Centre for Mathematical Sciences, Lund University, September 2009

Image Analysis, Laboratory Session 2

Preparatory Exercise

This computer laboratory session has the Fourier transform as a common theme. Repeat the properties of the Fourier transform and make sure that you have some feeling of how it is working. What would for instance happen with the Fourier transform if you translate the input signal?

Let $\mathcal{F}_N$ be the Fourier matrix of order N, i.e. with $\omega_N = e^{-2\pi i/N}$,

$$\mathcal{F}_N = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_N & \omega_N^2 & \dots & \omega_N^{N-1} \\ 1 & \omega_N^2 & \omega_N^4 & \dots & \omega_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_N^{N-1} & \omega_N^{2(N-1)} & \dots & \omega_N^{(N-1)(N-1)} \end{pmatrix} .$$

The discrete Fourier transform is the mapping

$$f \longrightarrow F = \mathcal{F}_N f \ .$$

Observe that we use the same definition of the Fourier transform as Matlab does. (Sometimes there is a factor $1/N$ in the definition.) The connection between the usual (continuous) Fourier transform, a Fourier series and the discrete Fourier transform might not be clear for everyone. In principle you obtain the discrete Fourier transform for f by the following four changes from the continuous one:

- cut off f to a suitable interval
- repeat the resulting function periodically
- sample this periodic extension
- calculate the Fourier series

The periodic continuation implies that the end-points of the initial interval in this case has to be considered as neighbors, which has practical consequences in image analysis, where all functions (images) has finite extent. As a result of the periodic extension and sampling, the discrete Fourier transform also becomes a periodic function, of the frequency variable.

Solve the following exercises:

1. What is $\mathcal{F}_N e_k$, where $e_k = [0 \ldots 0 \ 1 \ 0 \ldots \ 0]^T$ (a one on position k)? Usually, the absolute value of the Fourier transform (=the amplitude spectrum), is shown in graphical illustrations, sometimes its quadrat or logarithm. What will this become in our case?

2. What is $\mathcal{F}_N e$, where $e = [1 \ 1 \ \ldots \ 1]^T$? (Use a known property for $\mathcal{F}_N^{-1}$.)

3. Let $f = [1 \ 1 \ \ldots 1 \ 0 \ \ldots \ 0]^T$, with p ones. Show that

$$|\mathcal{F}_N f(u)| = \left| \frac{\sin(\pi u p/N)}{\sin(\pi u/N)} \right|, \quad u = 0, \ldots, N-1 \ .$$

   The expression within the absolute value is the periodic correspondence to the sinc-function. You can find out what the graph of this function looks like below.

4. What is the two-dimensional discrete Fourier transform of the $N \times N$-matrix f who has ones in the first row and zeros in the others?

One-dimensional Fourier transform

The one-dimensional discrete Fourier transform is easy to perform and visualize in Matlab. Try

```
>> in=zeros(1,64);
>> in(1:13)=ones(1,13);
>> repin = [in in in in];
>> plot(repin);  axis([0 256 0 2]);
>> ut=abs(fft(in));
>> reput = [ut ut ut ut];
>> plot(reput);
```

Is this consistent with preparatory exercise 3? Observe that you obtain the Fourier transform of the discrete continuation when applying the discrete Fourier transform.

There are some special properties of the discrete Fourier transform and in the Matlab implementation of this, that could be good to know:

- All vectors in Matlab are indexed with 1 as the first index. This implies that if `F=fft(f)`, then `F(1)` is the component with frequency 0 (corresponding to N times the mean value of f).
- The discrete Fourier transform is a periodic function of the frequency variable. This implies that $F(-k) = F(N-k)$ when F is a transform with N components. The highest frequency that can be represented is $k = N/2$. If `f` is real, then `F(N-k)=F(-k)` is the complex conjugate of `F(k)` and thus gives no additional information. In particular, the absolute values `abs(F(N-k))` and `abs(F(k))` are always equal.
- For real functions it is preferable to put the zero frequency *in the middle* of the periodic interval, since the symmetry around $k = 0$ is more evident then. There is a Matlab command, `fftshift`, which does exactly this. In principle, it switches places between the left and right part part of its vector input. When interpreting the vectors as periodically extended, this implies a translation of half a period. A minor difficulty is that because $N = 2^n$ for FFT always is an even number, there is no middle point in `1:N` and instead the zero frequency corresponds to `k=N/2+1`.

