GIBBS PHENOMENON – Square Wave

clear all;

close all;

clc;

% Time period and other variables

time\_period = 2; % square wave

step\_size\_t = 0.0001;

t = 0:step\_size\_t:time\_period;

tt = length(t);

omega\_o = 2 \* pi / time\_period; % Fundamental angular frequency

% Number of Fourier coefficients

no\_of\_fourier\_coeff = 50;

kk = -no\_of\_fourier\_coeff:1:no\_of\_fourier\_coeff;

% Initialize v\_t for square wave signal

v\_t = zeros(1, tt);

for ii = 1:tt

if (t(ii) <= 1) % High part of the square wave for 0 <= t <= 1

v\_t(ii) = 1;

else % Low part of the square wave for 1 < t <= 2

v\_t(ii) = -1;

end

end

% Plot the square wave

subplot(4,2,[1,2]), plot(t, v\_t);

xlabel("time(t)");

ylabel("v(t)");

title('Square Wave');

% Fourier coefficient array initialization

a\_k = zeros(1, 2 \* no\_of\_fourier\_coeff + 1); % For storing Fourier coefficients

% Calculate Fourier coefficients

for ii = 1:1:(2 \* no\_of\_fourier\_coeff + 1)

k = kk(ii); % Fourier series index

temp1 = v\_t .\* exp(-1i \* k \* omega\_o \* t); % v(t) \* exp(-j \* k \* omega\_o \* t)

% Integrate over the period using trapezoidal rule

int\_ans = my\_int\_func(temp1, step\_size\_t);

% Calculate Fourier coefficient

a\_k(ii) = (1 / time\_period) \* int\_ans;

end

% Plot Fourier coefficients

subplot(4,2,[3,4]), stem(kk, abs(a\_k));

xlabel('k');

ylabel('|a\_k|');

title('Magnitude of Fourier Coefficients');

% Initialize harmonics

harmonics = zeros(1, tt);

% Reconstruct signal from harmonics

mid\_term\_of\_fs = no\_of\_fourier\_coeff + 1;

for ii = 1:no\_of\_fourier\_coeff % Fixed range for ii

if ii == 1

harmonics = harmonics + a\_k(mid\_term\_of\_fs) \* exp(1i \* 0 \* omega\_o \* t);

else

harmonics = harmonics + a\_k(mid\_term\_of\_fs - ii + 1) \* exp(1i \* (ii-1) \* omega\_o \* t) + ...

a\_k(mid\_term\_of\_fs + ii - 1) \* exp(-1i \* (ii-1) \* omega\_o \* t);

end

end

% Plot the harmonics

subplot(4,2,[5,6]), plot(t, real(harmonics));

title('Harmonics of the Reconstructed Signal');

% Plot the reconstructed signal vs original signal

reconstructed\_signal = real(harmonics);

subplot(4,2,[7,8]), plot(t, reconstructed\_signal, 'b', t, v\_t, '--r');

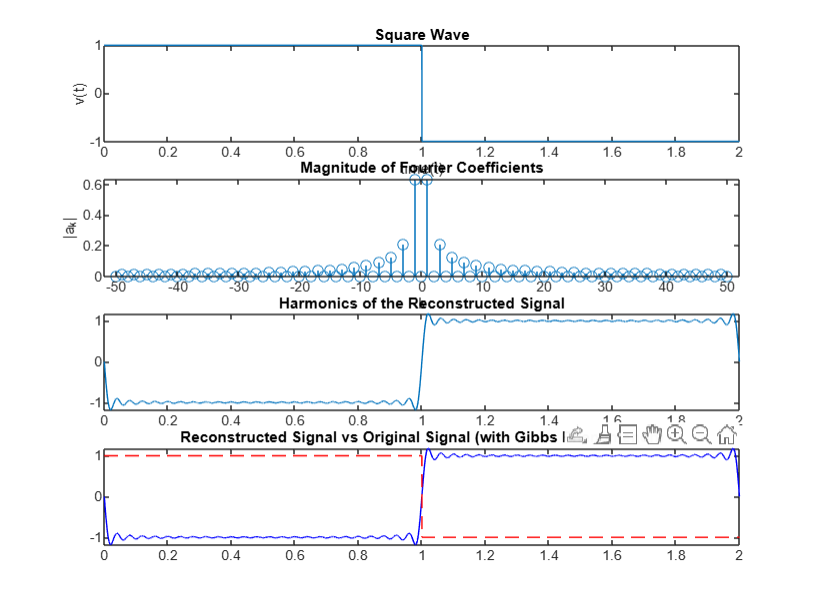
title('Reconstructed Signal vs Original Signal (with Gibbs Phenomenon)');

% Custom integration function

function [int\_ans] = my\_int\_func(Data, step\_size\_t)

int\_ans = (step\_size\_t / 2) \* (Data(1) + Data(end) + 2 \* sum(Data(2:end-1)));

end



GIBBS PHENOMENON – Triangular Wave

clc;

clear all;

close all;

% Time period and other variables

time\_period = 2; % for triangular wave

step\_size\_t = 0.0001;

t = 0:step\_size\_t:time\_period;

tt = length(t);

omega\_o = 2 \* pi / time\_period; % Fundamental angular frequency

% Number of Fourier coefficients

no\_of\_fourier\_coeff = 50;

kk = -no\_of\_fourier\_coeff:1:no\_of\_fourier\_coeff;

% Initialize v\_t for triangular wave signal

v\_t = zeros(1, tt);

for ii = 1:tt

if t(ii) <= time\_period / 2

% Linearly increase from 0 to 1 in the first half of the period

v\_t(ii) = (2 / (time\_period / 2)) \* t(ii);

else

% Linearly decrease from 1 to 0 in the second half of the period

v\_t(ii) = 2 - (2 / (time\_period / 2)) \* t(ii);

end

end

% Plot the triangular wave

subplot(4,2,[1,2]), plot(t, v\_t);

xlabel("time(t)");

ylabel("v(t)");

title('Triangular Wave');

% Fourier coefficient array initialization

a\_k = zeros(1, 2 \* no\_of\_fourier\_coeff + 1); % For storing Fourier coefficients

% Calculate Fourier coefficients

for ii = 1:1:(2 \* no\_of\_fourier\_coeff + 1)

k = kk(ii); % Fourier series index

temp1 = v\_t .\* exp(-1i \* k \* omega\_o \* t); % v(t) \* exp(-j \* k \* omega\_o \* t)

% Integrate over the period using trapezoidal rule

int\_ans = my\_int\_func(temp1, step\_size\_t);

% Calculate Fourier coefficient

a\_k(ii) = (1 / time\_period) \* int\_ans;

end

% Plot Fourier coefficients

subplot(4,2,[3,4]), stem(kk, abs(a\_k));

xlabel('k');

ylabel('|a\_k|');

title('Magnitude of Fourier Coefficients');

% Initialize harmonics

harmonics = zeros(1, tt);

% Reconstruct signal from harmonics

mid\_term\_of\_fs = no\_of\_fourier\_coeff + 1;

for ii = 1:no\_of\_fourier\_coeff % Fixed range for ii

if ii == 1

harmonics = harmonics + a\_k(mid\_term\_of\_fs) \* exp(1i \* 0 \* omega\_o \* t);

else

harmonics = harmonics + a\_k(mid\_term\_of\_fs - ii + 1) \* exp(1i \* (ii-1) \* omega\_o \* t) + ...

a\_k(mid\_term\_of\_fs + ii - 1) \* exp(-1i \* (ii-1) \* omega\_o \* t);

end

end

% Plot the harmonics

subplot(4,2,[5,6]), plot(t, real(harmonics));

title('Harmonics of the Reconstructed Signal');

% Plot the reconstructed signal vs original signal

reconstructed\_signal = real(harmonics);

subplot(4,2,[7,8]), plot(t, reconstructed\_signal, 'b', t, v\_t, '--r');

