CE Lab Notes

Last updated: 2025-10-01 14:40

LaTeX source here

Contents

1 AM modulation and demodulation **2**
- 1.1 Double Side Band Full carrier: DSBFC . . . 2
- 1.2 Double Side Band Suppressed carrier: DSBSC . . . 4
- 1.3 Single Side Band Suppressed carrier: SSBSC . . . 6
  - 1.3.1 Case where message and carrier are cos wave . . . 7
  - 1.3.2 General case (using Hilbert transform) . . . 7
  - 1.3.3 Demodulation . . . 8

2 FM modulation and demodulation **10**
- 2.1 Modulation . . . 10
- 2.2 Demodulation . . . 11
- 2.3 Program for narrow and wide band FM modulation and demodulation . . . 11

3 PAM modulator and demodulator **13**

4 Binary Shift Keying **15**

5 Line Codes **18**

6 M-ary modulation schemes **22**
- 6.1 Context . . . 22
- 6.2 M-ary ASK . . . 22
- 6.3 M-ary PSK . . . 24
- 6.4 M-ary FSK . . . 27
- 6.5 BER vs SNR, Signal constellation . . . 28

Note

These are notes I made along while I was studying for exam (and later refined). Many implementations done here are directly based on the theory. Hence certain things might be confusing at first. This is intended as a reference, and defintly not something to mug up night before exam. :)
Also in each program I've put plotting codes inside a function (such as `plot_sig_fft`, `plot_`, `stairz`, etc..) This is to make code look neater, so that logic of program is not much obstructed by code to plot things

1 AM modulation and demodulation

There're 3 types of AM modulation:

1. DSBSC
2. DSBFC
3. SSBSC

1.1 Double Side Band Full carrier: DSBFC

For message signal $m(t)$ and carrier $c(t)$, DSBFC is given by:

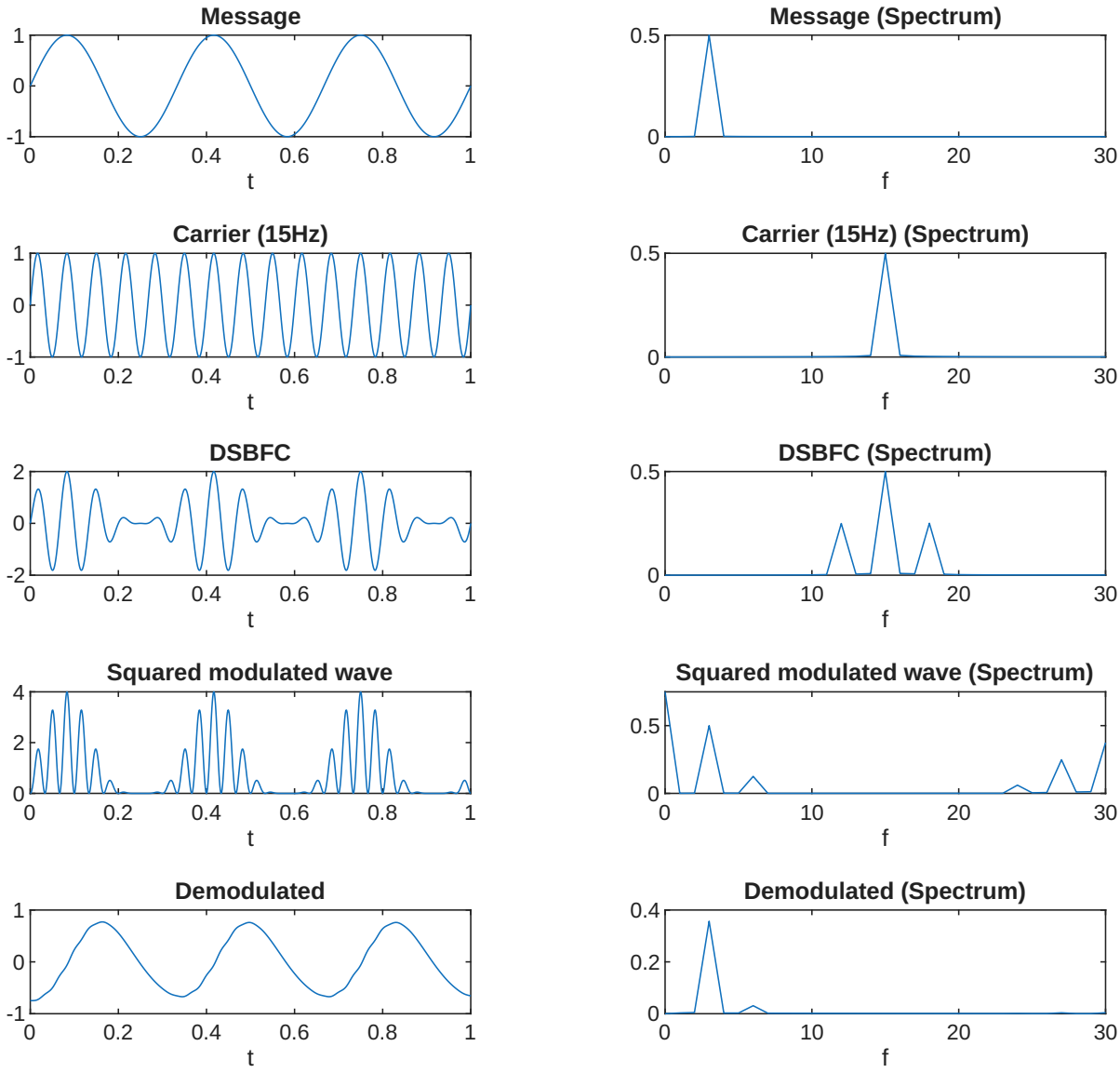
$$s(t) = (1 + k \times m(t))c(t)$$

where k is modulation order / modulation index
Demodulation can be done by a Square law demodulator. Here modulated wave is squared and passed through a LPF (with cutoff near to message frequency). Additionally DC component can be removed (if needed).

Program

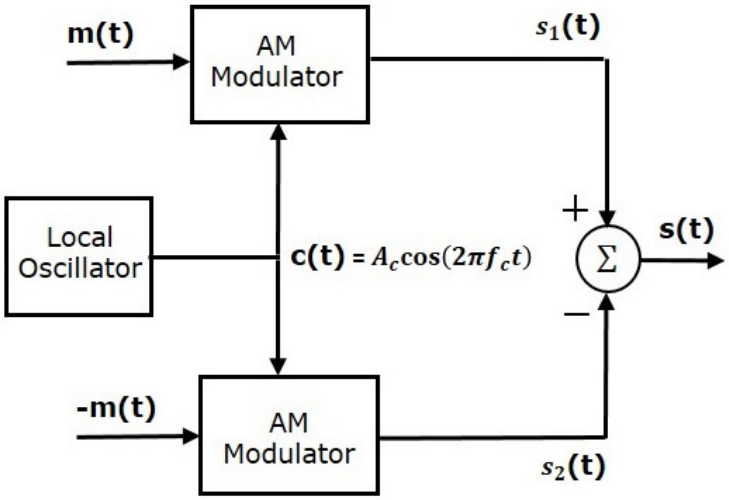
```
clearvars; % clears all variables
% to make a variable accessible even inside a function
% it has to be declared as global
global fs;
fs = 1000; % sampling frequency
fm = 3; % message frequency
fc = 15; % Carrier frequency
t = linspace(0, 1, fs);
k = 1; % modulatoin index
m = sin(2 * pi * fm * t); % message
c = sin(2 * pi * fc * t); % carrier
s = (1 + k * m) .* c; % modulated wave

tiledlayout(5, 2);
plot_sig_fft(t, m, "Message");
plot_sig_fft(t, c, "Carrier (15Hz)");
plot_sig_fft(t, s, "DSBFC");

%%% demodulation
sq = s .^ 2;
[b, a] = butter(2, fm / (fs / 2)); % 2nd order low pass filter with cut off fm
y = filter(b, a, sq);
y = y - mean(y); % remove DC component by subtracting avg value
```

```
plot_sig_fft(t, sq, "Squared modulated wave");
plot_sig_fft(t, y, "Demodulated");
function plot_sig_fft(x, y, name)
    % plots a signal y and its fourier transform
    global fs;
    nexttile;
    plot(x, y);
    title(name); xlabel("t")

    nexttile;
    N = length(y);
    freq = abs(fftshift(fft(y))) / N;
    f = (-N / 2:N / 2 - 1) * fs / N;
    plot(f, freq);
    xlim([0, 30]); xlabel("f")
    title(name + " (Spectrum)");
end
```



1.2 Double Side Band Suppressed carrier: DSBSC

Modulation can be done using a balenced modulator:



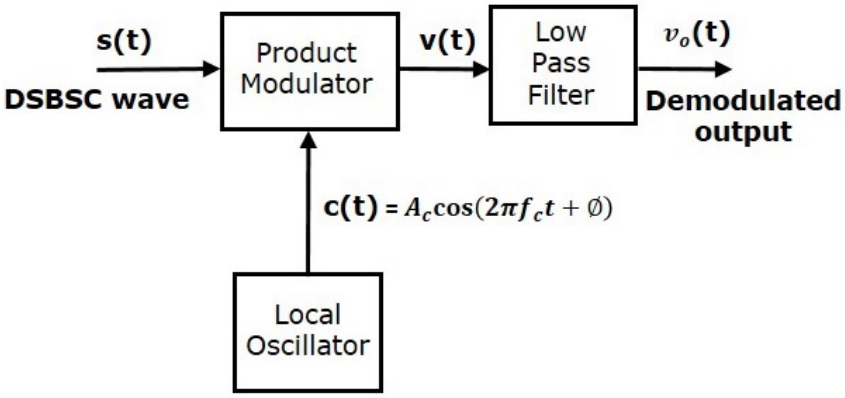
Hence expression for modulated wave is:

$$\begin{aligned} s(t) &= s_1(t) - s_2(t) \\ &= (1 + k \times m(t))c(t) - (1 - k \times m(t))c(t) \\ s(t) &= 2k \times m(t)c(t) \end{aligned}$$

if we ignore $2k$ factor:

$$s(t) = m(t)c(t)$$

Demodulation can be done by Coherent detection:



