Chapter 3

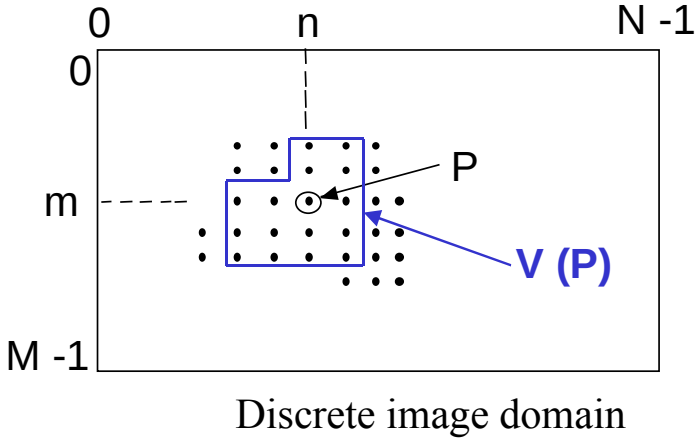
Fundamental of Image Processing

Linear Filtering

The Concept of Neighborhoods

The Concept of Neighborhoods

- Point P: affix p = (m, n)
- Neighborhood: V(P) = {P' connected to P}



Discrete image domain

Examples: 4-N  8-N 

4-connected neighborhood 8- connected neighborhood

Image processing is fundamentally based on techniques using neighborhoods. An image processing which is performed at the affix p of the pixel P depends not only on this pixel P but also on pixels in its neighboring area. Let us consider a pixel P whose location in the image is defined by the coordinates (m, n). Its affix is thus p = (m, n). A neighborhood V(P) of the pixel P can be defined by a set of pixels P' that are connected to P.
The pixel P (circled in the figure) belongs to its own neighborhood V(P).
We must here define the concept of connectivity: the criteria that describe how pixels within a discrete image form a connected group. Rather than developing this concept, we show here in the top figure the two most common examples:

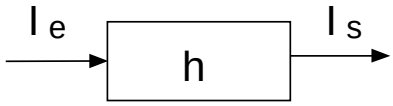
- a "4-connected" neighborhood: the surrounded pixel has only four neighboring pixels. The distance between the surrounded pixel and any pixel of its neighborhood is d_4 ;
- a "8-connected" neighborhood: the surrounded pixel has eight neighboring pixels. The distance between the surrounded pixel and any pixel of its neighborhood is d_8 ;

We define the two following distances in the case of a digital neighborhood (square sampling structure):

- $d_4(P, P') = | m - m' | + | n - n' |$;
- $d_8(P, P') = Sup(| m - m' |, | n - n' |)$.

Neighborhood operators

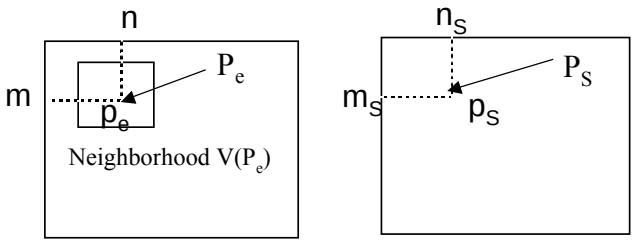
Linear filters (convolution)



➢ Transverse filter: h is the 2D impulse response called also the Point Spread Function (PSF) .

• ***Finite Impulse Response filter (FIR)***

$$I_S(m,n) = \sum_{(i,j)\in V(P)} \sum h(i,j)\, I_e(m-i\ ,n-j)$$



• ***Infinite Impulse Response filter (IIR)***

$$Is(m,n)=\frac{1}{b_{0,0}}\times\left(\sum_{(i',j')\in V(P)}\sum a_{i',j'}.Ie(m-i',n-j')-\sum_{(i,j)\in V'(P')}\sum b_{i,j}.Is(m-i-1,n-j-1)\right)$$

A linear filter builds an output image I_S from an input image I_e. Typically I_S is the same size as I_e. Each pixel P_S (affix $p_S = (m_S, n_S)$) within the output image I_S is computed from the neighboring pixels $V(P_e)$ of point P_e (affix $p_e = (m, n)$) in the Input image I_e. Generally: $p_S = p_e$, i.e. $m = m_S$, and $n = n_S$.

Let us consider a transverse filter (the simplest case): the output value of the pixel P_S is computed as a weighted sum of neighboring pixels $V(P_e)$. Linear filtering of an image is thus accomplished through an operation called convolution.

Here is the example for a transverse filter where the output value Is(m, n) of the pixel P_S is the linear sum of the values I_e (m, n) of the neighboring pixels P_e, weighted using the coefficients h (i, j) of the Point Spread Function (PSF):

$$I_S(m,n)=\sum_{(i,j)\in V(P_e)}\sum h(i,j).I_e(m-i,n-j)$$

Note that the neighborhood size is thus defined by the Finite Impulse Response (FIR) « h ».

There are also recursive filters. To compute the output values of the pixels I_S is more complex because the linear combination depends on:

- the neighborhood of I_e (m, n), the associated set of coefficients is $\{a_{i',j'}\}$;
- previously-calculated output values of Is(m, n) go back into the calculation of the latest output Is(m, n). The associated set of coefficients is $\{b_{i,j}\}$.

$$I_S(m,n)=\frac{1}{b_{0,0}}.\sum_{(i',j')\in V(P_e)}\sum a_{i',j'}.I_e(m-i',n-j')-\sum_{(i,j)\in V(P_S)}\sum b_{i,j}.I_S(m-i-1,n-j-1)$$

The result depends on the way you choose to scan the image Ie (causality) because that directly influences the previously-calculated values I_S (m, n).

Example – Convolution – Correlation

Example : filter support of size (3×5) (3 in vertical and 5 in horizontal)

Convolution kernel : $h = \begin{bmatrix} h_{-1,-2} & h_{-1,-1} & h_{-1,0} & h_{-1,1} & h_{-1,2} \\ h_{0,-2} & h_{0,-1} & h_{0,0} & h_{0,1} & h_{0,2} \\ h_{1,-2} & h_{1,-1} & h_{1,0} & h_{1,1} & h_{1,2} \end{bmatrix}$

Convolution

$$I_s(m,n) = \sum_{j=-2}^{+2} \sum_{i=-1}^{+1} h(i,j).I_e(m-i,n-j)$$

Correlation Let $h^*(i,j) = h(-i,-j)$

(=> symmetric of h with respect to (0,0))

$$I_s(m,n) = \sum_{j=-2}^{+2} \sum_{i=-1}^{+1} h^*(i,j).I_e(m+i,n+j)$$

$$h^* = \begin{bmatrix} h_{1,2} & h_{1,1} & h_{1,0} & h_{1,-1} & h_{1,-2} \\ h_{0,2} & h_{0,1} & h_{0,0} & h_{0,-1} & h_{0,-2} \\ h_{-1,2} & h_{-1,1} & h_{-1,0} & h_{-1,-1} & h_{-1,-2} \end{bmatrix}$$

In the case of linear filtering using convolution, the filter is completely characterized by the coefficients { h(i,j) } (which can also be written { $h_{i,j}$ } to remind us that "i" and "j" are discrete variables). These coefficients define the convolution "kernel" of the filter.
This kernel defines:

- the neighborhood $V(P_e)$ to use (in the example it is a (3×5) neighborhood where the position (0,0) must be centered on P_e);
- the respective weightings h(i, j) of each neighboring pixel needed to calculate the new value P_S.

When the size of the rectangular neighborhood support (I, J) and the weightings are known, we can calculate all the pixels of the image Is:

$$I_S(m,n) = \sum_{i \in I} \sum_{j \in J} h(i,j).I_e(m-i,n-j)$$

Note: When computing an output pixel at the boundary of an image, a portion of the convolution kernel is usually off the edge of the image I_e. It is thus necessary to specify "boundary conditions" to process border distortions (simple solution: to consider only the pixels in the image). Some ways of dealing with the boundaries are described in the exercise: "Linear filtering" in this chapter.

For example, let us consider a linear filter « h ». Its convolution kernel is:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

In fact this filter is the sum of a "Laplacian" filter (contour detection) and an Identity filter. Together they form a new filter "h", an "enhancement" filter.
Let the 3×3 image I_e be defined by:

$$\begin{bmatrix} 4 & 125 & 255 \\ 7 & 0 & 45 \\ 9 & 56 & 13 \end{bmatrix}$$

This image I_e is filtered by h.
Here are all the steps to compute the output value of the central pixel $I_S(1, 1)$:

$$I_S(1,1) = \sum_{i=-1}^{+1} \sum_{j=-1}^{+1} h(i,j) \,.\, I_e(1-i,1-j)$$

$$= \sum_{i=-1}^{+1} h(i,-1) \,.\, I_e(1-i,1-(-1)) + h(i,0) \,.\, I_e(1-i,1-0) + h(i,1) \,.\, I_e(1-i,1-1)$$

$$= \Big[h(-1,-1).I_e(1+1,2) + h(-1,0).I_e(1+1,1) + h(-1,1).I_e(1+1,0)$$

$$+ h(0,-1).I_e(1,2) + h(0,0).I_e(1,1) + h(0,1).I_e(1,0)$$

$$+ h(1,-1).I_e(1-1,2) + h(1,0).I_e(1-1,1) + h(1,1).I_e(1-1,0) \Big]$$

$$= (0\times13) + (-1\times56) + (0\times9) + (-1\times45) + (5\times0) + (-1\times7) + (0\times255) + (-1\times125) + (0\times4)$$

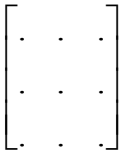
$$I_S(1,1) = -233$$

<u>Warning</u>: row and column indices of the input image are not the same as the indices of the convolution kernel. In the input image, indices run from 0 to M-1 (rows) and from 0 to N-1 (columns). In convolution kernel, the element $h_{0,0}$ is centered and the indices run from –I to +I (rows) and from –J to J (columns).

Note that the bi-dimensional function « h^* », which is the symmetric of h with respect to (0, 0), can be used to compute the output values of the pixel $I_S(m, n)$. The output values are thus defined as the correlation between I_e and h^* :

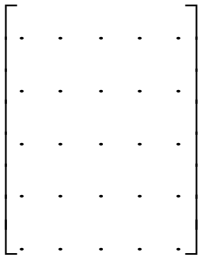
$$I_S(m,n) = \sum_{i \in I} \sum_{j \in J} h^*(i,j).I_e(m+i,n+j)$$

Typical examples of convolution masks



3×3 Mask

3×5 Mask



5×5 Mask

Convolution masks:

(~ circular: 45 pixels ; rectangular: 11×11)

• DC filter Gain: $\sum_i \sum_j h(i, j) = H(\nu_X = 0, \nu_Y = 0)$

• Symmetric filter : h (-i,-j) = h (i,j) then: convolution ≡ correlation

=> H (ν_X, ν_Y) is a real transfer function (no phase term)

The shape of a convolution mask can be one of many kinds. The size of the convolution mask also defines the size of the neighborhood. Some filter supports are presented here: 3×3, 3×5, and 5×5.

As in case of 1D signals, we define the frequency response $H(\nu_X, \nu_Y)$ of the filter. It depends on the horizontal (ν_X) and vertical (ν_Y) spatial frequencies. $H(\nu_X, \nu_Y)$ is the 2D Fourier transform of the impulse response:

$$H(\nu_X, \nu_Y) = \sum_n \sum_m h(m, n) \exp\left[-2j\pi (n\nu_X + m\nu_Y)\right]$$

The DC component $H(\nu_X = 0, \nu_Y = 0)$ can be computed as the sum of all the convolution kernel coefficients h(i, j): $Gain\_DC = \sum_i \sum_j h(i,j)$ (DC stands for direct current, an electrical engineering term).

The exercise "FFT" details the methods to compute and display the spectrum image and the frequency response of a filter. The notion of spatial frequency is developed there. In the special case, when the filter is symmetrical, $H(\nu_X, \nu_Y)$ is a real frequency response.

Note: in the following chapters we will use the term "transfer function" instead of "frequency response" (it is a misuse of language).

Exercise Chapter 3 – Convolution

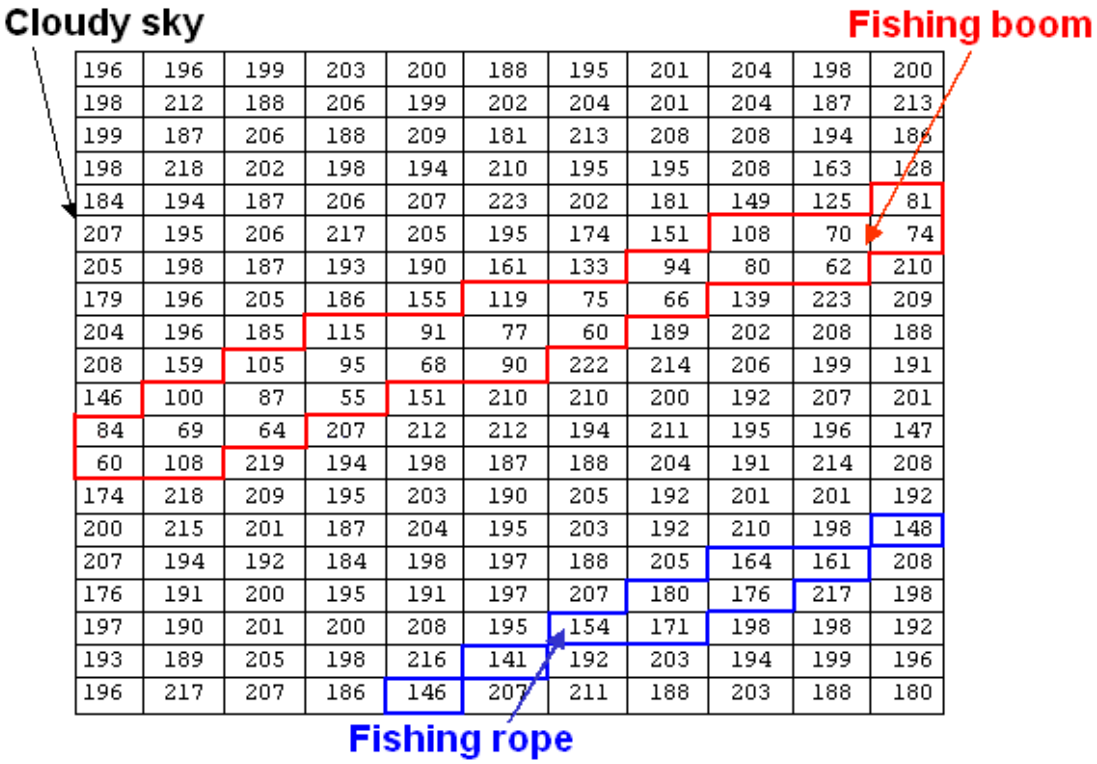
In this exercise, no programming is required. The goal is to perform a filtering by computing the convolution of an input image and the filter's kernel.

