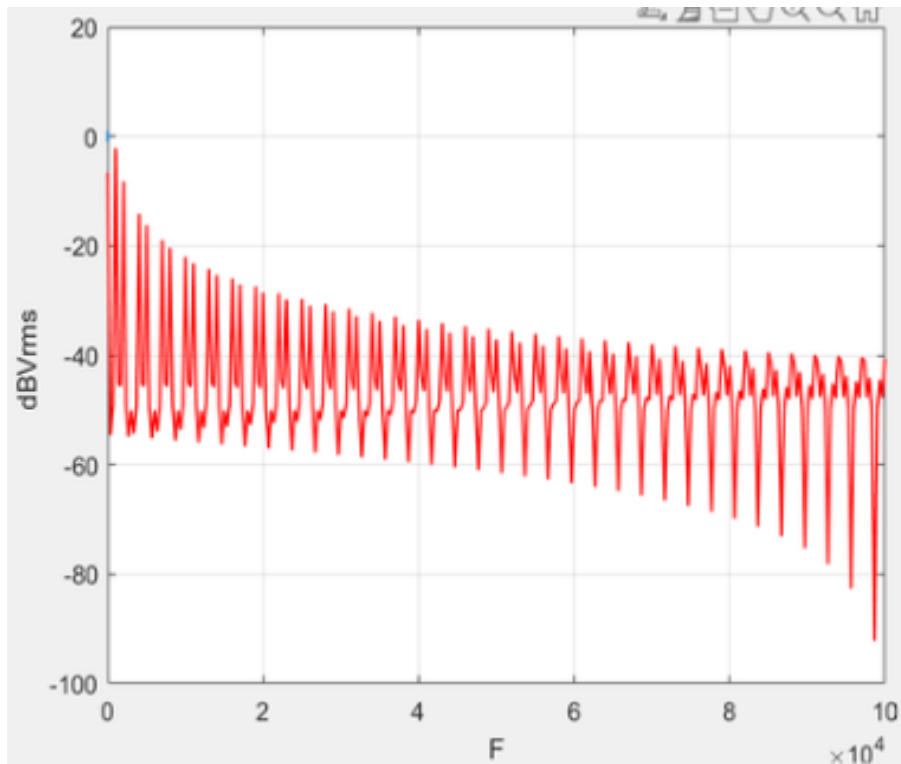
The square wave of the 33% duty cycle is below



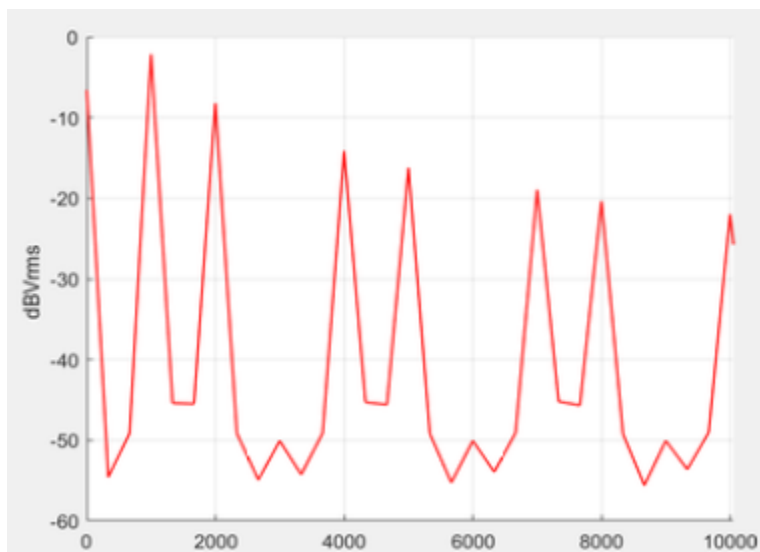
Matlab code for frequency and vector signal of 33% duty cycle

```
sf = 200000;  
fn = sf / 2;  
T = 0.001;  
  
t = 0:1/ sf :3* T ;  
  
x = square (2* pi *1000* t ,33) ;  
n = length ( x ) ;  
  
%plot (t,x, 'LineWidth ',1);  
hold on  
Y = 2 * (( abs ( fft ( x ) ) ) / n ) ;  
Y = Y (1:(( n +1) /2) ) ;  
d = 20* log10 ( Y / sqrt (2) ) ;  
f = linspace (0 , fn , length ( d ) ) ;  
  
plot (f ,d , 'r','LineWidth',1) ;  
xlim ([0 (3200* pi)])  
  
xlabel ('F')  
ylabel ('dBVrms ')  
grid on
```

The frequency of 33% duty cycle



The vector form of the 33% of duty cycle



Combining the codes above to compare the spectra of the 3 duty circles

```
>> % Parameters
fs = 200000; % Sampling frequency
fn = fs / 2; % Nyquist frequency
T = 0.001; % Period (1 ms)

t = 0:1/fs:3*T; % Time vector for 3 cycles

% Square wave for 50% duty cycle
x = square(2 * pi * 1000 * t, 50); % 1 ms period, 50% duty cycle
n = length(x); % Length of the signal

% Plot the 50% duty cycle square wave
figure;
subplot(3, 1, 1); % Subplot 1
plot(t, x, 'LineWidth', 1); % Plot square wave
hold on;

% FFT analysis for 50% duty cycle
Y = fft(x);
Y2 = 2 * ((abs(Y(1:n/2))) / n); % Single-sided amplitude spectrum
f = linspace(0, fn, length(Y2)); % Frequency vector
d = 20 * log10(Y2 / sqrt(2)); % Convert to dBVrms

% Plot the single-sided amplitude spectrum (50% duty cycle)
plot(f, d, 'r', 'LineWidth', 1);
xlim([0 (3200 * pi)]);
xlabel('F');
ylabel('dBVrms');
grid on;

% Square wave for 33% duty cycle
subplot(3, 1, 2); % Subplot 2
x1 = square(2 * pi * 1000 * t, 33); % 33% duty cycle
n1 = length(x1); % Length of the signal

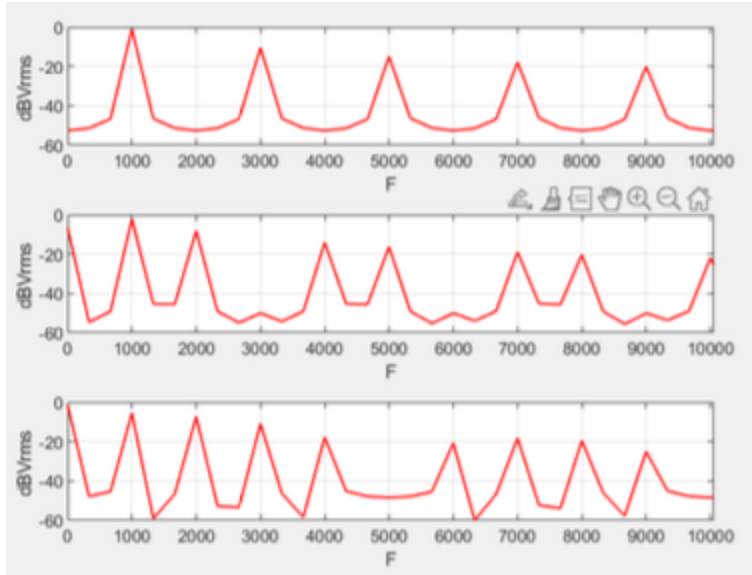
% FFT analysis for 33% duty cycle
Y1 = fft(x1);
Y2_33 = 2 * ((abs(Y1(1:n1/2))) / n1); % Single-sided amplitude spectrum
d1 = 20 * log10(Y2_33 / sqrt(2)); % Convert to dBVrms

% Plot the single-sided amplitude spectrum (33% duty cycle)
plot(f, d1, 'r', 'LineWidth', 1);
xlim([0 (3200 * pi)]);
xlabel('F');
ylabel('dBVrms');
grid on;

% Square wave for 20% duty cycle
subplot(3, 1, 3); % Subplot 3
x2 = square(2 * pi * 1000 * t, 20); % 20% duty cycle
n2 = length(x2); % Length of the signal

% FFT analysis for 20% duty cycle
Y2 = fft(x2);
Y2_20 = 2 * ((abs(Y2(1:n2/2))) / n2); % Single-sided amplitude spectrum
d2 = 20 * log10(Y2_20 / sqrt(2)); % Convert to dBVrms

% Plot the single-sided amplitude spectrum (20% duty cycle)
plot(f, d2, 'r', 'LineWidth', 1);
xlim([0 (3200 * pi)]);
xlabel('F');
ylabel('dBVrms');
grid on;
```



When narrower pulse widths, such as 20% and 33%, are employed, the spacing between consecutive harmonics decreases, making DC components more noticeable. Additionally, as more harmonics are introduced, their contributions reduce in magnitude with shorter pulse durations. For duty cycles that differ from 50%, some harmonic components—both even and odd—are present due to symmetry issues that are neither perfectly even nor strictly odd, while certain components are absent. As the pulse width decreases, the gaps between components become smaller, causing some components to disappear while others emerge.

Problem 4 : FFT of a sound sample

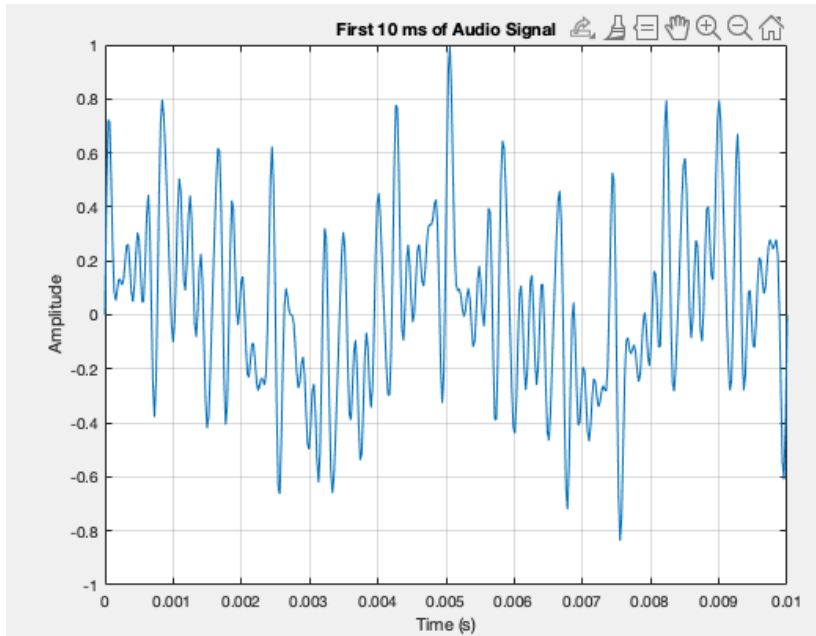
After downloading the file and Using Matlab, to read the sound file and plotting the first 10 ms of the signal we get:

```
[soundData, fs] = audioread('s_samp.wav');

% Calculate the time vector for the audio signal
time = (0:length(soundData)-1) / fs; % Time vector in seconds

% Extract the first 10 ms of the signal
time_10ms = time(time <= 0.01); % First 10 ms (0.01 seconds)
soundData_10ms = soundData(1:length(time_10ms)); % Corresponding signal data

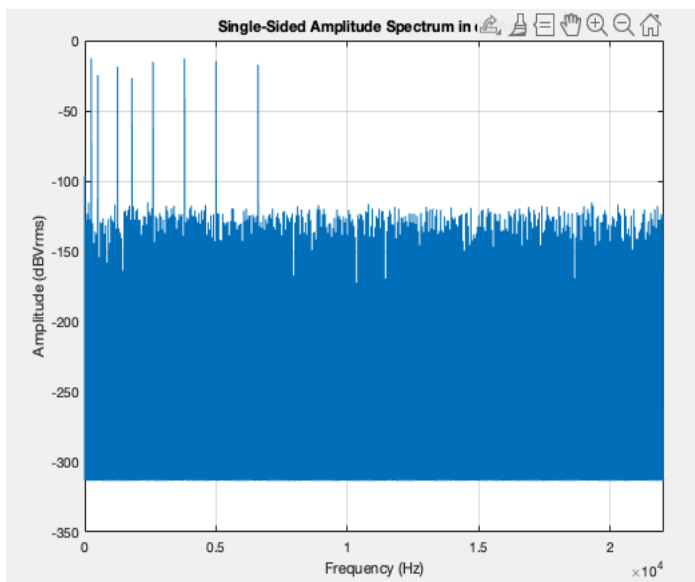
% Plot the first 10 ms of the signal
figure;
plot(time_10ms, soundData_10ms);
title('First 10 ms of Audio Signal');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
```



Using the Matlab FFT function to compute the spectrum and plot the single sided amplitude spectrum in dBVrms, we get:

```
>> % Plot the single-sided amplitude spectrum in dBVrms
figure;
plot(f1, P1_dBVrms);
title('Single-Sided Amplitude Spectrum in dBVrms');
xlabel('Frequency (Hz)');
ylabel('Amplitude (dBVrms)');
xlim([0 fs/2]); % Set x-axis limit from 0 to fs/2
grid on; % Enable grid for better visualization
```

how did you calculate for this plot??



