Title:

Fourier Series and Fourier Transform Using MATLAB

Objectives:

- To understand the mathematical foundation of Fourier Series and Fourier Transform.
- To learn how to represent periodic signals using the Fourier Series.
- To study how the Fourier Transform generalizes this concept for non-periodic signals.
- To visualize both representations using MATLAB for various signals.
- To observe the behavior of signal reconstruction and frequency domain analysis.

Theory:

The Fourier Series and Fourier Transform are essential mathematical tools for analyzing signals, especially in the fields of electrical engineering, signal processing, and communication systems. Both are based on the principle that any signal, whether periodic or non-periodic, can be expressed as a combination of simpler sine and cosine functions.

The **Fourier Series** is used specifically for periodic signals. Consider a function f(t) that is periodic with period T. According to Fourier's theorem, this function can be written as an infinite sum of sines and cosines, each with frequencies that are integer multiples of the fundamental frequency.

Mathematically, the Fourier Series of a function f(t) is expressed as:

$$f(t) = a0 + \Sigma[n=1 \text{ to } \infty]\ [\ an * \cos(n * \omega0 * t) + bn * \sin(n * \omega0 * t)\ ]$$

Where:

- a0 is the average or DC component of the signal,
- an and bn are the Fourier coefficients,
- $\omega0 = 2\pi / T$ is the fundamental angular frequency.

The coefficients are calculated using integration over one period:

$$a0 = (1 / T) * \int[0 \text{ to } T]\ f(t)\ dt$$

$$an = (2 / T) * \int[0 \text{ to } T]\ f(t) * \cos(n * \omega0 * t)\ dt$$

$$bn = (2 / T) * \int[0 \text{ to } T]\ f(t) * \sin(n * \omega0 * t)\ dt$$

In practical implementations using MATLAB, the series is truncated to a finite number of terms (say N harmonics). As N increases, the approximation becomes more accurate. For instance, a square wave approximated using 5, 10, or 20 harmonics begins to resemble the original waveform closely. However, near discontinuities, the Gibbs phenomenon appears, which refers to persistent oscillations or overshoots that do not vanish even as more harmonics are added.

While the Fourier Series is powerful for periodic signals, many real-world signals are **non-periodic**. In such cases, we move from the Fourier Series to the **Fourier Transform**, which is a more general tool used to analyze any signal in terms of its frequency components.

The **Fourier Transform** represents a non-periodic time-domain signal $f(t)$ in the frequency domain. Instead of expressing the signal as a sum of discrete frequency components, the Fourier Transform provides a continuous spectrum that shows how much of each frequency is present in the signal.

The continuous-time Fourier Transform is defined as:

$$F(\omega) = \int_{-\infty}^{\infty} f(t) * e^{-j\omega t}\, dt$$

And the inverse transform is given by:

$$f(t) = (1 / 2\pi) * \int_{-\infty}^{\infty} F(\omega) * e^{j\omega t}\, d\omega$$

Here, $F(\omega)$ is the frequency domain representation of the time-domain signal $f(t)$. The Fourier Transform translates time-domain signals into the frequency domain, allowing engineers and scientists to see the energy distribution across frequencies. This is extremely useful in fields such as audio engineering, image processing, biomedical signals (e.g., EEG, ECG), and communications.

In MATLAB, the **Fast Fourier Transform (FFT)** is often used to compute the discrete version of the Fourier Transform for digital signals. It allows efficient and fast conversion of time-domain sampled data into the frequency domain. The FFT provides both the magnitude and phase of frequency components, which can then be visualized using magnitude and phase spectra.

