# Signal processing report

## By Fredrik Sandhei (fsa011)

Side note: To organize the input images, I put them in a separate folder “static”. That is why there is a second directory in each image calling.

## Lecture 1

### Exercise 1

%Assignment 1

%Create two matrices, a and b. a should be 3x4, and b 4x4.

clc, clear

a = [1,2,3,4;

5,6,7,8;

9,10,11,12];

b = [1,2,3,4;

5,6,7,8;

9,10,11,12;

13,14,15,16];

disp('a = ')

disp(a)

disp('b = ')

disp(b)

out = matrixOperations(a,b);

disp('out = ')

disp(out)

diag\_out = diag(out);

%Returning number of rows and columns

[rows, cols] = size(out);

%replacing the diagonal elements of out with 1

for i = 1:rows

for j = 1:cols

if i == j

out(i,j) = 1;

end

end

end

disp('changed diagonal elements ')

disp('of out to 1: ')

disp(out)

outAdd = matrixAdd(a,b);

disp('outAdd = ')

disp(outAdd)

%Function that multiplies a with 10 and adds the

%multiplication of a and b to that value.

%Results in a 3x4 matrix, which is saved to out.

function out = matrixOperations(a,b)

out = 10 \* a + (a\*b);

end

%Advanced problem: Create a function that takes two matrices of different

%sizes and adds them assuming the missing elements or appended elements

%are zeros.

function out = matrixAdd(a,b)

%Number of rows and columns

[rowsa, colsa] = size(a);

%Number of rows and columns

[rowsb, colsb] = size(b);

%New matrix has the maximum number of rows and

%columns based on a and b

rowsOut = max(rowsa,rowsb);

colsOut = max(colsa, colsb);

out = zeros(rowsOut, colsOut);

%Loop through the new array and

%append every element from a

for i = 1:rowsa

for j = 1:colsa

out(i,j) = a(i,j);

end

end

%Loop through the new array and

%add the elements of b to those elements.

for i = 1:rowsb

for j = 1:colsb

out(i,j) = out(i,j) + b(i,j);

end

end

end

### Exercise 2

%Assignment 2

%Read data from pressure\_temp\_relhumidity\_CO2ppm.txt and

%plot the data with fileRead().

close all

fileRead('pressure\_temp\_relhumidity\_CO2ppm.txt')

%Reading data from .txt file and plotting

%the data with time in hours.

function [] = fileRead(file)

data = dlmread(file);

%1st column pressure data

pressure = data(:,1);

%2nd column temperature data

temperature = data(:,2);

%3rd column humidity data

humidity = data(:,3);

%4th column CO2 level data

gasLevel = data(:,4);

%Equal amount of data in each column

%therefore time has the same length.

time = length(pressure);

figure('name',['Measured data from ', file])

subplot(2,2,1)

plot(0:time-1,pressure), grid on

title('Pressure')

xlabel('Time [hour]')

ylabel('Pressure [KPa]')

subplot(2,2,2)

plot(0:time-1,temperature), grid on

title('Temperature')

xlabel('Time [hour]')

ylabel('Temperature [^\circC]')

subplot(2,2,3)

plot(0:time-1,humidity), grid on

title('Humidity')

xlabel('Time [hour]')

ylabel('Humidity [%]')

subplot(2,2,4)

plot(0:time-1,gasLevel), grid on

title('Gas emission')

xlabel('Time [Hour]')

ylabel('Emission [PPM]')

end

# 

### Exercise 3

%Assignment3

%Make a structure array with 20 movies, with its release year, rating and

%genre from IMDB. Make a function that based on given criterias can give

%out the movies who match the criterias.

%Declaring 20 empty struct array elements

movieStruct(20).Movie = [];

movieStruct(20).Year = [];

movieStruct(20).Rating = [];

movieStruct(20).Genre = [];

%Elements for field "Movie"

movies = {'Hot Fuzz','The Expendables 2','The Expendables','Iron Man 2'...

'Iron Man','Guardians of the Galaxy','Rogue One: A Star Wars Story',...

'Star Wars V - The Empire Strikes Back','Star Wars VI - A New Hope',...

'Star Wars VI - Return of the Jedi','Star Wars I - The Phantom Menace','Star Wars II - Attack of the Clones',...

'Star Wars III - Revenge of the Sith','The Green Mile','The Lord of the Rings: The Fellowship of the Ring',...

'The Lord of the Rings: The Two Towers','The Lord of the Rings: The Return of the King',...

'The Dark Knight', 'The Big Lebowski','The Good, the Bad and the Ugly'};

%Elements for field "Year"

years = [2007, 2012, 2010, 2010, 2008, 2014, 2016, 1980, 1977, 1983, 1999,...

2002, 2005, 1999, 2001, 2002, 2003, 2008, 1998, 1966];

%Elements for field "Rating"

ratings = [7.9, 6.6, 6.5, 7.0, 7.9, 8.1, 7.8, 8.8, 8.6, 8.3, 6.5, 6.6,...

7.6, 8.5, 8.8, 8.7, 8.9, 9.0, 8.2, 8.9];

%Elements for field "Genre"

genres = {'Action/Comedy', 'Action/Adventure/Sci-Fi','Action/Adventure/Sci-Fi','Action/Adventure/Sci-Fi',...

'Action/Adventure/Sci-Fi','Action/Adventure/Sci-Fi','Action/Adventure/Sci-Fi',...

'Action/Adventure/Sci-Fi','Action/Adventure/Sci-Fi','Action/Adventure/Sci-Fi',...

