Lab report-8

Name: D.Manogna

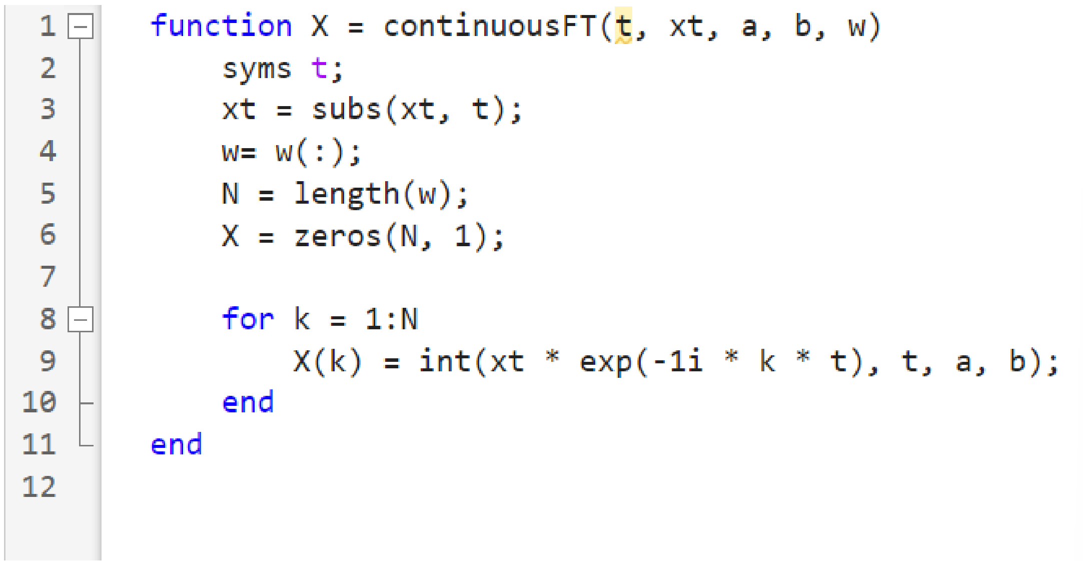
Roll No:2022102021

1)

- t – symbolic variable
- xt – signal whose FT is to be computed (function of symbolic variable t)
- a,b – the signal is equal to xt in the interval [a, b] and zero outside
- ω – the vector ω contains the values of frequency where FT is to be computed.

a)

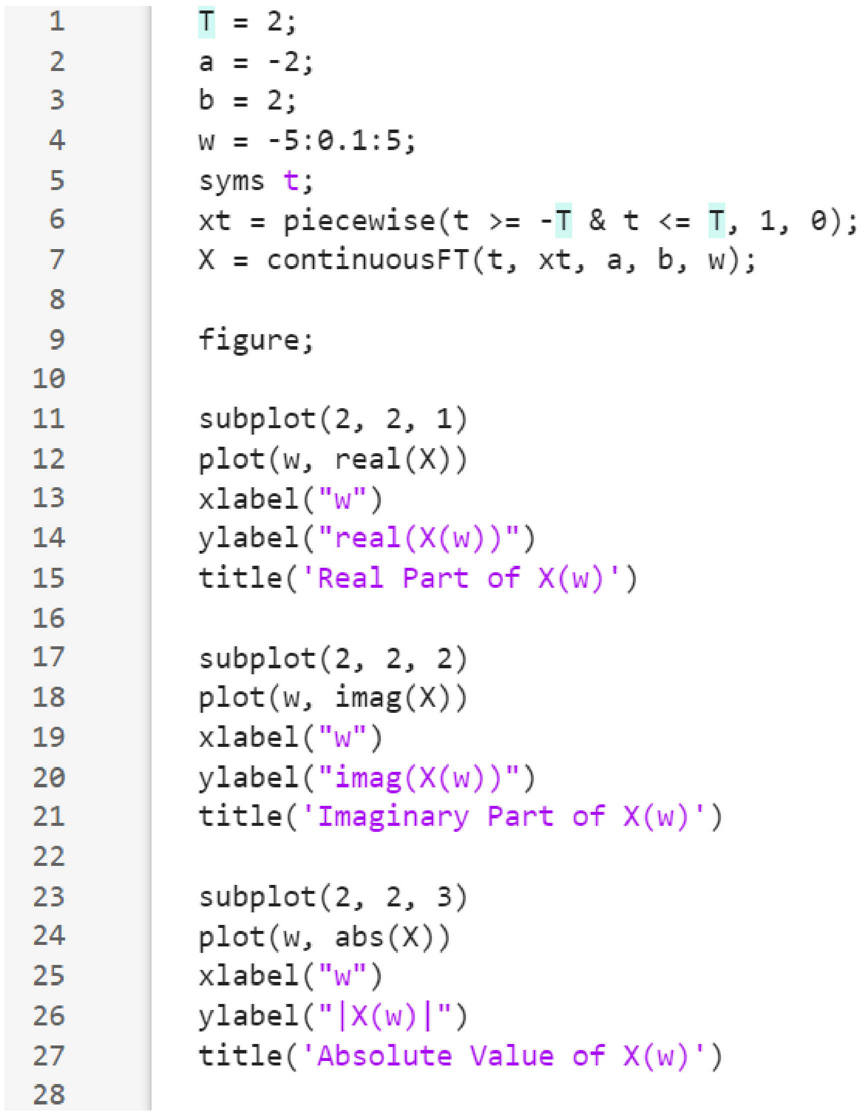
Code for the function continuousFT :



```
function X = continuousFT(t, xt, a, b, w)
    syms t;
    xt = subs(xt, t);
    w= w(:);
    N = length(w);
    X = zeros(N, 1);

    for k = 1:N
        X(k) = int(xt * exp(-1i * k * t), t, a, b);
    end
end
```

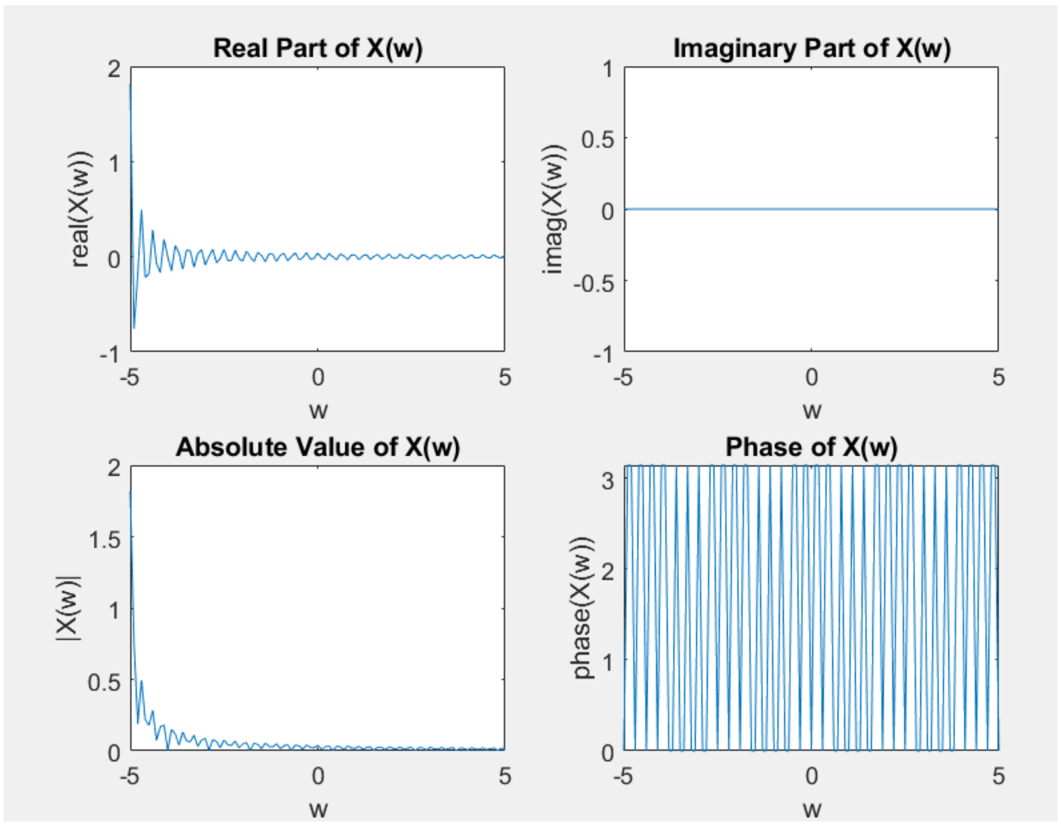
b)

code:



```
T = 2;
a = -2;
b = 2;
w = -5:0.1:5;
syms t;
xt = piecewise(t >= -T & t <= T, 1, 0);
X = continuousFT(t, xt, a, b, w);

figure;

subplot(2, 2, 1)
plot(w, real(X))
xlabel("w")
ylabel("real(X(w))")
title('Real Part of X(w)')

subplot(2, 2, 2)
plot(w, imag(X))
xlabel("w")
ylabel("imag(X(w))")
title('Imaginary Part of X(w)')

subplot(2, 2, 3)
plot(w, abs(X))
xlabel("w")
ylabel("|X(w)|")
title('Absolute Value of X(w)')

```

Plot:



Obsevations from the graph:

*The real part of the FT represents the even part of the signal's spectrum.

