ECE366: Honors Project

Doruk Alp Mutlu (mutludor@msu.edu)

November 29, 2024

Question 1

Question 1 (a)

This is an implementation of Box blur with an $n \times n$ mask where n can be any odd integer. Box blur provides blurring by taking the average of neighboring pixels. Increased effect of box blur can be observed when 3×3 and 11×11 masks are used for Box blur. The example is also in the appended MATLAB output.

```
% Box Blur with an n x n mask
% mat: image as matlab data structure
% n: size of mask, odd positive integer
function im = box_blur(mat, n)
    % init mask
    mask = 1/(n*n) * ones(n,n);
    % return object, borders init to white
    ret = 255 * ones(size(mat));
    [r, c] = size(mat);
    % border adjustment
    t = floor(n / 2) + 1;
    % nested for loop to iterate the whole image
    for i=1:r-(n-1)
        for j=1:c-(n-1)
            temp = mat(i:i+n-1, j:j+n-1);
            temp = sum(temp .* mask, "all");
            ret(i + n - t, j + n - t) = temp;
        end
    end
    im = ret;
end
```

This is an implementation of Gaussian blur with two masks, 3×3 and 5×5. Effect of Gaussian blur is to reduce the detail of the image which can be observed after the Gaussian blur operation with two different masks.

```
% Gaussian Blur with an n x n mask
% mat: image as matlab data structure
% n: size of mask, 3 or 5
function im = gaussian_blur(mat, n)
    mask = [];
    % init mask
    if n==3
        mask = 1/16 * [1 2 1; 2 4 2;  1 2 1];
    elseif n==5
        mask = 1/159 * [2 4 5 4 2; 4 9 12 9 4; 5 12 15
            12 5; 4 9 12 9 4; 2 4 5 4 2];
    end
    % return object, borders init to white
    ret = 255 * ones(size(mat));
    [r, c] = size(mat);
    % border adjustment
    t = floor(n / 2) + 1;
    % nested for loop to iterate the whole image
    for i=1:r-(n-1)
        for j=1:c-(n-1)
            temp = mat(i:i+n-1, j:j+n-1);
            temp = sum(temp .* mask, "all");
            ret(i + n - t, j + n - t) = temp;
        end
    end
    im = ret;
end
```

Laplacian blur is used for edge detection in the image. When the Laplacian blur is applied, we can observe the edges in the image in output.

```
% Laplacian Blur with an 3 x 3 mask
% mat: image as matlab data structure
% ch: choice of mask 1 or 2
function im = laplacian_blur(mat, ch)
    mask = [];
    % init mask
    if ch==1
        mask = [0 1 0; 1 -4 1; 0 1 0];
    elseif ch==2
        mask = [1 1 1; 1 -8 1; 1 1 1];
    end
    n = 3;
    % return object, borders init to white
    ret = 255 * ones(size(mat));
    [r, c] = size(mat);
    % border adjustment
    t = floor(n / 2) + 1;
    % nested for loop to iterate the whole image
    for i=1:r-(n-1)
        for j=1:c-(n-1)
            temp = mat(i:i+n-1, j:j+n-1);
            temp = sum(temp .* mask, "all");
            ret(i + n - t, j + n - t) = temp;
        end
    end
    % normalize the image since there are negative
       values
    ret = ret - min(ret, [],"all");
    ret = ret .* (255 / max(ret, [],"all"));
    im = ret;
end
```

Example applications are appended:

Contents

Question 1(a)

```
lena512 = load("lena512.mat");
square = load("square.mat");
lena512 = lena512.lena512;
square = square.A;
```

Box Blur 3x3, 5x5, 11x11 - Lena

```
blurred_lenna_3 = box_blur(lena512, 3);
blurred_lenna_5 = box_blur(lena512, 5);
blurred_lenna_11 = box_blur(lena512, 11);
tiledlayout(2, 2);
nexttile
imshow(lena512, [])
title("Original Image")
nexttile
imshow(blurred_lenna_3, [])
title("3x3 Box Blur")
nexttile
imshow(blurred_lenna_5, [])
title("5x5 Box Blur")
nexttile
imshow(blurred_lenna_11, [])
title("11x11 Box Blur")
```