Try `fftshift` in Matlab:

```
>> help fftshift
>> plot(in); axis([0 64 0 2])
>> plot(fftshift(in)); axis([0 64 0 2])
>> plot(ut); axis([0 64 0 15])
>> plot(fftshift(ut)); axis([0 64 0 15])
```

Note especially the symmetry around $k = 33$ in `ut`. Explain it!

How to make the Fourier transform of images in Matlab

You may try with one of your own images. If you don't have anyone, you may use `fkalle.pgm`. Read it with `im=double(imread('cameraman.tif'));`

Look at the image by entering

```
>> colormap(gray(256));
>> bild1=im;
>> image(bild1);
>> bild2=fftshift(im);
>> image(bild2);
```

What does the routine `fftshift` do in the two-dimensional case?

There is a special Matlab routine, `fft2`, for the two-dimensional fast Fourier transform. Transform the images:

```
>> f1=fft2(bild1);
>> f2=fft2(bild2);
```

The transformed images are hard to visualize. There are especially three things to consider:

- The transformed image will in general become complex, which makes it hard to visualize the image on the screen. Usually, the absolute value is displayed.
- Images are normally represented by matrices with positive elements. A consequence of this is that the zero frequency $(0, 0)$ has a much higher amplitude than other frequencies. This implies that if the result is displayed directly with `imagesc(abs(f1))`, you will usually only see one single white dot, while the rest of the image is black. (Where is the dot located?) Therefore, it is advantageous to make a gray-scale transformation when showing the Fourier transform as an image. Another solution would be to reduce the maximum values by thresholding. Try

  ```
  >> imagesc(min(abs(f1),5000));
  ```

  Another option is to take the logarithm of the gray-scale, in the same way as for a Bode diagram in automatic control theory. Try

  ```
  >> imagesc(log(abs(f1)));
  ```

- The low frequencies, which often have relatively high amplitude, are located in the corners of the image. For instance, the mean value (frequencey $(0, 0)$ is located in the matrix element `F(1,1)`. Usually, `fftshift`, is used to move the origin to the centre of the image. Try

  ```
  >> F1=fftshift(min(abs(f1),5000));
  >> imagesc(F1);
  ```

  Show the image of `f2` in the same way.

Look at the numerical values of `f1(1:5,1:5)` and `f2(1:5,1:5)`. What is the relation between the transforms? Explain this relation!

Transforms of simple images

Produce some simple synthetic images:

```
>> bild1=zeros(64,64);
>> bild1(1:3,1:3)=256*ones(3,3);
>> bild2=zeros(64,64);
>> bild2(1:16,1:16)=256*ones(16,16);
>> bild3=zeros(64,64);
>> bild3(1:4,1:16)=256*ones(4,16);
```

Display them and their Fourier transforms. Use `subplot` to display all the images in one window. Explain what you see! (Hint: Think about the one-dimensional result above and the scaling rule.)

Sinusoidal images

There is a script, called `sinus.m`, that creates sinusoidal images. Look at it in an editor. Why is there a one added to the sinus function?

Before the script can be used, you have to set the input parameter `k`, e.g. as

```
>> k=[3 4];
>> sinus;
```

Here, `k`, is the two-dimensional frequency (including direction) for the image. What do the dots in the image of the Fourier transform means? Note also that the line connecting them is exactly orthogonal to the edges in the input image.

Try with `k=[3 4]*2`, `k=[3 4]*4`, `k=[3 4]*7`, `k=[3 4]*9` and `k=[3 4]*17`. What is this phenomena called and what is the reason for it? Try also with `k=[6.2 8.52]`. What happened? Why? Change the script `sinus.m`, by changing row 7 from `imagesc(x2)` to `imagesc(fftshift(x2))`. Try again with `k=[6 8]` and `k=[6.2 8.52]`. What kind of unexpected effect can edges have on the discrete Fourier transform?

Filtering

Convolutions (or filterings) using kernels with very small support (i.e. kernels that are non-zero on a small set), as the ones studied in computer laboratory session 1, are simple and efficient to calculate directly in the image plane. However, if the kernel has a larger support, it is often advantageous to perform the convolution in the Fourier space instead, where it corresponds to a simple multiplication. Observe that this procedure gives a *periodic* convolution, where the image is continued periodically and not with zeros as in computer laboratory session 1.