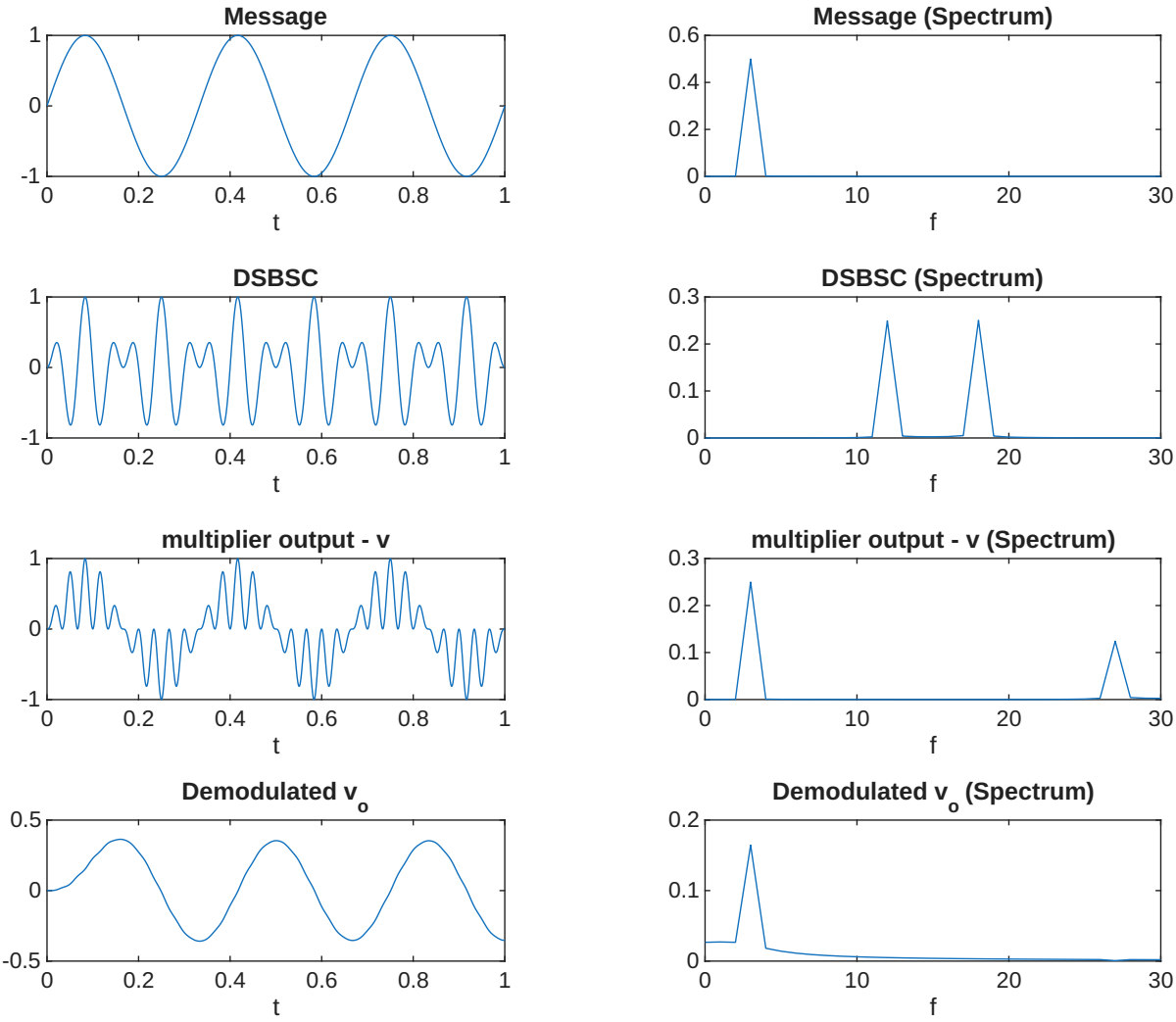
Hence demodulated output is:

$$v_o(t) = \text{LPF}\left[s(t) \cdot c(t)\right]$$

Program

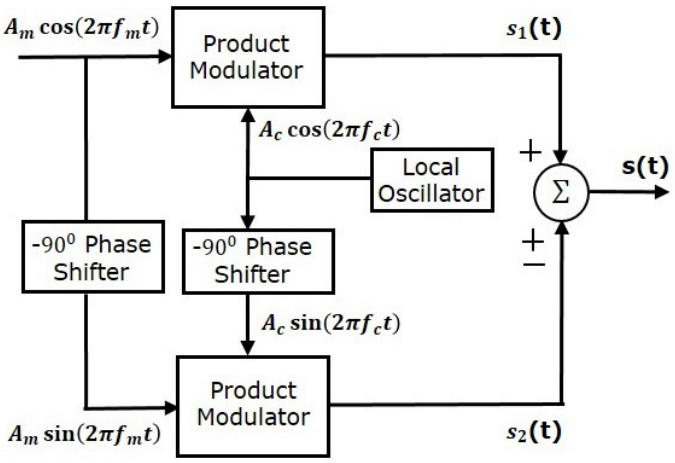
```
clearvars;
```

```
global fs;
fs = 1000;
fm = 3;
fc = 15;
t = linspace(0, 1, fs);

m = sin(2 * pi * fm * t);
c = sin(2 * pi * fc * t);
s = m .* c;

tiledlayout(4, 2);
plot_(t, m, "Message");
plot_(t, s, "DSBSC");

%%% demodulation
v = s .* c;
[b ,a] = butter(2, fm / (fs/ 2));
y = filter(b, a, v);
plot_(t, v, "multiplier output - v");
plot_(t, y, "Demodulated v_o");
function plot_(x, y, name)
    global fs;
    nexttile;
    plot(x, y);
    title(name); xlabel("t")

    nexttile;
    freq = abs(fftshift(fft(y))) / length(y);
    N = length(freq);
    f = (-N/2:N/2-1) * fs / N;
    plot(f, freq);
    xlim([0, 30]); xlabel("f")
    title(name + " (Spectrum)");
end
```



1.3 Single Side Band Suppressed carrier: SSBSC

A reliable Modulation method is Frequency Discrimination Method. This involves shifting phase of message and carrier signals. This can be done manually by writing their expressions, or by using using Hilbert transform.



1.3.1 Case where message and carrier are cos wave

For a specific case of sine waves: if message is $m(t) = A_m \cos(2\pi f_m t)$ and carrier is $c(t) = A_c \cos(2\pi f_c t)$.
Product modulator output in top section is:

$$\begin{aligned} s_1(t) &= A_m A_c \cos(2\pi f_m t) \cos(2\pi f_c t) \\ &= \frac{A_m A_c}{2} \{\cos[2\pi(f_c + f_m)t] + \cos[2\pi(f_c - f_m)t]\} \end{aligned}$$

In bottom section the message and the carrier signal are phase shifted by $-90°$ before applying as inputs to the lower product modulator.
Therefore the output of lower product modulator is

$$\begin{aligned} s_2(t) &= A_m A_c \cos\left(2\pi f_m t - 90^0\right) \cos\left(2\pi f_c t - 90^0\right) \\ &= \frac{A_m A_c}{2} \{\cos[2\pi(f_c - f_m)t] - \cos[2\pi(f_c + f_m)t]\} \end{aligned}$$

To get lower sideband Add $s_1(t)$ and $s_2(t)$

$$\begin{aligned} s(t) &= s_1(t) + s_2(t) \\ \boxed{s(t) &= A_m A_c \cos[2\pi(f_c - f_m)t]} \end{aligned}$$

To get upper sideband subtract $s_2(t)$ from $s_1(t)$

$$\begin{aligned} s(t) &= s_1(t) - s_2(t) \\ \boxed{s(t) &= A_m A_c \cos[2\pi(f_c + f_m)t]} \end{aligned}$$

1.3.2 General case (using Hilbert transform)

Hilbert transform on a signal $x(t)$ provides $\pm 90°$ phase shift to all frequencies present in the applied signal. $-90°$ phase shifted version of signal $x(t)$ can be obtained by taking imaginary part of Hilbert transform of the signal:

$$x'(t) = \mathfrak{Im}\{\mathcal{H}(x(t))\}$$

Hence the $s_2(t)$ can be written in terms of Hilbert transform as :

$$s_2(t) = \mathfrak{Im}\{\mathcal{H}(m(t))\} \times \mathfrak{Im}\{\mathcal{H}(c(t))\}$$

From this lower side band can be obtained as:

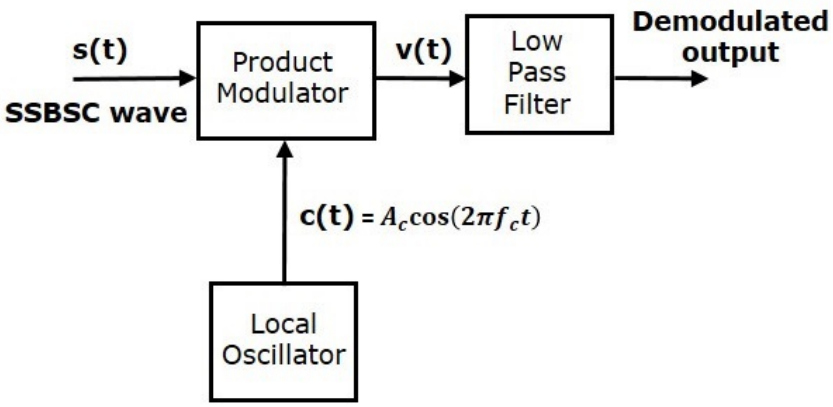
$$s_{\text{LSB}}(t) = s_1(t) + s_2(t)$$

And upper side band can be obtained as:

$$s_{\text{USB}}(t) = s_1(t) - s_2(t)$$

1.3.3 Demodulation

Demodulation can be done by a coherent detector:

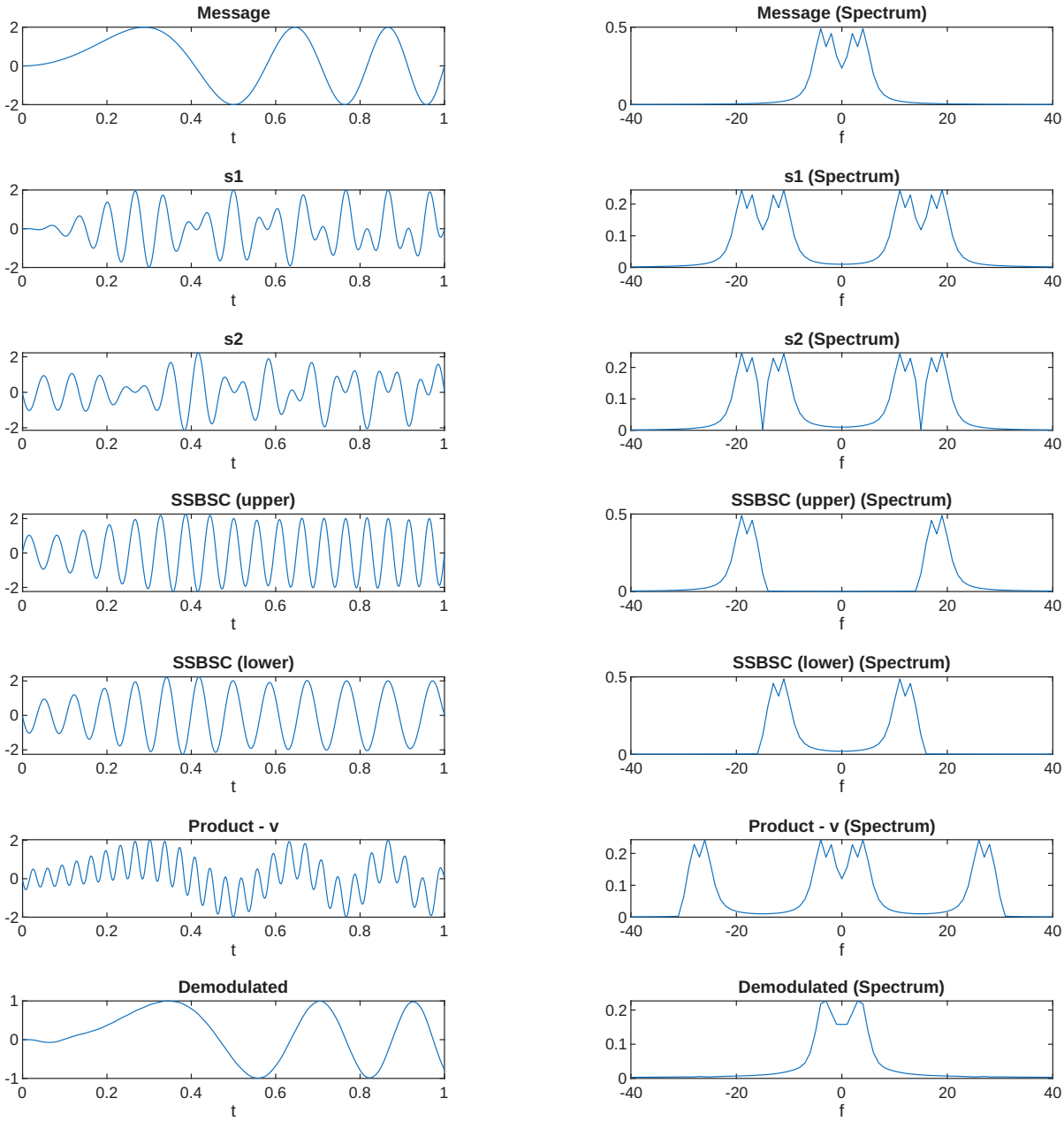


Program (with Hilbert transform)

Here I have demodulated the lower sideband. To get best result, cutoff frequency should be between $2f_m$ and f_c.
Message signal here is: $m(t) = 2\sin(2\pi f_m t^2)$;

```
clearvars;
global fs;
fs = 1000;
fm = 3;
fc = 15;
t = linspace(0, 1, fs);

m = 2*sin(2 * pi * fm * t .^2);
c = cos(2 * pi * fc * t);
s1 = m .* c;
s2 = imag(hilbert(m)) .* imag(hilbert(c));
su = s1 - s2;
sl = s1 + s2;

tiledlayout(7, 2);
plot_(t, m, "Message");
plot_(t, s1, "s1");
plot_(t, s2, "s2");
plot_(t, su, "SSBSC - upper");
plot_(t, sl, "SSBSC - lower");

%%% demodulation with lower sideband
v = sl .* c;
[b, a] = butter(5, fm*3 / (fs / 2));
y = filter(b, a, v);
plot_(t, v, "Product - v");
plot_(t, y, "Demodulated");

function plot_(x, y, name)
```

```
    global fs;
    nexttile;
    plot(x, y);
    title(name); xlabel("t")

    nexttile;
    freq = abs(fftshift(fft(y))) / length(y);
    N = length(freq);
    f = (-N/2:N/2-1) * fs / N;
    plot(f, freq);
    xlim([-40, 40]); xlabel("f")
    title(name + " (Spectrum)");
end
```



2 FM modulation and demodulation

2.1 Modulation

For a message signal $m(t)$ and carrier $c(t) = A_c \cos(2\pi f_c t)$ the modulated wave is:

$$s(t) = A_c \cos\left(2\pi f_c t + 2\pi k_f \int_0^t m(\tau) d\tau\right)$$

Where k_f is known as *frequency sensitiviy*. In the case of single tone modulation, where $m(t) = A_m \cos(2\pi f_m t)$ or $m(t) = A_m \sin(2\pi f_m t)$:

$$\begin{aligned} s(t) &= A_c \cos\left(2\pi f_c t + 2\pi k_f \int_0^t A_m \cos(2\pi f_m \tau) d\tau\right) \\ &= A_c \cos\left(2\pi f_c t + \frac{2\pi k_f A_m}{2\pi f_m} \sin(2\pi f_m \tau)\right) \\ s(t) &= A_c \cos\left(2\pi f_c t + \frac{k_f A_m}{f_m} \sin(2\pi f_m \tau)\right) \end{aligned}$$

Here $k_f A_m = \Delta f$ is called *frequency deviation* and $\frac{k_f A_m}{f_m} = \frac{\Delta f}{f_m} = \beta$ is called modulation index.
For a general message signal $m(t)$, modulation index can be defiend as (as far as I looked up in various sources):

$$\beta = \frac{k_f \cdot m_{peak}}{f_{max}}$$

where f_{max} is maximum frequency component present in the message signal. And m_{peak} is peak value of input signal:

$$m_{peak} = \frac{\max[m(t)] - \min[m(t)]}{2}$$

You can see that for a simple sine wave with amplitude A_m this just reduces to $m_{peak} = A_m$.

Note

There are some differences in how β is defined in various sources. Here, I've used one of such definitions. (see *page 212, Modern Digital and Analog Communication Systems, B.P. Lathi*)

2.2 Demodulation

Demodulation is done by finding Hilbert transform of FM, extracting instantaneous phase from it and differentiating it (minus phase offset due to carrier): Hilbert transform:

$$z = \mathcal{H}(s(t))$$

Instantaneous phase angle (obtained by `unwrap(angle(z))`)

$$\theta(t) = \measuredangle z(t)$$

Demodulated wave is:

$$m'(t) = \frac{\partial}{\partial t}(\theta(t) - 2\pi f_c t)$$

2.3 Program for narrow and wide band FM modulation and demodulation

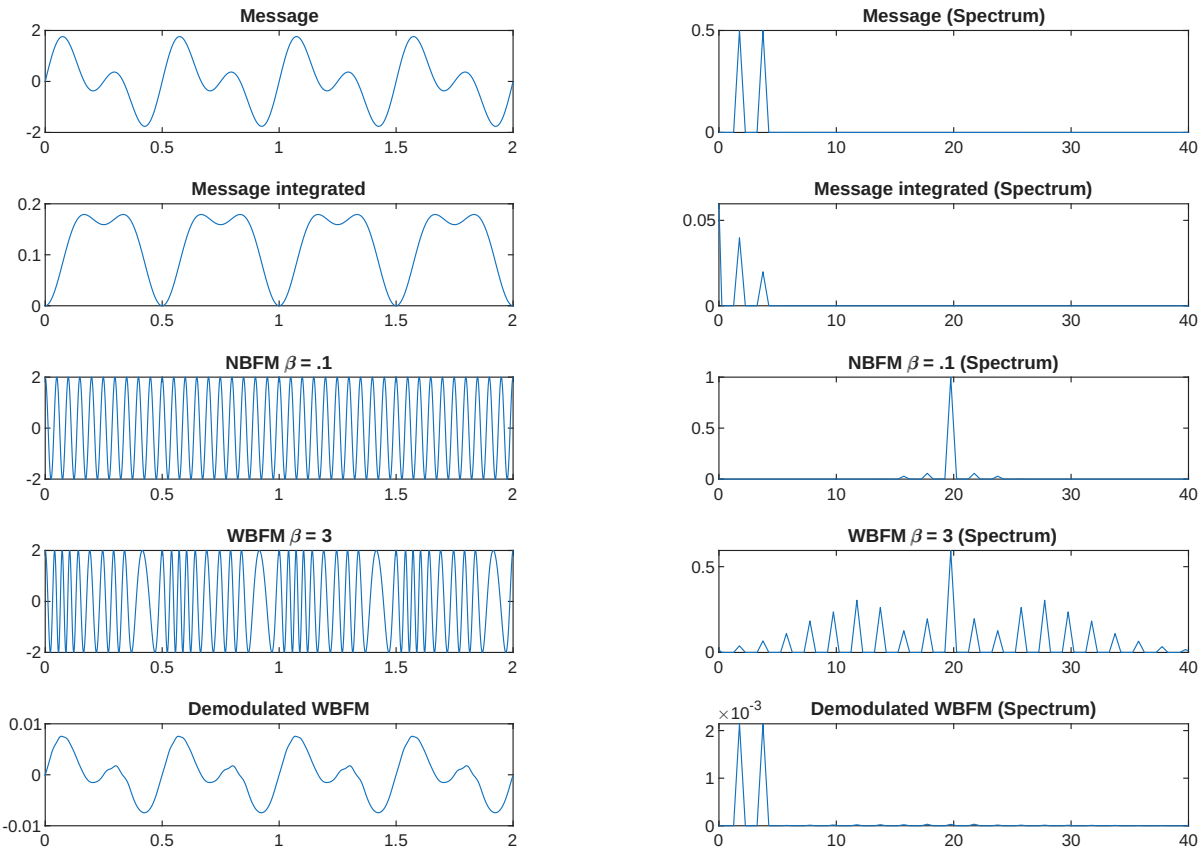
Narrow band FM has $\beta << 1$ and wide band FM has $\beta >> 1$
To modulate at a particular value of β, k_f has to be found out first