Convolution

We will use the convolution operator on a small portion of the original image "*BOATS*". On the figure below, this portion is delimited by a red rectangle that contains a part of the fishing boat's boom.

Image « BOATS »

The following array I_0 contains the luminance value of the pixels which are in the (20×11) delimited zone. In two diagonal bands, we can easily distinguish the pixels which belong to the fishing boom and to the rope.



196	196	199	203	200	188	195	201	204	198	200
198	212	188	206	199	202	204	201	204	187	213
199	187	206	188	209	181	213	208	208	194	186
198	218	202	198	194	210	195	195	208	163	128
184	194	187	206	207	223	202	181	149	125	81
207	195	206	217	205	195	174	151	108	70	74
205	198	187	193	190	161	133	94	80	62	210
179	196	205	186	155	119	75	66	139	223	209
204	196	185	115	91	77	60	189	202	208	188
208	159	105	95	68	90	222	214	206	199	191
146	100	87	55	151	210	210	200	192	207	201
84	69	64	207	212	212	194	211	195	196	147
60	108	219	194	198	187	188	204	191	214	208
174	218	209	195	203	190	205	192	201	201	192
200	215	201	187	204	195	203	192	210	198	148
207	194	192	184	198	197	188	205	164	161	208
176	191	200	195	191	197	207	180	176	217	198
197	190	201	200	208	195	154	171	198	198	192
193	189	205	198	216	141	192	203	194	199	196
196	217	207	186	146	207	211	188	203	188	180

The goal is to perform two kinds of filtering on some elements of this 2D array.

1 – Low-pass filtering

Let us consider a (3×3) low-pass filter. Its convolution kernel is:

$$\frac{1}{16} \cdot \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

After having performed this filter on the (20×11) area, we observe the following partial result:

111	149	149	150	149	146	148	150	150	149	113
149	199	198	199	199	198	201	203	201	197	149
149	200	199	198	198	199	202	204	200	188	135
148	200		198	201	203	200	196	185	161	105
146	197	199	203	206	205	194	175	151	121	75
149	197	199	203	201		169	142	112	96	75
148	197	196	193	181	157	129	108	102		113
145	195	189	170	146	117	98		140	175	146
146	185	163	131	105	96	115	155	187	200	149
134	154	121	98		126	169	197	202	200	147
99	111	95	109	143	178	201	204	200	196	143
68		117	158	189	200	201	200	199	193	138
78	130	169	193	199	196	196		199	198	145
123	185	200	199	196	195	196	197	200	198	143
150	204	200	194	196	197	197	196	194	187	134
148	197	194		194	196	196	191		184	141
142	192	195	195	196	194	189	183	184	193	149
142	193	197	201	197	186	179	182	191	198	148
146	197	200	198	189		184	191	195	195	144
112	153	151	141	133	139	147	148	147	144	105

Fill in the gray fields to have the full output array I_S (to round to the nearest integers less than or equal to the elements.).

2 – Contrast enhancement filtering

We wish to enhance the contrast of object edges in the delimited area. We use thus an "enhancement" filter which has the following convolution kernel:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

After having performed this filter, we observe the following partial result:

586	373	408	410	410	343	382	405	417	399	589
383	291	117	252	178	238	209	188	220	126	492
412	100	265	121	283	71	277	223	226	226	395
389	309	201	200		257	155	183	325	160	210
321	186	127	221	207	301	237	208	123	162	78
451	170	244	275	216	212	189	198	90		9
441	207	133	185	236	168	161	40	-3	-273	705
290	202	271	262	189	127		-167		497	424
437	236	304	18	40	25	-263	403	268	228	332
531	186		132		-127	536	253	223	183	367
338	39		-265	210	387	224	173	152	247	460
145	-11	-262	510	292	257	149	262		217	130
-66		520	151	194	147		238	141	274	487
392	384	212	182	228	160	252	158	211	200	403
404	262	202	151		181	235	150	295	270	142
465	165	181	148	214	207	128	301			533
285	195	221	200	157	195	316	141	121	352	373
426	172	210	198	238	275		120	251	184	368
383	140	230	183	387	-105	251	270	167	219	409
570	493	427	379	121	537	468	323	445	358	516

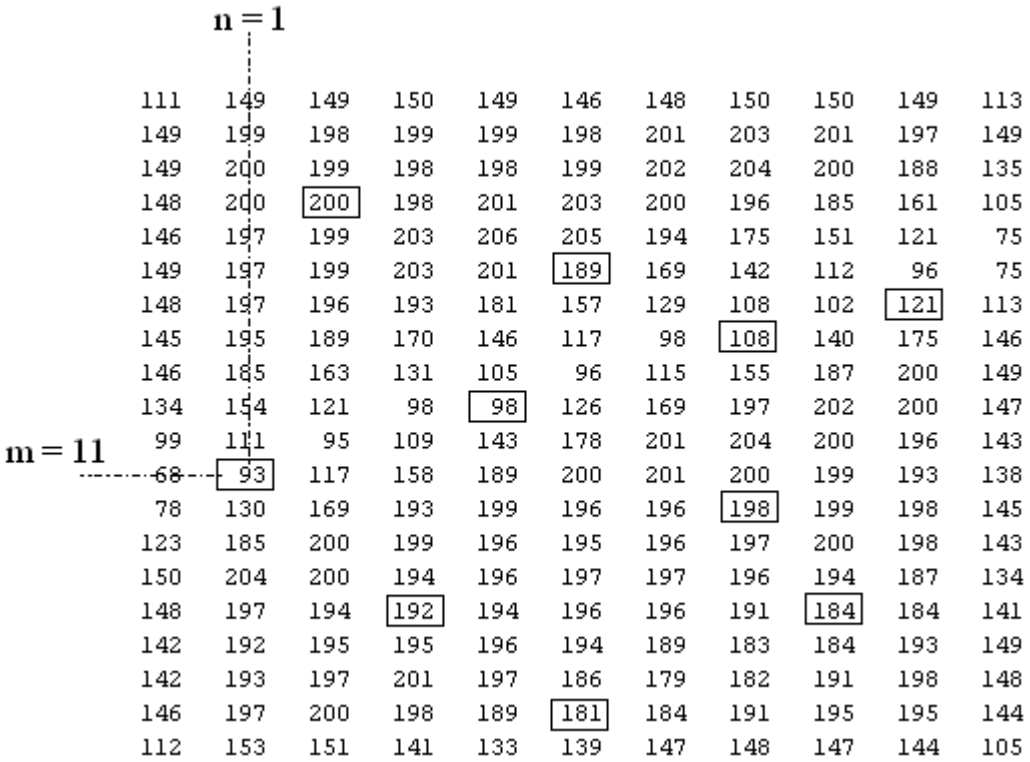
Again, fill in the gray fields to have the full output array.

3 – Border distortions

When computing an output pixel at the boundary of an image, a portion of the convolution kernel is usually off the edge of the image. Propose some examples of boundary conditions to fill in these “off-the-edge” image pixels.

Solution to the exercise on convolution

1 – Here is the 2D output array after low-pass filtering with the (3×3) binomial filter:



111	149	149	150	149	146	148	150	150	149	113
149	199	198	199	199	198	201	203	201	197	149
149	200	199	198	198	199	202	204	200	188	135
148	200	**200**	198	201	203	200	196	185	161	105
146	197	199	203	206	205	194	175	151	121	75
149	197	199	203	201	**189**	169	142	112	96	75
148	197	196	193	181	157	129	108	102	**121**	113
145	195	189	170	146	117	98	**108**	140	175	146
146	185	163	131	105	96	115	155	187	200	149
134	154	121	98	**98**	126	169	197	202	200	147
99	111	95	109	143	178	201	204	200	196	143
68	**93**	117	158	189	200	201	200	199	193	138
78	130	169	193	199	196	196	**198**	199	198	145
123	185	200	199	196	195	196	197	200	198	143
150	204	200	194	196	197	197	196	194	187	134
148	197	194	**192**	194	196	196	191	**184**	184	141
142	192	195	195	196	194	189	183	184	193	149
142	193	197	201	197	186	179	182	191	198	148
146	197	200	198	189	**181**	184	191	195	195	144
112	153	151	141	133	139	147	148	147	144	105

Here are all the steps to compute the output value of the pixel $I_S(11, 1)$:

$$I_S(11,1) = \frac{1}{16} \cdot \sum_{i=-1}^{+1} \sum_{j=-1}^{+1} h(i,j) \cdot I_0(11-i, 1-j)$$

$$= \frac{1}{16} \cdot \sum_{i=-1}^{+1} h(i,-1) \cdot I_0(11-i, 1-(-1)) + h(i,0) \cdot I_0(11-i, 1-0) + h(i,1) \cdot I_0(11-i, 1-1)$$

$$= \frac{1}{16} \cdot \Big[h(-1,-1) \cdot I_0(11+1, 2) + h(-1,0) \cdot I_0(11+1, 1) + h(-1,1) \cdot I_0(11+1, 0)$$

$$+ h(0,-1) \cdot I_0(11, 2) + h(0,0) \cdot I_0(11, 1) + h(0,1) \cdot I_0(11, 0)$$

$$+ h(1,-1) \cdot I_0(12-1, 2) + h(1,0) \cdot I_0(12-1, 1) + h(1,1) \cdot I_0(12-1, 0) \Big]$$

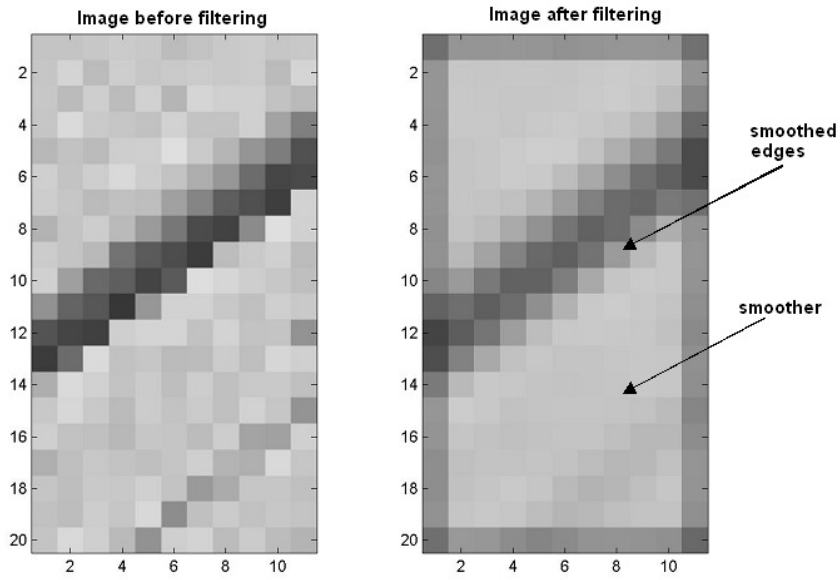
$$= \frac{1}{16} \cdot [\, 1x219 + 2x108 + 1x60 + 2x64 + 4x69 + 2x84 + 1x87$$

$$+ 2x100 + 1x146 \,]$$

$$= 1500/16 = 93.75$$

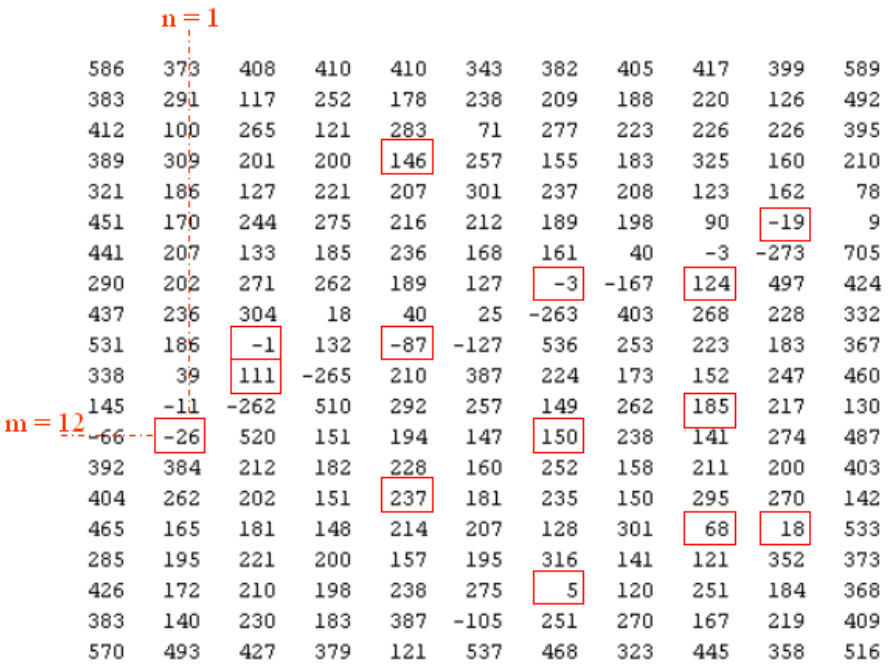
Thus: $I_S(11,1) = E[\, 93.75 \,] = 93$

Displayed here are the images before (on the left) and after (on the right) filtering:



Here, the smoothing effects (caused by removing the high spatial frequencies) of the low-pass filter are strongly visible.