Study how the convolutions from computer laboratory session 1 (x-derivation, y-derivation and Laplace) performs in the frequency plane. You have to extend the filters with zeros, so that it becomes of the same size as the image, e.g.

```
>> derx=[1 -1];
```

```
>> bild1=zeros(64,64);
>> bild1(1,1:2)=derx;
>> f1=fft2(bild1);
```

Are these filters high-pass or low-pass?

The fourier transform of noise is usually distributed evenly in the frequency plane. Since the noise-free images usually have a high amplitude for low frequencies and low amplitude for high frequencies, noise will be dominant in the high frequencies. Which kind of filter will amplify the noise? Which will reduce the noise?

Instead of specifying the filters in the image plane, it is also possible to specify them in the frequency plane and then see what it corresponds to in the image plane. For instance, an ideal low-pass filter should preserve the low frequencies and cut away the higher frequencies. Such a filter, `laagpass`, and a synthetic image, `svartvit`, is located in the Matlab file `laagpass.mat`. Load these with

```
>> load laagpass
```

and look at them and there transforms. Filter with the low-pass filter in the frequency domain using the command

```
>> sv2=abs(ifft2(laagpass.*fft2(svartvit)));
```

Observe the *ringings* with all the three edges. These are examples of the *Gibbs phenomena*. If you can't see the edges, you may look at a slightly larger part of the periodic extension using

```
>> imagesc([sv2 sv2 sv2 ;sv2 sv2 sv2;sv2 sv2 sv2]);
```

Edge detection

In computer laboratory exercise 1, you made some experiments with edge detection, using difference and Laplacian filters. In Forsyth-Ponce (Chapter 10.1) and in the lecture notes, better methods have been described. For derivatives in the x-direction you can use e.g. the Sobel- or the Prewitt-masks of size 3×3:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}.$$

The corresponding Sobel-operator of size 5×5 is

$$\begin{bmatrix} 1 & 2 & 0 & -2 & -1 \\ 2 & 4 & 0 & -4 & -2 \\ 2 & 6 & 0 & -6 & -2 \\ 2 & 4 & 0 & -4 & -2 \\ 1 & 2 & 0 & -2 & -1 \end{bmatrix}.$$

Usually, you would like the output of an edge detector to be a decision about what pixels that belong to and edge and those who don't. Such an edge detector can by obtained by first filtering with a suitable filter and then thresholding the result. A typical edge detection routine can look like this:

```
% kantprewitt.m
% KANTPREWITT - Kantprewitt is an edge detector based on the
%               Prewitt mask.
% In:  inbild   : The image where edges are yo be detected.
%      troeskel : Threshold.
% Out: utbild   : The output image containing edges.

prewitt3x=[
-1  0  1;
-1  0  1;
-1  0  1];
prewitt3y=prewitt3x';
derx = conv2(inbild,prewitt3x);
dery = conv2(inbild,prewitt3y);
gradabs = sqrt(derx.^2+dery.^2);
gradarg = atan2(dery,derx);
utbild=( gradabs > troeskel);
```

The resulting image after convolving with `prewitt3x` has intensities with high absolute value where the gray-levels in the input image are changing rapidly in the x-direction. By looking at the absolute value of the approximative gradient, we obtain a result that is sensitive to edges in both the horizontal and vertical direction.

Exercise: Design three different edge detectors based on the Sobel-, Prewitt- and Laplace-masks. A more sophisticated edge detector based on an idea by Canny is implemented in the Matlab routine `edge`, which is used as `utbild=edge(inbild,'canny')`. It works essentially in the same way as `kantprewitt`, i.e. first convolution and then thresholding. Canny has developed filters, that in some sense are optimal. Try out the four different edge detectors on a suitable image. If you don't have an image available, you can load one using `load klossar`. It is often difficult to set the different thresholds. A crude guess is:

```
Sobel:   troeskel=500;
Prewitt: troeskel=100;
Laplace: troeskel=40;
```

In the edge detectors you developed above, it is easy to try out new thresholds using

```
>> imagesc(filterbild > troeskel);
```

Investigate especially what happens when the threshold is changed. Is the result highly dependent on the choice of threshold? Does there exist a much better threshold than the ones proposed above? What edge detector works best?

When you have evaluated your own implementations, you can also check out Matlab's edge detectors using `edgedemo`.