'Action/Adventure/Sci-Fi','Action/Adventure/Sci-Fi','Action/Adventure/Sci-Fi',...

'Crime/Drama/Fantasy','Adventure/Drama/Fantasy','Adventure/Drama/Fantasy',...

'Adventure/Drama/Fantasy','Action/Crime/Thriller','Comedy/Crime','Western'};

%Looping through the struct array and appending each

%element to its appropriate field

for i = 1:length(movieStruct)

movieStruct(i).Movie = movies{i};

movieStruct(i).Year = years(i);

movieStruct(i).Rating = ratings(i);

movieStruct(i).Genre = genres{i};

end

%Function that lists up the movies that match to the

%criterias given.

%Genres to choose from:

%Action

%Comedy

%Sci-Fi

%Adventure

%Drama

%Fantasy

%Western

function [] = Find\_movies\_with\_parameters(year\_search, average\_rating\_search, genre\_search)

%Including the variables from assignment1\_3.m

assignment1\_3;

%Counter variable

movies = 0;

fprintf("year\_search: %d or later, average\_rating: %.2f or above, genre\_search: %s \n",...

year\_search, average\_rating\_search, genre\_search);

%Loop through the structure array

for i = 1:length(movieStruct)

%Checking criterias...

if movieStruct(i).Year >= year\_search...

&& movieStruct(i).Rating >= average\_rating\_search...

&& contains(movieStruct(i).Genre,genre\_search)

%If movie found, increment movies and print out the name

%Of that movie

movies = movies + 1;

fprintf('Matching demands: %s \n', movieStruct(i).Movie);

end

end

%If nothing could be found, then display that..

if movies == 0

disp('Nothing could be found!')

end

end

Example output:

>> Find\_movies\_with\_parameters(2000, 6.0, 'Fantasy')

year\_search: 2000 or later, average\_rating: 6.00 or above, genre\_search: Fantasy

Matching demands: The Lord of the Rings: The Fellowship of the Ring

Matching demands: The Lord of the Rings: The Two Towers

Matching demands: The Lord of the Rings: The Return of the King

>>

## Lecture 2 assignments

### Exercise 1

Below is the plot of different sample ratings of a signal . The highest frequency component is 3Hz and therefore the Nyquist-Shannon sampling frequency is

The sample plot with (1) gives an approximate plot of the original sample, but with slightly lower amplitudes than the original. The oversampled one (2) (higher sampling frequency than 6Hz), gives more correct representation of the signal. The worst is the under sampled (3), which has significantly lower amplitude and wrong phase. Theoretically speaking, the sampling which gave the best representation here would be option (2). It seems it can capture the original signal in its entirety. ‘

close all, clear all

%Nyquist - Shannon sampling frequency: 6Hz >= 3Hz

dt1 = 1/100; %Original sampling rate

dt2 = 1/6; %Nyquist sampling rate

dt3 = 1/50; %Higher sampling than Nyquist

dt4 = 1/2; %Less sampling than Nyquist

st = 0;

et = 8;

t1 = st:dt1:et;

t2 = st:dt2:et;

t3 = st:dt3:et;

t4 = st:dt4:et;

%Original signal

y1 = 4\*sin(2\*pi\*t1)-2\*cos(6\*pi\*t1)-3\*sin(4\*pi\*t1);

%Nyquist sample signal

y2 = 4\*sin(2\*pi\*t2)-2\*cos(6\*pi\*t2)-3\*sin(4\*pi\*t2);

%Oversampled signal

y3 = 4\*sin(2\*pi\*t3)-2\*cos(6\*pi\*t3)-3\*sin(4\*pi\*t3);

%Undersampled signal

y4 = 4\*sin(2\*pi\*t4)-2\*cos(6\*pi\*t4)-3\*sin(4\*pi\*t4);

subplot(2,2,1)

plot(t1,y1), grid on, title('Original signal')

subplot(2,2,2)

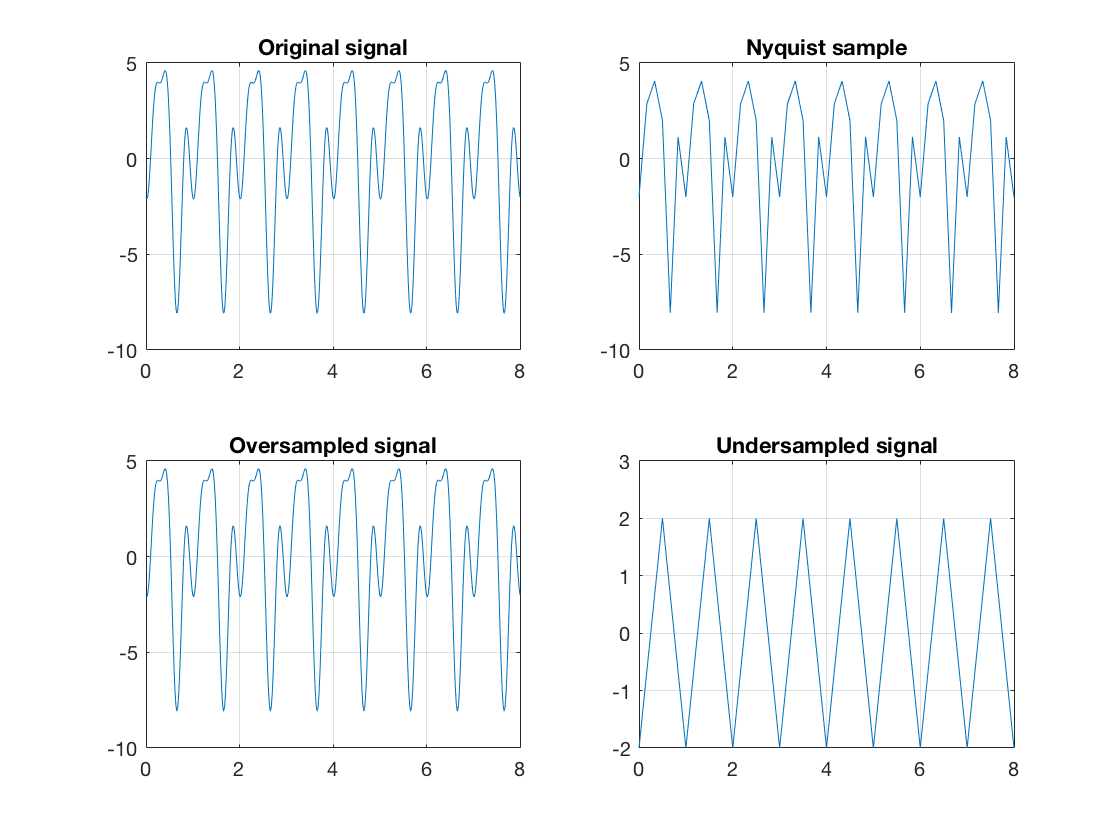
plot(t2,y2), grid on, title('Nyquist sample')