To get the tones forming the signal above we use the matlab code below

```
>> % Set a threshold for significant tones (adjust as needed)
threshold_dBVrms = -30; % dBVrms threshold to identify tones

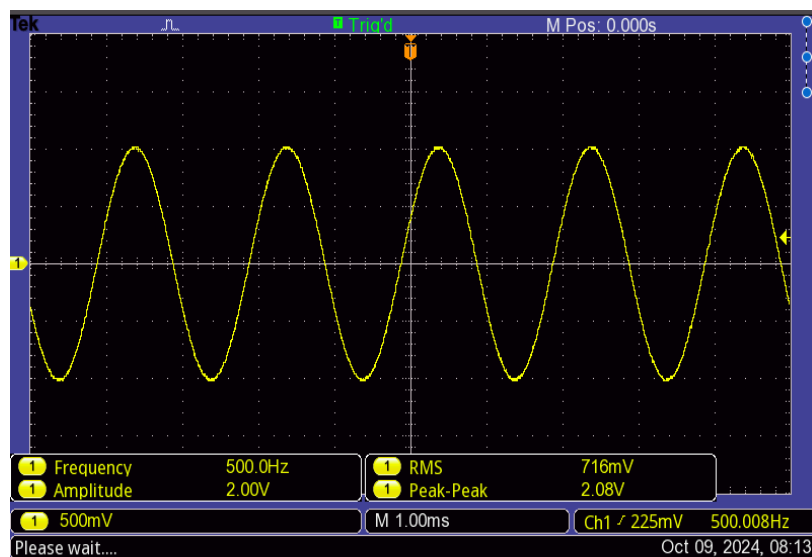
% Find significant frequencies and their amplitudes
significantIndices = find(P1_dBVrms > threshold_dBVrms);
significantFrequencies = f1(significantIndices);
significantAmplitudes = P1_dBVrms(significantIndices);

% Display the significant tones
disp('Significant Frequencies (Hz) and Amplitudes (dBVrms):');
for i = 1:length(significantFrequencies)
    fprintf('Frequency: %.2f Hz, Amplitude: %.2f dBVrms\n', ...
        significantFrequencies(i), significantAmplitudes(i));
end
Significant Frequencies (Hz) and Amplitudes (dBVrms):
Frequency: 250.00 Hz, Amplitude: -12.73 dBVrms
Frequency: 500.00 Hz, Amplitude: -24.77 dBVrms
Frequency: 1250.00 Hz, Amplitude: -18.75 dBVrms
Frequency: 1800.00 Hz, Amplitude: -26.71 dBVrms
Frequency: 2600.00 Hz, Amplitude: -15.23 dBVrms
Frequency: 3800.00 Hz, Amplitude: -12.73 dBVrms
Frequency: 5000.00 Hz, Amplitude: -14.67 dBVrms
Frequency: 6600.00 Hz, Amplitude: -17.16 dBVrms
```

EXPERIMENT SET-UP AND RESULTS

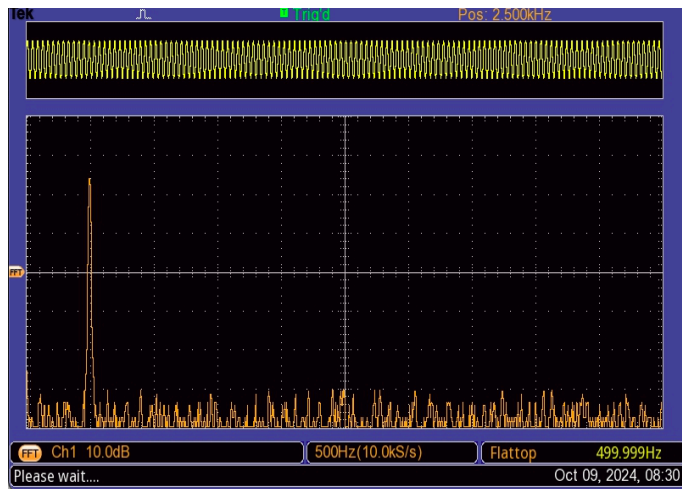
A sine wave with a frequency of 500 Hz, a 2 Vpp (peak-to-peak) amplitude, and no offset was generated using the function generator. The oscilloscope's measurement feature was then used to verify these parameters and obtain a printout of the signal in the time domain.

Below is a hardcopy showing the sine wave:

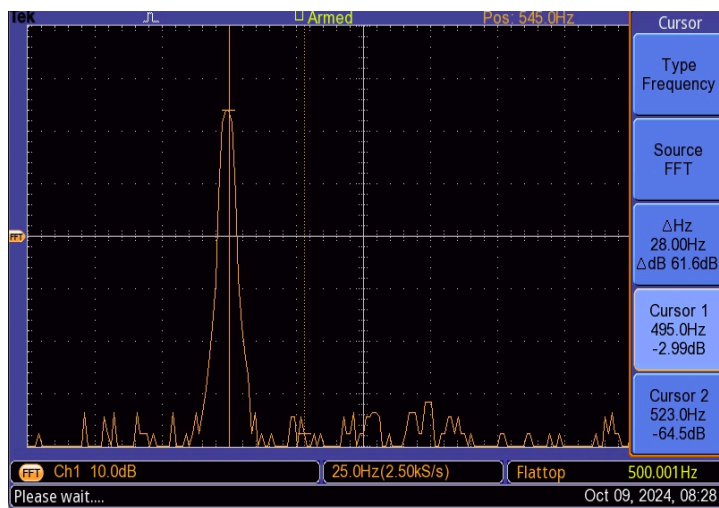


The FFT spectrum was obtained using the oscilloscope's FFT function. The cursor was used to measure the relevant properties, and hard copies of both the complete spectrum and the zoomed-in view of the spectral peak were taken.

Below is a hard copy of the complete spectra:



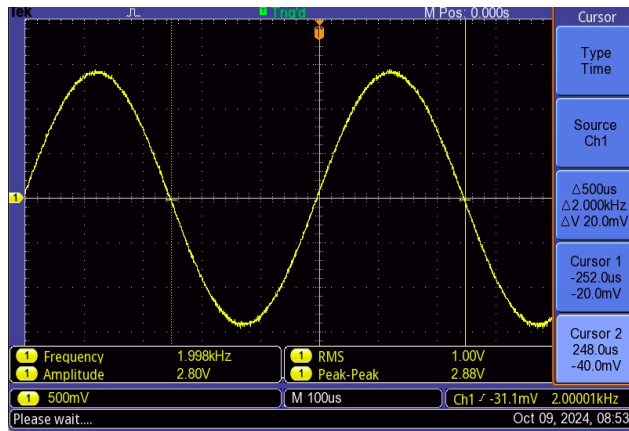
Below is a hard copy of the zoomed spectra peak:



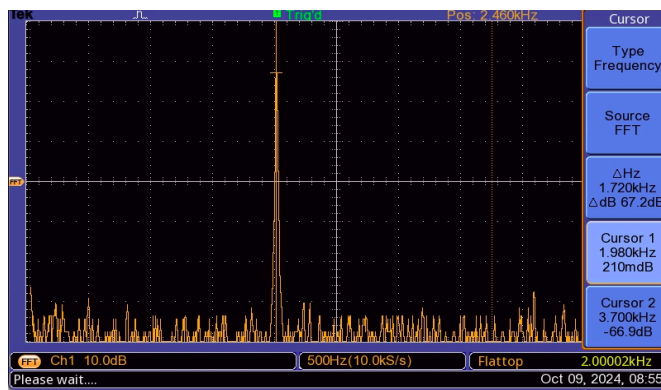
A sinusoidal wave with a 0 dB spectrum peak, a frequency of 2 KHz, and no DC offset was generated. The amplitude value was measured using the measure function and the cursors. Hard copies of both the time and frequency domain representations were obtained.

Where the Amplitude is in the picture below. Amplitude = 2.80V

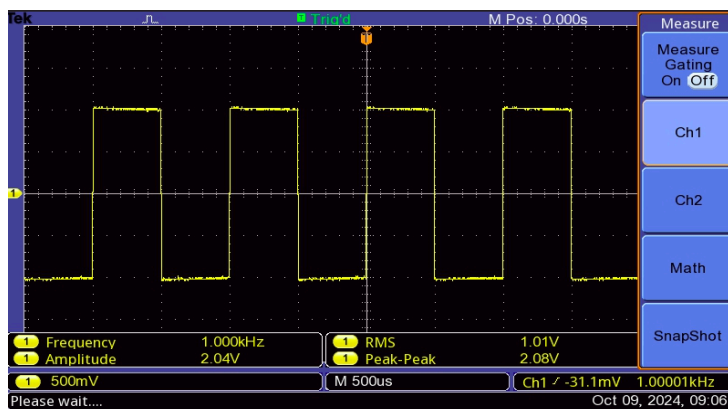
Below is the hard copy in time domain:



Below is the hard copy in frequency domain:

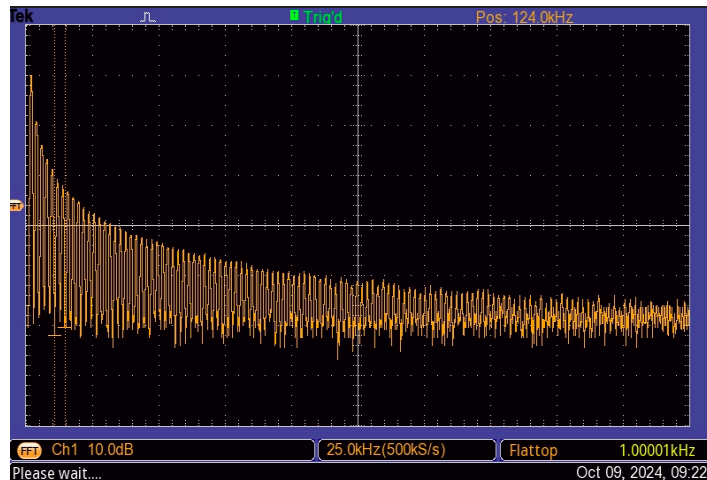


A square wave with a 1 ms period, 2 Vpp amplitude, and no offset was generated using the function generator. The properties were verified using the measure function, and a hard copy below was obtained.

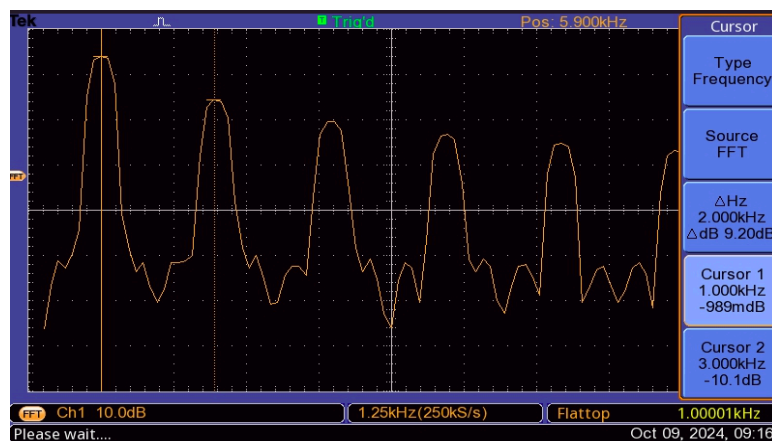


The FFT spectrum was obtained, and the FFT zoom control, with a zoom factor of up to 10, was used to accurately measure the frequency components instead of the time base (sec/div) control. Cursors were utilized to determine the amplitudes and frequencies of the fundamental and the first four harmonics. Hard copies of the FFT signal were taken.