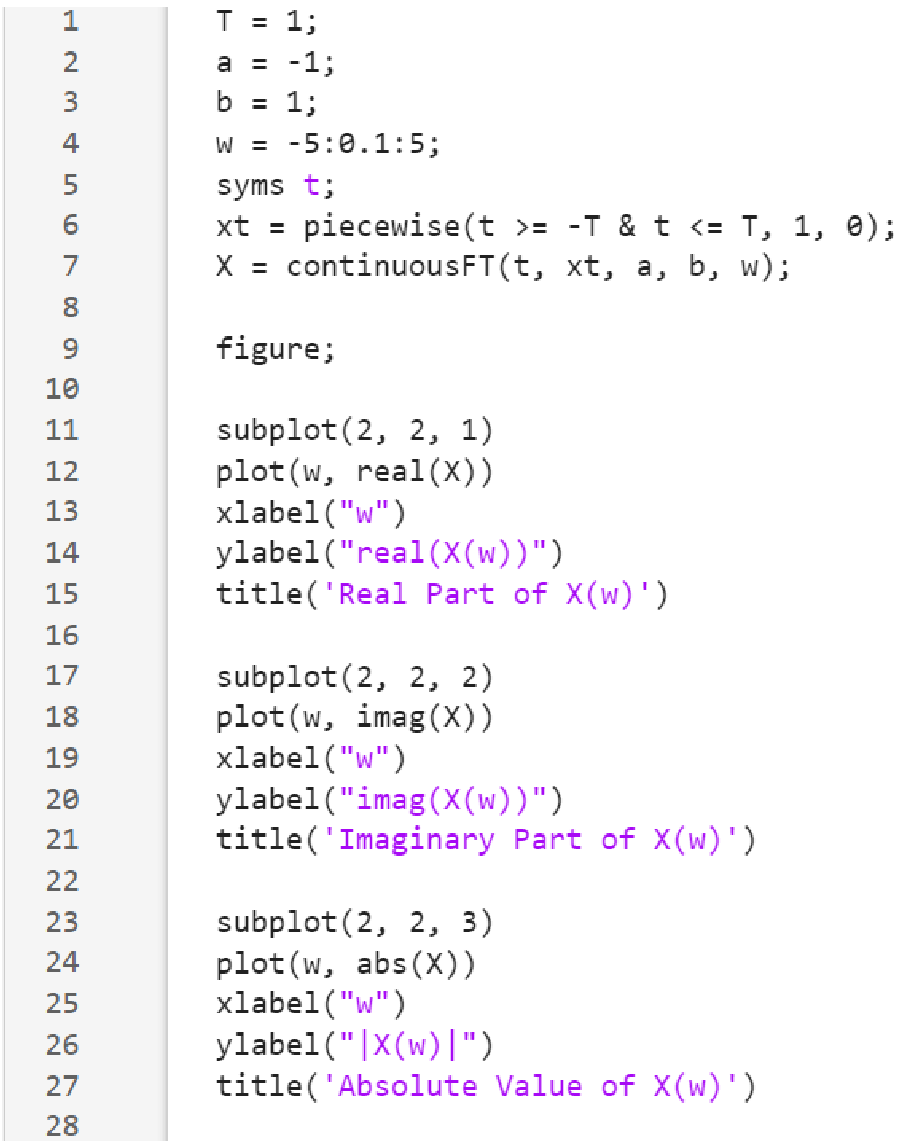
*The imaginary part of the FT represents the odd part of the signal's spectrum.

*The absolute value represents the magnitude of the FT at each frequency.

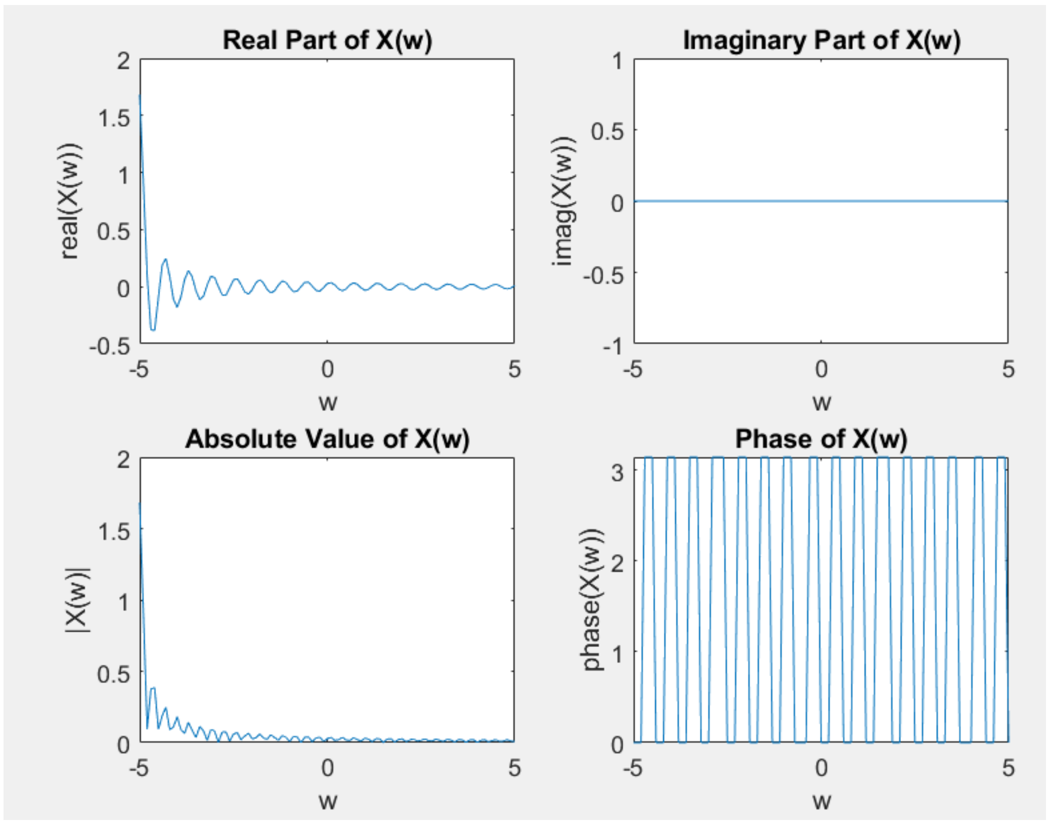
*The phase represents the phase shift of the sinusoidal components in the signal.

c)when T=1

Code:

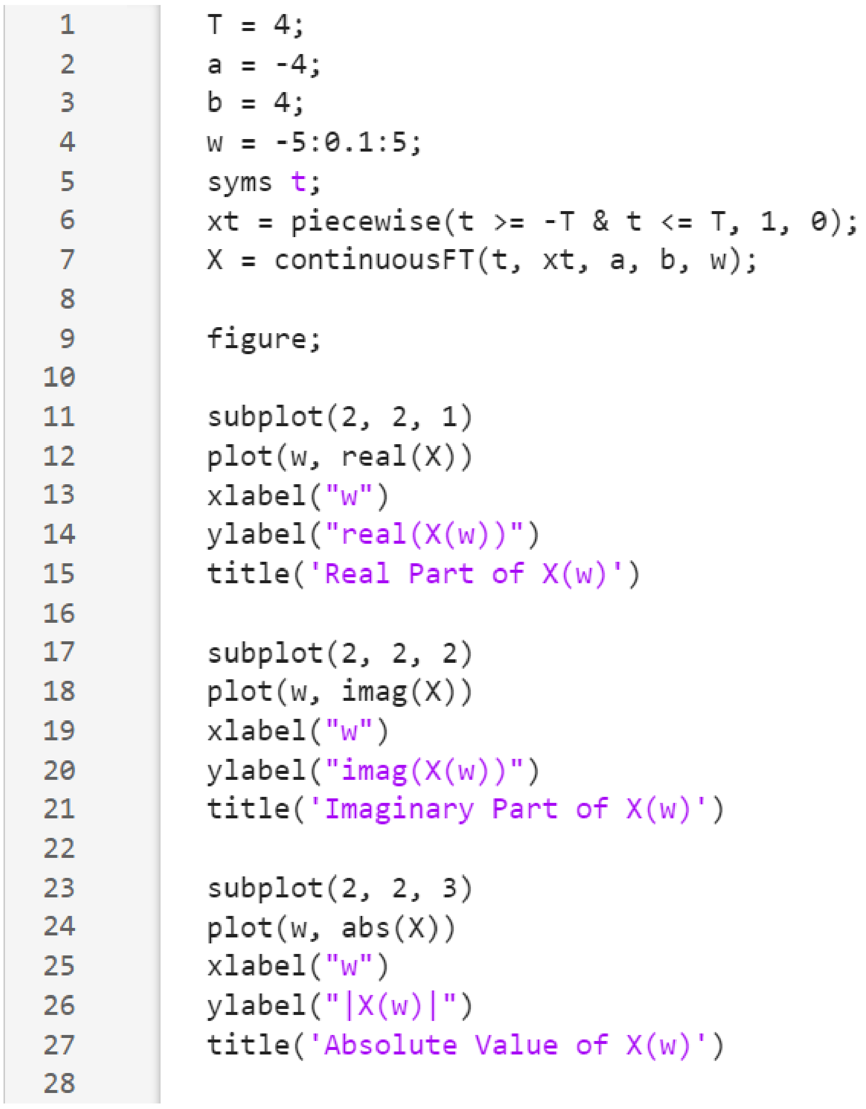


```
T = 1;
a = -1;
b = 1;
w = -5:0.1:5;
syms t;
xt = piecewise(t >= -T & t <= T, 1, 0);
X = continuousFT(t, xt, a, b, w);

figure;

subplot(2, 2, 1)
plot(w, real(X))
xlabel("w")
ylabel("real(X(w))")
title('Real Part of X(w)')

subplot(2, 2, 2)
plot(w, imag(X))
xlabel("w")
ylabel("imag(X(w))")
title('Imaginary Part of X(w)')

subplot(2, 2, 3)
plot(w, abs(X))
xlabel("w")
ylabel("|X(w)|")
title('Absolute Value of X(w)')

```


Real Part of X(w)
real(X(w))
w
Imaginary Part of X(w)
imag(X(w))
w
Absolute Value of X(w)
|X(w)|
w
Phase of X(w)
phase(X(w))
w