subplot(2,2,3)

plot(t3,y3), grid on, title('Oversampled signal')

subplot(2,2,4)

plot(t4,y4), grid on, title('Undersampled signal')



### Exercise 2

Like exercise 1, only with a more complex signal,

.

Here the Nyquist-Shannon sampling frequency is

The Nyquist-Shannon sample (1) makes a rather good representation of the original signal. It gets its shape and with some noise reduction, but with a reduction in amplitude and different phase (as it doesn’t pick up every signal value during the sampling). The overall shape is pretty good. The oversampled one (2) captures most of the original signal, but perhaps it is not so useful to keep all the noise from it. The under sampled (2) has the shape, but with a phase shift and somewhat lower amplitude. It has however a lot of the original noise taken away from it.

All in all, with capturing the original signal, I would say that option 2) looks the best. It might not be however possible to get the exact signal, as it has a lot of noise added to it.

%Exercise 2

%Nyquist-Shannon sampling frequency: 2\*12Hz = 24Hz >= 12Hz;

dt = 1/200; % Sampling rate

dt1 = 1/24; %Nyquist-Shannon sampling rate, double the samples.

dt2 = 1/100; %Oversampling

dt3 = 1/5; %Undersampling

st = 0; % Start time

et = 8; % End time

t = st:dt:et;

t1 = st:dt1:et;

t2 = st:dt2:et;

t3 = st:dt3:et;

%Original signal

y = 5\*sin(24\*pi\*t)+1\*cos(12\*pi\*t)-3\*sin(6\*pi\*t)+2\*cos(8\*pi\*t);

%Nyquist sample signal

y1 = 5\*sin(24\*pi\*t1)+1\*cos(12\*pi\*t1)-3\*sin(6\*pi\*t1)+2\*cos(8\*pi\*t1);

%Oversampled signal

y2 = 5\*sin(24\*pi\*t2)+1\*cos(12\*pi\*t2)-3\*sin(6\*pi\*t2)+2\*cos(8\*pi\*t2);

%Undersampled signal

y3 = 5\*sin(24\*pi\*t3)+1\*cos(12\*pi\*t3)-3\*sin(6\*pi\*t3)+2\*cos(8\*pi\*t3);

subplot(2,2,1)

plot(t,y), grid on, title('Original signal')

axis tight

subplot(2,2,2)

plot(t1,y1), grid on, title('Nyquist-Shannon signal')

axis tight

subplot(2,2,3)

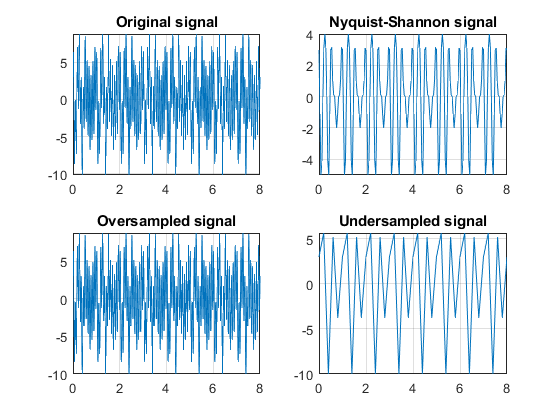
plot(t2,y2), grid on, title('Oversampled signal')

axis tight

subplot(2,2,4)

plot(t3,y3), grid on, title('Undersampled signal')

axis tight



### Exercise 3

The fast Fourier transform (fft) is an algorithm which computes the discrete Fourier transform (dft) of a set of samples, only in a more efficient matter. With a data size ***n***, the computation of dft takes time, while computing a fft takes time. What this means for the fft, is that for a data set of size ***n***, the algorithm must execute amount of operations. The input of a dft is a complex signal – a signal with a real and imaginary component. For computing dft, an amount of complex multiplications are needed.

The fft has a lower runtime than the dft and thus a higher efficiency rate. It is worth mentioning that the fft is not its own operation, but a family of algorithms whose purpose is to calculate the discrete Fourier transform. Examples of fft’s are the Cooley-Turkey algorithm and the Sande-Tukey algorithm.

