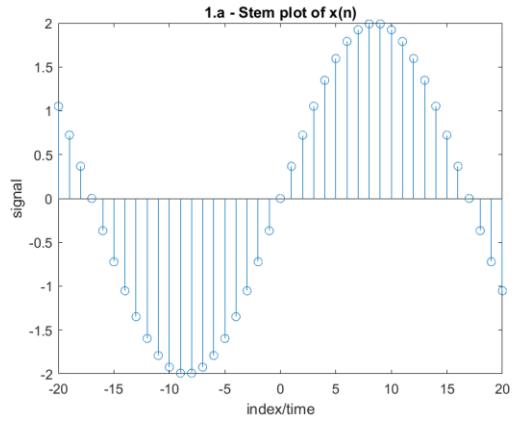
ELEC421 – HW1 - Or Bahari 51277200

1.a.

Stem plot of: $x(n) = 2\sin(\frac{\pi}{17}n)$

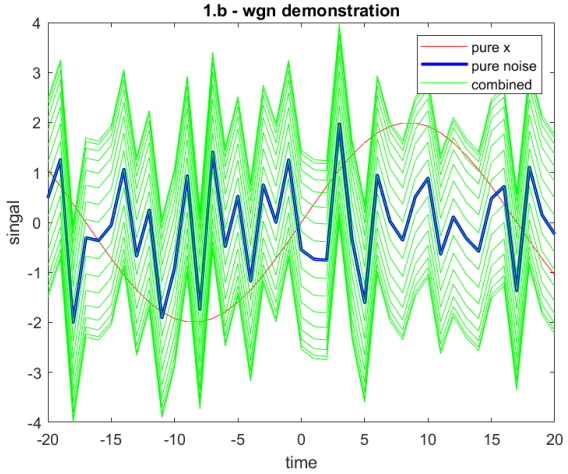


Code:

```
%%1.a
n=[-20:1:20];
x=2*sin(pi*n/17);
stem(n,x);
xlabel("index/time")
ylabel("signal")
title("1.a - Stem plot of x(n)");
```

1.b.

Now we add a wgn (white noise) on the signal, with variance v=1.



Code:

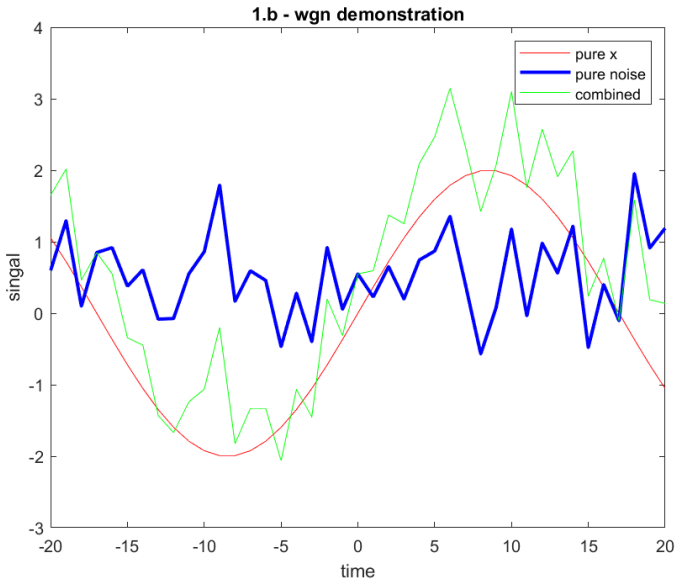
```
%1.b
n=[-20:1:20];
x=2*sin(pi*n/17);
```

```
variance = 1;
W= sqrt(variance)*randn(41,1)
figure(12)
plot(n,x,'red')
hold on;
h2=plot (n,W,'blue');
xlabel("time");
ylabel("singal")
hold on;
plot (n,W+x,'green');
legend('pure x','pure noise','combined');
set(h2,'LineWidth',2);
title('1.b - wgn demonstration');
hold off;
```

1.c.

Now the white noise wgn has properties: mean m=0.5 and variance=0.5.

