SIDE BAND POWER IN AMPLITUDE MODULATION

```
clc;
clear;

// Given values
Rl  = 100;      // Load resistance (ohm)
Fc  = 1;        // Carrier frequency (MHz)
Fm1 = 2;        // Modulating frequency 1 (kHz)
Fm2 = 3;        // Modulating frequency 2 (kHz)
Fm3 = 5;        // Modulating frequency 3 (kHz)

Ec  = 100;      // Carrier voltage (V)
Em1 = 10;       // Modulating voltage 1 (V)
Em2 = 20;       // Modulating voltage 2 (V)
Em3 = 30;       // Modulating voltage 3 (V)

// Modulation indices
m1 = Em1 / Ec;
disp("m1 = ");
disp(m1);

m2 = Em2 / Ec;
disp("m2 = ");
disp(m2);

m3 = Em3 / Ec;
disp("m3 = ");
disp(m3);

// Overall modulation index
m = sqrt(m1^2 + m2^2 + m3^2);
disp("Overall modulation index m = ");
disp(m);

// Sideband power calculation
disp("Power in upper and lower sidebands is equal");

Psb = (Ec^2 * m^2) / (8 * Rl);   // Power in one sideband (W)

disp("Sideband power (W):");
disp(Psb);
```

SIDE BAND FREQUENCIES IN AMPLITUDE MODULATION

```
clc;
clear;

// Given values
V  = 12;      // DC voltage (V)
Vm = 2;       // Peak voltage (V)
Fm = 4;       // Modulating frequency (kHz)
```

```
Vb = 4;       // DC voltage (V)

R1 = 100 * 10^3;       // Resistance in ohms (100 kΩ)
C1 = 0.001 * 10^-6;    // Capacitance in farads (0.001 μF)

// (a) Carrier frequency calculation
disp("(a) The carrier frequency is determined from fc = 1/(R1*C1)");

fc = 1 / (R1 * C1);    // Carrier frequency in Hz
Fc = fc / 10^3;        // Convert to kHz

disp("Carrier frequency fc (kHz):");
disp(Fc);

// (b) Upper and lower side frequencies
disp("(b) Upper and lower side frequencies:");

fu = Fc + Fm;          // Upper side frequency (kHz)
fl = Fc - Fm;          // Lower side frequency (kHz)

disp("Upper side frequency Fusf (kHz):");
disp(fu);

disp("Lower side frequency Flsf (kHz):");
disp(fl);
```

AM RECEIVER- OSCILLATOR FREQUENCY, IMAGE FREQUENCY AND IMAGE REJECTION RATIO

```
clc;
clear;

// Given values
IF  = 455 * 10^3;     // Intermediate frequency in Hz
f_s = 900 * 10^3;     // Signal frequency in Hz
Q   = 80;             // Quality factor

// Calculations
f_0  = f_s + IF;          // Local oscillator frequency (Hz)
f_si = f_s + 2 * IF;      // Image frequency (Hz)

p = (f_si / f_s) - (f_s / f_si);
a = sqrt(1 + (Q * p)^2);  // Image Frequency Rejection Ratio (IFRR)

// Results
printf("\n(i) Local oscillator frequency = %.2f Hz", f_0);
printf("\n(ii) Image frequency = %.2f Hz", f_si);
printf("\n(iii) Image frequency rejection ratio = %.2f\n", a);
```

