Linear filtering with Gaussian filters.

Read and study as well the document "3-linear filtering.pdf" provided, containing what it has been explained in class regarding linear filtering in Matlab.

Gaussian filters

Theory

The 2-dimensional Gaussian function:

$$G_{xy}(x,y) = \exp\left(-\frac{(x^2+y^2)}{2\sigma^2}\right) = \exp\left(-\frac{x^2}{2\sigma^2}\right)\exp\left(-\frac{y^2}{2\sigma^2}\right) = G_x(x)G_y(y)$$

is separable into two 1 dimensional Gaussian functions G_x and G_y.

This means that a 2-dimensional filtering of an image with the 2-dimensional filter G_{xy} can be performed with two 1-dimensional filtering: one in the x-direction with G_x, followed by one in the y-direction with G_y. Filtering with two 1-dim filters is much faster than filtering with 2-dim filters!

Figure 1 below shows examples of: 2-dim filters G_{xy} (top), 1-dim filter G_x (bottom left) and 1-dim filter G_y (bottom right).

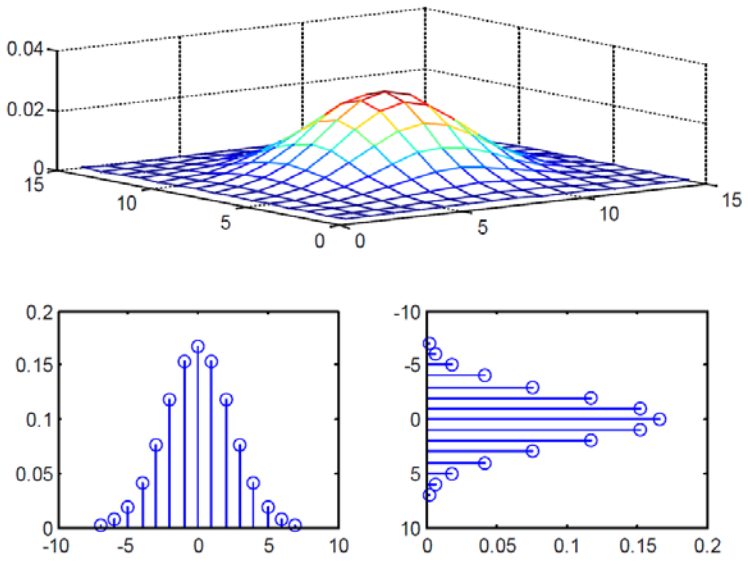
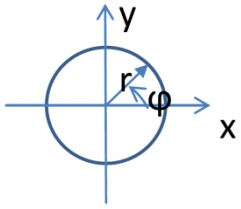


Figure 1. Example of a Gaussian low pass filter. Top: 2-dim, bottom left: 1-dim in the x-direction, bottom right: 1-dim in the y direction.

Moreover, the Gaussian function is *isotropic*. If we rewrite G_{xy} to polar coordinates (r, φ) as follows:

$$G_{xy}(x,y) = \exp\left(-\frac{(x^2+y^2)}{2\sigma^2}\right) = \exp\left(-\frac{r^2}{2\sigma^2}\right) = G_{r\varphi}(r,\varphi)$$



that function is *isotropic* means that its value does not depend on the angle φ but only on the radius r. This, in turn, means that all directions in the image are filtered in the same way.

Linear filtering

Theory

Linear filtering of an image f (x, y) with a 2-dimensional filter G_{xy} is calculated by the convolution operation $f * G_{xy}$. The convolution can be interpreted as follows: in each pixel (x, y), a new value is assigned based on the scalar product $<N, G_{xy},>$ between the filter G_{xy} and a patch N of the image around the pixel (x, y). The patch N must have the same size of the filter G_{xy}.

A 2-dimensional filtering with two 1-dimensional filters G_x and G_y is done with two convolutions as: $(f * G_x) * G_y$.

Convolution in the image (Cartesian) plane is equivalent to a multiplication of the respective frequency images in the frequency domain. That is:

$$f * G_{xy} = IFT\{FT(f) \cdot FT(G_{xy})\} \text{ (Equation 1)}$$

where FT stands for Fourier Transform and IFT for Inverse Fourier Transform.

Filtering in the frequency domain (multiplication in equation 1 above) can be carried out quickly when there are fast algorithms for the transformation from image (Cartesian) plane to the frequency plane (FT), and back to the image plane (IFT).

0. Take a picture with the web-camera.

Take a picture with the web-camera (Microsoft LifeCam VX-1000, format RGB24_320x240) using the "Image Acquisition Tool" (parameters: 1 frame, log to memory, 1 trigger and manual) and save it in Matlab's working memory (Export Data to MATLAB workspace).

```
>> imaqtool; % Load the Image Acquisition Tool
```

Take a picture and save in Matlab's working memory under the variable name: Picture. Check that the picture image is in the "workspace" with format = 240x320x3 uint8.

If the camera does not work, you can use the file Picture.jpg provided.