We get:



And the check for mean and variance of the noise W in matlab:

```
>> mean(W)

ans =

   0.5291

>> var(W)

ans =

   0.3471
```

Code:

```
%1.c
n=[-20:1:20];
x=2*sin(pi*n/17);
```

```
variance = 0.5;
m=0.5;
W=normrnd(m,variance,[1 41])
figure(12)
plot(n,x,'red')
hold on;
h2=plot (n,W,'blue');
xlabel("time");
ylabel("singal")
hold on;
plot (n,W+x,'green');
legend('pure x','pure noise','combined');
set(h2,'LineWidth',2);
title('1.b - wgn demonstration');
hold off;
var(W);
mean(W);
```

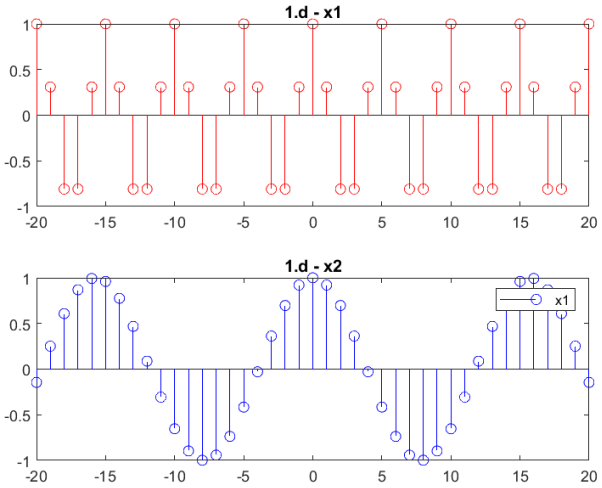
1.d.

2 discrete signals:

$x1(n) = \cos(0.4\pi n)$

$x2(n) = \cos(0.4n)$

(n is an integer)



Let's calculate the time period N for each of them:

For x1(n):

$x1(n) = x1(n+N) = \cos(0.4\pi n + 0.4\pi N)$

Therefore: $0.4\pi N = 2\pi \Rightarrow N = 5$

X1(n) is periodic with time period N=5.

For x2(n):

$x2(n) = x1(n+N) = \cos(0.4n + 0.4N)$

Therefore: $0.4N = 2 \Rightarrow N = 5\,\pi$

N must be an integer therefore x2(n) is not periodic..

** we can actually check our answer from x1(n)'s plot, which repeats itself every 5 points..

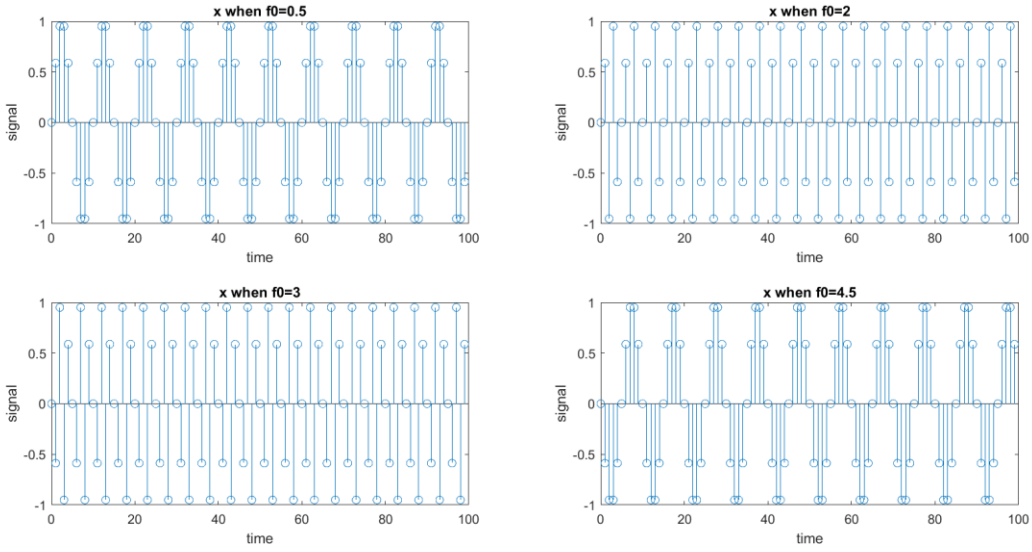
Code:

```
%1.d
n=[-20:1:20];
x1=cos(0.4*pi*n);
x2=cos(0.4*n);
figure(14)
subplot(2,1,1);
stem(n,x1,'red');
title('1.d - x1');
subplot(2,1,2);
stem (n,x2,'blue');
legend('x1','x2');
title('1.d - x2');
```

2.a.

Our sample frequency is Fs=5 therefore (according to Nyquist) signals with f0 frequency higher than 2.5 will be aliased. And in the example we see that f0=4.5 aliased and looks like f0=0.5 and f0=3 aliased and looks like f0=2..