Gaussian Blur 3x3,5x5

```
blurred_lenna_gaus_3 = gaussian_blur(lena512, 3);
blurred_lenna_gaus_5 = gaussian_blur(lena512, 5);
tiledlayout(2, 2);
nexttile
imshow(lena512, [])
title("Original Image")
nexttile
imshow(blurred_lenna_gaus_3, [])
title("3x3 Gaussian Blur")
nexttile
imshow(blurred_lenna_gaus_5, [])
title("5x5 Gaussian Blur")
```

Laplacian Blur 3x3, with 2 different kernels

```
lena_lapl_1 = laplacian_blur(lena512, 1);
lena_lapl_2 = laplacian_blur(lena512, 2);
tiledlayout(2, 2);
nexttile
imshow(lena512, [])
title("Original Image")
nexttile
imshow(lena_lapl_1, [])
title("3x3 Laplacian Blur - 1")
nexttile
imshow(lena_lapl_2, [])
title("3x3 Laplacian Blur - 2")
```

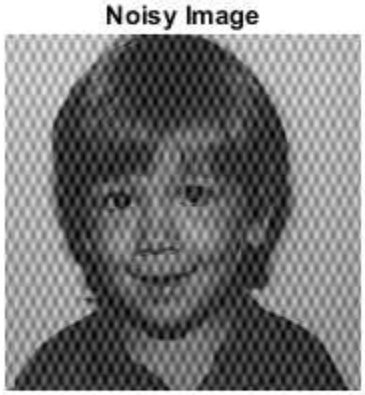
Published with MATLAB® R2023b

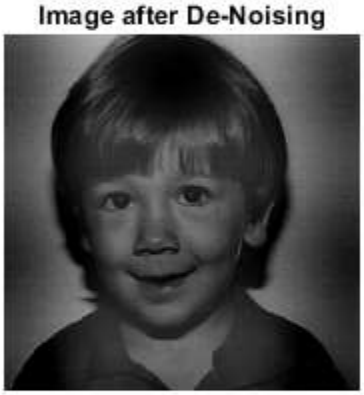
Question 1 (b)

In order to remove the noise, I applied the 2-D FFT in MATLAB and removed the frequeny content which was causing the noise in the image. I determined a threshold for this by trial and error. Then, I applied inverse 2-D FFT to recover the image. In the end, I increased the brightness in the image.

Question 1(b)

```
[img cmap] = imread("boy_noisy.gif", 'Frames', 'all');

s1 = 512*512;
sort_m = zeros(1, s1);
Y = fft2(img);
abs_Y = abs(Y);
k = 1;
for i=1:512
    for j=1:512
        sort_m(k) = abs_Y(i, j);
        % trial and error
        if abs_Y(i, j) > 2000000
            Y(i, j) = 0;
        end
        k = k + 1;
    end
end
% only to investigate the magnitudes
sorted = sort(sort_m, "descend");
Z = ifft2(Y);

tiledlayout(1, 2);
nexttile
imshow(img, cmap)
title("Noisy Image")
nexttile
% increasing brightness in the image
Z = Z + 60;
imshow(Z, cmap)
title("Image after De-Noising")
```





Published with MATLAB® R2023b

Question 1 (c)

After looking at the two resulting images, I can see that phase contains the critical information about the image such as the location of frequency in the image. On the other, when the magnitude is set to 1, we can still see the edges and some of the details about the image as the phase information is still stored.

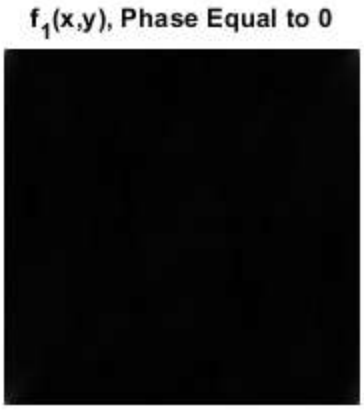
Contents

Question 1(c)

```
lena512 = load("lena512.mat");
lena512 = lena512.lena512;

Y = fft2(lena512);
```