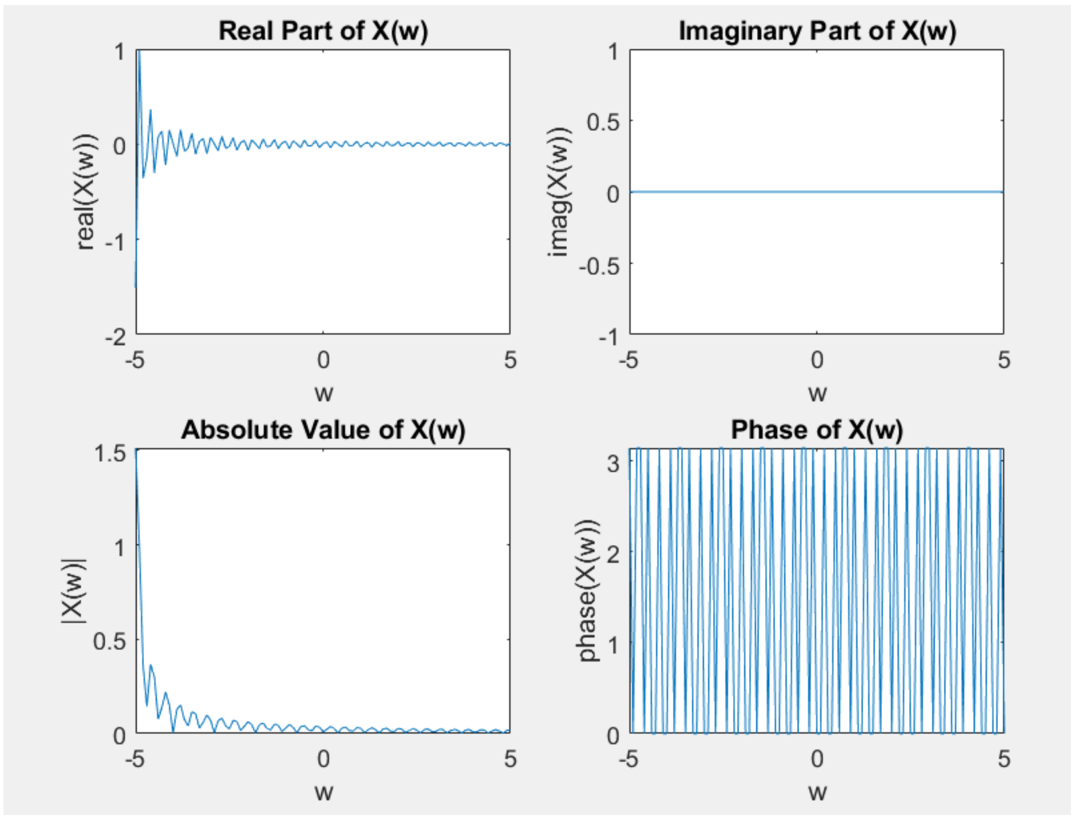
When T=4

Code:



```
T = 4;
a = -4;
b = 4;
w = -5:0.1:5;
syms t;
xt = piecewise(t >= -T & t <= T, 1, 0);
X = continuousFT(t, xt, a, b, w);

figure;

subplot(2, 2, 1)
plot(w, real(X))
xlabel("w")
ylabel("real(X(w))")
title('Real Part of X(w)')

subplot(2, 2, 2)
plot(w, imag(X))
xlabel("w")
ylabel("imag(X(w))")
title('Imaginary Part of X(w)')

subplot(2, 2, 3)
plot(w, abs(X))
xlabel("w")
ylabel("|X(w)|")
title('Absolute Value of X(w)')

```

Plot:



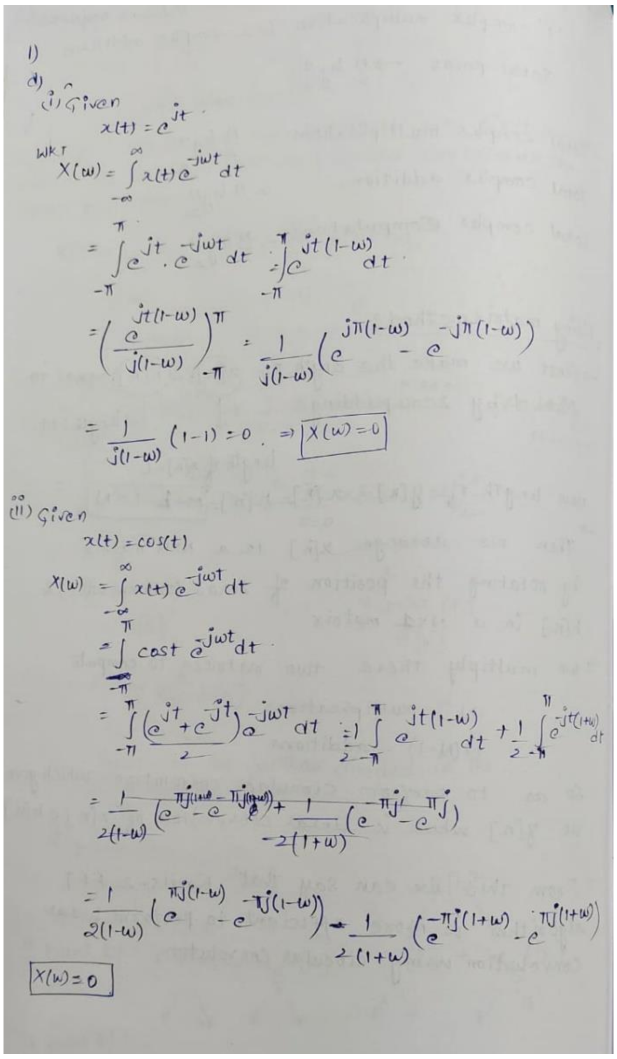
Observations:

The fourier transform property observed when the T is changed is the time-frequency trade-off .

As the T decreases , the signal's support in the time domain narrows , causing its spectrum in the frequency domain to spread out.

d)

Calculation part:



1)

d) (i) Given

$$x(t) = e^{jt}$$

WKT

$$X(\omega) = \int_{-\infty}^{\infty} x(t) e^{-j\omega t} dt$$

$$= \int_{-\pi}^{\pi} e^{jt} \cdot e^{-j\omega t} dt = \int_{-\pi}^{\pi} e^{jt(1-\omega)} dt$$

$$= \left(\frac{e^{jt(1-\omega)}}{j(1-\omega)} \right)_{-\pi}^{\pi} = \frac{1}{j(1-\omega)} \left(e^{j\pi(1-\omega)} - e^{-j\pi(1-\omega)} \right)$$

$$= \frac{1}{j(1-\omega)} (1-1) = 0 \Rightarrow \boxed{X(\omega) = 0}$$

(ii) Given

$$x(t) = \cos(t)$$

$$X(\omega) = \int_{-\infty}^{\infty} x(t) e^{-j\omega t} dt$$

$$= \int_{-\pi}^{\pi} \cos t \, e^{-j\omega t} dt$$

$$= \int_{-\pi}^{\pi} \frac{\left(e^{jt} + e^{-jt}\right)}{2} e^{-j\omega t} dt = \frac{1}{2} \int_{-\pi}^{\pi} e^{jt(1-\omega)} dt + \frac{1}{2} \int_{-\pi}^{\pi} e^{-jt(1+\omega)} dt$$

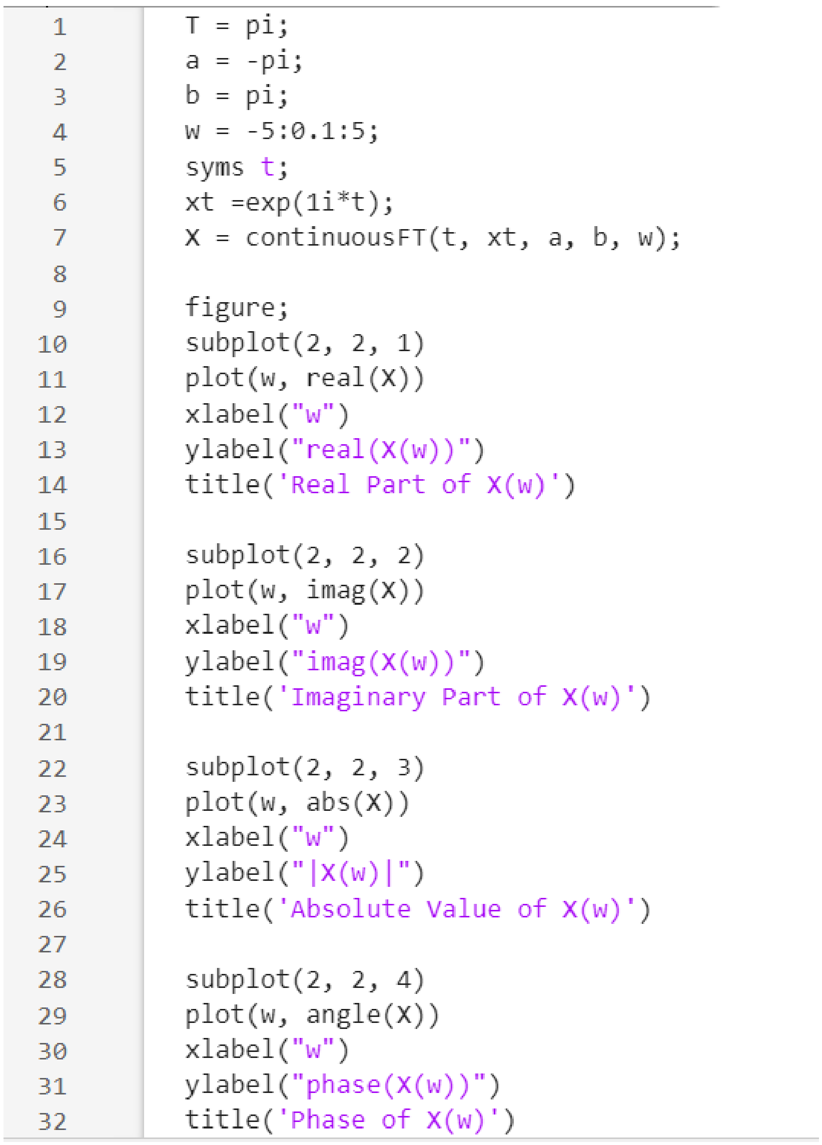
~~$= \frac{1}{2(1-\omega)} \left(e^{\pi j(1-\omega)} - e^{-\pi j(1-\omega)} \right) + \frac{1}{-2(1+\omega)} \left(e^{-\pi j} - e^{\pi j} \right)$~~

$$= \frac{1}{2(1-\omega)} \left(e^{\pi j(1-\omega)} - e^{-\pi j(1-\omega)} \right) - \frac{1}{2(1+\omega)} \left(e^{-\pi j(1+\omega)} - e^{\pi j(1+\omega)} \right)$$