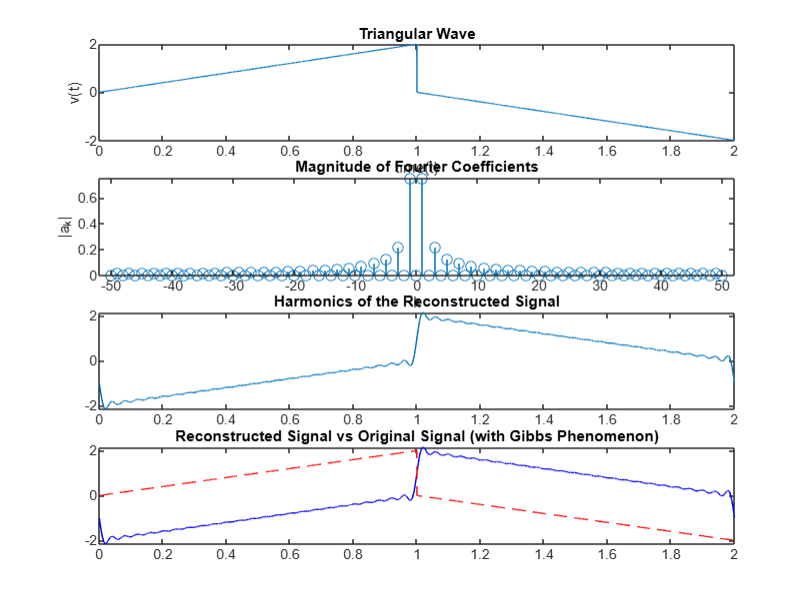
title('Reconstructed Signal vs Original Signal (with Gibbs Phenomenon)');

% Custom integration function

function [int\_ans] = my\_int\_func(Data, step\_size\_t)

int\_ans = (step\_size\_t / 2) \* (Data(1) + Data(end) + 2 \* sum(Data(2:end-1)));

end



GIBBS PHENOMENON – Signum Function

clc;

clear all;

close all;

% Time period and other variables

time\_period = 2; % for signum function representation

step\_size\_t = 0.0001;

t = -time\_period/2:step\_size\_t:time\_period/2; % Symmetric time range around zero

tt = length(t);

omega\_o = 2 \* pi / time\_period; % Fundamental angular frequency

% Number of Fourier coefficients

no\_of\_fourier\_coeff = 50;

kk = -no\_of\_fourier\_coeff:1:no\_of\_fourier\_coeff;

% Initialize v\_t for signum function signal

v\_t = zeros(1, tt);

for ii = 1:tt

if t(ii) < 0

v\_t(ii) = -1; % Negative values

elseif t(ii) == 0

v\_t(ii) = 0; % Zero

else

v\_t(ii) = 1; % Positive values

end

end

% Plot the signum function

subplot(4,2,[1,2]), plot(t, v\_t);

xlabel("time(t)");

ylabel("v(t)");

title('Signum Function');

% Fourier coefficient array initialization

a\_k = zeros(1, 2 \* no\_of\_fourier\_coeff + 1); % For storing Fourier coefficients

% Calculate Fourier coefficients

for ii = 1:1:(2 \* no\_of\_fourier\_coeff + 1)

k = kk(ii); % Fourier series index

temp1 = v\_t .\* exp(-1i \* k \* omega\_o \* t); % v(t) \* exp(-j \* k \* omega\_o \* t)

% Integrate over the period using trapezoidal rule

int\_ans = my\_int\_func(temp1, step\_size\_t);

% Calculate Fourier coefficient

a\_k(ii) = (1 / time\_period) \* int\_ans;

end

% Plot Fourier coefficients

subplot(4,2,[3,4]), stem(kk, abs(a\_k));

xlabel('k');

ylabel('|a\_k|');

title('Magnitude of Fourier Coefficients');

% Initialize harmonics

harmonics = zeros(1, tt);

% Reconstruct signal from harmonics

mid\_term\_of\_fs = no\_of\_fourier\_coeff + 1;

for ii = 1:no\_of\_fourier\_coeff % Fixed range for ii

if ii == 1

harmonics = harmonics + a\_k(mid\_term\_of\_fs) \* exp(1i \* 0 \* omega\_o \* t);

else

harmonics = harmonics + a\_k(mid\_term\_of\_fs - ii + 1) \* exp(1i \* (ii-1) \* omega\_o \* t) + ...

a\_k(mid\_term\_of\_fs + ii - 1) \* exp(-1i \* (ii-1) \* omega\_o \* t);

end

end

% Plot the harmonics

subplot(4,2,[5,6]), plot(t, real(harmonics))

title('Harmonics of the Reconstructed Signal');

% Plot the reconstructed signal vs original signal

reconstructed\_signal = real(harmonics);

subplot(4,2,[7,8]), plot(t, reconstructed\_signal, 'b', t, v\_t, '--r');

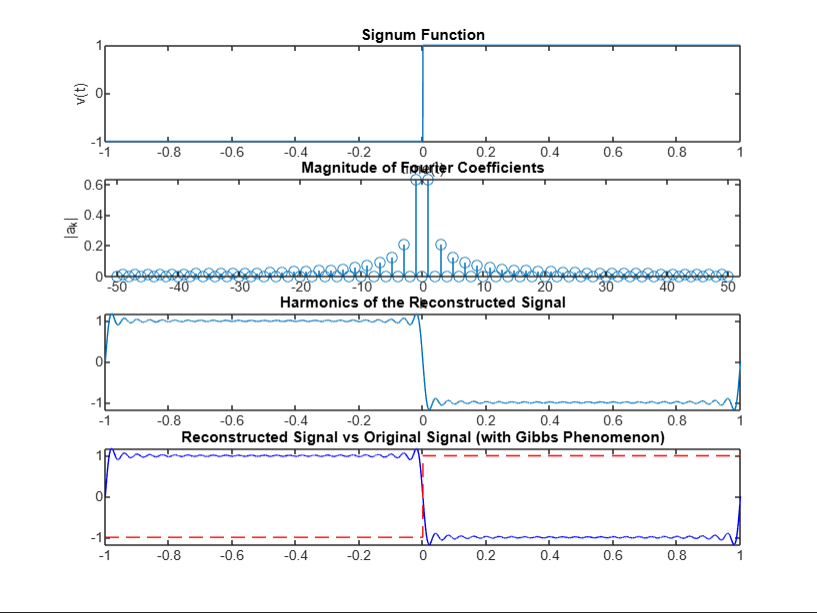
title('Reconstructed Signal vs Original Signal (with Gibbs Phenomenon)');

\% Custom integration function

function [int\_ans] = my\_int\_func(Data, step\_size\_t)

int\_ans = (step\_size\_t / 2) \* (Data(1) + Data(end) + 2 \* sum(Data(2:end-1)));

end



GIBBS PHENOMENON – Unit Step Function

clc;

clear all;

close all;

% Time period and other variables

time\_period = 2; % for unit step function representation

step\_size\_t = 0.0001;

t = -time\_period/2:step\_size\_t:time\_period/2; % Symmetric time range around zero

tt = length(t);

omega\_o = 2 \* pi / time\_period; % Fundamental angular frequency

% Number of Fourier coefficients

no\_of\_fourier\_coeff = 50;

kk = -no\_of\_fourier\_coeff:1:no\_of\_fourier\_coeff;

% Initialize v\_t for unit step function

v\_t = zeros(1, tt);

for ii = 1:tt

if t(ii) < 0

v\_t(ii) = 0; % Values before 0

else

v\_t(ii) = 1; % Values from 0 onward

end

end

% Plot the unit step function

subplot(4,2,[1,2]), plot(t, v\_t);

xlabel("time(t)");

ylabel("v(t)");

title('Unit Step Function');

% Fourier coefficient array initialization

a\_k = zeros(1, 2 \* no\_of\_fourier\_coeff + 1); % For storing Fourier coefficients

% Calculate Fourier coefficients

for ii = 1:1:(2 \* no\_of\_fourier\_coeff + 1)

k = kk(ii); % Fourier series index

temp1 = v\_t .\* exp(-1i \* k \* omega\_o \* t); % v(t) \* exp(-j \* k \* omega\_o \* t)

% Integrate over the period using trapezoidal rule

int\_ans = my\_int\_func(temp1, step\_size\_t);

% Calculate Fourier coefficient

a\_k(ii) = (1 / time\_period) \* int\_ans;

end

% Plot Fourier coefficients

subplot(4,2,[3,4]), stem(kk, abs(a\_k));

xlabel('k');

ylabel('|a\_k|');

title('Magnitude of Fourier Coefficients');

% Initialize harmonics

harmonics = zeros(1, tt);

% Reconstruct signal from harmonics

mid\_term\_of\_fs = no\_of\_fourier\_coeff + 1;

for ii = 1:no\_of\_fourier\_coeff

if ii == 1

harmonics = harmonics + a\_k(mid\_term\_of\_fs) \* exp(1i \* 0 \* omega\_o \* t);

else

harmonics = harmonics + a\_k(mid\_term\_of\_fs - ii + 1) \* exp(1i \* (ii-1) \* omega\_o \* t) + ...