FREQUENCY DEVIATION & MODULATION INDEX IN FM

```
clc;
clear;

// Given values (First case)
f_m1    = 500;         // Modulating frequency (Hz)
delta_f1 = 6.4 * 10^3;  // Frequency deviation (Hz)
V_m1    = 3.2;         // Modulating amplitude (V)

// Second case
V_m2 = 8.4;            // Modulating amplitude (V)

// Third case
V_m3 = 20;             // Modulating amplitude (V)
f_m3 = 200;            // Modulating frequency (Hz)

// Calculations
k_f = delta_f1 / V_m1;      // Frequency sensitivity (Hz/V)

delta_f2 = k_f * V_m2;      // Frequency deviation (second case)
delta_f3 = k_f * V_m3;      // Frequency deviation (third case)

m_1 = delta_f1 / f_m1;      // Modulation index (first case)
m_2 = delta_f2 / f_m1;      // Modulation index (second case)
m_3 = delta_f3 / f_m3;      // Modulation index (third case)

// Results
printf("\n(i-a) Frequency deviation for first case = %.2f Hz", delta_f1);
printf("\n(i-b) Modulation index for first case = %.2f", m_1);

printf("\n(ii-a) Frequency deviation for second case = %.2f Hz", delta_f2);
printf("\n(ii-b) Modulation index for second case = %.2f", m_2);

printf("\n(iii-a) Frequency deviation for third case = %.2f Hz", delta_f3);
printf("\n(iii-b) Modulation index for third case = %.2f\n", m_3);
```

GENERATION OF FM AND PM EQUATION

```
clc;
clear;

// Given values
fc  = 25e6;     // Carrier frequency (Hz)
fm  = 400;      // Modulating frequency (Hz)
del = 1e4;      // Frequency deviation (Hz)

// Angular frequencies
wc = 2 * %pi * fc;
wm = 2 * %pi * fm;
```

```
// Modulation index
mf = del / fm;

// Display equations

// Case (a): FM
printf("\n(a) Equation of FM wave:");
printf("\nV = 4 sin( %.2e t + %.2f sin( %.2e t ) )\n", wc, mf, wm);

// Case (b): PM
printf("\n(b) Equation of PM wave:");
printf("\nV = 4 sin( %.2e t + %.2f sin( %.2e t ) )\n", wc, mf, wm);

// Case (c): FM with higher modulating frequency
printf("\n(c) Equation of FM wave:");
printf("\nV = 4 sin( %.2e t + 5 sin( %.2e t ) )\n", wc, 5*wm);

// Case (d): PM with higher modulating frequency
printf("\n(d) Equation of PM wave:");
printf("\nV = 4 sin( %.2e t + %.2f sin( %.2e t ) )\n", wc, mf, 5*wm);
```

EQUALIZER TO COMPENSATE APERTURE EFFECT

```
clc;
clear;
close;

// Duty cycle range
T_Ts = 0.01 : 0.01 : 0.6;

// Pre-allocate array
E = zeros(1, length(T_Ts));

// Equalizer calculation (1 / sinc(0.5*T/Ts))
E(1) = 1;   // Defined explicitly to avoid division issue

for i = 2:length(T_Ts)
   E(i) = ((%pi/2) * T_Ts(i)) / sin((%pi/2) * T_Ts(i));
end

// Plot
a = gca();
a.data_bounds = [0, 0.8; 0.8, 1.2];

plot2d(T_Ts, E, 5);

xlabel("Duty cycle T/Ts");
ylabel("1 / sinc(0.5(T/Ts))");
title("Normalized Equalization to Compensate for Aperture Effect");
```

CONSTELLATION DIAGRAM OF BINARY PHASE SHIFT KEYING

```
// Caption : Signal Space diagram for coherent BPSK
// Figure : Signal Space Diagram for coherent BPSK system

clc;
clear;
close;

// Number of symbols
M = 2;
i = 1:M;

// BPSK signal points
y = cos(2 * %pi + (i - 1) * %pi);

// Binary annotation
annot = dec2bin([length(y)-1:-1:0], log2(M));

// Display coordinates and message points
disp("Coordinates of message points:");
disp(y);

disp("Message points (binary):");
disp(annot);

// Plot constellation
figure;
a = gca();
a.data_bounds = [-2, -2; 2, 2];
a.x_location = "origin";
a.y_location = "origin";

// Plot BPSK points
plot2d(real(y(1)), imag(y(1)), -9);
plot2d(real(y(2)), imag(y(2)), -5);

// Labels and title
xlabel("In-Phase");
ylabel("Quadrature");
title("Constellation for BPSK");

// Legend
legend(["Message point 1 (binary 1)"; "Message point 2 (binary 0)"], 5);
```