2 – Here is the array after performing the 3×3 enhancement filter:



Here are all the steps to compute the output value of the pixel $I_S(12, 1)$ for example:

$$Is(12,1)=\sum_{i=-1}^{+1}\sum_{j=-1}^{+1} h(i,j).I_0(12-i,1-j)$$

$$=\sum_{i=-1}^{+1} h(i,-1).I_0(12-i,1-(-1))+h(i,0).I_0(12-i,1-0)+h(i,1).I_0(12-i,1-1)$$

$$=\Big[h(-1,-1).I_0(12+1,2)+h(-1,0).I_0(12+1,1)+h(-1,1).I_0(12+1,0)$$

$$+h(0,-1).I_0(12,2)+h(0,0).I_0(12,1)+h(0,1).I_0(12,0)$$

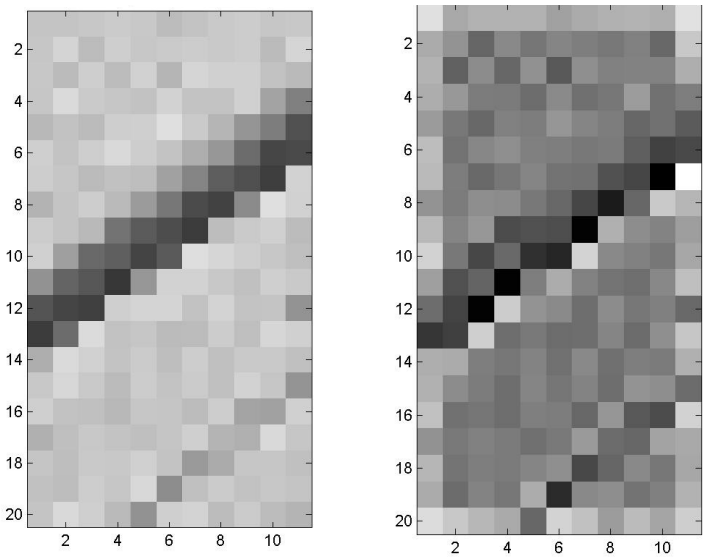
$$+h(1,-1).I_0(12-1,2)+h(1,0).I_0(12-1,1)+h(1,1).I_0(12-1,0)\Big]$$

$$= 0x209 + (-1)x218 + 0x174 + (-1)x219 + 5x108 + (-1)x60 + 0x64 + (-1)x69 + 0x84$$

$$= -26$$

Note: here, the output luminance values can be negative or higher than 255. To display the image result, it is thus necessary to scale the gray levels to the full range [0, 255].

Here is the display of the arrays before (on the left) and after (on the right) filtering:



The enhancement filter which is performed is indeed the sum of an Identity filter and a Laplacian filter (useful for edge detection):

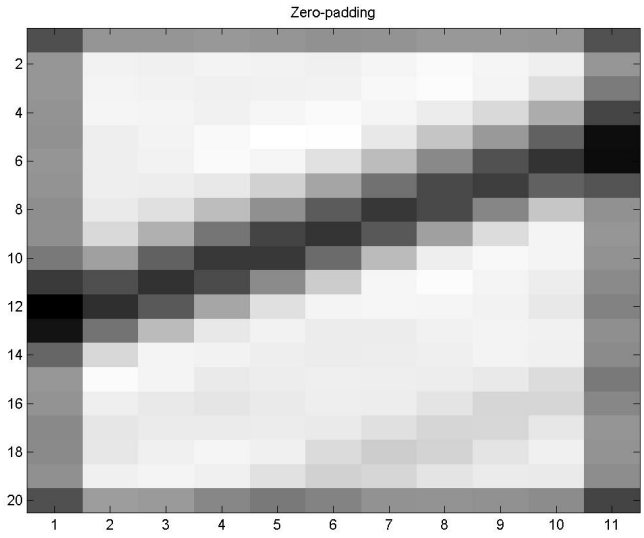
$$\underset{\text{Identity}}{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}} + \underset{\text{Laplacian}}{\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}} \equiv \underset{\text{Enhancement}}{\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}}$$

This filter enhances the contrast of the objects (like the fishing boom and rope) in the original image.

3 – A lot of methods are defined to compensate the border distortions: zero-padding, border replication, etc.
For more information about these boundary conditions, refer to the correction of the exercise: “Filtering” (chapter3).
Here, we will only show some results for the low-pass filtering with two methods used to remove these border distortions.

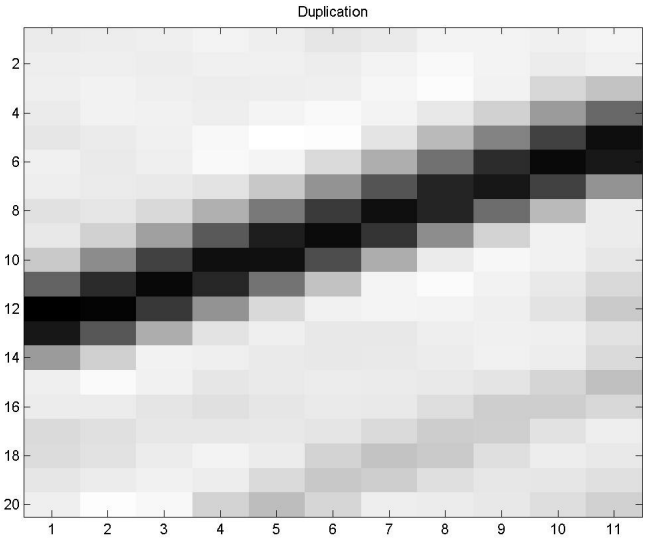
- Zero-padding:

That is the simplest method: the "off-the-edge" neighborhood is set to 0. Zero-padding can result in a dark band around the edge of the image:



- Replication:

The "off-the-edge" neighborhood is set to the value of the nearest border pixel in the original image (to filter using border replication, pass the optional argument ‘replicate’ to the Matlab function ***imfilter***). The border replication eliminates the zero-padding artifacts around the edge of the image.



Chapter 3

Fundamental of Image Processing

Linear Filtering

Examples of typical filter kernels

Support size (3×3) : the most common and the simplest

❶

$$h = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad h = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \qquad h = \begin{bmatrix} 1 & -3 & 1 \\ -3 & 9 & -3 \\ 1 & -3 & 1 \end{bmatrix}$$

Average Binomial Contrast enhancement

❷

$$h = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad h = \begin{bmatrix} -1 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \qquad h = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Horizontal Sobel Oblique gradient Laplacian

filters ❶ : $H(0,0) = \sum_i \sum_j h(i,j) = 1$

Filters ❷ : $H(0,0) = \sum_i \sum_j h(i,j) = 1$

The observation of the convolution kernel h gives some important characteristics about the kind and the effects of the filter used:

- if all the elements are positive: h is low pass-filter. It performs a weighted average. For a 8-bit pixel coding, when the result is more than 255, it is set to 255 by thresholding. This kind of filtering will smooth the image, it is useful to reduce noise in images;
- if some elements are positive and some others are negative: a partial or complete differentiation is performed. The convolution kernel corresponds partially or completely to the behavior of a high-pass filter. A high-pass filter can be applied to sharpen the image, therefore preserving boundary detail.

Some examples of typical filter kernels are presented here:

- low-pass filters: a mean (‘average’) filter and a binomial filter that smooth the image. Compared to the simple averaging image, edge enhancement with the binomial filter is notably smoother;
- high-pass filters: a contrast enhancement filter, oriented derivative filters (horizontal Sobel, oblique gradient) and a non-oriented derivative filter (Laplacian).

Example:

For example, a “binomial” low-pass filter (3×3) and an “oblique gradient” high-pass filter are applied to the image *LENA*:

After using the binomial low-pass filter, the edges and the textures are smoothed. One visualizes these effects on the shadows of the shoulder and the face.

After using the gradient oblique high-pass filter, the diagonal edges of the image are enhanced.

Filtering objectives

- Noise reduction
- Low-pass filtering before sub-sampling
- Contrast enhancement

Common filters:

"Low-pass" (with unitary DC gain)

$$H_1 = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad H_2 = \frac{1}{10}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad H_3 = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Very low Low Medium

(spatial averaging)

"High-pass" (with unitary DC gain)

$$H'_1 = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \qquad H'_2 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix} \qquad H'_3 = \begin{bmatrix} 1 & -2 & 1 \\ -2 & 5 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

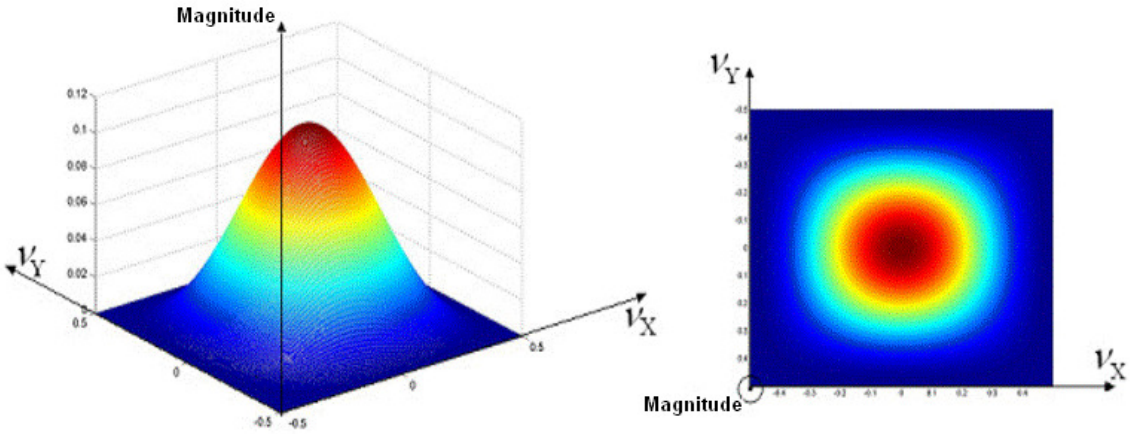
The spatial filtering can be divided into 2 fundamental categories:

- **Low-pass filtering that reduces luminance variations**. Therefore it smooths the image content and it reduces abrupt changes in intensity. The low-pass filter is typically used to reduce noise effects and to remove the spatial high frequency content of the image (details of the image) before sub-sampling (to remove aliasing effects or parasite frequencies cf. chapter 1) ;

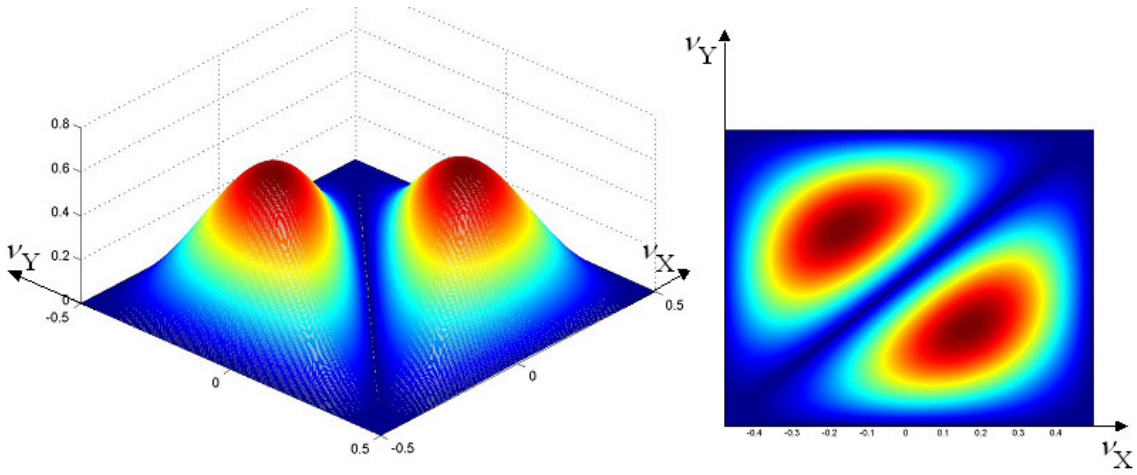
- **High-pass filtering that enhances abrupt luminance variations which characterize typically the object edges and the details in an image**. For example, a high-pass filter with unitary DC gain can enhance the contrast of the original image.

The presented convolution kernels are only some examples of typical filters. The user can choose or define specific filters that he/she needs to perform image processing.

The Fourier transform of the impulse response of a linear 2D filter gives the frequency response of the filter. The bottom figure presents the frequency response of the binomial filter H_3 (image on the left):



The two-dimensional spectrum of an image or a filter represents the magnitude of the Fourier transform according to spatial horizontal frequencies "ν_X" and vertical frequencies "ν_Y".
Usually the Fourier spectrum of an image is displayed on a plane (representation by a top view of the magnitude: image on the right).
We can observe on the Fourier spectrum that the binomial filter has the behavior of a low-pass filter for the horizontal and vertical frequencies. The variations of luminance along the horizontal and vertical spatial axes are thus reduced: the image is "smoothed".
Here is the Fourier spectrum of the « oblique gradient » low-pass filter which was used on a part of the image *LENA*:

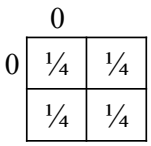


The filter lets only diagonal high frequencies pass. The variations of luminance along the diagonal directions are enhanced on the filtered image. The diagonal edges and details are highlighted.

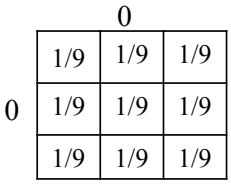
The concepts related to the 2D Fourier transform of an image or a filter and to spatial frequencies are more fully developed in the exercise "FFT" of this same chapter.

Examples of low-pass filters

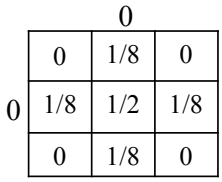
- Three examples of convolution kernels for image low-pass filtering.
- Unitary DC gain.



Filter 1



Filter 2



Filter 3

Let us perform three different low-pass filters on the grayscale image *Lena*. These filters have all a unitary DC gain. For each of them, the kernel origin h(0,0) is indicated by the horizontal and vertical marks "0".

Note: the filters 1 and 2 are respectively 2×2 and 3×3 average filters. After image filtering, the output luminance value of each pixel is the mean value of all the neighboring luminance values (size 2×2 for the filter 1 and 3×3 for the filter 2): the image is thus "smoothed".