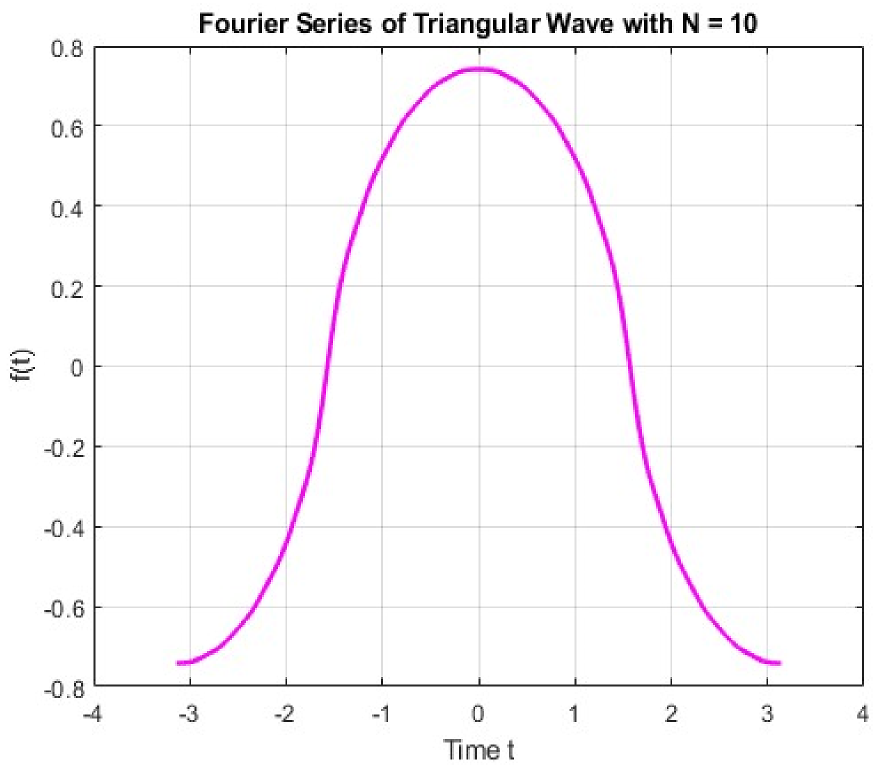
For example, a sampled sine wave or a combination of sine waves can be analyzed using FFT to reveal peaks at their respective frequencies. Similarly, noise in a signal can be identified and filtered by observing and manipulating its frequency content.

A key distinction between Fourier Series and Fourier Transform is that the former is defined only for periodic functions and gives a discrete spectrum, while the latter is applicable to non-periodic functions and results in a continuous frequency spectrum. Nonetheless, both share the fundamental idea of representing signals as weighted sums (or integrals) of sine and cosine functions, which are the building blocks of all signals.

Another point worth noting is the symmetry and energy-preserving nature of Fourier methods. For real-valued signals, the Fourier spectrum exhibits conjugate symmetry. Parseval's theorem connects time-domain and frequency-domain energy, ensuring that the total energy remains consistent across transformations.

In summary, both Fourier Series and Fourier Transform are vital tools for understanding signal behavior. The Fourier Series is ideal for analyzing repetitive signals like waveforms in electric circuits, while the Fourier Transform is suited for real-time, non-repetitive signals such as speech, radar, and biomedical signals. MATLAB helps simulate and visualize these representations, making abstract mathematical concepts more concrete and applicable.

```
 %Fourier Series of a Triangular Wave
clc; clear; close all

t = linspace(-pi, pi, 1000);
f = zeros(size(t));
N = 10;
for n = 1:2:(2*N-1)
   f = f + ((8 / (pi^2)) * ((-1)^((n-1)/2) / n^2)) * cos(n * t);
end
plot(t, f, 'm', 'LineWidth', 2);
grid on;
xlabel('Time t'); ylabel('f(t)');
title(['Fourier Series of Triangular Wave with N = ', num2str(N)]);
```



