

# IMAGE PROCESSING TUTORIALS with MATLAB®

Yann GAVET  
Johan DEBAYLE



Attribution 4.0 International  
(CC BY 4.0)

This is a human-readable summary of (and not a substitute for) the license, available at:  
<https://creativecommons.org/licenses/by-sa/4.0/legalcode>

## Your are free to:

**Share** - copy and redistribute the material in any medium or format

**Adapt** - remix, transform or build upon the material for any purpose, even commercially.

The licensor cannot revoke these freedoms as long as you follow the license terms.

## Under the following terms:



**Attribution** — You must give **appropriate credit**, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

**Appropriate credit** — You must mention the **authors** and their host institution (**MINES SAINT-ETIENNE**).

No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

## Credits

Cover image: Pepper & Carrot, David Revoy, [www.davidrevoy.com](http://www.davidrevoy.com).



## Informations

For MATLAB® product information, please contact:

The MathWorks, Inc.

3 Apple Hill Drive

Natick, MA, 01760-2098 USA

Tel: 508-647-7000

Fax: 508-647-7001

E-mail: [info@mathworks.com](mailto:info@mathworks.com)

Web: <https://www.mathworks.com>

How to buy: <https://www.mathworks.com/store>

Find your local office: <https://www.mathworks.com/company/worldwide>

## Contributors:

- Victor Rabiet
- Séverine Rivollier
- Place your name here and get special credit if you want to participate to this book.



# TABLE OF CONTENTS

13

PART I

## Enhancement and Restoration

1	Introduction to image processing .....	15
2	Image Enhancement .....	39
3	2D Fourier Transform .....	51
4	Introduction to wavelets .....	63
5	Image restoration: denoising .....	79
6	Image Restoration: deconvolution .....	93
7	Shape From Focus .....	111
8	Logarithmic Image Processing (LIP) .....	121
9	Color LIP .....	129
10	GANIP .....	139
11	Image Filtering using PDEs .....	151
12	Multiscale Analysis .....	161
13	Introduction to tomographic reconstruction .....	171

179

PART II

## Mathematical Morphology

14	Binary Mathematical Morphology .....	181
15	Morphological Geodesic Filtering .....	193
16	Morphological Attribute Filtering .....	199
17	Morphological skeletonization .....	205
18	Granulometry .....	213

219

**PART III****Registration and Segmentation**

<b>19</b>	<b>Histogram-based image segmentation</b>	<b>221</b>
<b>20</b>	<b>Segmentation by region growing</b>	<b>231</b>
<b>21</b>	<b>Hough transform and line detection</b>	<b>237</b>
<b>22</b>	<b>Active contours</b>	<b>243</b>
<b>23</b>	<b>Watershed</b>	<b>251</b>
<b>24</b>	<b>Segmentation of follicles</b>	<b>259</b>
<b>25</b>	<b>Image Registration</b>	<b>265</b>

**277****PART IV****Stochastic Analysis**

<b>26</b>	<b>Stochastic Geometry / Spatial Processes</b>	<b>279</b>
<b>27</b>	<b>Boolean Models</b>	<b>295</b>
<b>28</b>	<b>Geometry of Gaussian Random Fields</b>	<b>303</b>
<b>29</b>	<b>Stereology and Bertrand's paradox</b>	<b>313</b>
<b>30</b>	<b>Convex Hull</b>	<b>329</b>
<b>31</b>	<b>Voronoi Diagrams and Delaunay Triangulation</b>	<b>337</b>

**349****PART V****Image Characterization and Pattern Analysis**

<b>32</b>	<b>Integral Geometry</b>	<b>351</b>
<b>33</b>	<b>Topological Description</b>	<b>357</b>
<b>34</b>	<b>Image Characterization</b>	<b>365</b>
<b>35</b>	<b>Shape Diagrams</b>	<b>373</b>
<b>36</b>	<b>Freeman Chain Code</b>	<b>381</b>
<b>37</b>	<b>Machine Learning</b>	<b>391</b>
<b>38</b>	<b>Harris corner detector</b>	<b>397</b>
<b>39</b>	<b>Local Binary Patterns</b>	<b>403</b>

**411****PART VI****Exams**

<b>40</b>	<b>Practical exam 2016 .....</b>	<b>413</b>
<b>41</b>	<b>Theoretical exam 2016 .....</b>	<b>415</b>
<b>42</b>	<b>Theoretical exam 2017 .....</b>	<b>419</b>



# About the authors

## Yann GAVET

received his "Ingénieur Civil des Mines de Saint-Etienne" diploma in 2001. He then obtained a Master of Science and a PhD thesis on the segmentation of human corneal endothelial cells (in 2004 and 2008). He is now an assistant professor at the Saint-Etienne School of Mines, where he teaches computer science and image processing to engineering and master students. He is a member of the PMDM Department of the LGF Laboratory, UMR CNRS 5307, dedicated to granular media analysis and modelisation.

He is particularly interested in the world of free (LIBRE) software in computer science. His research interests include image processing and analysis, stochastic geometry and numerical simulations. He published more than 70 papers in international journals and conference proceedings. He is a member of the Institute of Electrical and Electronics Engineers (IEEE), the International Association for Pattern Recognition (IAPR), International Society for Stereology and Image Analysis (ISSIA). He has worked for CS-SI (Toulouse, France) as an IT engineer, and for Thalès-Angénieux (Saint-Héand, France) as an image processing expert.

## Johan DEBAYLE

received his M.Sc., Ph.D. and Habilitation degrees in the field of image processing and analysis, in 2002, 2005 and 2012 respectively. Currently, he is a Full Professor at the Ecole Nationale Supérieure des Mines de Saint-Etienne (ENSM-SE) in France, within the SPIN Center and the LGF Laboratory, UMR CNRS 5307, where he leads the PMDM Department interested in image analysis of granular media. In 2015, he was a Visiting Researcher for 3 months at the ITWM Fraunhofer / University of Kaiserslautern in Germany. In 2017 and 2019, he was invited as Guest Lecturer at the University Gadjah Mada, Yogyakarta, Indonesia. He was also Invited Professor at the University of Puebla in Mexico in 2018 and 2019. He is the Head of the Master of Science in Mathematical Imaging and Spatial Pattern Analysis (MISPA) at the ENSM-SE.

His research interests include image processing and analysis, pattern recognition and stochastic geometry. He published more than 120 international papers in international journals and conference proceedings and served as Program committee member in several international conferences (IEEE ICIP, MICCAI, ICIAR...). He has been invited to give a keynote talk in several international conferences (SPIE ICMV, IEEE ISIVC, SPIE-IS&T EI, SPIE DCS...) He is Associate Editor for 3 international journals: Pattern Analysis and Applications (Springer), Journal of Electronic Imaging (SPIE) and Image Analysis and Stereology (ISSIA).

He is a member of the International Society for Optics and Photonics (SPIE), International Association for Pattern Recognition (IAPR), International Society

for Stereology and Image Analysis (ISSIA) and Senior Member of the Institute of Electrical and Electronics Engineers (IEEE).

[

MINES SAINT-ÉTIENNE] ÉCOLE NATIONALE SUPÉRIEURE DES MINES DE SAINT-ÉTIENNE

One of the missions of École des Mines de Saint-Étienne, France, is scientific research at the highest level and contributions to companies' competitiveness. It aims at conveying the economical politics of the country and speeding up the sustainable industrial development by innovation and efficient contributions.

This high level scientific research leads to publications recognized by the international scientific community. Research and teaching are very closely interwoven and the consequence of this is the attractiveness of our Master's Degree courses and our renowned Doctoral School.



**Une école de l'IMT**

# Liminar considerations

This book is presented as a collection of tutorials. Each chapter presents different objectives, usually with practical applications, in order to develop the image processing and analysis skills by its own. For each tutorial, you will first find in the statement different questions that will guide you to the complete results. Our suggestion is to start by the first tutorials, that acts as a round-up stage. Then, choose the topic that interests and motivates you, and start to work on your code.

The statements does not always contain the necessary theoretical background that you need to answer to the questions. Find resources on the net. Wikipedia is usually a very good start. Go to the university library!

The correction gives you a proposition of solution: it is not unique and it might not be perfect (or it might unfortunately contain errors). This correction is also available in a dedicated website (links will be provided along with the tutorials), we suggest you have a look at it only when you encounter blocking problems or fastidious lines of code to program. Moreover, you can evaluate the processing time of your version of the code and compare it to our proposed version, which usually uses fast and optimal functions, except for readability purposes.

Within MATLAB<sup>®</sup>, you can use the following commands to measure the computation time.



```
1 tic ;
% run your code
3 toc;
```

## CC-By license?

If you manage to code a drastically faster method, if you notice errors or mistakes, if you want to add precisions, remember that this book as well as the code is published under a FREE CC-BY license (as in FREE speech). Contact us and we will be really happy to introduce modifications and insert you in the credits, or more...



The book is available in a high quality printed format, obviously not for free, but the pdf format is free (as in FREE beer). This is due to the fact that we (Johan Debayle and Yann Gavet) are employed by the French government, and we are paid to teach and disseminate informations to students. Our work is thus belongs to the society, and it seems a normal process to give the results back to the society. Feel free to use, modify and distribute these tutorials as you wish. Just remember to

keep the appropriate credits (our names and MINES Saint-Etienne). If you want to express your gratitude, send us a postcard at:

Yann GAVET or Johan Debayle  
 MINES Saint-Etienne  
 CS 62362  
 42023 SAINT-ETIENNE cedex 2 - FRANCE

You can also buy the printed version of the book, which will make our editor really happy.

## Images

The images belong to their authors, unless otherwise stated. If by mistake we used images without giving the appropriate credit, please contact us.

## Softwares

This book is available for two programming languages, MATLAB® and Python. MATLAB® is a proprietary software dedicated to scientific computing. Functions are grouped into so-called toolboxes. The documentation and the compatibility of the functions are very good, but the price is a consequence of this. Python is built upon open-source and Free softwares. Scientific modules are numerous, but other general purposes functions can also be found. Documentation and code may be of unequal qualities, but major scientific modules (numpy, scipy, opencv...) are really well presented and optimized.

Portions of code will be highlighted by the use of special boxes, like:



## Time to spend on a tutorial

The tutorials are almost independent. To evaluate the difficulty of the tutorial, stars are present at the header of each one. Depending on your knowledge, you will have to spend between 2 hours and 10 hours per tutorial.

- One star tutorial should be a relatively easy tutorial,

- two stars tutorials introduce some difficulties, either in programming or in the theoretical concepts,
- three stars tutorials are difficult both in theory and in programming.

They are divided into 6 parts, namely:

- Enhancement and Restoration: dedicated to image processing methods employed to eliminate noise and improve vision of data.
- Mathematical Morphology: introduction to basic operations of mathematical morphology.
- Registration and Segmentation: methods to segment, i.e. detect objects in images.
- Stochastic Analysis: method based on random processes.
- Characterization and Pattern Analysis: measures performed on objects in order to perform analysis and recognition.
- Exams: uncorrected problems and questions.

## Enjoy!

You will find online corrections on the editor website and at page: <http://iptutorials.science>. You will also find qrcodes for each correction that points to code and images.



# **Part I Enhancement and Restora- tion**



# 1

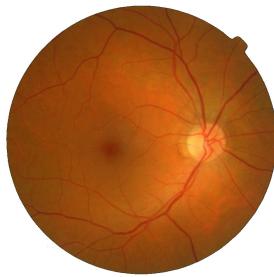
# Introduction to image processing

In this tutorial, you will discover the basic functions in order to load, manipulate and display images. The main informations of the images will be retrieved, like size, number of channels, storage class, etc. Afterwards, you will be able to perform your first classic filters.

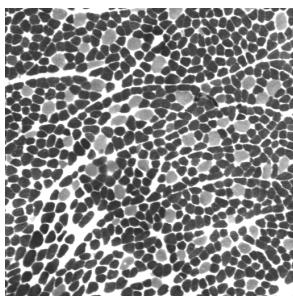
The different processes will be realized on the following images:

Figure 1.1: Image examples.

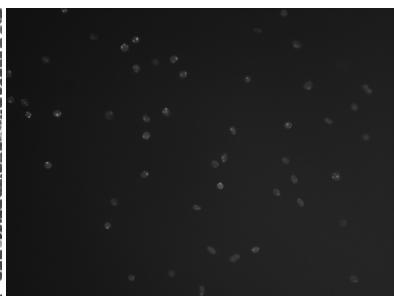
(a) Retinal vessels.



(b) Muscle cells.



(c) Cornea cells (BIIGC, Univ. Jean Monnet, Saint-Etienne, France).



## 1.1

## First manipulations



Image loading can be made by the use of the MATLAB® function `imread`. The visualization of the image in the screen is realized either by the MATLAB® function `imagesc` or `imshow`.



- Load and visualize the first image as below. Notice the differences.
- Look at the data structure of the image  $I$  such as its size, type....
- Visualize the green component of the image. Is it different from the red one? What is the most contrasted color component? Why?
- Enumerate some digital image file formats. What are their main differences? Try to write images with the JPEG file format with

different compression ratios (0, 50 and 100), as well as the lossless compression, and compare.



See `imwrite`.

## 1.2 Color quantization

Color quantization is a process that reduces the number of distinct colors used in an image, usually with the intention that the resulting image should be as visually similar as possible to the original image. In principle, a color image is usually quantized with 8 bits (i.e. 256 gray levels) for each color component.



- By using the gray level image 'muscle', reduce the number of gray levels to 128, 64, 32, and visualize the different resulting images.
- Compute the different image histograms and compare.

## 1.3 Image histogram

An image histogram represents the gray level distribution in a digital image. The histogram corresponds to the number of pixels for each gray level. The MATLAB® function that computes the histogram of any gray level image has the following prototype:



```
function h = histogram(I)
```



Compute and visualize the histogram of the image 'muscle.jpg'.

**1.4**

## Linear mapping of the image intensities

The gray level range of the image 'cellules\_cornee.jpg' can be enhanced by a linear mapping such that the minimum (resp. maximum) gray level value of the resulting image is 0 (resp. 255). Mathematically, it consists in finding a function  $f(x) = ax+b$  such that  $f(\min) = 0$  and  $f(\max) = 255$ .



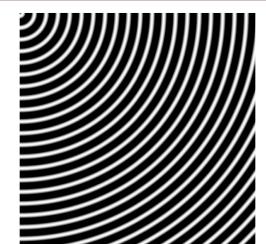
- Load the image and find its extremal gray level values.
- Adjust the intensities by a linear mapping into  $[0, 255]$ .
- Visualize the resulting image and its histogram.

**1.5**

## Aliasing effect

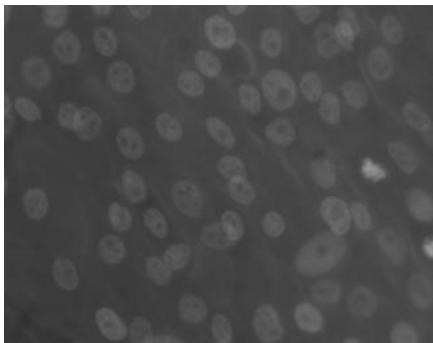


- Create an image (as right) that contains rings as sinusoids. The function takes two input parameters: the sampling frequency and the signal frequency.
- Look at the influence of the two varying frequencies. What do you observe? Explain the phenomenon from a theoretical point of view.

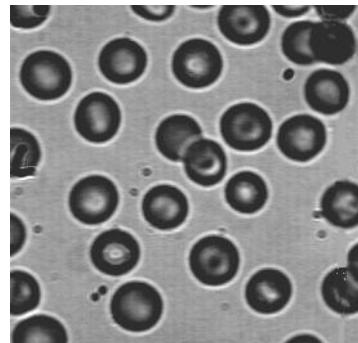
**1.6**

## Low-pass filtering

The different processes will be realized on the following images:



(a) osteoblast cells



(b) blood cells

Low-pass filtering aims to smooth the fast intensity variations of the image to be processed.



Test the low-pass filters 'mean', 'median', 'min', 'max' and 'gaussian' on the noisy image 'blood cells'.



The MATLAB<sup>®</sup> functions `imfilter` and `nlfilter` can be employed. Be careful to the function options for border problems. Also, the MATLAB<sup>®</sup> function `fspecial` enables an operational window to be generated.



Which filter is suitable for the restoration of this image?

## 1.7

## High-pass filtering

High-pass filtering aims to smooth the low intensity variations of the image to be processed.



- Test the high-pass filters  $HP$  on the two initial images in the following way:  $HP(f) = f - LP(f)$  where  $LP$  is a low-pass filtering (see the previous exercise).

- Test the Laplacian (high-pass) filter on the two initial images with the following convolution mask:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & +8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

## 1.8 Derivative filters

Derivative filtering aims to detect the edges (contours) of the image to be processed.



- Test the Prewitt and Sobel derivative filters (corresponding to first order derivatives) on the image 'blood cells' with the use of the following convolution masks:

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

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

- Look at the results for the different gradient directions.
- Define an operator taking into account the horizontal and vertical directions.

Remark : the edges could be also detected with the zero-crossings of the Laplacian filtering (corresponding to second order derivatives).

## 1.9 Enhancement filtering

Enhancement filtering aims to enhance the contrast or accentuate some specific image characteristics.



- Test the enhancement filter  $E$  on the image 'osteoblast cells' defined as:  $E(f) = f + HP(f)$  where  $HP$  is a Laplacian filter (see tutorial 2.).
- Parameterize the previous filter as:  $E(f) = \alpha f + HP(f)$ , where  $\alpha \in \mathbb{R}$ .

## 1.10

### Open question

Find an image filter for enhancing the gray level range of the image 'osteoblast cells'.



## 1.11. Matlab correction



### 1.11.1 First manipulations

The following function is usefull to display 4 images:



```
function affichePar4 (A, B, C, D)
2 % Display 4 images in the same window
figure () ;
4 subplot (2,2,1) ;
imshow(A);
6 subplot (2,2,2) ;
8 imshow(B);

10 subplot (2,2,3) ;
imshow(C);
12 subplot (2,2,4) ;
14 imshow(D);
```

### Load and save image



```
I=imread(' retine .png');
2 imagesc(I);
figure (2);
4 imshow(I);

6 % data inside image
imfinfo(' retine .png');
8 size(I)
```

```
Command window ▾

>> imfinfo('retine.png')
ans =
    Filename: '/home/yann/Documents/Cou...'
    FileModDate: '05-Nov-2013 12:04:24'
    FileSize : 741963
    Format: 'png'
    FormatVersion: []
    Width: 922
    Height: 911
    BitDepth: 24
    ColorType: 'truecolor'
    FormatSignature: [137 80 78 71 13 10 26 10]
    Colormap: []
    Histogram: []
    InterlaceType : 'none'
    Transparency: 'none'
    SimpleTransparencyData: []
    BackgroundColor: []
    RenderingIntent: []
    Chromaticities : []
        Gamma: []
    XResolution: 2835
    YResolution: 2835
    ResolutionUnit : 'meter'
        XOffset: []
        YOffset: []
        OffsetUnit : []
    SignificantBits : []
    ImageModTime: []
        Title : []
        Author: []
    Description : []
        Copyright: []
    CreationTime: []
        Software: []
    Disclaimer: []
        Warning: []
        Source: []
        Comment: []
    OtherText: []

>>
```

## JPEG file format

JPEG is a compressed file format that accept loss in quality. The following code illustrates the different quality parameters, shown in Fig.1.2.



```

1 % test read/write with loss in quality
imwrite(I, 'retine_lossy_25.jpg', 'jpg', 'Mode', 'lossy', 'Quality', 25);
3 a1=imread('retine_lossy_25.jpg');

5 imwrite(I, 'retine_lossy_50.jpg', 'jpg', 'Mode', 'lossy', 'Quality', 50);
a2=imread('retine_lossy_50.jpg');

7 imwrite(I, 'retine_lossy_75.jpg', 'jpg', 'Mode', 'lossy', 'Quality', 75);
9 a3=imread('retine_lossy_75.jpg');

11 imwrite(I, 'retine_lossy_100.jpg', 'jpg', 'Mode', 'lossy', 'Quality', 100);
a4=imread('retine_lossy_100.jpg');

13 % zoom in particular area
15 d1=imcrop(a1, [75 68 130 112]);
d2=imcrop(a2, [75 68 130 112]);
17 d3=imcrop(a3, [75 68 130 112]);
d4=imcrop(a4, [75 68 130 112]);
19 affichePar4(d1, d1, d3, d4);

```

## 1.11.2 Image histogram

Notice that MATLAB® indices begin at 1!



```

function h=myHist(image)
2 % histogram function of grayscale image coded in 8 bits
h=zeros(256, 1);

4 for i=1:size(image, 1)
6   for j=1:size(image, 2)
     h(image(i,j)+1) = h(image(i,j)+1) + 1;
8 end
end

```

This is the MATLAB® version.



```

1 muscle=imread('muscle.jpg');
h = imhist(muscle);
3 figure () ;plot(h);

```

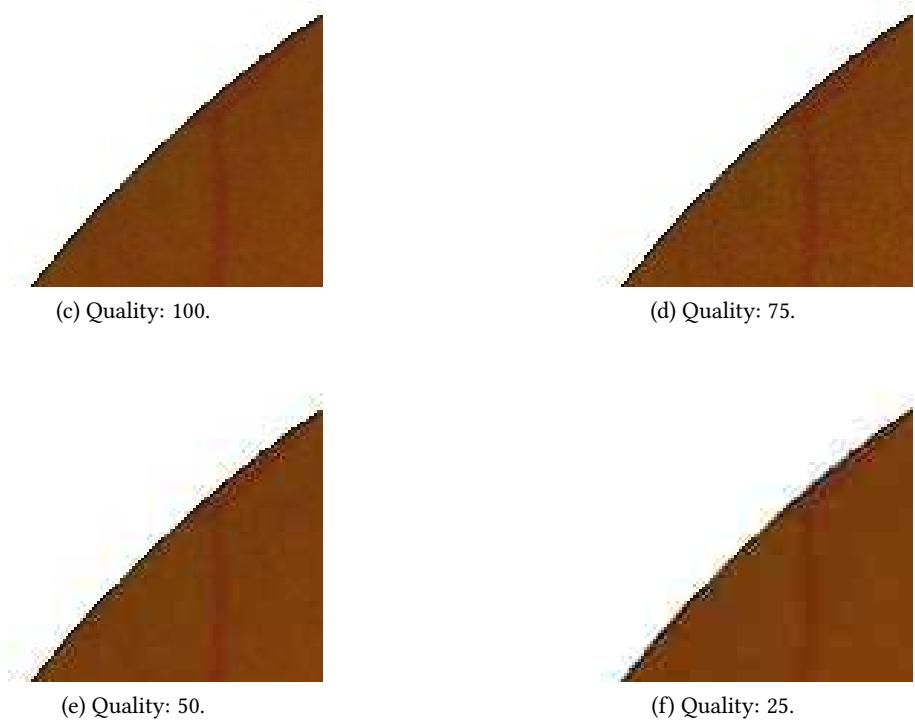


Figure 1.2: Illustration of different quality parameters used in JPEG compression. The original image is zoomed in order to emphasize the quality loss.

### 1.11.3 Linear mapping of the image intensities

The application of a linear stretching is quite simple. The histograms are illustrated in Fig.1.3.



```

1 cornea=imread(' cellules_cornee .jpg');
    minimum = min(cornee(:));
3 maximum = max(cornee(:));

5 a=255/(maximum-minimum);
    b=-255*minimum/(maximum-minimum);
7
    cornee2 = a*cornee + b;
9 figure ();
    subplot (2,2,1) ;imshow(cornee); title ('cornea');
11 subplot (2,2,2) ;imshow(cornee2); title (' stretched cornea');

13 % histograms
    subplot (2,2,3) :plot(imhist(cornee)); title ('histogram of cornea');
15 subplot (2,2,4) :plot(imhist(cornee2)); title (' stretched histogram');

```

### 1.11.4 Color quantization

The objective is to reduce the number of colors by 2 (for example). We use the properties of the data types: integer type rounds automatically while dividing. In the proposed example, the gray value is taken as the green channel of a color retina image (Fig.1.4).



```

1 image_gris = I (:,:,2) ; % green channel
    q4=image_gris/4*4;
3 q16=image_gris/16*16;
    q32=image_gris/32*32;
5 affichePar4 (image_gris, q4, q16, q32);

```

### 1.11.5 Aliasing (Moiré) effect

The aliasing effect occurs when two sampling are performed. This is illustrated in Fig.1.5 with the following code.

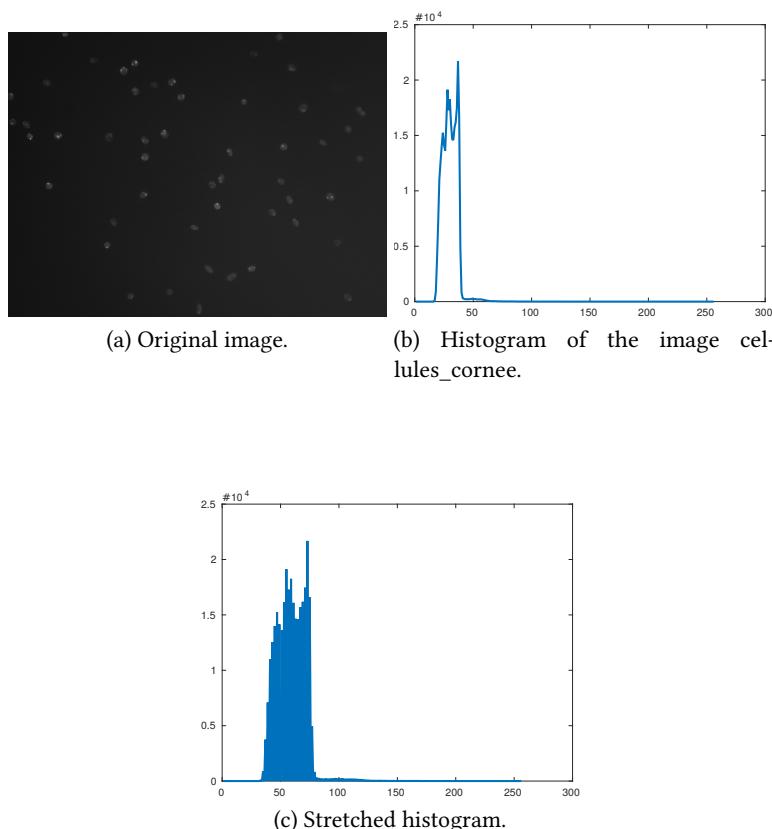


Figure 1.3: Result of histogram stretching.

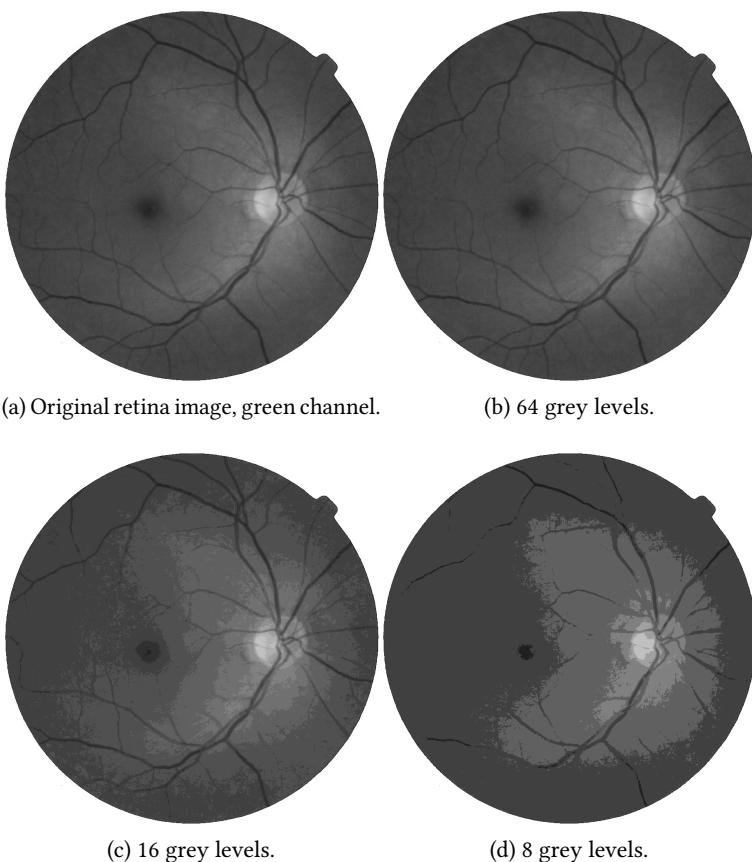


Figure 1.4: Illustration of different quality parameters used in JPEG compression.



```

1 function C=cercle(fs ,f)
% Generates an image with aliasing effect
3 % fs: sample frequency
% f : signal frequency
5
% time sampling
7 t =0:1/ fs :1;

9 C=zeros(size(t,2));
for i=1: size(t,2);
11     for j=1: size(t,2);
12         C(i,j)=sin(2* pi*f* sqrt(t(i)^2+t(j)^2));
13     end
end

```

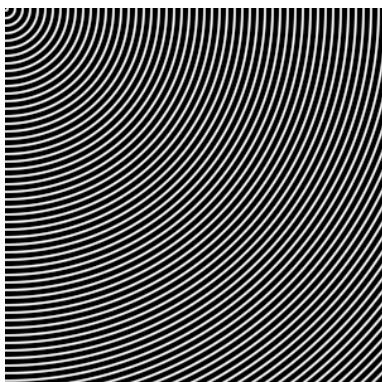
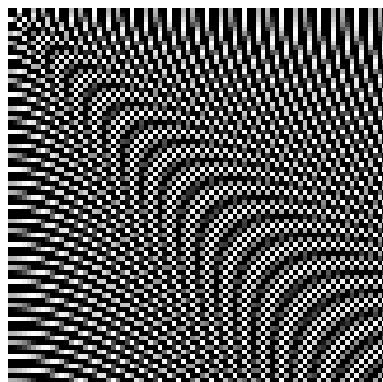
(a)  $f_s = 300, f = 50.$ (b)  $f_s = 80, f = 50.$ 

Figure 1.5: Illustration of the aliasing effect.