Original image *Lena*



Filters (1) (2) (3)

At the top is the original image. At the bottom, from the left to the right, the images which are respectively filtered by low-pass filters 1, 2 and 3. One can distinguish some smoothing effects on the shadows from the face of *Lena*, however these effects are hardly visible to the naked eye. The human eye is in fact less sensitive to the spatial high frequencies and it does not distinguish that removing the high frequencies has involved loss of details from the original image. The low-pass filtering thus involves only few losses in visual quality and it allows you to sub-sample without aliasing effects.

Separable 2D filters: convolution of two "1D filters"

$$[\,h_{2\text{-}D}\,] \equiv [\,h_{1\text{-}D}^{(V)}\,] \otimes [\,h_{1\text{-}D}^{(H)}\,]$$

$I_s(m,n) = h(m,n) \otimes I_e(m,n)$: 2D linear filtering

$I_s(m,n) = h_V(m) \otimes [h_H(n) \otimes I_e(m,n)]$: convolution of two 1D filters

$I_s(m,n) = h_V(m) \otimes I'(m,n) = h_H(n) \otimes I''(m,n)$: commutative operator

With:

$I'(m,n) = h_H(n) \otimes I_e(m,n)$ row by row 1D filtering

$I''(m,n) = h_V(m,n) \otimes I_e(m,n)$ column by column 1D filtering

- ***Constraint*** *: columns and rows of $h_{2\text{-}D}$ must have proportional elements.*

Example 1

$$[a\ b\ c] \otimes \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} a\alpha & b\alpha & c\alpha \\ a\beta & b\beta & c\beta \\ a\gamma & b\gamma & c\gamma \end{bmatrix}$$

Example 2

$$[a\ b\ a] \otimes \begin{bmatrix} a \\ b \\ a \end{bmatrix} = \begin{bmatrix} a^2 & ab & a^2 \\ ab & b^2 & ab \\ a^2 & ab & a^2 \end{bmatrix}$$

A 2D filter is "separable" if the kernel $[h_{2D}]$ can be decomposed into two 1D kernels (which are applied successively). The filtering is performed in one dimension (rows), followed by filtering in another dimension (columns): $[h_{2D}] = [h_{1D}^{(V)}] \otimes [h_{1D}^{(H)}]$, where the symbol $\otimes$ stands for convolution operator.

The rows and the columns in the original image are thus separately filtered. Whatever the first 1D filtering performed, the output image $I_S(m, n)$ is still the same. To be separable, a 2D filter must have proportional elements on the rows and the columns: mathematically that is seldom true, however several usual 2D filters are separable.

Examples of separable filters

- Averaging

$$\frac{1}{3}\begin{bmatrix}1 & 1 & 1\end{bmatrix} \otimes \frac{1}{3}\begin{bmatrix}1\\1\\1\end{bmatrix} = \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\1 & 1 & 1\\1 & 1 & 1\end{bmatrix}$$

- Contrast enhancement

$$\begin{bmatrix}-1 & 3 & -1\end{bmatrix} \otimes \begin{bmatrix}-1\\3\\-1\end{bmatrix} = \begin{bmatrix}1 & -3 & 1\\-3 & 9 & -3\\1 & -3 & 1\end{bmatrix}$$

- Binomial

$$\frac{1}{4}\begin{bmatrix}1 & 2 & 1\end{bmatrix} \otimes \frac{1}{4}\begin{bmatrix}1\\2\\1\end{bmatrix} = \frac{1}{16}\begin{bmatrix}1 & 2 & 1\\2 & 4 & 2\\1 & 2 & 1\end{bmatrix}$$

- Horizontal Sobel

$$\begin{bmatrix}-1 & 0 & 1\end{bmatrix} \otimes \begin{bmatrix}1\\2\\1\end{bmatrix} = \begin{bmatrix}-1 & 0 & 1\\-2 & 0 & 2\\-1 & 0 & 1\end{bmatrix}$$

- Advantage: low complexity
  - 2D Filter: $M \times N$ multiplications and additions per pixel (MAP)
  - 2D separable Filter: $M + N$ multiplications and accumulations per pixel

Here are some examples of separable 2D filters with the decomposition into 1D filters. The complexity is low for 2D separable filters because the number of operations (multiplications and additions) is reduced, thus the computation time is faster.
Typically if the kernel size is M×N, we need only (M+N) multiplications and (M+N-2) additions instead of M.N multiplications and M.N-1 additions for a non-separable 2D filter. Often the term “MAP” is preferred (multiplications and accumulations per pixel): there are (M+N) MAP for a separable filter instead of M.N MAP for a non-separable filter.

Exercise Chapter 3 – Fast Fourier Transform (FFT)

In this exercise, you will visualize the (spatial) frequency response for some examples of images.
Before starting, load and unzip the file "***fft.zip***" which contains the scripts you will need throughout this exercise.

Discrete Fourier Transform

1 – Add your own working folder paths to the path list in the path browser then open and analyze the script ***fft2d_sinus.m***. Use the Matlab help (command ***helpwin***) to understand how the Matlab functions that we are using work. What is the kind of the 2D input signal?

2 – Modify the spatial period value (called ***period*** in the script): enter the values 4, 8, and 16 (orientation 0). For all these period values, note the magnitude and the normalized frequencies of the lines down. Change also the amplitude and the DC offset (***dc***) of the 2D signal. Interpret the results. Next, test the period value 17 and interpret again.

3 – Use the script ***fft2d_resolution.m*** to compute a higher frequency resolution FFT. Change now the 2D signal orientation: test pi/2 then pi/4 (period = 16 and period = 16*sqrt(2)).

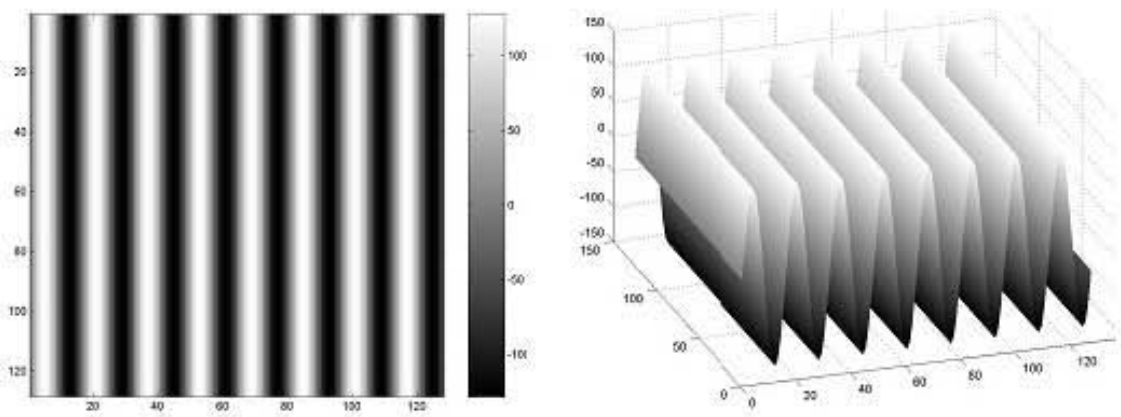
4 – Run the scripts ***fft2d_square.m*** and ***fft2d_checkerboard.m***. Change the parameters and interpret the results. By using the script ***fft2d_sinus.m***, write a script that displays a natural image and its frequency response. Run this script on several images.

Solution to the exercise on FFT

1 - The script *fft2d_sinus.m* allows you to visualize the discrete Fourier transform of a 2D sinusoidal signal.
First, we define the characteristics of the 2D signal: image size, spatial period, orientation, amplitude, and DC offset. Then the function *generation_sinus* generates a 2D signal from these characteristics (the function *generation_sinus* has been written for you).

```
Sizel = 128 % image size
period = 16 % spatial period
orientation = 0
amplitude = 128
dc = 0 % DC offset
iml= generation_sinus(sizel,period,orientation,amplitude,dc);
```

The image of the sinusoid is then visualized in two manners: in a view from the top (2D vision with the function *imagesc*) and in perspective view (3D vision with the function *surf*):



We note that the horizontal direction of the sinusoid involves the generation of vertical bands on the image (left). Once the sinusoid image is computed and stored in *im1*, we compute its fast Fourier transform (FFT) with the Matlab function *fft2*.

```
spectruml = fft2(iml)/(sizel*sizel);
```

Then we use the function *fftshift* to shift the zero-frequency component (0, 0) of the fast Fourier transform to the center of the spectrum.

```
spectruml = fftshift(spectruml);
```

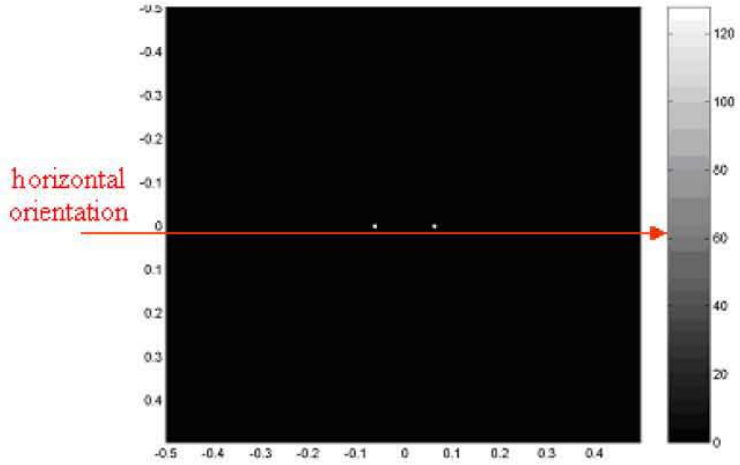
The normalized vectors represent the horizontal and vertical spatial frequencies. Here these vectors have the same length and the same components, giving:

```
vt =(- sizel/2:1:( sizel/2-1))/ sizel;
```

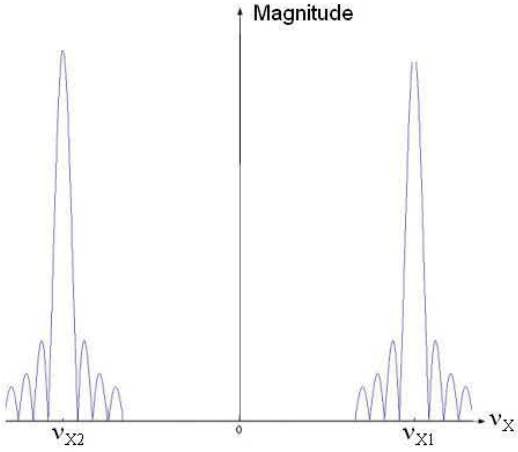
The absolute value (modulus) of the spectrum can be displayed with the command:

```
imagesc(vt,vt,(abs(spectrum1)));
```

Here is the image of the spectrum modulus:



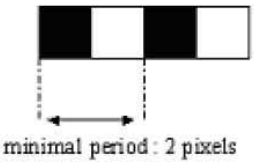
We see two white points. We will show later that these points are not two delta functions but they are the two main lobes of a sinus cardinal function because the 2D sinusoidal function is ***bandlimited***:



Note that the orientation is horizontal along these two points (it is orthogonal to the vertical orientations of the bands in the original sinusoid image).

Note:

The spatial period is the minimal number of pixels between two identical patterns in a "periodic" image. The minimal spatial period in an image is thus two pixels:



The maximal normalized spatial frequency ν_{max} is thus equal to:

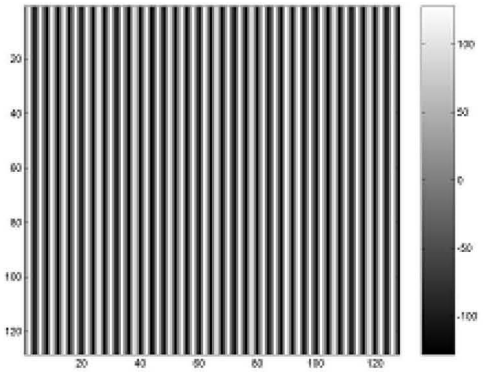
$$\nu_{max} = \frac{1}{\text{minimal period}} = \frac{1}{2}$$

We can plot the image frequency response along the normalized spatial frequencies which belong to the range [-0.5, 0.5].
Let T be the period. The corresponding normalized frequency ν_{norm} is defined by: $\nu_{norm} = 1/T$.

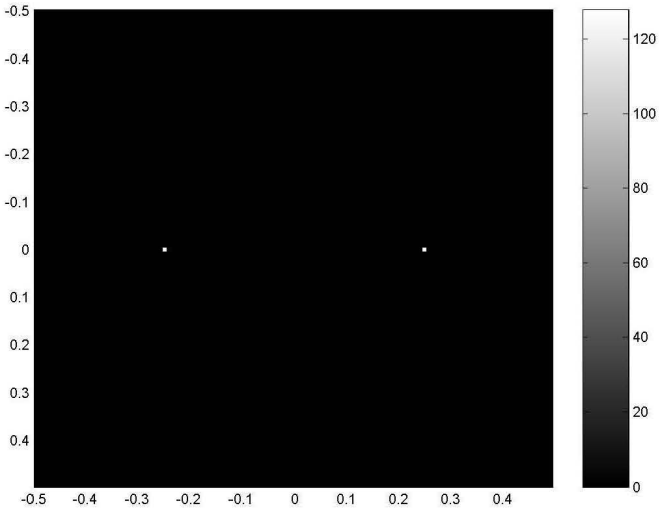
2 -

- Changing the period:

By decreasing the spatial period of the sinusoid (value = 4), we visualize the image below:



On the image, we observe one line per four pixels.
The magnitude image is displayed below:



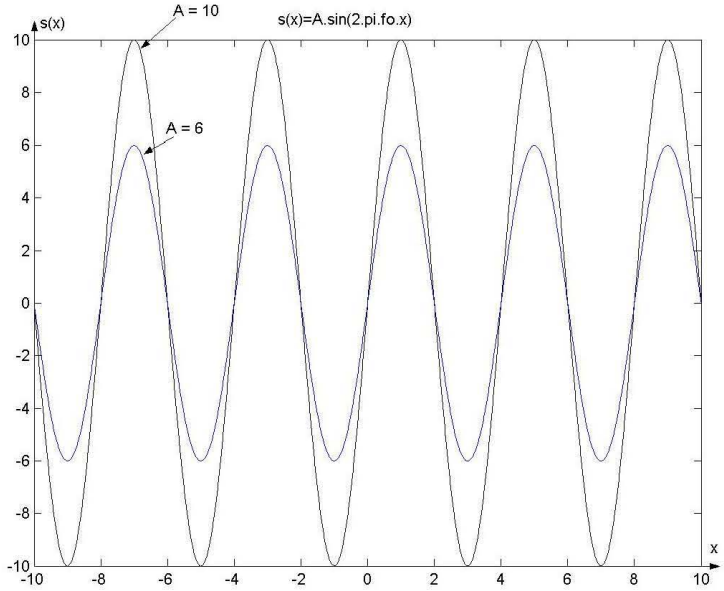
The two white points are located at the spatial frequency coordinates:

$$\{\nu_{1X}= -0.25,\ \nu_{1Y}= 0\} \text{ et } \{\nu_{2X}= 0.25,\ \nu_{2Y}= 0\}$$

When the orientation of the 2D sinusoid is zero, we can represent one line of this 2D signal by a 1D sinusoid signal which propagates along the horizontal axis (Ox).
The absolute value of the horizontal spatial frequency ν_{1X} is equal to the one of ν_{2X}. In fact: $\nu_{1X} = -\nu_{2X}$ thus $|\nu_{1X}| = |-\nu_{2X}|$. This absolute value is the inverse of the spatial period: $\nu_{1X}= -1/T$ and $\nu_{2X}= 1/T$.