(remark for TA – when I say that they are the same I mean they have the same frequency, not necessary the same phase..)



code:

```
%2.a
n=[0:1:99];
Fs=5;
f01=0.5
x1=sin(2*pi*f01*n/Fs);
subplot (2,2,1)
stem(n,x1);
```

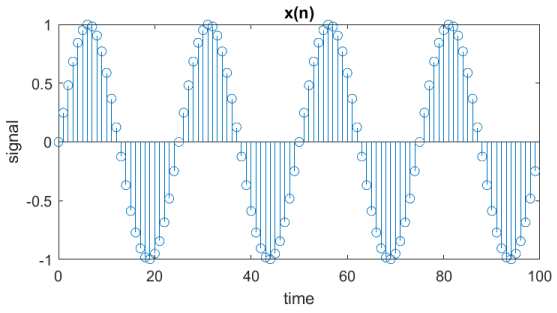
```
xlabel("time")
ylabel("signal")
title("x when f0=0.5")
subplot (2,2,2)
f02=2;
x2=sin(2*pi*f02*n/Fs);
stem(n,x2);
xlabel("time")
ylabel("signal")
title("x when f0=2")
subplot (2,2,3)
f03=3;
x3=sin(2*pi*f03*n/Fs);
stem(n,x3);
xlabel("time")
ylabel("signal")
title("x when f0=3")
subplot (2,2,4)
f04=4.5;
x4=sin(2*pi*f04*n/Fs);
stem(n,x4);
xlabel("time")
ylabel("signal")
title("x when f0=4.5")
```

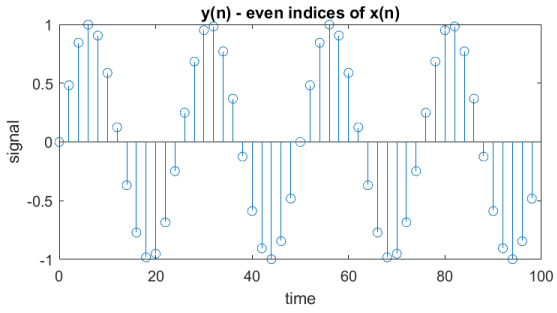
2.b.

Now F0=2, Fs=50.

The signal is: $x(n) = \sin(2\pi \cdot \frac{2}{50} n)$

Therefore: $f_0 = \frac{1}{25} kHz$





For y(n), we just do the assignment: n->0.5n and we get: $f_0 = \frac{1}{50} kHz$

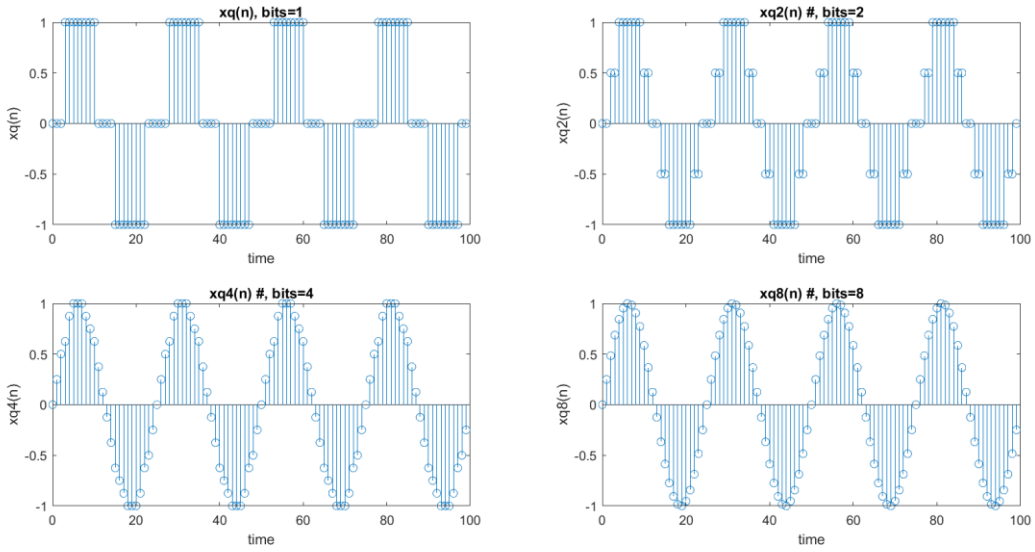
Which also makes sense because the 25th point doesn't show so the signal now repeats itself every 50 points..