$$k_f = \frac{\beta f_{max}}{m_{peak}}$$

But i don't think, this level of perfection is required in lab exam - they didnt ask for details of how it was implemented. In the program I've done this anyway. To integrate message signal cumulative summer `cumsum(m)` is used.
The message signal chosen here is: $m(t) = \sin(2\pi f_m t) + \sin(4\pi f_m t)$. Here $f_{max} = 2f_m$.

```
clearvars; close all;

global fs;
fs = 10000;
fc = 20;
Ac = 2;
fm = 2;
t = 0:1/fs:2;

msg = sin(2 * pi * fm * t) + sin(2 * pi * 2*fm * t);

fmax = 2*fm;

intg = cumsum(msg) / fs;
mpeak = (max(msg) - min(msg))/2;
kf = .1 * fmax / mpeak; % narrow band  = .1
nbfm = Ac * cos(2 * pi * fc * t + 2*pi*kf * intg);

kf = 3 * fmax / mpeak; % wide band  = 3
wbfm = Ac * cos(2 * pi * fc * t + 2*pi*kf * intg);
```

```
tiledlayout(5, 2);
plot_(t, msg, "Message");
plot_(t, intg, "Message integrated");
plot_(t, nbfm, "NBFM \beta = .1");
plot_(t, wbfm, "WBFM \beta = 3");

%%% demodulation
z = hilbert(wbfm);
phase = unwrap(angle(z));
demod = diff(phase - 2*pi*fc * t);
plot_(t, [0, demod], "Demodulated WBFM");
function plot_(x, y, title_)
    % plots y and its fourier transform
    global fs;
    nexttile;
    plot(x, y);
    xlim([min(x) max(x)])
    title(title_)

    nexttile;
    N = length(y);
    freq = abs(fftshift(fft(y))) / N;
    f = (-N/2:N/2-1) * fs / N;
    plot(f, freq);
    title(title_ + " (Spectrum)")
    xlim([0 40])
end
```



3 PAM modulator and demodulator

PAM modulation is done by multiplying a square wave by message signal. *The minimum amplitude of square wave should be 0*, which can be either constructed by `square` or `sin` function:

- `sin(2 * pi * fc * t) > 0`
- `square(2 * pi * fc * t) > 0`

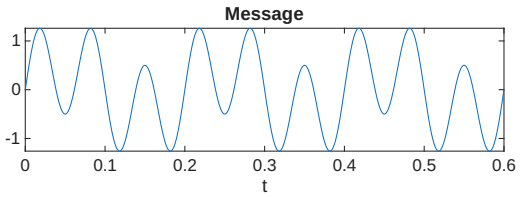
Demodulation is done by a passing PAM wave to low pass filter with cut off frequency slightly higher than highest frequency in the message signal.

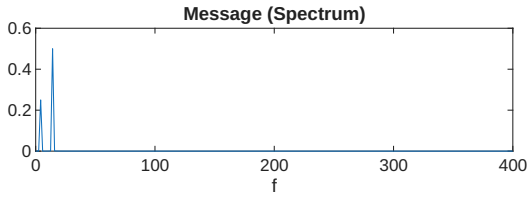
Program

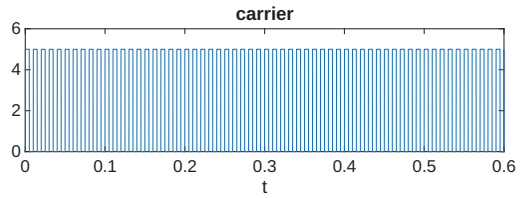
Here message is $m(t) = 0.5\sin(2\pi f_m t) + \sin(6\pi f_m t)$

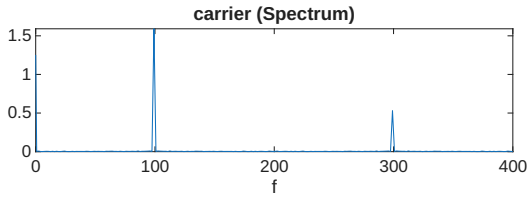
```
clc; clearvars; close all;

global fs; % sampling frequency
fs = 10000;

fm = 5;
fc = 100;
Ac = 5;
t = 0:1/fs:.6;
carrier = Ac * (sin(2 * pi * fc * t) > 0);
msg = .5 * sin(2 * pi * fm * t) + sin(2 * pi * 3*fm * t);

pam = carrier .* msg;

tiledlayout(4, 2);
plot_(t, msg, "Message");
plot_(t, carrier, "carrier");
plot_(t, pam, "PAM");

%%% demodulation
% a little more than highest frequency present in the message
% but far less than carrier frequency
fcutoff = 4*fm;
[b, a] = butter(5, fcutoff / (fs/2));
y = filter(b, a, pam);

plot_(t, y, "Demodulated wave");

function plot_(x, y, name)
    % plots y and its fourier transform
    global fs;
```

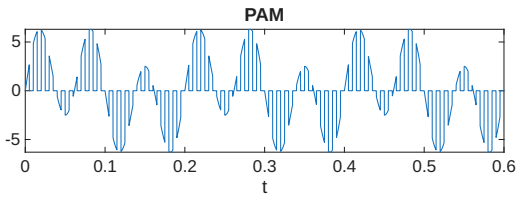
```
    nexttile;
    plot(x, y);
    xlabel("t");
    title(name)

    nexttile;
    N = length(y);
    freq = abs(fftshift(fft(y))) / N;
    f = (-N/2:N/2-1) * fs / N;
    plot(f, freq);
    title(name + " (Spectrum)")
    xlim([0 400])
    xlabel("f");
end
```

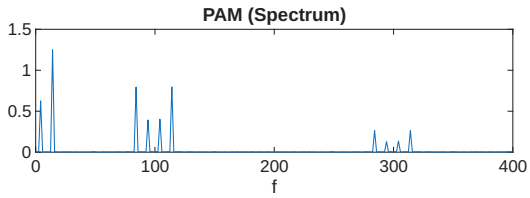


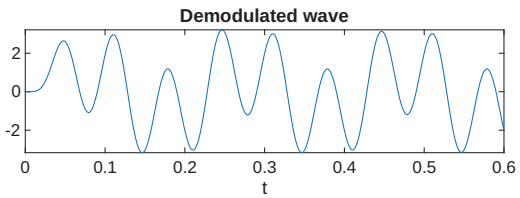


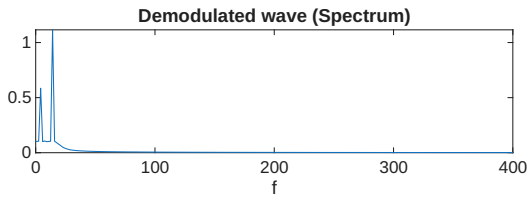












4 Binary Shift Keying

Orthonormal carrier signals in each modulation scheme are:

BASK	$s_1(t) = \sqrt{\frac{2E_b}{T_b}}\cos(2\pi f_c t)$ $s_2(t) = 0$
BPSK	$s_1(t) = \sqrt{\frac{2E_b}{T_b}}\cos(2\pi f_c t)$ $s_2(t) = \sqrt{\frac{2E_b}{T_b}}\cos(2\pi f_c t + \pi)$
BFSK	$s_1(t) = \sqrt{\frac{2E_b}{T_b}}\cos(2\pi f_{c1} t)$ $s_2(t) = \sqrt{\frac{2E_b}{T_b}}\cos(2\pi f_{c2} t)$

Transmitted wave is given by

$$s(t) = \begin{cases} s_1(t) & \text{if bit} = 1 \\ s_2(t) & \text{if bit} = 0 \end{cases}$$