## 1.11.6 Low-pass filtering

The mean and Gaussian filters are linear filters. Other filters are called rank filters. See Fig.1.6 for an illustration.



```

% read image and convert to double for convolution computation
2 A=imread('bloodCells.bmp');
A=double(A)/255;
4
% display images
6 figure ;

```



```
subplot(231);imshow(A);title('Original');

8 Amin=ordfilt2(A,1,ones(5,5), 'symmetric');
10 subplot(232);imshow(Amin);title('Low-pass filter : min');

12 Amax=ordfilt2(A,25,ones(5,5), 'symmetric');
14 subplot(233);imshow(Amax);title('Low-pass filter : max');

16 Amoyen=imfilter(A,1/25*ones(5,5), 'symmetric');
18 subplot(234);imshow(Amoyen);title('Low-pass filter : moyen');

20 Amedian=ordfilt2(A,13,ones(5,5), 'symmetric');
22 subplot(235);imshow(Amedian);title('Low-pass filter : median');

hgauss=fspecial('gaussian',[5 5],1);
Agauss=imfilter(A,hgauss);
subplot(236);imshow(Agauss);title('Low-pass filter : gaussien');
```

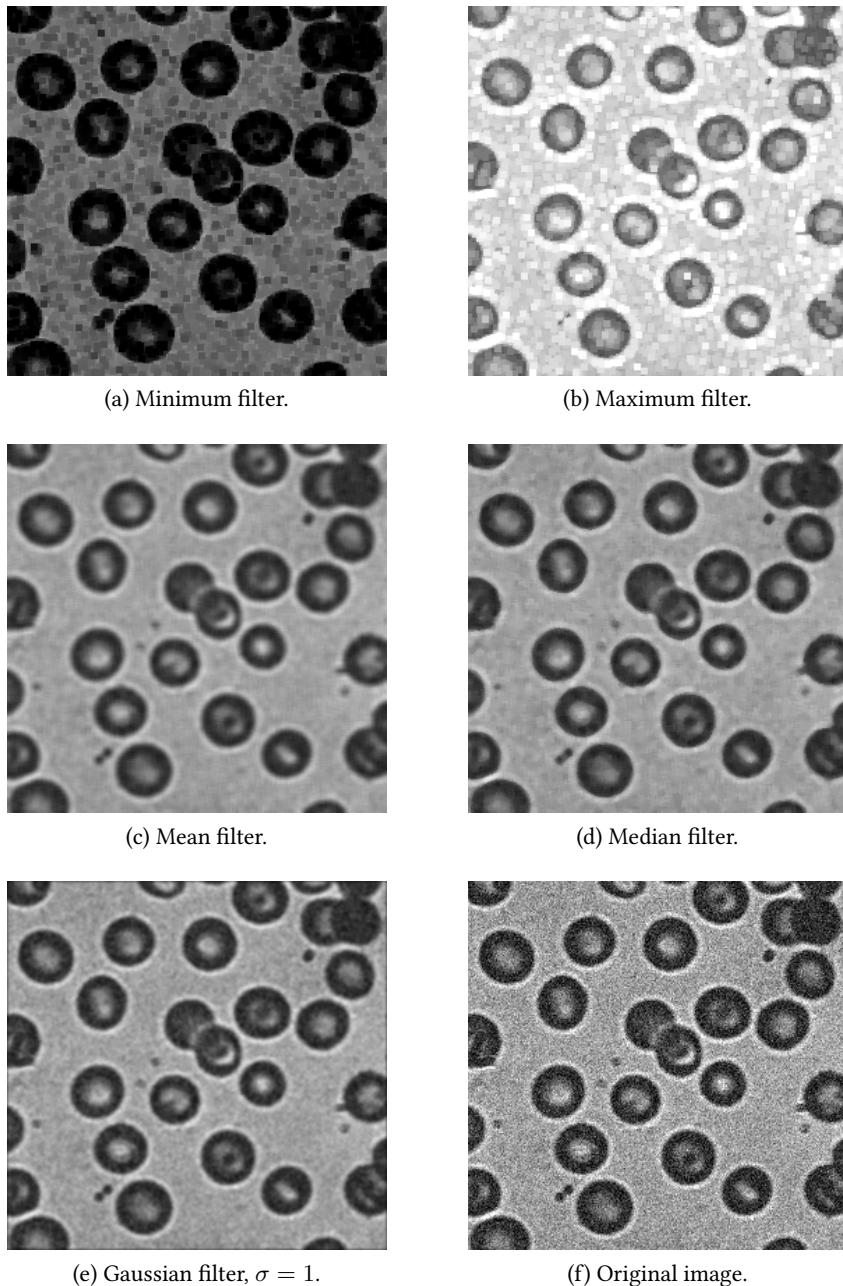


Figure 1.6: Low-pass filters computed in a  $5 \times 5$  neighborhood.

### 1.11.7 High pass filters

These high-pass filters are simple the difference (residu) between the original image and a low-pass filter (Fig.1.7).



```

1 figure ;
2 subplot(231);imshow(A);title('Original');
3
4 AminPH=A-Amin;
5 subplot(232);imshow(AminPH);title('High-pass : min');
6
7 AmaxPH=Amax-A;
8 subplot(233);imshow(AmaxPH);title('High-pass : max');
9
10 AmoyenPH=A-Amoyen;
11 subplot(234);imshow(AmoyenPH);title('High-pass : moyen');
12
13 AmedianPH=A-Amedian;
14 subplot(235);imshow(AmedianPH);title('High-pass : median');
15
16 AgaussPH=A-Agauss;
17 subplot(236);imshow(AgaussPH);title('High-pass : gaussien');

```

### Laplacian filter

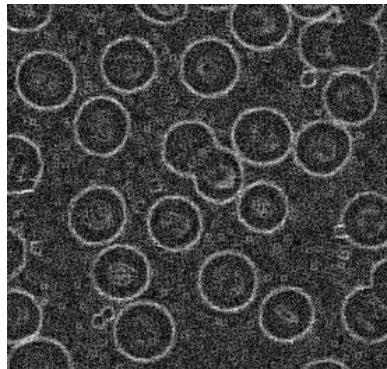
The Laplacian filter is based on the second derivative (Fig.1.8).



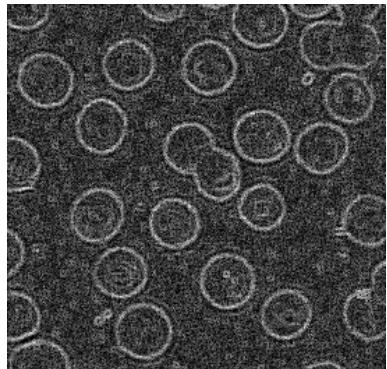
```

1 B=imread('osteoblaste.bmp');
2 B=double(B);
3 B=B/255;
4 hlaplacien=[-1 -1 -1; -1 8 -1;-1 -1 -1];
5 Blaplaciens=imfilter(B, hlaplacien);
6 Alaplaciens=imfilter(A, hlaplacien);
7 figure;
8 subplot(221);imshow(A);title('original image');
9 subplot(222);imshow(Alaplaciens);title('Laplacian filter');
10 subplot(223);imshow(B);title('originale image');
11 subplot(224);imshow(Blaplaciens);title('Laplacian filter');

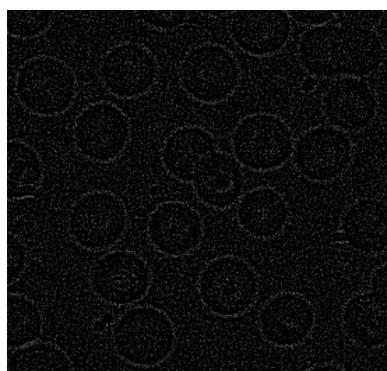
```



(a) Residu from minimum filter.



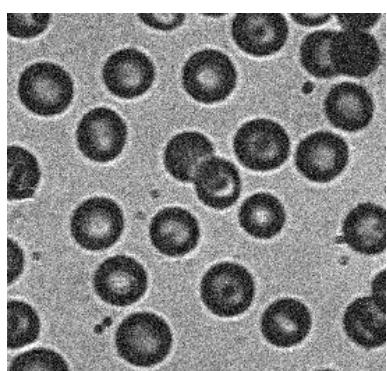
(b) Residu from maximum filter.



(c) Residu from mean filter.

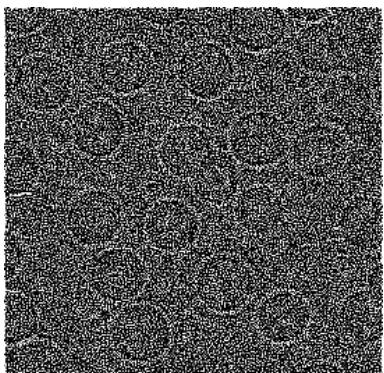


(d) Residu from median filter.

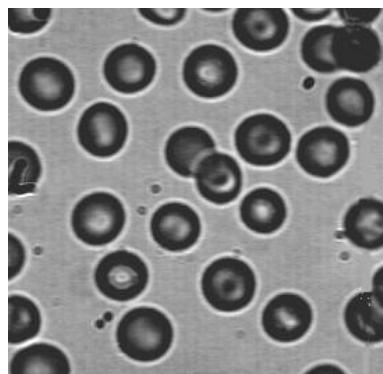
(e) Residu from Gaussian filter,  $\sigma = 1$ .

(f) Original image.

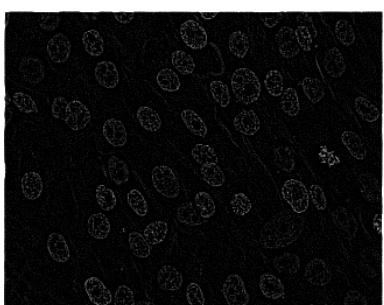
Figure 1.7: High-pass filters computed in a  $5 \times 5$  neighborhood.



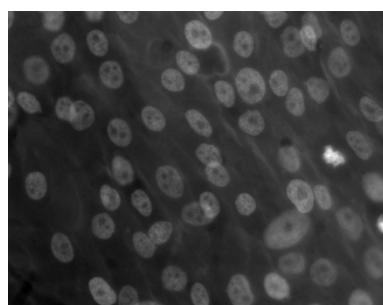
(a) Laplacian filter.



(b) Original image.



(c) Laplacian filter.



(d) Original image.

Figure 1.8: Laplacian filters.

## 1.11.8 Derivative filters

### Derivation: Prewitt gradient

A gradient is a vector of the first derivatives. The norm of this vector represents the intensity of the contours (Fig.1.9 for the Prewitt gradient, and 1.10 for the Sobel gradient). A derivative filter is very sensitive to noise.



```

1 hprewittx=[-1 0 1;-1 0 1;-1 0 1];
hprewitty=hprewittx';
3 Aprewittx= imfilter (A,hprewittx);
Aprewitty= imfilter (A,hprewitty);
5 Aprewittxy=(Aprewittx.^2+Aprewitty.^2) .^(0.5) ;
    subplot (221) ;imshow(A);title (' Original ');
7 subplot (222) ;imshow(Aprewittxy);title (' Prewitt : x and y');
    subplot (223) ;imshow(Aprewittx);title (' Prewitt : x');
9 subplot (224) ;imshow(Aprewitty);title (' Prewitt : y');

```

### Derivation: Sobel gradient



```

1 figure
hsobelx=[-1 0 1;-2 0 2;-1 0 1];
3 hsobely=hsobelx';
Asobelx= imfilter (A,hsobelx);
5 Asobely= imfilter (A,hsobely);
Asobelxy=(Asobelx.^2+Asobely.^2) .^(0.5) ;
7 subplot (221) ;imshow(A);title (' Original ');
    subplot (222) ;imshow(Asobelxy);title (' Sobel : x and y');
9 subplot (223) ;imshow(Asobelx);title (' Sobel : x');
    subplot (224) ;imshow(Asobely);title (' Sobel : y');

```

## 1.11.9 Enhancement filters

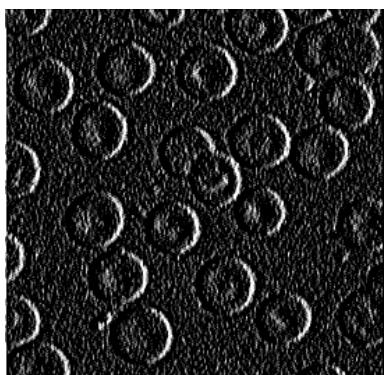
This quite simple method is used in photo manipulation softwares in order to artificially increase the focus of an image. The human visual perception positively responds to this type of filter.



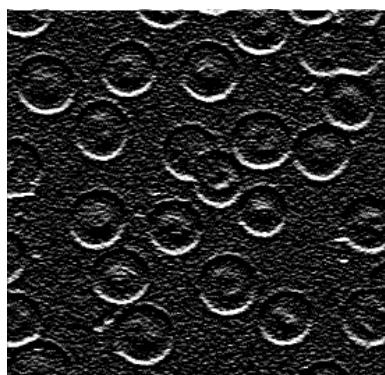
```

B=imread(' osteoblaste .bmp');
2 B=double(B);
B=B/255;

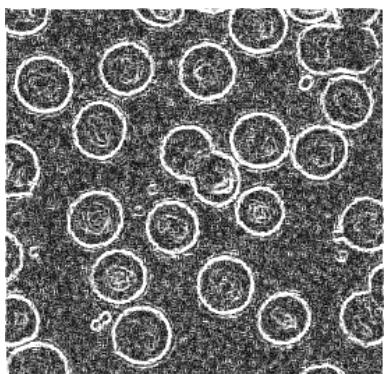
```



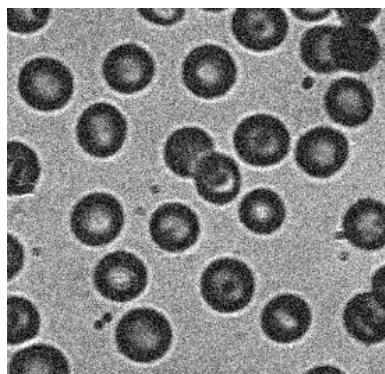
(a) Horizontal direction.



(b) Vertical direction.



(c) Norm of the gradient.



(d) Original image.

Figure 1.9: Prewitt derivative filter.

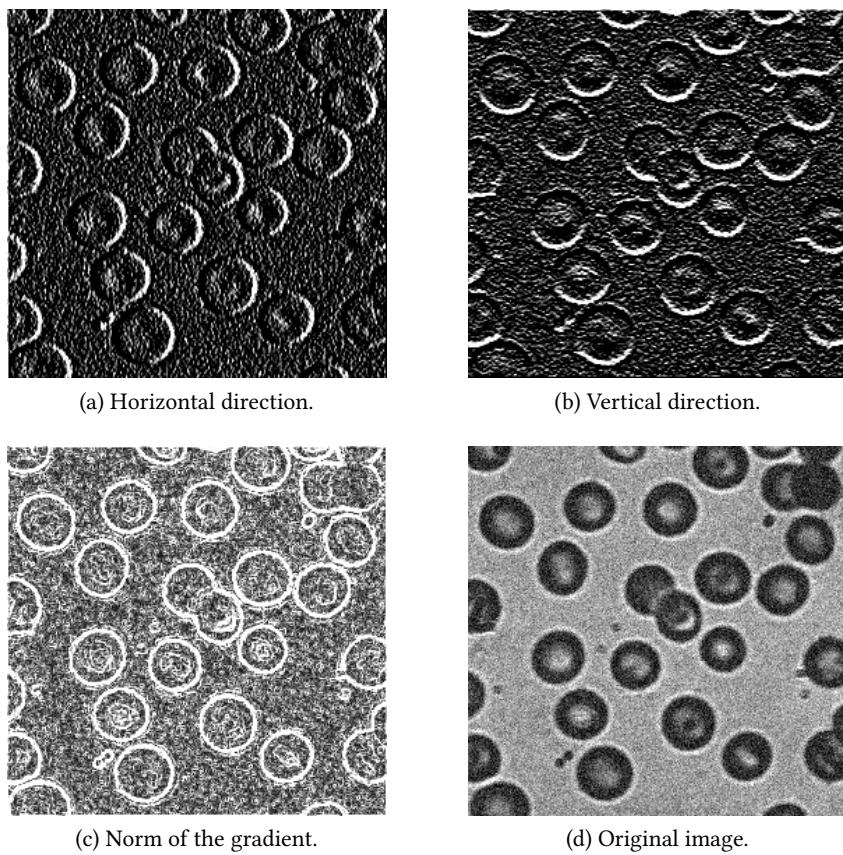


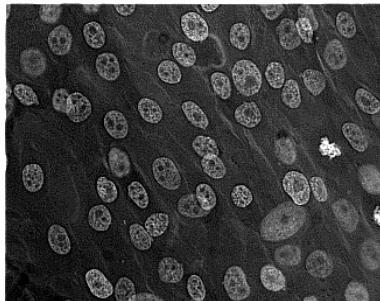
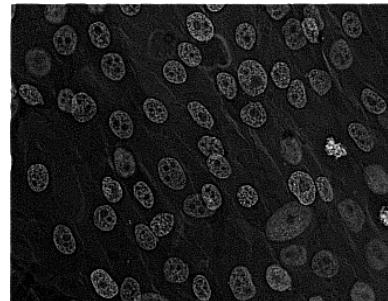
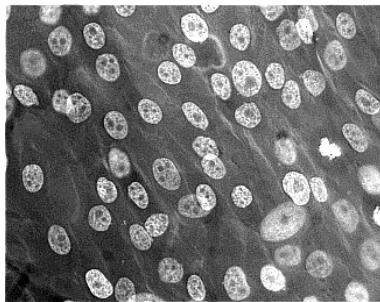
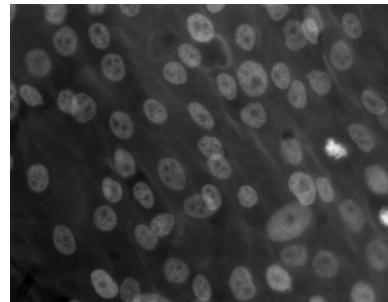
Figure 1.10: Sobel derivative filter.



```

4 hlaplacien = [-1 -1 -1; -1 8 -1;-1 -1 -1];
Blaplacien = imfilter (B, hlaplacien );
6 Benhance1=B+Blaplacien;
Benhance2=0.5*B+Blaplacien;
8 Benhance3=2*B+Blaplacien;
figure
10 subplot (221) ;imshow(B); title (' Original ');
subplot (222) ;imshow(Benhance1);title (' Enhancement : 1 ');
12 subplot (223) ;imshow(Benhance2);title (' Enhancement : 0.5 ');
subplot (224) ;imshow(Benhance3);title (' Enhancement : 2 ');

```

(a)  $\alpha = 1$ .(b)  $\alpha = 0.5$ .(c)  $\alpha = 2$ .

(d) Original image.

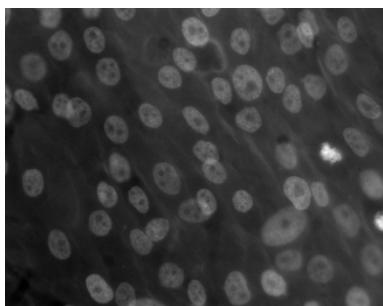
Figure 1.11: Image enhancement:  $I = \alpha \cdot I + HP(I)$ , where  $HP$  is a high-pass filter (the Laplacian filter in these illustrations).

## 1.11.10 Open question

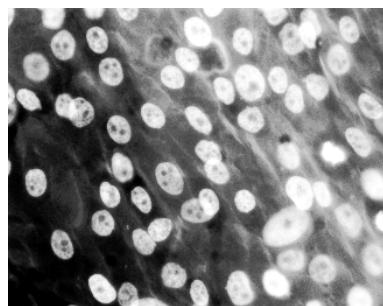
The histogram equalization is a method that will further developed. The result is presented in Fig.1.12.



```
1 figure  
Bhisteq=histeq(B);  
3 subplot(131);imshow(B);title('original image');  
subplot(132);imshow(Benhance3);title('enhancement by laplacian');  
5 subplot(133);imshow(Bhisteq);title('histogram equalization enhancement');
```



(a) Original image.



(b) Histogram equalization.

Figure 1.12: Image enhancement by histogram equalization. When extreme intensity values are present in a image (white or black values), histogram stretching is useless. The histogram equalization can thus be a solution in order to enhance the image.

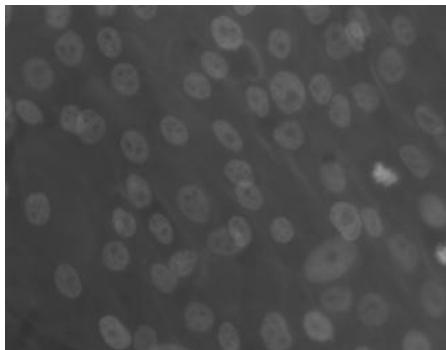
## ★ ★ 2

# Image Enhancement

The objective of this tutorial is to implement some image enhancement methods, based on intensity transformations or histogram modifications. It will make use of statistical notions like probability density functions or cumulative distribution functions.

Figure 2.1: The different processes of this tutorial will be applied on these images.

(a) Osteoblasts.



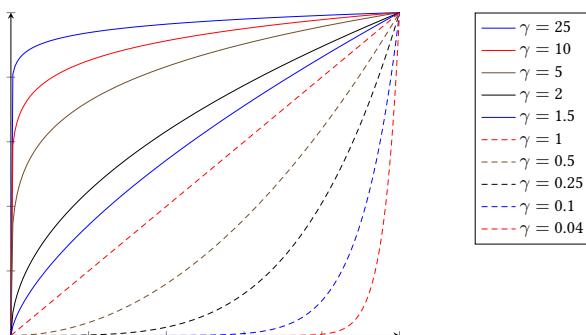
(b) Phobos (ESA/DLR/FU Berlin, CC-By-SA).



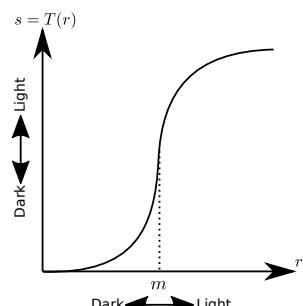
## 2.1

# Intensity transformations (LUT)

Two transformations will be studied: the  $\gamma$  correction and the contrast stretching. They enable the intensity dynamics of the gray tone image to be changed. These two operators are based on the following Look Up Tables (LUT):



(c)  $\gamma$  correction LUT.



(d) Contrast stretching LUT.



1. Test the transformation ' $\gamma$  correction' on the image 'osteoblast'.
2. Implement the operator 'contrast stretching' with the following LUT (also called cumulative distribution function cdf), with  $m$  being the mean gray value of the image, and  $r$  being a given gray value:

$$s = T(r) = \frac{1}{1 + (m/r)^E}$$

3. Test this transformation with different values of  $E$  on the image 'osteoblast'.



You can have a look at `imadjust` for  $\gamma$  correction.

## 2.2

## Histogram equalization

The objective is to transform the image so that its histogram would be constant (and its cumulative distribution function would be linear). The notations are:

- $I$  is the image of  $n$  pixels, with intensities between 0 and  $L$ .
- $h$  is the histogram, defined by:

$$h_I(k) = p(x = k) = \frac{n_k}{n}, \quad 0 \leq k < L$$

The following transformation  $T(I)$  is called histogram equalization.

$$T(x_k) = (L - 1) \sum_{j=0}^k p(x_j)$$

where

$$\text{cdf}_I(k) = \sum_{j=0}^k p(x_j)$$

is the cumulative distribution function (cumulative histogram).



1. Compute and visualize the histogram of the image 'osteoblast'.

2. Test this histogram equalization transformation on the image 'osteblast' (with builtin functions) and visualize the resulting histogram.
3. Code your own function.
4. The corresponding LUT to this transformation is the cumulative sum of the normalized histogram. Evaluate and visualize this intensity transformation.



See the `imhist`, `histcounts` and `histeq` functions.

## 2.3

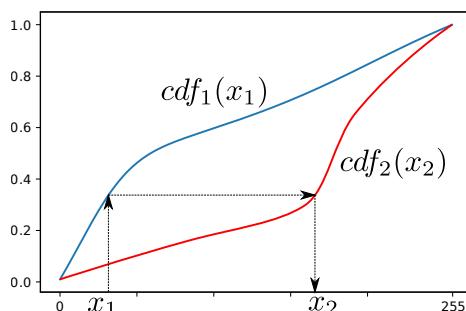
# Histogram matching

The objective is to enhance the original image by matching its histogram with a modeled one. The principle is to transform the gray value  $x_1$  of first image into  $x_2$ :  $T(x_1) = x_2$  (see Fig.2.2). Based on the fact that  $cdf_1(x_1) = cdf_2(x_2)$ , the formula to find  $x_2$  is:

$$x_2 = cdf_2^{-1}(cdf_1(x_1)).$$

As  $x_1$  and  $x_2$  are discrete values, this requires an interpolation.

Figure 2.2: Histogram matching principle.



1. Visualize the histogram of the image 'phobos'.
2. Make the histogram equalization and visualize the resulting image.
3. Construct a bi-modal histogram (for example) and code your own function for histogram matching.



See [interp1](#) for interpolation and LUT application.



## 2.4. Matlab correction



### 2.4.1 Intensity transformations

At first, the image is normalized between 0 and 1.



```
A=imread('osteoblaste.tif');
2 A=double(A);
A=A/255; % ensure values between 0 and 1
```

#### Gamma transform

The MATLAB® function imadjust is used to adjust gamma. Be careful that the range given in argument does not imply a strict gamma correction. Results are shown in Fig.2.3.



```
figure
2 subplot (2,2,1) ;imshow(A); title ('original image');

4 Ar=imadjust(A,[min(min(A)) max(max(A))],[0 1],1);
 subplot (2,2,2) ;imshow(Ar); title ('enhanced, g=1');

6 Ar=imadjust(A,[0.25 0.75],[0 1],0.5);
8 subplot (2,2,3) ;imshow(Ar); title ('enhanced, g=0.5');

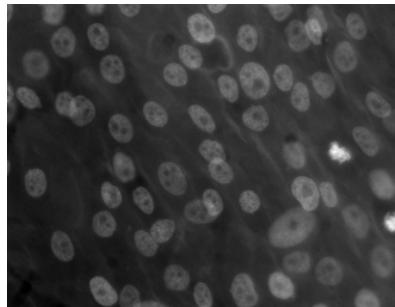
10 Ar=imadjust(A,[0.25 0.5],[0 1],2);
 subplot (2,2,4) ;imshow(Ar); title ('enhanced, g=2');
```

#### Contrast stretching

The function used will tend to saturate values, see Fig.2.4.



```
1 m=mean(mean(A));
 figure
3 subplot (2,2,1) ;imshow(A); title ('original image');
 Ar=1./(1+(m./(A+eps)).^5);
5 subplot (2,2,2) ;imshow(Ar); title ('contrast stretching : E=5');
 Ar=1./(1+(m./(A+eps)).^10);
7 subplot (2,2,3) ;imshow(Ar); title ('contrast stretching : E=10');
```



(a) Original image.

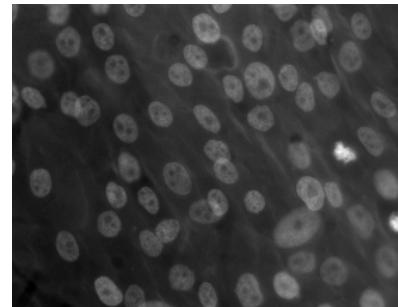
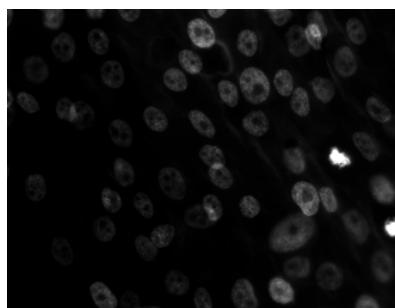
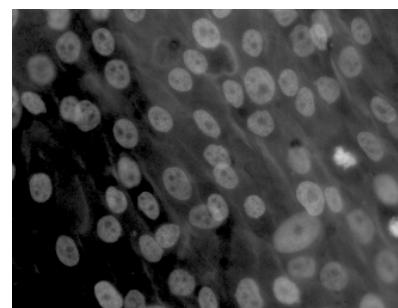
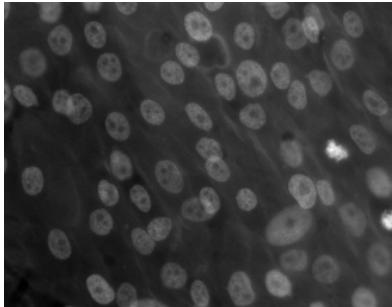
(b)  $\gamma = 1$ .(c)  $\gamma = 2$ .(d)  $\gamma = 0.5$ .

Figure 2.3: Gamma transform. Notice that these transforms are not exactly gamma transforms (due to the range in argument).



```
Ar=1./(1+(m./(A+eps)).^1000);
9 subplot (2,2,4) ;imshow(Ar);title (' contrast stretching : E=1000');
```



(a) Original image.

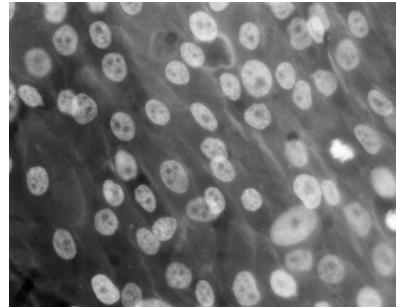
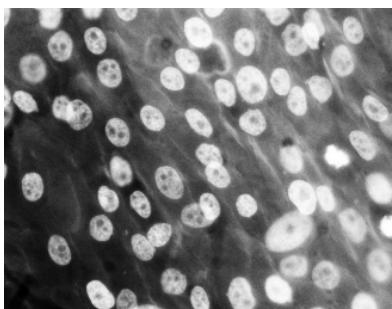
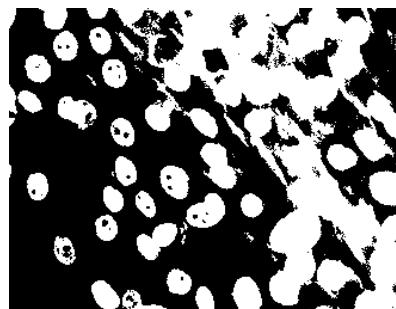
(b)  $E = 5$ .(c)  $E = 10$ .(d)  $E = 1000$ .

Figure 2.4: Contrast stretching.

## 2.4.2 Histogram equalization

The principle of this operation is to evaluate a lookup-table in order to perform the transformation. This LUT is the cumulative distribution function, and can be applied by using an interpolation function `interp1` or the LUT function `intlut`. The results are shown in Fig.2.5.



```
A=imread('osteoblaste . tif ');
2 Ar=histeq(A);
figure
4 subplot (3,2,1) ;imshow(A);title (' original image');
5 subplot (3,2,2) ;imshow(Ar);title (' histogram equalization ');
6 subplot (3,2,3) ;imhist(A);title (' original histogram');
```



```

1 subplot (3,2,4) ; imhist(Ar); title ('equalized histogram');
2 hnrm = imhist(A) ./ numel(A);
3 cdf=255.* cumsum(hnrm);
4 subplot (3,2,5) ; plot (1:1:256, cdf); title ('LUT (cdf)');
5 axis ([0 255 0 255]);

```

The previous code uses the MATLAB® function for histogram equalization. You can code it as follows:



```

1 function I2 = histo_eq(I)
2 %
3 % histogram equalization , version with look-up-table
4 % I: original image, with values in 8 bits integer
5 %
6 [ hist , edges] = histcounts(I, 0:256);
7 cdf = cumsum(hist);
8 cdf = cdf / cdf(end);
9 %
10 % the LUT could be applied by this function :
11 %I2 = intlut (I, uint8(255*cdf));
12 %
13 I2 = interp1(edges(1:end-1), cdf, double(I (: )));
14 I2 = uint8(255 * I2);
15 I2 = reshape(I2, size(I));

```

### 2.4.3 Histogram matching

The histogram matching will be applied in the Phobos image (see credits).



```

1 A=imread('phobos.jpg');

```

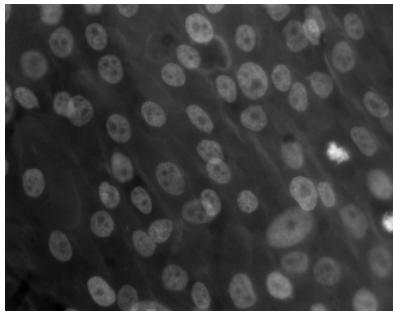
First, the function to generate the probability density function is an addition of two Gaussian functions (normalized):



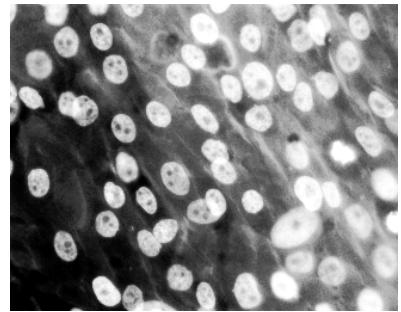
```

1 function p =twomodegauss(m1, sig1, m2, sig2, A1, A2,k)
2 c1 = A1 *(1/((2* pi) ^ .5* sig1));
3 k1 =2*( sig1 ^ 2);
4 c2 = A2 *(1/((2* pi) ^ .5* sig2));
5 k2 = 2*( sig2 ^ 2);
6 z = linspace (0,1,256) ;

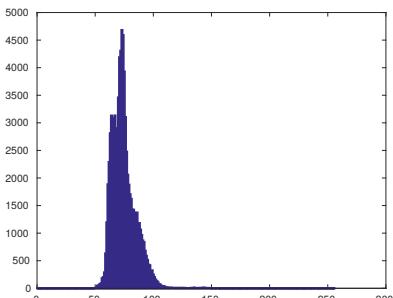
```



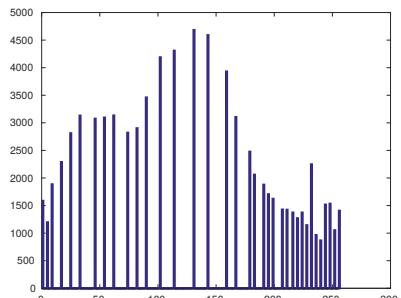
(a) Original image.