Code:

```
n=[0:1:99];
Fs=50;
F0=2
x=sin(2*pi*F0*n/Fs);
figure(22);
subplot(2,2,1);
stem(n,x);
xlabel("time");
```

```
ylabel("signal");
title("x(n)");
hold on;
nEven=[0:2:99];
y=sin(2*pi*F0*nEven/Fs);
subplot(2,2,2);
stem(nEven,y);
xlabel("time");
ylabel("signal");
title("y(n) - even indices of x(n)");
```

2.c.



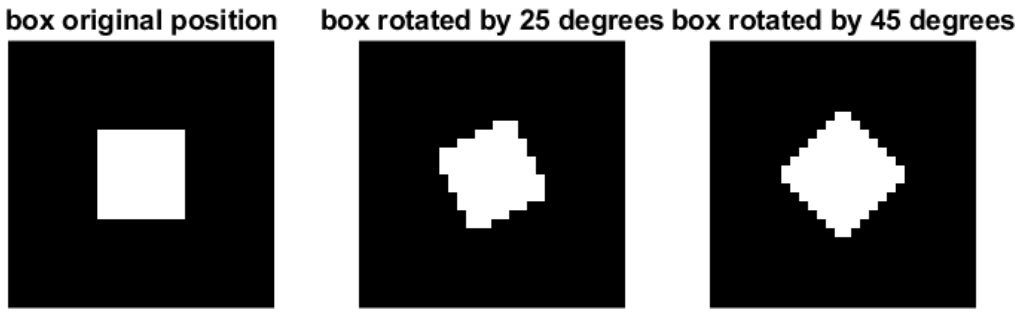
we can see that the more bits we have the more accurate the quantized signal is.

Code:

```
%2.c
bits=1;
L=2^bits; %levels
xq1=round(x*L/2)/(L/2);
figure(23);
subplot(2,2,1);
stem(n,xq1);
xlabel("time");
ylabel("xq(n)");
title("xq(n), bits=1");

bits=2;
L=2^bits; %levels
xq2=round(x*L/2)/(L/2);
subplot(2,2,2);
stem(n,xq2);
xlabel("time");
ylabel("xq2(n)");
title("xq2(n) #, bits=2");

bits=4;
L=2^bits; %levels
xq4=round(x*L/2)/(L/2);
subplot(2,2,3);
```

```
stem(n,xq4);
xlabel("time");
ylabel("xq4(n)");
title("xq4(n) #, bits=4");

bits=8;
L=2^bits; %levels
xq8=round(x*L/2)/(L/2);
subplot(2,2,4);
stem(n,xq8);
xlabel("time");
ylabel("xq8(n)");
title("xq8(n) #, bits=8");
```

3.a. + 3.b.

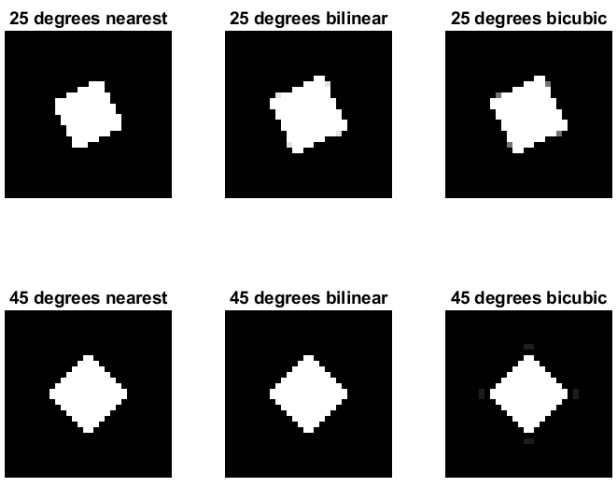


You can clearly see that the rotated images lines have a `zig-zag' shape (distortion/not smooth/blurry/not as sharp as in the original image).

Code:

```
%3.a
img=zeros([30 30]);
img(11:20,11:20)= (255*ones([10,10]));
figure(3132);
subplot (1,3,1)
imshow(img);
title('box original position');
%3.b
img25=imrotate(img,25,'nearest','crop');
subplot(1,3,2);
imshow(img25);
title('box rotated by 25 degrees');
img45=imrotate(img,45,'nearest','crop');
subplot(1,3,3);
imshow(img45);
title('box rotated by 45 degrees');
```

3.c.