Below is the picture of the function:

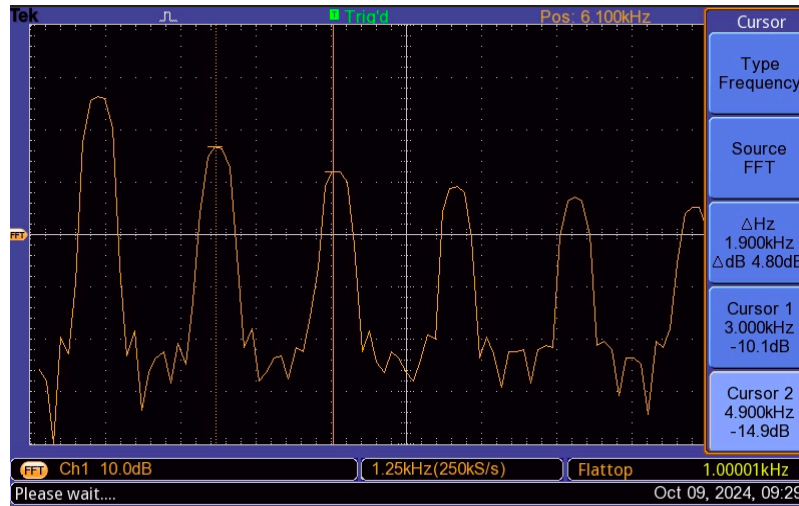


Below is the hard copy of the fundamental components:



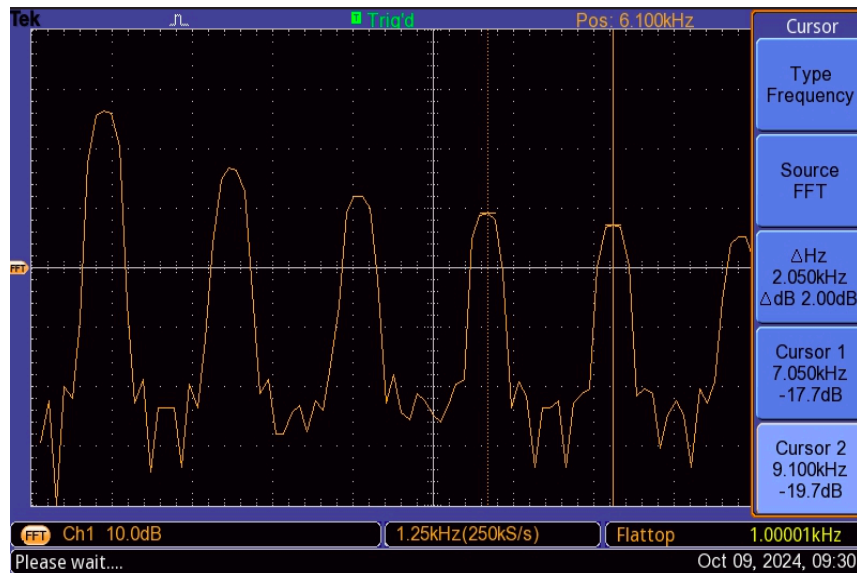
Below is the hard copy of the first and second harmonic components:

Where cursor 1 is first and cursor 2 is second harmonic



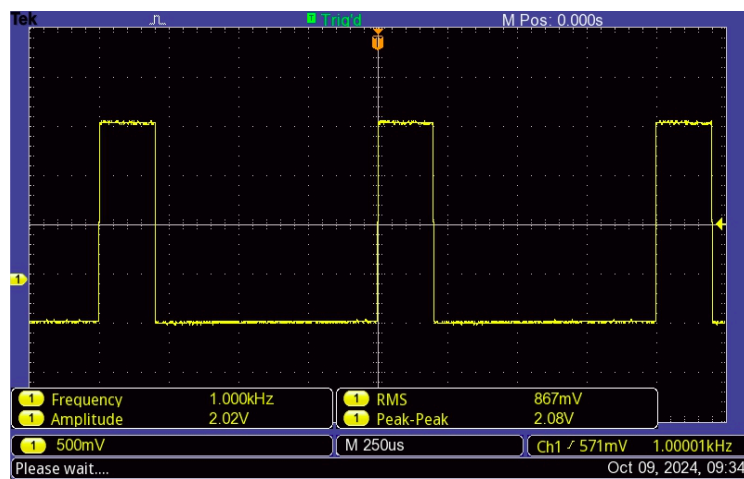
Below is the hard copy of the third and fourth harmonic components:

Where cursor 1 is third and cursor 2 is fourth

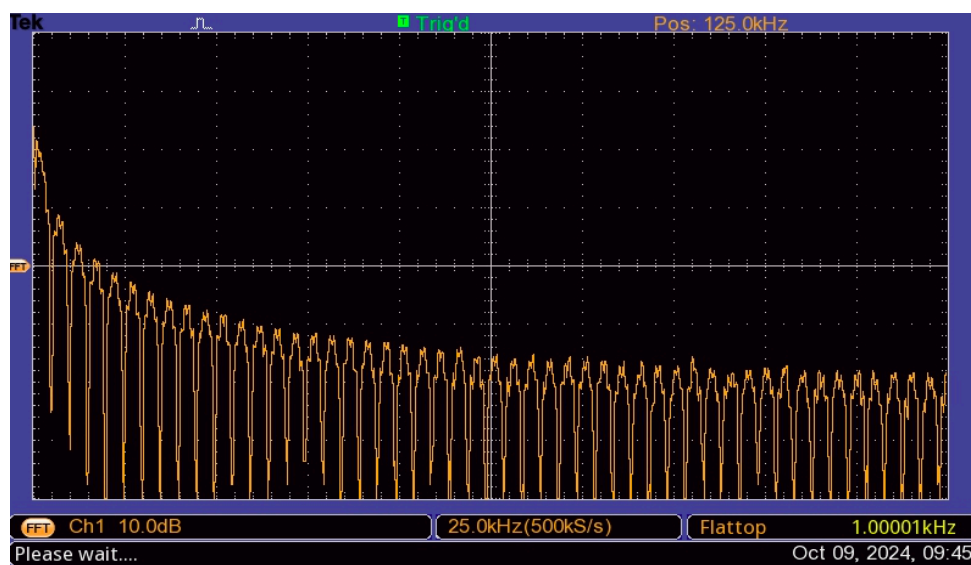


The FFT spectrum for a 20% duty cycle was obtained, and the amplitudes of the fundamental frequency and the first four harmonics were determined. Hard copies of the signal in both the time and frequency domains were taken.

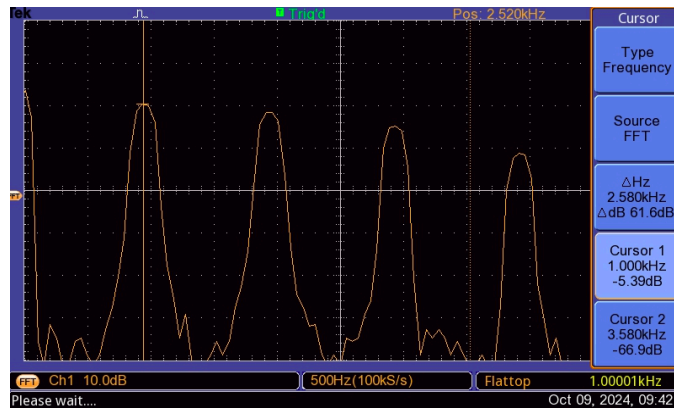
Below is the hard copy in time domain:



Below is the hard copy in frequency domain:

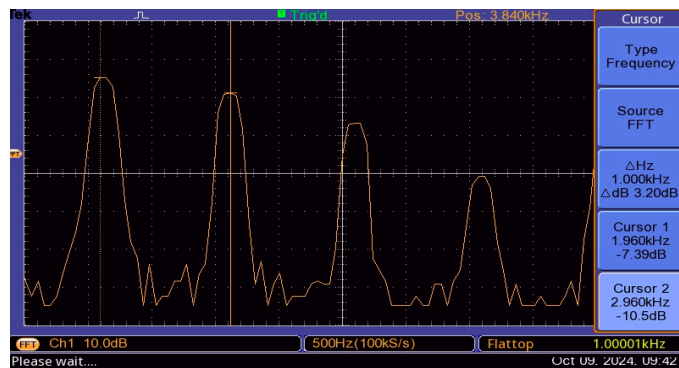


Below is the hard copy of the fundamental components:



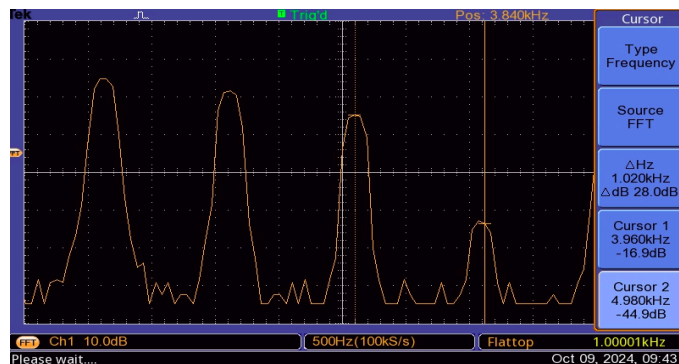
Below is the hard copy of the first and second harmonic components:

Where cursor 1 is first and cursor 2 is second harmonic

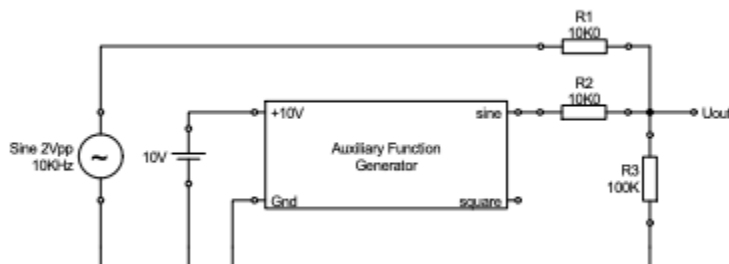


Below is the hard copy of the third and fourth harmonic components:

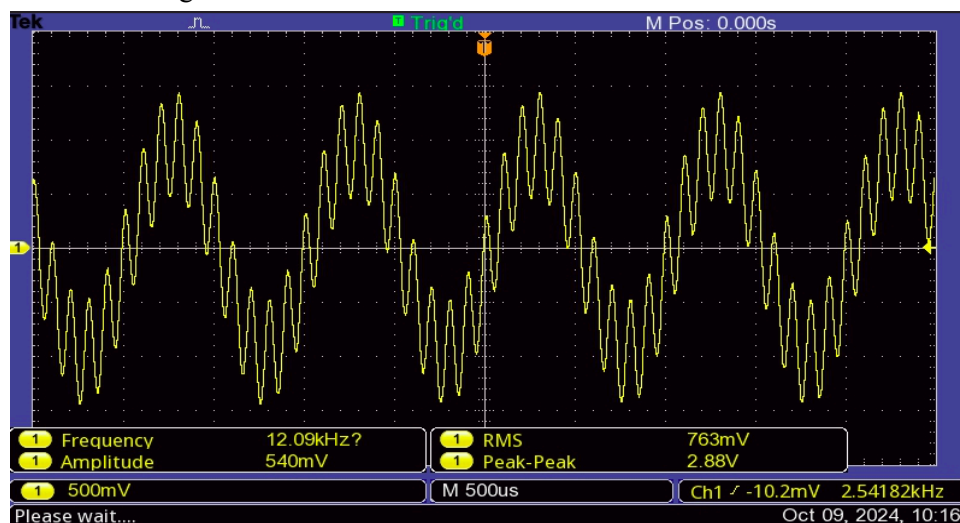
Where cursor 1 is third and cursor 2 is fourth



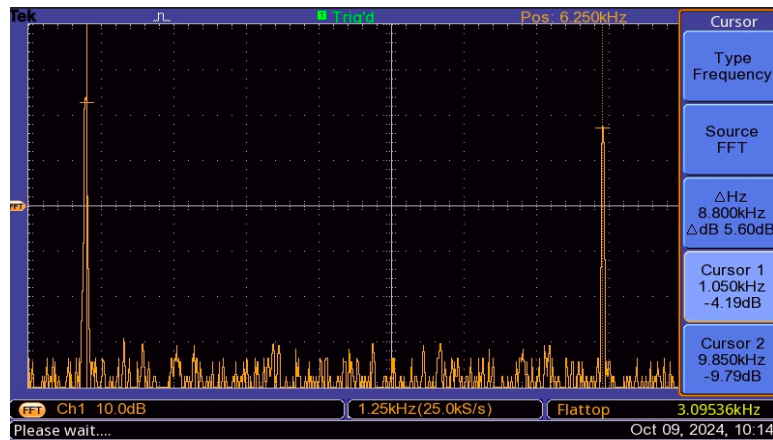
The signal from the sine output of the auxiliary signal generator was combined with a 2 Vpp, 10 KHz sinusoidal wave from the Agilent signal generator using the circuit below.



Below is the signal in time domain:



Below is the signal in frequency domain:



EVALUATION

Problem 1 : FFT of Single Tone sinusoidal wave

- 1) The reference value of the oscilloscope for 0dB is $1V_{RMS}$.
- 2) MATLAB was used to calculate the expected FFT spectrum based on the parameters provided in the execution section. The Fast Fourier Transform had a frequency of 500 Hz, an amplitude of 2Vpp, and no offset.