### Exercise 4

a)

Below are four signals y1, y2, y3 and y4 plotted alongside with their Fourier transformation. For these signals, who each have ***n*** number of frequency components, there seems to be a relationship between frequency components and number of peaks in the fft. In other words, for ***n*** frequency components, the Fourier transformation has 2***n*** number of peaks.

b)

A fourth signal, y4 was also considered. The Fourier transform of this random signal gives several peaks. This may be explained through the rand() function, which assigns pseudo random numbers to every index. The output results in a function whose amplitudes describe the amount of several frequency components represented in y4, and thus giving several peaks in the Fourier transform.

%Exercise 4

close all

clear

dt = 1/200; %sampling rate

st = 0; %start time

et = 8; %end time

t = st:dt:et; %time steps (vector)

y1 = 2\*sin(10\*pi\*t);

y2 = 2\*sin(10\*pi\*t) + 3\*sin(40\*pi\*t);

y3 = 5\*sin(10\*pi\*t)+2\*cos(40\*pi\*t)+3\*sin(80\*pi\*t);

y4 = rand(1,200);

%Takes the Fourier transform of the signal

%and shifts the zero frequency component to origin.

F1 = fftshift(fft(y1));

F2 = fftshift(fft(y2));

F3 = fftshift(fft(y3));

F4 = fftshift(fft(y4));

%Amount of peaks in the Fft - plot is

%relatable with amount of frequency components \* 2

%Amount of peaks in the Fft plot of y4 is at the most equal

%to the length of y4, as there might be values that are similar

%within the vector.

figure(1)

subplot(3,2,1)

plot(t,y1), title('y1(t)')

xlabel('t [s]')

subplot(3,2,2)

stem(abs(F1)), title('Fourier trans. of y1'), grid on

xlabel('f [s^{-1}]')

subplot(3,2,3)

plot(t,y2), title('y2(t)')

xlabel('t [s]')

subplot(3,2,4)

stem((abs(F2))), title('Fourier trans. of y2'), grid on

xlabel('f [s^{-1}]')

subplot(3,2,5)

plot(t,y3), title('y3(t)')

xlabel('t [s]')

subplot(3,2,6)

stem(abs(F3)), title('Fourier trans. of y3'), grid on

xlabel('f [s^{-1}]')

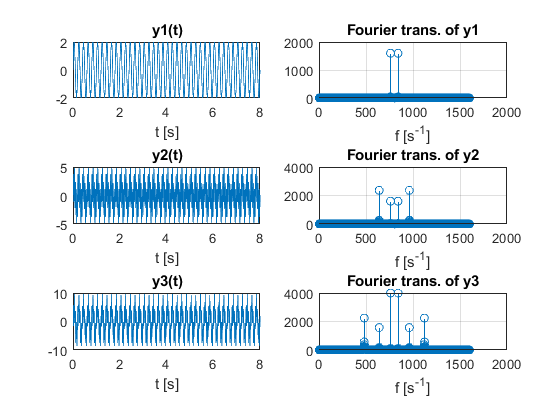
figure(2)

subplot(1,2,1)

plot(y4), title('y4')

subplot(1,2,2)

stem(abs(F4)), title('Fourier trans. of y4'), grid on



# 

## Lecture 3

### Exercise 1

%Exercise 1

%Read 'test.jpg' and do a histographic equalization on it

close all, clear all

I = imread('Static/test.jpg');

R = I(:,:,1); %Red

G = I(:,:,2); %Green

B = I(:,:,3); %Blue

%Square root of green channel

G = double(G);

G = sqrt(G);

%Normalizing channel

G = (G-min(min(G))).\*((255-0)./(max(max(G))-min(min(G)))) + 0;

%Typecast it back into uint8

G = uint8(G);

%Square of blue channel

B = double(B);

B = B.^2;

%Normalizing channel

B = (B-min(min(B))).\*((255-0)./(max(max(B))-min(min(B)))) + 0;

%Typecast it back into uint8

B = uint8(B);

%Concatenating channels to make new image

newI = cat(3, R, G, B);

%Image plot

figure(1)

subplot(1,2,1)

imshow(I), title('Original')

subplot(1,2,2)

imshow(newI), title('Modified')

%Histography plot

figure(2)

subplot(3,1,1)

imhist(R), title('Histography, red channel')

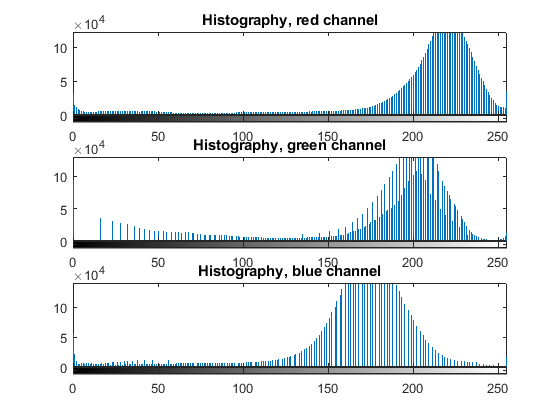
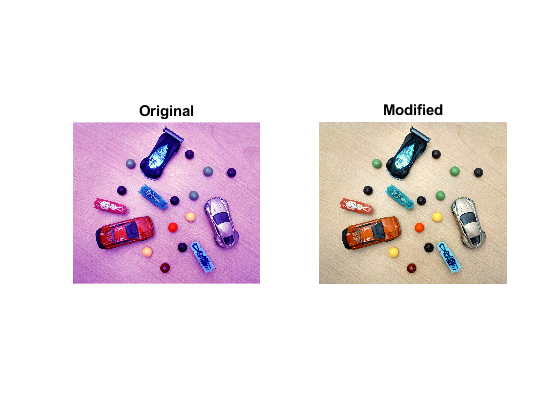
subplot(3,1,2)

imhist(G), title('Histography, green channel')

subplot(3,1,3)

imhist(B), title('Histography, blue channel')

#### Output



### Exercise 2

The convolution of two matrices/vectors u and v represents the area of overlap under the points as v slides across u.