DIFFERENTIAL PHASE SHIFT KEYING

```
clc;
clear;
close;
```

```
// Input binary sequence
bk = [1 0 0 1 0 0 1 1];
N = length(bk);

// Differential encoding (DPSK)
dk = zeros(1, N);
dk(1) = bk(1);

for i = 2:N
   dk(i) = modulo(dk(i-1) + bk(i), 2);
end

// Phase assignment
phase = zeros(1, N);
for i = 1:N
   if dk(i) == 1 then
      phase(i) = 0;
   else
      phase(i) = %pi;
   end
end

// Time and carrier
Tb = 1;                    // Bit duration
t  = 0:0.001:N*Tb;         // Time axis
fc = 5;                    // Carrier frequency (Hz)

// DPSK waveform generation
s = zeros(1, length(t));
for i = 1:N
   idx = find(t >= (i-1)*Tb & t < i*Tb);
   s(idx) = cos(2*%pi*fc*t(idx) + phase(i));
end

// Plot DPSK waveform
figure;
plot(t, s);
xlabel("Time");
ylabel("Amplitude");
title("DPSK Signal Waveform");
```

CONSTELLATION DIAGRAM OF QUADRATURE PHASE SHIFT KEYING

```
// Caption : Signal space diagram for coherent QPSK waveform
// Figure  : Signal Space Diagram for coherent QPSK system
```

```
clc;
clear;
close;

// Number of symbols
M = 4;
i = 1:M;

// QPSK constellation points
y = cos((2*i - 1) * %pi / 4) - %i * sin((2*i - 1) * %pi / 4);

// Dibit labels
annot = dec2bin(0:M-1, log2(M));

// Display values
disp("Coordinates of message points:");
disp(y);

disp("Dibits value:");
disp(annot);

// Plot constellation
figure;
a = gca();
a.data_bounds = [-1.5, -1.5; 1.5, 1.5];
a.x_location = "origin";
a.y_location = "origin";

// Plot each QPSK point
plot2d(real(y(1)), imag(y(1)), -2);
plot2d(real(y(2)), imag(y(2)), -4);
plot2d(real(y(3)), imag(y(3)), -5);
plot2d(real(y(4)), imag(y(4)), -9);

// Labels and title
xlabel("In-Phase");
ylabel("Quadrature");
title("Constellation for QPSK");

// Legend
legend([
    "Message point 1 (dibit 10)";
    "Message point 2 (dibit 00)";
    "Message point 3 (dibit 01)";
    "Message point 4 (dibit 11)"
], 5);
```

Spread Spectrum Modulation- Processing gain, PN sequence length and Jamming margin

```
clc;
clear;
close;

// Given values
Tb = 4.095 * 10^-3;   // Information bit duration (s)
Tc = 1 * 10^-6;       // PN chip duration (s)

// Processing Gain
PG = Tb / Tc;
disp("The processing gain (PG) is:");
disp(PG);

// PN sequence length
N = PG;
disp("The required PN sequence length (N) is:");
disp(N);

// Feedback shift register length
m = log2(N + 1);
disp("The feedback shift register length (m) is:");
disp(m);

// Energy to noise density ratio
Eb_No = 10;

// Jamming Margin
J_P = PG / Eb_No;
JM_dB = 10 * log10(J_P);

disp("Jamming Margin in dB:");
disp(JM_dB);
```

SLOW AND FAST FREQUENCY HOPPING

```
clc;
close;

// Given values
K = 2;          // Number of bits per symbol
M = 2^K;        // Number of MFSK tones
N = 2^M - 1;    // Period of the PN sequence
k = 3;          // Length of PN sequence per hop

// Display results
disp(K, "number of bits per symbol K =");
```

```
disp(M, "Number of MFSK tones M =");
disp(N, "Period of the PN sequence N =");
disp(k, "length of PN sequence per hop k =");
disp(2^k, "Total number of frequency hops =");
```