2. Fourier Series of a Sawtooth Wave

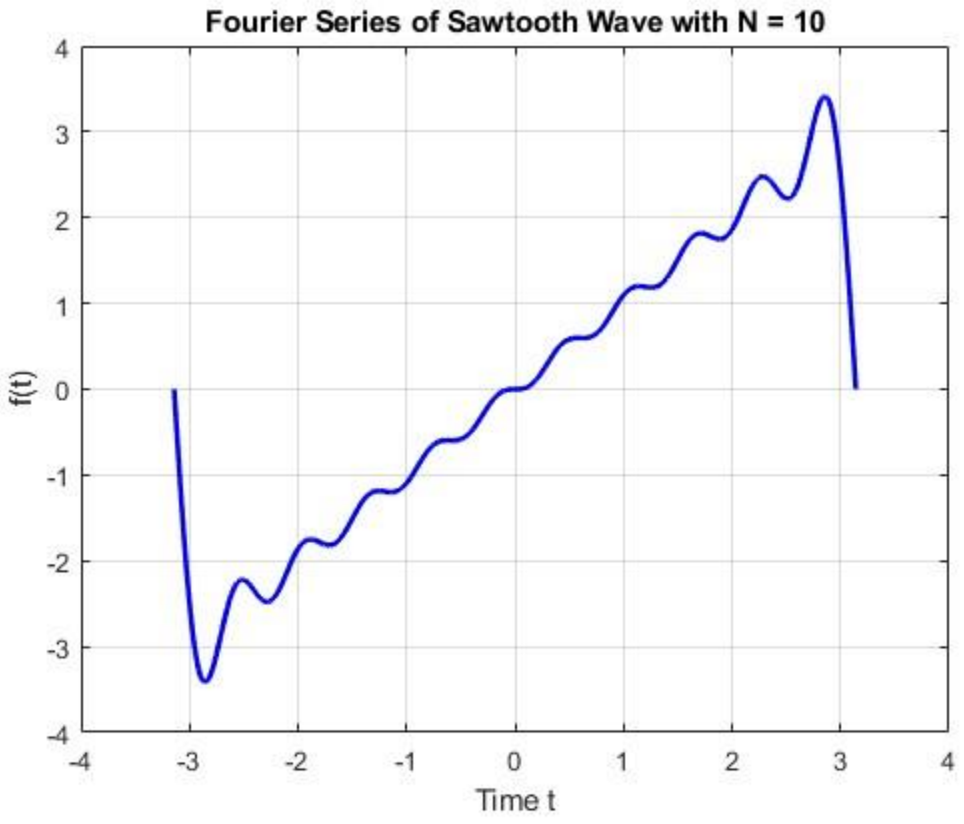
Mathematical Form:

f(t) = 2 * [(-1)^(n+1)/n] * sin(n*t), summed over n from 1 to ∞.

This is an odd function, and hence only sine terms appear.

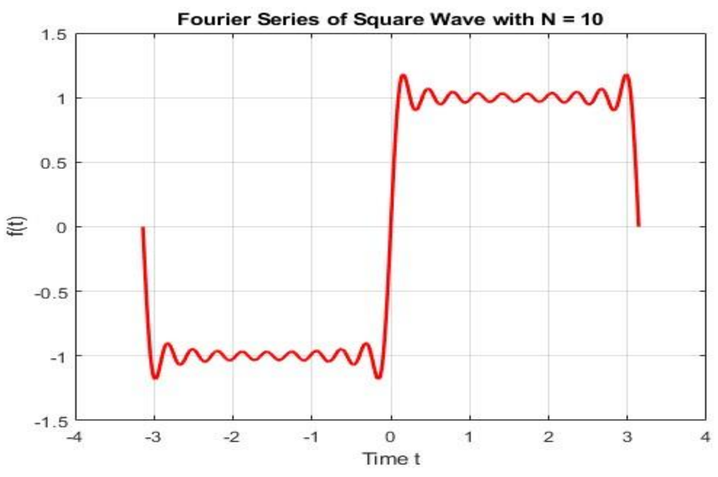
MATLAB Code:

```
clc; clear; close all

t = linspace(-pi, pi, 1000);
f = zeros(size(t));
N = 10;

for n = 1:N
   f = f + ((-1)^(n+1) * 2 / n) * sin(n * t);
end

plot(t, f, 'b', 'LineWidth', 2);
grid on;
xlabel('Time t'); ylabel('f(t)');
title(['Fourier Series of Sawtooth Wave with N = ', num2str(N)]);
```