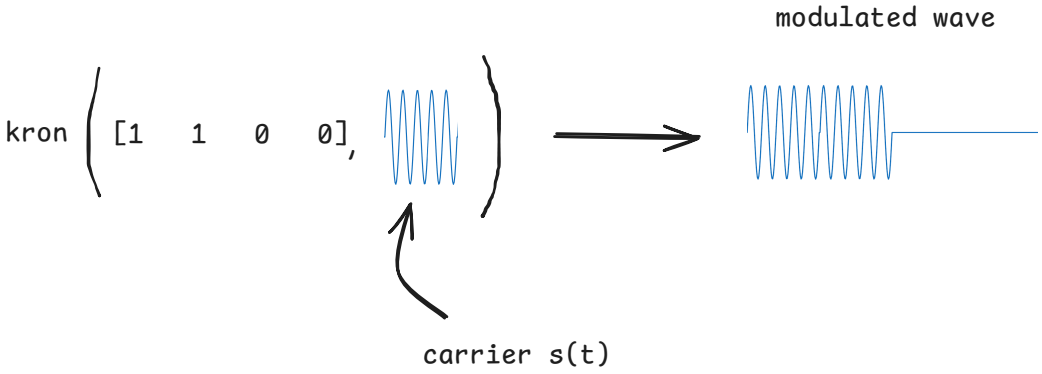
For demodulation, refer: M-ary modulation schemes
Throughout the entire program (and in other programs like line codes, etc), I've used `kron` function. more on that function below:

`kron` function

`kron` is used to implement Kronecker product which is here used to create copies of a carrier signal based on the message signal. Kronecker product of two matricies $A = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$ and $B = \begin{bmatrix} 1 & 1 & 0 & 0 \end{bmatrix}$ is:

$$\begin{aligned} A \otimes B &= \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \otimes \begin{bmatrix} 1 & 1 & 0 & 0 \end{bmatrix} \\ &= \begin{bmatrix} 1\begin{bmatrix} 1 & 1 & 0 & 0 \end{bmatrix} & 2\begin{bmatrix} 1 & 1 & 0 & 0 \end{bmatrix} & 3\begin{bmatrix} 1 & 1 & 0 & 0 \end{bmatrix} \end{bmatrix} \\ &= \begin{bmatrix} 1 & 1 & 0 & 0 & 2 & 2 & 0 & 0 & 3 & 3 & 0 & 0 \end{bmatrix} \end{aligned}$$

From this you can see `kron` can produce any required copies of a carrier signal based on message bit stream:

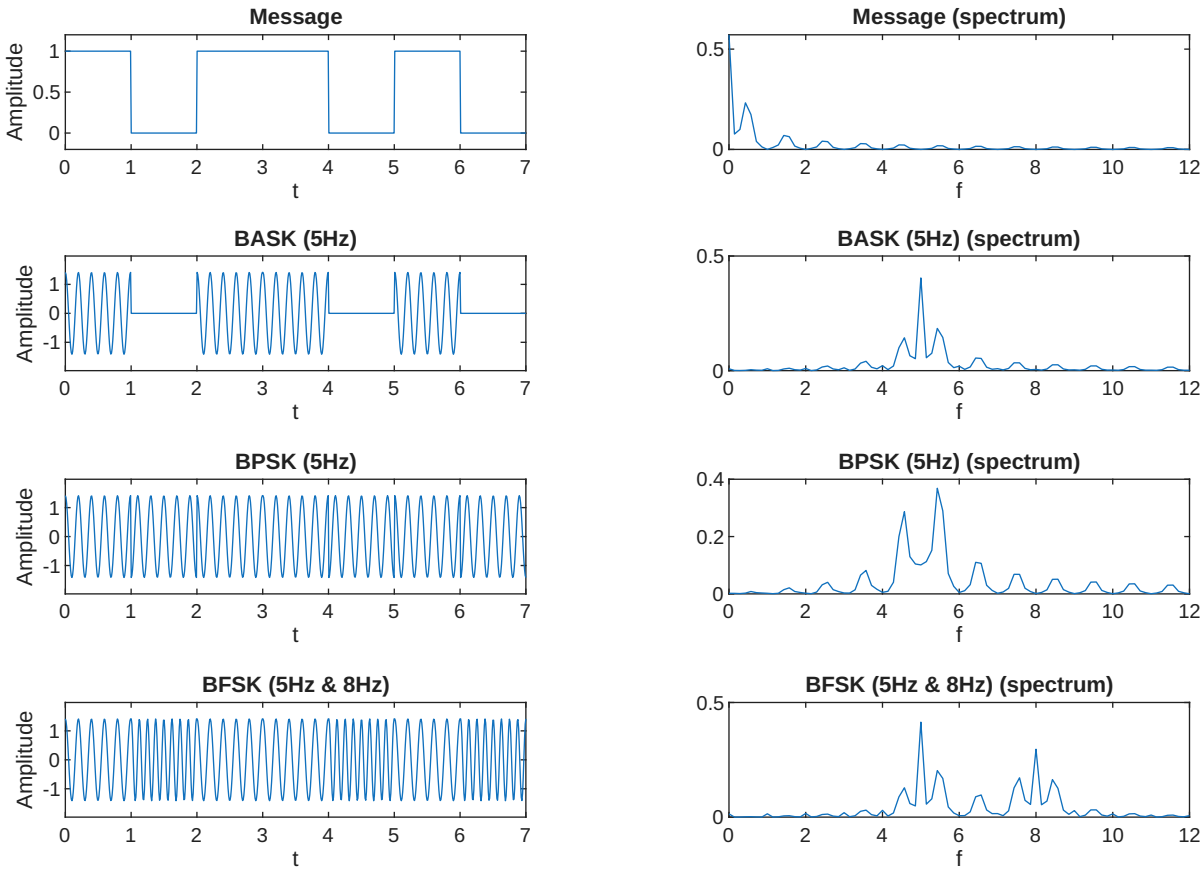


This function is used extensively throughout the remaining MATLAB codes in this PDF.

Program

```
clearvars; close all;

Tb = 1; % symbol period
Eb = 1; % symbol energy
global fs
fs = 100; % freq at which basis functions are sampled
t = linspace(0, Tb, fs);

msgbits = [1 0 1 1 0 1 0];
N = length(msgbits);

% basis functions
fc1 = 5;
fc2 = 8;
s1 = sqrt(2*Eb/Tb) * cos(2*pi*fc1.*t);
s2 = sqrt(2*Eb/Tb) * cos(2*pi*fc1.*t+pi);
s3 = sqrt(2*Eb/Tb) * cos(2*pi*fc2.*t);

msgsampled = kron(msgbits, ones(1, fs));
bask = kron(msgbits, s1);

% "~msgbits" inverts the bitstream
bpsk = kron(msgbits, s1) + kron(~msgbits, s2);
bfsk = kron(msgbits, s1) + kron(~msgbits, s3);

tiledlayout(4, 2);
t = linspace(0, Tb*N, fs*N);
plot_(t, msgsampled, "Message");
plot_(t, bask, "BASK (5Hz)");
plot_(t, bpsk, "BPSK (5Hz)");
plot_(t, bfsk, "BFSK (5Hz & 8Hz)");
```

```
function plot_(t, fx, title_)
    global fs;
    nexttile;
    plot(t, fx);

    % just a hack to increase ylimit according to whatever signal is
    k = (max(fx) - min(fx)) * 0.2;
    ylim([min(fx) - k, max(fx) + k]);

    title(title_);
    xlabel("t"); ylabel("Amplitude");

    N = length(fx);
    freq = abs(fftshift(fft(fx))) / N;
    f = (-N/2:N/2-1) / N * fs;

    nexttile;
    plot(f, real(freq));
    xlim([0, 12]); xlabel("f");
    title(title_ + " (spectrum)");
end
```



5 Line Codes

There are many line coding methods, Few of them are:

1. Non-Polar Return to Zero (NPRZ)
2. Non-Polar Non Return to Zero (NPNRZ)
3. Polar Return to Zero (PRZ)
4. Polar Non Return to Zero (PNRZ)
5. Manchester
6. Bipolar
7. Differential

To encode a input bitstrean $x(n)$, over a symbol period T_b, following expression are used to generate the required line code sequence.

NPRZ	$s(t) = \begin{cases} 1 & \text{if } x(n) = 1 \\ 0 & \text{if } x(n) = 0 \end{cases}$ For entire bit interval
NPNRZ	for $x(n) = 1$, $s(t) = \begin{cases} 1 & 0 \le t \le \frac{T_b}{2} \\ 0 & \frac{T_b}{2} < t \le T_b \end{cases}$ for $x(n) = 0, s(t) = 0$
PRZ	$s(t) = \begin{cases} 1 & 0 \le t \le \frac{T_b}{2} \quad \text{for } x(n) = 1 \\ -1 & 0 \le t \le \frac{T_b}{2} \quad \text{for } x(n) = 0 \\ 0 & \frac{T_b}{2} < t \le T_b \end{cases}$
PNRZ	$s(t) = \begin{cases} 1 & \text{if } x(n) = 1 \\ -1 & \text{if } x(n) = 0 \end{cases}$ For entire bit interval

Manchester (IEEE 802.3)	for $x(n) = 0 \quad s(t) = \begin{cases} 1 & 0 \le t \le \frac{T_b}{2} \\ -1 & \frac{T_b}{2} < t \le T_b \end{cases}$ for $x(n) = 1 \quad s(t) = \begin{cases} -1 & 0 \le t \le \frac{T_b}{2} \\ 1 & \frac{T_b}{2} < t \le T_b \end{cases}$
Bipolar	for $x(n) = 0$, $s(t) = 0$ for $x(n) = 1$, $s(t) = 1$ or -1 alternatively
Differential	$s(n) = x(n) \odot s(n-1)$ (xnor operation)