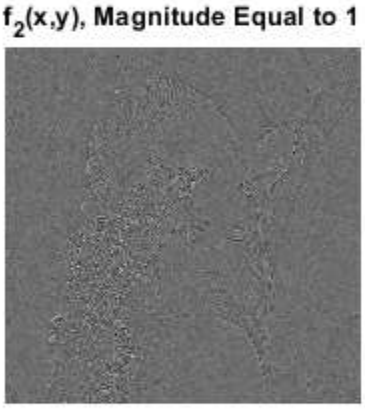
set the phase equal to 0

```
Y1 = abs(Y);

img1 = ifft2(Y1);
% normalizing the values to 0 - 255
img1 = img1 - min(img1, [],"all");
img1 = img1 .* (255 / max(img1, [],"all"));

tiledlayout(1, 2)
nexttile
imshow(lena512, [])
title("Original Image")
nexttile
imshow(img1, [])
title("f_{1}(x,y), Phase Equal to 0")
```



set the magnitude equal to 1

```
Y2 = Y;
[r, c] = size(Y2);
for i=1:r
    for j=1:c
        Y2(i, j) = Y2(i, j) / abs(Y2(i, j));
    end
end

img2 = ifft2(Y2);
% normalizing the values to 0 - 255
img2 = img2 - min(img2, [],"all");
img2 = img2 .* (255 / max(img2, [],"all"));


tiledlayout(1, 2)
nexttile
imshow(lena512, [])
title("Original Image")
nexttile
imshow(img2, [])
title("f_{2}(x,y), Magnitude Equal to 1")
```



Published with MATLAB® R2023b

Question 2

The function *ttdecode* is as follows:

```
% ttdecode
% frq: discrete frequencies for touch-tone signals
% sig: touch-tone signal, 1x7600
function num=ttdecode(frq, sig)
    phone_no = zeros(1, 7);
    tones = zeros(7, 1000);
    start = 1;

    for i=1:7
        tones(i, :) = sig(start : start+999);
        start = start + 1100;
    end

    for r = 1:7
        r_ind = 0;
        c_ind = 0;
        sig = tones(r, :);
        Y_sig = abs(fftshift(fft(sig, 2048))).^2;
        Y_sig_pos = Y_sig(1025:end);
        % col value is the highest freq
        % find the index of the two strongest
        % frequencies
        % assign the larger index to col
        % smaller one to row
        [max_1, max_1_ind] = max(Y_sig_pos);
        Y_sig_pos(max_1_ind) = 0;
        max_1_ind = max_1_ind * 2 * pi / 2048; % scale
            index

        [max_2, max_2_ind] = max(Y_sig_pos);
        max_2_ind = max_2_ind * 2 * pi / 2048; % scale
            index
        if max_2_ind >  max_1_ind
            c_ind = max_2_ind;
            r_ind = max_1_ind;
        else
            c_ind = max_1_ind;
            r_ind = max_2_ind;
        end
        for j = 1:10
            % can do numerical approximation since
```

```
            % none of the r, c values are 1% within
                any other value
            if (((frq(j, 1) * 99/100)<r_ind) && (r_ind
                < (frq(j, 1) * 101/100)))...
                    && (((frq(j, 2) * 99/100)<c_ind)
                        && (c_ind < (frq(j, 2) *
                        101/100)))
                phone_no(r) = j-1;%0 vs 1 indexing
                break;
            end
        end
    end
    num = phone_no;
end
```

The function *ttdecodehard* is as follows:

```
% ttdecode_hard
% frq: discrete frequencies for touch-tone signals
% sig: touch-tone signal, length unknown
function num=ttdecode_hard(frq, sig)
    phone_no = zeros(1, 7);
    tones = {}; % using a cell since length of a
       single tone is unknown
    start = 1;
    seen = 0;
    i = 1;
    %skip the zeros at the beginning
    while i < length(sig)
        if sig(i) == 0
            i = i + 1;
        else
            break
        end
    end
    sig = sig(i:end);
    j = 1;
    while j < length(sig)
        % skip zeros if not continuous
        % do not need to check for out of bounds
           indexing
        % since the start and end of signals are non-
           zero tones
        if sig(j)==0 && sig(j-1)~=0 && sig(j+1)~=0
            j=j+1;
        end
        % rest of the logic for detectinf
```

```
    if sig(j)~=0 && seen
        seen = 0;
        start = j;
        j = j + 100;
    elseif sig(j)~=0 && ~seen
        j = j + 1;
    elseif sig(j)==0 && seen
        j = j + 1;
    else
        tones = [tones; sig(start:j-1)];
        seen = 1;
        j = j + 100;
    end
end
% for the last digit
tones = [tones; sig(start:end)];

for r = 1:7
    r_ind = 0;
    c_ind = 0;
    sig = tones{r};
    Y_sig = abs(fftshift(fft(sig, 2048))).^2;
    Y_sig_pos = Y_sig(1025:end);
    % col value is the highest freq
    % find the index of the two strongest
    % frequencies
    % assign the larger index to col
    % smaller one to row
    [max_1, max_1_ind] = max(Y_sig_pos);
    Y_sig_pos(max_1_ind) = 0;
    max_1_ind = max_1_ind * 2 * pi / 2048; % scale
        index

    [max_2, max_2_ind] = max(Y_sig_pos);
    max_2_ind = max_2_ind * 2 * pi / 2048; % scale
        index
    if max_2_ind >  max_1_ind
        c_ind = max_2_ind;
        r_ind = max_1_ind;
    else
        c_ind = max_1_ind;
        r_ind = max_2_ind;
    end
    for j = 1:10
        % can do numerical approximation since
        % none of the r, c values are 1% within
```

```
                any other value
            if (((frq(j, 1) * 99/100)<r_ind) && (r_ind
                < (frq(j, 1) * 101/100)))...
                    && (((frq(j, 2) * 99/100)<c_ind)
                        && (c_ind < (frq(j, 2) *
                        101/100)))
                phone_no(r) = j-1; %0 vs 1 indexing
                break;
            end
        end
    end
    num = phone_no;
end
```