Cross correlation of two matrices/vector u and v checks for any correlation between u and v. That is, can u be expressed by v?

Note when the vectors u and v are symmetric, the convolution and cross correlation of these vectors will be the same.

S2 is a periodic sample which repeats its pattern after 100 values, while S1 a logarithmic sample, increasing for every sample.

As seen in the convolution plot, the convolution climbs almost linearly, until it reaches every 100th point, where it takes a small dip (due to the periodic characteristics of S2). After the third period of S2, the convolution descends with a rather identical shape to the ascent, with small increases after every 100th point.

In the cross-correlation it seems that ascent and descent are equal in shape, only inverted, hence why the center of the cross correlation is not at x = 300.

%Exercise 2

close all, clear all

S1 = log(1:300); %Signal 1

S1 = S1/max(S1); %Make maximal value 1

S2 = [100:-1:1, 100:-1:1, 100:-1:1]; %Signal 2

S2 = S2/max(S2); %Make maximal value 1

C = conv(S1,S2); %Convolute signals

C = C/max(C); %Make maximal value 1

XC = xcorr(S1,S2); %Cross Correlation

XC = XC/max(XC); %Make Maximal value 1

A1 = xcorr(S1,S1); %Auto correlation

A1 = A1/max(A1); %Make maximal value 1

A2 = xcorr(S2,S2); %Auto correlation

A2 = A2/max(A2); %Make maximal value 1

%Plot signals

subplot(3,2,1)

plot(S1), title('Sample S1'), axis tight

subplot(3,2,2)

plot(S2), title('Sample S2'), axis tight

subplot(3,2,3)

plot(C), title('Convolution'), axis tight

subplot(3,2,4)

plot(XC), title('Cross correlation'), axis tight

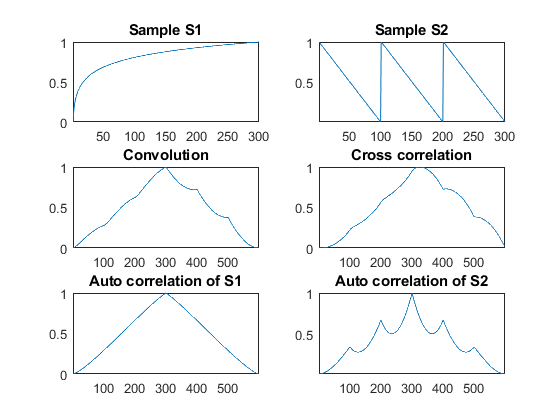
subplot(3,2,5)

plot(A1), title('Auto correlation of S1'), axis tight

subplot(3,2,6)

plot(A2), title('Auto correlation of S2'), axis tight

#### Output



### Exercise 3

Below are different filter options tried on a .jpg image with convoluting the original image and a matrix. Essentially were three kinds of filters added: A low pass filter (h\_lp, h\_gauss), a high pass (h\_hori, h\_veri, h\_both).

%Exercise 2

close all, clear all

image = imread('Static/test.jpg');

h\_lp = [1/9,1/9,1/9;

        1/9,1/9,1/9;

        1/9,1/9,1/9];

h\_hori = [-1 2 1;

          -1 2 1;

          -1 2 1];

h\_vert = [-1 -1 -1;

           2 2 2;

           -1 -1 -1];

h\_both = [-1 -1 -1;

          -1 8 -1;

          -1 -1 -1];

h\_gauss = [ 0.0113    0.0838    0.0113;

            0.0838    0.6193    0.0838;

            0.0113    0.0838    0.0113];

%imageFilter = conv2(double(image), h\_lp, 'same');

imageHlp= imfilter(image,h\_lp, 'conv');

imageHhori = imfilter(image,h\_hori,'conv');

imageHvert = imfilter(image,h\_vert,'conv');

imageHboth = imfilter(image,h\_both, 'conv');

imageHgauss = imfilter(image,h\_gauss,'conv');

figure('Name','Different filters applied to lenna.jpg')

subplot(3,2,1)

imshow(image), title('Original')

subplot(3,2,2)

imshow(imageHlp), title('Hlp filter')

subplot(3,2,3)

imshow(imageHhori), title('Horisontal filter')

subplot(3,2,4)

imshow(imageHvert), title('Vertical filter')