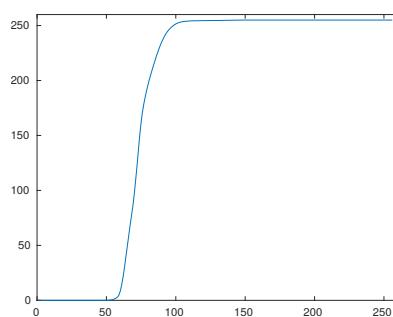
(b) Histogram equalization.



(c) Histogram of original image.



(d) Histogram of enhanced image.



(e) LUT (cumulative distribution function).

Figure 2.5: Histogram equalization.



```
7 p = k+c1*exp(-((z-m1).^ 2)./ k1)+ c2 *exp(-((z-m2).^ 2)./k2);
p = p./ sum(p(:));
```

Then, the histogram matching is performed as follows:



```
% histogram to match
2 p=twomodegauss (0.05,0.1,0.8,0.2,0.04,0.01,0.002) ;

4 % histogram matching, \ matlabregistered {} version
Ar=uint8(histeq(A,p));
6 figure
subplot (3,2,6) ;plot (p); title ('model of bi-modal histogram');
8 xlim([0 255])
subplot (3,2,1) ;viewImage(A); title ('original image');
10 subplot (3,2,2) ;viewImage(Ar); title ('enhanced image');
12 subplot (3,2,3) ;imhist(A,256); title ('original histogram');
12 subplot (3,2,4) ;imhist(Ar,256); title ('matched histogram');
```

The histeq MATLAB® function handles histogram matching. The following code is also proposed. These are illustrated in Fig.2.6.



```
function I2 = histo_matching(I, cdf_target)
2 %
% histogram matching, version with look-up-table
4 % I: original image, with values in 8 bits integer
%
6 [hist , edges] = histcounts (I, 0:256) ;
cdf = cumsum(hist);
8 cdf = cdf / cdf(end);

10 % 1st apply histogram equalization
LUT = interp1(edges(1:end-1), cdf, double(I (:)));
12
% the apply inverse transformation to match target cdf
14 im2 = interp1 ( cdf_target , edges(1:end-1), LUT);

16 % finally , reshape and convert to uint8
I2 = reshape(im2, size (I,1) , size (I,2));
18 I2 = uint8(I2);
```

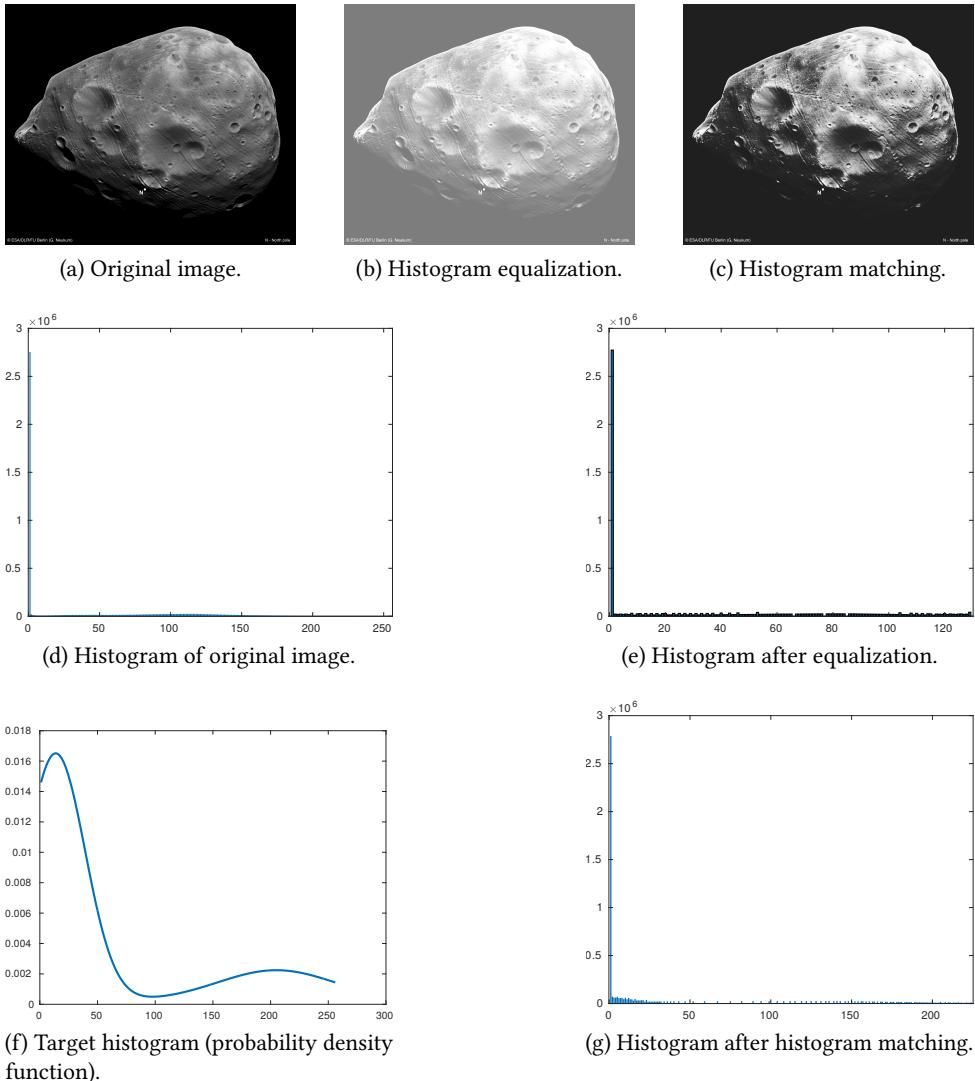


Figure 2.6: Histogram matching.



## 3 2D Fourier Transform

The main objective of this tutorial is to study image filters applied in the frequencial or spatial domain with the Fourier transform.

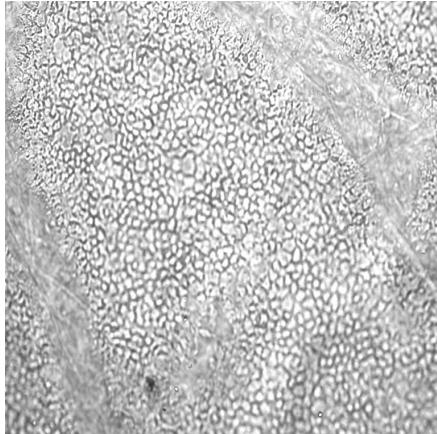


Use `fft2` , `ifft2` , `fftshift` functions to compute the Fourier Transform, `angle` and `abs` for phase and amplitude.

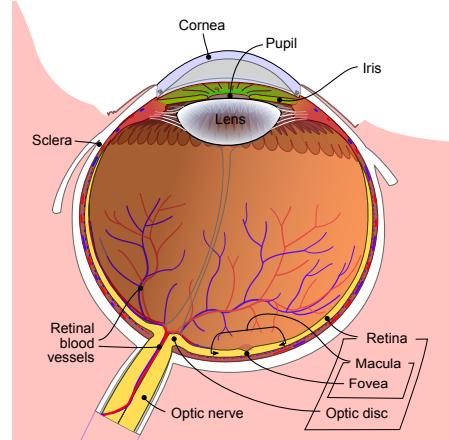
The image to be used comes from a human cornea endothelium observed ex vivo by optical microscopy.

Figure 3.1: Image of human cornea endothelium observed by optical microscopy. The ophthalmologists would like to know the cell density, without manually counting all the cells.

(a) Human corneal endothelium, extracted from a donor and observed here before grafting.



(b) Human eye (from Wikipedia, authors: Rhcastilhos and Jmarchn, CC-By-SA).



### 3.1

## Fourier transform



1. Load an image and visualise it.
2. Compute the Fourier Transform by the fft algorithm.
3. Visualise the images of the phase and amplitude of the Fourier Transform.

### 3.2

## Inverse Fourier transform

In this exercise, it can be interesting to consider different images, like a Lena picture for example.



1. Apply the inverse Fourier transform on the Fourier transform to find the original image.
2. Now, apply the inverse Fourier transform on the phase information only (without using the frequency informations).
3. In a same spirit, apply the inverse Fourier transform on the frequency informations only (without the phase).

### 3.3

## Low-pass and high-pass filtering



1. Modify the Fourier transform of the image to
  - keep only low frequencies,
  - keep only high frequencies.
2. Apply the inverse Fourier transform on both and comment.

**3.4**

## Application: evaluation of cellular density

The ophthalmologists would like to evaluate the cell density of the cornea endothelium observed in Fig. 3.1.



1. Compute the Fourier transform of the image. If one considers that the cells constitute a repeated pattern on the whole image, locate the repetition frequency on the amplitude image.
2. Can this frequency be linked to the cell density ?

The answer to this last question is obviously yes. Here follows a simple method to evaluate the cell density (or the mean cell radius), if these cells are considered as circular [?].



1. The amplitude information is very noisy. First of all, a gaussian filter should be applied (see functions `fspecial` and `imfilter` ).
2. Find a simple way to evaluate the mean radius of the cells.



### 3.5. Matlab correction



#### 3.5.1 Fourier transform

In order to display the Fourier transform, the following function is used to show both spectrum and phase. Notice the logarithmic function.



```

1 function viewImageSpectrum(I,F)
2 % I: original image
3 % F: Fourier transform of I
4
5 subplot (1,3,1) ;
6 imshow(I,[]);
8 % phase
9 subplot (1,3,3) ;
10 Im=angle(S);
11 imshow(Im,[]);
12
13 % amplitudes
14 subplot (1,3,2) ;
15 Ia=abs(S);
16 Ia2=log(1+Ia);
17 imshow(Ia2,[]);

```

The `fftshift` functions centers the frequency  $(0, 0)$  in the image. The following functions will be used:



```

% Fourier transform utility of image I
1 function S=FT(I)
2 S= fftshift ( fft2 (double(I)));

```



```

1 % Inverse Fourier transform utility of spectrum S
2 function I=iFT(S)
3 I=real( ifft2 ( fftshift (S)));

```

### 3.5.2 Inverse Fourier Transform

The important thing to notice in the 2D Fourier transform is that the information is present in the phase, not in the amplitude. The following code highlights this property (see also Fig.3.2).



```

1 Spectrum=FT(A);

3 % amplitude and phase
amplitude = abs(Spectrum);
5 phase = angle(Spectrum);

7 % reconstruction with amplitude only
C=real( ifft2 ( fftshift (amplitude)));
9 figure ;imshow(C,[]); title ('Reconstruction from amplitude only');

11 % reconstruction with phase only
D=real( ifft2 ( fftshift ((exp(1i*phase))))) ;
13 figure ;imshow(D,[]); title ('Reconstruction from phase only');

```

### 3.5.3 Low-pass and high-pass filtering

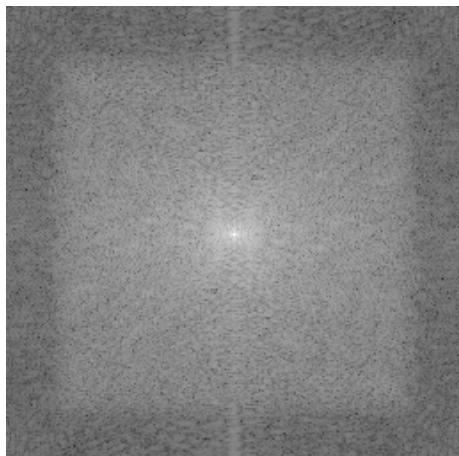
This is done by selecting only low or high frequencies, respectively. A binary window is employed.



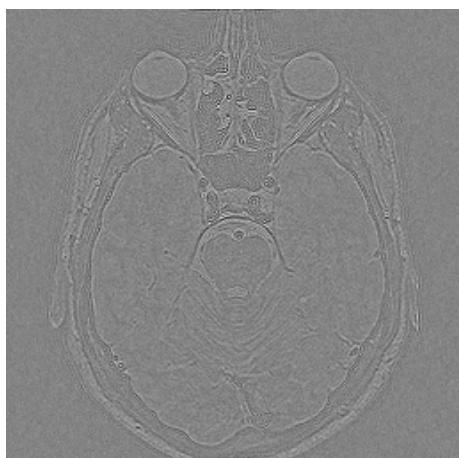
```

1 function Sr=LPfilter (S, fC)
% low pass filtering
3 % S: spectrum (Fourier Transform)
% fC: cut-off frequency
5 fC=floor(fC);
if(fC<=0 | 2*fC >= size(S,2) | 2*fC >= size(S,1) )
7 disp('Wrong cut-off frequency');
Sr=0;
9 return;
end
11 So=zeros( size(S));
13 So(( size(S,1)/2-fC):( size(S,1)/2+fC), ( size(S,2)/2-fC):( size(S,2)/2+fC))=1;
Sr=S.*So;

```



(a) Amplitude only.

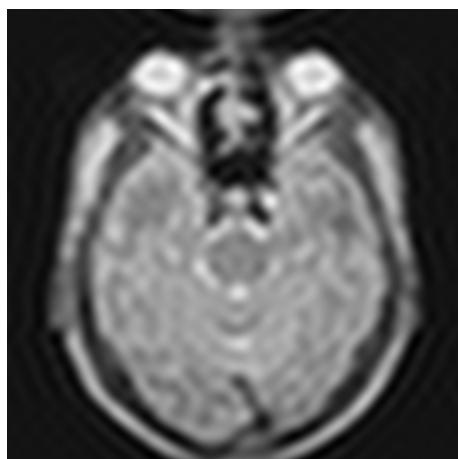


(b) Phase only.

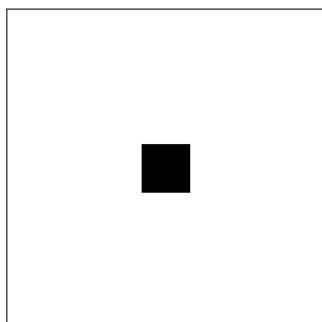
Figure 3.2: Reconstruction of partial informations (phase or amplitude only). Notice that the main visual informations are contained in the phase, and not in the amplitude.



(a) High Pass filter.



(b) Low Pass filter.



(c) Filtering (binary) mask.

Figure 3.3: Fourier basic filtering of the brain image.



```

function Sr=FiltrePH(S, fC)
2 % High pass filter
% S: spectrum (fourier transform)
4 % fC: cut-off frequency

6 fC=floor(fC);
if(fC<=0 | 2*fC >= size(S,2) | 2*fC >= size(S,1) )
8 disp(' Taille de coupure incorrecte ');
Sr=0;
10 return ;
end
12
So=ones(size(S));
14 So((size(S,1)/2-fC):(size(S,1)/2+fC), (size(S,2)/2-fC):(size(S,2)/2+fC))=0;
Sr=S.*So;

```

The two previous functions are used to filter the image, which is done with:



```

1 Spectre_PB0=FiltrePB(Spectrum,20);
Spectre_PH0=FiltrePH(Spectrum,110);
3 % inverse Fourier transform
A_PB0=iFT(Spectre_PB0);
5 A_PH0=iFT(Spectre_PH0);
% display results
7 viewImageSpectre(A_PB0,Spectre_PB0);title ('Low-pass filter');
viewImageSpectre(A_PH0,Spectre_PH0);title ('High-pass filter');

```

### 3.5.4 Application: evaluation of corneal cell density

The principle consists in isolating the annulus (see Fig.3.4) that corresponds to a frequency of repetition of the cells, that can lead us to a cell density. First of all, the image is loaded and the FFT is applied.



```

A=imread('cornee.tif ');
2 % spectrum
Spectrum=FT(A);
4 % amplitude and phase computation
amplitude = abs(Spectrum);
phase = angle(Spectrum);
6

```

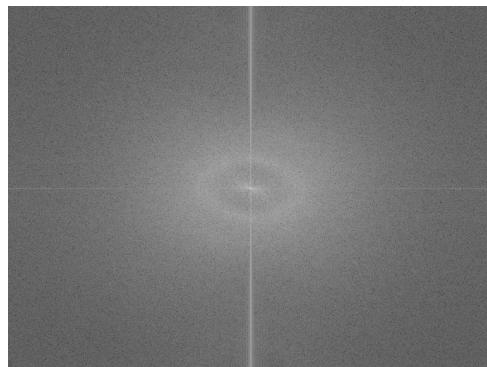


Figure 3.4: Amplitude of the spectrum of the cornea image. Notice the circular shape that denotes a certain regularity of the cellular pattern, and can be used in order to evaluate the cell density.

Then, the amplitude is filtered. A central line is kept. The objective is now to extract the second peak. The results of the `findpeaks` function are displayed in Fig.3.5.



```

% Filtering of
2 PSF = fspecial ('gaussian' ,30,30) ;
Blurred = imfilter (amplitude,PSF,'symmetric','conv');
4
V = Blurred (:, end/2);
6 plot (V);

8 % In order to find the peaks, the method is elementary:
% we are looking for the 2nd peak, at length(V)/2
10 [pk, locs] = findpeaks(V);
hold on
12 plot(locs, pk, 'sr');
locs2 = sort(abs(length(V)/2-locs))

14
% result is displayed
16 disp('frequency of repetition :')
f=locs2(2)/length(V)
18 disp('cornea diameter: ')
d=1/f

```

Command window

```
1 locs2 =
2     1
3     49
4     50
5     200
6     214
7     224
8     225
9     234
10    274
11    277
12    281
13    281
14    284
15    285
16    286
17
18 frequency of repetition :
19 f =
20     0.0851
21
22 cornea diameter:
23 d =
24     11.7551
25 >>
```

More informations can be see in [?, ?, ?, ?].

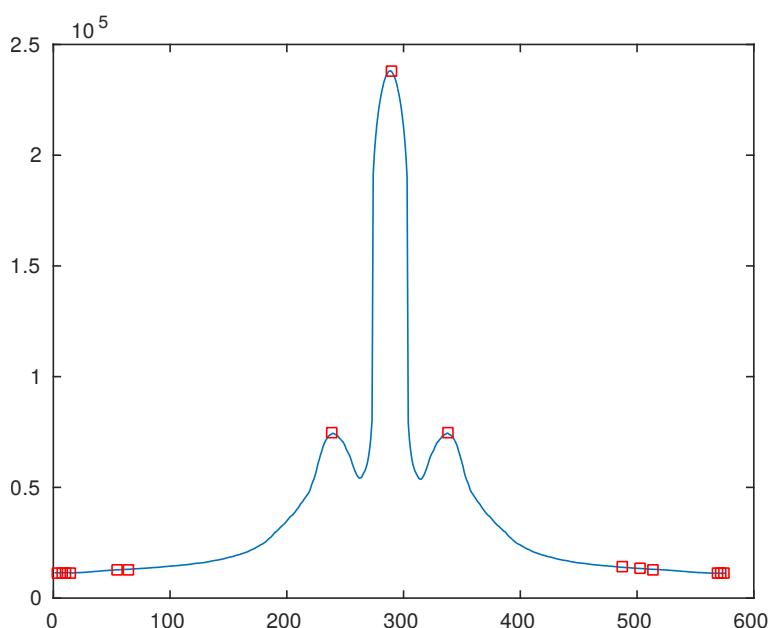


Figure 3.5: Detected peaks.





## 4

# Introduction to wavelets

This tutorial introduces practically the basic wavelet decomposition and reconstruction algorithms. The objectives are to code some basic programs that will decompose and reconstruct 1D and 2D signals.

## 4.1

### Introduction

Wavelets are based on a mother wavelet  $\Psi$  ( $s$  is a scale parameter,  $\tau$  is the time translation factor  $(s, \tau) \in \mathbb{R}_+^* \times \mathbb{R}$ ):

$$\forall t \in \mathbb{R}, \psi_{s,\tau}(t) = \frac{1}{\sqrt{s}} \Psi \left( \frac{t - \tau}{s} \right)$$

The continuous wavelet transform is written as follows, where  $\psi^*$  means the complex conjugate of  $\psi$  and  $\langle \cdot, \cdot \rangle$  is the complex scalar product (Hermitian form):

$$g(s, \tau) = \int_{-\infty}^{\infty} f(t) \psi_{s,\tau}^*(t) ds d\tau = \langle f, \psi_{s,\tau} \rangle$$

The reconstruction is defined by:

$$f(t) = \frac{1}{C} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \frac{1}{|s|^2} g(s, \tau) \psi_{s,\tau}(t) ds d\tau$$

with

$$C = \int_{-\infty}^{\infty} \frac{|\hat{\Psi}(\omega)|^2}{|\omega|} d\omega$$

and  $\hat{\Psi}$  is the Fourier Transform of  $\Psi$ .

The discrete wavelet transform corresponds to a sampling of the scales. To compute the different scales, one has to introduce a “father” wavelet, that similarly defines a family of functions orthogonal to the family  $\psi_{s,\tau}$ .

## 4.2

### Fast discrete wavelet decomposition / reconstruction

A simple algorithm (cascade algorithm, from Mallat) is defined as two convolutions (by a lowpass  $ld$  filter for the projection on the  $\psi$  family, and a high pass  $hd$  filter for the orthogonal projection) followed by a subsampling (see Fig. 4.1).

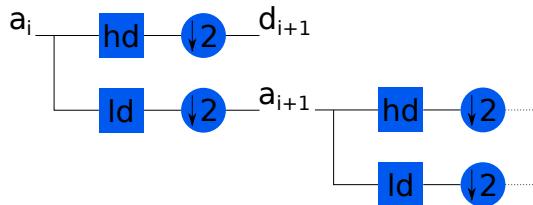


Figure 4.1: Algorithm for wavelet decomposition. First, a convolution is performed (square node), then, a subsampling.  $a_i$  stands for decomposition at scale  $i$ ,  $d_i$  stands for detail at scale  $i$ .

#### 4.2.1 Simple 1D example

Let's consider the signal  $[4; 8; 2; 3; 5; 18; 19; 20]$ . We will use the Haar wavelets defined by  $ld = [1; 1]$  and  $hd = [-1; 1]$ . Basically, these filters perform a mean and a difference (see Table 4.1). The result of the decomposition in 3 scales with these wavelets is the concatenation of the details and the final approximation

$$C = \{[-4; -1; -13; -1]; [7; -16]; [-45]; [79]\}$$

For the sake of simplicity, it is recommended to use the structure cell of MATLAB® to store the decomposition of all detail vectors as well as the final approximation vector.



Scale $i$	Approximation $a_i$	Details $d_i$
0 (original signal)	$[4; 8; 2; 3; 5; 18; 19; 20]$	
1	$[12; 5; 23; 39]$	$[-4; -1; -13; -1]$
2	$[17; 62]$	$[7; -16]$
3	$[79]$	$[-45]$

Table 4.1: Illustration of Haar decomposition in a simple signal.



In the case of these Haar wavelets, code a function that performs the decomposition for a given number of scales. The prototype of this function will be:

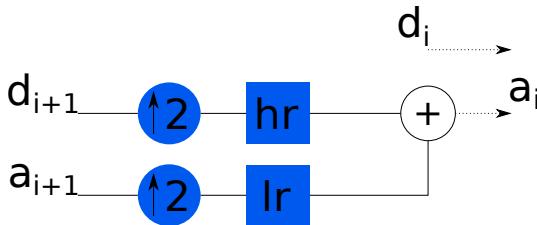


```
function C=simpleWaveDec(signal1D, nb_scales)
% wavelet decomposition of <signal1D> into <nb_scales> scales
```

### 4.2.2 Reconstruction

To reconstruct the original signal (see Fig. 4.2), we need the definition of two reconstruction filters,  $hr$  and  $lr$ . For the sake of simplicity, we use  $lr = ld/2$  and  $hr = -hd/2$ . These filter will perform an exact reconstruction of our original signal.

Figure 4.2: Reconstruction algorithm. The oversampling is done by inserting zeros.



Using this algorithm, you can now go on to the next exercise.



Code a function that performs the reconstruction of the signal. The prototype of this function will be, with the previous definition of C:



```
function signal=simpleWaveRec(C)
% wavelet simple reconstruction function of a 1D signal
% C: Wavelet coefficients in cell of arrays
% signal: reconstructed signal
```

## 4.3

## 2D wavelet decomposition

Let  $A$  be the matrix of an image. We consider that  $A$  is of size  $2^n \times 2^n$ ,  $n \in \mathbb{N}$ . We consider, as for the 1D transform, the filters  $ld$  and  $hd$ .

The wavelet decomposition is as follows:

- Apply  $ld$  and  $hd$  on rows of  $A$ . Results are denoted  $ld_r A$  and  $hd_r A$ , of size  $2^n \times 2^{n-1}$ , with  $r$  standing for row.
- Then, apply  $ld$  and  $hd$  again, to get the four new matrices:  $ld_c ld_r A$ ,  $ld_c hd_r A$ ,  $hd_c ld_r A$  and  $hd_c hd_r A$ , of sizes  $2^{n-1} \times 2^{n-1}$ , with  $c$  standing for column. The matrix  $ld_c ld_r A$  is the approximation, and the other matrices are the details.



- Code a function to perform the wavelet decomposition and for a given number of scales (lower than  $n$ ). Test it on images of size  $2^n \times 2^n$ .
- Code the reconstruction function.

## 4.4

## Built-in functions

Let us explore some built-in functions.

### 4.4.1 Continuous wavelet decomposition

One has to make the difference between the continuous wavelet transform and the discrete transform.

The MATLAB® function that performs continuous wavelet transform is `cwt`. A GUI dedicated to wavelets is available in MATLAB®: type the command `wavemenu` and try some 1D and 2D signals.



```

load noissin; % load a signal1D
%
% perform continuous wavelet transform at scales specified by
%
c = cwt(noissin, 1:48, 'db4', 'plot');

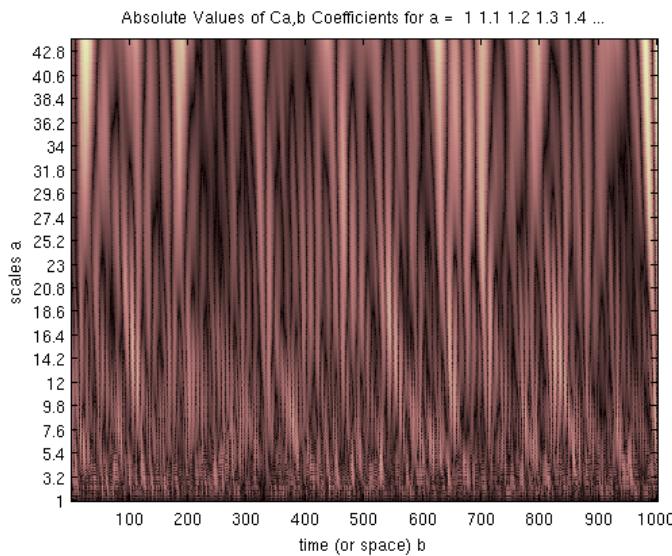
```



#### Informations

In `scipy.signal` you can find function to perform wavelets decomposition, and among them `cwt` for the continuous wavelet transform. There is also the module `pywt` that may be more developed (see also `cwt`).

Figure 4.3: Continuous wavelet transform.



#### 4.4.2 Discrete decomposition



1. Generate the signal  $\sin(2\pi * f_1 * t) + \sin(2\pi * f_2 * t)$ , with  $f_1 = 3\text{Hz}$  and  $f_2 = 50\text{Hz}$ .
2. Apply the wavelet transformation for a given wavelet (like 'db4').
3. Display the 5th approximation of the wavelet decomposition.
4. Display the 4th detail coefficients of the wavelet decomposition.

Each of the MATLAB® function is available for 1D, 2D or even 3D signals.  
You can test the following functions wavedec, wavedec2 or wavedec3:



```
[C,L] = wavedec(signal, scale, 'wavelet_name')
2 [C,L] = wavedec(signal, scale, ld, hd)
```



## 4.5. Matlab correction



This correction illustrates the wavelet decomposition in 1D or 2D.

### 4.5.1 1D signals

Two functions are required: a function that loops over the different scales and calls the second function that performs the single step wavelet decomposition. The Haar wavelet is illustrated in Fig.4.4.

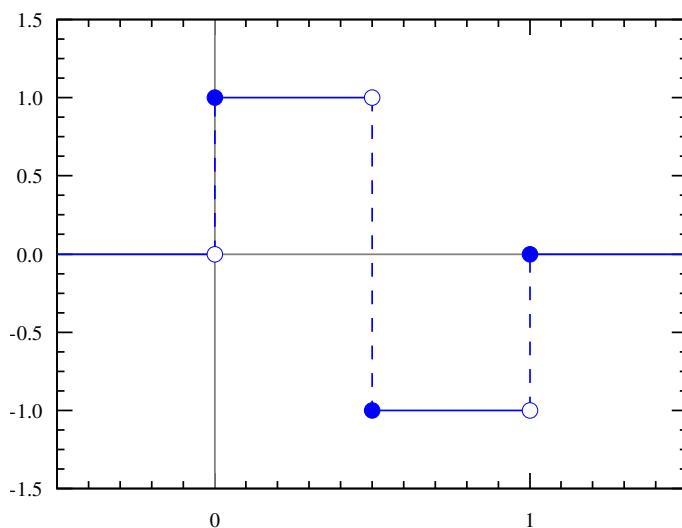


Figure 4.4: Haar wavelets. From wikipedia, author Omegatron.