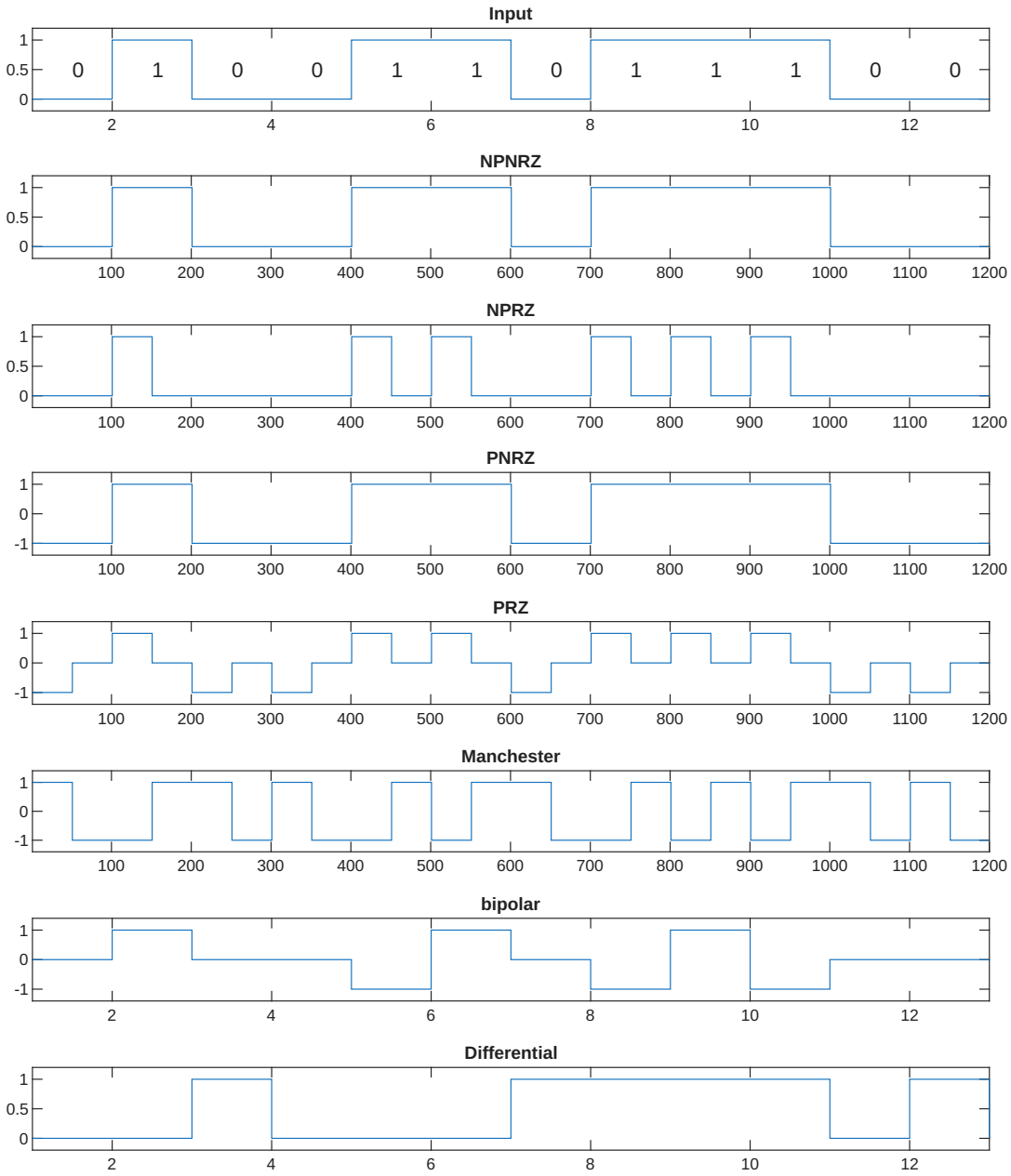
Program

Here bipolar and differential line code output are lists with same number of elements as that of input bitstrean. This is because i haven't sampled them like other codes, (but you can sample them, using `kron` or in other ways).

```
clearvars;

bits = [0 1 0 0 1 1 0 1 1 1 0 0];
N = length(bits);
Tb = 100;
npnrz = kron(bits, ones(1, Tb));
nprz = kron(bits, [ones(1, Tb/2), zeros(1, Tb/2)]);
pnrz = kron(bits * 2  - 1, ones(1, Tb));
prz = kron(bits * 2  - 1, [ones(1, Tb/2), zeros(1, Tb/2)]);
manch = kron(bits * 2  - 1, [-ones(1, Tb/2), ones(1, Tb/2)]);
bipolar = zeros(1, N);
n = 1;
for i = 1:N
    bipolar(i) = bits(i) * n;
    if (bits(i) == 1)
        n = n * -1;
    end
end

bitst = [0, bits]; % pad zero to beginning to make xor comparison easier

% reference bit is 1, if you want 0 as reference bit, use zeros()
diffnt = ones(1, N + 1);
for i = 2:N + 1
    diffnt(i) = ~xor(bitst(i), diffnt(i-1));
end

tiledlayout(8, 1);
stairz([bits, 0], "Input");
```

```
% put bit stream
for i = 1:length(bits)
    text(i+0.5, 0.5, string(bits(i)), 'FontSize', 10);
end
stairz(npnrz, "NPNRZ");
stairz(nprz, "NPRZ");
stairz(pnrz, "PNRZ");
stairz(prz, "PRZ");
stairz(manch, "Manchester");
stairz([bipolar, 0], "bipolar");
stairz([diffnt(2:end), 0], "Differential");

function stairz(bits, title_)
    nexttile;
    stairs(bits);
    title(title_);
    xlim([1 length(bits)])

    % to adjust y limit according to the wave
    d = (max(bits) - min(bits)) * 0.2;
    ylim([min(bits)-d, max(bits)+d]);
end
```

For the purpose of getting nice plot, x axis of all plots starts from 1 (can you guess why?)


Input
0 1 0 0 1 1 0 1 1 1 0 0
NPNRZ
NPRZ
PNRZ
PRZ
Manchester
bipolar
Differential

6 M-ary modulation schemes

In the following programs, I've implemented a general M-ary shift keying modulator. So to get 4-ary modulation (or 8-ary) just set $M = 4$ (or $M = 8$) in the program.

6.1 Context

An M-ary modulation means, for modulating quantity, M different level are used for transmitting M symbols.

- in M-ary FSK: M different frequencies are used.
- in M-ary PSK: M different phases are used.
- in M-ary ASK: M different amplitudes are used.

Usually all these modulated quantities are equally spaced.
When a bit sequence is given, they have to be grouped and converted to equivalent decimal format to do rest of modulation. For M-ary modulation, number of bits in each group is $n = log_2(M)$. If Enough bits are not there (ie we have to group every 2 bits, but there're only 5 bits), additional zeros has to added. For eg. grouping in 4-ary will be:
00101 -> 00 10 10 (grouping of 2 bits)
and in 8-ary:
00101 -> 001 010 (grouping of 3 bits)
Where the zero should be added? It cab be added in either front or back of the sequence, as long as you can create a demodulate scheme which discards this extra zero. (in this case I've added it at the end of each group).

6.2 M-ary ASK

Generally for M signals, M different amplitude levels should be used. For eg. in QASk, 4 equally spaced amplitude levels will be used. In practice, it can be of any value: [1 2 3 4] or [-3 -1 1 3], etc...
In my implementation, I created amplitude levels corresponding to each decimal digit d which ranges from 0 to $M - 1$ by interpolating d to range from L_{min} to L_{max} (which corresponds to minimum and maximum amplitude level). Hence amplitude corresponds to decimal digit d is

$$\therefore \text{amplitude(d)} = \frac{d}{M-1} * (L_{max} - L_{min}) + L_{min}$$

Program

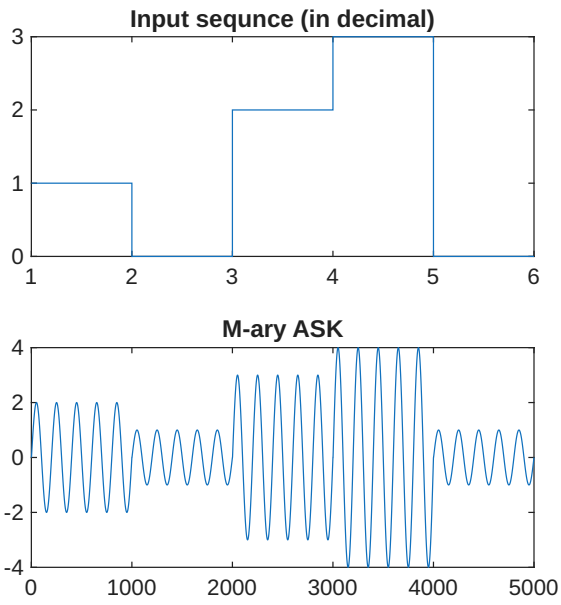
```
% message
x = [0 1 0 0 1 0 1 1 0];
```

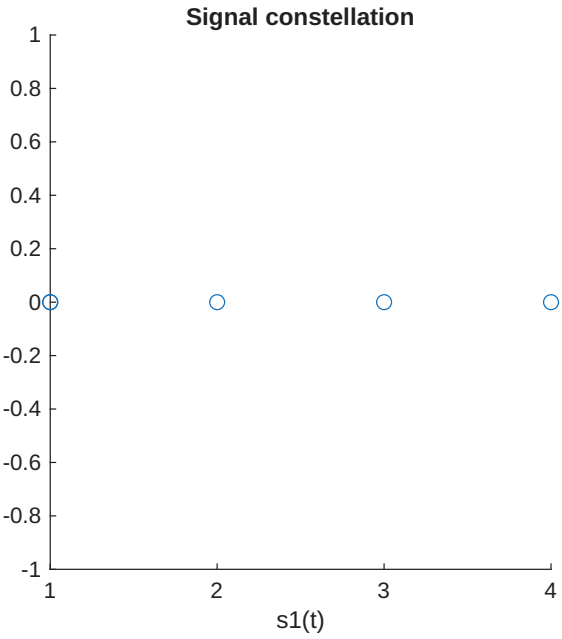
```
M = 4; % modulation order
fc = 5; % carrier freq
fs = 1000; % sampling freq
t = linspace(0, 1, fs);
s1 = sin(2 * pi * fc * t); % carrier
% group each bit to form a decimal number
n = log2(M); % number of bits to group

if mod(length(x), n) ~= 0
    % if length of sequence is not multiple of n, add some zeros
    x = [x, zeros(1, n - mod(length(x), n))];
end

% length of sequence when converted to decimal
seqLength = length(x)/n;
x_decimal = zeros(1, seqLength);
for i=0:seqLength-1
    bitgroup = x((i * n + 1):(i * n + n));
    x_decimal(i+1) = bi2de(bitgroup, "left-msb");
end

% M-ary ASK
max_level = 4;
min_level = 1;
amplitudes = x_decimal / (M-1) * (max_level - min_level) + min_level;
mASK = kron(amplitudes, s1);

subplot(2, 2, 1);
stairs([x_decimal, 0]);  % one more 0 is added to fully show the array
title("Input sequnce (in decimal)");

subplot(2, 2, 3); plot(mASK);
title("M-ary ASK");

subplot(2, 2, [2, 4]);
scatter(real(amplitudes), imag(amplitudes));
title("Signal constellation"); xlabel("s1(t)")
```

The length of sequnce has to be adjusted in such a way that grouping is possible. for 4-ary, input length should be multiple of 2, for 8-ary, input length should be multiple of 3, and so on.. To do such adjustment, some number of bits should be added. The following code checks if input length is multiple of n and if not, it adds required number of zeros.

```
if mod(length(x), n) ~= 0
    x = [x, zeros(1, n - mod(length(x), n))];
end
```

for eg.: if n = 3 (for 8-ary modulation), and input length is 7, `mod(7, 3) = 1`, hence 3 - 1 = 2 more zeros are added.