We can see that for 25deg the 'nearest' has best results and for 45deg the 'nearest' and 'bilinear' are better than 'bicubic'. The reason might be that bicubic averages the values while nearest just reorganizes them. The artefacts discussed earlier still appear of course (because of the small number of pixels..)

Code:

```
%3.c
figure(33);
img25_n=imrotate(img,25,'nearest','crop');
subplot(2,3,1);
imshow(img25_n);
title('25 degrees nearest');
img25_b=imrotate(img,25,'bilinear','crop');
subplot(2,3,2);
imshow(img25_b);
title('25 degrees bilinear');
img25_c=imrotate(img,25,'bicubic','crop');
subplot(2,3,3);
imshow(img25_c);
title('25 degrees bicubic');
img45_n=imrotate(img,45,'nearest','crop');
subplot(2,3,4);
imshow(img45_n);
title('45 degrees nearest');
img45_b=imrotate(img,45,'bilinear','crop');
subplot(2,3,5);
imshow(img45_b);
title('45 degrees bilinear');
img45_c=imrotate(img,45,'bicubic','crop');
subplot(2,3,6);
imshow(img45_c);
title('45 degrees bicubic');
```

3.d.

** I sliced the image (zoom-in effect) to see the differences better. I took x2 parameters on the resized images which mean they really doubled their size...

Well the bilinear looks the best but they are all similar in quality (for me).

Code:

```
%3.d
heart=imread("heart.jpg");
figure(34);
subplot(1,4,1);
imshow(heart(250:300,250:300));
title("original");
heart_n=imresize(heart,2,'nearest');
subplot(1,4,2);
imshow(heart_n(500:600,500:620));
title("resized nearest");
heart_b=imresize(heart,2,'bilinear');
subplot(1,4,3);
imshow(heart_b(500:600,500:620));
title("resized bilinear");
heart_c=imresize(heart,2,'bicubic');
subplot(1,4,4);
imshow(heart_c(500:600,500:620));
title("resized bicubic");
```

4.a.

Termolo – the signal I picked is:

$$x(n) = \sin\left(2\pi \frac{440}{11025} n\right)$$

And the modulator signal is:

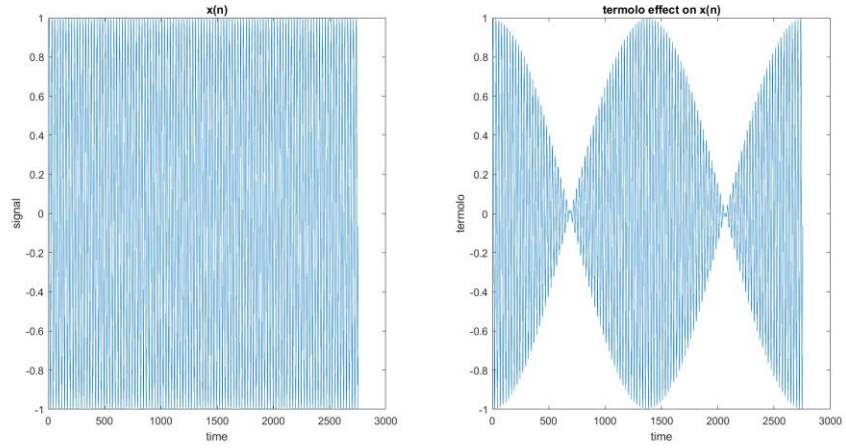
$$m(n) = \cos(2\pi \cdot 4n)$$

I used ‘plot’ instead of ‘stem’ even though it’s a discrete-time signal to make it look better.

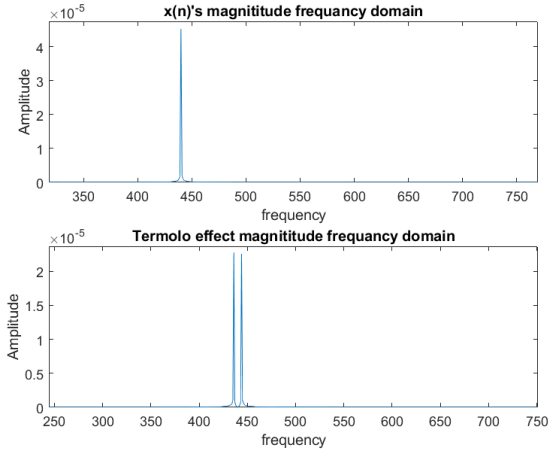
To play the audio I just typed the commands:

```
sound(x);
pause(2);
sound(x.*m);
```