### Simple 1D decomposition



```

function C = simpleWaveDec(signal, nb_scales)
% wavelet decomposition of <signal> into <nb_scales> scales
% This function uses Haar wavelets for demonstration purposes.
%
% Haar Wavelets filters for decomposition and reconstruction
ld = [1 1];
hd = [-1 1];
%
% transformation
C=cell(nb_scales+1, 1);
A = signal; % approximation
for i=1:nb_scales
    C{i} = filter(hd, ld, A);
    A = filter(ld, hd, A);
end

```



```

[A, D] = waveSingleDec(A, ld, hd);
% get the coefficients
C{i}=D;
end
C{nb_scales+1}=A;

```



```

% function single step wavelet decomposition
function [A, D]=waveSingleDec(signal, ld, hd)
% 1D wavelet decomposition into
% A: approximation vector
% D: detail vector
% ld: low pass filter
% hd: high pass filter

% convolution
A = conv(signal, ld, 'same');
D = conv(signal, hd, 'same');

% subsampling
A = A(1:2:end);
D = D(1:2:end);

```

### Simple 1D reconstruction

The reconstruction starts from the highest scale and computes the approximation signal with the given details.



```

function A = simpleWaveRec(C)
% wavelet simple reconstruction function of a 1D signal
% C: Wavelet coefficients
%
% The Haar wavelet is used
ld = [1 1];
hd = [-1 1];
lr = ld/2;
hr = -hd/2;

nb_scales = length(C)-1;
A = C{nb_scales+1};
for i=nb_scales:-1:1
    A = waveSingleRec(A, C{i}, lr, hr);
end

```



```
1 function approx = waveSingleRec(a, d, lr, hr)
% 1D wavelet reconstruction at one scale
3 % a: vector of approximation
% d: vector of details
5 % lr: low pass filter defined by wavelet
% hr: high pass filter defined by wavelet
7 %
% This is Mallat algorithm.
9 % NB: to avoid side effects , the convolution function does not use the 'same'
    %> option

11 approx = zeros(1, length(a)*2);
approx(1:2:end) = a;
13 approx = conv(approx, lr);

15 detail = zeros(1, length(a)*2);
detail(1:2:end) = d;
17 detail = conv(detail, hr);

19 approx = approx + detail;
approx = approx(1:length(a)*2);
```

## Results

**Command window**

```

>> signal = [4;8;2;3;5;18;19;20];
2 >> C = simpleWaveDec(signal,3)
C =
4      [4x1 double]
5      [2x1 double]
6      [
7          -45]
8      [
9          79]

>> for i=1:4, C{i}, end
10 ans =
11      -4
12      -1
13      -13
14      -1

16 ans =
17      7
18      -16

20 ans =
21      -45
22

24 ans =
25      79

```

## 4.5.2 2D signals

### Decomposition



```

%% 2D simple wavelet decomposition
2 function [LcLrA, HcLrA, LcHrA, HcHrA] = decWave2D(image, ld, hd)
% wavelet decomposition of a 2D image into four new images.
4 % The image is supposed to be square, the size of it is a power of 2 in the x and
% → y dimensions.

6 % We manipulate doubles
image = double(image);
8

%% Decomposition on rows
10 sx=size(image, 1);
sy=size(image, 2);
12 LrA = zeros(sx, sy/2);

```



```

14 HrA = zeros(sx, sy/2);
for i=1:sx
    [A, D]= waveSingleDec(image(i,:), ld, hd);
    LrA(i,:) = A;
    HrA(i,:) = D;
end
20

22 %% Decomposition on cols
LcLrA = zeros(sx/2, sy/2);
24 HcLrA = zeros(sx/2, sy/2);
LcHrA = zeros(sx/2, sy/2);
26 HcHrA = zeros(sx/2, sy/2);
for j=1:sy/2
    [A, D]= waveSingleDec(LrA(:,j), ld, hd);
    LcLrA(:,j) = A;
    HcLrA(:,j) = D;
30

32 [A, D]= waveSingleDec(HrA(:,j), ld, hd);
LcHrA(:,j) = A;
34 HcHrA(:,j) = D;
end
36

38 %% Display result
figure();
subplot(2, 2, 1); imshow(LcLrA, []);
40 subplot(2, 2, 2); imshow(HcLrA, []);
subplot(2, 2, 3); imshow(LcHrA, []);
42 subplot(2, 2, 4); imshow(HcHrA, []);

```



```

function C = simpleImageDec(image, nb_scales)
2 % wavelet decomposition of <image> into <nb_scales> scales
% This function uses Haar wavelets for demonstration purposes.
4
% Haar Wavelets filters for decomposition and reconstruction
6 ld = [1 1];
hd = [-1 1];
8
% transformation
10 C=cell(nb_scales+1, 1);
A = image; % approximation
12
coeffs = cell (3,1);
14 for i=1:nb_scales
[A, HcLrA, LcHrA, HcHrA] = decWave2D(A, ld, hd);
16 coeffs {1} = HcLrA;
coeffs {2} = LcHrA;

```



```

18 coeffs {3} = HcHrA;
% set the coefficients
20 C{i}= coeffs ;
end
22 C{nb_scales+1} = A;

```

## 2D reconstruction



```

%% 2D simple wavelet reconstruction
2 function A = recWave2D(LcLrA, HcLrA, LcHrA, HcHrA, lr, hr)
% Reconstruction of an image from lr and hr filters and from the wavelet
% ← decomposition.
4 % A: resulting (reconstructed) image
%
6 % NB: This algorithm supposes the number of pixels in x and y dimensions is
% a power of 2.
8 [sx, sy] = size(LcLrA);
10
%% Allocate temporary matrices
12 LrA = zeros(sx*2, sy);
HrA = zeros(sx*2, sy);
14 A = zeros(sx*2, sy*2);

16 %% Reconstruct from cols
for j=1:sy,
18     LrA(:,j) = waveSingleRec(LcLrA(:,j), HcLrA(:,j), lr, hr);
     HrA(:,j) = waveSingleRec(LcHrA(:,j), HcHrA(:,j), lr, hr);
20 end

22 %% Reconstruct from rows
for i=1:sx*2,
24     A(i,:) = waveSingleRec(LrA(i,:), HrA(i,:), lr, hr);
end
26
%% Display reconstructed image
28 figure();
imshow(A,[]);

```



```

1 function A = simpleImageRec(C)
% wavelet reconstruction of an image described by the wavelet coefficients C
3

```



```
% The Haar wavelet is used
5 ld = [1 1];
hd = [-1 1];
7 lr = ld/2;
hr = -hd/2;
9
nb_scales = length(C)-1;
11 A = C{nb_scales+1};
for i=nb_scales:-1:1
13 A = recWave2D(A, C{i}{1}, C{i}{2}, C{i}{3}, lr, hr);
end
```

## Results

The illustration Fig. 4.5 is obtained by the following code. The useful functions are presented below.



```
>> I = imread('lena256.png')
>> C = simpleImageDec(I, 3);
3 >> A = displayImageDec(C);
>> imwrite(uint8(255*A), 'lena_3lvl.png');
```



```
function [ A ] = displayImageDec( C )
% Construct a single image from a wavelet decomposition
% C: the decomposition
4 % A: the entire illustration image

6 n_scales = length(C)-1;
[n, m] = size(C{1}{1});
8 A = zeros(2*n, 2*m);

10 prev = C{n_scales+1};
for s=n_scales:-1:1
12 ns = n / 2^(s-2);
ms = m / 2^(s-2);
14 A(1:ns, 1:ms) = imdec2im(prev, C{s});
prev = A(1:ns, 1:ms);
16 end

18 end
```



```

function A = imdec2im(LcLrA, lvlC)
2 % constructs a single image from:
% LcLrA: the approximation image
4 % lvlC: the wavelet decomposition at one level
%
6 % for display purposes

8 HcLrA=lvlC{1};
LcHrA=lvlC{2};
10 HcHrA=lvlC{3};
[n, m] = size(HcLrA);

12 A = zeros(2*n, 2*m);
14
% approximation image can be with high values when using Haar coefficients
16 A(1:n, 1:m) = LcLrA/ max(LcLrA(:));

18 % details are low, and can be negative
A(1:n, m+1:2*m) = imadjust(HcLrA, stretchlim(HcLrA), [0 1]);
20 A(n+1:2*n, 1:m) = imadjust(LcHrA, stretchlim(LcHrA), [0 1]);
A(n+1:2*n, m+1:2*m) = imadjust(HcHrA, stretchlim(HcHrA), [0 1]);

22 imshow(A)

```

### 4.5.3 Matlab functions

#### 1D

Example on a sample function:



```

1 % signal
t = 0:0.001:10;
3 signal = sin(2*pi*3*t) + .2* sin(2*pi*50*t);

5 % parameters
wavelet = 'db1'; % Daubechies wavelets
7 lvl = 5; % decomposition level

9 % Decomposition
[C, S] = wavedec(signal, lvl, wavelet);
11
% Reconstruction
13 srec = waverec(C, S, wavelet);

```

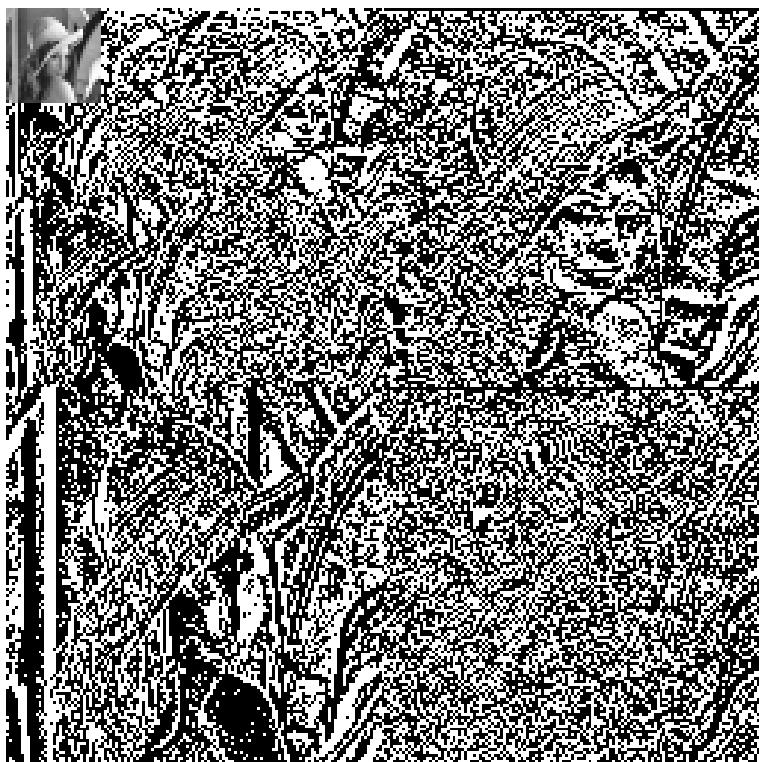


Figure 4.5: (Haar) Wavelet decomposition of the Lena image.

2D



```
1 % read an image
I = imread('Couche_18.png');
3 lvl = 8;

5 % decomposition
[C, S] = wavedec2(I, lvl, 'db4');

7 % threshold after lvl
9 newC = wthcoef2('a', C, S);

11 % reconstruction
I2 = waverec2(newC, S, 'db4');

13 % display result
15 imshow(I,[]);

17 figure(); imshow(I2,[]);
```





5

## Image restoration: denoising

This tutorial aims to study some random noises and to test different image restoration methods (image denoising).

The different processes will be applied on the following MR image.

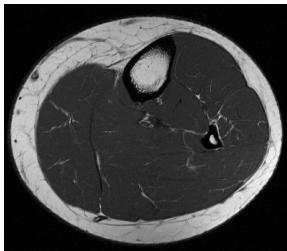


Figure 5.1: Leg.

### 5.1

## Generation of random noises

Some random noises are defined with the given functions (corresponding to specific distributions):

- uniform noise:  $R = a + (b - a) * U(0, 1)$ .
- Gaussian noise:  $R = a + b * N(0, 1)$ .
- Salt and pepper noise:  $R : \begin{cases} 0 \leq U(0, 1) \leq a & \mapsto 0 \\ a < U(0, 1) \leq b & \mapsto 0.5 \\ b < U(0, 1) \leq 1 & \mapsto 1 \end{cases}$
- Exponential noise :  $R = -\frac{1}{a} * \ln(1 - U(0, 1))$



Generate sample images with the four kinds of random noise and visualize their histograms with the built-in functions. Pay attention to the intensity range of the resulting images when calculating the histograms



The MATLAB® functions `rand` and `randn` are used to generate random numbers with uniform and normal laws, respectively. The `imhist` function displays the image histogram. The `imadjust` functions can be useful in order to adjust the values of the image and get a correct display.

## 5.2

# Noise estimation

The objective is to evaluate the characteristics of the noise in a damaged/noisy image (in a synthetic manner in this tutorial).



1. Visualize the histogram of a Region Of Interest (ROI) of the image of Fig.5.1. The ROI should be extracted from a uniform (intensity) region.
2. Add an exponential noise to the original image and visualize the histogram of the selected ROI.
3. Add a Gaussian noise to the original image and visualize the histogram of the selected ROI.



The function `imnoise` adds a specific noise on an image. The function `roipoly` specifies a polygonal Region of Interest.

## 5.3

# Image restoration by spatial filtering



1. Add a salt-and-pepper noise to the image of Fig.5.1.
2. Test the 'min', 'max', 'mean' and 'median' image filters.



Use the function `imnoise` to add noise to an image. See `imfilter`, `ordfilt2` and `medfilt2`.

The median filter is efficient in the case of salt-and-pepper noise. However, it replaces every pixel value (first problem) by the median value determined at a given

scale (second problem). In order to avoid these two problems, Gonzalez and Woods proposed an algorithm [?].

**Data:** Input (noisy) image  $I$   
**Data:** Maximal scale  $S_{max}$   
**Result:** Filtered image  $F$

**Main Function**  $amf(I, S_{max})$ :

```

forall pixels  $(i, j)$  do
     $S \leftarrow 1$ 
    while  $\text{isMedImpulseNoise}(I, i, j, S)$  AND  $S \leq S_{max}$  do
         $S \leftarrow S + 1$ 
         $med \leftarrow \text{Med}(I, i, j, S)$ 
    end
    if  $I(i, j) = \text{Min}(I, i, j, S)$  OR  $I(i, j) = \text{Max}(I, i, j, S)$  OR
         $S = S_{max}$  then
             $F(i, j) \leftarrow \text{Med}(I, i, j, S)$ 
        else
             $F(i, j) \leftarrow I(i, j)$ 
        end
    end

```

**Algorithm 1:** Adaptive median filter [?]. The main function takes two arguments: the image  $I$  to be filtered and the maximal size of neighborhood  $S_{max}$ . For each pixel  $(i, j)$ , the good scale is the scale where the median value is different from the min and the max (median value is thus not an impulse noise). Then, if the pixel value is an impulse noise or the maximal scale has been reached, it must be filtered. Otherwise, it is kept untouched. Min, Max and Med functions compute the minimum, maximum and the median value in a neighborhood of size  $S$  centered at a pixel  $(i, j)$  in image  $I$ .

**Function**  $\text{isMedImpulseNoise}(I, i, j, S)$ :

```

 $med \leftarrow \text{Med}(I, i, j, S)$ 
if  $med = \text{Max}(I, i, j, S)$  OR  $med = \text{Min}(I, i, j, S)$  then
    return True
else
    return False
end

```

**Algorithm 1:** End.



Implement the adaptive median filter from the following algorithm (Alg.1) based on two steps (the operator acts on an operational window of size  $k \times k$  where different statistics are calculated: median, min or max).



## 5.4. Matlab correction



### 5.4.1 Generation of random noises

In order to convert the images into 8bits unsigned variables, the following function can be used:



```
function A=hist_stretch (B)
2 % histogram stretching .
% ensure the range is [0; 255]
4 A = B - min(B(:));
A = 255 * A / max(A(:));
6 A = uint8(A);
```

In order to display the results in Figs.5.2 and 5.3, a size  $S = 32$  will be used to generate the noisy images.

#### Uniform noise

The MATLAB<sup>®</sup> `rand` function generates values between 0 and 1 with uniform distribution.



```
S=32; a=0; b=255;
2 R1=a+(b-a)*rand(S);
```

#### Gaussian noise

The MATLAB<sup>®</sup> `randn` function generates values with normal centered distribution.



```
a=0; b=1;
2 R2=a+b*randn(S);
R2= hist_stretch (R2);
```

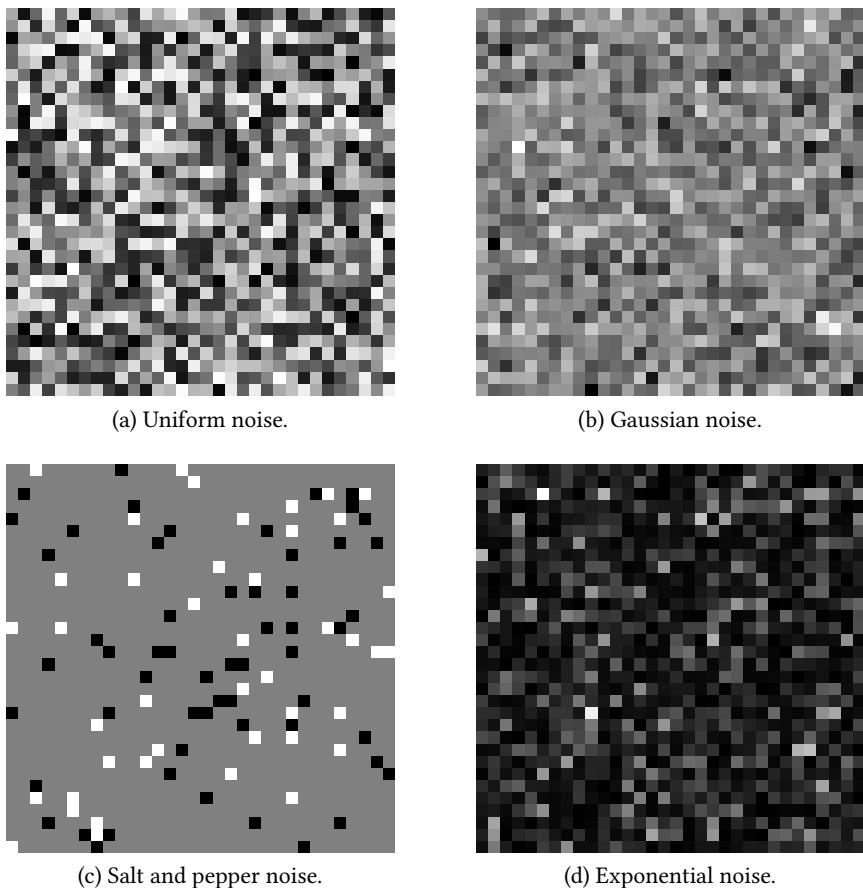


Figure 5.2: Resulting noise images.

### Salt and pepper noise



```

1 a=0.05;b=0.05;
R3=0.5*ones(S);
3 X=rand(S);
R3(X<=a)=0;
5 R3(X>a & X<=a+b)=1;
R3=hist_stretch (R3);

```

### Exponential noise



```

a=1;
2 R4=-1/a*log(1-rand(S));
R4=hist_stretch (R4);

```

## 5.4.2 Noise estimation

In order to estimate the noise, a ROI of visually constant gray level is chosen, and its histogram is displayed. This is simulated by the following code, the result is displayed in Fig.5.4:



```

1 A=imread('jambe.tif');
A=double(A);
3 A=A/255;
c=[160 200 200 160];
5 r=[200 200 240 240];
C=roipoly(A,c,r);
7 imhist(A(C));

9 % exponential noise
[m,n]=size(A);
11 expnoise = 1/0.5* log(1-rand(m,n));
B=A-expnoise/max(abs(expnoise(:)));B= hist_stretch (B);
13 imwrite(B, 'im_exp.png');
figure ;
15 subplot (1,2,1) ;viewImage(B); title ('Image with exponential noise');
subplot (1,2,2) ;imhist(B(C)); title ('histogram in ROI');
17
% Gaussian noise
19 B=imnoise(A,'gaussian' ,0,0.004) ;

```

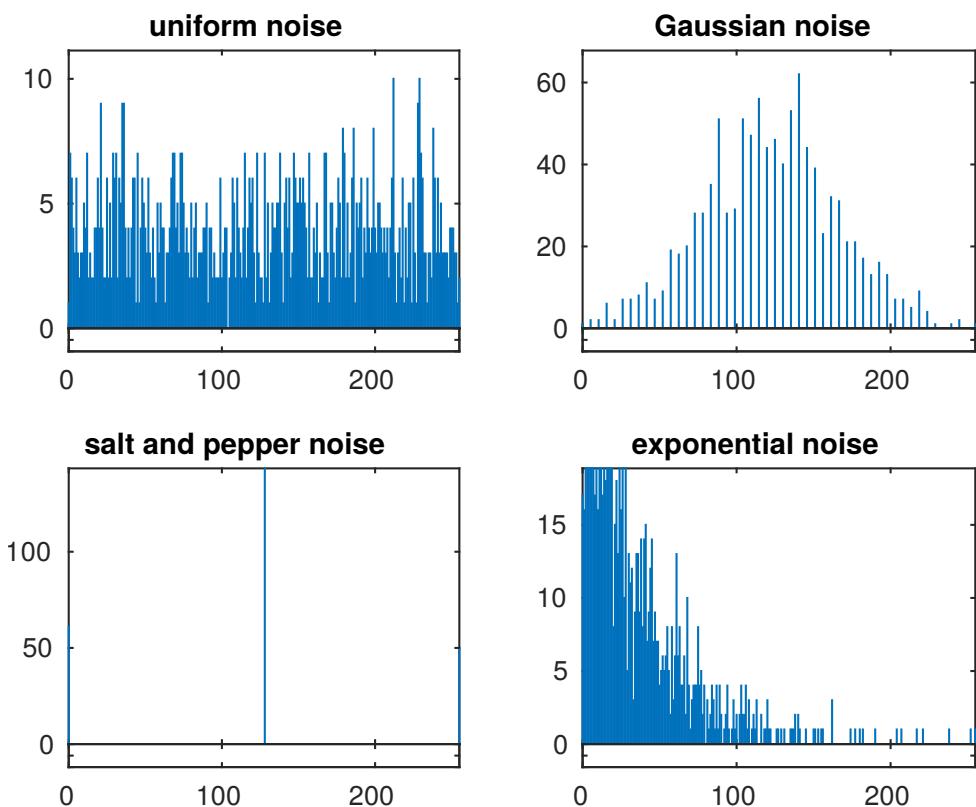
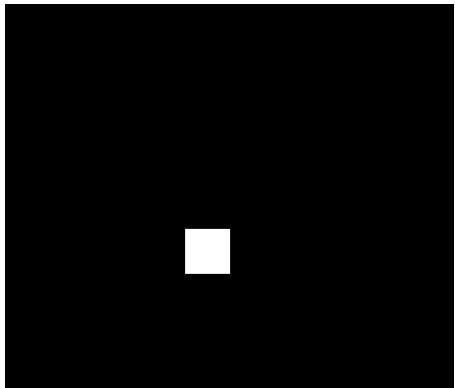


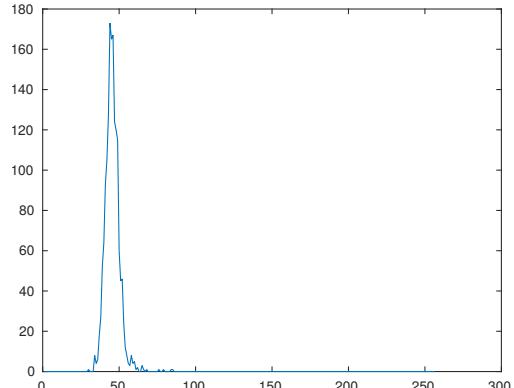
Figure 5.3: Histograms of the noise images generated with  $32 \times 32$  pixels.



```
imwrite(B, 'im_gauss.png');
21 figure;
subplot (1,2,1) ;viewImage(B); title ('image with Gaussian noise');
23 subplot (1,2,2) ;imhist(B(C)); title ('histogramin ROI');
```



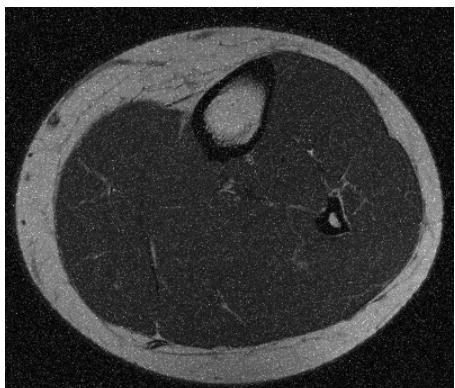
(a) ROI.



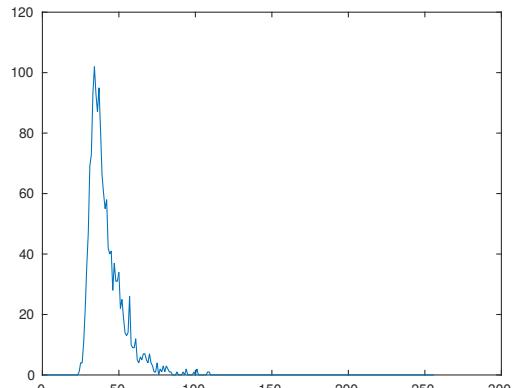
(b) Histogram.

Figure 5.4: Histogram of the Region of Interest.

In the case of exponential and Gaussian noise added to the image, The histograms are displayed in Figs.5.5 and 5.6.



(a) Addition of exponential noise to the original image of the leg.



(b) Histogram of the ROI.

Figure 5.5: Exponential noise.

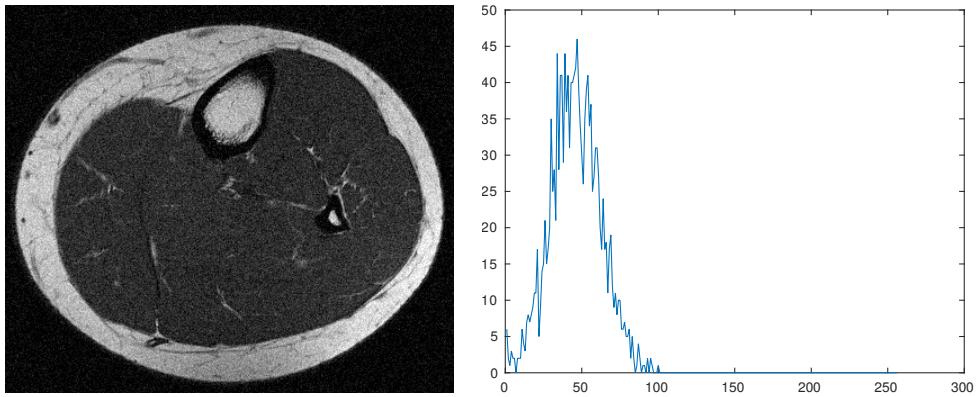


Figure 5.6: Gaussian noise.

### 5.4.3 Image restoration by spatial filtering

The following code is used to filter the images. The results are displayed in Fig.5.7.



```

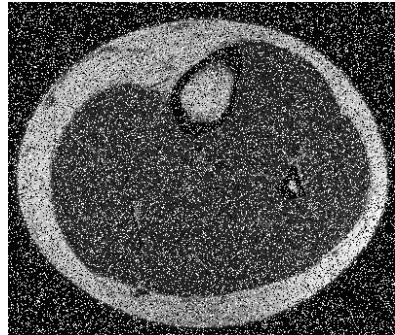
1 A=imread('jambe.tif');
A=double(A);
3 A=A/255;
[m,n]=size(A);
5 % add salt and pepper noise
B=imnoise(A,'salt & pepper', 0.25);
7 % filtering
% mean
9 w=fspecial('average',5);
B1=imfilter(B,w);
11 % max
B2=ordfilt2(B,9,ones(3,3));
13 % min
B3=ordfilt2(B,1,ones(3,3));
15 % median
B4=medfilt2(B,[7,7]);

```

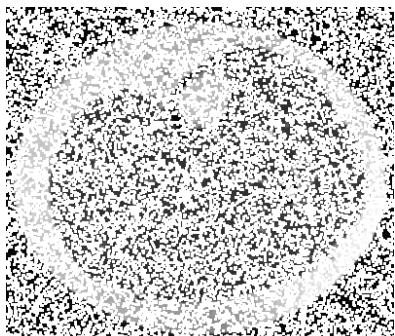
What can be noticed is that min and max filters are unable to restore the image (opening and closing filters, from the mathematical morphology, could be a solution to explore). The mean filter is a better solution, but an average value is highly modified by an impulse noise. The median filter is the optimal solution in order to suppress the noise, but fine details are lost. An interesting solution is to apply an adaptive median filter.



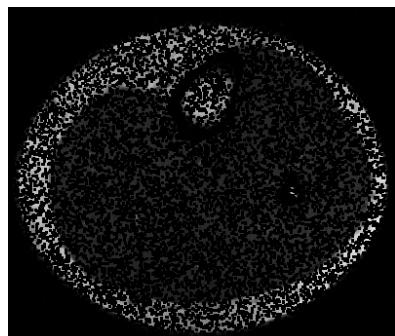
(a) Original image.



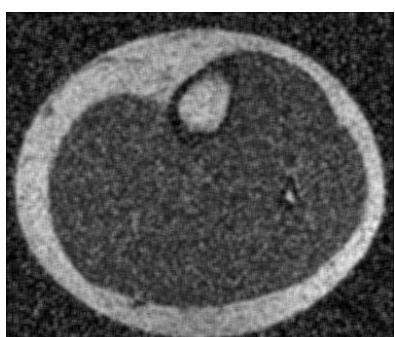
(b) Noisy image (salt and pepper).



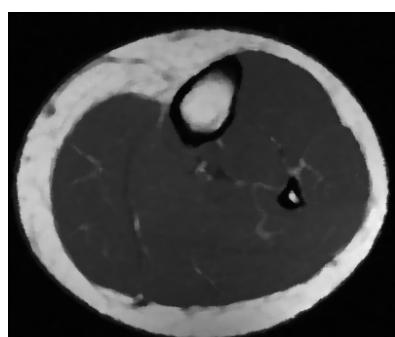
(c) Maximum filter.



(d) Minimum filter.



(e) Mean filter.



(f) Median filter.

Figure 5.7: Different filters applied to the noisy image. The median filter is particularly adapted in the case of salt and pepper noise (impulse noise), but still destroy the structures observed in the images.