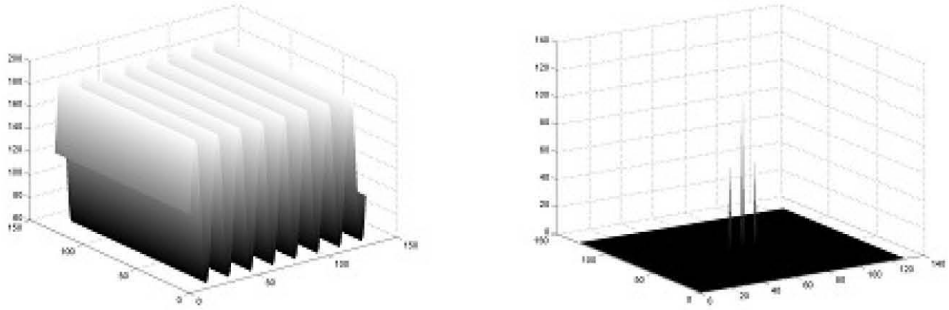
- Changing the amplitude

The sinusoid values belong to the range [-A, A], where A stands for the amplitude. The change of the amplitude A influences the magnitude values of the two white points that we had previously noticed on the spectrum image.



- Changing the DC offset:

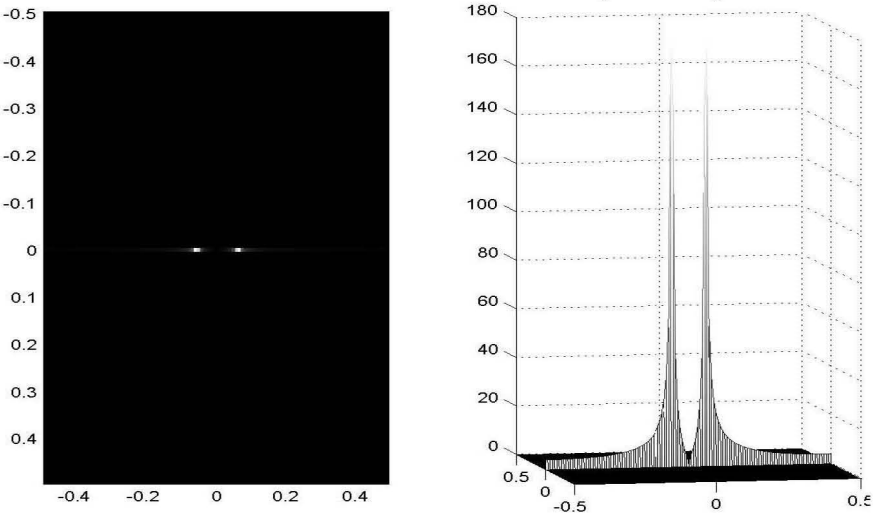
By increasing the DC offset V_M, we visualize the images below:



The spectrum modulus has now a new frequency component at the center (image on the right) adding to the two preceding frequency components. This component corresponds to the spectrum DC gain located at the spatial frequency coordinates $\{\nu_X = 0, \nu_Y = 0\}$. This DC gain is equal to the DC offset of the sinusoid.

- Period T = 17:

We compute the fast Fourier transform with the spatial period 17:



On the Fourier plane (left), the two white points are "spread out". On the right image, note that the frequency response is not composed of only two impulses any more. The frequency response looks like the envelope of a sinus cardinal function.

Let us consider a 1D sinusoidal signal to explain this result. This signal is called f(x) and defined by:

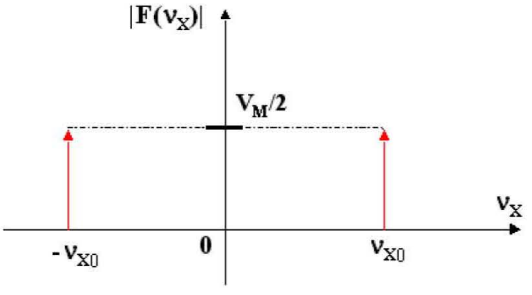
$$f(x) = V_M . \sin(2.\pi.f_0.x)$$

The spectrum $F(\nu_X)$ of this sinusoidal signal is thus defined by:

$$F(\nu_X) = V_M . \left(\frac{1}{2j} . \delta(\nu_X - \nu_{X0}) + \frac{j}{2} . \delta(\nu_X + \nu_{X0}) \right)$$

where "j" stands for the complex number such as $j^2 = -1$, and $\nu_{X0} = 1/T$.

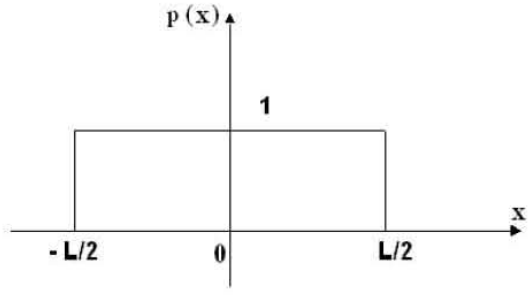
The spectrum modulus $|F(\nu_X)|$ is plotted on the figure below:



This spectrum is computed for a 1D sinusoidal signal of infinite support(x ranging from $-\infty$ to $+\infty$).
In practice we cannot represent an image on an infinite support, therefore the 1D sinusoidal signal is multiplied by a rectangular pulse p(x). This rectangular pulse is defined by:

$$p(x) = \begin{cases} 1 \text{ if } x \in [-L/2; L/2] \\ 0 \text{ else} \end{cases}$$

Here is the graph of the function p(x):

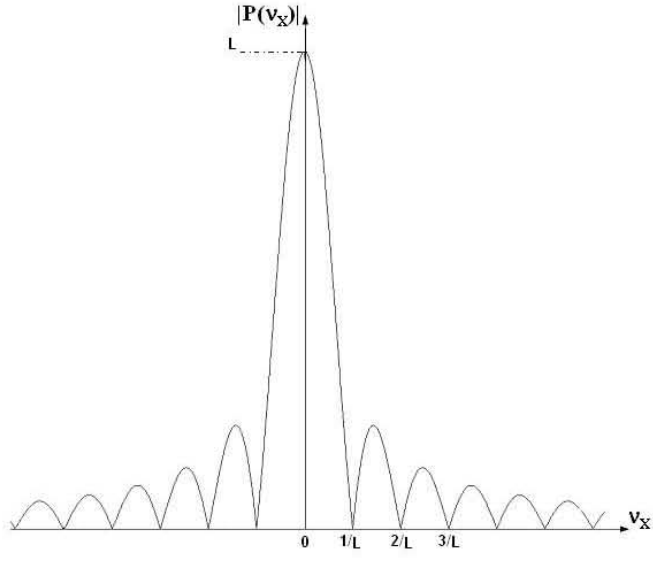


The spectrum $P(\nu_X)$ of this rectangular pulse function is defined by:

$$P(\nu_X) = L.\mathrm{sinc}(L.\pi.\nu_X)$$

Where "sinc" stands for sinus cardinal: sinc(x) = sin(x)/x.
In the script ***fft2d_sinus.m***, the size of the input image is 128 × 128 pixels. The sinusoidal signal propagates along the axe (Ox), therefore L = 128.

The spectrum modulus $|P(\nu_X)|$ is displayed on the figure below:



Note that the length of the lobes is 1/L (except for the main lobe whose length is 2×1/L).

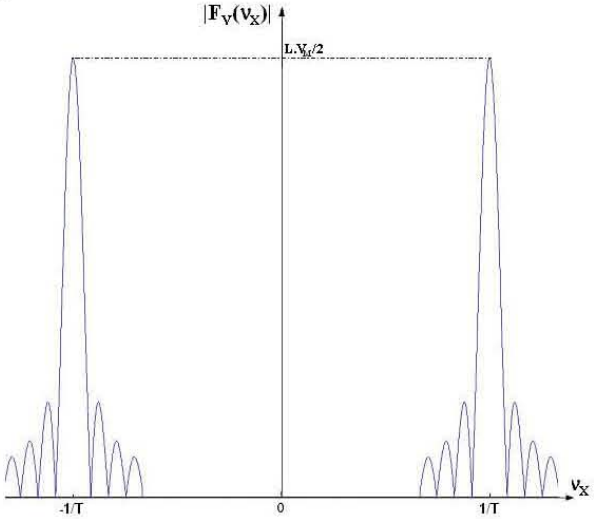
The visualized sinusoid $f_V(x)$ is thus the function defined by:

$$f_V(x) = f(x).p(x)$$

Its spectrum $F_V(\nu_X)$ is given by the relation:

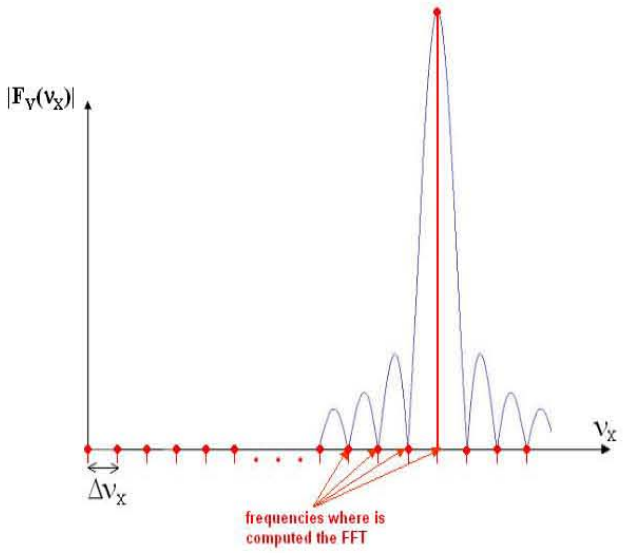
$$F_V(\nu_X) = F(\nu_X) \otimes P(\nu_X) = \frac{1}{2}.V_M.\left(\frac{1}{j}.P(\nu_X-\nu_{X0})+j.P(\nu_X+\nu_{X0})\right)$$

Let us consider the T-periodic sinusoidal signal $f_V(x)$ has an amplitude V_M, and is band-limited on a support whose length is L. Here, its frequency response is plotted:



Moreover, to represent the spectrum of this sinusoid, a fast Fourier transform is computed. The spectrum is thus computed for a finite number of its points. By default, with the Matlab function ***fft2***, this number of points is equal to the number of pixels in each direction (horizontal and vertical).
Here the number of pixels is the same one according to two directions. After FFT computation the spectrum modulus $|F_{Vs}|$ is thus represented by 128 points of the spectrum modulus $|F_V|$ according to the horizontal spatial frequencies. The horizontal frequency resolution $\Delta\nu_X$ is then defined by: $\Delta\nu_X = 1/128$.

Note that $1/L = \Delta\nu_X$: on the spectrum of the sinusoidal signal; the length of any lobe (except the main lobe) is equal to the horizontal frequency resolution.
If the period T of the 1D sinusoidal signal is inversely proportional to the horizontal frequency resolution (i.e. $1/T = k.\Delta\nu_X$, where "k" is a non-null natural integer), the spectrum modulus $|F_{Vs}|$ represents only the two main lobes of the spectrum modulus $|F_V|$ (the other lobes are set to 0):

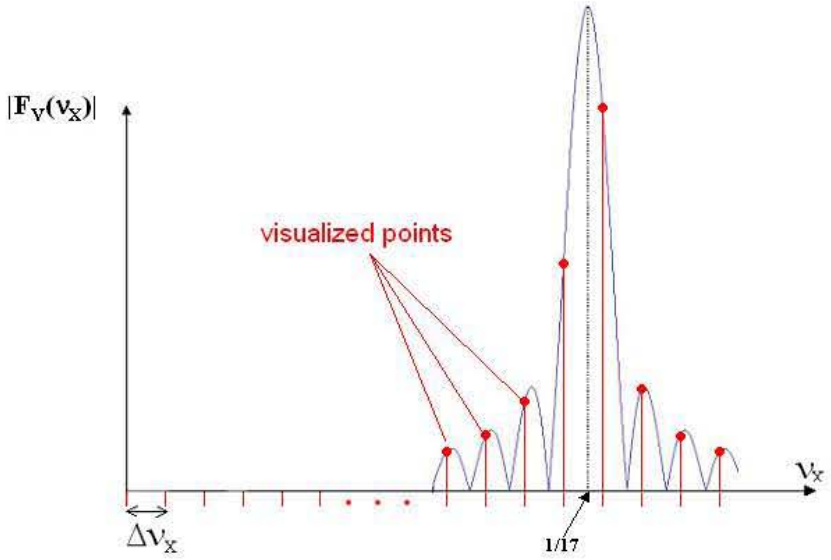


Then for the periods:

- T = 4, $\frac{1}{4}=32\times\frac{1}{128}$,
- T = 8, $\frac{1}{8}=16\times\frac{1}{128}$,
- T = 16, $\frac{1}{16}=8\times\frac{1}{128}$,

The condition $1/T = k.\Delta\nu_X$ is thus true for these three periods. We visualize only the two maximal values of the main lobes. The other components are zeroing therefore the spectrum modulus looks like two delta functions (cf. 1).