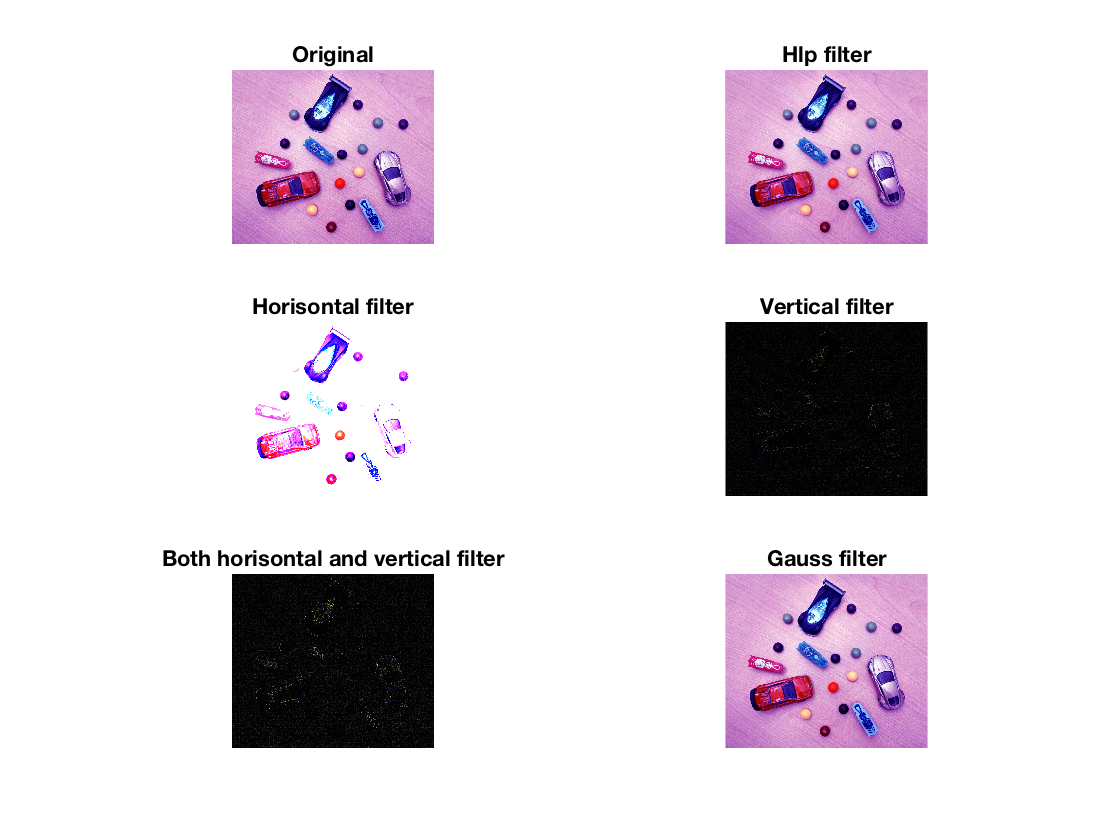
subplot(3,2,5)

imshow(imageHboth), title('Both horisontal and vertical filter')

subplot(3,2,6)

imshow(imageHgauss), title('Gauss filter')

#### Output



### Exercise 4

Below are three different filter options attempted to filter out the noise gained from the ‘salt & pepper’ effect of imnoise(), with a noise density of 0.5. With the averaging filter, I tried with two different matrix sizes, 3x3 and 5x5. The second filter is the Gaussian with standard deviation of 2 (instead of the default at 0.5). The third filter option is median filtering of each of the RGB channels.

From observing, I noticed that the average filter with a 5x5 matrix filtered better than with the 3x3 matrix. Same pattern could be noticed with Gaussian by increasing the standard deviation.

The best result however was obtained with the median filtering.

clc, close all, clear all

I = imread('Static/lenna.png');

noisyI = imnoise(I,'salt & pepper', 0.05);

%Gaussian filtering

%hGaussian = fspecial('gaussian',3,0.5);

%FilterGaussian = imfilter(noisyI, hGaussian);

FilterGaussian = imgaussfilt(noisyI, 2);

%Average filtering

hAverage = fspecial('average',3);

hAverage5 = fspecial('average',5);

FilterAverage = imfilter(noisyI, hAverage);

FilterAverage5 = imfilter(noisyI, hAverage5);

%Median filter

FilterR = uint8(medfilt2(double(noisyI(:,:,1))));%Red channel: I(:,:,1)

FilterG = uint8(medfilt2(double(noisyI(:,:,2))));%Green channel: I(:,:,2)

FilterB = uint8(medfilt2(double(noisyI(:,:,3))));%Blue channel: I(:,:,3)

FilterMedian = cat(3,FilterR, FilterG, FilterB);

subplot(3,2,1)

imshow(I), title('Original')

subplot(3,2,2)

imshow(noisyI), title('Image with noise')

subplot(3,2,3)

imshow(FilterAverage), title('Image filtered with average 3x3')

subplot(3,2,4)

imshow(FilterAverage5), title('Image filtered with average 5x5')

subplot(3,2,5)

imshow(FilterGaussian), title('Image filtered with Gaussian \sigma = 2')

subplot(3,2,6)

imshow(FilterMedian), title('Image filtered with median')



## Lecture 4

### Exercise 1: Code understanding

The code provided is a filtering algorithm, which is essentially applies a low pass filter on an image by grayscaling the image, and then convolute it with an average filter matrix, . This algorithm is equivalent to using MATLAB’s function imfilter:

imfilter(grayIm, h, 'conv');

### Exercise 2

Read the image “test\_img.jpg” and extract each object from it and display it in separate binary images.