## Adaptive median filter

The following code is a simple implementation of the algorithm previously presented. It has not been optimized in order to have a simple presentation. A more sophisticated version can be found in [?]. The results are illustrated in Fig.5.8 for  $S_{max} = 7$ .



```

function f = amf(I, Smax)
% adaptive median filter
% I: original image
% Smax: size maxi of neighborhood

f = I;

sizes = 1:Smax;
zmin = zeros([ size(I) length(sizes) ]);
zmax = zeros([ size(I) length(sizes) ]);
zmed = zeros([ size(I) length(sizes) ]);

for k=1:length(sizes),
    zmin (:,:, k) = ordfilt2 (I, 1, ones(sizes(k)), 'symmetric');
    zmax (:,:, k) = ordfilt2 (I, sizes(k)^2, ones(sizes(k)), 'symmetric');
    zmed (:,:, k) = medfilt2(I, [ sizes(k) sizes(k) ], 'symmetric');
end

% determines for all scales at the same time if zmed is an impulse noise.
% this enables the choice of the scale.
isMedImpulse = (zmin==zmed) | (zmax==zmed);

for i=1: size(I,1)
    for j=1: size(I,2)

        % finds the right scale
        % determines k (neighborhood size) where the median value is not an
        % impulse noise
        k=1;
        while isMedImpulse(i,j,k) && k<length(sizes)
            k = k+1;
        end

        % if the value of the pixel I(i,j) is an impulse noise, it is
        % replaced by the median value at scale k, if not, it is kept
        % untouched (already set)
        if I(i,j)==zmin(i,j,k) || I(i,j)==zmax(i,j,k) || k == length(sizes)
            f(i,j) = zmed(i,j,k);
        end

    end
end

```

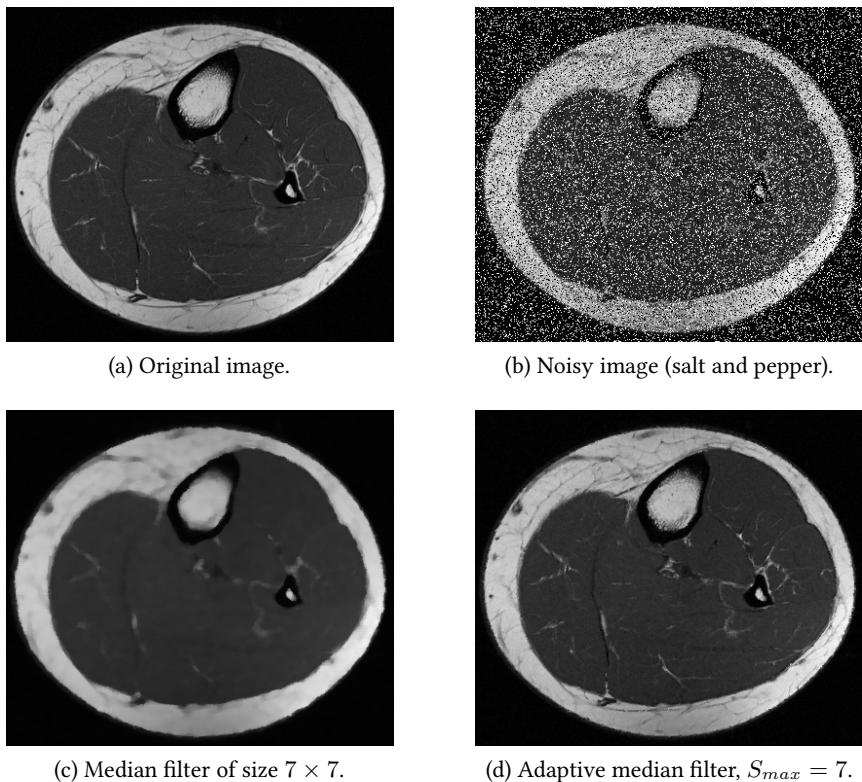


Figure 5.8: Illustration of the adaptive median filter.





## 6

# Image Restoration: deconvolution

This tutorial aims to study and test different image restoration methods for blurred and noisy images. Although the different software tools contain functions for each filter the objective is to exhaustively code these methods to fully understand their principles. The reader can refer to [?] for more informations on the algorithms.

Some of the images are based on observations made with the NASA/ESA Hubble Space Telescope, and obtained from the Hubble Legacy Archive, which is a collaboration between the Space Telescope Science Institute (STScI/NASA), the Space Telescope European Coordinating Facility (ST-ECF/ESA) and the Canadian Astronomy Data Centre (CADC/NRC/CSA). You may access to these resources at <https://hla.stsci.edu/>

The different processes will be applied on the following images.

## 6.1

# Damage modeling

A damage process can be modeled by both a damage function  $D$  and an additive noise  $n$ , acting on an input image  $f$  for producing the damaged image  $g$ .

$$g = D(f) + n \quad (6.1)$$

Knowing  $g$ , the objective of restoration is to get an estimate  $\hat{f}$  (the restored image) of the original image  $f$ . If  $D$  is a linear process, spatially invariant, then the equation can be simplified as:

$$g = h * f + n \quad (6.2)$$

where  $h$  is the spatial representation of the damage function (called the Point Spread Function - PSF), and  $*$  denotes the convolution operation. In general, the more knowledge you have about the function  $H$  and the noise  $n$ , the closer  $\hat{f}$  is to  $f$ .

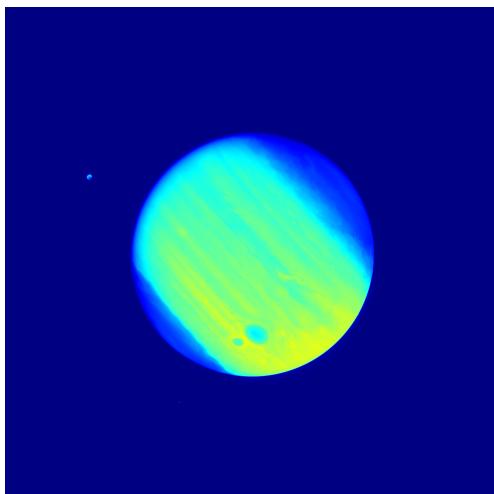
This equation can be written in the frequency domain:

$$G = H \cdot F + N \quad (6.3)$$

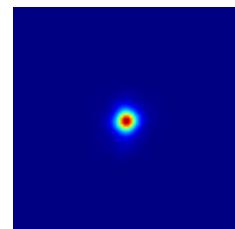
where the letters in uppercase are the Fourier transforms of the corresponding terms in the eq. 6.2.

Figure 6.1: Images that can be used for testing the different restoration algorithms. Intensity levels are represented in false colors.

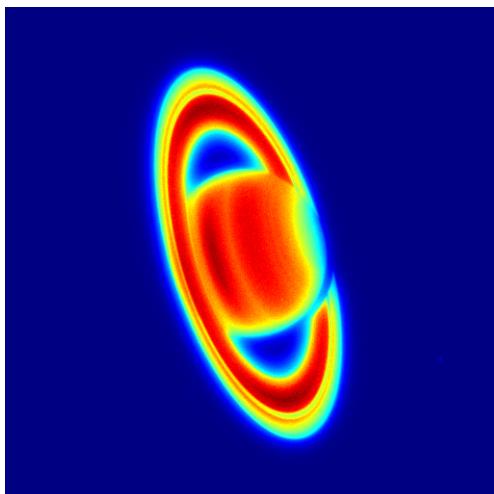
(a) Jupiter, from Hubble Space Telescope,  
ads/Sa.HST#ic3g01qlq.



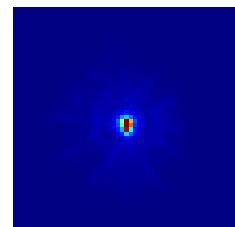
(b) PSF for Jupiter im-  
age



(c) Saturn, HST.

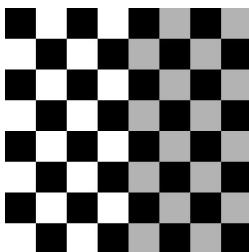


(d) PSF for Saturn image



- Generate the 'chessboard' image.
- Generate a PSF corresponding to a blur (motion or isotropic).

Figure 6.2: Chessboard image.



- Create a damaged image (a circular option is used, because the FFT implies that functions are periodical) and add a Gaussian noise.
- Visualize the different images.



MATLAB® has a checkerboard function for the 'chessboard' image. The `fspecial` function generates a convolution matrix.

## 6.2

## Simple case: no noise

The objective is now to restore the damaged image.

For the first steps, no noise will be added. The degradation functions is  $g = h * f$ . Knowing  $H$  (Fourier Transform of the psf  $h$ ) and ignoring the noise in the damage model, a simple estimate of the initial image can be done by the inverse filtering:

$$f = FT^{-1} \left( \frac{G}{H} \right) \quad (6.4)$$

where  $FT^{-1}$  denotes the inverse Fourier transform.



- Try the inverse filter in the Fourier space. Why is this filter not working?
- Try to prevent division by 0, with  $f = FT^{-1} \left( \frac{G}{H+\alpha} \right)$ ,  $\alpha$  being a small constant value (e.g.  $\alpha = 0.1$ ).

## 6.3 Wiener filter

The Wiener filter is an optimal filter that minimizes the following error:

$$e^2 = E\{(f - \hat{f})^2\}$$

where  $E$  denotes the expected (mean) value,  $f$  is the original image and  $\hat{f}$  is the restored image.

The solution to this expression, within the frequency domain, is given by:

$$\hat{F} = \underbrace{\left( \frac{1}{H} \frac{|H|^2}{|H|^2 + R} \right)}_{H_w} G$$

The value of  $R$  can be arbitrarily fixed. Another way is to define  $R = S_n/S_f$ , where  $S_n$  and  $S_f$  denote the power spectrum of the noise  $n$  and the original image  $f$ , i.e.  $|N|^2$  and  $|F|^2$  (matrices), respectively. This ratio can be defined globally (as the constant) or locally (i.e. it is computed for each pixel). If the ratio  $S_n/S_f$  (function) is non null, it can be passed to the Wiener filtering process.

### 6.3.1 Simple case: no noise

In the noiseless case, the Wiener filter reduces to the inverse filter:

$$H_w = \begin{cases} \frac{1}{H(u, v)} & \text{if } |H(u, v)| \neq 0 \\ 0 & \text{otherwise} \end{cases}$$



Try a Wiener filter for the noiseless case.

### 6.3.2 Noisy images

Generally, the ratio  $S_n/S_f$  used in the Wiener filter is replaced by a constant value equal to the ratio of the mean power spectrum:

$$R = \frac{\frac{1}{PQ} \cdot \sum_u \sum_v S_n(u, v)}{\frac{1}{PQ} \cdot \sum_u \sum_v S_f(u, v)} \quad (6.5)$$

where  $PQ$  denotes the matrix size (the number of elements).



see numel function.



1. Calculate the constant ratio  $R$  and code the Wiener filter .
2. Test the different algorithms on the image 'brain' damaged by a blur and an additive noise.

## 6.4

# Iterative filters for noisy images

### 6.4.1 Van Cittert iterative filter

The VanCittert iterative filter is based on the following iterative relation :

$$f_{k+1} = f_k + \beta(g - h * f_k)$$

with  $f_0 = g$ , the damaged image.



Code and test on different cases a function that performs the VanCittert restoration filter.

It should have the following prototype:



```
function [ fr ] = vca( g, psf, n_iter )
% Van Cittert iteration algorithm for deconvolution
%
% g: degraded image
% psf: point spread function
% n_iter : number of iterations
```



```
def vca(g, psf, n_iter, beta=.01):
    """
    Van Cittert iterative filter
    g: noisy image
    psf: point spread function
    n_iter : number of iterations
    beta: Jansson parameter
    """
```

### 6.4.2 Lucy-Richardson filtering

We are now going to use the nonlinear restoration filtering of Lucy-Richardson that is based on a maximum likelihood formulation in which the image is modeled by Poisson statistics. This optimization leads to the solution if the following iterative equation converges:

$$f_{k+1}(x, y) = f_k(x, y) \cdot \left( h(-x, -y) * \frac{g(x, y)}{h(x, y) * f_k(x, y)} \right) \quad (6.6)$$



Code and test your own function for the Lucy-Richardson filtering process. It should have the same prototype as the VanCittert function. Remember that  $\cdot$  is the classical multiplication (point to point) and that  $*$  is the convolution operation. You then have the choice to make a direct call to the convolution function, or to perform this operation in the Fourier space.

## 6.5

### Blind deconvolution: restoration with no a priori

If the PSF is unknown, one of the main difficulties in image restoration is to get an accurate estimate of this PSF in order to use it in the deconvolution algorithms. These methods are called blind deconvolution methods. Generally, the PSF is estimated in an iterative way from an initial PSF.



1. Generate a damaged image of 'chessboard' with a Gaussian blur and an additive Gaussian noise.
2. Restore from a blind deconvolution the damaged image and compare the results with the previous ones.



Use the function `deconvblind`.



Informations

Use the function `skimage.restoration.unsupervised_wiener` for example.

## 6.6

# Astronomy images

A classical file format used in astronomy is the FITS format.

The **FITS** files can be read using the function `fitsread`. For example, the following commands will read an image and its PSF.



```
% 1st example
2 saturn = fitsread ('saturn.fits');
psf = fitsread ('saturn_psf.fits');
4
% 2nd example
6 jupiter = fitsread ('ic3g01qlq_flt.fits', 'image', 1);
psf = fitsread ('PSFSTD_WFC3UV_F275W.fits');
8 jupiter_psf = p (:,:1); % select 1st psf only
```



## 6.7. Matlab correction



### 6.7.1 Damage modeling

The damaged image is generated using the following method, illustrated in Fig. 6.3, for a motion perturbation. Notice the 'circular' option in the convolution, which acts as if the psf would be periodic (this is the case in the Fourier space).



```
% generates a noisy image, with motion blur
2 f = checkerboard(8);
psf = fspecial ('motion',7,45) ;
4 gb = imfilter (f,psf, 'circular ');
noise = imnoise(zeros( size(f)), 'gaussian' ,0,0.001) ;
6 g = gb + noise;
```

The noise can also be generated by the following command:



```
sigma = .1;
2 noise = randn(size(f)) * sigma;
```

If the psf used is gaussian, the command will be:



```
psf = fspecial ('gaussian', size(f), 1);
```

### 6.7.2 Inverse filtering with no noise

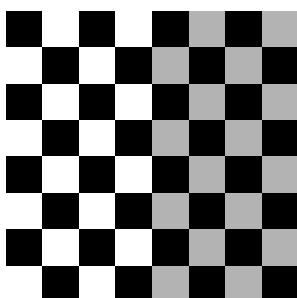


Be careful to use `psf2otf` instead of `fft2`. This is basically the same purpose, but the latter lacks centering when performing the array padding.

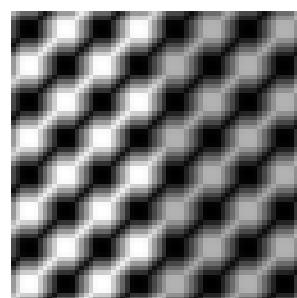
The inverse filter of the damaged image is performed using the following MATLAB® command. The result is illustrated in Fig. 6.4.



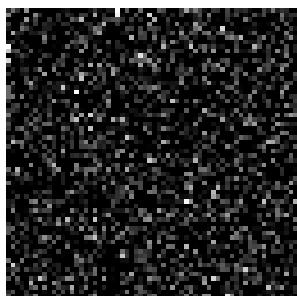
```
% Fourier Transform of the psf
2 H = psf2otf(psf, size(f));
```



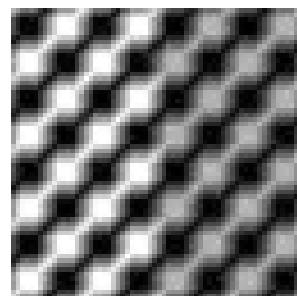
(a) Original image.



(b) Movement blur.



(c) Gaussian noise.



(d) Noisy image.

Figure 6.3: Simulation of damaged images.



```

G = fft2(g);
4
% pseudo inverse: ensures there is no division by 0
6 alpha = 0.01;
F = G./(H + alpha);
8 fr = ifft2(F);

```

### 6.7.3 Wiener filter

Simple case: no noise



```

% Fourier Transform of the psf
2 H = psf2otf(psf, size(f));
G = fft2(g);
4
% Wiener filter when no noise
6 % Eliminates zeros values in H

```

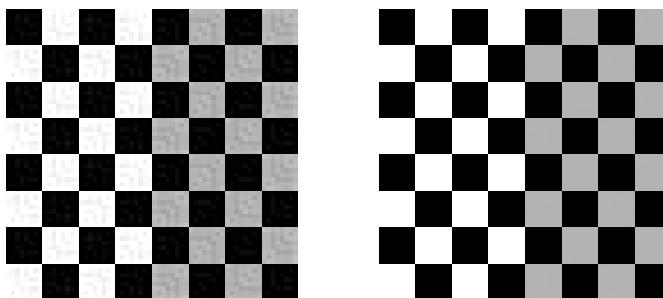


Figure 6.4: Motion blur, 45 degrees, with no noise.



```
H(H==0) = Inf;
8 F = G./H;
fw = ifft2 (F);
```

The result is illustrated in Fig.6.4

### Noisy images

To evaluate the ratio  $R$ , the following commands are performed:



```
1 SpecPuissNoise=abs( fft2 ( noise )) .^2;
PuissMoyNoise=sum(SpecPuissNoise(:))/numel(noise);
3 SpecPuissImageOrig=abs(fft2 (f)) .^2;
PuissMoyImageOrig=sum(SpecPuissImageOrig(:))/numel(f);
5 ratio =PuissMoyNoise/PuissMoyImageOrig;
```

And then, the Wiener filter is applied either as a MATLAB<sup>®</sup> function or as your own function. The result of this second version is presented in Fig. 6.5.



```
1 % \ matlabregistered {} function
fr2=deconvwnr(g,psf, ratio );
```

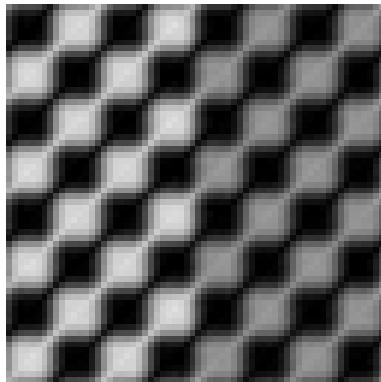


Figure 6.5: Wiener filter with approximated ratio  $R$ .



```
% create a damaged image with some noise
2 f=checkerboard(8);
psf = fspecial ('gaussian', size (f), 2);
4 % noise
sigma=.01;
6 N = randn(size (f)) * sigma;
g = imfilter (f, psf, 'circular') + N;
8 g = max(0,g);
g = min(1,g);
10
% performs the restoration
12 H = psf2otf(psf, size(f)); % Fourier Transform of the psf
Hw = conj(H)./(abs(H).^2+PuissMoyNoise/PuissMoyImageOrig);
14 fr = ifft2 ( Hw.* fft2 (g));
```

### 6.7.4 Iterative filters for noisy images

#### Van Cittert iterative filter

The VanCittert algorithm is maybe the simplest iterative method. It is really sensitive to noise, as illustrated in Fig. 6.6.

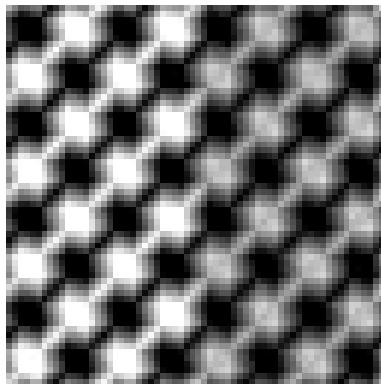


Figure 6.6: VanCittert iterative restoration algorithm with 10 iterations, applied on an image damaged with motion blur and gaussian white noise. This algorithm is really sensitive to noise.



```
function [ fr ] = vca( g, psf, n_iter )
% Van Cittert iteration algorithm for deconvolution
%
% g: degraded image
% psf: point spread function
% n_iter : number of iterations

% Fourier Transform of the psf
H = psf2otf(psf, size(g));
%
% initialization
fr=g;

beta = .1; %Jansson parameter
for iter = 1:n_iter
    fr = fr + beta*(g - ifft2(H .* fft2(fr)));
end
end
```

### Lucy-Richardson filtering

The Lucy-Richardson deconvolution filter is one of the most employed filter. The convolution operations can be either performed in the Fourier (frequency) domain, or in the spatial domain. For performance reasons, it is usually better to work in the Fourier domain, where the convolution is transform into a simple product. The results are presented in Fig. 6.7



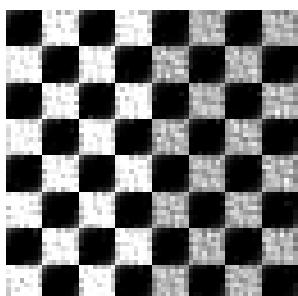
```

1 function fr = rla(g, psf, n_iter)
% Richardson–Lucy Algorithm for deconvolution
3 %
% g: image to restore. initial value of fr is g
5 % psf: point spread function
% n_iter : number of iterations
7 %
% Optical Transfer Function
9 H = psf2otf(psf, size(g));

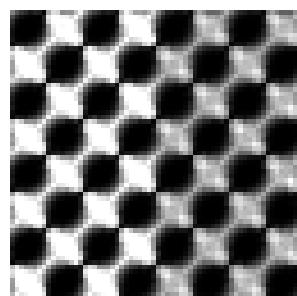
11 % initial value
fr = g;
13 %
% iterations
15 for iter =0: n_iter
    % estimated blurred image
17     yk = ( ifft2 (H.* fft2 (fr)));

19     M = ifft2 (conj(H) .* fft2 (g./ yk));
    fr = max(0,fr .* M);
21 end

```



(a) Algorithm with 100 iterations.



(b) Algorithm with 10 iterations.

Figure 6.7: Lucy-Richardson algorithm applied on the chessboard image with motion blur and gaussian white noise.

### 6.7.5 Blind deconvolution

The first step is to generate a degraded image, as before.

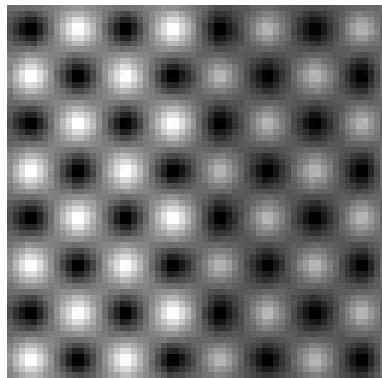


```
1 f=checkerboard(8);
2 S = 10; % size of PSF
3 psf= fspecial ('gaussian',S,10);
4 sd=0.01; % std deviation of the generated noise
5
6 % add some gaussian noise with 0 mean and variance sd^2
7 g=imnoise( imfilter (f,psf, 'circular'),'gaussian', 0, sd^2);
```

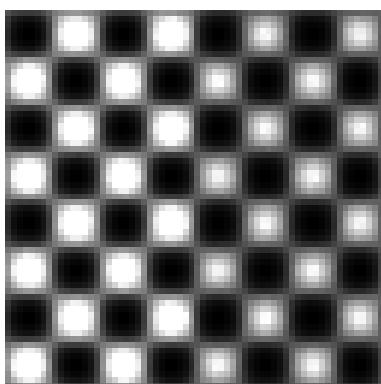
The dampar variable is a damping parameter to minimize the noise (see MATLAB® help).



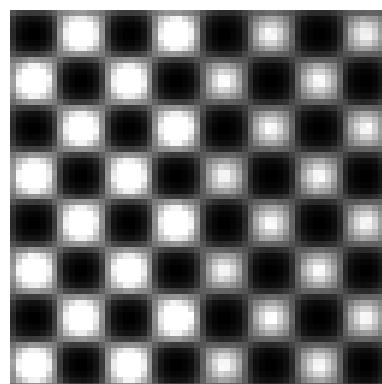
```
1 % damping parameter
2 dampar=10*sd;
3
4 % create initial psf
5 initpsf =ones(S);
6
7 % apply blind deconvolution for nb_iterations
8 nb_iterations = 5;
9 [ fr_blind , psfe]=deconvblind(g, initpsf , nb_iterations , dampar);
```



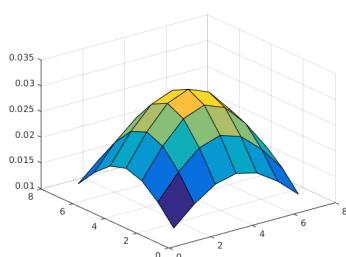
(a) Original image.



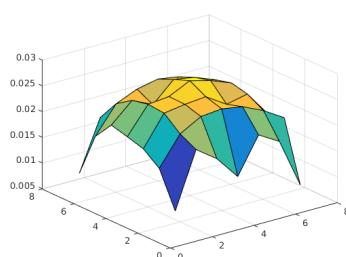
(b) Blind deconvolution for 100 iterations.



(c) Blind deconvolution for 5 iterations.

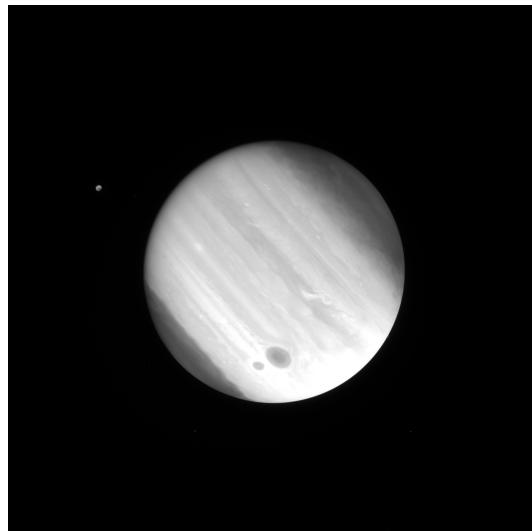


(d) PSF used for image degradation.

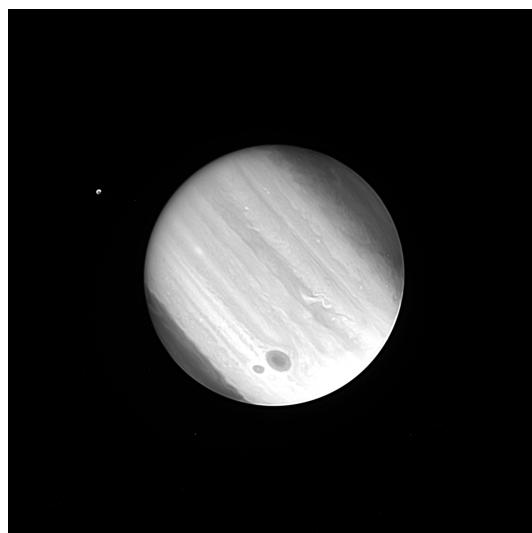


(e) PSF estimated after blind deconvolution.

Figure 6.8: Blind deconvolution

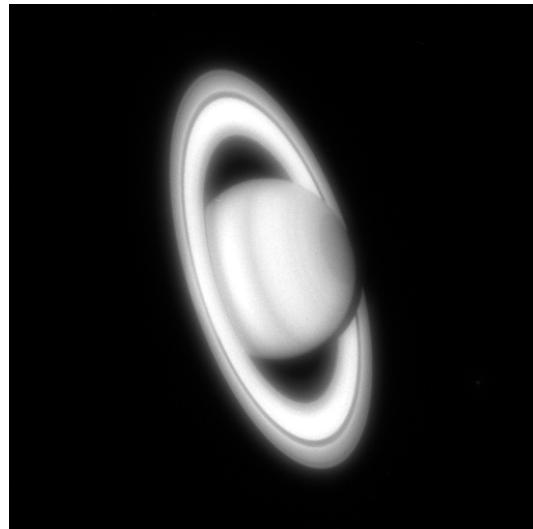


(a) Original image of Jupiter.

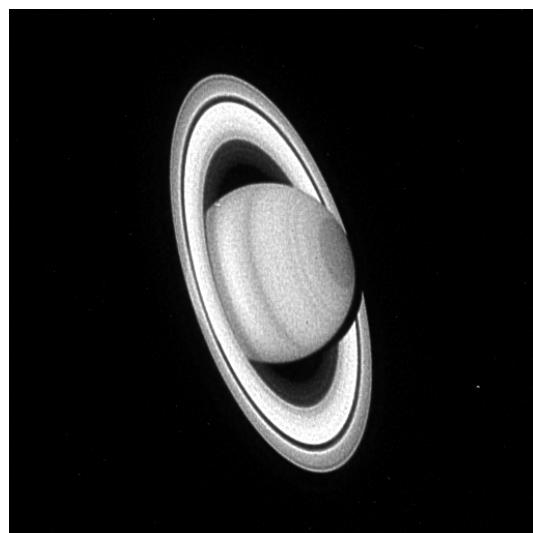


(b) Jupiter image after Lucy-Richardson algorithm with 10 iterations.

Figure 6.9: Jupiter, from Hubble Space Telescope, ads/Sa.HST#ic3g01qlq.



(a) Original image of Saturn.



(b) Saturn image after Lucy-Richardson algorithm with 100 iterations.

Figure 6.10: Saturn, from Hubble Space Telescope.



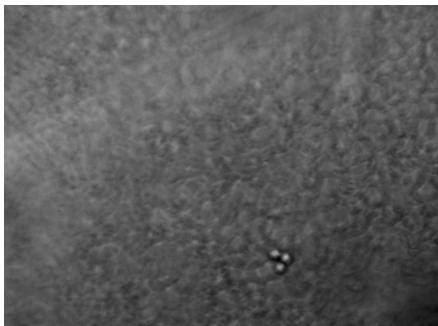
## 7

# Shape From Focus

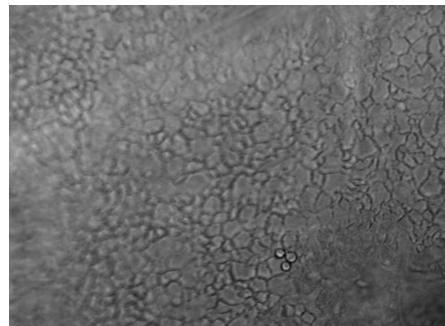
This tutorial introduces the basic shape from focus concepts. The objective is to reconstruct a focused image from a serie (generally a stack) of images with inhomogeneous focus (see Fig. 7.1).

Figure 7.1: Different images of the stack, from a corneal endothelium in optical microscopy (with 3D microscope).

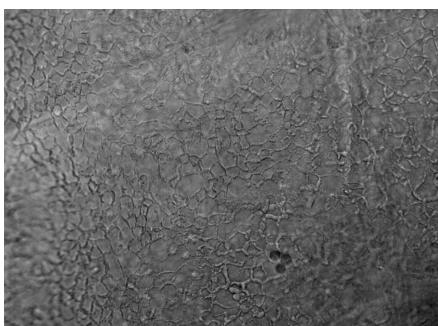
(a) Layer 1



(b) Layer 9



(c) Layer 18



(d) Layer 24



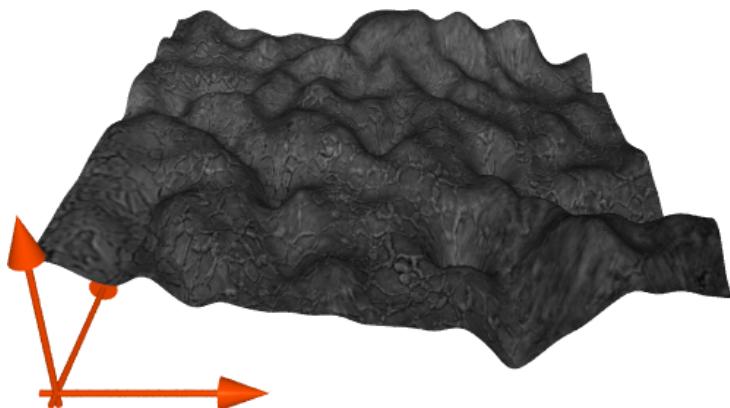
## 7.1

### Introduction to classical methods

An optical system has a limited depth of field. When observing non plane surfaces with a microscope, some parts of the observation may be blurred as well as some others may be correctly focused.

To overcome this problem and reconstruct an all-focused image as well as a surface (see Fig. 7.2), an algorithm will look at every pixel of the images in the stack and select the most focused one, by the way of a focus measure. This tutorial proposes to test some classical focus measures. You can have a look at [?] and [?] to see real applications.