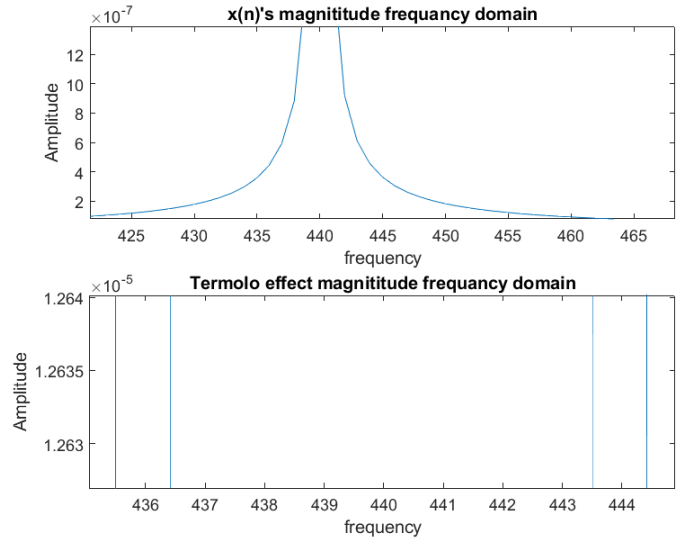
The termolo sounds like it has distortion in the volume.

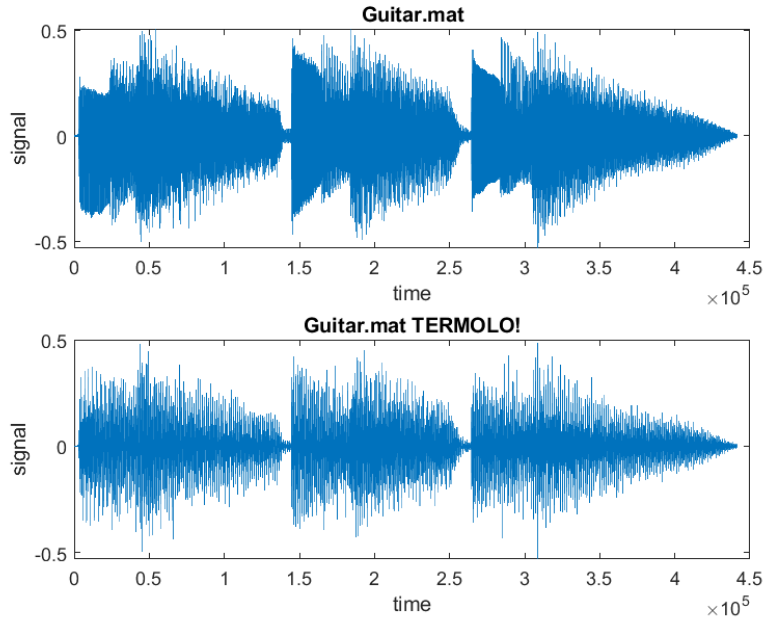


And the frequency domain analysis:



After zooming-in, we can see that the main 440Hz frequency has changed to 440+4 and 440-4 frequencies…