$$\boxed{X(\omega) = 0}$$

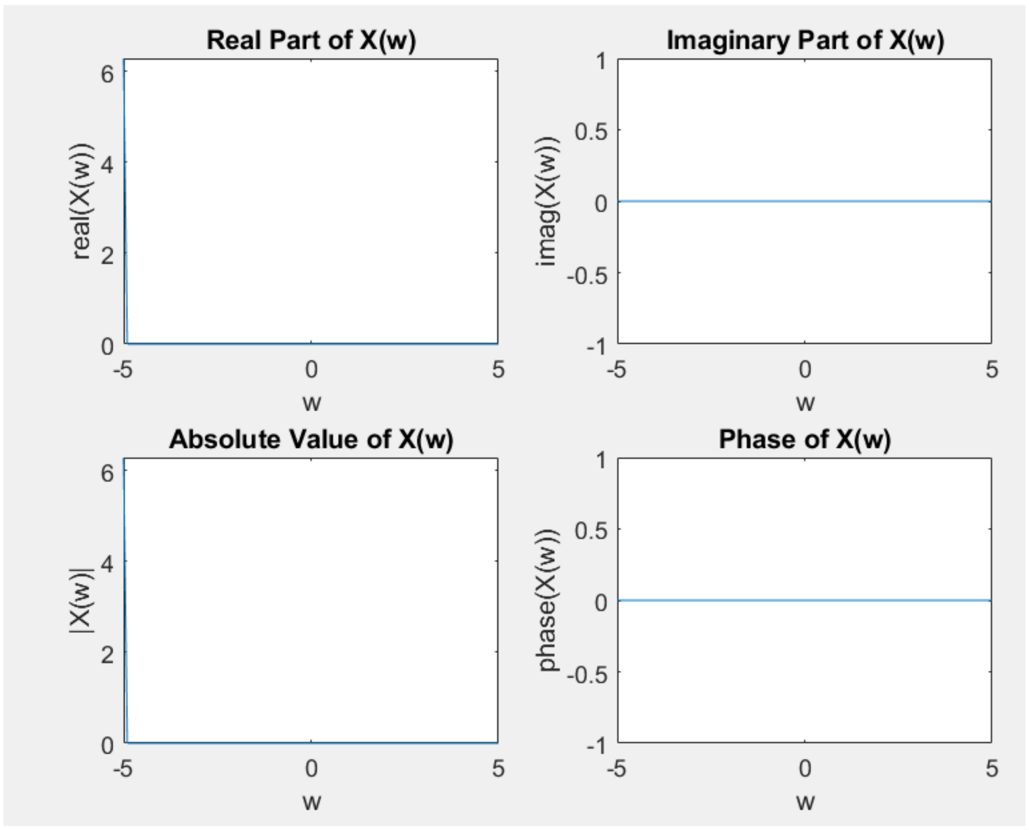
when x(t)=e^(jt)

Code:



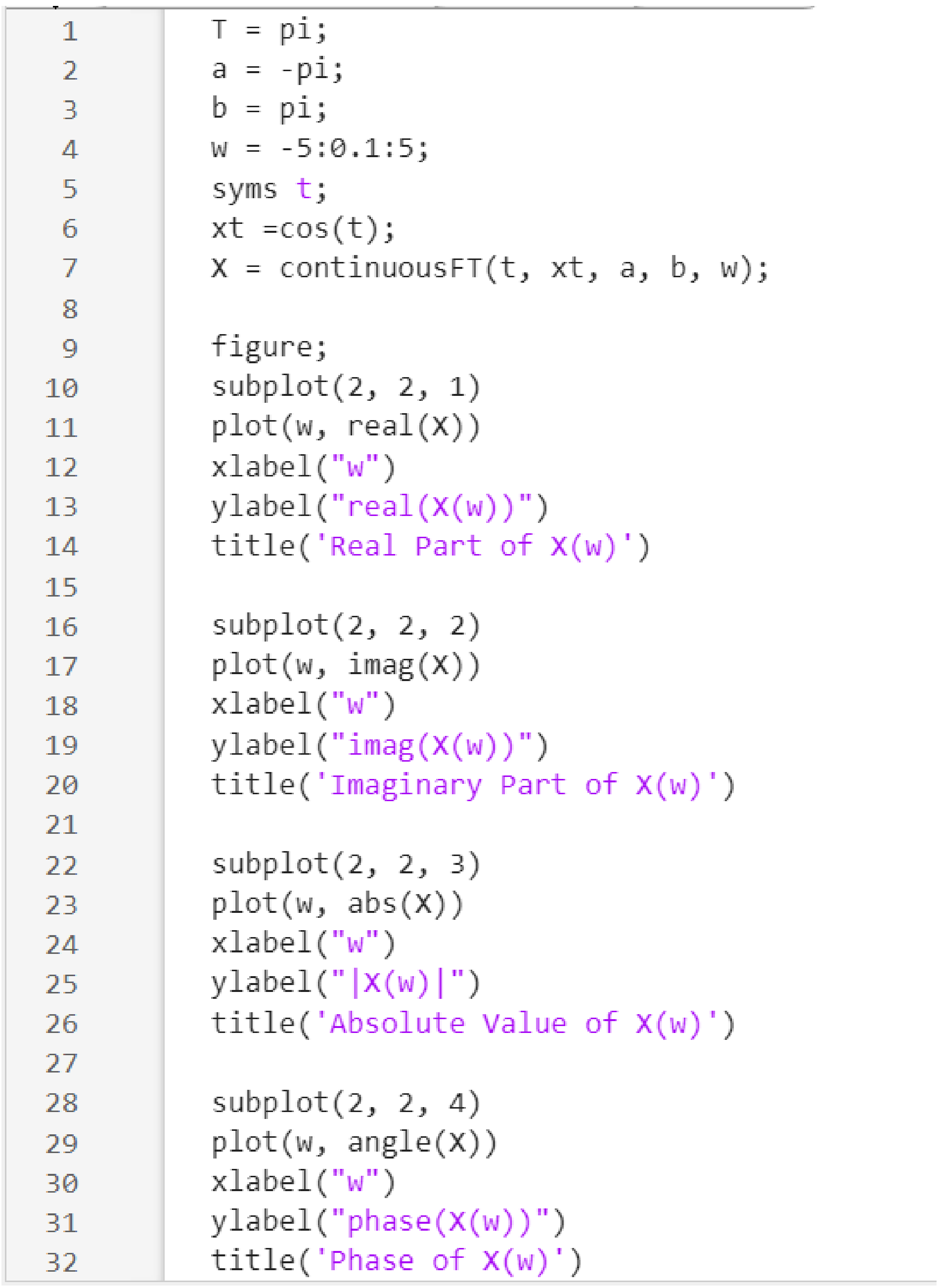
```matlab
T = pi;
a = -pi;
b = pi;
w = -5:0.1:5;
syms t;
xt =exp(1i*t);
X = continuousFT(t, xt, a, b, w);

figure;
subplot(2, 2, 1)
plot(w, real(X))
xlabel("w")
ylabel("real(X(w))")
title('Real Part of X(w)')

subplot(2, 2, 2)
plot(w, imag(X))
xlabel("w")
ylabel("imag(X(w))")
title('Imaginary Part of X(w)')

subplot(2, 2, 3)
plot(w, abs(X))
xlabel("w")
ylabel("|X(w)|")
title('Absolute Value of X(w)')

subplot(2, 2, 4)
plot(w, angle(X))
xlabel("w")
ylabel("phase(X(w))")
title('Phase of X(w)')
```

Plot:



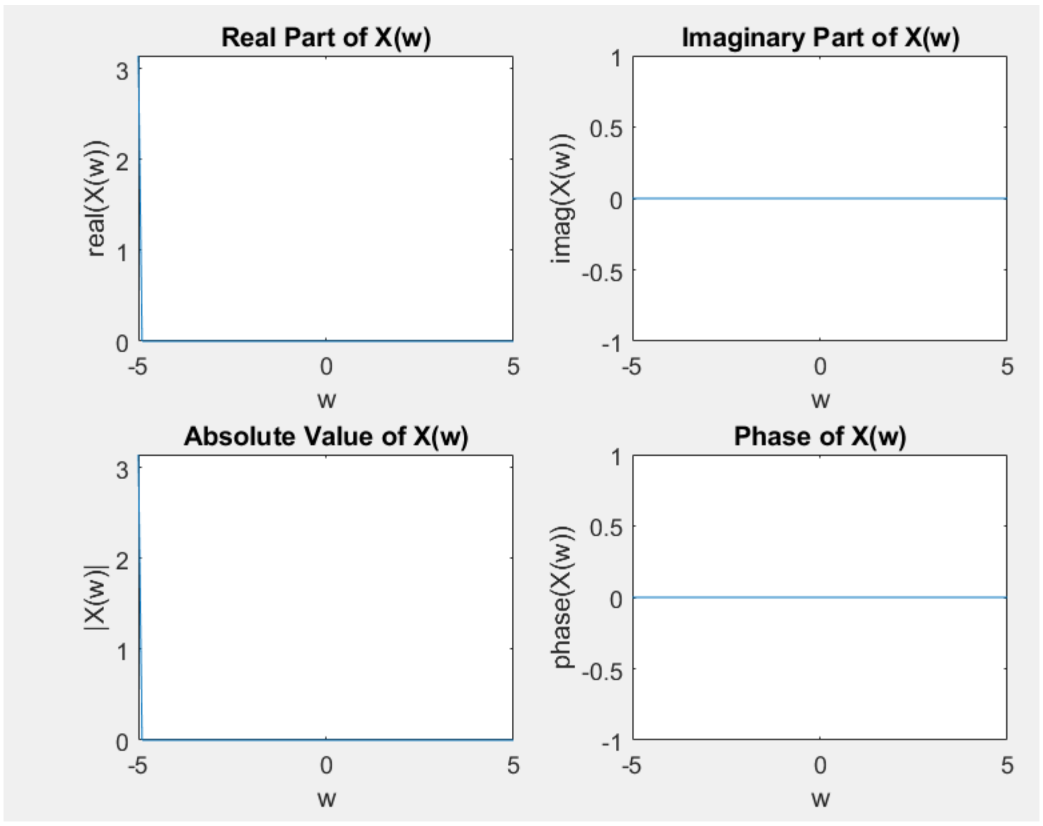
When x(t)=cos(t)