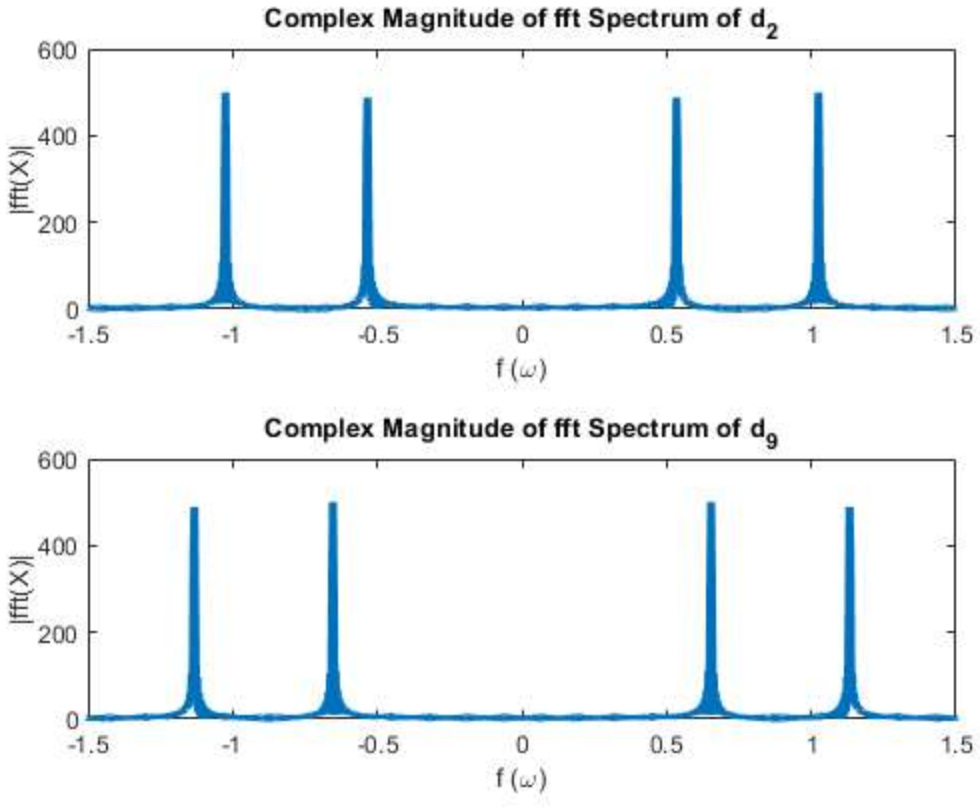
Contents

Question 2 (a)

```
fs = 8192;

n = 0:999;

% Frequencies from Table 1
frq = [
    0.7217, 1.0247;
    0.5346, 0.9273;
    0.5346, 1.0247;
    0.5346, 1.1328;
    0.5906, 0.9273;
    0.5906, 1.0247;
    0.5906, 1.1328;
    0.6535, 0.9273;
    0.6535, 1.0247;
    0.6535, 1.1328
];

digits = ones(10, 1000);

% Signals for digits 0 to 9
for i = 1:10
    % row and column freq's
    omega_r = frq(i, 1);
    omega_c = frq(i, 2);
    % combination of two sinusoid
    digits(i, :) = sin(omega_r*n) + sin(omega_c*n);
end
d0 = digits(1, :);
d1 = digits(2, :);
d2 = digits(3, :);
d3 = digits(4, :);
d4 = digits(5, :);
d5 = digits(6, :);
d6 = digits(7, :);
d7 = digits(8, :);
d8 = digits(9, :);
d9 = digits(10, :);
% sound for d2
% sound(d2, fs);
```