Figure 7.2: Reconstruction of the surface and the texture.



Practically, TIF files can handle stacks of images. Here is a way to open such a file:



```
1 from skimage import io
2
3 I = io.imread('volume.tif');
I = I.astype('float');
```



```
% load a stack of images in file
2 info = imfinfo( stackfile );
num_images = numel(info);
4 % store the images into stack
stack=zeros(info (1).Height, info (1).Width, num_images);
6 for k = 1:num_images
    stack (:,:, k)=imread( stackfile , k);
8     % you can now do something with each image
end
```

In the following, each of the proposed methods will compute a focus measure layer by layer, and will maximize this measure over the stack to find the most focused layer.

### 7.1.1 Sum of Modified Laplacian

One of the first methods had been proposed by [?]. It is based on the second derivatives, specifically the Laplacian operator:

$$\Delta^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

The problem with this operator is that the second derivatives in the x and y dimensions can have opposite signs. One way to overcome this problem is to introduce the modified Laplacian as follows:

$$\Delta_M^2 I = \left| \frac{\partial^2 I}{\partial x^2} \right| + \left| \frac{\partial^2 I}{\partial y^2} \right|$$

The discrete approximation of the modified Laplacian is computed as:

$$ML(x, y) = |2I(x, y) - I(x-1, y) - I(x+1, y)| + |2I(x, y) - I(x, y-1) - I(x, y+1)| \quad (7.1)$$

Then, the focus measure based on the modified Laplacian is:

$$F_{ML} = \sum_{(x,y) \in \omega} ML(x, y)$$

### 7.1.2 Variance

The measure of focus based on the variance is today the mainly used method [?, ?]. It is based on the computation of the variance in a window  $\omega$ , with  $N = \#\omega$  being the size of the window:

$$F_v = \sum_{\omega} \left( I - \underbrace{\frac{1}{\#\omega} \sum_{\omega} I}_{\text{Mean of } I} \right)^2$$

### 7.1.3 Tenengrad

In [?], we can find the definition of the tenengrad [?]. Let  $S(x, y)$  be the norm of the Sobel gradient of image  $I$ .

$$F_t(I) = \sum_{\omega} S^2$$

Notice that the original definition requires a threshold value that requires heuristic choices, which is out of the topic of this tutorial.

### 7.1.4 Variance of Tenengrad

We define the variance of Tenengrad measure of focus by:

$$F_{vt}(I) = F_v(S)$$

## 7.2 Texture and surface reconstruction



A set of images (Vickers indentation test [?], and a human corneal endothelium [?]) are proposed. For all the detailed methods:

- reconstruct the surface and the texture of the image (an image focused on its all field of view).

### 7.2.1 Open question



Several methods have been implemented in order to perform the 3D surface/-texture reconstruction. Propose a numerical measure that could compare the results and measure the efficiency of these methods? What is the best method?



## 7.3. Matlab correction



### 7.3.1 Main function

This function is used as the main function to perform shape-from-focus reconstruction. The parameter method is the name of the function. Notice that the variable stack is of type double to handle the cornea stack (which is a 16 bits image).



```

function [Z, T]=SFF( file , N, method)
% Shape From Focus by SML method
% Z: heights / altitudes ( indices )
% T: texture
%
% file : filename ( tif stack -file )
% N: parameter of 'method' function
% method: method to use (SML, variance, etc .)
info = imfinfo( file );
num_images = numel(info);

% create stack into memory
stackF=zeros( info (1) .Height, info (1) .Width, num_images);
%
% load stack
stack =double(stackF);
for k = 1:num_images
    stack (:,:, k)=imread( file , k);
    % ... compute SML
    stackF (:,:, k) = feval (method, double( stack (:,:, k)), N);
end

% search for maximum of function
[~, Z] = max(stackF, [], 3);
Z = uint8(Z);
T = double(zeros( size ( stack ,1) , size ( stack ,2)));
for i=1:size (T, 1)
    for j=1:size (T,2)
        T(i, j) = stack(i, j,Z(i, j));
    end
end

figure ();
subplot (1,2,1) ; imshow(Z,[]); title (' altitudes ');
subplot (1,2,2) ; imshow(T,[]); title (' textures ');
end

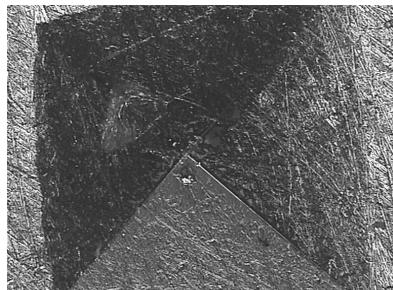
```

### 7.3.2 Sum of Modified Laplacian

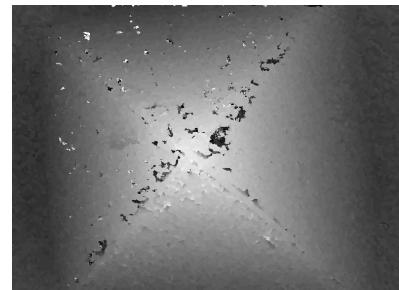
Results are illustrated in Fig.7.3.



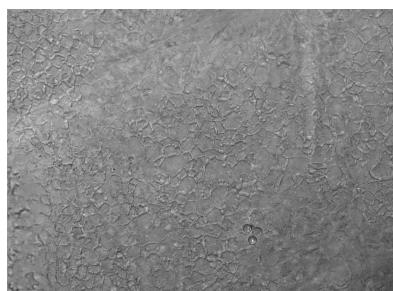
```
% main function
2 function SML=modifiedLaplacian(A, N)
    h1 = [0 0 0; -1 2 -1; 0 0 0];
4    h2 = h1';
    ML = abs(conv2(A, h1, 'same')) + abs(conv2(A, h2, 'same'));
6
    h = ones(N);
8    SML = conv2(ML, h, 'same');
end
```



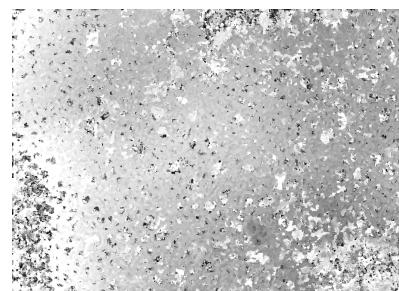
(a) Texture.



(b) Altitudes.



(c) Texture.

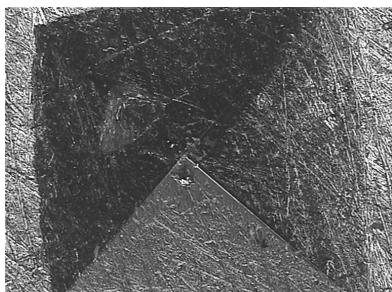


(d) Altitudes.

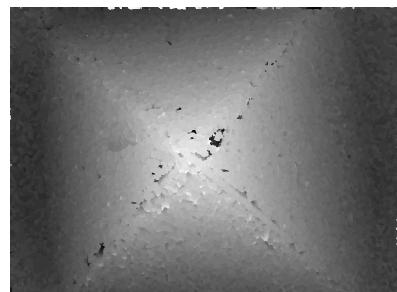
Figure 7.3: Texture and altitude reconstruction with the SML method.

### 7.3.3 Variance

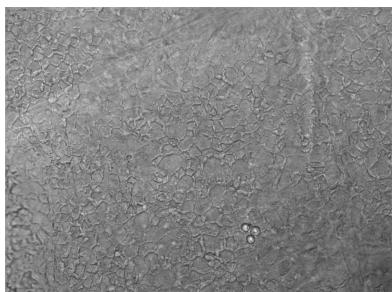
The focus measure based on the variance is a really simple method that works in most cases, see Fig.7.4.



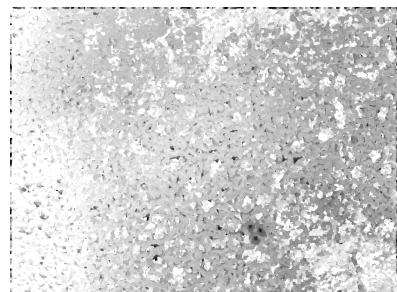
(a) Texture.



(b) Altitudes.



(c) Texture.



(d) Altitudes.

Figure 7.4: Texture and altitude reconstruction with the variance method.



```

1 function V=variance(A, N)
% A: single image
3 % N: size of the window
h = ones(N);
5
moyenne = (1/N^2) * conv2(A, h, 'same');
7 D2=(A-moyenne).^2;
% variance
9 V = conv2(D2, h, 'same');
end

```

### 7.3.4 Tenengrad

The tenengrad method is base on a Sobel filter, see Fig.7.5.



```

function T=tenengrad(A, ~)
% A: single image

```



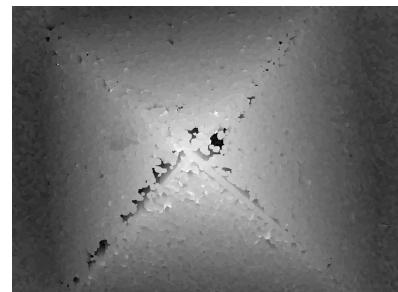
```

Sx = fspecial ('sobel');
4   Gx = imfilter (double(A), Sx, 'replicate', 'conv');
    Gy = imfilter (double(A), Sx, 'replicate', 'conv');
6   T = Gx.^2 + Gy.^2;
end

```



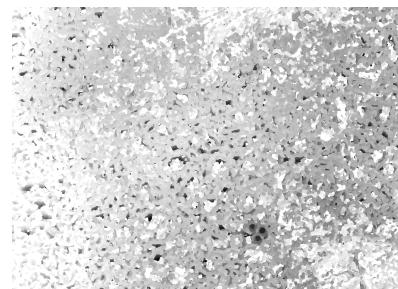
(a) Texture.



(b) Altitudes.



(c) Texture.



(d) Altitudes.

Figure 7.5: Texture and altitude reconstruction with the SML method.

### 7.3.5 Variance of Tenengrad

The variance of Tenengrad is an improvement of the Tenengrad method, see Fig.7.6.



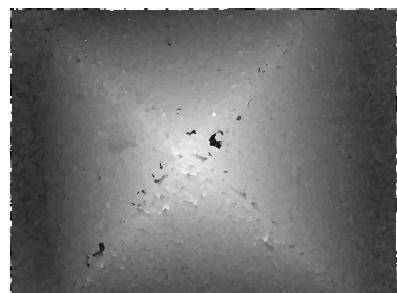
```

1 function vt=varianceTenengrad(A, N)
  h1 = [1 2 1; 0 0 0; -1 -2 -1];
3  h2 = h1';
  ML = sqrt(conv2(A, h1, 'same').^2 + conv2(A, h2, 'same').^2);
5
  vt=variance(ML, N); % variance function previously defined
7 end

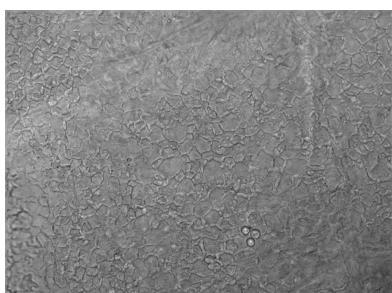
```



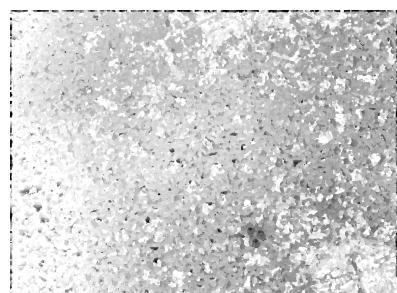
(a) Texture.



(b) Altitudes.



(c) Texture.



(d) Altitudes.

Figure 7.6: Texture and altitude reconstruction with the variance of Tenengrad method.



# \* 8 Logarithmic Image Processing (LIP)

This tutorial aims to study the LIP model, which is a vector space of gray-level images, consistent with the physical laws of Weber and Fechner as well with the visual perception laws. More particularly, the LIP dynamic expansion, used for image enhancement, will be implemented as well as the Sobel LIP filter for edge detection.

The different processes will be applied on a mammographic image.

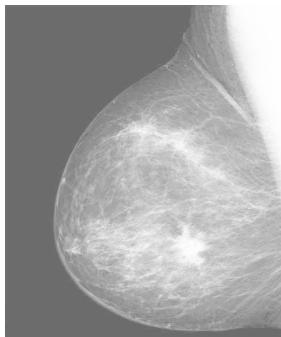


Figure 8.1: Breast

## 8.1 Introduction

The LIP model (Logarithmic Image processing) has been introduced in the mid 1980s. It defines a mathematical framework for image processing in a bounded interval. It is mathematically rigorous and physically justified. For more informations, refer to [?, ?, ?].

An image is represented by its gray-tone function  $f$ , defined on a spatial domain  $D \subset \mathbb{R}^2$ , and with values into  $[0, M]$ , where  $M > 0$ . For all gray-tone functions  $S$  defined on  $D$ , a vector space is defined by the operations of addition  $\triangle$  and

multiplication  $\triangle$ , and by extension with the operations of subtraction  $\triangle$  and negation:

$$\forall f, g \in S, f \triangle g = f + g - \frac{fg}{M} \quad (8.1)$$

$$\forall f \in S, \forall \lambda \in \mathbb{R}, \lambda \triangle f = M - M \left(1 - \frac{f}{M}\right)^\lambda \quad (8.2)$$

$$\forall f \in S, \triangle f = \frac{-Mf}{M-f} \quad (8.3)$$

$$\forall f, g \in S, f \triangle g = M \frac{f-g}{M-g} \quad (8.4)$$

The vector space  $S$  of gray-tone functions is algebraically and topologically isomorph to the classical vector space, defined by the isomorphism  $\varphi$ :

$$\forall f \in S, \varphi(f) = -M \ln \left(1 - \frac{f}{M}\right) \quad (8.5)$$

The inverse isomorphism  $\varphi^{-1}$  is then defined by:

$$f = \varphi^{-1}(\varphi(f)) = M \left(1 - \exp\left(-\frac{\varphi(f)}{M}\right)\right) \quad (8.6)$$

This fundamental isomorphism allows the definition of the following operations on gray-tone functions:

- Dot product:  $\forall f, g \in S, \langle f, g \rangle_\Delta = \int \varphi(f) \varphi(g)$
- Euclidean norm:  $\forall f \in S, \|f\|_\Delta = |\varphi(f)|_{\mathbb{R}}$ , with  $|\cdot|_{\mathbb{R}}$  is the classical absolute value.

## 8.2

## Elementary LIP operations

Classically, gray-tones are coded with 8 bits, and thus one can choose  $M = 256$ . Be careful that there might exist sampling issues, and it might be required to consider images in the LIP space with a double precision.



1. Implement the elementary LIP operations  $\triangle$ ,  $\triangle$  and  $\triangle$ .
2. Test these operators on the image 'breast'.

## 8.3

# LIP dynamic expansion

The LIP model enables to define an image transformation that expanses, in an optimal way, the overall dynamic range (gray-tones) while preserving a physical or visual sense. Let a graytone function denoted  $f$  be defined on the spatial support  $D \subset \mathbb{R}^2$ . Its upper and lower bounds are denoted  $f_u = \sup_{x \in D} f(x)$  and  $f_l = \inf_{x \in D} f(x)$ , with  $0 < f_l < f_u < M_0$ .

The dynamic range of  $f$  on  $D$  is defined by  $R(f) = f_u - f_l$ . The LIP scalar multiplication of  $f$  by a real number  $\lambda > 0$  yields to the following dynamic :

$$R(\lambda \triangleq f) = \lambda \triangleq f_u - \lambda \triangleq f_l \quad (8.7)$$

It has been shown [?] that there exists an optimal value  $\lambda_0(f)$  that maximizes the dynamic range, i.e.:

$$R(\lambda_0(f) \triangleq f) = \max_{\lambda > 0} R(\lambda \triangleq f). \quad (8.8)$$



1. Determine the explicit (analytic) expression of the parameter  $\lambda_0(f)$ .
2. Calculate the value of the parameter  $\lambda_0(f)$  for the image 'breast'.
3. Enhance the image and compare the result with the histogram equalization.



Histogram equalization in MATLAB® is performed with the `histeq` function.

## 8.4

# LIP edge detection



1. Implement the Sobel filter within the LIP framework.
2. Compare the results with the classical Sobel filter.



## 8.5. Matlab correction



### 8.5.1 Elementary operations

The most important function to code is the graytone transformation function. It considers  $M$  as the maximum value (the absolute white). This code means that the absolute white cannot be reached, and thus  $f \in [0; M[$ . After discretizing the gray values,  $F \in [0; M - 1]$ .



```

1 function f=graytone(F, M)
2 % graytone transform
3 % this is the most important
4 f = M-eps(M)-F;

```



```

1 function l=phi(f, M)
2 % isomorphism
3 l = -M*log(1-f/M);
4 end

6 function f=invphi(l)
7 f = M*(1-exp(-l/M));
8 end

```



```

1 function z=plusLIP(x,y,M)
2 a=double(x);
3 b=double(y);
4 z=a+b-a.*b/M;

```



```

1 function z=timesLIP(alpha,x,M)
2 a=double(x);
3 z=M-M*(1-a/M).^alpha;

```

### 8.5.2 LIP dynamic expansion

The optimal value for dynamic expansion is given by  $\lambda_0$ :

$$\lambda_0(f) = \arg \max_{\lambda} \{ \max(\lambda \triangle f) - \min(\lambda \triangle f) \}$$

Let  $A(\lambda) = \max(\lambda \triangle f) - \min(\lambda \triangle f) = \lambda \triangle \max(f) - \lambda \triangle \min(f)$  and  $B = \ln(1 - \min(f)/M)$  et  $C = \ln(1 - \max(f)/M)$ .

$$\begin{aligned} A'(\lambda) = 0 &\Leftarrow [(M - M \exp(\lambda C)) - (M - M \exp(\lambda B))]' = 0 \\ &\Leftarrow [\exp(\lambda B) - \exp(\lambda C)]' = 0 \\ &\Leftarrow B \exp(\lambda B) - C \exp(\lambda C) = 0 \\ &\Leftarrow \ln(B) + \lambda B = \ln(C) - \lambda C \\ &\Leftarrow \lambda = \frac{\ln(C) - \ln(B)}{B - C} \\ &\Leftarrow \lambda = \frac{\ln(C/B)}{B - C} \end{aligned}$$

Thus, yielding to:

$$\lambda_0(f) = \frac{\ln\left(\frac{\ln(1 - \max(f)/M)}{\ln(1 - \min(f)/M)}\right)}{\ln\left(\frac{M - \min(f)}{M - \max(f)}\right)}$$

The results are shown in Fig. 8.2.

### 8.5.3 Edge detection

When applying an operator in the LIP framework, it is better to apply first the isomorphism, then the operator, and then get back into the classical space. This is applied for example for an edge detection operator.

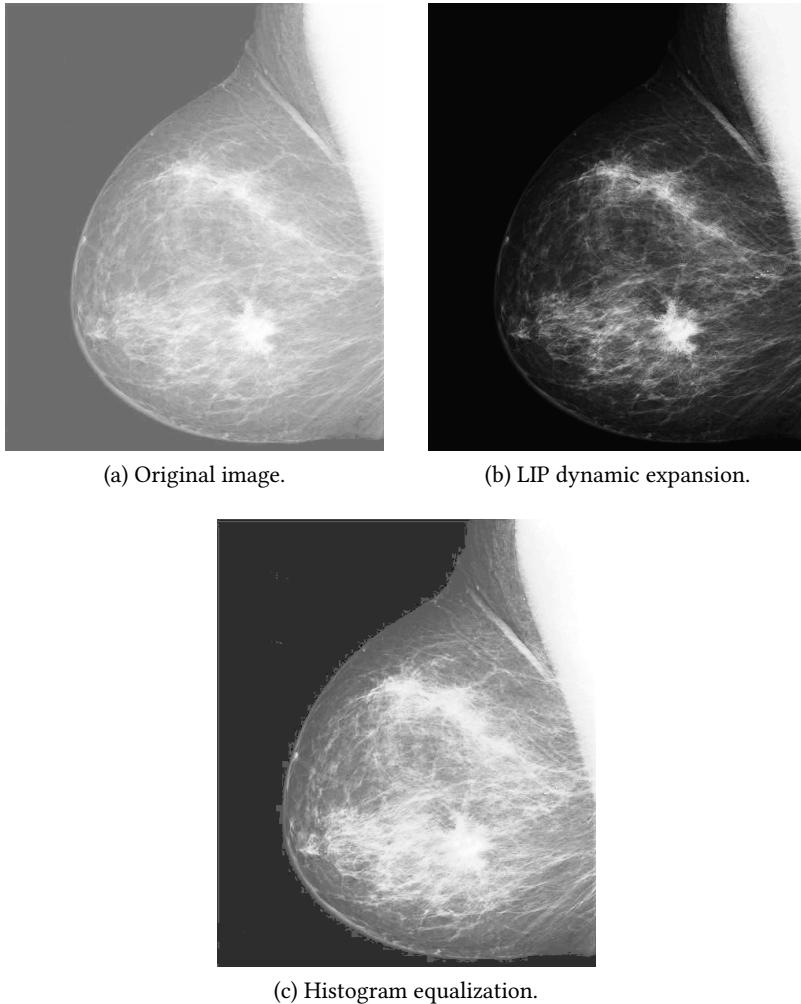
### 8.5.4 Complete MATLAB® code



```

1 %% 1 – Elementary LIP operations
M=256;
3
% read image
5 B=imread('breast . tif ');
B=double(B);
7 % show image
figure ;viewImage(B); title ('Image originale ');
9 % gray-tone function

```



(a) Original image.

(b) LIP dynamic expansion.

(c) Histogram equalization.

Figure 8.2: Results of dynamic expansion. The dark part of the image appears darker.



```
tone = graytone(B, M);
11
D=graytone(timesLIP(2, tone), M);
13 figure ;imshow(D,[0 256]); title ('LIP scalar multiplication by 2');

15 %% 2 - Dynamic expansion

17 l=computeLambda(tone)
E=graytone(timesLIP(l,tone), M);
19 figure
 subplot (1,3,1) ;imshow(B, [0 256]); title ('Original image');
21 subplot (1,3,2) ;imshow(E,[0 256]); title ('Dynamic expansion');
imwrite(uint8(E), 'expanded.png');
23 subplot (1,3,3) ;imshow(255*histeq(B/255),[0 256]);
 title ('Histogram equalization');
25 imwrite(uint8(255* histeq(B/255)), 'histeq .png');

27 %% 3 - Contour detection
% The results are not really convincing in this case.
29 % The important thing is the use of the isomorphism to simplify the computations.
BW = edge(phi(tone, M));
31 figure () ; imshow(BW); title ('LIP edge detection');

33 BW = edge(B);
figure () ; imshow(BW); title ('Classic edge detection');
```



## \*\*\* 9

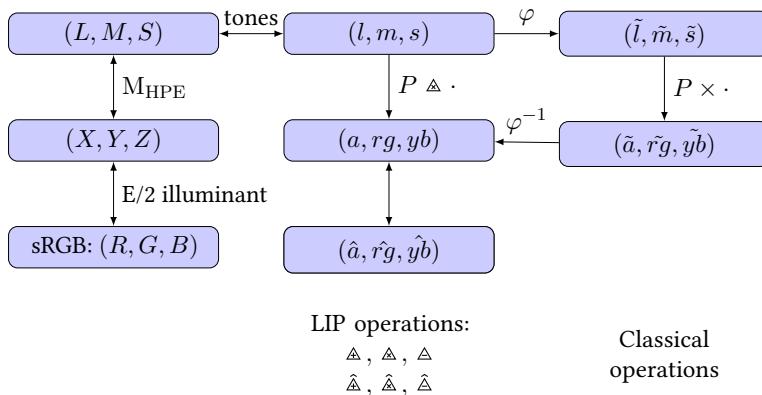
# Color Logarithmic Image Processing (CoLIP)

The objective of this tutorial is to have an overview of the Color Logarithmic Image Processing (CoLIP) framework [?, ?, ?, ?]. The CoLIP is a mathematical framework for the representation and processing of color images. It is psychophysically well justified since it is consistent with several human visual perception laws and characteristics. It allows to consider color images as vectors in an abstract linear space, contrary to the classical color spaces (e.g., RGB and  $L^*a^*b^*$ ). The purpose of this tutorial is to present the mathematical fundamentals of the CoLIP by manipulating the color matching functions and the chromaticity diagram.

## 9.1

# Definitions

This diagram summarizes the transitions between the classical RGB space and the CoLIP space. The transfer matrices are presented in the following.



### 9.1.1 RGB to XYZ

Here, we choose to keep the conventions of the CIE 1931, i.e. a  $2^\circ$  1931 observer, with the equal energy illuminant E and the sRGB space. Depending on the illuminant, there exist many conversion matrices.



Use `rgb2xyz` function to perform the conversion.

### 9.1.2 XYZ to LMS

The matrix  $M_{HPE}$  (Hunt, Pointer, Estevez) is defined by:

$$\begin{pmatrix} L \\ M \\ S \end{pmatrix} = M_{HPE} \times \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}, \quad (9.1)$$

with

$$M_{HPE} = \begin{pmatrix} 0.38971 & 0.68898 & -0.07868 \\ -0.22981 & 1.18340 & 0.04641 \\ 0.00000 & 0.00000 & 1.00000 \end{pmatrix}. \quad (9.2)$$

### 9.1.3 LMS to lms

$$\forall c \in \{l, m, s\}, C \in \{L, M, S\}, c = M_0 \left( 1 - \frac{C}{C_0} \right), \quad (9.3)$$

with  $C_0$  is the maximal transmitted intensity value.  $M_0$  is arbitrarily chosen at normalized value 100. Notice that  $C \in ]0; C_0]$  and  $c \in [0; M_0[$ .

### 9.1.4 CoLIP homeomorphism

$$\forall f \in S, \varphi(f) = -M_0 \ln \left( 1 - \frac{f}{M_0} \right).$$

The logarithmic response of the cones, as in the LIP theory, is modeled through the isomorphism  $\varphi$ :

$$\text{for } c \in \{l, m, s\}, \tilde{c} = \varphi(c) = -M_0 \ln \left( 1 - \frac{c}{M_0} \right), \quad (9.4)$$

where  $(\tilde{l}, \tilde{m}, \tilde{s})$  are called the logarithmic chromatic tones.

### 9.1.5 Opponent process

$$\begin{pmatrix} \tilde{a} \\ \tilde{r}g \\ \tilde{y}b \end{pmatrix} = P \times \begin{pmatrix} \tilde{l} \\ \tilde{m} \\ \tilde{s} \end{pmatrix}, \quad (9.5)$$

with

$$P = \begin{pmatrix} 40/61 & 20/61 & 1/61 \\ 1 & -12/11 & 1/11 \\ 1/9 & 1/9 & -2/9 \end{pmatrix}. \quad (9.6)$$

### 9.1.6 Bounded vector space

Three channels  $\hat{f} = (\hat{a}, \hat{rg}, \hat{yb})$  are defined by:

$$\hat{a} = a \quad (9.7)$$

$$\hat{rg} = \begin{cases} rg & \text{if } rg \geq 0 \\ -\Delta rg & \text{if } rg < 0 \end{cases} \quad (9.8)$$

$$\hat{yb} = \begin{cases} yb & \text{if } yb \geq 0 \\ -\Delta yb & \text{if } yb < 0 \end{cases} \quad (9.9)$$

## 9.2 Applications

### 9.2.1 CMF



Represent the classical chromaticity by using the color matching functions.



The color matching functions in the different spaces, as well as the wavelength and the purple line, are provided in the matrix '`cmf.mat`'.

### 9.2.2 CoLIP chromaticity diagram



Represent the chromaticity diagram in the plane  $(\hat{r}g, \hat{y}b)$  as well as the Maxwell triangle of the RGB colors, as in Fig.9.1.

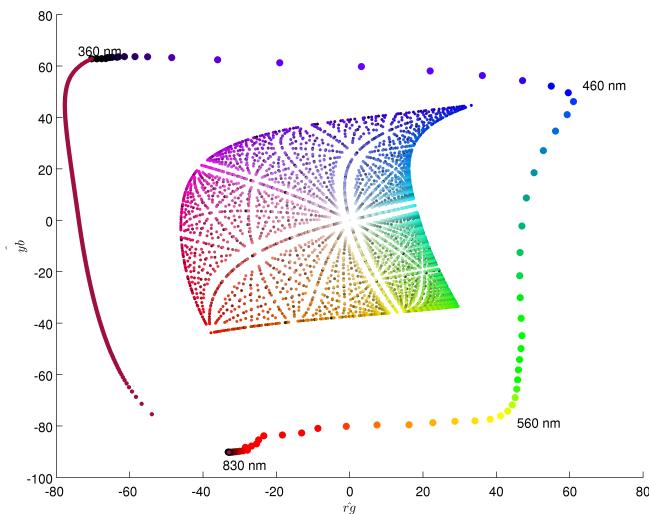


Figure 9.1: Chromaticity diagram in the plane  $(\hat{r}g, \hat{y}b)$ .



### 9.3. Matlab correction



The maximal value is  $M_0$ , arbitrarily fixed at 100.



```
function M0 = getColipM0()
2 % return M0 value
M0 = 100;
```

#### 9.3.1 LMS tones

This is the difficult part of LIP and CoLIP. Be careful with the use of the function `eps` that returns the precision at a given double value.



```
function [ lms ] = lmstone( LMS )
2 % convert LMS values to color tones
% each LMS channel is normalized
4 M0 = getColipM0();
lms = (M0 - eps(M0))*(1-LMS/M0);
```

#### 9.3.2 Isomorphism

The isomorphism is the conversion into/back from the logarithmic space.



```
1 function x = phi(f, M0)
% LIP isomorphism
3 % f : graytone function
% M0: maximal value
5 x = -M0*log(1-f/M0);
```



```
1 function f = invphi(x, M0)
% inverse isomorphism
3 f = M0 * (1-exp(-x/M0))
```

### 9.3.3 XYZ to LMS



```

1 %% convert from XYZ to LMS
% XYZ: data array of dimensions [m, n, 3]
3 % MatPassage: string 'hpe', 'hp64', 'bradford', 'ciecam02'

5 function LMS=XYZ2LMS(XYZ,MatPassage)
if ndims(XYZ) == 3
7     s=3;
[M,N]=size(XYZ (:,:,1) );
9     XYZ=reshape(XYZ,[M*N,3])';
else
11    s=2;
end

13 switch (MatPassage)
15     case('hpe')
        U=[0.38971, 0.68898, -0.07869; -0.22981, 1.18340, 0.04641; 0, 0, 1];
17 end

19 % conversion
LMS=U*XYZ;
21 if s==3
    LMS=reshape(LMS',[M,N,3]);
23 end

```



```

1 function XYZ=LMS2XYZ(LMS,MatPassage)
% convert from LMS into XYZ
3 % LMS: data array of dimensions [m, n, 3]
% MatPassage: string 'hpe', 'hp64', 'bradford', 'ciecam02'
5
7 [M,N]=size(LMS (:,:,1) );
LMS=reshape(LMS,[M*N,3])';
switch (MatPassage)
9     case('hpe')
        U=[0.38971, 0.68898, -0.07869; -0.22981, 1.18340, 0.04641; 0, 0, 1];
11 end

13 XYZ=U\LMS; % inv(U)*LMS
XYZ=reshape(XYZ',[M,N,3]);

```

### 9.3.4 CMF

The color matching functions are provided for convenience. There exist many resources on the internet where they can be found. The classical diagram in the xy space (the horseshoe) is shown in Fig.9.2.



```

load 'cmf.mat'

2
xn = SpecXYZ (:,:,1) ./ sum(SpecXYZ, 3);
4 yn = SpecXYZ (:,:,2) ./ sum(SpecXYZ, 3);
zn = 1-xn-yn;
6 scatter (xn, yn, 30, cmap, ' filled ');

```

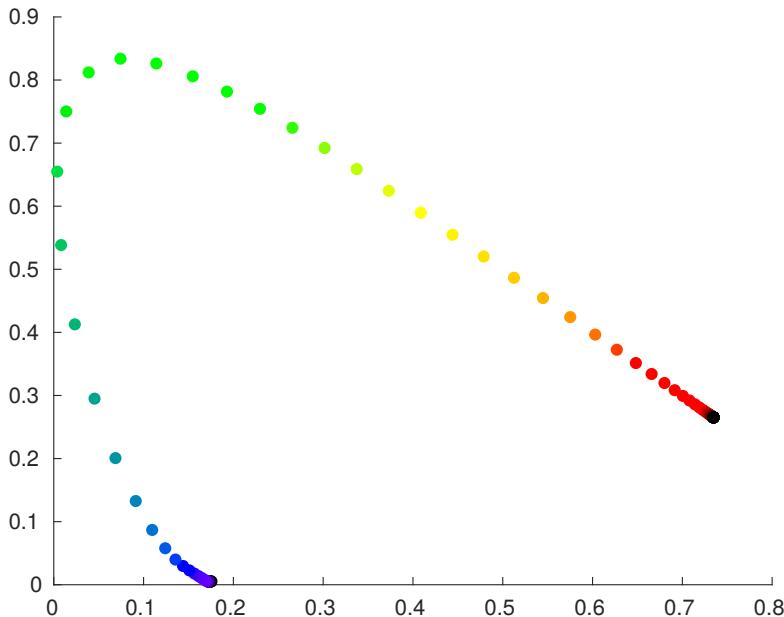


Figure 9.2: Color matching functions in the xy space.