a\_k(mid\_term\_of\_fs + ii - 1) \* exp(-1i \* (ii-1) \* omega\_o \* t);

end

end

% Plot the harmonics

subplot(4,2,[5,6]), plot(t, real(harmonics));

title('Harmonics of the Reconstructed Signal');

% Plot the reconstructed signal vs original signal

reconstructed\_signal = real(harmonics);

subplot(4,2,[7,8]), plot(t, reconstructed\_signal, 'b', t, v\_t, '--r');

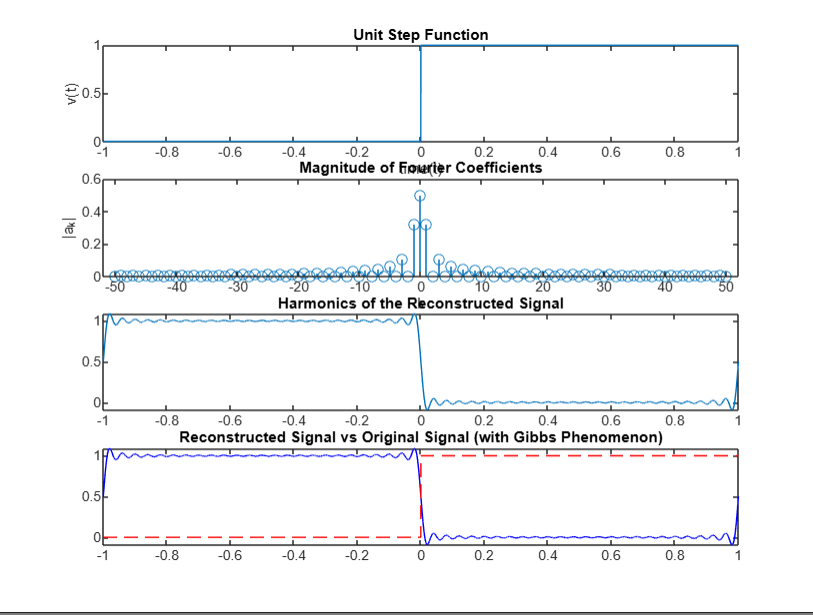
title('Reconstructed Signal vs Original Signal (with Gibbs Phenomenon)');

% Custom integration function

function [int\_ans] = my\_int\_func(Data, step\_size\_t)

int\_ans = (step\_size\_t / 2) \* (Data(1) + Data(end) + 2 \* sum(Data(2:end-1)));

end



GIBBS PHENOMENON – Rectified Sine Wave

clc;

clear all;

close all;

% Time period and other variables

time\_period = 2.\*pi; % Time period for rectified sine function representation

step\_size\_t = 0.0001;

t = -time\_period/2:step\_size\_t:time\_period/2; % Symmetric time range around zero

tt = length(t);

omega\_o = 2 \* pi / time\_period; % Fundamental angular frequency

% Number of Fourier coefficients

no\_of\_fourier\_coeff = 50;

kk = -no\_of\_fourier\_coeff:1:no\_of\_fourier\_coeff;

% Initialize v\_t for rectified sine function

v\_t\_sine = zeros(1, tt); % Preallocate v\_t\_sine with zeros

for ii = 1:tt

if t(ii) >= 0 && t(ii) <= pi

v\_t\_sine(ii) = sin(t(ii)); % Correct assignment with indexing

else

v\_t\_sine(ii) = 0; % Assign zero for negative values

end

end

% Plot the rectified sine function

subplot(4,2,[1,2]), plot(t, v\_t\_sine);

xlabel("time(t)");

ylabel("v(t)");

title('Rectified Sine Function');

% Fourier coefficient array initialization

a\_k = zeros(1, 2 \* no\_of\_fourier\_coeff + 1); % For storing Fourier coefficients

% Calculate Fourier coefficients

for ii = 1:(2 \* no\_of\_fourier\_coeff + 1)

k = kk(ii); % Fourier series index

temp1 = v\_t\_sine .\* exp(-1i \* k \* omega\_o \* t); % v(t) \* exp(-j \* k \* omega\_o \* t)

% Integrate over the period using trapezoidal rule

int\_ans = my\_int\_func(temp1, step\_size\_t);

% Calculate Fourier coefficient

a\_k(ii) = (1 / time\_period) \* int\_ans;

end

% Plot Fourier coefficients

subplot(4,2,[3,4]), stem(kk, abs(a\_k));

xlabel('k');

ylabel('|a\_k|');

title('Magnitude of Fourier Coefficients');

% Initialize harmonics

harmonics = zeros(1, tt);

% Reconstruct signal from harmonics

mid\_term\_of\_fs = no\_of\_fourier\_coeff + 1;

for ii = 1:no\_of\_fourier\_coeff

if ii == 1

harmonics = harmonics + a\_k(mid\_term\_of\_fs) \* exp(1i \* 0 \* omega\_o \* t);

else

harmonics = harmonics + a\_k(mid\_term\_of\_fs - ii + 1) \* exp(1i \* (ii-1) \* omega\_o \* t) + ...

a\_k(mid\_term\_of\_fs + ii - 1) \* exp(-1i \* (ii-1) \* omega\_o \* t);

end

end

% Plot the harmonics

subplot(4,2,[5,6]), plot(t, real(harmonics));

title('Harmonics of the Reconstructed Signal');

% Plot the reconstructed signal vs original signal

reconstructed\_signal = real(harmonics);

subplot(4,2,[7,8]), plot(t, reconstructed\_signal, 'b', t, v\_t\_sine, '--r');

xlabel('time(t)');

ylabel('Signal Amplitude');

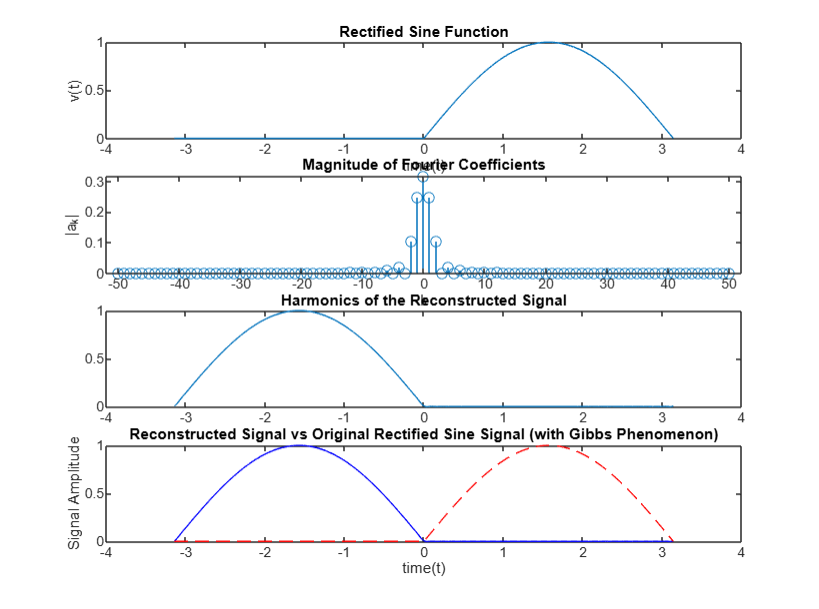
title('Reconstructed Signal vs Original Rectified Sine Signal (with Gibbs Phenomenon)');

% Custom integration function

function [int\_ans] = my\_int\_func(Data, step\_size\_t)

int\_ans = (step\_size\_t / 2) \* (Data(1) + Data(end) + 2 \* sum(Data(2:end-1)));

end



GIBBS PHENOMENON – Rectified Cosine Wave

clc;

clear all;

close all;

% Time period and other variables

time\_period = 2.\*pi; % Time period for rectified cosine function representation

step\_size\_t = 0.0001;

t = -time\_period/2:step\_size\_t:time\_period/2; % Symmetric time range around zero

tt = length(t);

omega\_o = 2 \* pi / time\_period; % Fundamental angular frequency

% Number of Fourier coefficients

no\_of\_fourier\_coeff = 50;

kk = -no\_of\_fourier\_coeff:1:no\_of\_fourier\_coeff;