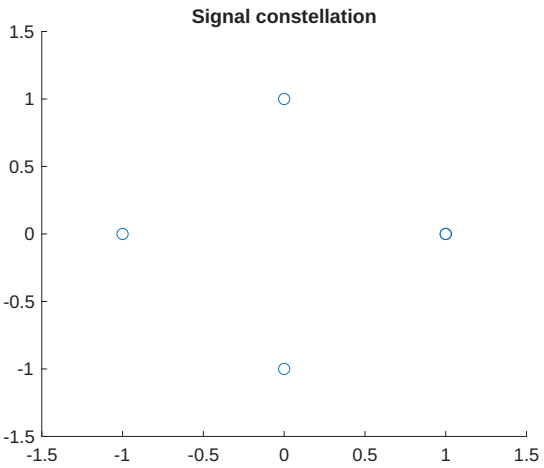


6.3 M-ary PSK

M-ary here divides phases from 0 to 2π into M intervals. If each group of bits corresponds to digit d. Then corresponding M-PSK signal is:

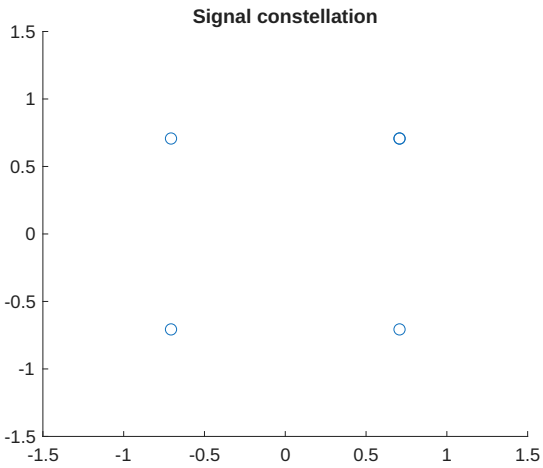
$$s(t) = A_c \sin\left(2\pi f_c t + \frac{2\pi}{M} d\right)$$

Where d can take valus either from 0 to $M-1$ or 1 to M. Notice that in this representation the signal constellation will contain a signal vector at 0°. This signal vector at 0° can be avoided by adding additional phase shift of π/M to whole signal. This is illustrated in following:



With signal vector at 0°

$$s(t) = A_c \sin\left(2\pi f_c t + \frac{2\pi}{M} d\right)$$



No signal vector at 0°

$$s(t) = A_c \sin\left(2\pi f_c t + \frac{2\pi}{M} d + \frac{\pi}{M}\right)$$

Program

Here I've implemented a modulator that doesn't produce signal vector at 0° phase shift. Here's how it is implemented:
I've expressed all signals in complex form, this allows phase shift to be applied to signals by just multiplying by another complex number. The carrier signal is written in complex form:

$$c(t) = A_c \exp\left(j2\pi f_c t\right)$$

Phase shift for a partial symbol d is:

$$\text{phase}(d) = \exp\left[j\left(2\pi\frac{d}{M} + \frac{\pi}{M}\right)\right] \tag{6.1}$$

Hence signal corresponding to given decimal d is:

$$\begin{aligned} s(t) &= c(t) \cdot \text{phase}(d) \\ &= A_c \exp\left(j2\pi f_c t\right) \exp\left[j\left(2\pi\frac{d}{M} + \frac{\pi}{M}\right)\right] \end{aligned}$$

$$\boxed{s(t) = \exp\left[j\left(2\pi f_c t + 2\pi\frac{d}{M} + \frac{\pi}{M}\right)\right]}$$

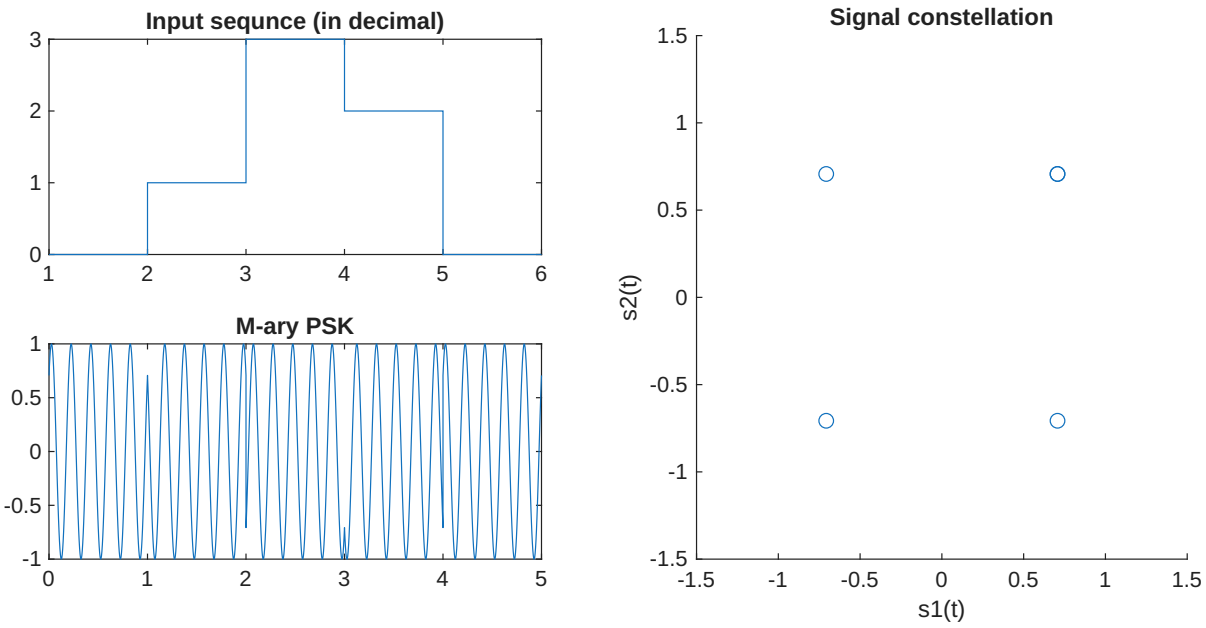
To plot the signal, either real or imaginary part of $s(t)$ can be taken. If imaginary part is taken:

$$\mathfrak{Im}(s(t)) = A_c \sin\left[2\pi f_c t + 2\pi\frac{d}{M} + \frac{\pi}{M}\right]$$

NOTE: in the program, A_c is taken as 1.

```
% message
x = [0 0 0 1 1 1 1 0 0 0 ];

M = 4; % modulation order
fc = 5; % carrier freq
fs = 10000; %sampling freq
t = 0:1/fs:1;
% group each bit to form a decimal number
n = log2(M); % number of bits to group

if mod(length(x), n) ~= 0
    % if length of sequence is not multiple of n, add some zeros
    x = [x, zeros(1, n - mod(length(x), n))];
end

% length of sequence when converted to decimal
seqLength = length(x)/n;
x_decimal = zeros(1, seqLength);
for i=0:seqLength-1
    bitgroup = x((i * n + 1):(i * n + n));
    x_decimal(i+1) = bi2de(bitgroup, "left-msb");
end
```

```
%%%%%%%% M-ary psk
phases = exp(1j * (2 * pi * x_decimal / M + pi/M));
carrier= exp(1j * 2 * pi *  fc * t);
% This generates sequence of carriers that has some phase shift described
% in `phases` array
mPSK = kron(phases, carrier);

subplot(2, 2, 1);
stairs([x_decimal, 0]);  % one more 0 is added to fully show the array
title("Input sequnce (in decimal)");
xlim([1, seqLength+1]);

subplot(2, 2, 3);
t = linspace(0, seqLength, length(mPSK));
plot(t, imag(mPSK));
title("M-ary PSK");

subplot(2, 2, [2, 4]);
scatter(real(phases), imag(phases));
title("Signal constellation");
xlabel("s1(t)"); ylabel("s2(t)");
axis([-1.5, 1.5, -1.5, 1.5]);
```



The signal constellation will have axis which are orthonormal basis functions used in PSK. Here, I've used *orthogonal* signals: $s1(t) = \cos(2\pi f_c t)$ and $s2(t) = \sin(2\pi f_c t)$ which are real and imaginary component of phase(d) in Eq. 6.1.