To display the CMF and the cube of all RGB colors in the  $(\hat{r}g, \hat{y}b)$  space, the following code is used:



```

1 ARGYB_hat = LMStoARGYB_chapeau(SpecLMS);
figure (2),
3 hold on
scatter (ARGYB_hat(:,1,2), ARGYB_hat(:,1,3), 30, cmap, ' filled ');
5

```



```
% purple line
7 purple_ARGUMENT_hat = LMStoARGYB_chapeau(pourpresLMS);
scatter (purple_ARGUMENT_hat(:,1,2), purple_ARGUMENT_hat(:,1,3), 30, 'black', 'filled');
9
% RGB cube
11 step=10;
[R, G, B] = ndgrid (0: step :255, 0:step :255, 0:step :255) ;
13
R = reshape(R, numel(R), 1);
15 G = reshape(G, numel(G), 1);
B = reshape(B, numel(B), 1);
17
cubeRGB = cat(2, R, G, B)/255;
19 cubeRGB = reshape(cubeRGB, size(cubeRGB, 1), 1, 3);
colCubeRGB = reshape(cubeRGB, size(cubeRGB,1), 3);
21
% conversion from RGB to a,rg,yb hat
23 cubeXYZ = rgb2xyz(cubeRGB, 'WhitePoint', 'e');
cubeLMS = xyz2lms(cubeXYZ, 'hpe');
25 cube_ARGUMENT_hat = LMStoARGYB_chapeau(cubeLMS);
% display result
27 scatter (cube_ARGUMENT_hat(:,1,2), cube_ARGUMENT_hat(:,1,3), 30, colCubeRGB, 'filled');
```

The results is shown in Fig.9.3. The following functions are used for the conversions.



```
1 function ARGYB_chap=LMStoARGYB_chapeau(LMS)
% LMS: normalized values in ]0;M0]
3
ARGYBtilde = LMStoARGYBtilde(LMS);
5
% conversion
7 ARGYB_chap = ARGYBtildetoARGYBchap(ARGYBtilde);
```



```
1 function ARGYBtilde = LMStoARGYBtilde(LMS)
%%%%%%%%%%%%%
3 % maximal values definition
M0 = getColipM0();
5 [m,n,-] = size (LMS); % image size
7 % conversion to (L~,M~,S~): applying LIP isomorphism
%warning('notice epsilon for conversion into lms tones')
9 LMSton = lmstone(LMS);
LMStilde = phi(LMSton, M0);
```



```

11 % conversion to antagonist color space (a~,rg~,yb~)
13 P = [40/61,20/61,1/61;1,-12/11,1/11;1/9,1/9,-2/9]; % antagonist matrix
14 LMStilde = reshape(LMStilde,[m*n,3]);
15 LMStilde = LMStilde';
16 ARGYBtilde = P*LMStilde;
17 ARGYBtilde = ARGYBtilde';
18 ARGYBtilde = reshape(ARGYBtilde,[m,n,3]);

```



```

function ARGYBchap = ARGYBtildeToARGYBchap(ARGYBtilde)
% function for conversion, takes absolute value
Max = getColipM0();
4 ARGYBchap(:,1) = invphi(ARGYBtilde (:,:,1) , Max);
6 for c=2:3
8     tmp = abs(ARGYBtilde (:,:, c));
10    ARGYBchap(:,:c) = sign(ARGYBtilde (:,:, c)) .* invphi(tmp, Max);
12 end
end

```

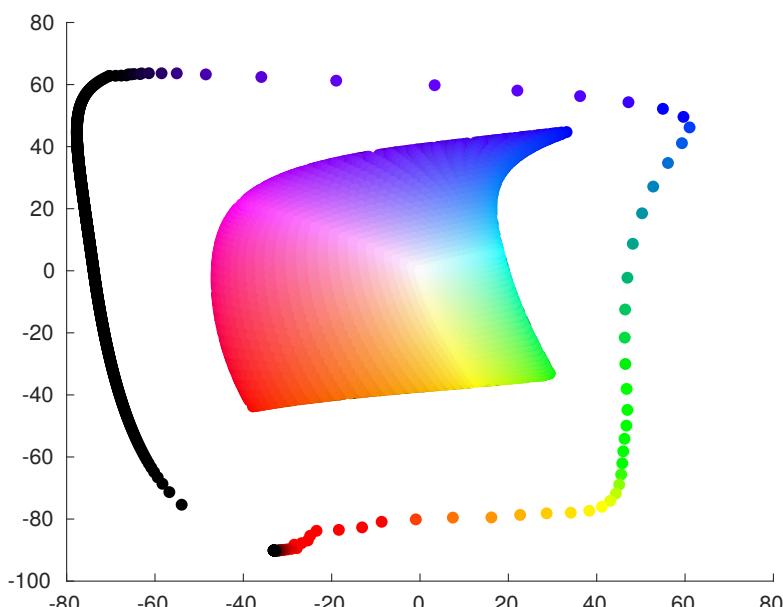


Figure 9.3: Color matching functions in the  $(\hat{r}g, \hat{y}b)$  space.





## 10 GANIP

This tutorial aims to test the elementary operators of the GANIP framework. The General Adaptive Neighborhood Image Processing (GANIP) is a mathematical framework for adaptive processing and analysis of gray-tone and color images. An intensity image is represented with a set of local neighborhoods defined for each pixel of the image to be studied. Those so-called General Adaptive Neighborhoods (GANs) are simultaneously adaptive with the spatial structures, the analyzing scales and the physical settings of the image to be addressed and/or the human visual system.

The following figure illustrates the GANs of two points on an image of retinal vessels.



The GANs are then used as adaptive operational windows for local image transformations (morphological filters, rank/order filters...) and for local image analysis (local descriptors, distance maps...).

The different processes will be applied on the following gray-tone images:

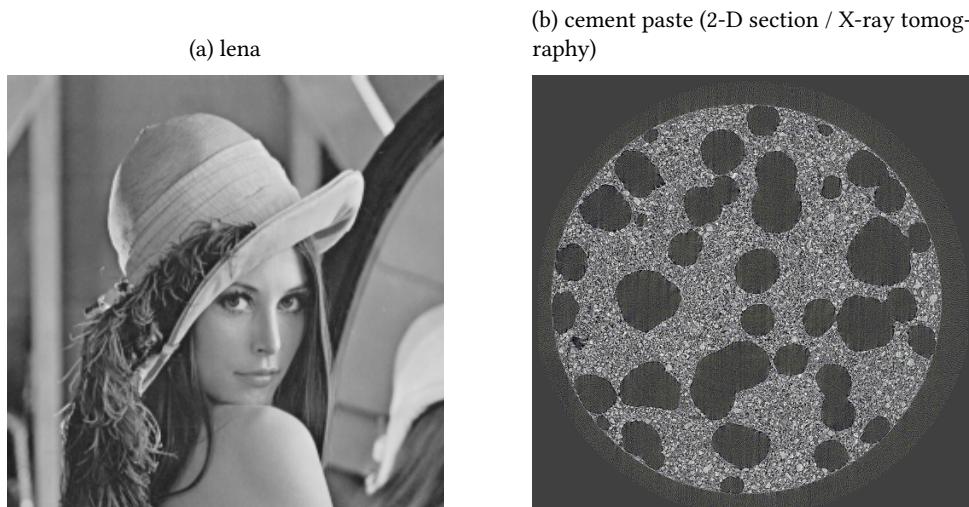
### 10.1 GAN

Let  $f$  be a gray-tone image. The GAN of a point  $x$  using the luminance criterion and the homogeneity tolerance  $m$  within the CLIP framework is defined as:

$$V_m^f(x) = C_{\{y; |f(y) - f(x)| \leq m\}}(x) \quad (10.1)$$

where  $C_X(x)$  denotes the connected component of  $X$  holding  $x$ .

Figure 10.1: Two example images for testing the GAN framework.



1. Load the image 'lena' and compute the GAN of a selected point in the image.
2. Look at the influence of the homogeneity tolerance.
3. Compute the GAN of different points and comment.

## 10.2

## GAN Choquet filtering

The Choquet filters generalize the rank-order filters. In this exercise, we are going to implement some GAN Choquet filters such as the GAN mean operator. For each point  $x$  of the image, the mean value of all the intensities of the points inside the GAN of  $x$  is computed. So, a first algorithm consists in making a loop on the image points for computing the different GAN and the mean intensity values, but it is time consuming. Nevertheless, by using some properties of the GAN (particularly the one giving that iso-valued points can have exactly the same GAN), it is possible to create a second algorithm by making a loop on the gray-tone range:

```
Data: original (8-bit) image  $f$ , homogeneity tolerance  $m$ 
Result: GAN mean filtered image  $g$ 
for  $s = 0$  to 255 do
     $seed =$  points  $x$  with intensity  $f(x) = s$ ;
     $thresh =$  points  $y$  with intensity satisfying  $s - m \leq f(y) \leq s + m$ ;
     $threshGAN =$  connected components of  $thresh$  holding  $seed$ ;
    foreach label of  $threshGAN$  do
         $currentLabel =$  current connected component of  $threshGAN$ ;
         $meanValue =$  intensity mean value of the image points inside
         $currentLabel$ ;
        set  $g(x) = meanValue$  to the points  $x$  of  $seed$  belonging to
         $currentLabel$ ;
    end
end
```



1. Implement the proposed algorithm.
2. Test this operator on the 'lena' image with different homogeneity tolerances.
3. Compare the result with a classical mean filtering.



See `imfilter`.

## 10.3

# GAN morphological filtering

As previously explained, it is also possible to compute the GAN dilation and GAN erosion by making a loop on the gray-tone range. The algorithm for the GAN dilation is:

```

Data: original (8-bit) image  $f$ , homogeneity tolerance  $m$ 
Result: GAN dilated image  $g$ 
set  $g(x) = 0$  for all points  $x$ ;
for  $s = 0$  to 255 do
     $seed =$  points  $x$  with intensity  $f(x) = s$ ;
     $thresh =$  points  $y$  with intensity satisfying  $s - m \leq f(y) \leq s + m$ ;
     $threshGAN =$  connected components of  $thresh$  holding  $seed$ ;
    foreach label of  $threshGAN$  do
         $currentLabel =$  current connected component of  $threshGAN$ ;
         $maxValue =$  intensity maximum value of the image points inside
         $currentLabel$ ;
        set  $g(x) = max(g(x), maxValue)$  to the points  $x$  belonging to
         $currentLabel$ ;
    end
end
```

**Algorithm 2:** GAN dilation

The algorithm for the GAN erosion is similar.



1. Implement the GAN dilation and GAN erosion.
2. Test this operator on the 'lena' image with different homogeneity tolerances.
3. Compare the results with the classical morphological dilation and erosion..
4. Compute, test, compare and check the properties (idempotence, extensivity/anti-extensivity) of the GAN opening and GAN closing.



Compare with the functions `imdilate` and `imerode`.



## 10.4. Matlab correction



### 10.4.1 GAN

In order to show the General Adaptive Neighborhood (GAN) of one image point, the following function is implemented. Note that the function is given within the Classical Linear Image Processing (CLIP) framework, i.e. with the usual addition, subtraction and scalar multiplication. But the function could be generalized to other models such as the Logarithmic Image Processing (LIP).



```

function RES = GAN(A,p,m)
% A is the original gray level image
% p is the image point coordinates
% m is the homogeneity tolerance
RES = zeros( size(A));
RES(p(2),p(1)) = 1;
s = A(p(2),p(1));
thresh = (A >= s-m) & (A <= s+m);
RES = imreconstruct( logical (RES),thresh ,8) ;

```

In this way, we can visualize the GAN of any point of the 'Lena' image:



```

1 A=imread('lena256.bmp');
A=double(A);
3 p =[200,100];
mtol =[5,50,75];
5 GAN1=GAN(A,p,mtol(1));
GAN2=GAN(A,p,mtol(2));
7 GAN3=GAN(A,p,mtol(3));
figure
9 subplot (231);imshow(A,[]);title (' original image')
subplot (232);imshow(A,[]);hold on;plot (p(1),p(2),'+r');title ('seed point')
11 subplot (234);imshow(GAN1,[]);title ('GAN / m=5')
subplot (235);imshow(GAN2,[]);title ('GAN / m=50')
13 subplot (236);imshow(GAN3,[]);title ('GAN / m=75')

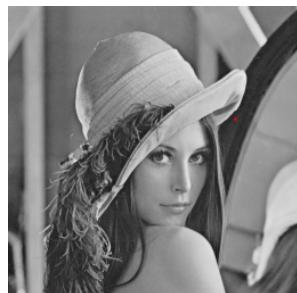
```

### 10.4.2 GAN Choquet filtering

In order to compute the GAN mean filtering, the basic idea is to make a loop on the image points, compute the GAN of the current point and then calculate the mean intensity of the points within the GAN. But this is time computing in the sense that two points with the same intensity can have exactly the same GAN (when one



(a) Original image.



(b) Point location.

(c) GAN ( $m=5$ ).(d) GAN ( $m=50$ ).(e) GAN ( $m=75$ ).

Figure 10.2: GAN of a specific point of the 'Lena' image using different homogeneity tolerances  $m$  within the CLIP framework.

point is included in the GAN of the second point with same intensity). Therefore, a more effective way is to make a loop on the gray levels of the image. The GAN mean filtering is then implemented as:



```

1 function RES = GANmean(A,m)
% A: original image
3 % m: homogeneity tolerance
RES = zeros( size (A));
5 parfor s = 0:255
    thresh = (A >= s-m) & (A <= s+m);
7    seed = (A == s);
    thresh = imreconstruct(seed,thresh ,8) ;
9    label = bwlabeln(thresh ,8) ;
nbLabel = max(label (:));
11   for n = 1:nbLabel;
        currentLabel = ( label == n);
13       values = A(currentLabel);
        meanValue = mean(values);
15       result = meanValue.*currentLabel .* seed;
        RES = RES + result ;
17   end
end

```

Looking at this function, for the first loop on the gray levels  $s$ , we detect the GANs of the pixels with gray level  $s$ . So 'thresh' contains all these connected components (GANs), where some points with identical gray levels  $s$  can have the same GAN. 'label' enables the different GANs to be labeled. For the second loop on these GANs, we extract for each GAN its gray levels which are stored in 'values'. Afterwards, we take the mean value 'meanValue' that is stored as the resulting gray level for pixels inside the GAN with the same gray level  $s$ .

We can compare this GAN filtering with the classical one using a fixed operational window.

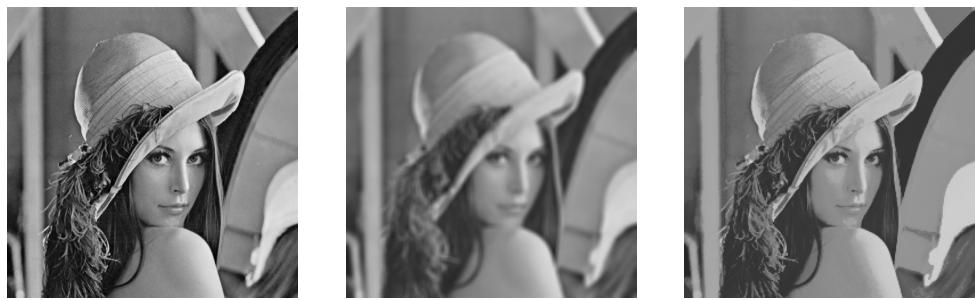


```

h=ones(5,5) /25;
2 B=imfilter (A,h, 'symmetric');
mtol=30;
4 C=GANmean(A,mtol);
figure
6 subplot (131);imshow(A,[]); title (' original image')
    subplot (132);imshow(B,[]); title (' classical mean filtering ')
8 subplot (133);imshow(C,[]); title ('GAN mean filtering ')

```

You can see the blurring effect caused by the classical filtering, contrary to the adaptive GAN filtering where the transitions are much more preserved while smoothing the image.



(a) Original image. (b) Classical filtering ( $r = 5$ ). (c) GAN filtering ( $m = 30$ ).

Figure 10.3: Classical ( $r = 5$ ) vs. GAN ( $m = 30$ ) mean filtering of the 'Lena' image.

### GAN morphological filtering

The followinh codes enable the GAN dilation and erosion to be computed. As previously mentioned, it is based on a loop on the gray levels and not on the image points. Note that the GANs ae computed on the criterion image (it is important for satisfying the good properties of opening and closing, such as idempotence).



```

1 function RES = GANDilation(A,h,m)
2 % A: original image
3 % h: criterion image
4 % m: homogeneity tolerance
5 RES = zeros( size(A));
6 parfor s = 0:255
7     thresh = (h >= s-m) & (h <= s+m);
8     seed = (h == s);
9     thresh = imreconstruct(seed,thresh ,8);
10    label = bwlabeln(thresh ,8);
11    nbLabel = max(label (:));
12    for n = 1:nbLabel
13        currentLabel = (label == n);
14        values = A(currentLabel);
15        values = sort(values);
16        result = double(values(length(values)))*currentLabel;
17        RES = max(RES,result);
18    end
end

```



```

1 function RES = GANerosion(A,h,m)
2 % A: original image
3 % h: criterion image

```



```
% m: homogeneity tolerance
5 RES = 255*ones( size (A));
parfor s = 0:255
7     thresh = (h >= s-m) & (h <= s+m);
    seed = (h == s);
9     thresh = imreconstruct(seed,thresh ,8) ;
    label = bwlabeln(thresh ,8) ;
11    nbLabel = max(label (:));
    for n = 1:nbLabel;
13        currentLabel = ( label == n);
        values = A(currentLabel);
        values = sort(values);
        result = double(values (1))* currentLabel +255*(~ currentLabel);
17        RES = min(RES,result);
    end
19 end
```

The following script shows the difference between classical and adaptive morphology.



```
1 se=strel ('disk',2);
Bdil=imdilate(A,se);
3 Bero=imerode(A,se);
mtol=30;
5 Cdil=GANdilation(A,mtol);
Cero=GANerosion(A,mtol);
7 figure
    subplot (231);imshow(A,[]);title (' original image')
9 subplot (232);imshow(Bdil,[]);title (' classical dilation ')
    subplot (233);imshow(Bero,[]);title (' classical erosion ')
11 subplot (235);imshow(Cdil,[]);title ('GAN dilation ')
    subplot (236);imshow(Cero,[]);title ('GAN erosion')
```

As previously mentioned with the Choquet filtering, you can see the blurring effect caused by the classical filtering, contrary to the adaptive GAN filtering where the transitions are much more preserved while smoothing the image.

Now, we can compute the GAN opening and closing as combined operators of dilation and erosion.



```
function RES = GANopening(A,h,m)
2 % A: original image
% h: criterion image
4 % m: homogeneity tolerance
temp = GANerosion(A,h,m);
```



(a) Original image.

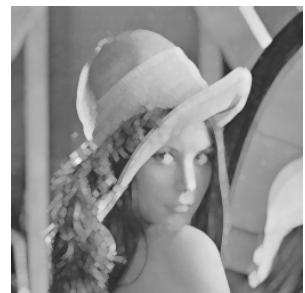
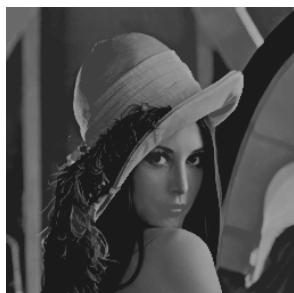
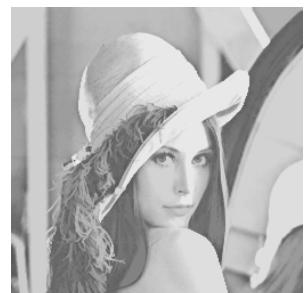
(b) Classical erosion ( $r = 2$ ).(c) Classical dilation ( $r = 2$ ).(d) GAN erosion ( $m = 30$ ).(e) GAN dilation ( $m = 30$ ).

Figure 10.4: Classical ( $r = 2$ ) vs. GAN ( $m = 30$ ) morphological filtering of the 'Lena' image.



```
6 RES = GANDilation(temp,h,m);
```



```
function RES = GANClosing(A,h,m)
2 % A: original image
% h: criterion image
4 % m: homogeneity tolerance
temp = GANDilation(A,h,m);
6 RES = GANerosion(temp,h,m);
```

It is very important to use the same criterion  $h$  when combining these two operators of GAN dilation and erosion. It means that we compute the GANs at the beginning and thereafter we use these same GANs for computing dilation and erosion. It enables the idempotence, extensivity/anti-extensivity of the GAN opening and GAN closing to be satisfied.

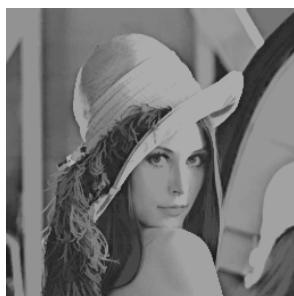


```
Copen = GANopening(A,A,mtol);
2 Cclose = GANClosing(A,A,mtol);
figure
4 subplot(131); imshow(A,[]); title('original image')
subplot(132); imshow(Copen,[]); title('GAN opening')
6 subplot(133); imshow(Cclose,[]); title('GAN closing')
```

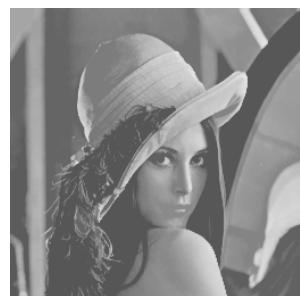
The result of such morphological filters is presented in Fig.10.5.



(a) Original image.



(b) GAN opening ( $m = 30$ ).



(c) GAN closing ( $m = 30$ ).

Figure 10.5: GAN ( $m = 30$ ) opening and closing of the 'Lena' image.

We can check some properties of these morphological filters such as extensivity/anti-extensivity and idempotence:



```
Copen = GANopening(A,A,mtol);
2 Copen2 = GANopening(Copen,A,mtol);
Cclose = GANClosing(A,A,mtol);
4 Cclose2 = GANClosing2(Cclose,A,mtol);
% extensivity / anti-extensivity :
6 min(min(Copen<=A)))
min(min(Cclose>=A)))
8 % idempotence
min(min(Copen2==Copen)))
10 min(min(Cclose2==Cclose)))
```

The result is 1 for all these Boolean tests.



## 11 Image Filtering using PDEs

This tutorial aims to process images with the help of partial differential equations.

The different processes will be applied on the following MR image Fig. 11.1.



Figure 11.1: MRI image of a human brain.

## Notations

The different operators used here are:

$$\vec{A} = \begin{pmatrix} A_x \\ A_y \end{pmatrix} \quad (11.1)$$

$$\operatorname{div} \vec{A} = \frac{\partial A_x}{\partial x} + \frac{\partial A_y}{\partial y} \quad (11.2)$$

$$\vec{\operatorname{grad}} u(x, y) = \nabla u = \begin{pmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \end{pmatrix} \quad (11.3)$$

The Laplacian operator is denoted  $\Delta u = \nabla^2 u = \operatorname{div} \vec{\operatorname{grad}} u$ . A numerical approximation can be used, but is not recommended in this tutorial.

## 11.1 Linear diffusion

The heat equation is defined as follows:

$$\begin{cases} \frac{\partial u}{\partial t}(x, y, t) = \operatorname{div}(\nabla u(x, y, t)) \\ \quad = \frac{\partial^2 u}{\partial x^2}(x, y, t) + \frac{\partial^2 u}{\partial y^2}(x, y, t) \\ u(x, y, 0) = f(x, y) \end{cases} \quad (11.4)$$

If  $f(x, y)$  is an image (with  $(x, y) \in D \subset \mathbb{R}^2$ ,  $D$  is the spatial support), this defines a filtering method that is equivalent to the convolution of  $f$  by a Gaussian function [?]. This equation can be solved by a finite difference numerical method.

### 11.1.1 Numerical scheme

The following notations are employed.  $N, S, E, W$  stand for north, south, east and west.

$$\begin{aligned} +\delta^N u &= \frac{u(x, y+h) - u(x, y)}{h} \\ -\delta^S u &= \frac{u(x, y-h) - u(x, y)}{h} \\ -\delta^E u &= \frac{u(x-h, y) - u(x, y)}{h} \\ +\delta^W u &= \frac{u(x+h, y) - u(x, y)}{h} \end{aligned} \quad (11.5)$$

Thus, the numerical scheme is ( $h$  has value 1):

$$\frac{u^{t+1} - u^t}{\delta t} = \frac{1}{h} \left\{ \delta^N u - \delta^S u + \delta^E u - \delta^W u \right\} \quad (11.6)$$



1. Code the numerical scheme. It should use two parameters: the number of iterations  $n$ , and the step time  $\delta t$ .
2. Filter the original image by varying the parameters of the discrete diffusion process.
3. Comment the filtering results. Compare the results with the Gaussian filter.

## 11.2 Nonlinear diffusion

Image filtering by nonlinear diffusion reduces noise in a controlled way. The diffusion coefficient is locally adapted, becoming negligible as object boundaries are

approached.

The heat equation is replaced by the following PDE:

$$\begin{cases} \frac{\partial u}{\partial t}(x, y, t) = \operatorname{div}(c(\|\nabla u(x, y, t)\|) \cdot \nabla u(x, y, t)) \\ u(x, y, 0) = f(x, y) \end{cases} \quad (11.7)$$

with  $c$  the diffusion function satisfying the following properties:

- $c(0) = 1$ ,
- $\lim_{s \rightarrow +\infty} sc(s) = 0$ ,
- $\forall s > 0, c'(s) \leq 0$ .

Perona and Malik have proposed 2 diffusion coefficients [?, ?]:

- $c_1 : \exp\left(-\left(\frac{s}{\alpha}\right)^2\right)$ ,
- $c_2 : \frac{1}{1 + \left(\frac{s}{\alpha}\right)^2}$ .

### 11.2.1 Numerical scheme

The numerical scheme is the following:

$$\frac{u^{t+1} - u^t}{\delta t} = \frac{1}{h^2} \left\{ c(|\delta^N u|) \cdot \delta^N u + c(|\delta^S u|) \cdot \delta^S u + c(|\delta^E u|) \cdot \delta^E u + c(|\delta^W u|) \cdot \delta^W u \right\} \quad (11.8)$$



1. Code the numerical scheme
2. Filter the original image by varying the parameters of the discrete nonlinear diffusion process.
3. Comment and compare the filtering results with the previous scheme.

## 11.3 Degenerate diffusion

The following degenerate PDEs have been shown to be equivalent to the morphological operators of dilation and erosion (using a disk as structuring element, see tutorial on mathematical morphology):

$$\begin{cases} \frac{\partial u}{\partial t}(x, y, t) = +\|\nabla u(x, y, t)\| \\ u(x, y, 0) = f(x, y) \end{cases} \quad (11.9)$$

$$\begin{cases} \frac{\partial u}{\partial t}(x, y, t) = -\|\nabla u(x, y, t)\| \\ u(x, y, 0) = f(x, y) \end{cases} \quad (11.10)$$

### 11.3.1 Numerical schemes

The previous numerical scheme is easy to find, but the results present large differences with the dilation and erosion operators (shocks due to discontinuities in the original image). This is why the following numerical scheme is preferred [?]:

For the erosion:

$$\frac{u^{t+1} - u^t}{\delta t} = \frac{1}{h^2} \sqrt{max^2(0, -\delta^N u) + min^2(0, -\delta^S u) \dots + max^2(0, -\delta^W u) + min^2(0, -\delta^E u)} \quad (11.11)$$

For the dilation:

$$\frac{u^{t+1} - u^t}{\delta t} = \frac{1}{h^2} \sqrt{min^2(0, -\delta^N u) + max^2(0, -\delta^S u) \dots + min^2(0, -\delta^W u) + max^2(0, -\delta^E u)} \quad (11.12)$$



1. Code the numerical scheme.
2. Filter the original image while varying the parameters of the discrete degenerate diffusion process.
3. Comment the morphological filtering results and compare the numerical scheme to the approach with operational windows.