% Initialize v\_t for rectified cosine function

v\_t\_cosine = zeros(1, tt); % Preallocate v\_t\_cosine with zeros

for ii = 1:tt

if t(ii) >= -pi/2 && t(ii) <= pi/2

v\_t\_cosine(ii) = cos(t(ii)); % Rectified cosine function

else

v\_t\_cosine(ii) = 0; % Assign zero for negative values

end

end

% Plot the rectified cosine function

subplot(4,2,[1,2]), plot(t, v\_t\_cosine);

xlabel("time(t)");

ylabel("v(t)");

title('Rectified Cosine Function');

% Fourier coefficient array initialization

a\_k = zeros(1, 2 \* no\_of\_fourier\_coeff + 1); % For storing Fourier coefficients

% Calculate Fourier coefficients

for ii = 1:(2 \* no\_of\_fourier\_coeff + 1)

k = kk(ii); % Fourier series index

temp1 = v\_t\_cosine .\* exp(-1i \* k \* omega\_o \* t); % v(t) \* exp(-j \* k \* omega\_o \* t)

% Integrate over the period using trapezoidal rule

int\_ans = my\_int\_func(temp1, step\_size\_t);

% Calculate Fourier coefficient

a\_k(ii) = (1 / time\_period) \* int\_ans;

end

% Plot Fourier coefficients

subplot(4,2,[3,4]), stem(kk, abs(a\_k));

xlabel('k');

ylabel('|a\_k|');

title('Magnitude of Fourier Coefficients');

% Initialize harmonics

harmonics = zeros(1, tt);

% Reconstruct signal from harmonics

mid\_term\_of\_fs = no\_of\_fourier\_coeff + 1;

for ii = 1:no\_of\_fourier\_coeff

if ii == 1

harmonics = harmonics + a\_k(mid\_term\_of\_fs) \* exp(1i \* 0 \* omega\_o \* t);

else

harmonics = harmonics + a\_k(mid\_term\_of\_fs - ii + 1) \* exp(1i \* (ii-1) \* omega\_o \* t) + ...

a\_k(mid\_term\_of\_fs + ii - 1) \* exp(-1i \* (ii-1) \* omega\_o \* t);

end

end

% Plot the harmonics

subplot(4,2,[5,6]), plot(t, real(harmonics));

title('Harmonics of the Reconstructed Signal');

% Plot the reconstructed signal vs original signal

reconstructed\_signal = real(harmonics);

subplot(4,2,[7,8]), plot(t, reconstructed\_signal, 'b', t, v\_t\_cosine, '--r');

xlabel('time(t)');

ylabel('Signal Amplitude');

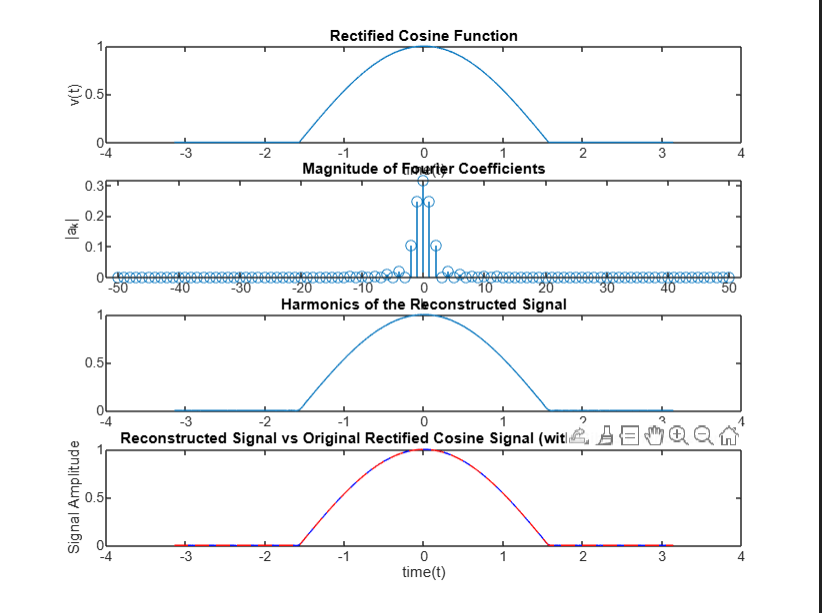
title('Reconstructed Signal vs Original Rectified Cosine Signal (with Gibbs Phenomenon)');

% Custom integration function

function [int\_ans] = my\_int\_func(Data, step\_size\_t)

int\_ans = (step\_size\_t / 2) \* (Data(1) + Data(end) + 2 \* sum(Data(2:end-1)));

end



FOURIER TRANSFORMS – Sinc Function

clear all;

close all;

clc

% Constants

step\_omega= 0.01 \* pi;

step\_size\_t = 0.1;

t = -10:step\_size\_t:10;

length\_t = length(t);

% Omega range for frequency domain

omegax = -(1 / step\_size\_t) \* pi : step\_omega : (1 / step\_size\_t) \* pi;

length\_omega = length(omegax);

% Pre-allocate arrays

expo\_omega = zeros(length\_omega, length\_t);

x\_omega = zeros(1, length\_omega);

x\_t\_reconstructed = zeros(1, length\_t);

% Create Fourier exponentials

for ii = 1:length\_omega

expo\_omega(ii, :) = exp(-1j \* omegax(ii) \* t);

end

% Pre-allocate array for sinc function

x\_t = zeros(1, length\_t);

% Define sinc function in time domain

for ii = 1:length\_t

if t(ii) == 0

x\_t(ii) = 1; % Define sinc(0) = 1

else

% Corrected sinc function formula (as per original structure)

x\_t(ii) = sin((2 \* pi \* (1/(2 \* step\_size\_t))) \* 0.5 \* t(ii)) / ...

((2 \* pi \* (1/(2 \* step\_size\_t))) \* 0.5 \* t(ii));

end

end

% Plot the sinc function

subplot(4,1,1), plot(t, x\_t);

xlabel('time (t)');

ylabel('x(t)');

title('Sinc Function in Time Domain');

% Compute Fourier Transform X(ω)

for ii = 1:length\_omega

temp = x\_t .\* expo\_omega(ii, :); % x(t) \* exp(-jωt)

x\_omega(ii) = my\_int\_func(temp, step\_size\_t); % Fourier Transform

end

% Plot the magnitude of the Fourier transform

subplot(4,1,2), plot(omegax ./ pi, abs(x\_omega));

xlabel('\omega/\pi');

ylabel('|X(\omega)|');

title('Fourier Transform of the Sinc Function');

% Inverse Fourier Transform (reconstruct x(t))

expo\_omega\_2 = exp(1j \* omegax' \* t); % Create matrix of exp(jωt) for inverse transform

for ii = 1:length\_t

temp\_2 = x\_omega .\* expo\_omega\_2(:, ii).'; % X(ω) \* exp(jωt)

x\_t\_reconstructed(ii) = (1 / (2 \* pi)) \* my\_int\_func(temp\_2, step\_omega); % Inverse Fourier Transform

end

% Plot the reconstructed signal

subplot(4,1,3), plot(t, real(x\_t\_reconstructed));

xlabel('time (t)');

ylabel('Reconstructed x(t)');

title('Reconstructed Signal from Fourier Transform');

% Calculate and plot the phase of X(omega)

Angle\_X\_Omega = atan2(imag(x\_omega), real(x\_omega)); % Compute phase using atan2 for correct quadrant

% Plot the phase of the Fourier transform

subplot(4,1,4), plot(omegax ./ pi, Angle\_X\_Omega);

xlabel('\omega/\pi');

ylabel('Phase \angle X(\omega) (radians)');