Code:



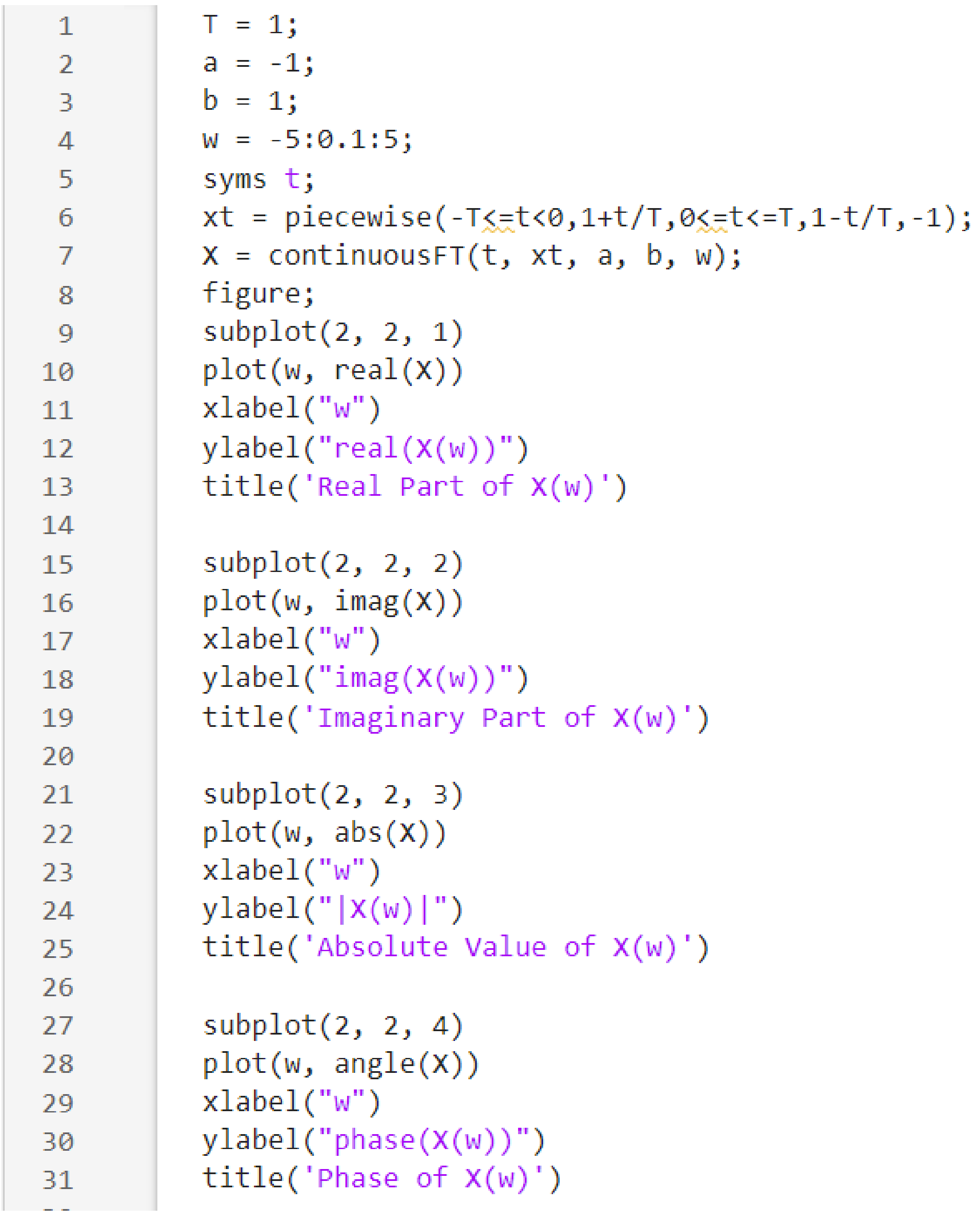
```matlab
T = pi;
a = -pi;
b = pi;
w = -5:0.1:5;
syms t;
xt =cos(t);
X = continuousFT(t, xt, a, b, w);

figure;
subplot(2, 2, 1)
plot(w, real(X))
xlabel("w")
ylabel("real(X(w))")
title('Real Part of X(w)')

subplot(2, 2, 2)
plot(w, imag(X))
xlabel("w")
ylabel("imag(X(w))")
title('Imaginary Part of X(w)')

subplot(2, 2, 3)
plot(w, abs(X))
xlabel("w")
ylabel("|X(w)|")
title('Absolute Value of X(w)')

subplot(2, 2, 4)
plot(w, angle(X))
xlabel("w")
ylabel("phase(X(w))")
title('Phase of X(w)')
```

Plot:


Real Part of X(w)
real(X(w))
w
Imaginary Part of X(w)
imag(X(w))
w
Absolute Value of X(w)
|X(w)|
w
Phase of X(w)
phase(X(w))
w

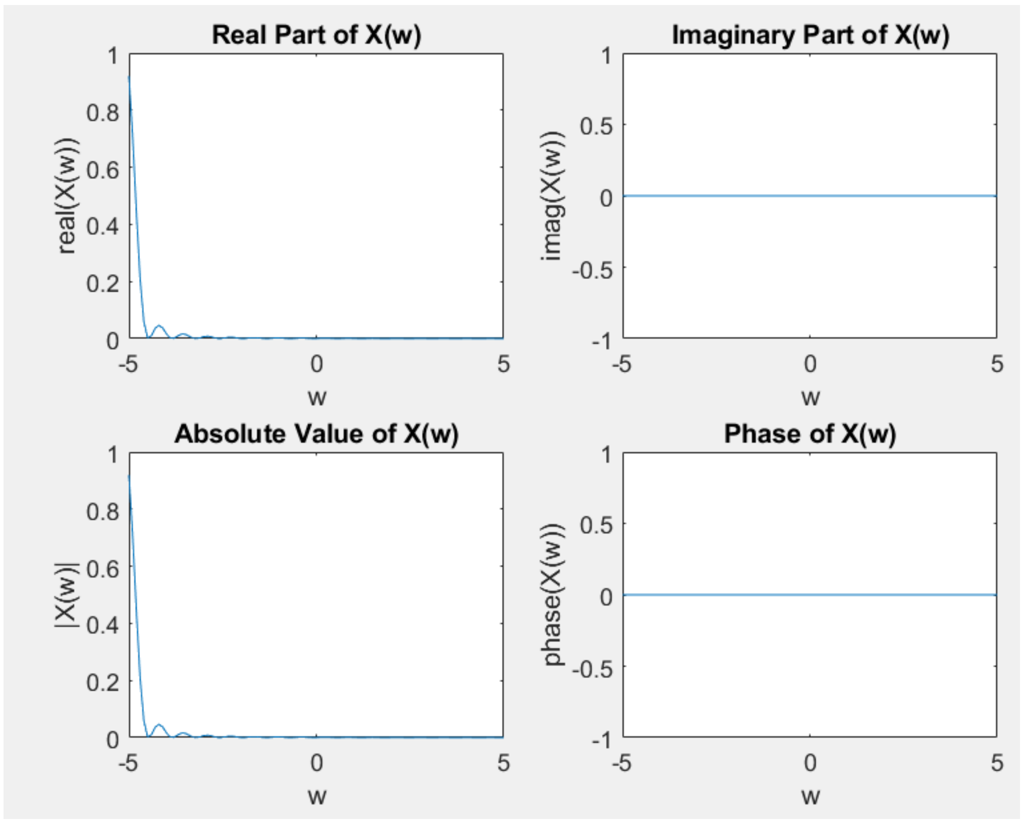
e)

Code:



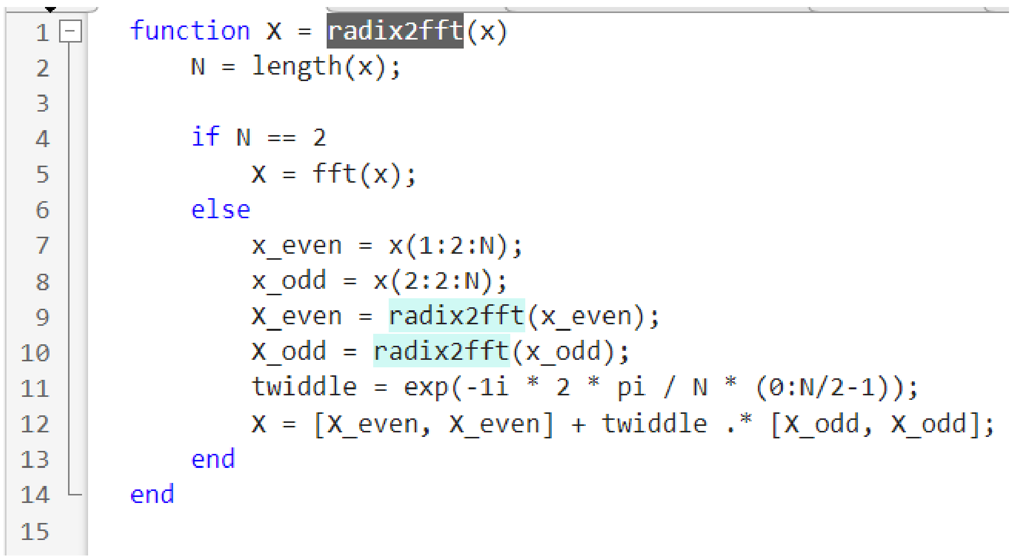
```
T = 1;
a = -1;
b = 1;
w = -5:0.1:5;
syms t;
xt = piecewise(-T<=t<0,1+t/T,0<=t<=T,1-t/T,-1);
X = continuousFT(t, xt, a, b, w);
figure;
subplot(2, 2, 1)
plot(w, real(X))
xlabel("w")
ylabel("real(X(w))")
title('Real Part of X(w)')

subplot(2, 2, 2)
plot(w, imag(X))
xlabel("w")
ylabel("imag(X(w))")
title('Imaginary Part of X(w)')

subplot(2, 2, 3)
plot(w, abs(X))
xlabel("w")
ylabel("|X(w)|")
title('Absolute Value of X(w)')

subplot(2, 2, 4)
plot(w, angle(X))
xlabel("w")
ylabel("phase(X(w))")
title('Phase of X(w)')
```