%Exercise 2

close all, clear all

%Read image

rgbImg = imread('Static/test\_img.jpg');

%RGB Channels

R = rgbImg(:,:,1);

G = rgbImg(:,:,2);

B = rgbImg(:,:,3);

[imgRows, imgCols, dim] = size(rgbImg);

%Initialize new arrays based on the original

%image's size

newImgCircle = zeros(imgRows, imgCols);

newImgTriangle = zeros(imgRows, imgCols);

newImgStar = zeros(imgRows, imgCols);

newImgRectangle = zeros(imgRows, imgCols);

%Loop through the empty arrays and fill them up

%with logical values depending on color criterias:

%triangle [63, 71, 234]

%rectangle [35, 177, 77]

%circle [237 27 36]

%Star [254 242 0]

for i = 1:imgRows

for j = 1:imgCols

if((R(i,j) > 230) && G(i,j) < 100)

newImgCircle(i,j) = 1;

else

newImgCircle(i,j) = 0;

end

if(B(i,j) > 120)

newImgTriangle(i,j) = 1;

else

newImgTriangle(i,j) = 0;

end

if((R(i,j) > 230) && G(i,j) > 200)

newImgStar(i,j) = 1;

else

newImgStar(i,j) = 0;

end

if(G(i,j) > 150 && (B(i,j) > 50) && (R(i,j) < 50))

newImgRectangle(i,j) = 1;

else

newImgRectangle(i,j) = 0;

end

end

end

%Display images

subplot(3,2,1)

imshow(rgbImg), title('Original')

subplot(3,2,2)

imshow(newImgTriangle), title('Triangle extracted')

subplot(3,2,3)

imshow(newImgCircle), title('Circle extracted')

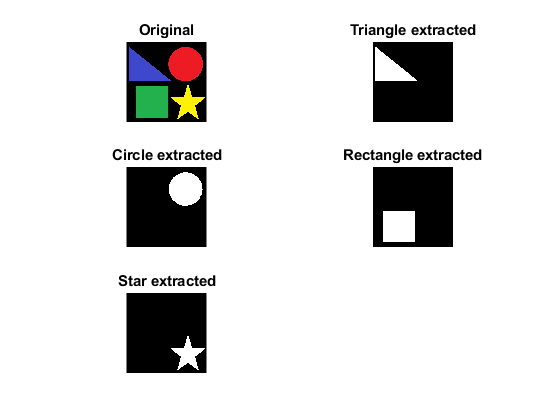
subplot(3,2,4)

imshow(newImgRectangle), title('Rectangle extracted')

subplot(3,2,5)

imshow(newImgStar), title('Star extracted')

#### Output



### Exercise 3

From PowerPoint:

“Read ‘test\_img\_shapes.jpg’ in Matlab and use ‘Image viewer’ tool from ‘Apps’ in Matlab to see the color values for the three different objects. Note these values down and use them for creating three new binary images with each object as shown in the figure.”

%Exercise 3

clear all, close all

%Read image

rgbImg = imread('../Static/test\_img\_shapes.jpg');

%Convert to grayscale

grayImg = rgb2gray(rgbImg);

%Convert grayscale to binary

newImg = imbinarize(grayImg);

%RGB channels

R = rgbImg(:,:,1);

G = rgbImg(:,:,2);

B = rgbImg(:,:,3);

[imgRows, imgCols, dim] = size(rgbImg);

%Initialize arrays for the three image objects

redRect = zeros(imgRows, imgCols);

greenRect = zeros(imgRows, imgCols);

blueCircle = zeros(imgRows, imgCols);

%Loop through the rows and columns of

%the original image and set the indexes if the criterias

%for the different objects match:

%red [237 27 36]

%green [48 180 70]

%blue [38 144 186]

%Note that it is only the objects that are being set, not the

%inscribed objects. This will be inverted later (due to imfill)

for i = 1:imgRows

for j = 1:imgCols

if((R(i,j) > 200) && (G(i,j) < 60) && (B(i,j) > 20))

redRect(i,j) = 1;

else

redRect(i,j) = 0;

end

if((G(i,j) > 170) && B(i,j) < 70)

greenRect(i,j) = 1;

else

greenRect(i,j) = 0;

end

if(G(i,j) > 150 && (B(i,j) > 190) && R(i,j) < 10)

blueCircle(i,j) = 1;

else

blueCircle(i,j) = 0;

end

%if((R(i,j) > 230) && (G(i,j) > 200))

% newImg(i,j) = 1;

% else

% newImg(i,j) = 0;

% end

end

end

%Fill the black holes

redRect = imfill(redRect, 'holes');

greenRect = imfill(greenRect, 'holes');

blueCircle = imfill(blueCircle, 'holes');

%Invert the matrices

redRect = imcomplement(logical(redRect));

greenRect = imcomplement(logical(greenRect));

blueCircle = imcomplement(logical(blueCircle));

subplot(3,2,1)

imshow(rgbImg)

title('Original');

subplot(3,2,2)

imshow(greenRect)

title('Green rectangle');

subplot(3,2,3)

imshow(redRect)

title('Red rectangle');

subplot(3,2,4)

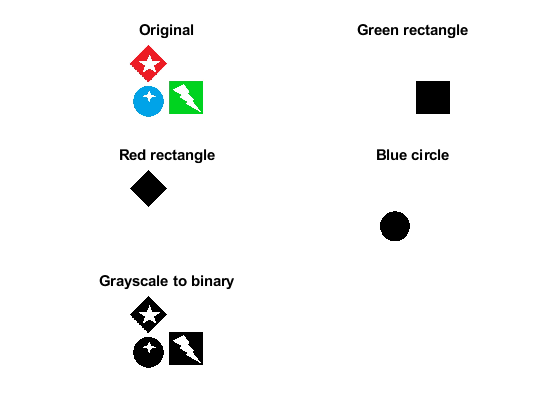
imshow(blueCircle)

title('Blue circle');

subplot(3,2,5)

imshow(newImg), title('Grayscale to binary')

#### Output



### Exercise 4

From PowerPoint:

“Use the example (Color image segmentation) to count and label the objects in ‘test\_img\_shapes.jpg’.”