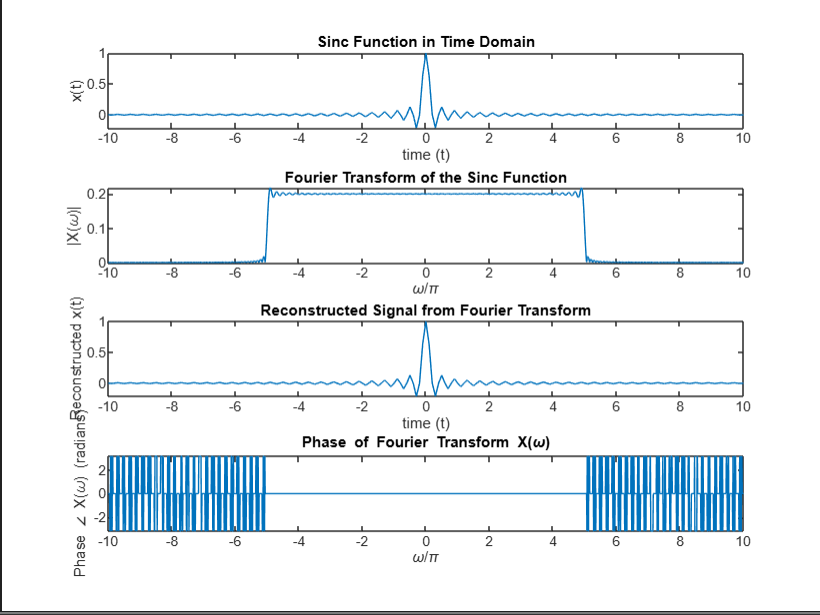
title('Phase of Fourier Transform X(\omega)');

%Custom integration function (trapezoidal rule)

function [int\_ans] = my\_int\_func(Data, step\_size)

int\_ans = (step\_size / 2) \* (Data(1) + Data(end) + 2 \* sum(Data(2:end-1)));

end



FOURIER TRANSFORMS – Exponential Decay Function

clear all;

close all;

clc

% Constants

step\_omega = 0.01 \* pi;

step\_size\_t = 0.1;

t = -10:step\_size\_t:10;

length\_t = length(t);

% Omega range for frequency domain

omegax = -(1 / step\_size\_t) \* pi : step\_omega : (1 / step\_size\_t) \* pi;

length\_omega = length(omegax);

% Pre-allocate arrays

expo\_omega = zeros(length\_omega, length\_t);

x\_omega = zeros(1, length\_omega);

x\_t\_reconstructed = zeros(1, length\_t);

% Create Fourier exponentials

for ii = 1:length\_omega

expo\_omega(ii, :) = exp(-1j \* omegax(ii) \* t);

end

% Define exponential decay function

for ii = 1:length\_t

if t(ii) < 0

x\_t(ii) = 0; % Set to 0 for t < 0

else

x\_t(ii) = 3 \* exp(-2 \* t(ii)); % Exponential decay for t >= 0

end

end

% Plot the exponential decay function

subplot(4,1,1), plot(t, x\_t);

xlabel('time (t)');

ylabel('x(t)');

title('Exponential Decay Function in Time Domain');

% Compute Fourier Transform X(ω)

for ii = 1:length\_omega

temp = x\_t .\* expo\_omega(ii, :); % x(t) \* exp(-jωt)

x\_omega(ii) = my\_int\_func(temp, step\_size\_t); % Fourier Transform

end

% Plot the magnitude of the Fourier transform

subplot(4,1,2), plot(omegax ./ pi, abs(x\_omega));

xlabel('\omega/\pi');

ylabel('|X(\omega)|');

title('Fourier Transform of the Exponential Decay Function');

% Inverse Fourier Transform (reconstruct x(t))

expo\_omega\_2 = exp(1j \* omegax' \* t); % Create matrix of exp(jωt) for inverse transform

for ii = 1:length\_t

temp\_2 = x\_omega .\* expo\_omega\_2(:, ii).'; % X(ω) \* exp(jωt)

x\_t\_reconstructed(ii) = (1 / (2 \* pi)) \* my\_int\_func(temp\_2, step\_omega); % Inverse Fourier Transform

end

% Plot the reconstructed signal

subplot(4,1,3), plot(t, real(x\_t\_reconstructed));

xlabel('time (t)');

ylabel('Reconstructed x(t)');

title('Reconstructed Signal from Fourier Transform');

% Calculate and plot the phase of X(omega)

Angle\_X\_Omega = atan2(imag(x\_omega), real(x\_omega)); % Compute phase using atan2 for correct quadrant

% Plot the phase of the Fourier transform

subplot(4,1,4), plot(omegax ./ pi, Angle\_X\_Omega);

xlabel('\omega/\pi');

ylabel('Phase \angle X(\omega) (radians)');

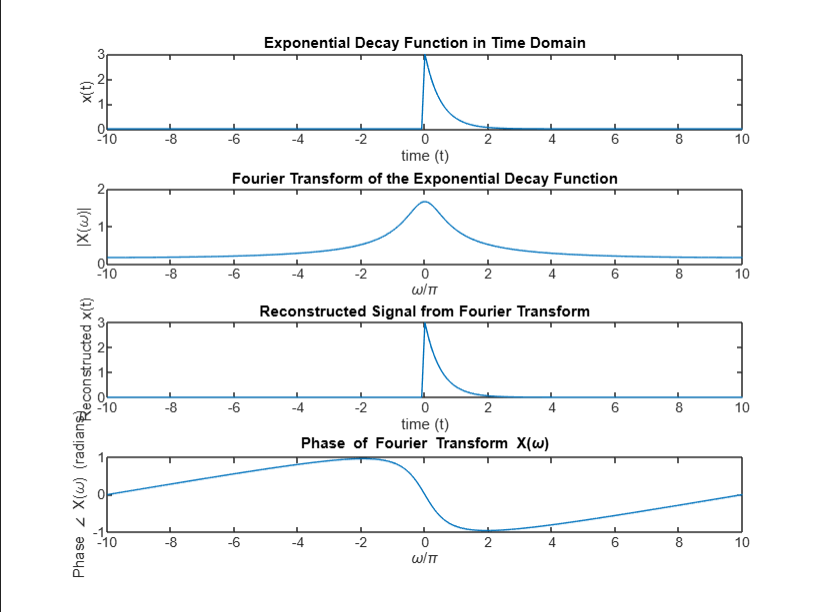
title('Phase of Fourier Transform X(\omega)');

% Custom integration function (trapezoidal rule)

function [int\_ans] = my\_int\_func(Data, step\_size)

int\_ans = (step\_size / 2) \* (Data(1) + Data(end) + 2 \* sum(Data(2:end-1)));

end



FOURIER TRANSFORMS – Double-Sided Exponential Decay Function

clear all;

close all;

clc

% Constants

step\_omega = 0.01 \* pi;

step\_size\_t = 0.1;

t = -10:step\_size\_t:10;

length\_t = length(t);

% Omega range for frequency domain

omegax = -(1 / step\_size\_t) \* pi : step\_omega : (1 / step\_size\_t) \* pi;

length\_omega = length(omegax);

% Pre-allocate arrays

expo\_omega = zeros(length\_omega, length\_t);

x\_omega = zeros(1, length\_omega);

x\_t\_reconstructed = zeros(1, length\_t);

% Create Fourier exponentials

for ii = 1:length\_omega

expo\_omega(ii, :) = exp(-1j \* omegax(ii) \* t);

end

% Define double-sided exponential function

x\_t = zeros(1, length\_t); % Initialize x\_t with zeros

% Define exponential growth for t < 0 and decay for t >= 0

for ii = 1:length\_t

if t(ii) < 0

x\_t(ii) = 3 \* exp(2 \* t(ii)); % Exponential growth for t < 0

else

x\_t(ii) = 3 \* exp(-2 \* t(ii)); % Exponential decay for t >= 0

end

end

% Plot the double-sided exponential function

subplot(4,1,1), plot(t, x\_t);

xlabel('time (t)');

ylabel('x(t)');

title('Double-Sided Exponential Function in Time Domain');

% Compute Fourier Transform X(ω)

for ii = 1:length\_omega

temp = x\_t .\* expo\_omega(ii, :); % x(t) \* exp(-jωt)

x\_omega(ii) = my\_int\_func(temp, step\_size\_t); % Fourier Transform

end

% Plot the magnitude of the Fourier transform

subplot(4,1,2), plot(omegax ./ pi, abs(x\_omega));

xlabel('\omega/\pi');

ylabel('|X(\omega)|');

title('Fourier Transform of the Double-Sided Exponential Function');

% Inverse Fourier Transform (reconstruct x(t))

expo\_omega\_2 = exp(1j \* omegax' \* t); % Create matrix of exp(jωt) for inverse transform

for ii = 1:length\_t

temp\_2 = x\_omega .\* expo\_omega\_2(:, ii).'; % X(ω) \* exp(jωt)

x\_t\_reconstructed(ii) = (1 / (2 \* pi)) \* my\_int\_func(temp\_2, step\_omega); % Inverse Fourier Transform

end

% Plot the reconstructed signal

subplot(4,1,3), plot(t, real(x\_t\_reconstructed));

xlabel('time (t)');

ylabel('Reconstructed x(t)');

title('Reconstructed Signal from Fourier Transform');