When the period is set to 17, the condition $1/T = k.\Delta\nu_X$ is not true. We visualize thus some samples of the other lobes:

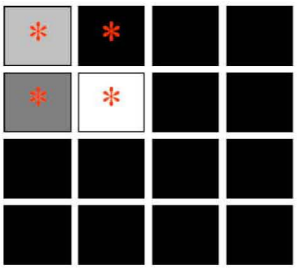


If you want to visualize the other lobes whatever the period is, you must increase the spatial frequency resolution to satisfy the condition: $\Delta\nu_X < 1/L$.

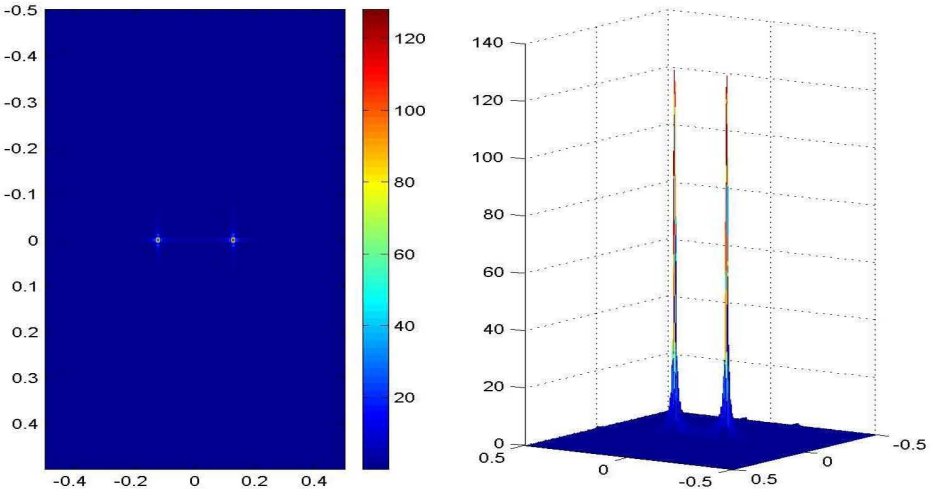
3 - The script ***fft2d-resolution.m*** allows you to increase the **frequency resolution**. We choose along one direction (the horizontal one then the vertical one, or conversely) the number of pixels N_2 to compute the FFT. This number must be superior to the number N_1 of points in the original image along the same direction (horizontal or vertical). The value of the padded points is zero.

Let us consider a simple (2×2) grayscale image:

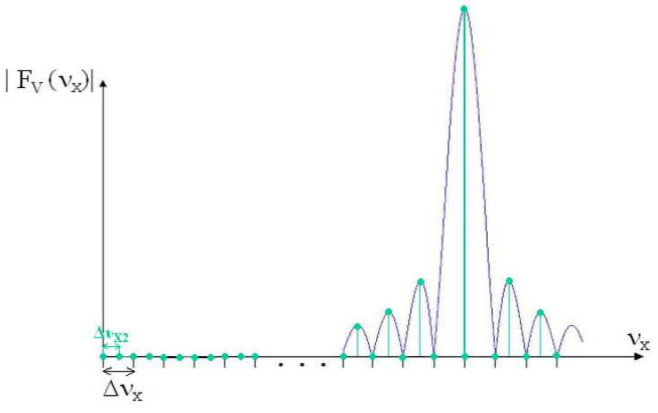
We want to increase the frequency resolution to compute the fast Fourier transform with 4 points along each direction instead of the two original points. The fast Fourier transform is thus computed on the following image (N_2=4):



The four original pixels are marked by a red star. The other black pixels (zero padding) have been added to create a (4×4) image.
When you want to compute the FFT in Matlab with a higher frequency resolution, you can pass the optional argument N_2 to the Matlab function ***fft2***: fft2(im1,N_2,N_2). Here are the results obtained with a 2D 8-periodic sinusoid:



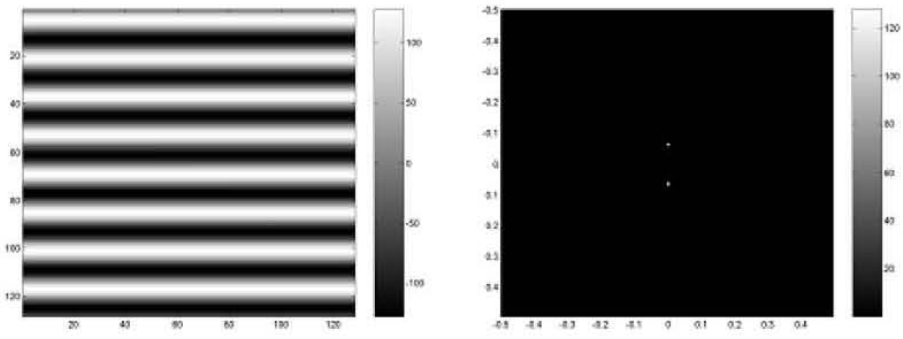
By increasing the frequency resolution, all the spectrum lobes are sampled even if the period is like 1/T = k.$\Delta\nu_X$.
Let us consider for example a ×2 frequency resolution: $\Delta\nu_{X2}$= 2.$\Delta\nu_X$:



All the lobes are now represented by a non-zero value (maximal value here).

Changing the orientation:

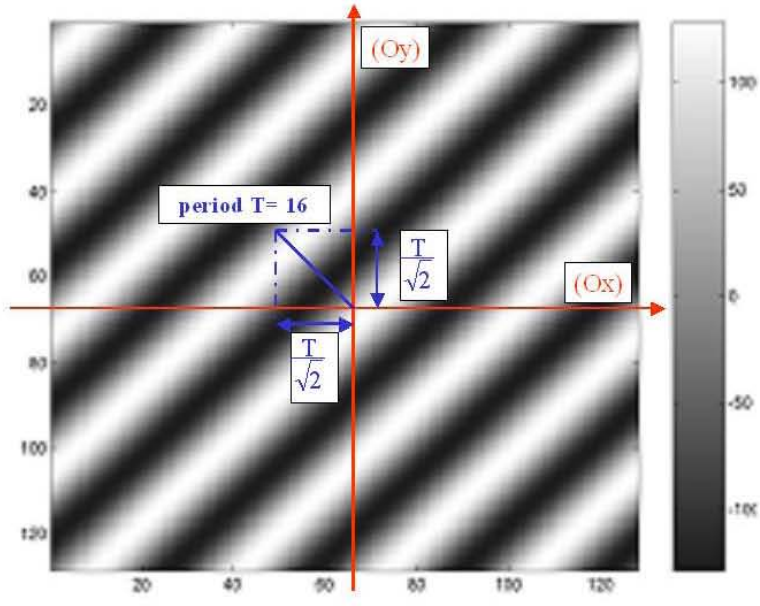
Here are the results obtained after having set the orientation to $\pi/2$ (period = 16 pixels):



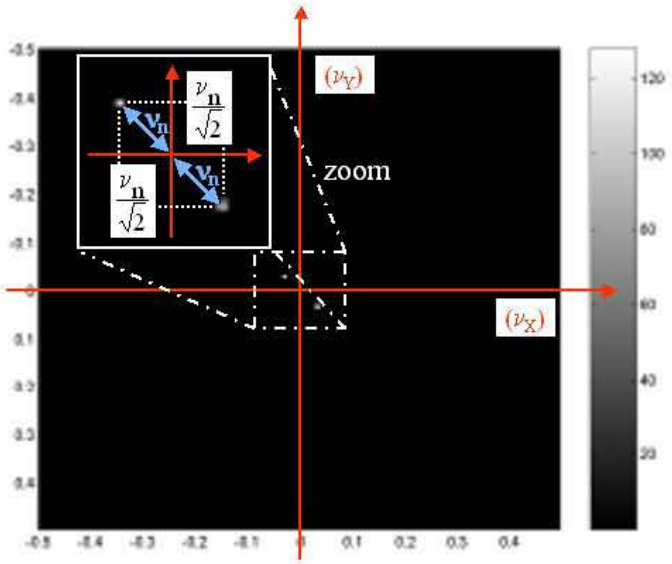
As expected:

- The propagation is perpendicular to the original one (the bands are now horizontal in the image on the left);
- On the right, the two points of the spectrum define a direction which is perpendicular to the direction in the original spectrum (orientation = 0) and to the bands of the associated left image.

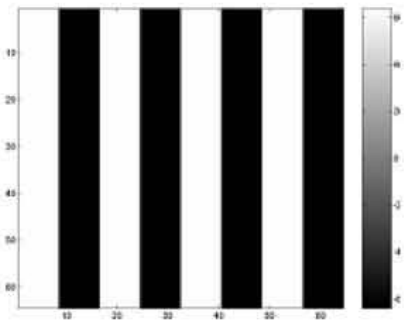
In the same way, by setting the orientation to $\pi/4$ (period T = 16 pixels), the propagation orientation of the sinusoid is rotated by $-\pi/4$. We visualize horizontal and vertical periodic patterns. These ones also create a diagonal periodic pattern whose period is equal to 16 pixels. The horizontal and vertical periods are thus equal to $\frac{T}{\sqrt{2}}$ pixels:



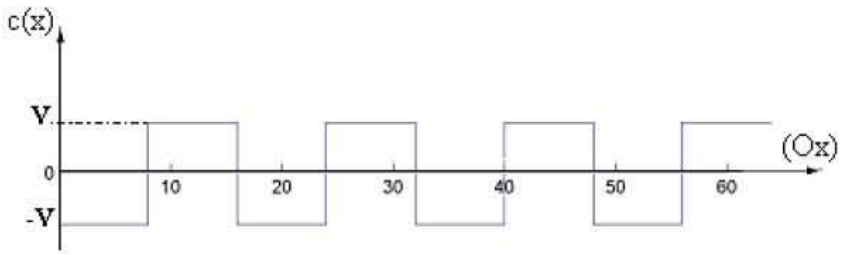
On the image below, the spectrum modulus is represented by two white points which are located at the coordinates: $\left(-\frac{\nu_n}{\sqrt{2}};\frac{\nu_n}{\sqrt{2}}\right)$ and $\left(\frac{\nu_n}{\sqrt{2}};-\frac{\nu_n}{\sqrt{2}}\right)$, where $\nu_n=\frac{1}{T}$. These two points define a direction which is perpendicular to the bands in the associated original image. The 2D image of the frequency response gives several pieces of information about the spatial orientation of the original signal.



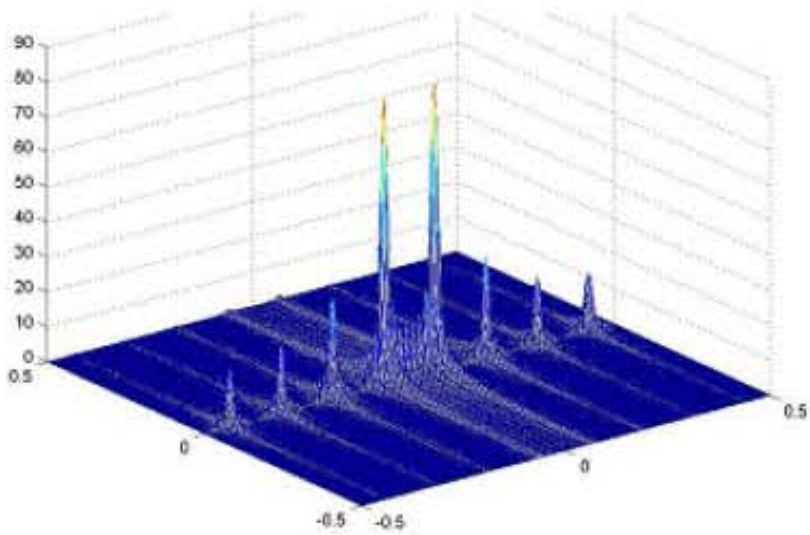
4 - By launching the script ***ff2d_square.m***, you visualize the following image:



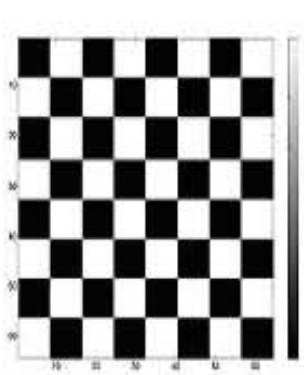
The 2D signal is a square signal which propagates along the horizontal axis (Ox). We can thus represent one line of this 2D signal by a 1D square signal which propagates along the horizontal axis (Ox):

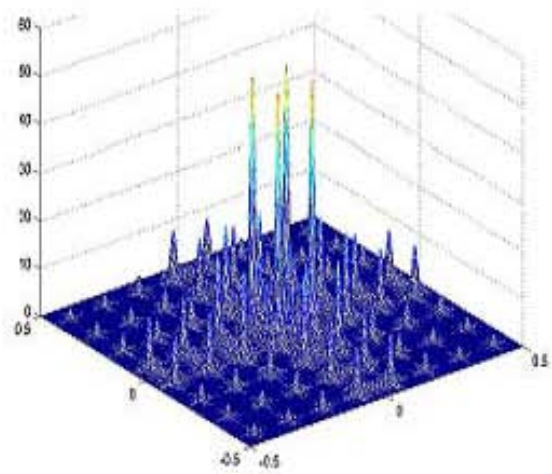


Here is the frequency response of the 2D band-limited square signal. As expected the representation of the spectrum modulus is given by the product of two sinus cardinal functions:



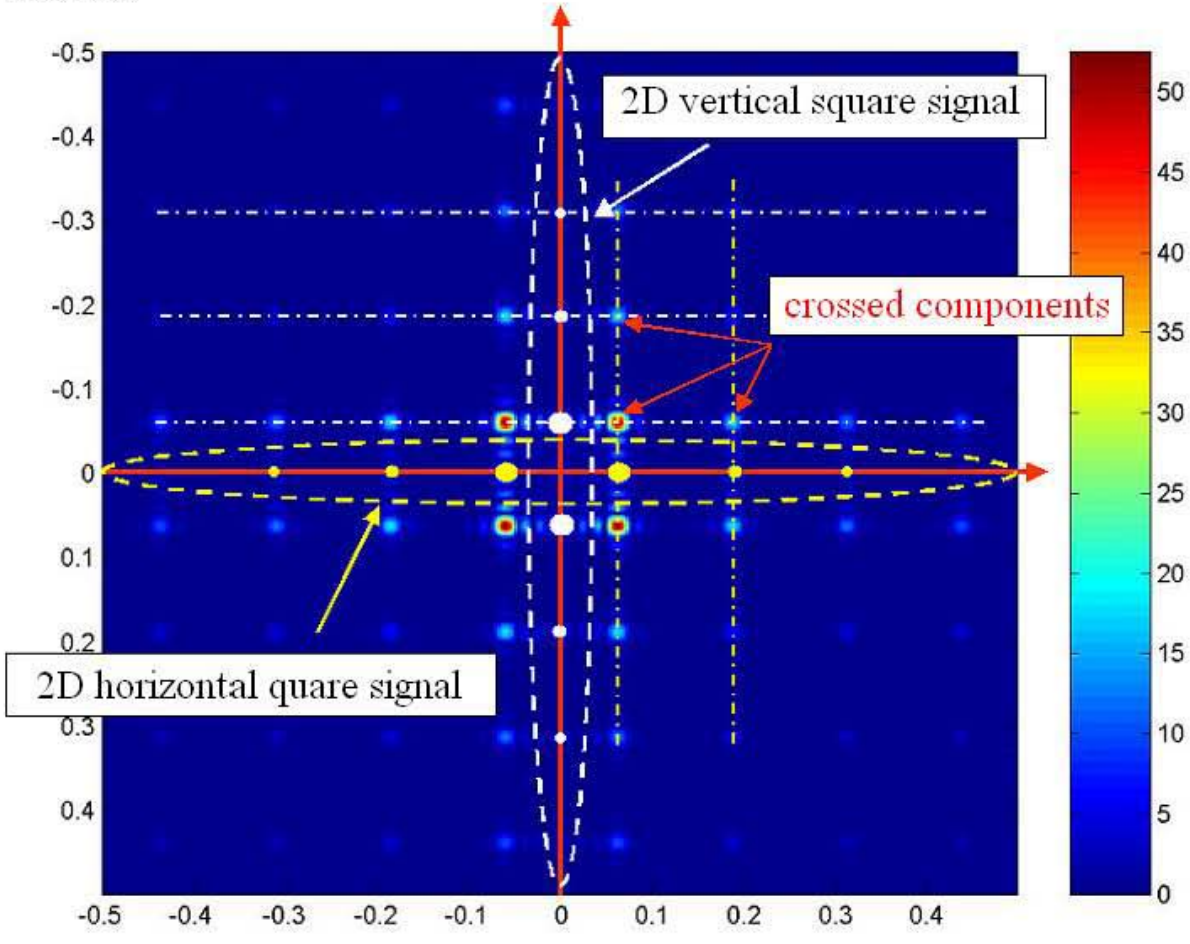
By launching the script *ff2d_checkerboard.m*, we obtain the following results (period = 16):





We perform the XOR-operator between two orthogonal square waves to create the square checkerboard (you can also use the Matlab function *checkerboard*)
When you visualize the frequency response of this image, you can notice crossed magnitudes which are similar to the one obtained with a single square wave.
In addition, note that the checkerboard patterns (on the left) are periodic along the diagonal directions of the image.

The frequency response of the checkerboard image is displayed on the image below (period = 16). Here we plot the spectrum modulus of the two basic square signals: the horizontal one and the vertical one (warning: they do not appear when you display the frequency response). The spectrum components of the checkerboard image are equal to the crossed components of the two frequency responses which are associated with the two basic square signals.

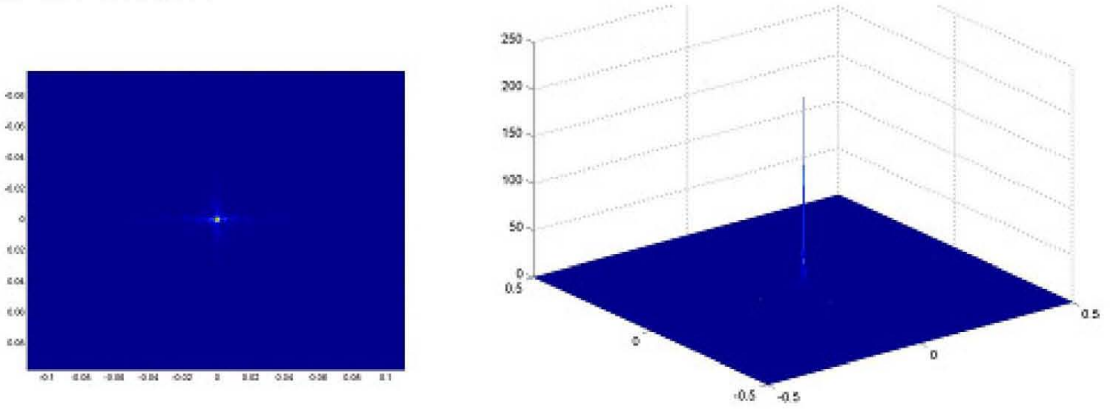


5 - Here is an example of a script for displaying the frequency response of the image *CLOWN_LUMI*:

```
im1=imread('CLOWN_LUMI.BMP') ;
% colormap for displaying in gray levels
figure(1);
imagesc(im1);
map = 0:1/255:1;
map = [map',map',map'];
colormap(map);
% FFT
[size1, size2]=size(im1(:,:,1)) ;
nb_point1 = 2*size1;
nb_point2 = 2*size2;
spectrum1 = fft2(im1,nb_point1,nb_point2) /(size1*size2);
spectrum1 = fftshift(spectrum1);
% To reduce the DC component
spectrum1 = 2*spectrum1;
spectrum1(size1/2+1,size2/2+1)=spectrum1(size1/2+1,size2/2+1)/2;
% normalized frequency vectors
vt1=(-size1/2:size1/nb_point1:(size1/2-size1/nb_point1))/size1;
vt2=(-size2/2:size2/nb_point2:(size2/2-size2/nb_point2))/size2;
```

```
% To display the spectrum modulus
figure(2);
imagesc(vt1,vt2,(abs(spectrum1)));
% 3-D representation
figure(3)
[X,Y]=meshgrid(vt1,vt2);
mesh(X,Y,abs(spectrum1));
```

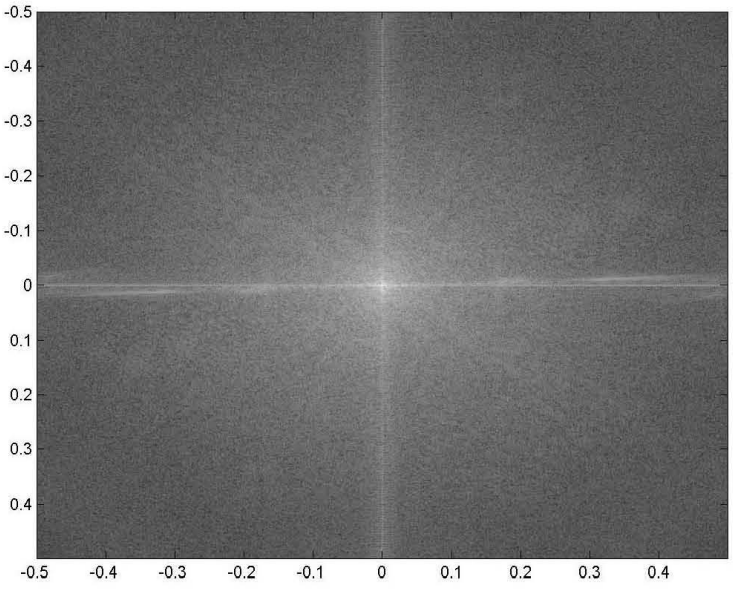
Here is the result:



In natural images the DC component is very often higher than the other frequency components (middle and high frequency components). We prefer generally to display the natural logarithm of the spectrum modulus instead of the spectrum modulus itself. Here are the commands to type to display the natural logarithm of the "*Clown_lumi.bmp*" image spectrum modulus:

```
% We display the spectrum modulus
figure(2);
imagesc(vt1,vt2,(log10(abs(spectrum1))));
colormap(map) ;
```

Here is the result:



Exercise Chapter 3 – Linear filtering

In this exercise, you will use Matlab to perform a linear filtering with various techniques. Before starting, load and unzip the file "***linearFilter.zip***" which contains scripts you will need throughout this exercise.

Linear Filtering in frequency domain then in spatial domains

1 – Open the script ***lowpass.m***. Use the F9 key to launch the first part of the script (that computes the transfer function of the filter). Modify the support size and run the script again with F9. Explain the results.

2 – In the second part of the script, we process an image with a spatial filtering. This filtering is performed by using the Matlab function ***conv2*** (2D convolution) with two different optional parameters. Compare the results and conclude (you can change the support size then you must run the first part again).

3 – In the third part of the script, we use the Matlab function ***imfilter*** with two different optional parameters. Compare the pixels on the right edge of the image and conclude. Use now the last part of the script to perform the frequency filtering instead of the spatial filtering. Observe the results.

4 – Run the script ***filtering_spatialvsfreq.m***. You will compare the computation time of the spatial filtering with the frequency filtering. Run the script again after having modified the support sizes and conclude.

5 – Analyze then run the script ***highpass.m***. Interpret the results.

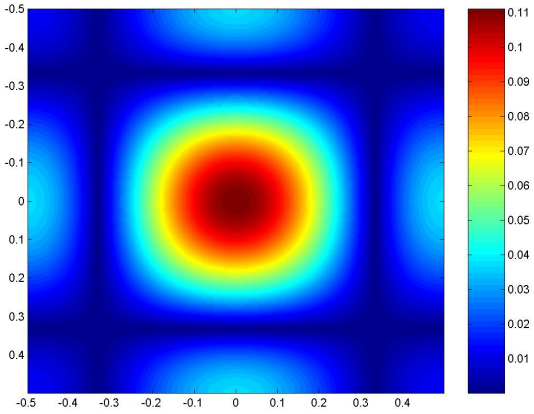
Solution to the exercise on linear filtering

Linear filtering in the frequency domain and in the spatial domain

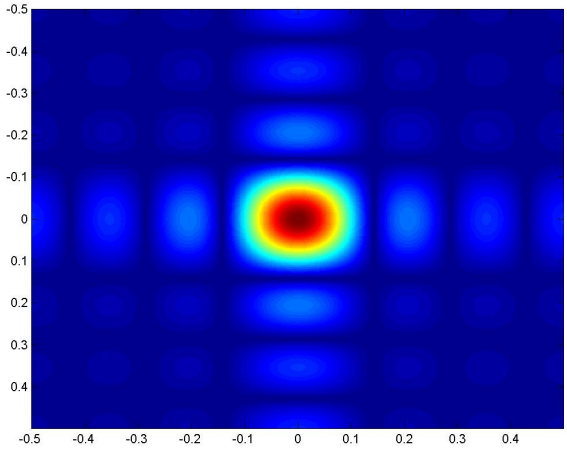
1 – The first part of the script ***lowpass.m*** allows you to define a low-pass filter kernel. Here, the convolution kernel defines an average filtering. Each pixel value is the mean value of the neighboring luminance values:

$$\frac{1}{9}\cdot\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

The magnitude of the frequency response is displayed:

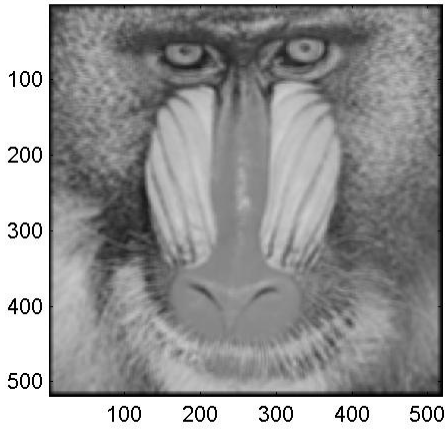


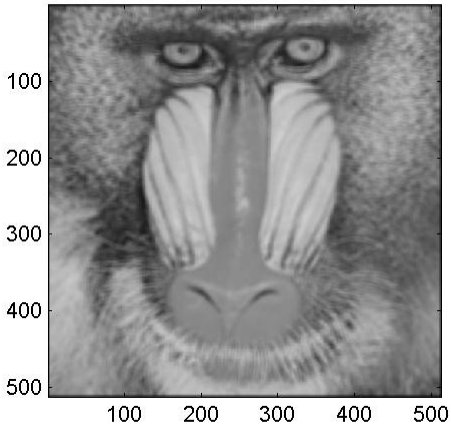
As expected the high frequency components have been removed. However the magnitude is still too significant for horizontal and vertical high frequencies. This low-pass filter is not strongly selective.
By increasing the size of the filter support (support=7), we display the following transfer function:



By increasing the support size, we visualize that the filter passband is reduced: the filter is more selective. The magnitude is less significant for the middle and high frequencies. In fact the larger the size of the filter support is, the larger the neighborhood is, therefore pixel values are the mean values of large neighborhoods: these pixel values are thus close to each other (smoothing effect). **The luminance variations** according to (Ox) or (Oy) **are** thus **low**, that corresponds to **low spatial frequencies**.