The case illustrated above is 4-ary. If you want to plot constellation for 8-ary, appropriate message bit stream must be provided, that generates signals of all phases (this is true for all other modulation schemes)

6.4 M-ary FSK

In the same way, M-ary FSK will have M different frequencies:

$$f_c, \quad f_c + \Delta f, \quad f_c + 2\Delta f, \quad \cdots, \quad f_c + (M-1)\Delta f$$

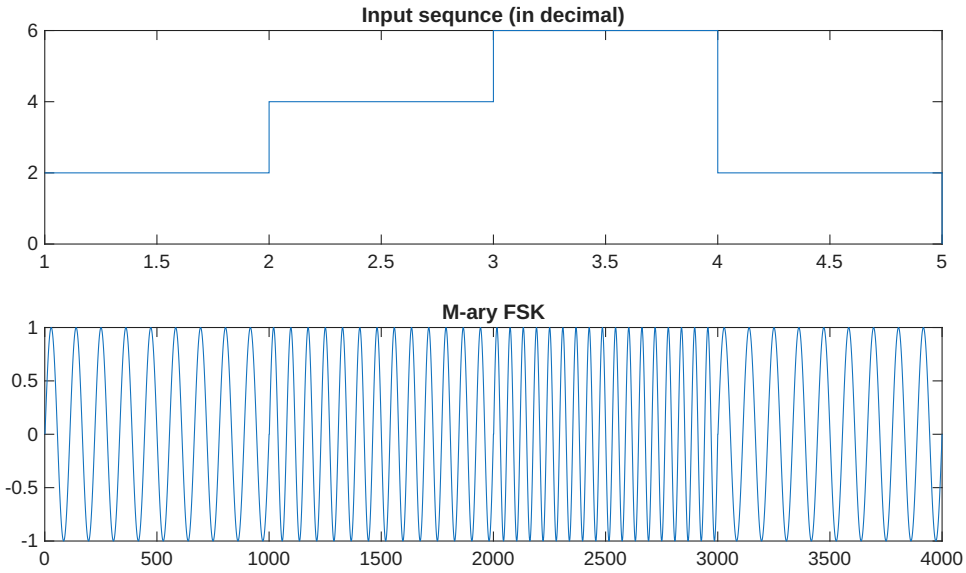
Where Δf is the distance between each frequency.
Hence for each decimal digit d corresponding to some group of binary, signal is given by:

$$s(t) = \sin\left(2\pi(f_c + d\Delta f)t\right)$$

Program

```
% message
x = [0 0 1 0 1 0 0 1 1 0 0 1];

M = 8; % modulation order
fc = 5; % base carrier freq
Df = 3; % distance between each frequency

fs = 1000; % sampling freq
t = linspace(0, 1, fs);
% group each bit to form a decimal number
n = log2(M); % number of bits to group

if mod(length(x), n) ~= 0
    % if length of sequence is not multiple of n, add some zeros
    x = [x, zeros(1, n - mod(length(x), n))];
end

% length of sequence when converted to decimal
seqLength = length(x)/n;
x_decimal = zeros(1, seqLength);
for i=0:seqLength-1
    bitgroup = x((i * n + 1):(i * n + n));
    x_decimal(i+1) = bi2de(bitgroup, "left-msb");
end

%%%%%%%% M-ary FSK
mFSK = [];
for d = x_decimal
    mFSK = [mFSK, sin(2 * pi * (fc + Df * d) * t)];
end
subplot(2, 1, 1);
stairs([x_decimal, 0]);  % one more 0 is added to fully show the array
title("Input sequnce (in decimal)");

subplot(2, 1, 2); plot(mFSK); title("M-ary FSK")
```

The signal constellation of M-ary FSK will have M-dimensions, which is hard to visualize for 4-ary or 8-ary FSK.



6.5 BER vs SNR, Signal constellation

The following is an example for BER(Bit error rate) or Symbol error rate (for M-ary modulation) vs SNR plot of BPSK and QPSK. BER is ratio of number of wrong symbols detected after demodulation to total number of symbols. Usually its a small number hence it is expressed in log scale:

$$\text{BER} = 10\log_{10}\left(\frac{\text{No. of incorrectly decoded symbols}}{\text{Total no. of symbols}}\right)$$

The errors can happen due to noise in the channel. This noise is simulated by using function `awgn(x, snr)` . This function adds noise to input x in such a way that the output SNR is as given to it. Hence to obtain BER at a given `snr` following steps are done: (this is same for any modulation scheme)

1. Modulate signal $x(n)$ and obtain $m(t)$ (modulated wave)
2. apply noise: `y = awgn(x, snr)`
3. demodulate it, and get $d(n)$
4. Count number of symbol errors (using `sum(x ~= d)`)

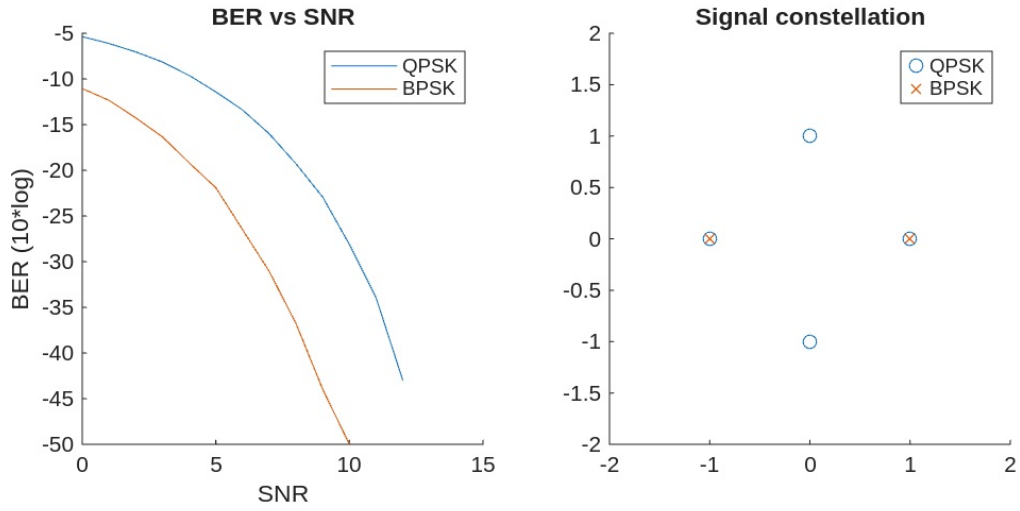
Use a loop to find BER for a range of SNRs

Program

This uses matlab's built in `pskmod` and `pskdemod` functions.

```
clearvars; close all;

% create a random symbol sequene & modulate it
samples = 100000;
qx = randi([0, 3], 1, samples); % for qpsk
bx = randi([0, 1], 1, samples); % for bpsk
```

```
qpsk = pskmod(qx, 4);
bpsk = pskmod(bx, 2);
q_ber = [];
b_ber = [];

for snr = 0:100
    % pass psk to a noisy channel with a particular snr
    q_ch_output = awgn(qpsk, snr);
    b_ch_output = awgn(bpsk, snr);
    q_demod = pskdemod(q_ch_output, 4);
    b_demod = pskdemod(b_ch_output, 2);

    % Calculates BER by counting no. of instances
    % where demodulated output != original input
    be = 10*log10(sum(qx ~= q_demod) / samples);
    q_ber = [q_ber, be];
    be = 10*log10(sum(bx ~= b_demod) / samples);
    b_ber = [b_ber, be];
end
subplot(1, 2, 1); hold on;
plot(0:100, q_ber);
plot(0:100, b_ber);
legend("QPSK", "BPSK");
xlabel("SNR");
ylabel("BER (10*log)")
title("BER vs SNR");

subplot(1, 2, 2); hold on;
scatter(real(qpsk), imag(qpsk), "o");
scatter(real(bpsk), imag(bpsk), "x");
legend("QPSK", "BPSK");
title("Signal constellation");
axis([-2, 2, -2, 2]);
```

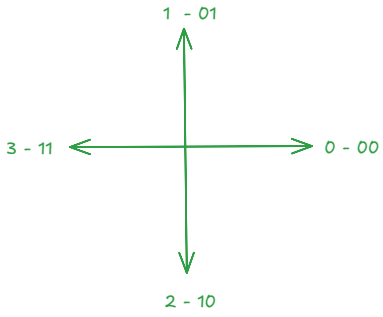


More on Matlab's build-in modulators & demodulators

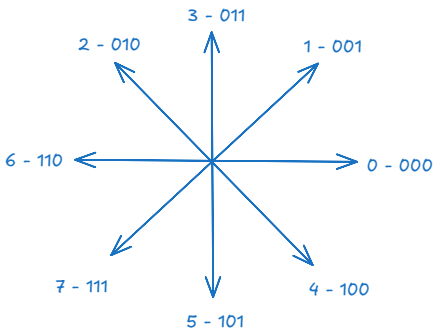
In my implementation, I took a bit sequence and converted it to decimal to do any useful operation. However MATLAB's build in functions only takes decimal values. The modulation also works differently (for eg: in case of PSK, matlab uses gray codes to map each input to corresponding phase value, also it uses different set of phase differences). If you try with PSK, you'll get following points:

```
>> pskmod([0 1 2 3], 4)
ans =
1.0000 + 0.0000i
0.0000 + 1.0000i
-0.0000 - 1.0000i
-1.0000 + 0.0000i

>> pskmod([0 1 2 3 4 5 6 7], 8)
ans =
1.0000 + 0.0000i
0.7071 + 0.7071i
-0.7071 + 0.7071i
0.0000 + 1.0000i
0.7071 - 0.7071i
-0.0000 - 1.0000i
-1.0000 + 0.0000i
-0.7071 - 0.7071i
```

This is visualised as following: (notice that grey code is assigned in clockwise manner)



4-PSK



8-PSK

Subsequently, the demodulation also produces integer sequence corresponding to each modulated vector.

```
>> pskdemod([1, i, -i, -1], 4)
ans =
0     1     2     3
```

Similarly there is `fmmod` and `fmdemod` functions for FM modulation. More on it here: https://in.mathworks.com/help/comm/ref/fmmod.html