Plot:



2)

Code for the Function `radix2fft`



```
function X = radix2fft(x)
    N = length(x);

    if N == 2
        X = fft(x);
    else
        x_even = x(1:2:N);
        x_odd = x(2:2:N);
        X_even = radix2fft(x_even);
        X_odd = radix2fft(x_odd);
        twiddle = exp(-1i * 2 * pi / N * (0:N/2-1));
        X = [X_even, X_even] + twiddle .* [X_odd, X_odd];
    end
end
```

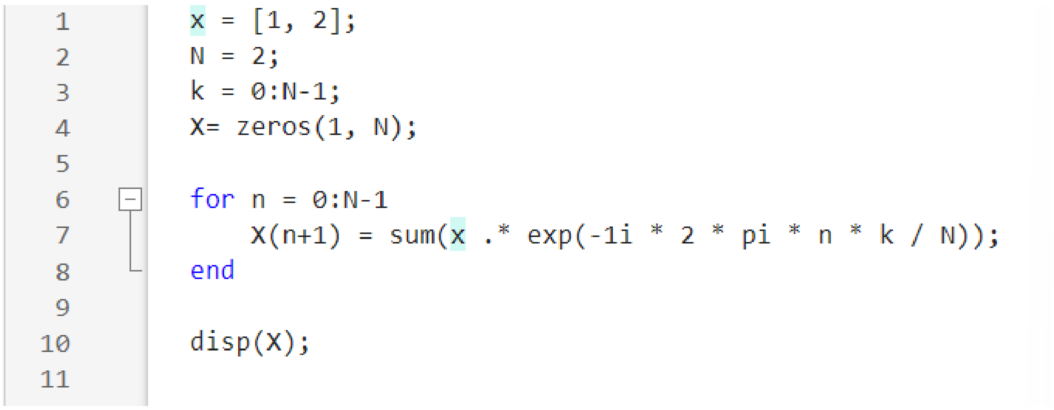
Consider

X[0]=1 and

X[1]=2

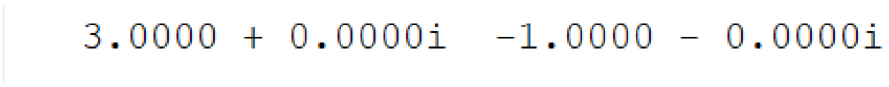
When N=2

Code:



```
x = [1, 2];
N = 2;
k = 0:N-1;
X= zeros(1, N);

for n = 0:N-1
    X(n+1) = sum(x .* exp(-1i * 2 * pi * n * k / N));
end

disp(X);

```

The observed value is:

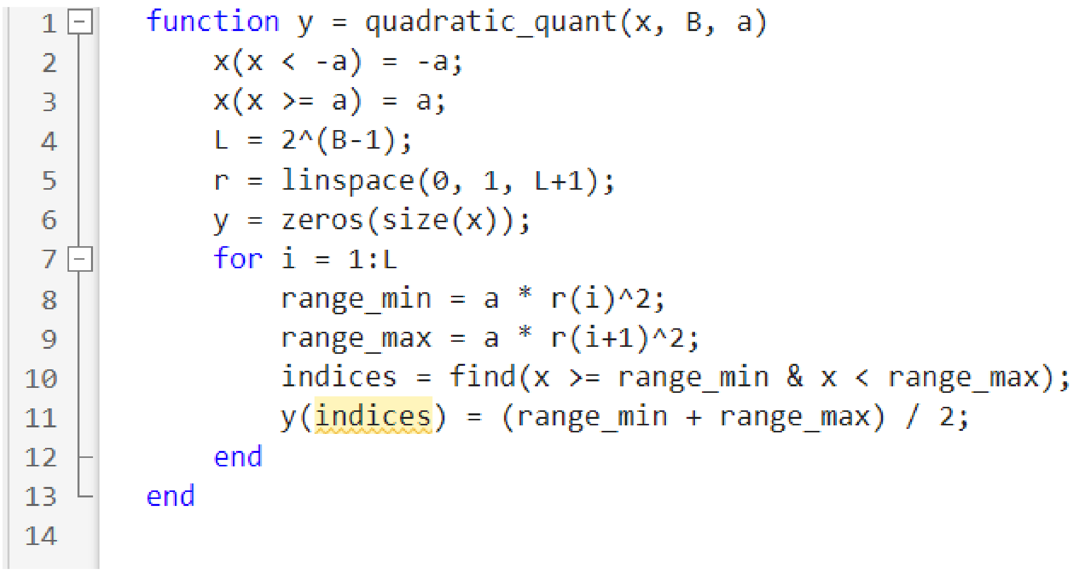


```
   3.0000 + 0.0000i  -1.0000 - 0.0000i
```

The calculated values:

3)

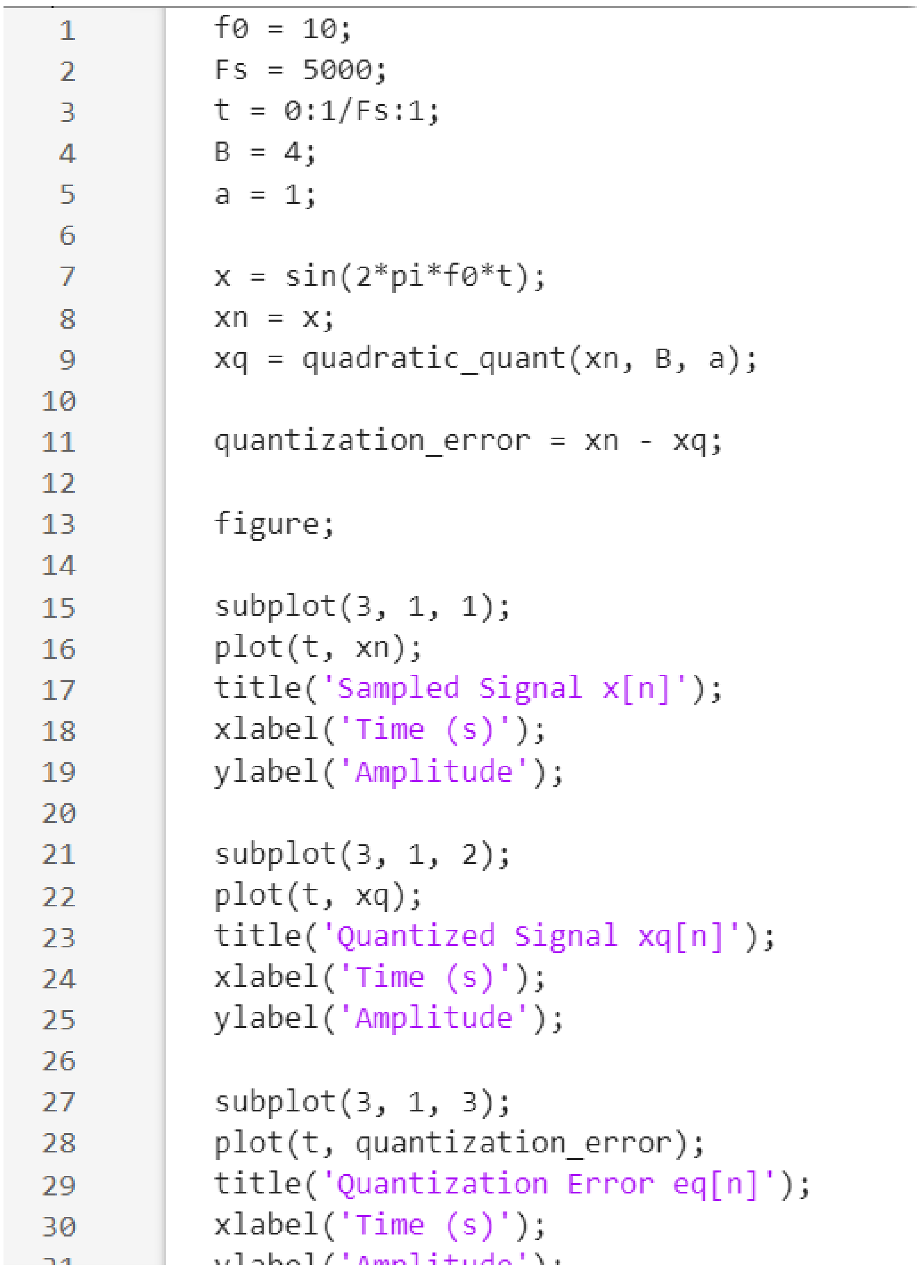
Code for the function quadratic_quant



```
function y = quadratic_quant(x, B, a)
    x(x < -a) = -a;
    x(x >= a) = a;
    L = 2^(B-1);
    r = linspace(0, 1, L+1);
    y = zeros(size(x));
    for i = 1:L
        range_min = a * r(i)^2;
        range_max = a * r(i+1)^2;
        indices = find(x >= range_min & x < range_max);
        y(indices) = (range_min + range_max) / 2;
    end
end