The orientation tensor

We are now going to construct the orientation tensor of an image. Take an image and convert it to gray-scale. For simplicity, we assume that the the image is in the format `double` and that the gray-scale is between 0 and 255. If you don't have an image you can load 'klipp020.jpg'.

```
bild = double(rgb2gray(imread('klipp020.jpg')));
```

Calculate smoothened first order derivatives, e.g. using

```
a=1.5;
filtsize1=round(3*a);
[x,y]=meshgrid(-filtsize1:filtsize1,-filtsize1:filtsize1);
filtx=-2*x.*exp(- (x.^2 + y.^2)/a^2)/(a^4*pi);
filty=-2*y.*exp(- (x.^2 + y.^2)/a^2)/(a^4*pi);
Lx=conv2(im,filtx,'same');
Ly=conv2(im,filty,'same');
```

Why is the size of the filter dependent on a? Display (using `imagesc`) the filter functions `filtx` and `filty`. Study the results `Lx` och `Ly`. Discuss the appearance with someone. Is it what you expected?

Now we are going to calculate the elements in the orientation tensor

```
b=4;
filtsize2=round(3*b);
[x,y]=meshgrid(-filtsize2:filtsize2,-filtsize2:filtsize2);
filt=exp(- (x.^2 + y.^2)/b^2)/(b^2*pi);
Wxx=conv2(Lx.*Lx,filt,'valid');
Wxy=conv2(Lx.*Ly,filt,'valid');
Wyy=conv2(Ly.*Ly,filt,'valid');
```

Display the images `Wxx`, `Wxy` and `Wyy`. Construct the determinant D, trace T and eigenvalues λ_1 and λ_2. Since the relation between λ_1 and λ_2 and the trace and determinant is $\lambda_1\lambda_2 = D$ and $\lambda_1 + \lambda_2 = T$ (why?), how can you calculate λ_1 and λ_2 from D and T?

```
Wdet=Wxx.*Wyy-Wxy.^2;
Wtr=Wxx+Wyy;
Wslask = (Wtr.^2/4-Wdet);
Wl1 = Wtr/2 + sqrt(Wslask);
Wl2 = Wtr/2 - sqrt(Wslask);
```

Is it possible that complex eigenvalues appear?

Segment the image in different areas, using some threshold, for big and small eigenvalues. Display the results for some different thresholds.

```
threshold = 5;
Wtexture = (Wl2>threshold);
Wflow = (Wl2<threshold) & (Wl1>threshold);
Wflat = (Wl1<threshold);
```

Corner detection

Apart from edges, it is often interesting to detect corners in an image. A common method to detect corners is the one called Harris corner detector, named after its inventor. However, similar algorithms existed before Harris invented his corner detector. Let the matrix M denote the same orientation tensor as above.

If both eigenvalues to M are large, it means that a small area around the centre not can be moved in any direction without changing the gray-levels significantly. This indicates that the point is a corner point. If one eigenvalue is large and one is small, the area can be moved in one direction without significant changes in the gray-levels, indicating that the point is an edge point. If both eigenvalues are small, the area can be moved in any direction without significantly changing the gray-levels, implying that the point is in an area with little variation.

We would like to find an expression that becomes large when both eigenvalues are large. For numerical reasons, the determinant and trace is used instead of the eigenvalues. The corner response of a point is given by

$$R = (k + 1/k)|\det(M)| - |\operatorname{tr}(M)^2 - 2\det(M)| \ .$$

The constant k is used to suppress big differences between the eigenvalues.

The corner detection algorithm is implemented in the file `harris.m`. Try to find the 20 strongest corners in an image using

```
>> harris(bild,20);
```

Extra: Specially designed filter

Load the image `skalle.pgm` in Matlab and look at it and its Fourier transform.

```
>> skalle=double(imread('skalle.pgm'));
>> subplot(2,2,1); image([skalle skalle;skalle skalle]);
>> fskalle=fft2(skalle);
>> subplot(2,2,2); imagesc(min(abs(fftshift(fskalle)),9000));
```

As you can see the image is destroyed by a sinusoidal signal. In the frequency plane it corresponds to two spikes. These are located in the matrix elements $(13, 10)$ and $(53, 56)$. Remove these spikes and make the inverse transform:

```
>> fskalle(13,10)=0; fskalle(53,56)=0;
>> subplot(2,2,4); imagesc(min(abs(fftshift(fskalle)),9000));
>> nyskalle=real(ifft2(fskalle));
>> subplot(2,2,3); image([nyskalle nyskalle;nyskalle nyskalle]);
```

What does the result look like? Look at the Fourier transform. Some directions are more common. What does these correspond to in the original image? Explain what you are seeing for the teacher before you log out and go home.