Below is the matlab code for 2):

```
fs = 10000; % Sampling frequency (arbitrarily set to 10 kHz for accuracy)
T = 1/fs; % Sampling period
L = 1000; % Number of samples
t = (0:L-1)*T; % Time vector

% Signal parameters
f = 500; % Frequency from the image (500 Hz)
A = 1; % Amplitude (1 V peak, since 2 Vpp = 1 V peak)
x = A * sin(2 * pi * f * t); % Sine wave

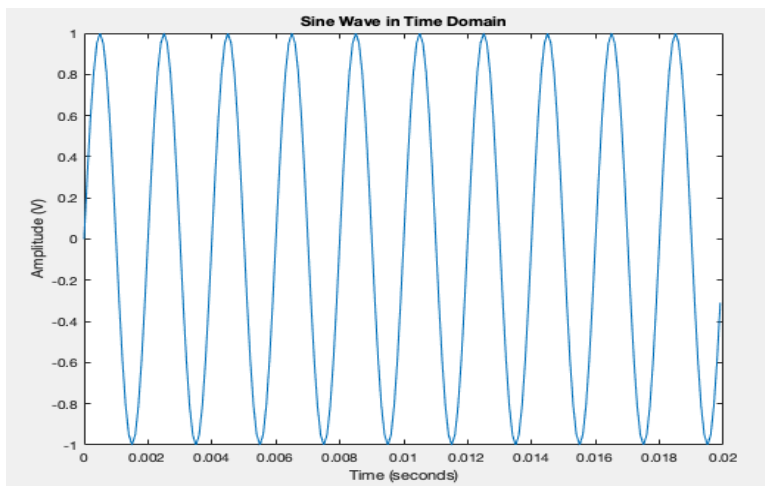
% Plot the signal in time domain (optional)
figure;
plot(t(1:200), x(1:200)); % Plot only the first 200 points for visibility
title('Sine Wave in Time Domain');
xlabel('Time (seconds)');
ylabel('Amplitude (V)');

% FFT calculation
X = fft(x);
P2 = abs(X/L); % Two-sided spectrum
P1 = P2(1:L/2+1); % Single-sided spectrum
P1(2:end-1) = 2*P1(2:end-1); % Scaling
f_axis = fs*(0:(L/2))/L; % Frequency axis

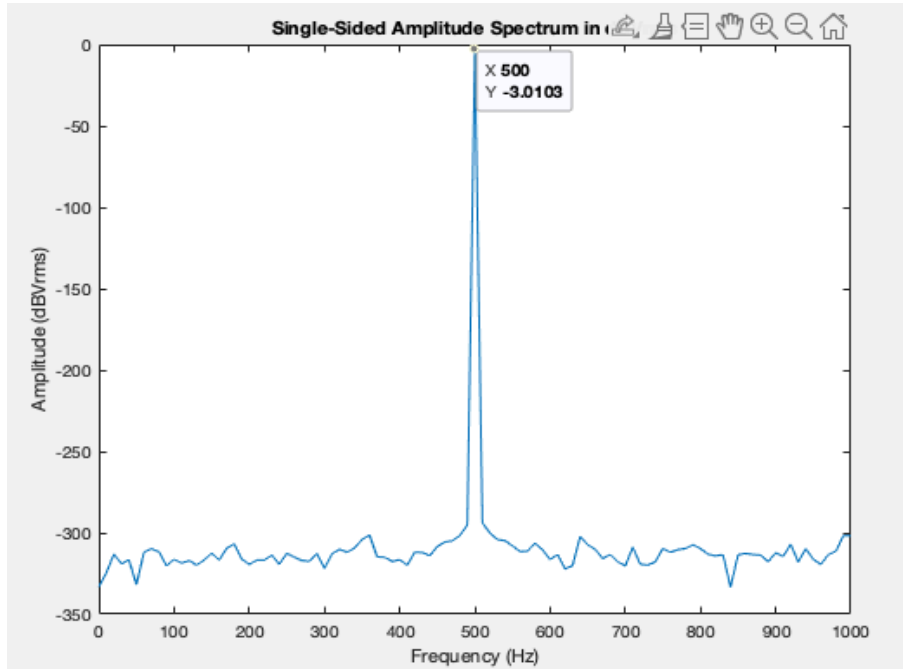
% Convert to dBVrms
P1_dBVrms = 20*log10(P1/sqrt(2));

% Plot the single-sided amplitude spectrum in dBVrms
figure;
plot(f_axis, P1_dBVrms);
title('Single-Sided Amplitude Spectrum in dBVrms');
xlabel('Frequency (Hz)');
ylabel('Amplitude (dBVrms)');
xlim([0 1000]); % Focus on frequencies between 0 and 1000 Hz
```

Below is the plot for 2) in time domain:



Below is the plot for 2) in frequency domain:



The frequency value displayed in the picture is 500 Hz at -3.0103 dB. The overall appearance of the spectrum closely resembles that generated by the oscilloscope. However, the FFT calculated in MATLAB provides an idealized representation, free from the limitations posed by instrumental errors such as those introduced by the signal generator, oscilloscope, or internal resistances.

- 3) Employing MATLAB to compute the anticipated FFT spectrum for a signal with a 0 dB peak, a frequency of 2 kHz, an amplitude of 2 Vpp, and no DC offset.

Matlab code for 3)

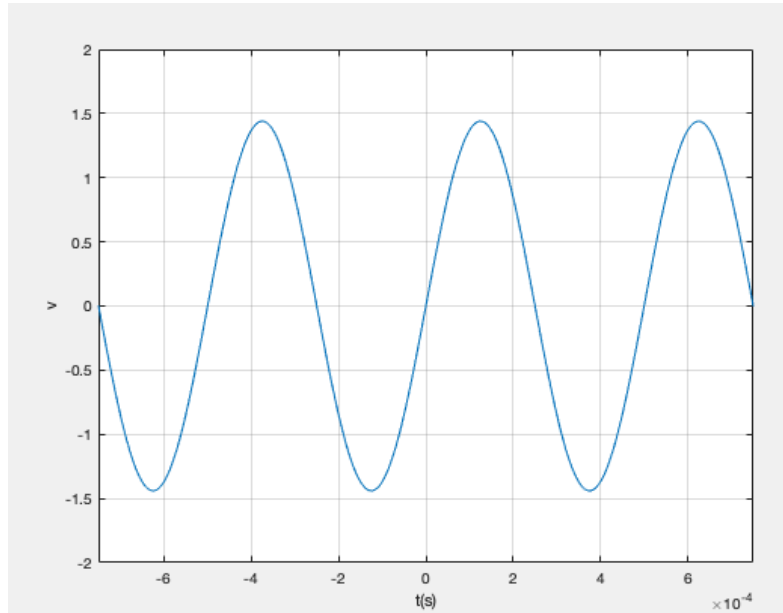
```
>> fs = 400000;           % Sampling frequency
T = 0.0005;               % Period (f = 1/T)
t = -0.5:1/fs:0.5;        % Time vector
vpp = 2.88;               % Peak-to-peak voltage from oscilloscope
y = vpp * 0.5 * sin(2 * pi * 2000 * t); % Signal generation
n = length(y);            % Number of samples

% Time-domain plot
figure(1);
plot(t, y, 'LineWidth', 1);
xlim([-3*T/2 3*T/2]);
ylim([-2 2]);
xlabel('t(s)');
ylabel('v');
grid on;

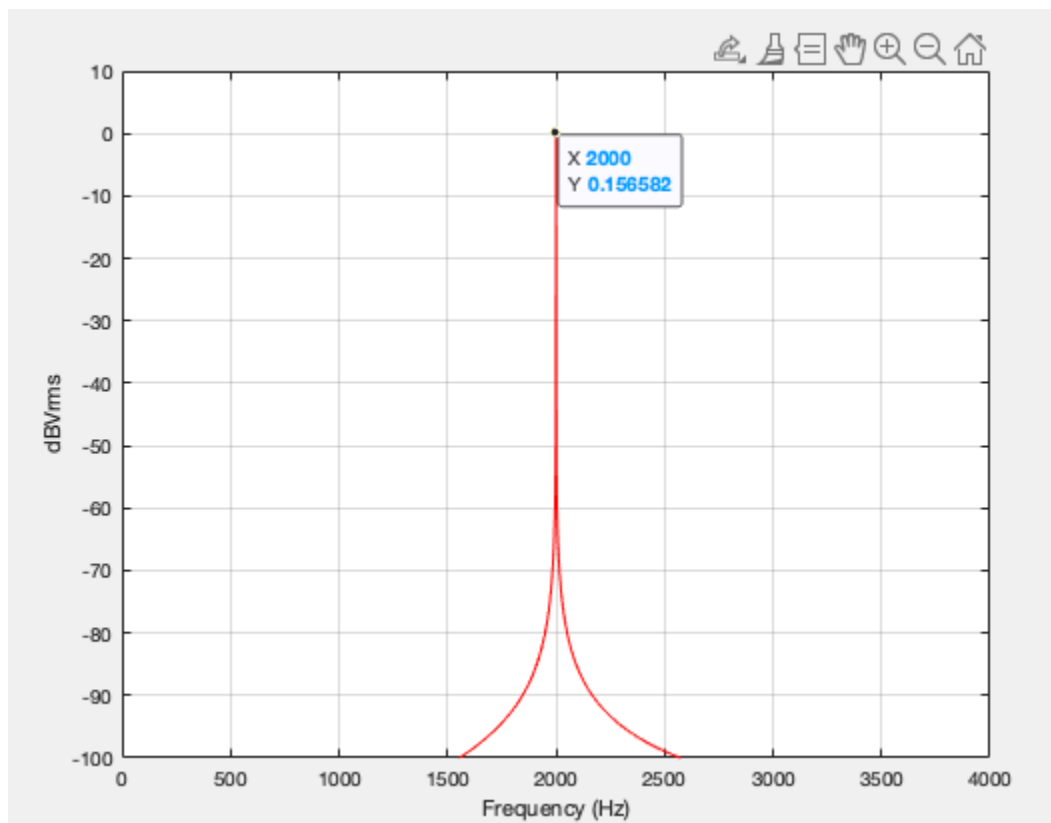
% Frequency-domain plot
figure(2);
Y = 2 * (abs(fft(y)) / n); % FFT normalization
Y = Y(1:floor(n/2) + 1);   % Extract half of the spectrum
d = 20 * log10(Y / sqrt(2)); % Convert to dB scale

f = linspace(0, fs/2, length(Y)); % Frequency vector
plot(f, d, 'r', 'LineWidth', 1); % Plot dB vs frequency
xlim([0 4000]);
ylim([-100 10]);
xlabel('Frequency (Hz)');
ylabel('dBVrms');
grid on;
```

Below is the plot for 3) in time domain:



Below is the plot for 3) in frequency domain:



Similar to the previous signal, the overall shape of the spectrum remains consistent, although there are minor differences due to MATLAB's assumption of an ideal environment. Upon examining the oscilloscope, it becomes evident that the frequency at which the spectral peak occurs for the two sine waves diverges from the values obtained through MATLAB and theoretical calculations.

This discrepancy may have resulted from instrumental errors associated with the signal generator and oscilloscope, as well as any unaccounted resistances in the setup or inherent internal resistances.

- 4) The peak values observed on the oscilloscope and in MATLAB are in close proximity to the theoretical values of 3 dB and 0 dB, respectively. The discrepancies in the results are primarily attributed to instrumental errors, while the limitations in MATLAB stem from mathematical inaccuracies.

Problem 2 : FFT of square wave

- 1) The sampling frequency is directly influenced by the time base (sec/div) setting. Lowering the sampling frequency results in a narrower bandwidth, while increasing the sampling frequency leads to a broader bandwidth.

- 2) Using the hardcopies taken, we can discuss the effect of changing the duty cycle on the FFT results as follows:

Changing the duty cycle of square waves significantly influences the contribution of the DC frequency component. For a square wave with a 50% duty cycle, the waveform exhibits odd symmetry, resulting in the absence of a DC component. Conversely, in the case of a 20% duty cycle, a DC component is present, which produces a noticeable spectral peak on the oscilloscope due to the waveform's lack of odd symmetry.

Examining the harmonic distribution further reveals insights into symmetry. The fundamental frequency remains constant across both duty cycles. In the 50% duty cycle wave, only odd harmonics are present. However, in the 20% duty cycle wave, both odd and even harmonics are evident, with certain harmonics, such as the fourth and fifth, disappearing, as shown in the oscilloscope images. This phenomenon occurs because the Fourier series encompasses both sine and cosine components, and their combination may yield contributions from both odd and even harmonics, or none for specific harmonics, as the waveform lacks both odd and even symmetry.

Overall, the changes in duty cycle lead to variations in the spectral content, demonstrating how symmetrical properties of the waveform directly affect the FFT results.

Problem 3 : FFT of square wave


The generated multiple-tone sinusoidal signal is created by combining two sine waves with frequencies of 1000 Hz and 10,000 Hz. The FFT of this composite signal corresponds to the sum of the FFTs of the individual waves, as confirmed by the oscilloscope. This linearity property of the FFT is evident in the presence of two spectral peaks at 1.000 kHz and 9.900 kHz, despite potential instrumental errors.

Furthermore, the Fourier Series can be utilized for periodic functions, while the Fourier Transform is applicable to non-periodic signals. The Fourier Series enables the reconstruction of periodic functions, demonstrating its effectiveness in frequency analysis. The FFT provides a more efficient means of analyzing signal spectra and can be performed using both MATLAB and oscilloscopes.