```

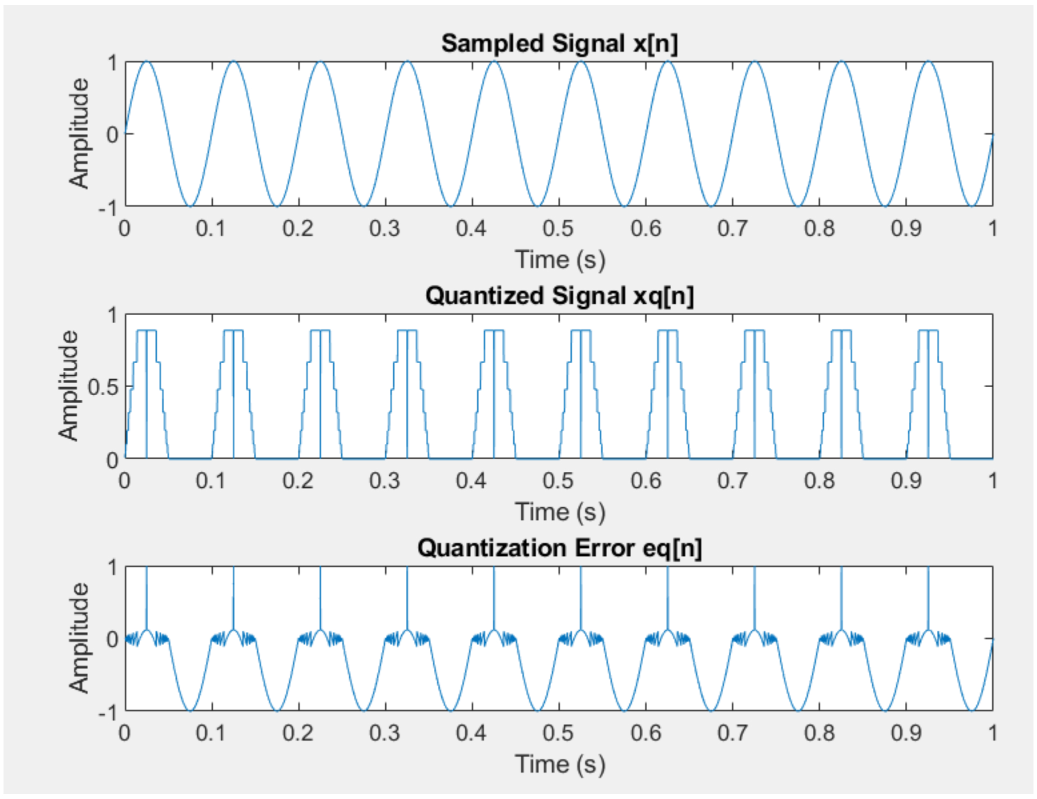
a)

Code:



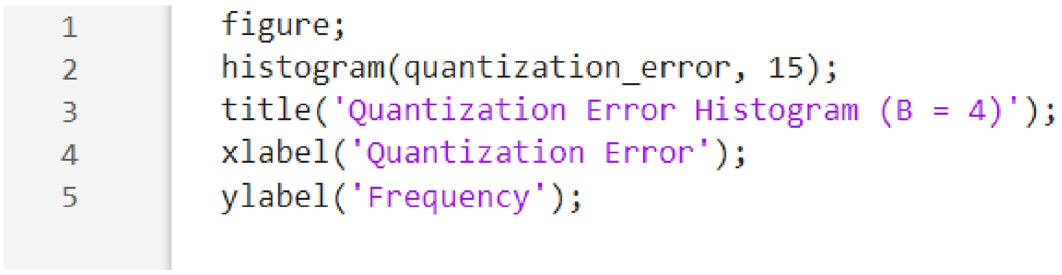
```matlab
f0 = 10;
Fs = 5000;
t = 0:1/Fs:1;
B = 4;
a = 1;

x = sin(2*pi*f0*t);
xn = x;
xq = quadratic_quant(xn, B, a);

quantization_error = xn - xq;

figure;

subplot(3, 1, 1);
plot(t, xn);
title('Sampled Signal x[n]');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(3, 1, 2);
plot(t, xq);
title('Quantized Signal xq[n]');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(3, 1, 3);
plot(t, quantization_error);
title('Quantization Error eq[n]');
xlabel('Time (s)');
ylabel('Amplitude');
```

Plot:



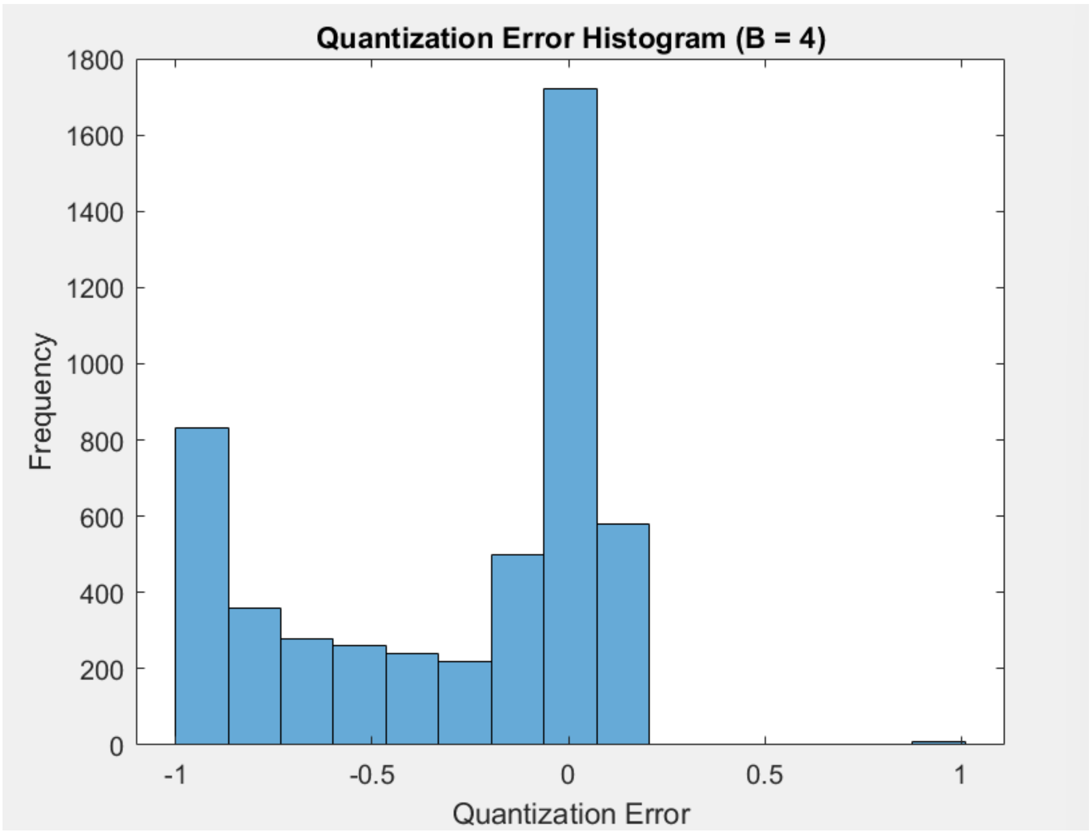
d)

Code:



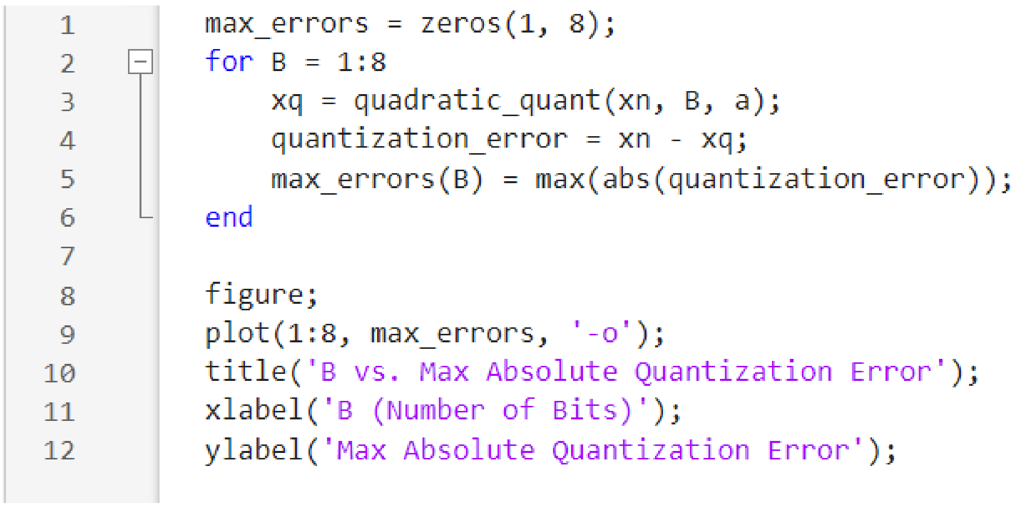
```
figure;
histogram(quantization_error, 15);
title('Quantization Error Histogram (B = 4)');
xlabel('Quantization Error');
ylabel('Frequency');
```

Plot:



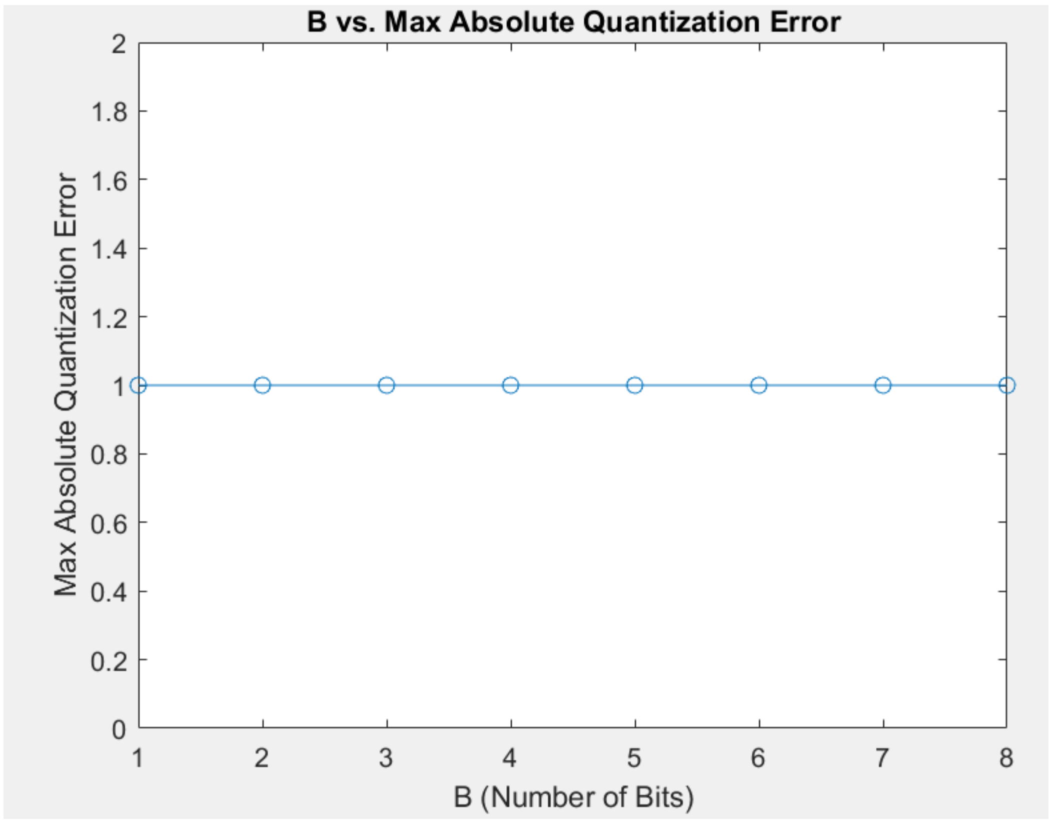
e)

Code:



```
max_errors = zeros(1, 8);
for B = 1:8
    xq = quadratic_quant(xn, B, a);
    quantization_error = xn - xq;
    max_errors(B) = max(abs(quantization_error));
end

figure;
plot(1:8, max_errors, '-o');
title('B vs. Max Absolute Quantization Error');
xlabel('B (Number of Bits)');
ylabel('Max Absolute Quantization Error');
```

Plot:



Process:

Here we are comparing the maximum absolute quantization error for the different values of ‘B’ (the number of bits used for the quantization)

*After quantization we calculate the quantization error for each value of “B” .