Convert the image to grayscale values format with double precision. Plot the image in a figure window.

```
>> GI = double (rgb2gray (Picture)); % convert to grayscale and double
>> figure (1); imagesc (GI); colormap (gray); truesize; % Display the image in Figure 1
```

1. Low-pass filtering.

Exercise 1.1: Building a low-pass 2D filter

Construct a 2-dimensional Gaussian filter G_{xy} using Matlab command fspecial (>> help fspecial) and display the filter.

```
>> s=1.4; %σ-value
>> hgauss=fspecial('gaussian',2*round(3*s)+1,s); %hgauss is a 2D-filter
>> figure(2); subplot(2,1,1); mesh(hgauss); %display filter
```

a) Print out the filter values on the screen (>>hgauss) and find out the size of the filter.

b) What would be the filtered value (output number) of a picture with the constant value 37 in all of its pixels? Figure it out by hand with the scalar product operation (remember that the amount of filter values sum = 1) and check it with Matlab (create an image of the same size as the filter with all values = 37 and calculate the scalar product).

Exercise 1.2: Building a low-pass 2D filter with two separable 1D filters

Construct two 1-dimensional filters G_x and G_y that arise from the 2-dimensional filter G_{xy}

```
>> gx=hgauss(5,:);  %middle row in the x-direction
>> gy=hgauss(:,5); %middle column in the y-direction
%make the sum of filter values =1
>> gx=gx/sum(gx); %gx is the 1-dim filter in the x-direction
>> gy=gy/sum(gy); % gy is the 1-dim filter in the y-direction
>> figure(2);subplot(2,1,2);stem(-4:4,gx);%display filter gx
```

a) Print the filter values on the screen (>>gx, >>gy) and find out the size of the filters and the "orientation" of their axis.

b) Verify that the 2-dimensional filter G_{xy} *can be created as gx * gy (>> gg = conv2 (gx, gy);), i.e. that the 2-dimensional hgauss filter is separable into gx and gy. Print filter values for hgauss and gg on the screen and compare them.*

*The comparison can be made analytically by computing the square mean error between hgauss and gg as follows (why is it computed (e. * e) before sum?):*

```
>> e=hgauss-gg; %error
>> e2m=sqrt(sum(sum(e.*e)));% square mean error
>> e2m
```

Exercise 1.3: Comparing different filtering methods

Linear filtering can be done in three different ways: convolution with a 2 dimensional filter in the image plane, convolution with two 1-dimensional filters in the image plane, or multiplication in the frequency domain). Theoretically, the three methods provide the "same" results!

Now you will filter the grayscale image that you captured with the web-camera (variable name GI).

```
% Filter with 2- dimensional filter in the image plane
>> y=conv2(GI,hgauss); %filtering by convolution
>> figure(3);subplot(2,2,1);imshow(GI/255); %Original image
>> figure(3); subplot(2,2,2); imshow(y/255); title('2-dim filter in the image plane');

% Filter with two 1- dimensional filters in the image plane
>> yy=conv2(conv2(GI,gx),gy); % two convolutions with 1-dim filters
>> figure(3);subplot(2,2,3); imshow(yy/255); title('two 1-dim filters in the image plane');

% Filter in the frequency domain according to (Equation 1)
>> [rf,cf]=size(GI); %image size
>> [rg,cg]=size(hgauss); %filter size
>> Nr=rf+rg-1; %number of frequency points in row-direction
>> Nc=cf+cg-1; % number of frequency points in column-direction
%NOTE! The image and the filter must have the same number of pixels in the frequency domain,
%since filtering is done by pixel-wise multiplication!!
>> F=fft2(GI,Nr,Nc); %frequency description of the image using the FT
>> G=fft2(hgauss,Nr,Nc); %frequency description of the filter using the FT
>> YYY=F.*G; %filtering by pixel-wise multiplication
>> yyy=real(ifft2(YYY)); %return to the image plane using the IFT
>> figure(3); subplot(2,2,4); imshow(yyy/255); title('filtering in the frequency plane');

% Check if the different methods give the same result by calculating the square mean error.
>> fel1=y-yy; %Difference between 2-dim convolution and two 1-dim convolution
>> fel2=y-yyy; % Difference between 2-dim convolution and frequency multiplication
>> kmf1=sum(sum(fel1.*fel1))
>> kmf2=sum(sum(fel2.*fel2))
```

a) How many multiplications per pixel are needed with filtering via hgauss (2-dim filters) and via gx and gy (two 1-dim filters)?

b) Do we obtain the same result with the different filtering methods?

c) What is the effect observed in the picture after filtering with Gaussian filters?

Exercise 1.4: Increasing the size of the filter

Increase the size of the Gaussian filter, use now sigma = 2.4 (s = 2.4 in Matlab code of exercise 1.1). Build a 2-dimensional filter as previously done with Matlab code, and filter the image GI with the new 2-dimensional filter in the image plane. Plot the resulting image.

a) Compare the size of the 2-dimensional Gaussian filters with s = 1.4 and s = 2.4 as a function of the value of s.

b) Compare the effect on the image when you use a smaller (s = 1.4) or greater (s = 2.4) filter.