Question 2 (b)

```
Fs = 8192;
T = 1/Fs;
L = 1000;
t = (0:L-1)*T;
omega = 2*pi*(1/2048)*(0:2047);

tiledlayout(2,1);
nexttile
Y_d2 = fftshift(fft(d2, 2048));
plot(-pi:(2*pi/2048):pi-(2*pi/2048), abs(Y_d2),"LineWidth",3)
title("Complex Magnitude of fft Spectrum of d_2")
xlabel("f (\omega)")
ylabel("|fft(X)|")
xlim([-1.5 1.5])

nexttile
Y_d9 = fftshift(fft(d9, 2048));
plot(-pi:(2*pi/2048):pi-(2*pi/2048), abs(Y_d9),"LineWidth",3)
title("Complex Magnitude of fft Spectrum of d_9")
xlabel("f (\omega)")
ylabel("|fft(X)|")
xlim([-1.5 1.5])
```



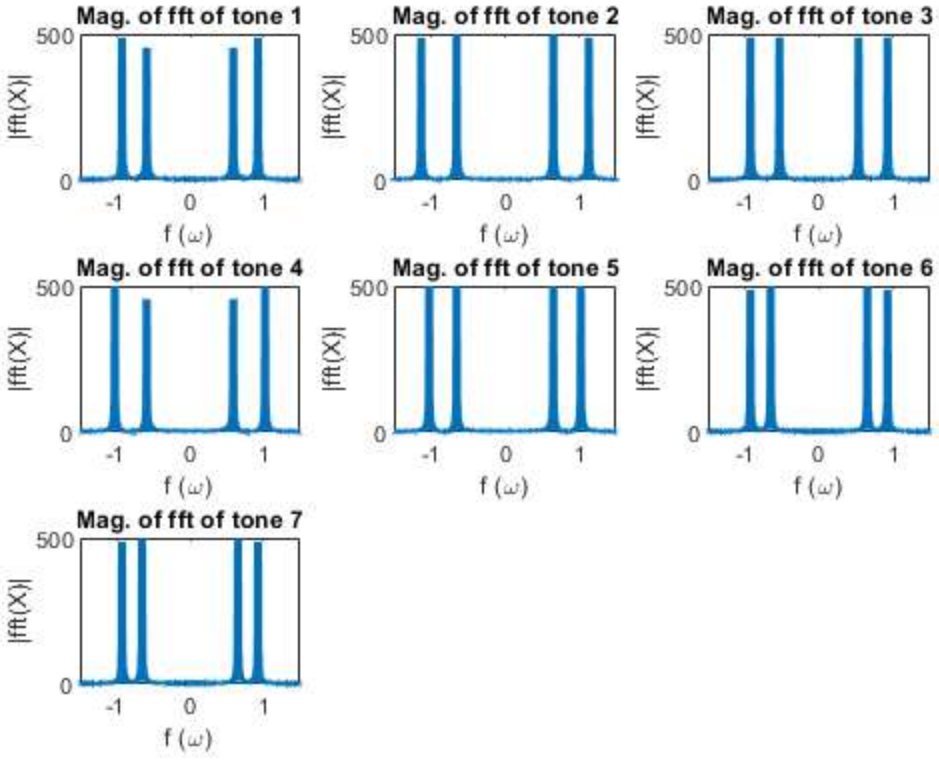
Question 2 (c)

```
space = zeros(1, 100);
% 7 digits --> 303 68 34
```

```
phone = [d3 space d0 space d3 space d6 space d8 space d3 space d4];
sound(phone, fs)
```