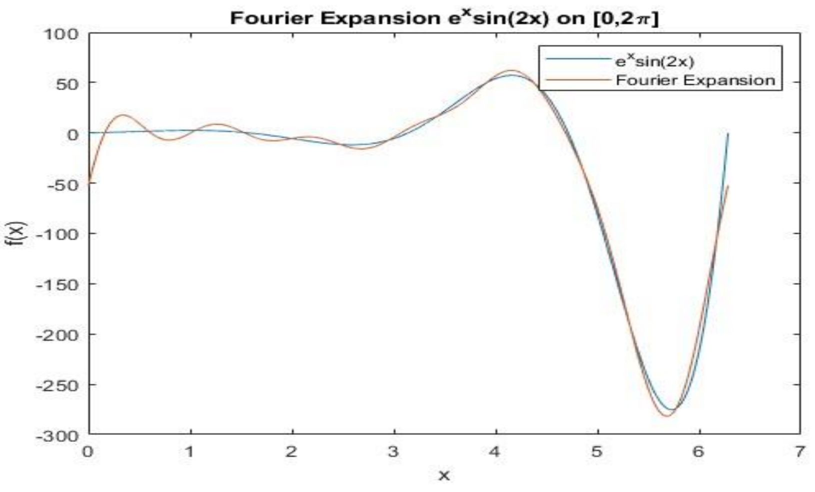
3. Fourier Series of a Square Wave::

```
clc; clear; close all;
t = linspace(-pi, pi, 1000);
f = zeros(size(t));
N = 10;
for n = 1:2:(2*N-1)
   f = f + (4/(n*pi)) * sin(n*t);
end
plot(t, f, 'r', 'LineWidth', 2);
grid on;
xlabel('Time t'); ylabel('f(t)');
title(['Fourier Series of Square Wave with N = ', num2str(N)]);
```



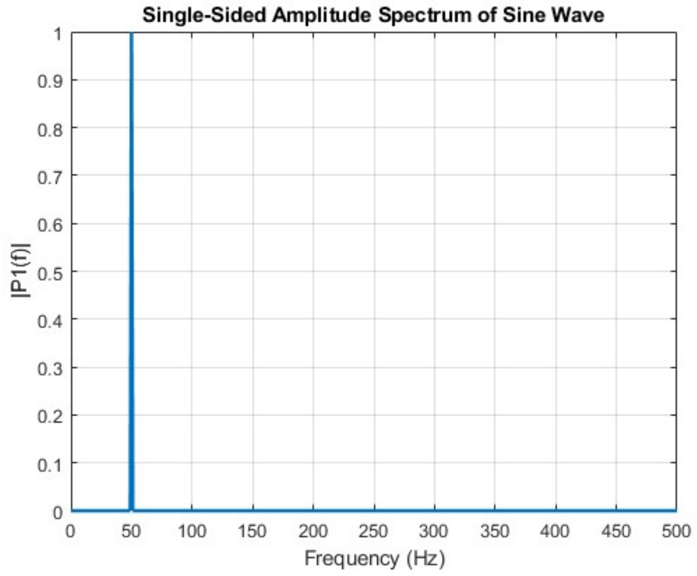
4. MATLAB script to plot the above expansion:

```
x = 0:1/1000:2*pi;         %discretize interval
f = exp(x).*sin(2*x);      %declare function
f_exp=0;                   %initialize expansion
N = 6;                     %number of harmonics
for n=1:N                  %iterate partial sum
An = (exp(2*pi)-1)/(2*pi)*(4*n^2-20)/(n^4-6*n^2+25);
Bn = (exp(2*pi)-1)/(2*pi)*(8*n)/(n^4-6*n^2+25);
f_exp = f_exp + An*cos(n*x) + Bn*sin(n*x);
end
A0 = (1-exp(2*pi))/(5*pi);
f_exp = f_exp+A0;          %add A0 term
plot(x,f,x,f_exp)          %plot function and expansion
title('Fourier Expansion e^xsin(2x) on [0,2\pi]');
xlabel('x'); ylabel('f(x)');
legend('e^xsin(2x)','Fourier Expansion')
```