However, it is important to acknowledge that mathematical errors can occur when computing the FFT using MATLAB, and these errors may be beyond the user's control. To enhance accuracy, compensations should be considered, especially when analyzing sound samples in MATLAB. Additionally, the impact of instrumental errors can be minimized by utilizing more precise measurements.

conclusion?

In conclusion, the observed spectral peaks not only confirm the linearity of the FFT but also highlight the importance of accuracy in both measurement and computation for effective frequency analysis.



REFERENCES

I. Inst. Uwe Pagel: 'Electrical Engineering II Lab' (2019). AC Properties and Measurements'
<http://www.faculty.jacobs-university.de/upagel>

APPENDIX

PreLab Sampling

Problem 1: The Sampling Theorem



1. Passing analog signals through a low-pass filter before sampling is necessary to prevent **aliasing**, which can distort the digital representation of the signal.

Here's why:

Aliasing: When an analog signal is sampled, it must adhere to the Nyquist-Shannon sampling theorem, which states that the sampling rate must be at least twice the highest frequency present in the signal. If higher frequencies exist in the signal and are not filtered out, they can "fold back" into the lower frequencies, creating aliasing—distortion that makes it impossible to accurately reconstruct the original signal from the samples.

Pre-filtering: A low-pass filter, known as an anti-aliasing filter, removes frequencies higher than the Nyquist frequency (half the sampling rate), ensuring that the sampled signal does not contain frequency components that would cause aliasing.

In summary, low-pass filtering ensures that the sampled signal accurately represents the original analog signal by removing unwanted high-frequency components before digitization.

2. $\omega_s > 2\omega_m$, $\omega_s = 2 \times 2\pi \times 3000 = 37699.1\text{Hz}$

3. The Nyquist frequency is half of the sampling rate of a signal, and it represents the highest frequency that can be accurately captured without introducing aliasing.

4. Resulting frequencies for the following input sinusoids 500Hz, 2.5KHz, 5KHz and 5.5KHz if the signals are sampled by a sampling frequency of 5KHz

Input frequency(Hz)	Resulting frequencies(Hz)
500	500
2500	0
5000	0
5500	500

5. Three frequencies of signal that alias to a 7Hz signal when the signal is sampled by a constant 30 Hz sampling frequency.

$$f_a(N) = f_{in} - Nf_s$$

7+1(30)	37Hz
7+2(30)	67Hz
7+3(30)	97Hz

Problem 2: Impulse Train Sampling and Real Sampling

1.

The input signal $x(t)$ was given by a sine function, with an amplitude of 5V peak and a frequency of 50Hz. The sampling signal $p(t)$ is represented by a unity impulse train. Using an overall sampling rate of 100 k samples/s) for the following graphs graphed using matlab.

1) Simulations for the following cases:

a) Under Sampling (using 48 Hz)

Matlab code used

```
t = 0:1/100000:1; % these are the samples
f = 50; % frequency
A = 5; % amplitude
x = A*sin(2*pi*f*t); % the equation of the input wave

subplot(3,1,1);
plot(t,x);
grid on;
hold on;

axis([0 0.5 -7 7]);
xlabel('Time [s]');
ylabel('Amplitude [V]');
title('input signal x(t)')

% For the impulse train
fs = 48; % the under sampling stated in the manual
tt = 0:1/fs:1;
p = pulstran(t,tt,'rectpuls',1/100000);

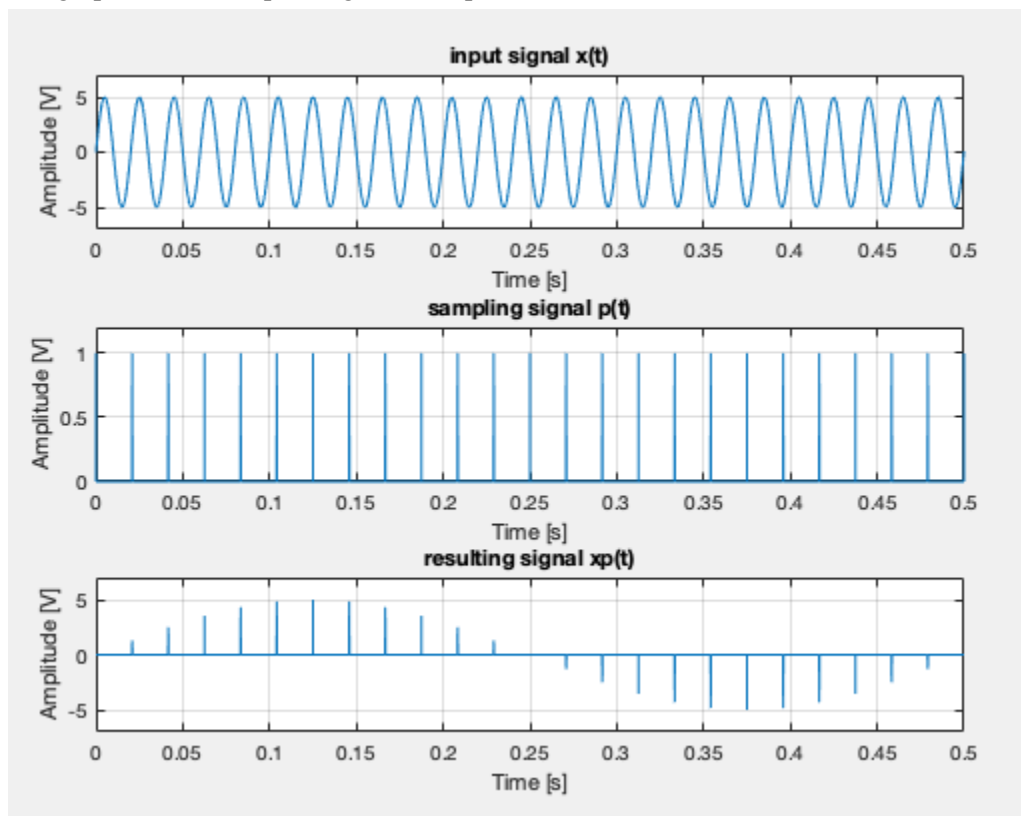
subplot(3,1,2);
plot(t,p);
grid on;
hold on;

axis([0 0.5 0 1.2]);
xlabel('Time [s]');
ylabel('Amplitude [V]');
title('sampling signal p(t)');

% Resulting signal after sampling
xp = x.*p;
subplot(3,1,3);
plot(t,xp);
grid on;
hold on;

axis([0 0.5 -7 7]);
xlabel('Time [s]');
ylabel('Amplitude [V]');
title('resulting signal xp(t)');
```

The graphs of the sampled signal the input.



b) Nyquist Sampling

Matlab code used

```
>> % Nyquist for 100 samples
t = 0:1/100000:1; % these are the samples
f = 50; % frequency
A = 5; % amplitude
x = A*sin(2*pi*f*t); % input signal

subplot(3,1,1);
plot(t,x);
grid on;
hold on;

ylabel('Amplitude [V]');
xlabel('Time [s]'); % Added xlabel for consistency
title('Input signal x(t)');

% For the impulse train
fs = 100; % the Nyquist sampling
tt = 0:1/fs:1;
p = pulstran(t,tt,'rectpuls',1/100000);

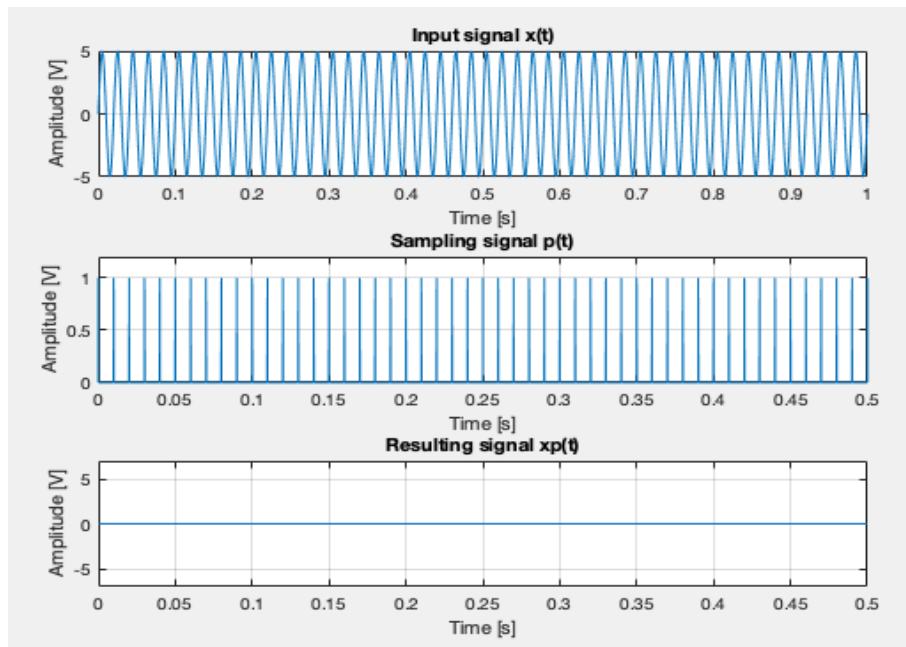
subplot(3,1,2);
plot(t,p);
grid on;
hold on;

axis([0 0.5 0 1.2]);
xlabel('Time [s]');
ylabel('Amplitude [V]');
title('Sampling signal p(t)');

% Resulting signal after the sampling
xp = x.*p;
subplot(3,1,3);
plot(t,xp);
grid on;
hold on;

axis([0 0.5 -7 7]); % Fixed typo in axis
xlabel('Time [s]');
ylabel('Amplitude [V]');
title('Resulting signal xp(t)');
```

The graphs of the sampled signal the input.



c) Over Sampling (using 1000 Hz)

Matlab code used

```
>> % Over Sampling (using 1000 Hz)
t = 0:1/100000:1; % these are the samples
f = 50; % Frequency
A = 5; % Amplitude
x = A*sin(2*pi*f*t); % input signal

subplot(3,1,1);
plot(t,x);
grid on;
hold on;

axis([0 0.5 -7 7]);
xlabel('Time [s]');
ylabel('Amplitude [V]');
title('Input signal x(t)')

% For the impulse train
fs = 1000; % for the over sampling
tt = 0:1/fs:1;
p = pulstran(t,tt,'rectpuls',1/100000); % sampling

subplot(3,1,2);
plot(t,p);
grid on;
hold on;

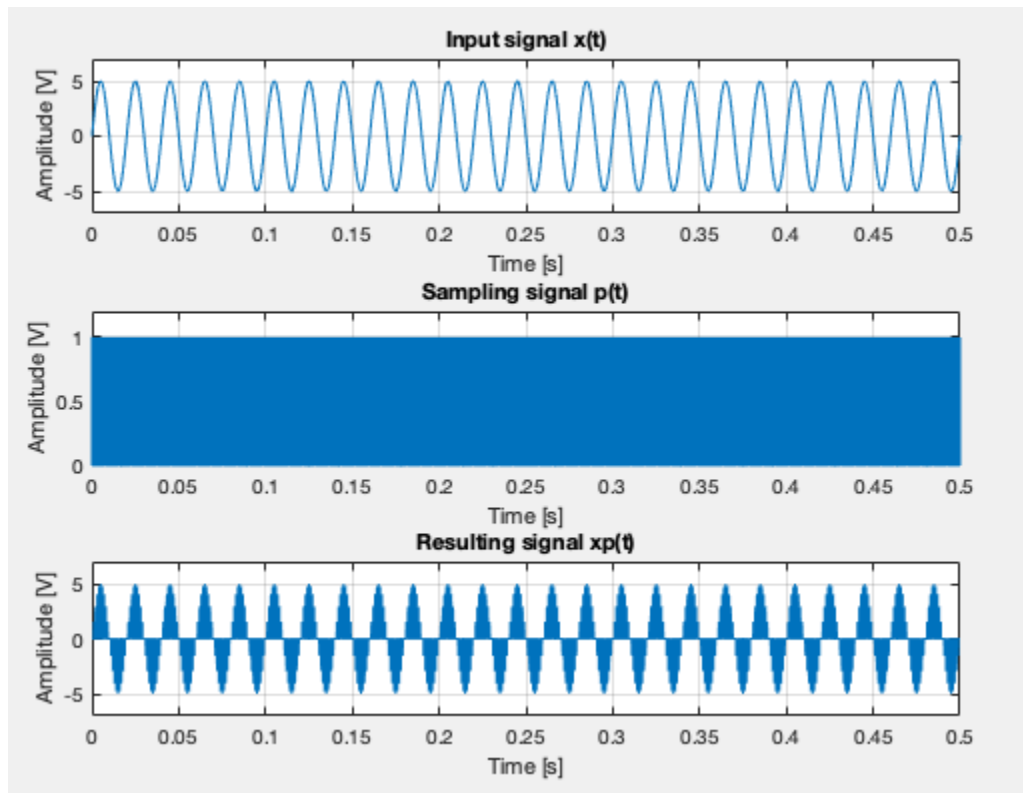
axis([0 0.5 0 1.2]);
xlabel('Time [s]');
ylabel('Amplitude [V]');
title('Sampling signal p(t)');

% Resulting signal after sampling
xp = x.*p; % sampled graph
subplot(3,1,3);
plot(t,xp);
grid on;
hold on;

axis([0 0.5 -7 7]);
xlabel('Time [s]');
ylabel('Amplitude [V]');
title('Resulting signal xp(t)');
```

← it's easier to use the square pulse function !! square()

The graphs of the sampled signal the input.