*The quantization error is the difference between the original sampled signal and the quantized signal.

It represents the discrepancy between the continuous signal and the discrete approximation.

f)

SQNR is the ratio of the signal power to quantization noise power:

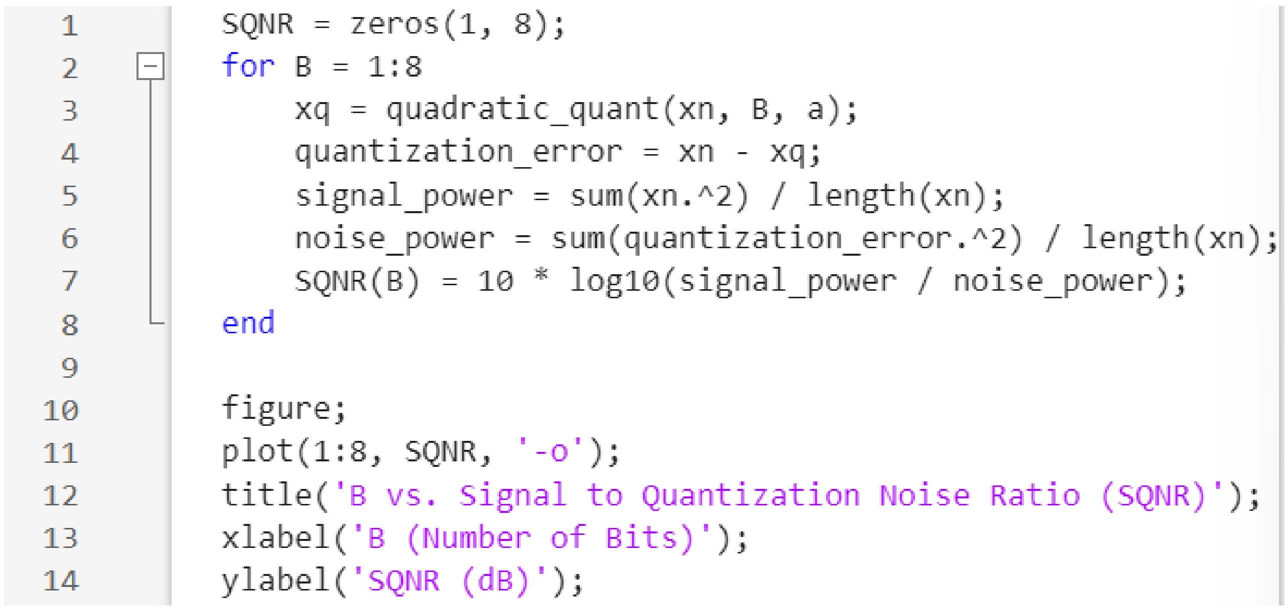
$$SQNR = \frac{\sum_n |x[n]|^2}{\sum_n |e_q[n]|^2}$$

SQNR is the measure of the quality of quantization.

Higher SQNR indicates better quantization performance , as it means that the signal’s power is much larger than the noise introduced by quantization.

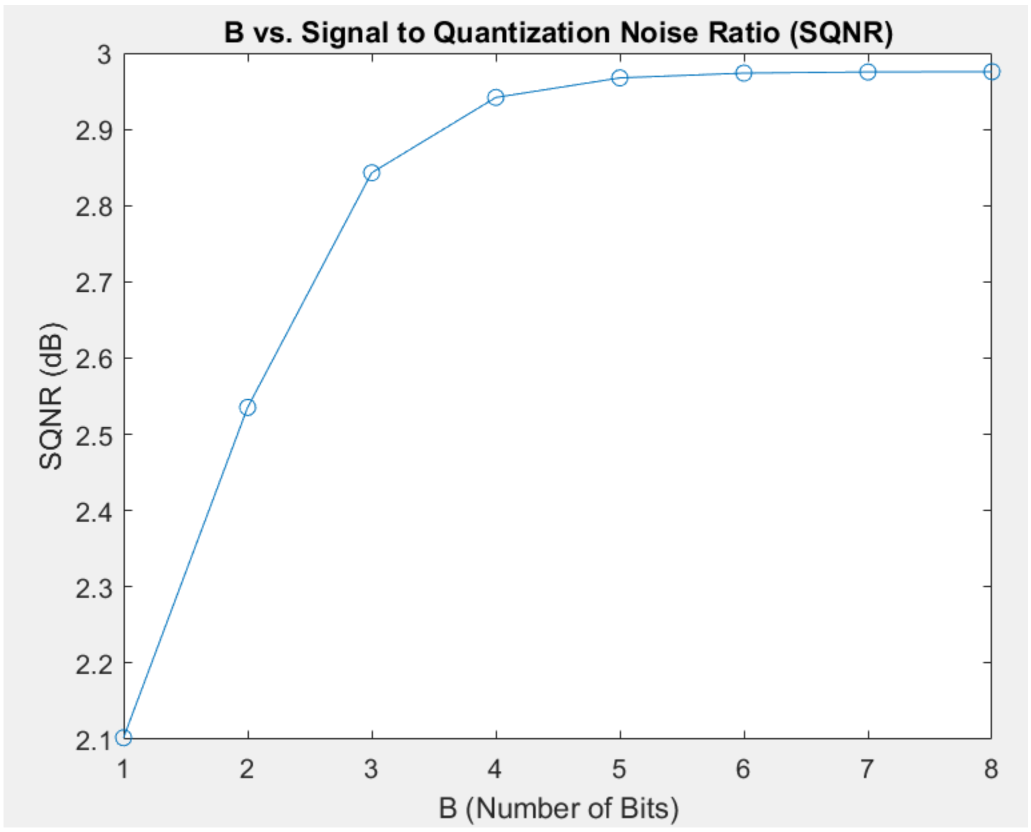
- When we compute the SQNR in decibles (dB) by taking the 10 times the base 10 logarithm of the signal power divided by the quantization noise power.

Code:



```matlab
SQNR = zeros(1, 8);
for B = 1:8
    xq = quadratic_quant(xn, B, a);
    quantization_error = xn - xq;
    signal_power = sum(xn.^2) / length(xn);
    noise_power = sum(quantization_error.^2) / length(xn);
    SQNR(B) = 10 * log10(signal_power / noise_power);
end

figure;
plot(1:8, SQNR, '-o');
title('B vs. Signal to Quantization Noise Ratio (SQNR)');
xlabel('B (Number of Bits)');
ylabel('SQNR (dB)');
```

Plot:



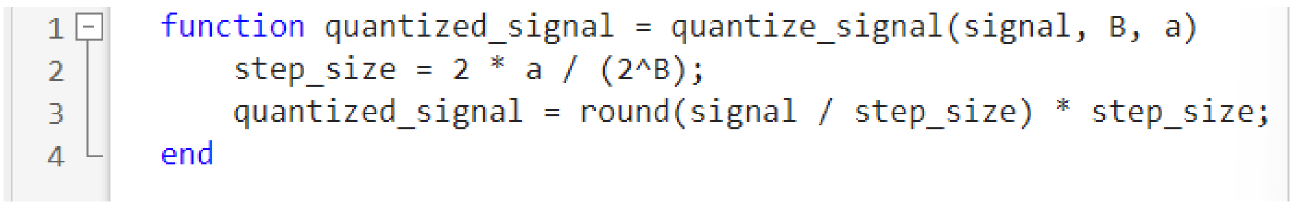
g)Here we consider the accuracy of the non-uniform quantizer in various regions of the interval [-a,a) and compare it to a hypothetical uniform quantizer that instead considers intervals of the form [ari,ar(i+1))

* In regions close to zero (where the signal amplitude is high) the non uniform quantizer provides better accuracy . This is because it allocates more quantization levels to these regions resulting in smaller quantization errors.

- In regions far from zero (where the signal amplitude is low), the uniform quantizer might be more appropriate because it provides consistent quantization accuracy.
- The choice between the non uniform and uniform quantization depends on the specific characteristics of the signal and the application's requirements . If the signal has variations in amplitude , a non uniform quantizer can be more efficient in preserving the signal's details.
- If the signal has a relatively uniform amplitude distribution across the entire range, a uniform quantizer might be a better choice because it simplifies the quantization process.

4)

Code for the function `quantized_signal`



```
function quantized_signal = quantize_signal(signal, B, a)
    step_size = 2 * a / (2^B);
    quantized_signal = round(signal / step_size) * step_size;
end
```

As the B increases the sound quality improves , and the quantized signals better preserve the original frequency content. Lower B values introduce more audible distortion and additional frequency components due to quantization noise.

It’s a trade-off between bit depth and sound quality , with higher bit depths providing better audio fidelity at the cost of increased data rate and storage requirements.