% Calculate and plot the phase of X(omega)

Angle\_X\_Omega = atan2(imag(x\_omega), real(x\_omega)); % Compute phase using atan2 for correct quadrant

% Plot the phase of the Fourier transform

subplot(4,1,4), plot(omegax ./ pi, Angle\_X\_Omega);

xlabel('\omega/\pi');

ylabel('Phase \angle X(\omega) (radians)');

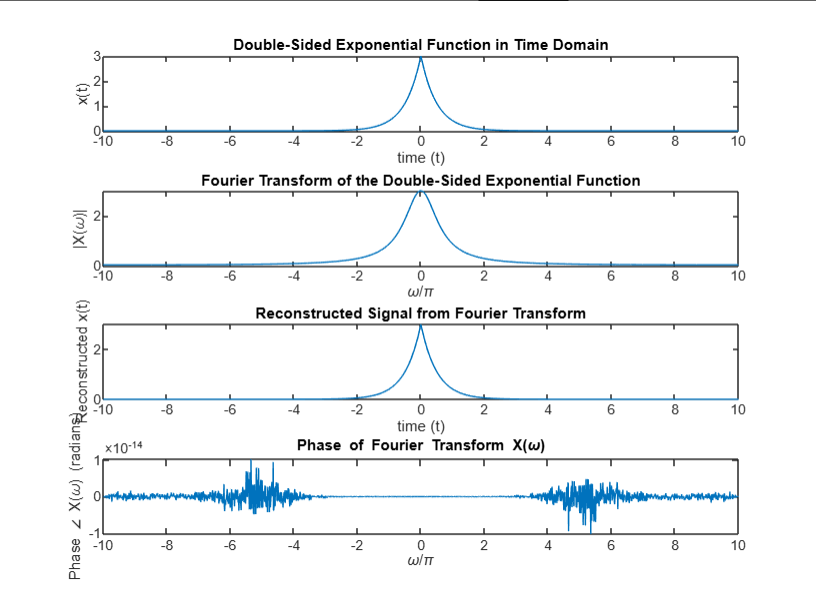
title('Phase of Fourier Transform X(\omega)');

% Custom integration function (trapezoidal rule)

function [int\_ans] = my\_int\_func(Data, step\_size)

int\_ans = (step\_size / 2) \* (Data(1) + Data(end) + 2 \* sum(Data(2:end-1)));

end



FOURIER TRANSFORMS – Sine Function

clear all;

close all;

clc

% Constants

step\_omega = 0.01 \* pi;

step\_size\_t = 0.1;

t = -10:step\_size\_t:10;

length\_t = length(t);

% Omega range for frequency domain

omegax = -(1 / step\_size\_t) \* pi : step\_omega : (1 / step\_size\_t) \* pi;

length\_omega = length(omegax);

% Pre-allocate arrays

expo\_omega = zeros(length\_omega, length\_t);

x\_omega = zeros(1, length\_omega);

x\_t\_reconstructed = zeros(1, length\_t);

% Create Fourier exponentials

for ii = 1:length\_omega

expo\_omega(ii, :) = exp(-1j \* omegax(ii) \* t);

end

% Define sine function within a specified range

x\_t = zeros(1, length\_t); % Initialize x\_t with zeros

for ii = 1:length\_t

x\_t(ii) = sin(t(ii));

end

% Plot the sine function in the time domain

subplot(4,1,1), plot(t, x\_t);

xlabel('time (t)');

ylabel('x(t)');

title('Sine Function in Time Domain');

% Compute Fourier Transform X(ω)

for ii = 1:length\_omega

temp = x\_t .\* expo\_omega(ii, :); % x(t) \* exp(-jωt)

x\_omega(ii) = my\_int\_func(temp, step\_size\_t); % Fourier Transform

end

% Plot the magnitude of the Fourier transform

subplot(4,1,2), plot(omegax ./ pi, abs(x\_omega));

xlabel('\omega/\pi');

ylabel('|X(\omega)|');

title('Fourier Transform of the Sine Function');

% Inverse Fourier Transform (reconstruct x(t))

expo\_omega\_2 = exp(1j \* omegax' \* t); % Create matrix of exp(jωt) for inverse transform

for ii = 1:length\_t

temp\_2 = x\_omega .\* expo\_omega\_2(:, ii).'; % X(ω) \* exp(jωt)

x\_t\_reconstructed(ii) = (1 / (2 \* pi)) \* my\_int\_func(temp\_2, step\_omega); % Inverse Fourier Transform

end

% Plot the reconstructed signal

subplot(4,1,3), plot(t, real(x\_t\_reconstructed));

xlabel('time (t)');

ylabel('Reconstructed x(t)');

title('Reconstructed Signal from Fourier Transform');

% Calculate and plot the phase of X(omega)

Angle\_X\_Omega = atan2(imag(x\_omega), real(x\_omega)); % Compute phase using atan2 for correct quadrant

% Plot the phase of the Fourier transform

subplot(4,1,4), plot(omegax ./ pi, Angle\_X\_Omega);

xlabel('\omega/\pi');

ylabel('Phase \angle X(\omega) (radians)');

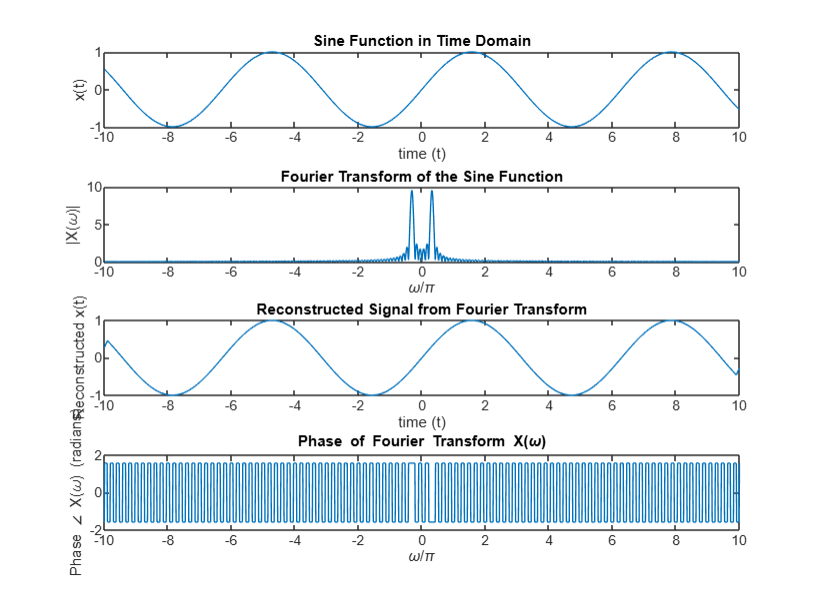
title('Phase of Fourier Transform X(\omega)');

% Custom integration function (trapezoidal rule)

function [int\_ans] = my\_int\_func(Data, step\_size)

int\_ans = (step\_size / 2) \* (Data(1) + Data(end) + 2 \* sum(Data(2:end-1)));

end



FOURIER TRANSFORMS – Cosine Function

clear all;

close all;

clc

% Constants

step\_omega = 0.01 \* pi;

step\_size\_t = 0.1;

t = -10:step\_size\_t:10; % Time vector

length\_t = length(t); % Length of time vector

% Omega range for frequency domain

omegax = -(1 / step\_size\_t) \* pi : step\_omega : (1 / step\_size\_t) \* pi;

length\_omega = length(omegax);

% Pre-allocate arrays

expo\_omega = zeros(length\_omega, length\_t);

x\_omega = zeros(1, length\_omega);

x\_t\_reconstructed = zeros(1, length\_t);

% Create Fourier exponentials

for ii = 1:length\_omega

expo\_omega(ii, :) = exp(-1j \* omegax(ii) \* t); % Exponential matrix for Fourier Transform

end

% Define cosine function

x\_t = zeros(1, length\_t); % Initialize x\_t with zeros

for ii = 1:length\_t

x\_t(ii) = cos(t(ii)); % Cosine function

end

% Plot the cosine function in the time domain

subplot(4,1,1), plot(t, x\_t);

xlabel('time (t)');

ylabel('x(t)');

title('Cosine Function in Time Domain'); % Updated title for cosine function

% Compute Fourier Transform X(ω)

for ii = 1:length\_omega

temp = x\_t .\* expo\_omega(ii, :); % x(t) \* exp(-jωt)

x\_omega(ii) = my\_int\_func(temp, step\_size\_t); % Fourier Transform

end

% Plot the magnitude of the Fourier transform

subplot(4,1,2), plot(omegax ./ pi, abs(x\_omega));

xlabel('\omega/\pi');

ylabel('|X(\omega)|');

title('Fourier Transform of the Cosine Function'); % Updated title for cosine function