The command subplot was used to visualize the continuous signal $x(t)$, the sampling signal $p(t)$ and the result for each of these cases.

2) The signal $x(t)$ was sampled by a rectangular pulse train. Modifying the sampling function $p(t)$, so that the width of the sampling pulse is 50% of the sampling period, do the same simulations we did in the previous sections

a) Under Sampling

Matlab code used

```
t = 0:1/100000:1; % these are the samples
f = 50; % frequency
A = 5; % Amplitude
x = A*sin(2*pi*f*t); % the input signal

% First subplot: Input signal x(t)
subplot(3,1,1);
plot(t,x);
grid on;
hold on;
axis([0 0.25 -7 7]);
xlabel('Time [s]');
ylabel('Amplitude [V]');
title('Input Signal x(t)');

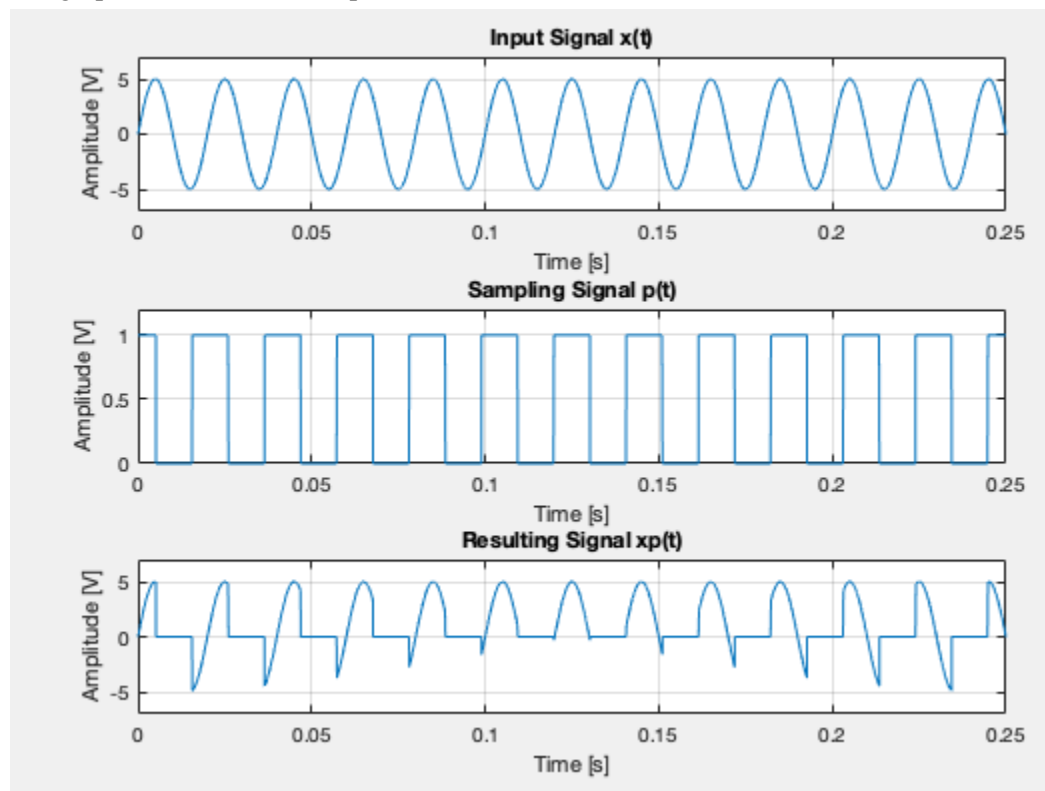
% For the impulse train
fs = 48; % the under sampling
tt = 0:1/fs:1; % Define tt for the sampling signal
p = pulstran(t,tt,'rectpuls',0.5*1/fs); % the sampling signal

% Second subplot: Sampling signal p(t)
subplot(3,1,2);
plot(t,p);
grid on;
hold on;
axis([0 0.25 0 1.2]);
xlabel('Time [s]');
ylabel('Amplitude [V]');
title('Sampling Signal p(t)');

% Resulting signal
xp = x .* p; % the sampled signal

% Third subplot: Resulting signal xp(t)
subplot(3,1,3);
plot(t,xp);
grid on;
hold on;
axis([0 0.25 -7 7]);
xlabel('Time [s]');
ylabel('Amplitude [V]');
title('Resulting Signal xp(t)');
```

The graphs for the under sampled



b) Nyquist Sampling

Matlab code used

```
t = 0:1/100000:1; % These are the samples
f = 50; % Frequency
A = 5; % Amplitude
x = A*sin(2*pi*f*t); % Input signal

% First subplot: Input signal x(t)
subplot(3,1,1);
plot(t,x);
grid on;
hold on;
axis([0 0.25 -7 7]);
xlabel('Time [s]');
ylabel('Amplitude [V]');
title('Input Signal x(t)');

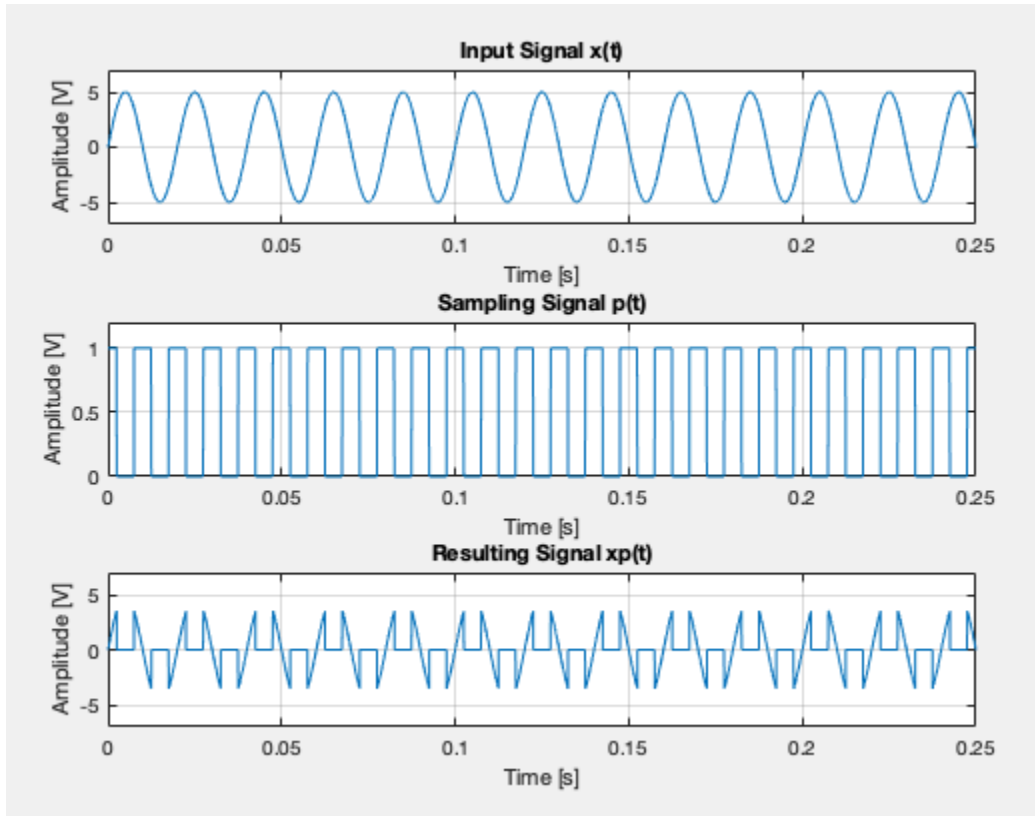
% For the impulse train
fs = 100; % The Nyquist sampling
tt = 0:1/fs:1; % Define tt for the sampling signal
p = pulstran(t,tt,'rectpuls',0.5*1/fs); % The sampling signal

% Second subplot: Sampling signal p(t)
subplot(3,1,2);
plot(t,p);
grid on;
hold on;
axis([0 0.25 0 1.2]);
xlabel('Time [s]');
ylabel('Amplitude [V]');
title('Sampling Signal p(t)');

% Resulting signal
xp = x .* p; % The sampled signal

% Third subplot: Resulting signal xp(t)
subplot(3,1,3);
plot(t,xp);
grid on;
hold on;
axis([0 0.25 -7 7]);
xlabel('Time [s]');
ylabel('Amplitude [V]');
title('Resulting Signal xp(t)').
```

The graphs of the Nyquist sampling



c) Over Sampling

Matlab code used

```
t = 0:1/100000:1; % These are the samples
f = 50; % Frequency
A = 5; % Amplitude
x = A*sin(2*pi*f*t); % Input signal

% First subplot: Input signal x(t)
subplot(3,1,1);
plot(t,x);
grid on;
hold on;
axis([0 0.25 -7 7]);
xlabel('Time [s]');
ylabel('Amplitude [V]');
title('Input Signal x(t)');

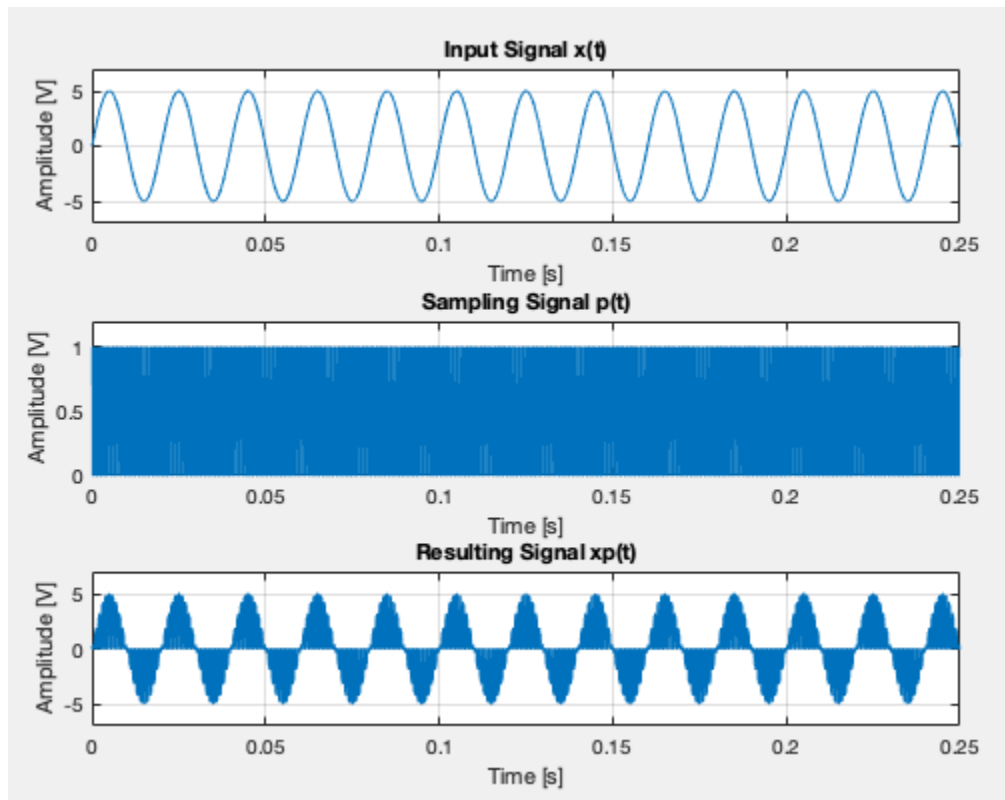
% For the impulse train
fs = 1000; % The over sampling
tt = 0:1/fs:1; % Define tt for the sampling signal
p = pulstran(t,tt,'rectpuls',0.5*1/fs); % Sampling signal

% Second subplot: Sampling signal p(t)
subplot(3,1,2);
plot(t,p);
grid on;
hold on;
axis([0 0.25 0 1.2]);
xlabel('Time [s]');
ylabel('Amplitude [V]');
title('Sampling Signal p(t)');

% Resulting signal
xp = x .* p; % Sampled signal

% Third subplot: Resulting signal xp(t)
subplot(3,1,3);
plot(t,xp);
grid on;
hold on;
axis([0 0.25 -7 7]);
xlabel('Time [s]');
ylabel('Amplitude [V]');
title('Resulting Signal xp(t)').
```

The oversampled graph plotted using the code

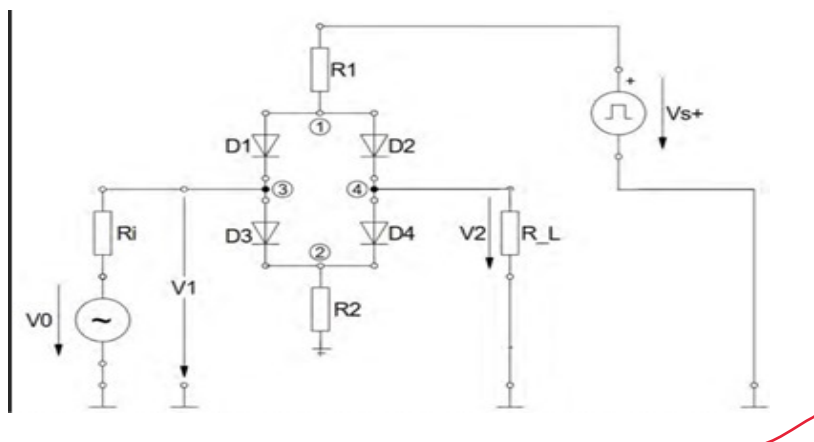


why did you show the MatLab code 6 times, one time is enough!! You only change values, just mention in a comment and that's it...