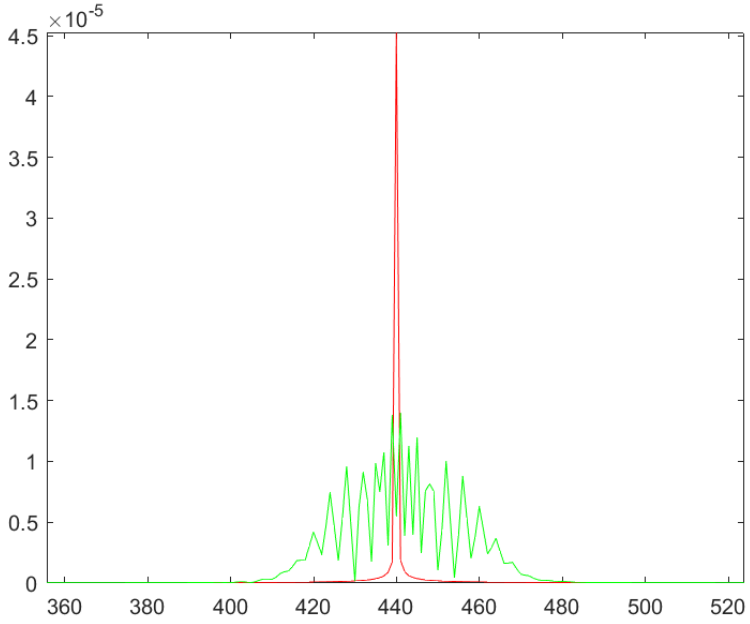


Code:

```
%4.a.1
n=[0:1:(0.25*11025)];
x=sin(2*pi*440*n/11025);
m=(cos(2*pi*n*4/11025));
figure(41)
subplot (1,2,1);
plot(n,x);
xlabel("time");
ylabel('signal');
title("x(n)");
subplot(1,2,2);
plot(n,x.*m);
xlabel('time');
title("termolo effect on x(n)");
ylabel('termolo')
%4.a.2
n=[0:1:(1*11025)];
x=sin(2*pi*440*n/11025);
m=(cos(2*pi*n*4/11025));
sound(x);
pause(2);
sound(x.*m);
%4.a.3
figure(43)
subplot(2,1,1);
X=fft(x);
L=length(n);
D=(abs(X/L));
D=D(1:L/2+1)/L;
f1=11025*(0:L/2)/L;
plot(f1,D);
xlabel('frequency');
title("x(n)'s magnititude frequancy domain");
ylabel('Amplitude');
subplot(2,1,2);
Y=fft(x.*m);
D=(abs(Y/L));
D=D(1:L/2+1)/L;
f2=11025*(0:L/2)/L;
plot(f2,D);
xlabel('frequency');
```

```
title("Termolo effect magnititude frequancy domain");
ylabel('Amplitude');
%4.a.4
load('guitar.mat');
figure(44);
subplot(2,1,1);
plot(y);
xlabel('time');
ylabel('signal');
title('Guitar.mat');
subplot(2,1,2);
n=[0:1:441343];
m=(cos(2*pi*n*4/11025));
m=transpose(m);
gt=y.*m;
plot(gt);
title('Guitar.mat TERMOLO!');
xlabel('time');
ylabel('signal');
sound(y,Fs);
sound(gt,Fs);
```

4.b.



The Vibrato takes the main frequency and ‘mixes’ it to all surrounding frequencies.

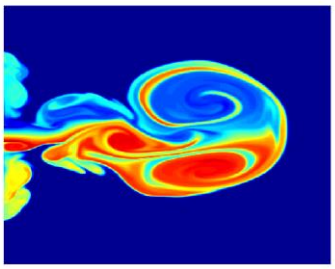
The vibrated sound (y) sounds like a guitar distortion.

Code:

```
%4.b.1
n=[0:1:(1*11025)];
x=sin(2*pi*440*n/11025);
figure(45)
X=fft(x);
L=length(n);
D=(abs(X/L));
D=D(1:L/2+1)/L;
```

```
f1=11025*(0:L/2)/L;
xlabel('frequancy');
title('Magnititude of normal/vibrated x(n)');
ylabel('Amplitude');
plot(f1,D,'red');
hold on;
y=sin(2*pi*n.*(440+(cos(2*pi*n*4/11025)))/11025);
Y=fft(y);
P2=(abs(Y/L));
P2=P2(1:L/2+1)/L;
f2=11025*(0:L/2)/L;
plot(f2,P2,'green');
hold off;
sound(x);
sound(y);
```

5.a



5.b.

Size of X is 400x300 and size of map is 64x3.

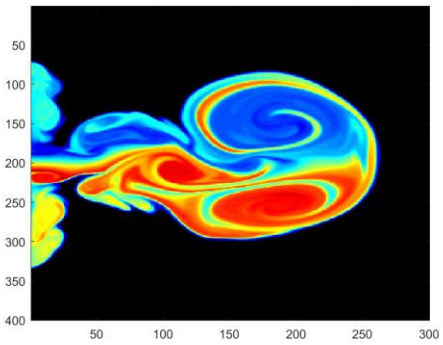
```
>> size(X)

ans =

  400   300

>> size(map)

ans =

   64    3
```

5.c.

X is the image. Map is a pre-defined matlab color map according to which matlab draws the image (X).

5.d.

The first row stands for background color and 0 stands for black. Therefore we want to change map's first row to zeros.



Code:

```
%5.d
map(1,:)=[0 0 0];
imagesc(X);colormap(map);
```