% Inverse Fourier Transform (reconstruct x(t))

expo\_omega\_2 = exp(1j \* omegax' \* t); % Create matrix of exp(jωt) for inverse transform

for ii = 1:length\_t

temp\_2 = x\_omega .\* expo\_omega\_2(:, ii).'; % X(ω) \* exp(jωt)

x\_t\_reconstructed(ii) = (1 / (2 \* pi)) \* my\_int\_func(temp\_2, step\_omega); % Inverse Fourier Transform

end

% Plot the reconstructed signal

subplot(4,1,3), plot(t, real(x\_t\_reconstructed));

xlabel('time (t)');

ylabel('Reconstructed x(t)');

title('Reconstructed Signal from Fourier Transform'); % Title for reconstructed signal

% Calculate and plot the phase of X(ω)

Angle\_X\_Omega = atan2(imag(x\_omega), real(x\_omega)); % Compute phase using atan2 for correct quadrant

% Plot the phase of the Fourier transform

subplot(4,1,4), plot(omegax ./ pi, Angle\_X\_Omega);

xlabel('\omega/\pi');

ylabel('Phase \angle X(\omega) (radians)');

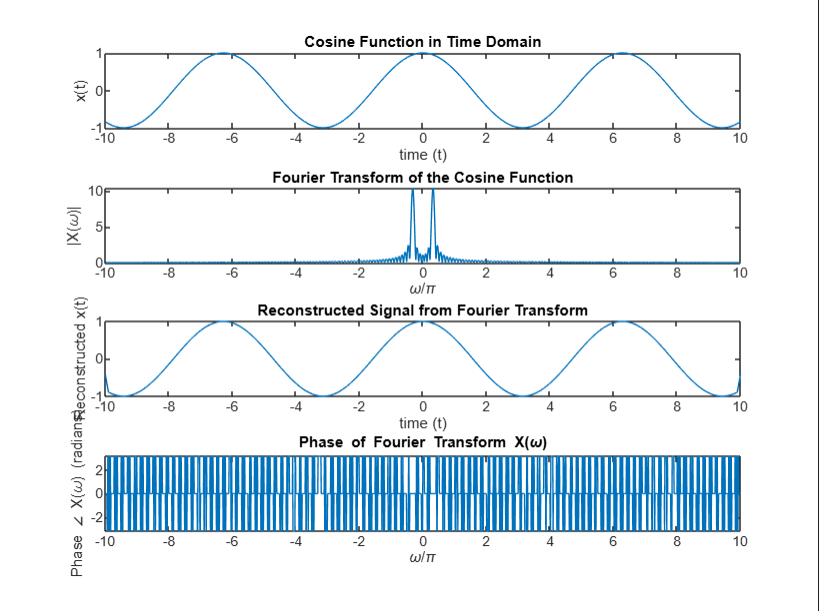
title('Phase of Fourier Transform X(\omega)'); % Title for phase plot

% Custom integration function (trapezoidal rule)

function [int\_ans] = my\_int\_func(Data, step\_size)

int\_ans = (step\_size / 2) \* (Data(1) + Data(end) + 2 \* sum(Data(2:end-1))); % Trapezoidal rule

end



FOURIER TRANSFORMS – Unit Step Function

clear all;

close all;

clc;

% Constants

step\_omega = 0.01 \* pi;

step\_size\_t = 0.1;

t = -10:step\_size\_t:10; % Time vector

length\_t = length(t); % Length of time vector

% Omega range for frequency domain

omegax = -(1 / step\_size\_t) \* pi : step\_omega : (1 / step\_size\_t) \* pi;

length\_omega = length(omegax);

% Pre-allocate arrays

expo\_omega = zeros(length\_omega, length\_t);

x\_omega = zeros(1, length\_omega);

x\_t\_reconstructed = zeros(1, length\_t);

x\_t = zeros(1, length\_t); % Initialize x(t)

% Create Fourier exponentials

for ii = 1:length\_omega

expo\_omega(ii, :) = exp(-1j \* omegax(ii) \* t); % Exponential matrix for Fourier Transform

end

% Define Unit Step Function

for ii = 1:length\_t % Use length\_t instead of undefined tt

if t(ii) < 0

x\_t(ii) = 0; % Values before 0

else

x\_t(ii) = 1; % Values from 0 onward

end

end

% Plot the unit step function in the time domain

subplot(4,1,1), plot(t, x\_t);

xlabel('time (t)');

ylabel('x(t)');

title('Unit Step Function in Time Domain'); % Updated title for unit step function

% Compute Fourier Transform X(ω)

for ii = 1:length\_omega

temp = x\_t .\* expo\_omega(ii, :); % x(t) \* exp(-jωt)

x\_omega(ii) = my\_int\_func(temp, step\_size\_t); % Fourier Transform

end

% Plot the magnitude of the Fourier transform

subplot(4,1,2), plot(omegax ./ pi, abs(x\_omega));

xlabel('\omega/\pi');

ylabel('|X(\omega)|');

title('Magnitude of Fourier Transform of Unit Step Function'); % Updated title

% Inverse Fourier Transform (reconstruct x(t))

expo\_omega\_2 = exp(1j \* omegax' \* t); % Create matrix of exp(jωt) for inverse transform

for ii = 1:length\_t

temp\_2 = x\_omega .\* expo\_omega\_2(:, ii).'; % X(ω) \* exp(jωt)

x\_t\_reconstructed(ii) = (1 / (2 \* pi)) \* my\_int\_func(temp\_2, step\_omega); % Inverse Fourier Transform

end

% Plot the reconstructed signal

subplot(4,1,3), plot(t, real(x\_t\_reconstructed));

xlabel('time (t)');

ylabel('Reconstructed x(t)');

title('Reconstructed Signal from Fourier Transform'); % Title for reconstructed signal

% Calculate and plot the phase of X(ω)

Angle\_X\_Omega = atan2(imag(x\_omega), real(x\_omega)); % Compute phase using atan2 for correct quadrant

% Plot the phase of the Fourier transform

subplot(4,1,4), plot(omegax ./ pi, Angle\_X\_Omega);

xlabel('\omega/\pi');

ylabel('Phase \angle X(\omega) (radians)');

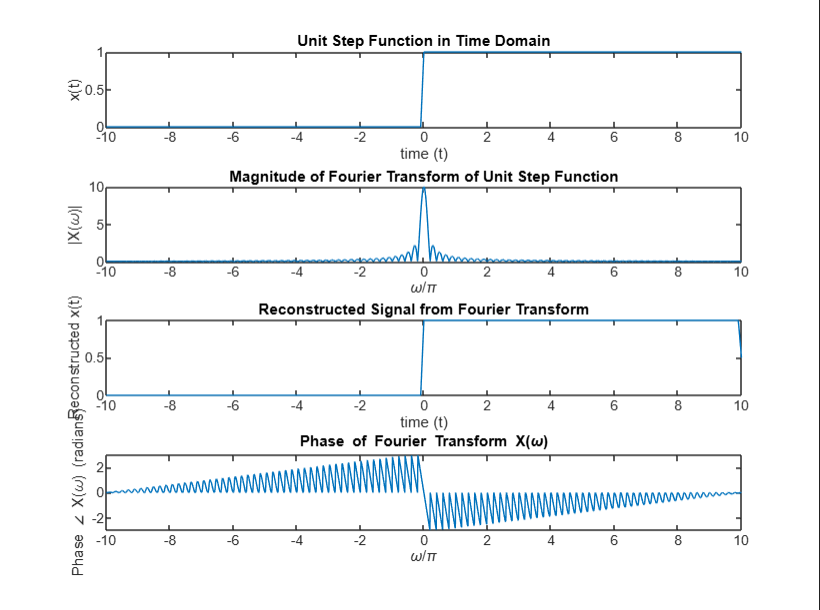
title('Phase of Fourier Transform X(\omega)'); % Title for phase plot