Problem 3: Sampling using a Sampling bridge

To Modify the circuit in such a way that a single sampling source can be used to sample the input signal, we short circuited V_s

1) Below is the sketch of the modified circuit.



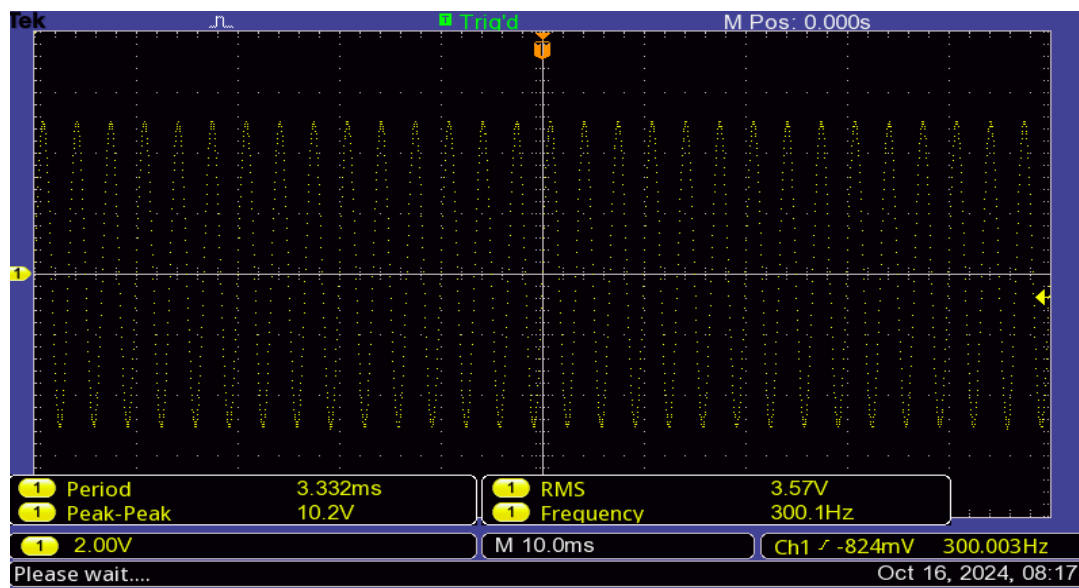
- 2) The input consists of V_0 and has an input resistance of R_i . The sampling circuit includes four diodes (D_1, D_2, D_3, D_4) and three resistors (R_1, R_2, R_L). The circuit receives a square wave as the positive sampling signal, with node 1 initiating the positive sampling pulse. When V_s is sufficiently high, the diodes become forward-biased, creating a conductive path between nodes 3 and 4. This allows the signal on the input side to pass through to the output load resistor. In this scenario, V_2 should match V_1 , where V_1 represents the sum of V_0 and the voltage drop across the input resistor R_i . For proper operation, the values of R_1 and R_2 must be within a suitable range—not too high or too low. .. and what is the difference with only one sampling source??

Execution Sampling

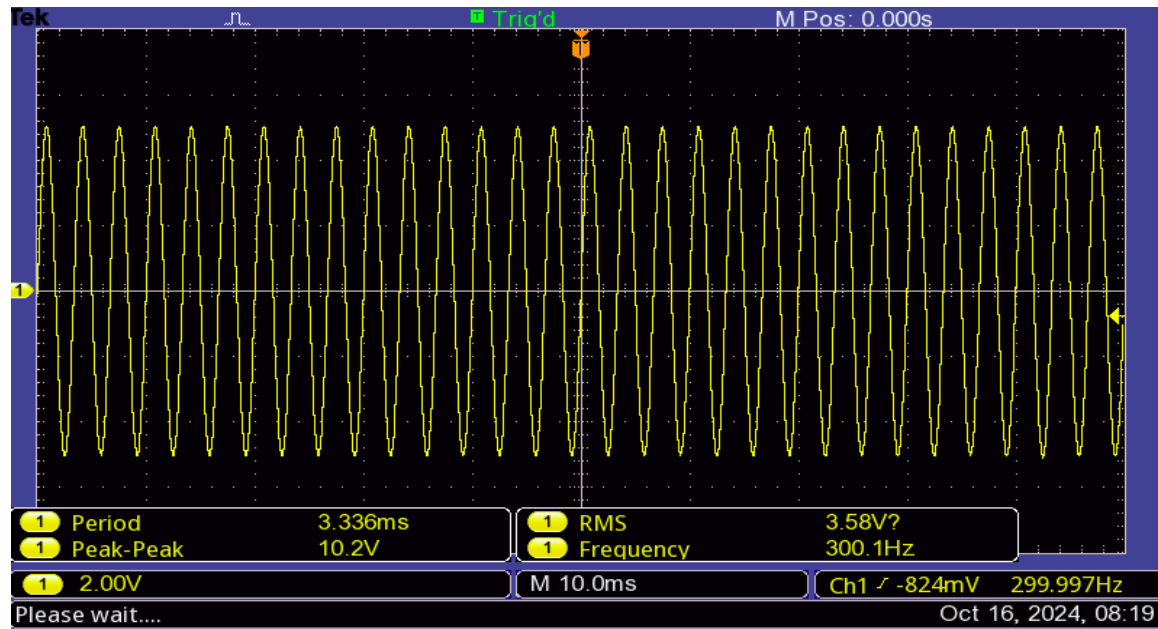
Problem 1: Digital Sampling Oscilloscope

Connecting the function generator to the oscilloscope, using $f = 300 \text{ Hz}$, $A = 5 \text{ Vpp}$, no offset and setting the sampling rate at the oscilloscope to 25 kS/s .

Below is a hardcopy showing dot form.

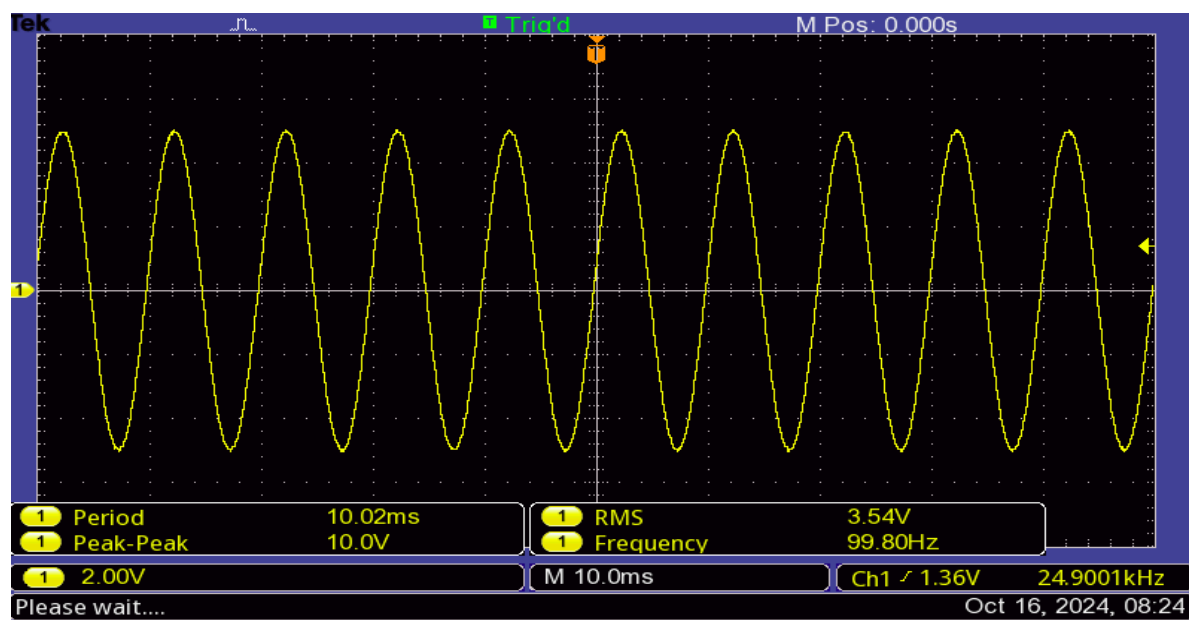


Below is a hardcopy showing vector form.



Keeping the sampling rate at the oscilloscope at 25 kS/s

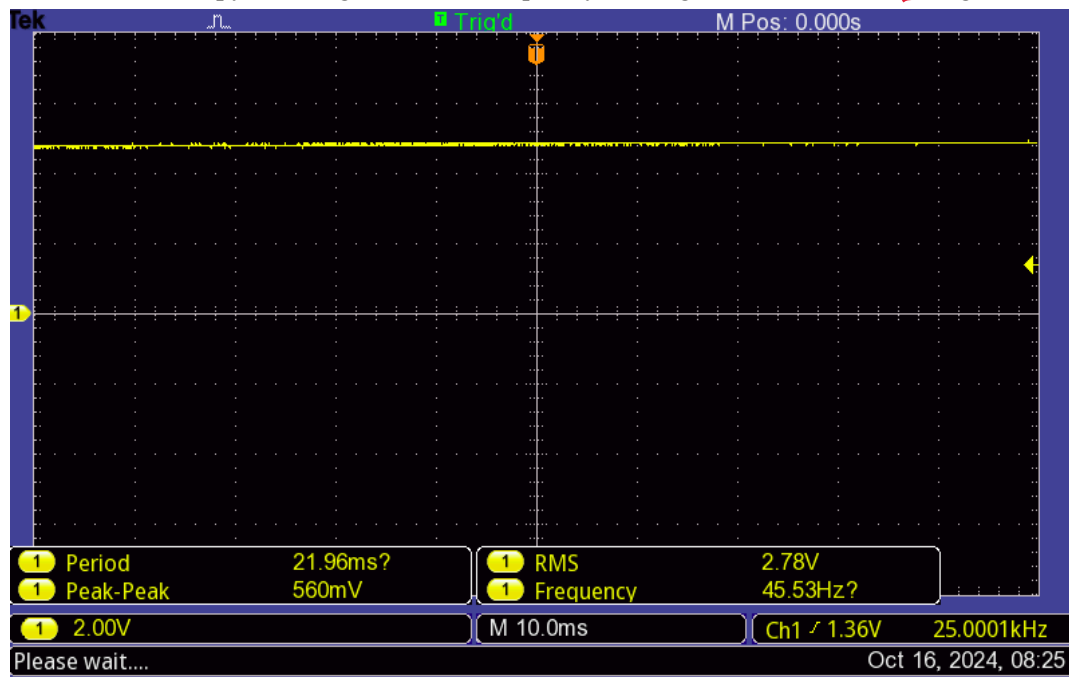
Below is a hardcopy showing 24900 Hz frequency at the generator and 99.8Hz aliasing frequency:



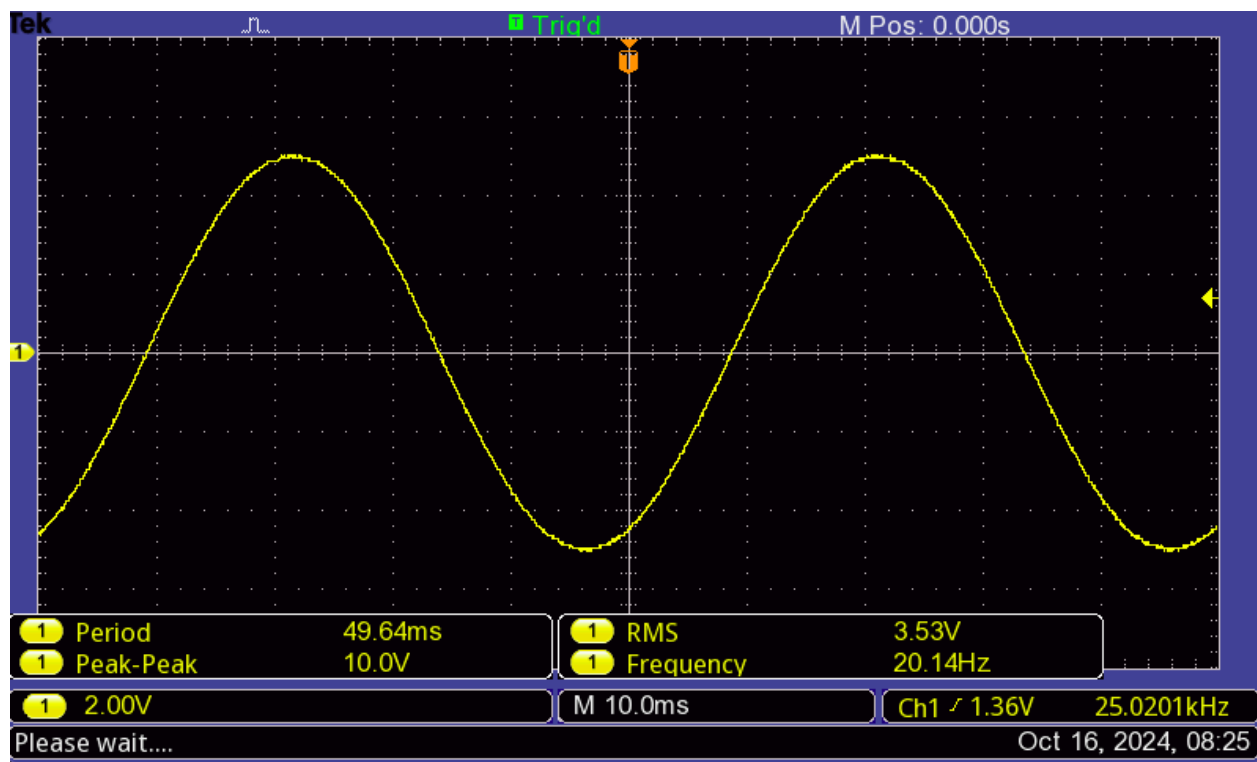


of course aliasing... 0Hz!!!