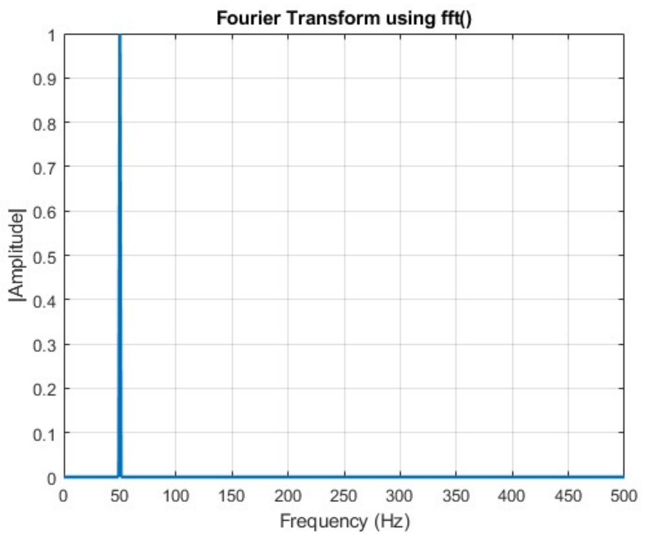
LAB 5:

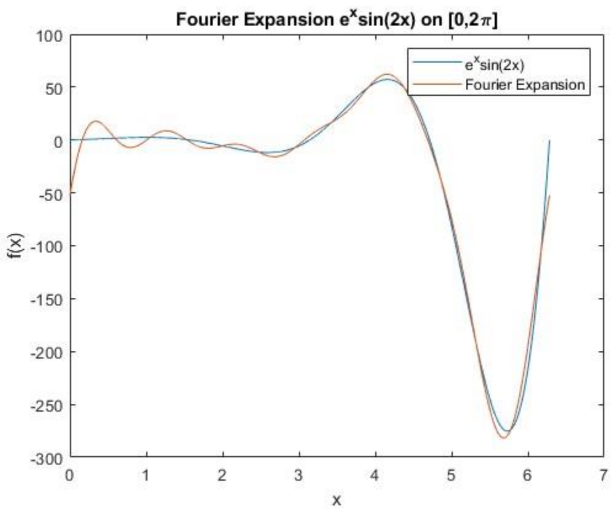
```
%1. Fs = 1000;          % Sampling frequency (Hz)
T = 1/Fs;           % Sampling period
L = 1000;           % Number of samples
t = (0:L-1)*T;       % Time vector
f_sin = 50;         % Frequency of sine wave
(Hz)
x = sin(2*pi*f_sin*t); % Sine wave signal
% FFT
Y = fft(x);
P2 = abs(Y/L);       % Two-sided spectrum
P1 = P2(1:L/2+1);     % One-sided
spectrum
P1(2:end-1) = 2*P1(2:end-1);
f = Fs*(0:(L/2))/L;    % Frequency vector
% Plot
plot(f, P1, 'LineWidth', 2);
title('Single-Sided Amplitude Spectrum of Sine Wave');
xlabel('Frequency (Hz)');
ylabel('|P1(f)|');
grid on;
```