% Custom integration function (trapezoidal rule)

function [int\_ans] = my\_int\_func(Data, step\_size)

int\_ans = (step\_size / 2) \* (Data(1) + Data(end) + 2 \* sum(Data(2:end-1))); % Trapezoidal rule

end



FOURIER TRANSFORMS – Signum Function

clear all;

close all;

clc;

% Constants

step\_omega = 0.01 \* pi;

step\_size\_t = 0.1;

t = -10:step\_size\_t:10; % Time vector

length\_t = length(t); % Length of time vector

% Omega range for frequency domain

omegax = -(1 / step\_size\_t) \* pi : step\_omega : (1 / step\_size\_t) \* pi;

length\_omega = length(omegax);

% Pre-allocate arrays

x\_omega = zeros(1, length\_omega);

x\_t\_reconstructed = zeros(1, length\_t);

v\_t = zeros(1, length\_t); % Initialize signum function

% Initialize v\_t for signum function signal

for ii = 1:length\_t % Use length\_t instead of undefined tt

if t(ii) < 0

v\_t(ii) = -1; % Negative values

elseif t(ii) == 0

v\_t(ii) = 0; % Zero

else

v\_t(ii) = 1; % Positive values

end

end

% Plot the signum function in the time domain

subplot(4,1,1), plot(t, v\_t);

xlabel('time (t)');

ylabel('v(t)');

title('Signum Function in Time Domain'); % Updated title for signum function

% Compute Fourier Transform X(ω)

for ii = 1:length\_omega

temp = v\_t .\* exp(-1j \* omegax(ii) \* t); % v(t) \* exp(-jωt)

x\_omega(ii) = my\_int\_func(temp, step\_size\_t); % Fourier Transform

end

% Plot the magnitude of the Fourier transform

subplot(4,1,2), plot(omegax ./ pi, abs(x\_omega));

xlabel('\omega/\pi');

ylabel('|X(\omega)|');

title('Magnitude of Fourier Transform of Signum Function'); % Updated title

% Inverse Fourier Transform (reconstruct v(t))

expo\_omega\_2 = exp(1j \* omegax' \* t); % Create matrix of exp(jωt) for inverse transform

for ii = 1:length\_t

temp\_2 = x\_omega .\* expo\_omega\_2(:, ii).'; % X(ω) \* exp(jωt)

x\_t\_reconstructed(ii) = (1 / (2 \* pi)) \* my\_int\_func(temp\_2, step\_omega); % Inverse Fourier Transform

end

% Plot the reconstructed signal

subplot(4,1,3), plot(t, real(x\_t\_reconstructed));

xlabel('time (t)');

ylabel('Reconstructed v(t)');

title('Reconstructed Signal from Fourier Transform'); % Title for reconstructed signal

% Calculate and plot the phase of X(ω)

Angle\_X\_Omega = atan2(imag(x\_omega), real(x\_omega)); % Compute phase using atan2 for correct quadrant

% Plot the phase of the Fourier transform

subplot(4,1,4), plot(omegax ./ pi, Angle\_X\_Omega);

xlabel('\omega/\pi');

ylabel('Phase \angle X(\omega) (radians)');

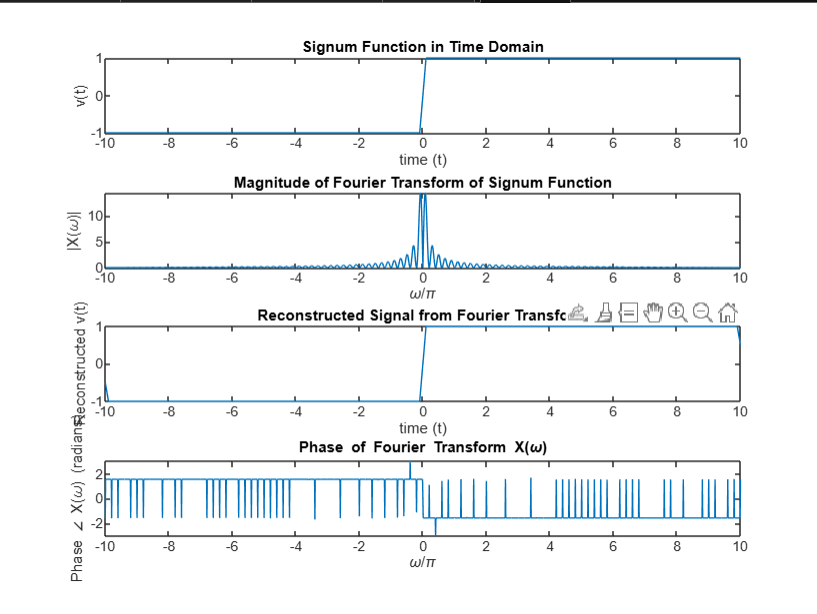
title('Phase of Fourier Transform X(\omega)'); % Title for phase plot

% Custom integration function (trapezoidal rule)

function [int\_ans] = my\_int\_func(Data, step\_size)

int\_ans = (step\_size / 2) \* (Data(1) + Data(end) + 2 \* sum(Data(2:end-1))); % Trapezoidal rule

end



FOURIER TRANSFORMS – Gate Function / Square Wave

clear all;

close all;

clc;

% Constants

step\_omega = 0.01 \* pi;

step\_size\_t = 0.1;

t = -5:step\_size\_t:5; % Time vector

length\_t = length(t);

% Omega range for frequency domain

omegax = -(1 / step\_size\_t) \* pi : step\_omega : (1 / step\_size\_t) \* pi;

length\_omega = length(omegax);

% Pre-allocate arrays

expo\_omega = zeros(length\_omega, length\_t);

x\_omega = zeros(1, length\_omega);

x\_t\_reconstructed = zeros(1, length\_t);

% Create Fourier exponentials

for ii = 1:length\_omega

expo\_omega(ii, :) = exp(-1j \* omegax(ii) \* t);

end

% Define parameters for the square wave (gate function)

period = 5; % Duration of one full period

half\_period = period / 2; % Half of the period

% Pre-allocate array for the square wave function

x\_t = zeros(1, length\_t);

% Create the square wave signal

for ii = 1:length\_t

mod\_time = mod(t(ii), period); % Determine position within the current period

if (mod\_time <= half\_period) % High part of the square wave

x\_t(ii) = 1; % Set value to 1

else % Low part of the square wave

x\_t(ii) = -1; % Set value to -1

end

end

% Plot the square wave function

subplot(4, 1, 1), plot(t, x\_t);

xlabel('time (t)');

ylabel('x(t)');

title('Square Wave Function in Time Domain');

% Compute Fourier Transform X(ω)

for ii = 1:length\_omega

temp = x\_t .\* expo\_omega(ii, :); % x(t) \* exp(-jωt)

x\_omega(ii) = my\_int\_func(temp, step\_size\_t); % Fourier Transform

end

% Plot the magnitude of the Fourier transform

subplot(4, 1, 2), plot(omegax ./ pi, abs(x\_omega));

xlabel('\omega/\pi');

ylabel('|X(\omega)|');

title('Fourier Transform of the Square Wave Function');

% Inverse Fourier Transform (reconstruct x(t))

expo\_omega\_2 = exp(1j \* omegax' \* t); % Create matrix of exp(jωt) for inverse transform

for ii = 1:length\_t

temp\_2 = x\_omega .\* expo\_omega\_2(:, ii).'; % X(ω) \* exp(jωt)

x\_t\_reconstructed(ii) = (1 / (2 \* pi)) \* my\_int\_func(temp\_2, step\_omega); % Inverse Fourier Transform

end

% Plot the reconstructed signal

subplot(4, 1, 3), plot(t, real(x\_t\_reconstructed));

xlabel('time (t)');

ylabel('Reconstructed x(t)');

title('Reconstructed Signal from Fourier Transform');

% Calculate and plot the phase of X(ω)

Angle\_X\_Omega = atan2(imag(x\_omega), real(x\_omega)); % Compute phase using atan2 for correct quadrant

% Plot the phase of the Fourier transform

subplot(4, 1, 4), plot(omegax ./ pi, Angle\_X\_Omega);

xlabel('\omega/\pi');

ylabel('Phase \angle X(\omega) (radians)');

title('Phase of Fourier Transform X(\omega)');

% Custom integration function (trapezoidal rule)

function [int\_ans] = my\_int\_func(Data, step\_size)

int\_ans = (step\_size / 2) \* (Data(1) + Data(end) + 2 \* sum(Data(2:end-1)));

end