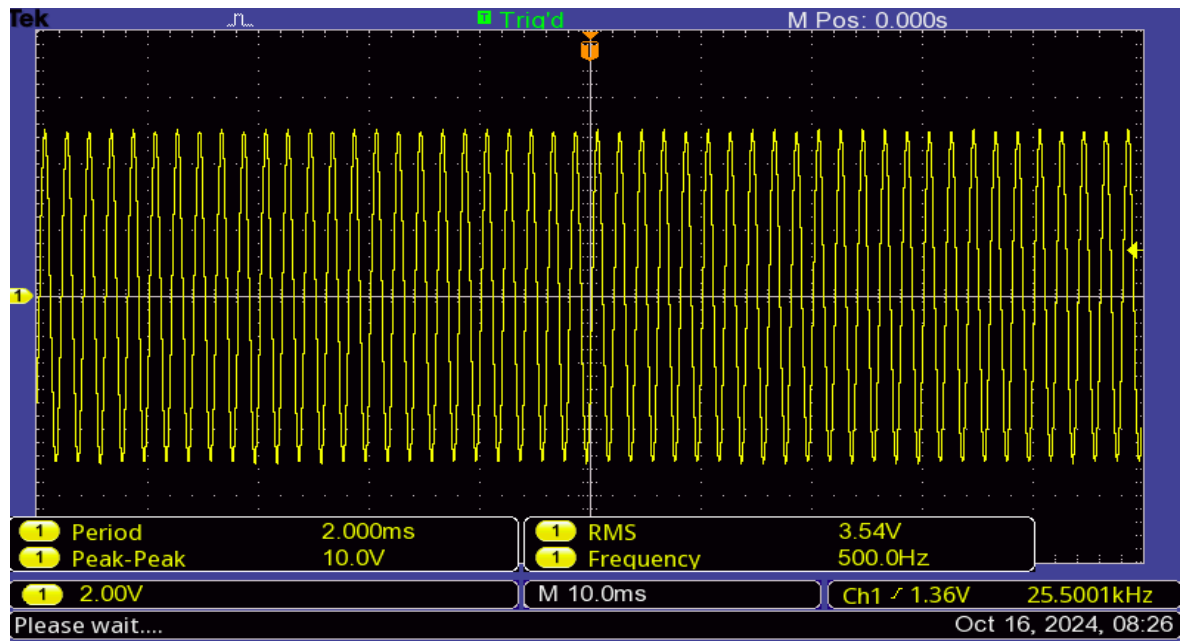
Below is a hardcopy showing 25000 Hz frequency at the generator and no aliasing:



Below is a hardcopy showing 25020 Hz frequency at the generator and aliasing frequency 20.14Hz:

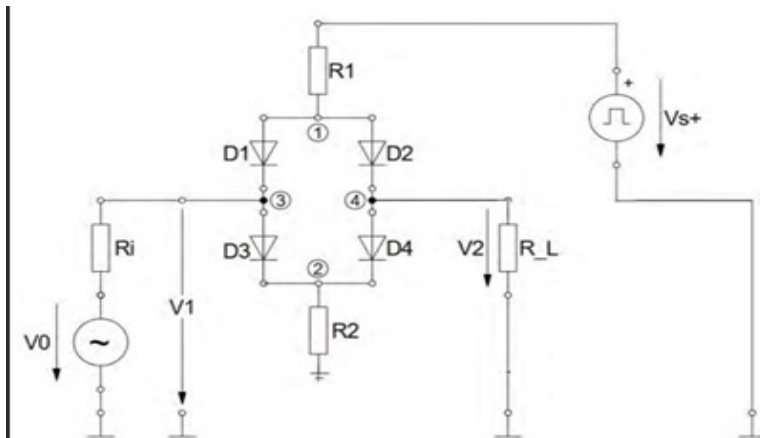


Below is a hardcopy showing 25500 Hz frequency at the generator and 500Hz aliasing frequency:

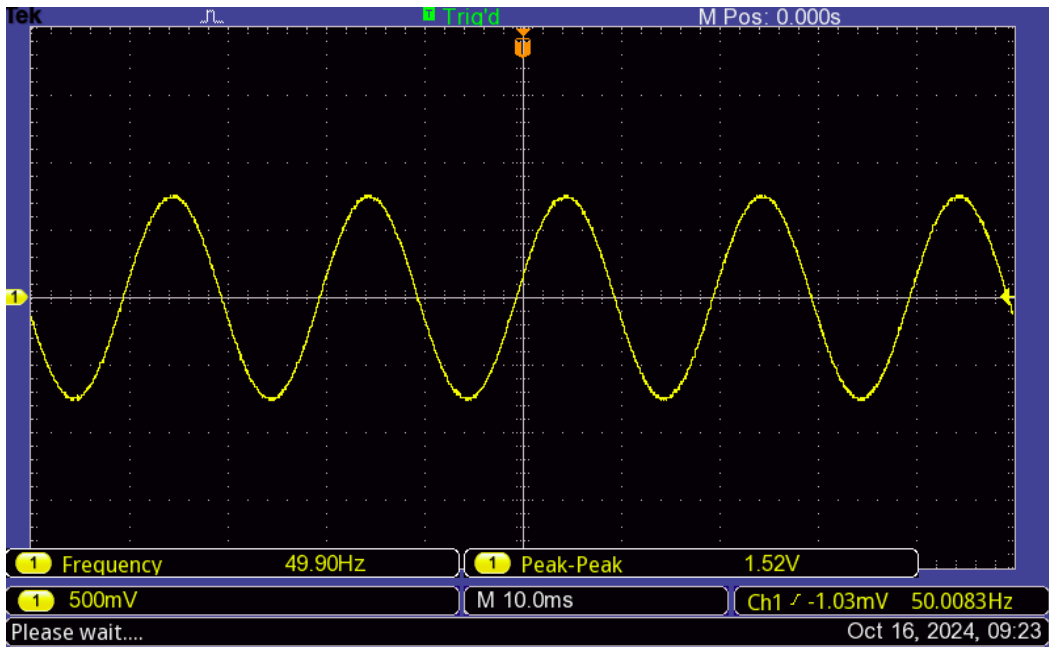


Problem 2: Sampling using a sampling bridge

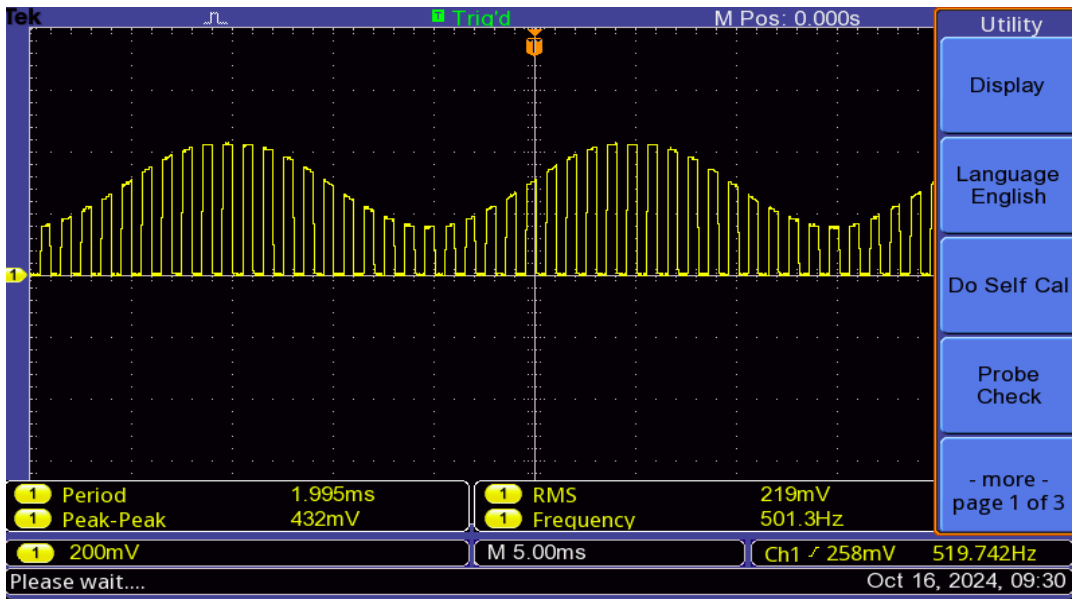
Implementing a modified sampling circuit below using a single sampling source. Using the square signal of the auxiliary function generator as the sampling signal and 1N4148 diodes and 10K Ω resistors for the bridge and a 100 K Ω resistor as load. . Setting the amplitude to $V_{pp} = 1.5$ V, and the offset to $V = 2.5$ V.



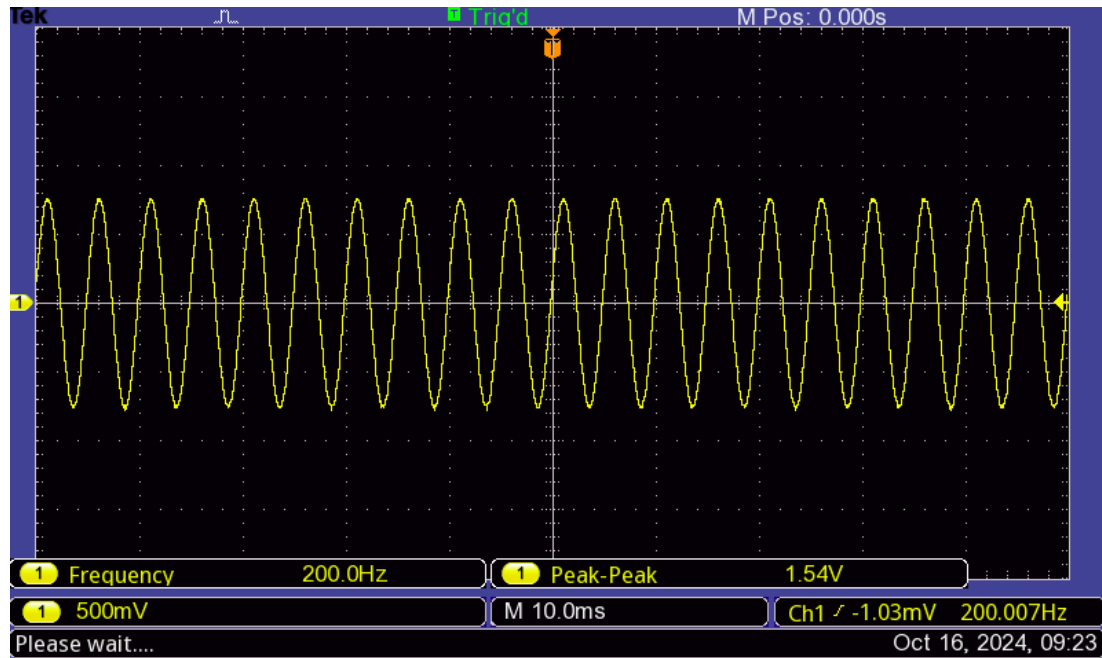
Below is the hard copy showing the input signal at 50Hz



Below is the hard copy showing a sampled signal at 50Hz



Below is the hard copy showing the input signal at 200Hz



Below is the hard copy showing a sampled signal at 200 Hz