Exercise 1.5: Noise reduction

Noise reduction with linear filtering.

Add noise to the image GI and then, filter it with the 2-dimensional filter of s = 2.4. Display both the noisy image and the filtered image (see earlier MATLAB code to see how you can filter and display a picture).

You can add noise to the image as:

```
>> fnoise=imnoise(GI/255,'gaussian',0,0.01).*255; %gaussian noise
```

Compare the noisy and filtered images. What is the result of filtering?

2. High-pass filtering with derivative filters.

Exercise 2.1: Building a high-pass 2D filter

Construct 2-dimensional derivative filters (in this exercise we will use a Sobel filter) and display them:

```
>> hsobel=fspecial('sobel'); %derivative filter in y-direction
>> vsobel=hsobel'; %derivative filter in x-direction
```

Print the two filters on the screen and guess how they will behave when applied to an image. Compare them with the high-pass filters seen during the lectures.

What is the sum of the filter values in a derivative filter? Calculate this sum using Matlab.

Filter image GI to create derivative images in the x-direction and y-direction, and display them:

```
>> fy=conv2(GI,hsobel);
>> fx=conv2(GI,vsobel);
>> figure(5);imshow(GI/255); %original image
>> figure(6); imagesc(fx); colormap(gray); truesize; %derivative in x-direction
>> figure(7) ;imagesc(fy); colormap(gray); truesize; % derivative in y- direction
```

Study the derivative images. Explain the result of the derivation operation in the regions of the image with small variation in grayscale (homogenous areas), and rapid transition from light-to-dark and dark-to-light (edges).

Exercise 2.2: Computing the gradient image

The gradient of a pixel is defined as the vector $\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)^t$ and can be represented as a complex number $\nabla f = \frac{\partial f}{\partial x} + i \frac{\partial f}{\partial y}$. The gradient defined in this way is thus a vector.

The length of the gradient vector (magnitude) depends on how much grey values change around the pixel (x,y), and the direction (angle) of the vector is perpendicular to the direction of grey change.

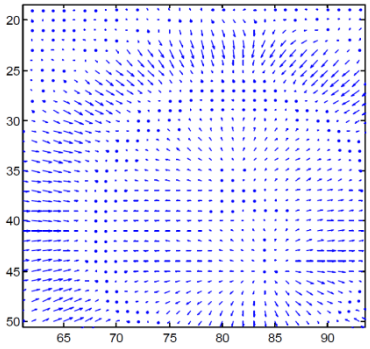


Figure 2. Example of a gradient image.

In the previous exercise 2.1 you have calculated the derivatives in the x and y directions of the image, i.e. fx and fy. Now calculate the gradient image and the display it:

```
>> vektorbild=fx+i*fy; %complex gradient
>> figure(8); quiver(real(vektorbild),imag(vektorbild)); axis('ij'); axis image;
```

How do you interpret the vector (gradient) of a pixel? Give an interpretation of the vector length (magnitude) and its direction (angle). Zoom the image in homogeneous areas and in edges (light-to-dark and dark -to-light) for help with your answer.

The magnitude of the gradient (= vector length) is known as the edge image. Calculate the edge image and display it as follows:

```
>> edge=abs(vektorbild); %edge image
>> figure(9); imagesc(edge);colormap(gray); truesize;
```

Exercise 2.3: Image binarization

Create a binary image by thresholding the edge image. This will give only those edges that have a higher value than the value of T (= threshold). To find an appropriate value of T, we use the maximum value of the edge image.

```
>> max(max(edge)) %obtain the highest value of the edge image
% test with values of T (choose a smaller value than the largest edge value)
>> T=?; % threshold value
>> figure(10); imagesc(edge>T); colormap(gray); truesize;
```

What value of T gives the "best binary image" ("best binary image" is the image that shows the "most important" contours of the image)?

Exercise 2.4:

Calculate the edge image of the noisy picture fnoise (obtained in exercise 1.5) and display it. Look at the code above employed in exercises 2.1-2.2 to calculate the edge image of GI without noise, and modify it appropriately to be applied to fnoise.

Threshold the edge image obtained. Try to find an optimal threshold for "best binary image" and display it (see previous code in 2.3).

Noise produces structures in the image with high frequency, that is, they survive the high-pass filtering (derivative filtering) that occurs when computing the edge image and the binary image.
Can you see this effect? Compare edge and binary images from exercise 2.3 (with no noise) and exercise 2.4 (with noise).

3. Filtering with web-camera.

If your webcam does not work, you can see this demo in the teacher´s computer.

Run the m-file EdgeDetLive with the command >>EdgeDetLive;

Do the necessary changes in the EdgeDetLive m-file to do smoothing with a Gaussian filter instead of edge-filtering and save it in a new m-file GaussianLive.m.

Run your new m-file with the command >>GaussianLive;