Question 2 (d) - x_1

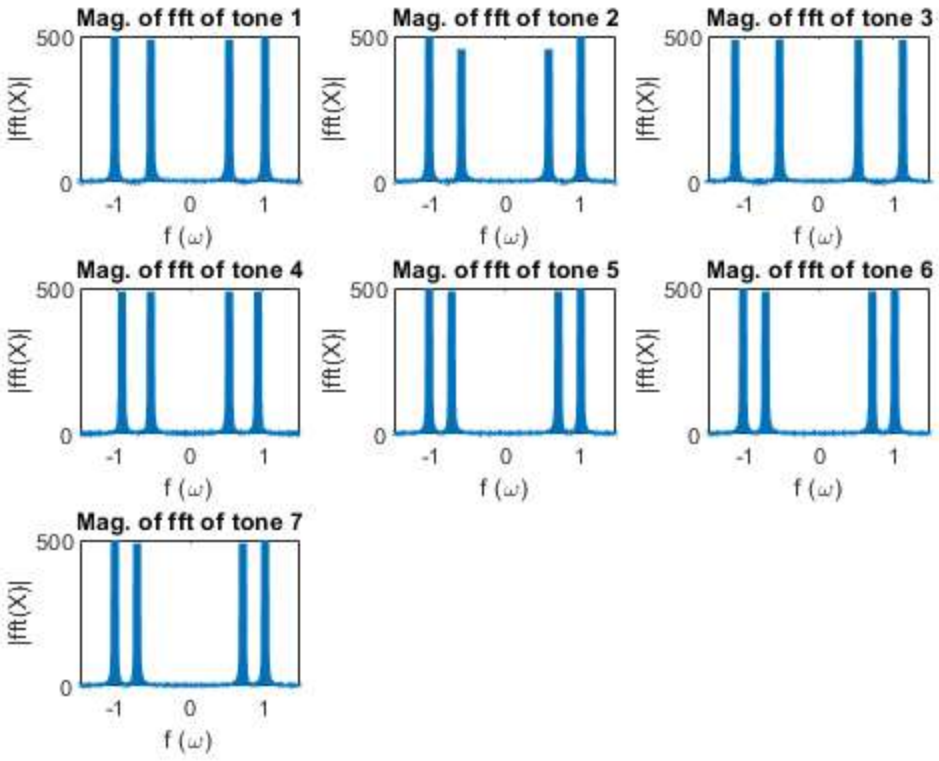
```
touch = load("touch.mat");
x1 = touch.x1;
x2 = touch.x2;
% for x1
% separate the signal into 7 digits
x1digs = zeros(7, 1000);
start = 1;
for i=1:7
    x1digs(i, :) = x1(start : start+999);
    start = start + 1100;
end

for r = 1:7
    subplot(3,3,r);
    sig = x1digs(r, :);
    Y_sig = fftshift(fft(sig, 2048));
    plot(-pi:(2*pi/2048):pi-(2*pi/2048), abs(Y_sig),"LineWidth",3)
    title("Mag. of fft of tone " + r)
    xlabel("f (\omega)")
    ylabel("|fft(X)|")
    xlim([-1.5 1.5])
end

%phone number 491 58 77
```



Question 2 (d) - x_2

separate the signal into 7 digits

```
x2digs = zeros(7, 1000);
start = 1;
for i=1:7
    x2digs(i, :) = x2(start : start+999);
    start = start + 1100;
end

for r = 1:7
    subplot(3,3,r);
    sig = x2digs(r, :);
    Y_sig = fftshift(fft(sig, 2048));
    plot(-pi:(2*pi/2048):pi-(2*pi/2048), abs(Y_sig),"LineWidth",3)
    title("Mag. of fft of tone " + r)
    xlabel("f (\omega)")
    ylabel("|fft(X)|")
    xlim([-1.5 1.5])
end

%phone number 253 10 00
```



Question 2 (e)

```
x_1_phone_no = ttdecode(frq, x1)
x_2_phone_no = ttdecode(frq, x2)
```

```
x_1_phone_no =

     4     9     1     5     8     7     7


x_2_phone_no =

     2     5     3     1     0     0     0
```

Question 2 (f)

```
% Question about index 1101

hardx1 = touch.hardx1;
hardx2 = touch.hardx2;

hardx_1_phone_no_ = ttdecode_hard(frq, hardx1)
hardx_2_phone_no = ttdecode_hard(frq, hardx2)
```

```
hardx_1_phone_no_ =

     4     9     1     5     8     7     7


hardx_2_phone_no =

     2     5     3     1     0     0     0
```

Published with MATLAB® R2023b