Solution: I based my code on the example given and removed the unnecessary components, and tweaked the radius in *strel()* to include as much as possible of the rotated square.

%Exercise 4

clear all, close all

%Read image

I = imread('../Static/test\_img\_shapes.jpg');

%Thresholding each channel

Im = double(I)/255;

rmat=Im(:,:,1); %Red channel

gmat=Im(:,:,2); %Green channel

bmat=Im(:,:,3); %Blue channel

levelr = 0.63;

levelg = 0.5;

levelb = 0.4;

i1=im2bw(rmat,levelr);

i2=im2bw(gmat,levelg);

i3=im2bw(bmat,levelb);

Isum = (i1&i2&i3);

%% Complement Image and Fill in holes

Icomp = imcomplement(Isum);

Ifilled = imfill(Icomp,'holes');

%%

se = strel('disk', 0);

Iopenned = imopen(Ifilled,se);

%% Extract features

[labeled,numObjects] = bwlabel(Iopenned,4);

stats = regionprops(labeled,'Eccentricity','Area','BoundingBox');

eccentricities = [stats.Eccentricity];

%% Use feature analysis to count skittles objects

idxOfSkittles = find(eccentricities);

statsDefects = stats(idxOfSkittles);

figure, imshow(I);

hold on;

for idx = 1 : length(idxOfSkittles)

h = rectangle('Position',statsDefects(idx).BoundingBox,'LineWidth',2);

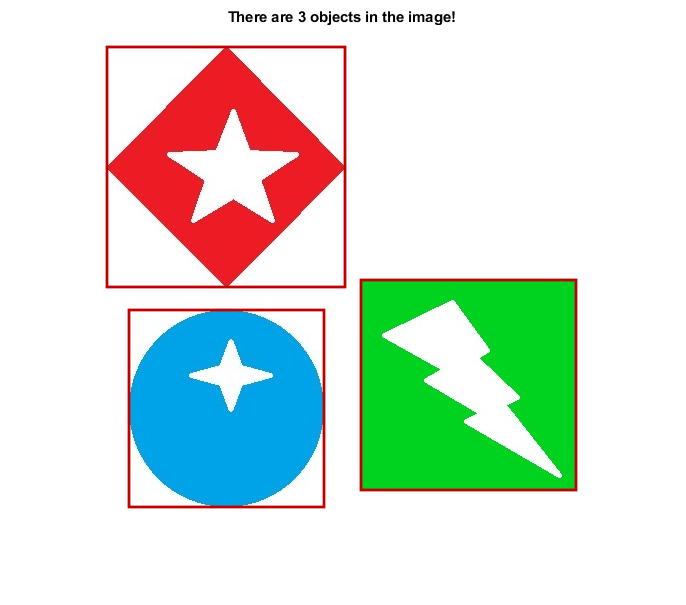
set(h,'EdgeColor',[.75 0 0]);

title(['There are ', num2str(numObjects), ' objects in the image!']);

end

hold off;

#### Output



### Exercise 5

Note: When I plotted the edge detection that I had made versus MATLAB’s imfilter, I see that they are equal, but it seems like they are shifted upwards/downwards. I tried to correct this with a fftshift() but it only seemed to distort the image. I still believe that my attempt of solving this exercise is correct.

Note nr. 2: I couldn’t find any other good high pass filter matrices than the one you had given in one of your lectures, hence why I am using that one. I also tried making my own high pass filtering matrix by subtracting an all pass filter matrix from a low pass filter matrix. Then I ended up with a high filtering matrix, but it didn’t give any satisfactory results, compared to h (in the code below).

%Exercise 5

%Create a program to detect edges, both horizontal

%and vertical edges by using convolution in Fourier domain.

clear, close all

x = 500;

y = 500;

no\_of\_indx = 20; % Size of the blocks

I = zeros(x,y);

for i=1:2\*no\_of\_indx:x

for j=1:2\*no\_of\_indx:y

I(i:i+no\_of\_indx,j:j+no\_of\_indx) = 255;

end

end

[rows, cols] = size(I);

%Highpass filter matrix

h = [-1 -1 -1;

-1 8 -1;

-1 -1 -1];

%hLowPass = fspecial('gaussian',3, 2);

%hAllPass = zeros(size(hLowPass,1),size(hLowPass,2));

%hAllPass(2,2) = 1;

%hHighPass = hAllPass - hLowPass;

ImFilter = imfilter(I, h,'conv');

hFilter = zeros(rows, cols);

for i = 1:size(h,1)

for j = 1:size(h,2)

hFilter(i,j) = h(i,j);

end

end

%take the Fourier transform of image and

%filter, and thereafter multiply them together

%and take inverse Fourier transform of result

fourierI = fft2(I);

fourierH = fft2(hFilter);

filterI = real(ifft2(fourierI .\* fourierH));

%Plot the images for comparison

subplot(2,2,1)

imshow(uint8(I)), title('Original')

subplot(2,2,3)

imshow(uint8(filterI)),title('Convolution with Fourier Transform')

subplot(2,2,4)

imshow(uint8(ImFilter)), title('imfilter()')

#### Output