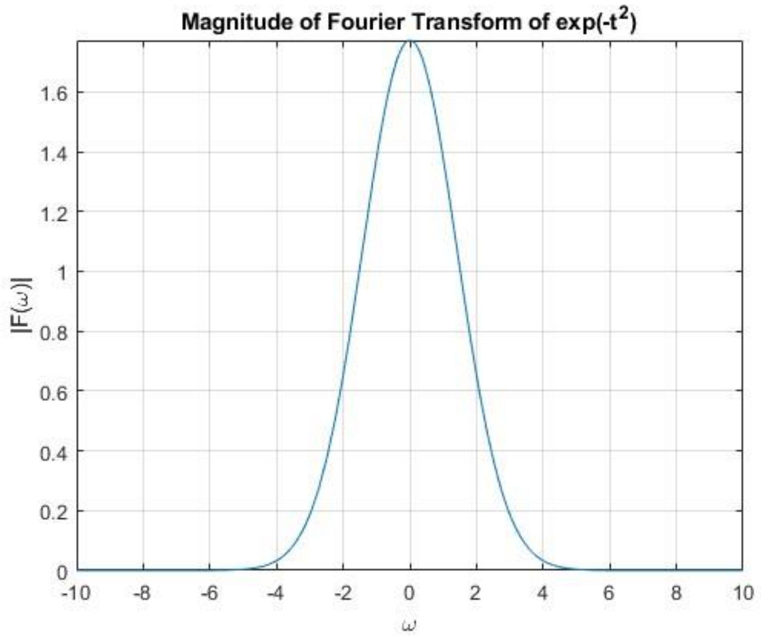
```
%2.% Parameters
Fs = 1000;           % Sampling frequency (Hz)
T = 1/Fs;            % Sampling period
L = 1000;            % Number of samples
t = (0:L-1)*T;        % Time vector
% Composite signal: sum of 3 sine waves
f1 = 50;             % Frequency 1 (Hz)
f2 = 120;            % Frequency 2 (Hz)
f3 = 200;            % Frequency 3 (Hz)
x = 0.7*sin(2*pi*f1*t) + sin(2*pi*f2*t) + 0.5*sin(2*pi*f3*t);
% FFT
Y = fft(x);
P2 = abs(Y/L);         % Two-sided spectrum
P1 = P2(1:L/2+1);       % One-sided spectrum
P1(2:end-1) = 2*P1(2:end-1);
f = Fs*(0:(L/2))/L;     % Frequency vector
% Plot
plot(f, P1, 'LineWidth', 2);
```

```
title('FFT of Composite Signal');
xlabel('Frequency (Hz)');
ylabel('|P1(f)|');
grid on;

%% Parameters
Fs = 1000;          % Sampling frequency (Hz)
T = 1/Fs;           % Sampling period
L = 1000;           % Length of signal
t = (0:L-1)*T;      % Time vector

% Signal: sine wave at 50 Hz
f_sin = 50;
x = sin(2*pi*f_sin*t);

% Signal: sine wave at 50 Hz

f_sin = 50;

x = sin(2*pi*f_sin*t);



% 3.Fourier Transform using fft()
% Parameters
Fs = 1000;          % Sampling frequency (Hz)
T = 1/Fs;           % Sampling period
L = 1000;           % Length of signal
t = (0:L-1)*T;      % Time vector

% Signal: sine wave at 50 Hz
f_sin = 50;
x = sin(2*pi*f_sin*t);



% Fourier Transform using fft()
Y = fft(x);         % Apply FFT
P2 = abs(Y/L);      % Two-sided spectrum
P1 = P2(1:L/2+1);   % One-sided spectrum
P1(2:end-1) = 2*P1(2:end-1);  % Adjust amplitudes

% Frequency vector
f = Fs*(0:(L/2))/L;
```





```
% Plot the amplitude spectrum
plot(f, P1, 'LineWidth', 2);
title('Fourier Transform using fft()');
xlabel('Frequency (Hz)');
ylabel('|Amplitude|');
grid on;


%4
syms t w
f = exp(-t^2);  % Gaussian function

% Compute Fourier Transform
F = fourier(f, t, w);

disp('Fourier Transform F(w):')
pretty(F)
```



Discussion and Conclusion:

This experiment provided hands-on insight into how periodic and non-periodic signals can be decomposed and analyzed using Fourier techniques. The use of MATLAB made it easier to implement both the Fourier Series and the Fourier Transform, visualize the signal approximations, and explore the impact of varying the number of harmonics or sampling rates.

The Fourier Series allowed for accurate reconstruction of periodic signals like square and triangular waves. As more harmonics were added, the signal approximation improved, though the Gibbs phenomenon remained near points of discontinuity. This highlighted both the power and limitations of the Fourier Series in practical applications.

The Fourier Transform extended the analysis to non-periodic signals, giving a complete view of their frequency content. Using MATLAB's FFT function, we could clearly observe how energy is distributed among different frequencies. This is particularly useful in filtering, spectral analysis, and system design.

Overall, this lab reinforced the foundational concepts of signal decomposition, both in theory and through practical simulation. Understanding these tools is essential for any student or engineer working in the domains of signal processing, electrical systems, communications, or control engineering.