2 – By running the second part of the script ***lowpass.m***, we visualize the following images:





This images are computed with a 7×7 filter. The filtering is applied by computing the convolution "filter⊗image" with the Matlab function ***conv2***. A smoothing effect appears on the two output images. However, in the Matlab Workspace, you can note that the size of the image imf2 (on the right) is smaller than the size of the image imf (on the left). In fact these two output images are computed with the same function ***conv2*** but the input optional parameters of this function are different:

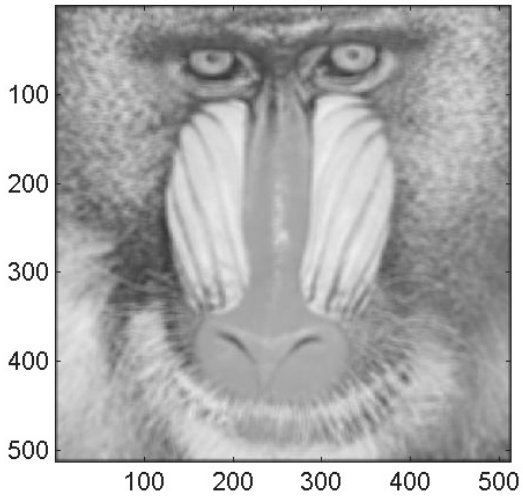
```
imf = conv2(im1,filtre); % (can be written imf=conv2(im1,filtre,'full');)
imf2 = conv2(im1,filtre,'same');
```

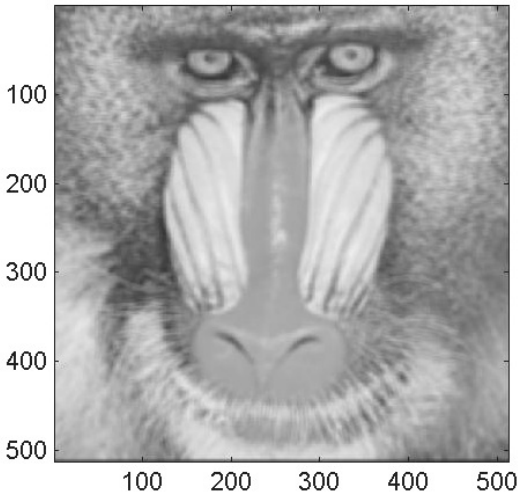
Let us consider an input image whose size is M_1 x N_1, and a filter whose size is M_2 x N_2:

- The output image imf is created by computing the two-dimensional convolution of two 2D signals. Its size is thus (M_1+M_2-1) x (N_1+N_2-1).
- The output image imf2 is created by computing the same two-dimensional convolution but only the central part of the result is preserved. The output image size will be thus the same as the input image size.

Note that there is a third optional parameter ('valid') of the function ***conv2***. The output image is only those parts of the convolution that are computed without the zero-padded edges. The output image size is thus (M_1-M_2+1) x (N_1-N_2+1).

3 – By running the third part of the script ***lowpass.m***, we display the following images (7×7 filter support):





These two images are computed with the Matlab function ***imfilter***:

```
imf3 = imfilter(im1,filtre);
imf4 = imfilter(im1,filtre,'replicate');
```

Note that the right side is darker on the left image. When you compute the pixels at the boundary of an image, a portion of the convolution kernel is usually off the edge of the image. To compute the convolution, it is necessary to define boundary conditions to fill in these “off-the-edge” image pixels. Matlab allows you to use some usual boundary conditions: zero-padding (method by default), replication (`'replicate'`), symmetry (`'symmetric'`), etc.

Examples of boundary conditions:

Let us consider the following M × N grayscale image I:

7	1	72	1	2	1
25	5	100	5	10	209
87	125	40	87	248	28
154	42	25	111	2	1
0	45	114	5	194	68
58	18	47	220	4	8

We want to apply the 3×3 average filter “h”. The pixels marked red have no neighborhood off the edge.

- Zero-padding:

Let us consider that the "off-the-edge" neighborhood is set to 0. The convolution can thus be performed on the pixels marked red.

0	0	0	0	0	0	0	0
0	7	1	72	1	2	1	0
0	25	5	100	5	10	209	0
0	87	125	40	87	248	28	0
0	154	42	25	111	2	1	0
0	0	45	114	5	194	68	0
0	58	18	47	220	4	8	0
0	0	0	0	0	0	0	0

That is the default method used by Matlab.

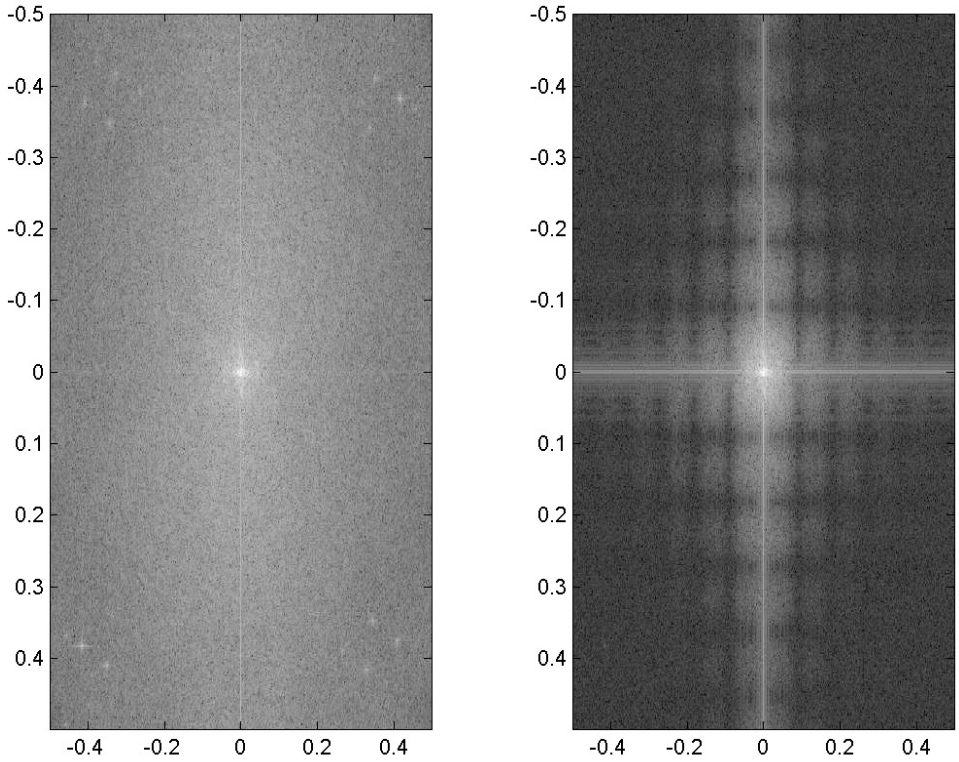
- Duplication:

Let us consider that the boundary pixels are replicated to create an off-the-edge neighborhood. The "off-the-edge" pixels are set to the value of the nearest border pixel in the original image.

7	7	1	72	1	2	1	1
7	7	1	72	1	2	1	1
25	25	5	100	5	10	209	209
87	87	125	40	87	248	28	28
154	154	42	25	111	2	1	1
0	0	45	114	5	194	68	68
58	58	18	47	220	4	8	8
58	58	18	47	220	4	8	8

Note: in the special case of the 3×3 filters, the replication method is equivalent to the symmetry method.

The last part of the script ***highpass.m*** allows you to display the frequency response of the image *MANDRILL_LUMI.BMP* before and after filtering:



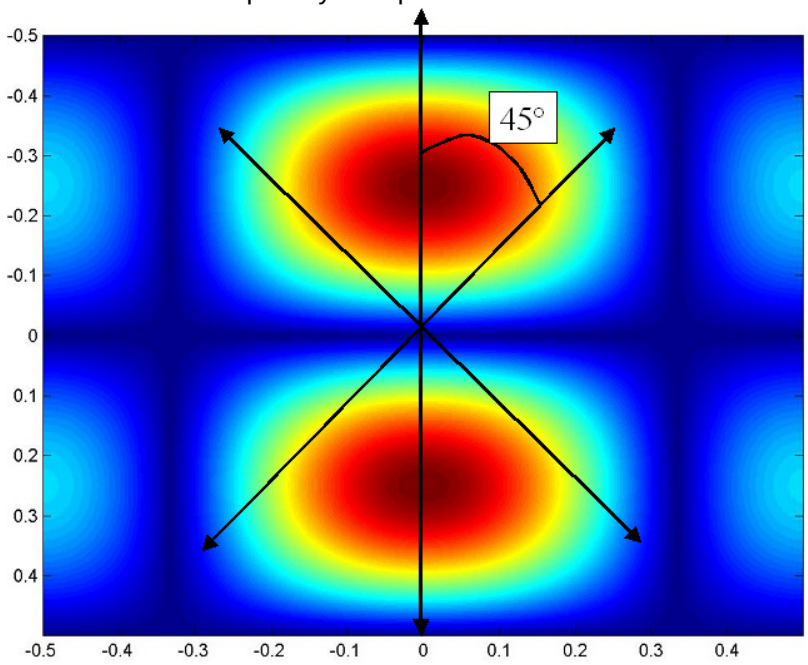
The image on the left shows the frequency response of the image *MANDRILL_LUMI.BMP* before filtering. There is a white zone on the image center which corresponds to the image's DC (i.e. continuous) component.
The group of points around this DC component corresponds to some middle and high frequency components of the image (contours, details, transitions, etc.).
The image on the right shows the frequency response of the image *MANDRILL_LUMI.BMP* after low-pass filtering.
The DC component is preserved because it is a low spatial frequency (the DC component is theoretically obtained at the spatial frequency (ν_X=0, ν_Y=0)).
The group of points corresponding to the middle and high frequencies is removed globally. However the frequency gain according to the horizontal and vertical pure spatial frequencies is not reduced enough. This result was expected thanks to the analysis of the filter's frequency response.

4 – The script ***filtering_spatialvsfreq.m*** uses the Matlab commands "***tic***" and "***toc***" and allows you to compare the computation times of two filtering methods: the first one uses the spatial domain (convolution with ***imfilter***) and the second one uses the frequency domain (computation of FFT with ***fft2***, ***fftshift***, and ***ifft2***).
We note that it is computationally faster to perform two 2D Fourier transforms and a filter multiplication than to perform a convolution in the spatial domain. It is particularly true when the filter size increases.
In fact when the filter size increases, the spatial filtering needs more operations: "multiplication and accumulation" whereas the frequency filtering still needs two Fourier transforms (one for the image and another for the filter), a multiplication and one inverse Fourier transform (IFFT).

5 – The last part of the script ***highpass.m*** allows us to display the frequency response of the performed high-pass filter. Here, we consider the vertical Prewitt filter. Its convolution kernel is:

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

After a fast analysis of this kernel, we note that the pixel value differences will be increased according to the vertical direction and reduced according to the horizontal direction. The filter is thus a high pass filter for the vertical frequency. This result is strongly visible on the image of the filter frequency response:



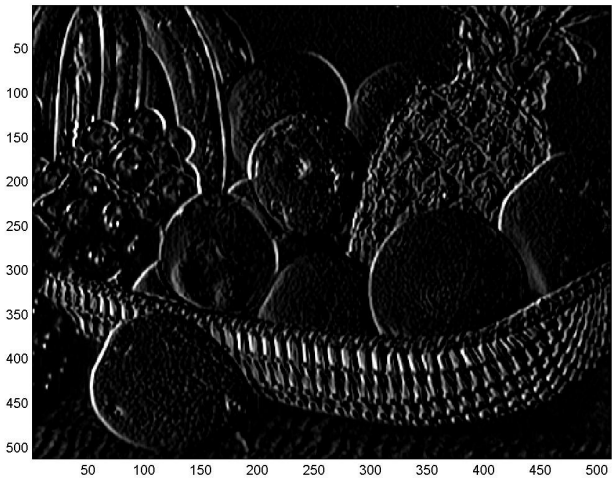
We note that the filter is strongly selective neither for the spatial frequencies nor for the orientation (tolerance of inclinations is about ± 45°).
By filtering the image *FRUIT_LUMI.BMP* with the vertical Prewitt filter, we create the following image:

The horizontal edges of the original image are detected.

Another filter is performed. This is the horizontal Prewitt filter. Its convolution kernel is:

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

The vertical edges of the original image are detected.



Such filters are typically used to enhance edges in an image by adding the original image to the filtered image.

Chapter 3 – Linear filtering

TEST

Let us consider linear filters $\boldsymbol{H}$ which are defined by a 2D convolution kernel called $\boldsymbol{h}$. The kernel size will be 3×3 for this entire test. The h(0,0) element is located on the center of the kernel support. $\boldsymbol{I_0}$ stands for the input image and $\boldsymbol{I_S}$ stands for the output image. $\boldsymbol{I_0}$ is a grayscale image. Its luminance values belong to the range [0, 255].

1 – What is the 3×3 convolution kernel $\boldsymbol{h_1}$ of the ***Identity filter*** (such as $\boldsymbol{I_S} = \boldsymbol{I_0}$)?

2 – Let $\boldsymbol{H_2}$ be the filter whose convolution kernel $\boldsymbol{h_2}$ is:

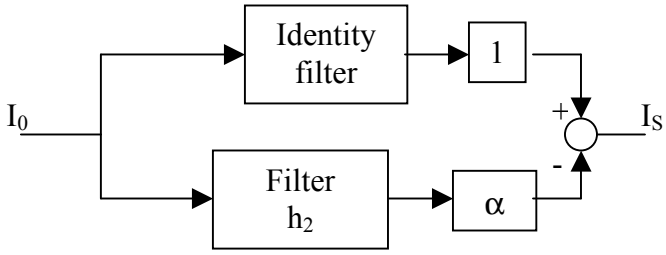
$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

2.1 – What is the DC gain of this filter (gain according to the spatial frequencies $\nu_X = 0$ and $\nu_Y = 0$)?

2.2 – If the image signal is constant and equal to A on an image area whose size is larger or equal to 3×3, what is the $\boldsymbol{I_S}$ computed value at the center of this area?

3 – By editing a Matlab program, perform the filtering of an (M × N) image $\boldsymbol{I_0}$ whose kernel is $\boldsymbol{h_2}$. The output image must be the same size as the input image $\boldsymbol{I_0}$. What do you notice on the object edges of the image "***Boats_lumi.bmp***"?

4 – We want to enhance the contrast of the image $\boldsymbol{I_0}$ objects. To do that, we want to create a third, functionally equivalent filter to find the difference between the Identity filter and a fraction (value: α) of the filter whose kernel is $\boldsymbol{h_2}$.



Write the Matlab program to create this third filter. Visualize the results with the following values of α: 0, 1/10, 1/4, 1/2. Describe the results obtained on the zones corresponding to the fishing boom and the fishing rope on the image "***Boats_lumi.bmp***".