## 11.4. Matlab correction



### 11.4.1 Linear diffusion

The numerical scheme is coded as follows with matlab:



```

1 function Z = linearDiffusion (I, nbIter , dt)
2 % I: Original image
3 % nbIter : number of iterations
4 % dt: step time

5 h = [[0 1 0];[1 -4 1];[0 1 0]];
6 Z=I; % initialization
7
8 for i=1:nbIter ;
9     Z = Z + dt * conv2(Z, h, 'same');
10    end;

```

Another way of coding this operator is to compute 4 gradients. This simplifies the formulation of the nonlinear diffusion filter, illustrated in Fig.11.2.



```

1 function Z = linearDiffusion (I, nbIter , dt)
2 % I: Original image
3 % nbIter : number of iterations
4 % dt: step time
5
6 % masks for gradients computation
7 hW = [1 -1 0];
8 hE = [0 -1 1];
9 hN = hW';
10 hS = hE';
11
12 Z = I; % initialization
13
14 for i=1:nbIter ;
15     % calculate gradient in all directions (N,S,E,W)
16     gW = imfilter (Z,hW);
17     gE = imfilter (Z,hE);
18     gN = imfilter (Z,hN);
19     gS = imfilter (Z,hS);

20     % next Image
21     Z = Z + dt*(gN + gS + gW + gE);
22
23 end;

```

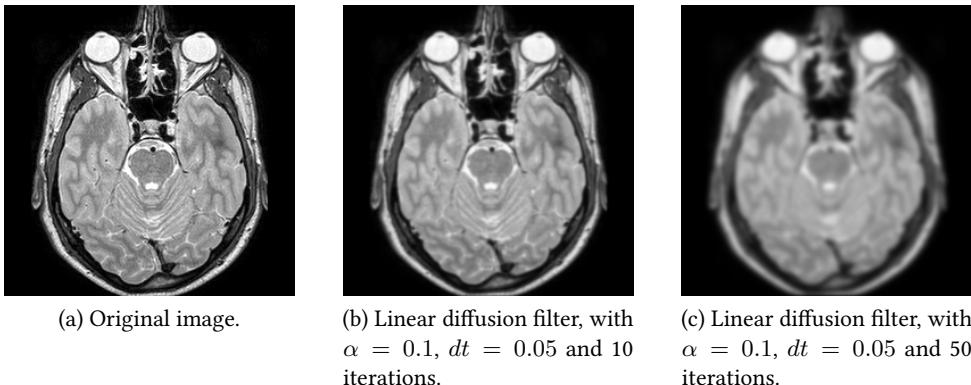


Figure 11.2: Linear diffusion filter. Contours are not preserved: this is equivalent to a Gaussian filter.

### 11.4.2 Nonlinear diffusion

The nonlinear diffusion is an adaptation of the diffusion to the informations contained in the images (see Fig.11.3). These informations are high frequency components, evaluated by the gradients in the 4 directions. A specific coefficient is thus applied to each of these directions.



```

1 function Z = nonlinearDiffusion (I, nbIter ,dt ,alpha)
% I: Original image
3 % nbIter : number of iterations
% dt: step time
5 % alpha: diffusion parameter

7 hW = [1 -1 0];
hE = [0 -1 1];
9 hN = hW';
hS = hE';

11 Z = I;
13
14 for i=1:nbIter
% calculate gradient in all directions (N,S,E,W)
15 gW = imfilter (Z,hW);
gE = imfilter (Z,hE);
17 gN = imfilter (Z,hN);
gS = imfilter (Z,hS);

19
% next Image
Z = Z + dt.*c(gN,alpha).*gN + c(gS,alpha).*gS + c(gW,alpha).*gW + c(gE,alpha)
    ↵ .* gE);
21
22 end

```

with  $c$  defined as follows:



```

1 function Z = c(I, alpha)
% Perona Malik diffusion coefficient
3 % I: input image
% alpha: diffusion parameter
5 %
% absolute value is not necessary in this case
7 Z = exp(-(I/alpha).^2);

```

Another possibility is:



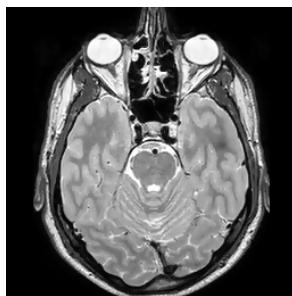
```

1 function Z = c2(I, alpha)
% Perona Malik diffusion coefficient
3 % I: input image
% alpha: diffusion parameter
5 %
% absolute value is not necessary in this case
7 Z = 1/(1+(I/alpha).^2);

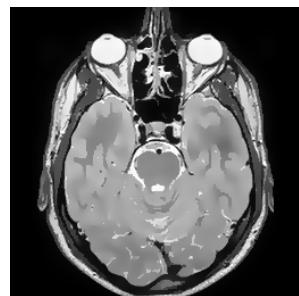
```



(a) Original image.



(b) Non linear diffusion filter, with  $\alpha = 0.1$ ,  $dt = 0.05$  and 10 iterations.



(c) Non linear diffusion filter, with  $\alpha = 0.1$ ,  $dt = 0.05$  and 100 iterations.

Figure 11.3: Nonlinear diffusion filter. Contours are preserved.

### 11.4.3 Degenerate diffusion

Erosion and dilation can theoretically be coded with this numerical scheme. However, some numerical problems appear in the first version (Fig.11.4), which are corrected in the second version (Fig.11.5).

### First version

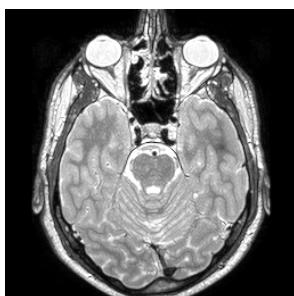
This first version is the direct transcription of the numerical scheme. As observed in Fig.11.4, shocks (peaks) appear after a few iterations.



```

1 function [Zerosion, Zdilation ] = morphologicalDiffusion2(I, nbIter , dt)
2 % I: original image
3 % nbIter: number of iterations
4 % dt: time increment
5 h = [-1 0 1];
6
7 Zerosion = I;
8 Zdilation = I;
9
10 for i=1:nbIter
11     % calculate gradient in vertical V and horizontal H directions , for
12     % dilation
13     gH = imfilter ( Zdilation , h);
14     gV = imfilter ( Zdilation , h');
15
16     % same computation, for erosion
17     jH = imfilter (Zerosion, h);
18     jV = imfilter (Zerosion, h');
19
20     % next step
21     Zdilation = Zdilation + dt * sqrt(gV.^2 +gH.^2);
22     Zerosion = Zerosion - dt * sqrt(jV.^2 +jH.^2);
23
24 end

```



(a) Dilation by diffusion, for  
 $dt = 0.02$  and  $\text{nbIter}=20$ .



(b) Dilation by diffusion, for  
 $dt = 0.02$  and  $\text{nbIter}=50$ .

Figure 11.4: Mathematical morphology operations by diffusion.

### More sophisticated version

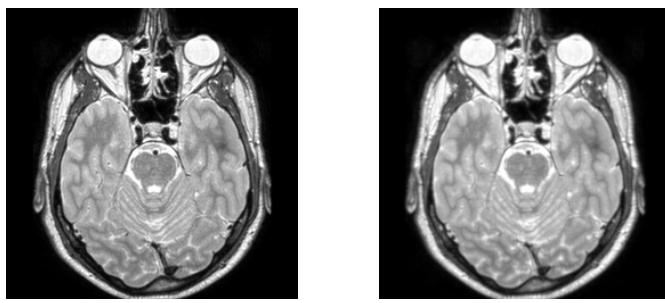
Another scheme, more numerically stable, can be preferred to the previous one. Results are presented in Fig. 11.5.



```
function [Zerosion, Zdilation ] = morphologicalDiffusion(I, nbIter ,dt)
2 % I: original image
% nbIter: number of iterations
4 % dt: time increment
hW = [1 -1 0];
6 hE = [0 -1 1];
hN = hW';
8 hS = hE';

10 Zerosion = I;
Zdilation = I;
12
for i=1:nbIter ;
% calculate gradient in all directions (N,S,E,W)
14 gW = imfilter ( Zdilation ,hW);
16 gE = imfilter ( Zdilation ,hE);
gN = imfilter ( Zdilation ,hN);
18 gS = imfilter ( Zdilation ,hS);

20 jW = imfilter (Zerosion,hW);
jE = imfilter (Zerosion,hE);
22 jN = imfilter (Zerosion,hN);
jS = imfilter (Zerosion,hS);
24
% next step
26 g = sqrt( min(0,-gW).^2 + max(0,gE).^2 + min(0,-gN).^2 + max(0,gS).^2 );
j = sqrt( max(0,-jW).^2 + min(0,jE).^2 + max(0,-jN).^2 + min(0,jS).^2 );
28 Zdilation = Zdilation + dt * g;
Zerosion = Zerosion - dt * j;
30
end;
```



(a) Dilation by diffusion, for  
 $dt = 0.02$  and nbIter=20.

(b) Dilation by diffusion, for  
 $dt = 0.02$  and nbIter=50.

Figure 11.5: Mathematical morphology operations by diffusion, sophisticated version.

# ★ 12 Multiscale Analysis

This tutorial aims to study some multiscale image processing and analysis methods, based on pyramidal and scale-space representations.

The different processes will be applied on the following MR image.

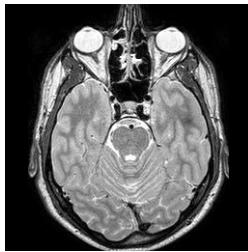


Figure 12.1: Brain MR image.

## 12.1 Pyramidal decomposition and reconstruction

### 12.1.1 Decomposition

```
Data: image  $A_0$ 
Result: pyramid of approximations  $\{A_i\}_i$ , pyramid of details  $\{D_i\}_i$ 
for  $i=1$  to  $3$  do
    filtering:  $F = \text{filt}(A_{i-1})$ ;
    subsampling:  $A_i = \text{ech}(F, 0.5)$ ;
    details:  $D_i = A_{i-1} - \text{ech}(A_i, 2)$ ;
end
```

**Algorithm 3:** The algorithm of the pyramidal decomposition

The initial image (with the highest resolution), denoted  $A_0$ , represents the level 0 of the pyramid. To build the pyramid, we successively carry out the following steps:

1. a Gaussian filtering,
2. a subsampling,
3. a calculation of the details (residues).



Make a pyramidal decomposition with 4 levels of the image 'brain'.



You can use the MATLAB® command cell to store the images of the different levels of the pyramid. imfilter and fspecial will be used to perform the filtering process, imresize is used to down- or over-sampling the image.

### 12.1.2 Reconstruction

To reconstruct the original image, we carry out the following steps at each level of the pyramid:

1. an oversampling,
2. an addition of the details.

**Data:** image  $A_3$ , pyramid of details  $\{D_i\}_i$

**Result:** reconstructed pyramid  $\{B_i\}_i$

initialization :  $B_3 = A_3$ ;

**for**  $i=3$  to 1 **do**

1    oversampling:  $R = ech(B_i, 2)$ ;

2    adding details:  $B_{i-1} = R + D_i$

**end**

**Algorithm 4:** The algorithm of the pyramidal reconstruction



1. Reconstruct the original image from the last level of the pyramid.
2. Make the same reconstruction without adding the details.
3. Calculate the resulting error between the reconstructed image and the original image.

## 12.2

# Scale-space decomposition and multiscale filtering

We are going to decompose (without any sampling) the image with the highest resolution with a morphological operator, dilation or erosion. The resulting images will have the same size.

## 12.2.1 Morphological multiscale decomposition



Build the two scale-space decompositions of the 'brain' image, with a disk of increasing radius as a structuring element



See the MATLAB® functions `imdilate` and `imerode`.



Informations

See the functions `morphology.erosion` and `morphology.dilation` from `skimage`.

## 12.2.2 Kramer and Bruckner multiscale decomposition

Now, we are going to use the iterative filter of Kramer and Bruckner [?] defined as:

$$MK_B^n(f) = K_B(MK_B^{n-1}(f)) \quad (12.1)$$

where  $B$  denotes a disk of a fixed radius  $r$ , and:

$$K_B(f)(x) = \begin{cases} D_B(f)(x) & \text{if } D_B(f)(x) - f \leq f - E_B(f)(x) \\ E_B(f)(x) & \text{otherwise} \end{cases} \quad (12.2)$$

where  $D_B(f), E_B(f)$  represent respectively the dilation and the erosion of the image  $f$  with the structuring element  $B$ .



Implement and test this filter on the image 'brain' for different values of  $n$ .



## 12.3. Matlab correction



### 12.3.1 Pyramidal decomposition and reconstruction

#### Decomposition

The following function makes the decomposition of the Laplacian and Gaussian pyramids at the same time. The Laplacian pyramid can be reconstructed without any additional information. This is illustrated in Fig. 12.2.



```

function [pyrG, pyrL] = LaplacianPyramidDecomposition(Image, levels, mode)
% Gaussian and Laplacian Pyramid decomposition
% Image must be either 2D (grayscale) or 3D (color)
% levels : number of levels of decomposition (not including the original level)
% mode: approximation mode 'bilinear', 'nearest', etc. for use in imresize
%         bilinear by default
% pyrG: Gaussian pyramid
% pyrL: Laplacian pyramid
%
% The Laplacian pyramid is of size levels +1. The last image pyrL{levels +1}
% is the approximation image of the last level. The original image is
% exactly reconstructed by the LaplacianPyramidReconstruction
% function .
%
% pyramids
pyrL = cell ( levels +1, 1);
pyrG = cell ( levels +1, 1);
%
% gaussian filter
H = fspecial ('gaussian');

if ~exist ('mode', 'var')
    mode = 'bilinear';
end

for i=1: levels
    ImagePrec = Image; % previous image
    g= imfilter (Image,H,'same');

    Image = imresize (g, .5, mode);
    ImagePrime = imresize (Image, [ size (g,1), size (g,2)], mode);

    pyrL{i} = ImagePrec - ImagePrime;
    pyrG{i} = ImagePrec;

end
pyrL{ levels +1} = Image;
pyrG{levels +1} = Image;

```



Notice that the images are of type double, which implies a special care when displaying them, for example by using the command `imshow(I, [])`.

## Reconstruction

The reconstruction is straightforward and exact because of the construction of the residue. The details can be filtered (removed for example), thus giving the following result Fig. 12.3.



```

1 function Image = LaplacianPyramidReconstruction(pyr, mode)
2 % Laplacian Pyramid Reconstruction
3 % pyr: Laplacian pyramid
4 % mode: approximation mode for imresize, bilinear by default
5
6 if ~exist ('mode', 'var')
7     mode = 'bilinear';
8 end
9
10 levels = size (pyr, 1)-1;
11
12 Image = pyr{ levels +1};
13 % reconstruction from bottom to top
14 for i=levels:-1:1
15     Image = pyr{i} + imresize (Image, ...
16                             [ size (pyr{i},1) , size (pyr{i}, 2)], mode);
17 end

```

## 12.3.2 Scale-space decomposition and multiscale filtering



```

1 % Morphological scale –space decomposition
2 k=3; % number of decompositions
3 ss=cell (2,k);
4 for i=1:k
5     se = strel ('disk',2*i); % structuring element
6     ss {1, i}=imdilate (A,se); % dilation
7     ss {2, i}=imerode(A,se); % erosion
8 end

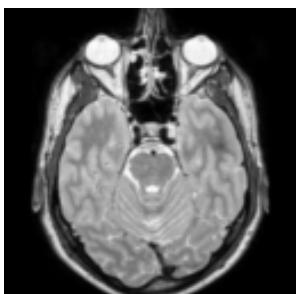
```

## 12.3.3 Kramer and Bruckner multiscale decomposition

The results are illustrated in Fig.12.6.



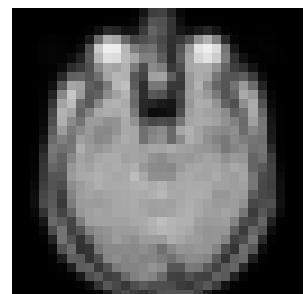
(a) Original image.



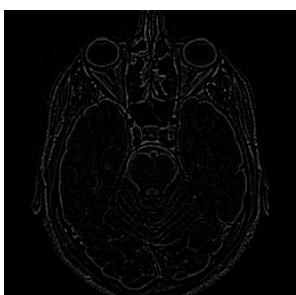
(b) Gaussian pyramid level 1.



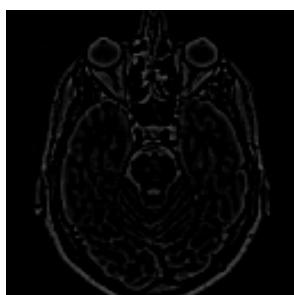
(c) Level 2.



(d) Level 3.



(e) Laplacian pyramid level 1.



(f) Level 2.



(g) Level 3.

Figure 12.2: Gaussian and Laplacian pyramids, for 3 levels of decomposition. The Laplacian pyramid in addition to the last level of the Gaussian pyramid is required to exactly reconstruct the original image.

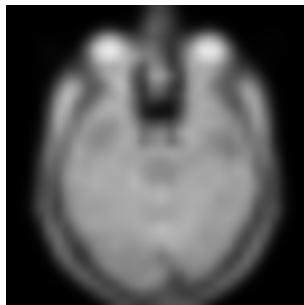
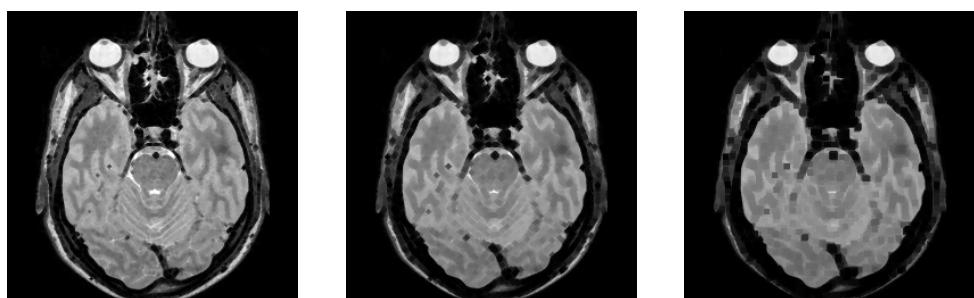


Figure 12.3: Reconstruction of the pyramid without any detail.



(a) Dilation scale 1. (b) Dilation scale 2. (c) Dilation scale 3.

Figure 12.4: Morphological multiscale decomposition by dilation.



(a) Erosion scale 1. (b) Erosion scale 2. (c) Erosion scale 3.

Figure 12.5: Morphological multiscale decomposition by erosion.

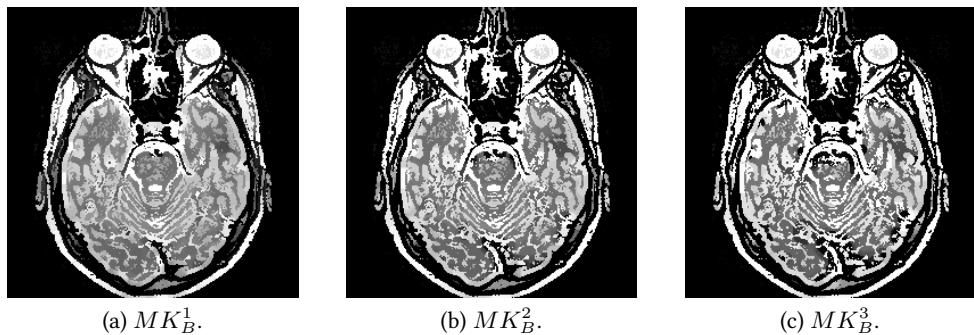


Figure 12.6: Kramer and Bruckner multiscale decomposition, for  $r = 5$ .



```

sskb= cell (1, k+1);
2 sskb {1, 1} = A;
r = 5;
4 for i=2:k+1
    sskb {1, i}=kb(sskb {1, i-1}, r);
6 end

```



```

function K = kb(I, r)
2 % Kramer and Bruckner iterative filter
% I: originale image
4 % r: size of neighborhood
%
6 % return filtered image
se = strel ('disk', r);
8 D = imdilate(I, se);
E = imerode(I, se);
10 difbool=double((D-I)<(I-E));
12 K = D.* difbool+E.*(1-difbool);

```



# \* 13 Introduction to tomographic reconstruction

## 13.1 X ray tomography

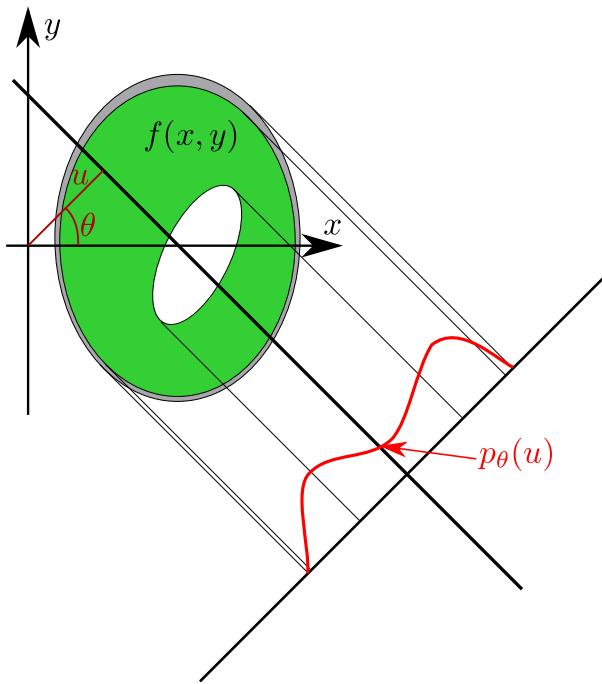


Figure 13.1: Radon transform notations.

An X-ray beam (considered as monochromatic) going through objects is attenuated, following a logarithmic law ( $f$  is the linear attenuation,  $v$  an elementary volume,  $I$  the intensity of the beam entering the volume  $dv$  and  $I + dI$  the intensity exiting the volume):

$$\log \frac{I + dI}{I} = -f dv$$

By integrating this formula on a line  $D$  (direction of the beam), yields:

$$I = I_0 \exp \left( \int_D f(x, y) dv \right)$$

where  $f(x, y)$  is the attenuation at point  $(x, y)$ . This principle is used in scanners, scintigraphy, PET...

## 13.2

# Acquisition simulation

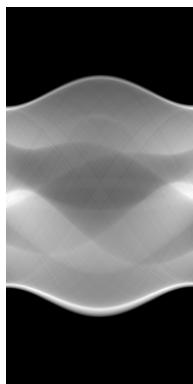
This first exercise will allow us to simulate the projections. The mathematical operator behing this is the Radon transform.



- Generate a “phantom” image of size  $256 \times 256$ . Display it. Notice that any image may be good for testing the simulation and reconstruction via backprojection.
- To simulate the attenation process, considere the sum of all gray levels of all pixels. Angles of projections will be between 0 and 180 degrees with a unit step. The result is presented as a sinogram  $p_\theta(u)$  (see Fig. 13.2), with  $u$  the length (in ordinates), and  $\theta$  the projection angle (in abscissa).



Use the MATLAB® function `phantom`.



(a) Sinogram.



(b) Phantom image.

Figure 13.2: Simulated sinogram of the phantom image.

## 13.3

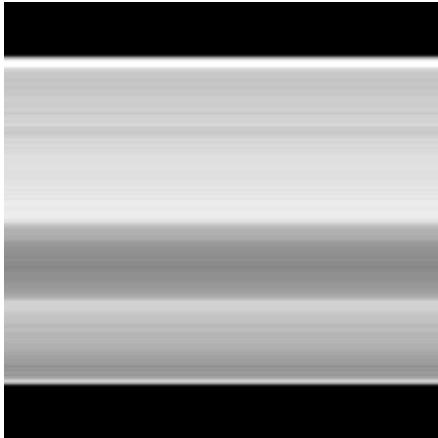
# Backprojection algorithm

A non exact reconstruction is the backprojection operator  $B$  (it is not the inverse Radon transform). It attributes the value  $p_\theta(u)$  to each point of the projection line that gave this attenation, and sum up all contributions from all projections (at the different angles).

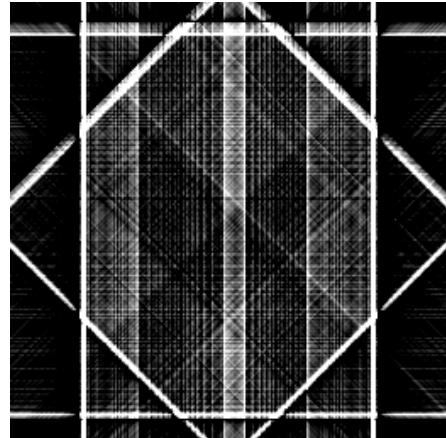
$$B[p](x, y) = \int_0^\pi p_\theta(x \cos \theta + y \sin \theta) d\theta$$

A simple (but slow) method is described to compute the backprojection.  $p_\theta$  is the attenuation vector for a given angle  $\theta$ .

- Use a command to replicate a matrix to obtain the contribution of each line  $D$ , like on Fig. 13.3a.



(a) Contribution of a line  $D$  before rotation.



(b) Reconstruction after projection every 45 degrees (4 projections).

Figure 13.3: Backprojection

- Each contribution line  $D$  should be turned of a certain angle to be added to the overall contribution. This backprojection is really fuzzy compared to the original phantom image (see Fig. 13.3b).



The function `repmat` replicates a matrix a certain number of times.

## 13.4

## Filtered backprojection

It can be shown that backprojection is in fact the convolution of  $f$  by a filter. A solution would be to make a deconvolution in the Fourier domain. The solution used here is the filtered backprojection.

Numerous filters can be used. Among them, the Ram-Lak filter  $RL$  is approximated by :

$$\begin{aligned} k \in [-B; B] \subset \mathbb{N}, RL(k) &= \frac{\pi}{4} \text{ if } k = 0 \\ &= 0 \text{ if } |k| \text{ is even} \\ &= \frac{-1}{\pi k^2} \text{ if } |k| \text{ is odd} \end{aligned}$$



- Write a function that generates this filter (see prototype). The parameter *width* is the number of points of the resulting vector.
- Modify the backprojection algorithm to filter (by convolution of the projection vector) each contribution line before the summation. Observe the improved result.



- Define a function with prototype:  
`function ramlak = RamLak(width)`.
- MATLAB has builtin functions `radon` and `iradon` for projection and filtered backprojection algorithms. Test them.



## 13.5. MATLAB correction



### 13.5.1 Acquisition simulation

The MATLAB built-in function is used to generate a phantom image.



```
% phantom image generation
2 I = phantom();

4 %% Projection with an angular step of 1 degree
angle = 1;
6 theta = 0:angle :180;
S=simuProjection(I, theta);
8 imshow(S, []);
```

The simulation of the projection is simply an addition of all gray-levels of the pixels, after rotating the image in order to simulate the rotation of the object (or of the sensor). See Fig.13.3a



```
function S=simuProjection(I, theta)
2 % simulation of the generation of a sinogram
% I : original image (phantom for example)
4 % theta: angles of projection
taille =size(I);
6 S=zeros( taille (2),length(theta));

8 for i=1:length(theta)-1,
image1=imrotate(I, theta(i), 'bilinear', 'crop');
10
S (:, i)=sum(image1');
12 end
```

### 13.5.2 Backprojection algorithm

The backprojection algorithm will sum-up all the contributions of each projection.



```
function R=backprojection(P, theta, filtre )
2 % Backprojection of a projected image P,
% at all angles 'theta'
4 % filtre : bool, applies filtering if True
```



```

N = size(P,1);
6 R = zeros(N);

8 % in case of filtered back-projection
h = RamLak(31);

10
% loops over all angles
12 for i=1:length(theta),
    proj = P(:, i);

14
% filtered back-projection
16 if filtre ==1
    proj = conv(proj, h, 'same');
18 end

20 proj2 = repmat(proj, 1, N);
proj2 = imrotate(proj2, -theta(i), 'bilinear', 'crop');

22 R = R + proj2;
24 end

```

The results is better in the case of a filtered backprojection. The RamLak function is provided and illustrated in Fig.13.4.



```

function [ramlak] = RamLak(width)
2 % Ramlak filter of size width
% width must be odd
4 k=-width:1:width;

6 for indice = 1:length(k);
    if (k(indice)==0) % valeur du centre
        ramlak(indice)=pi/4;
    elseif (mod(k(indice),2)==1) % indices pairs
        ramlak(indice)=-1/(pi*k(indice)^2);
    else % indices impairs
        ramlak(indice)=0;
    end
14 end

```

The reconstruction of the original image is obtained by the following code:



```

%% reconstruction: simple back-projection
2 R1=backprojection(S, theta, 0);

```

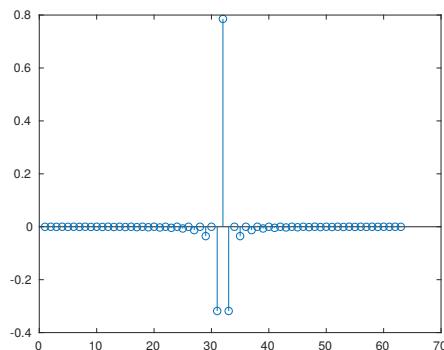


Figure 13.4: RamLak function.

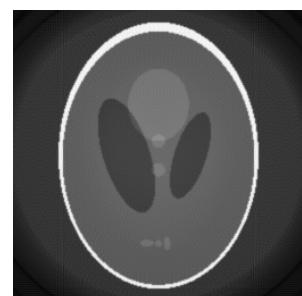
```
4 imshow(R1, []);  
6 %% Filtered back-projection  
R2=backprojection(S, theta, 1);  
8 imshow(R2, []);  
  
10 %% \matlabregistered {} built-in functions  
s=radon(I, theta);  
12 imshow(s, []);  
  
14 r=iradon(s, theta);  
figure();  
16 imshow(r, []);
```



(a) Original phantom image.



(b) Unfiltered backprojection.



(c) Filtered backprojection.

Figure 13.5: Reconstruction by backprojection.



## **Part II Mathematical Morphology**



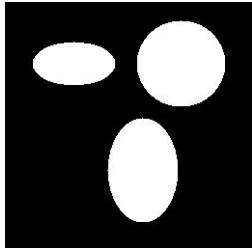
## ★ 14 Binary Mathematical Morphology

The objective of this tutorial is to process binary images with the elementary operators of mathematical morphology. More particularly, different image transformations, based on the morphological reconstruction, will be studied (closing holes, removing small objects...).

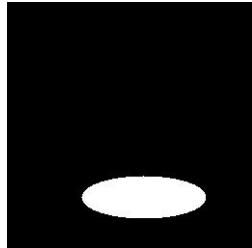
The different transformations will be applied on the following images Fig. 14.1:

Figure 14.1: Images to use for this tutorial.

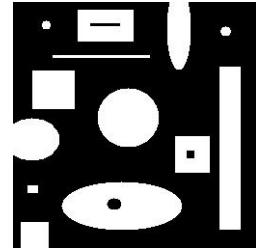
(a)  $I_1$ .



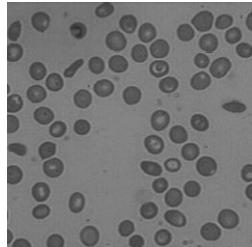
(b)  $M$ .



(c)  $I_2$ .



(d) Cells.



### 14.1

## Introduction to mathematical morphology

Mathematical morphology started in the 1960s with Serra and Matheron [?]. It is based on Minkowski addition of sets. The main operators are erosion and dilation, and by composition, opening and closing. More informations can be found in [?].

The erosion of a binary set  $A$  by the structuring element  $B$  is defined by:

$$\varepsilon_B(A) = A \ominus B = \{z \in A | B_z \subseteq A\} \quad (14.1)$$

where,  $B_z = \{b + z | b \in B\}$ .

The dilation can be obtained by:

$$\delta_B(A) = A \oplus B = \{z \in A | (B^s)_z \cap A \neq \emptyset\} \quad (14.2)$$

where  $B^s = \{x \in E | -x \in B\}$  is the symmetric of  $B$ .

By composition, the opening is defined as:  $A \circ B = (A \ominus B) \oplus B$ . The closing is defined as:  $A \bullet B = (A \oplus B) \ominus B$ .

## 14.2 Elementary operators



Test the functions of dilation, erosion, opening and closing on the image  $I_2$  by varying:

1. the shape of the structuring element,
2. the size of the structuring element.



The MATLAB® functions are `imdilate`, `imerode`, `imopen` and `imclose`. The function `strel` creates a structuring element.

## 14.3 Morphological reconstruction

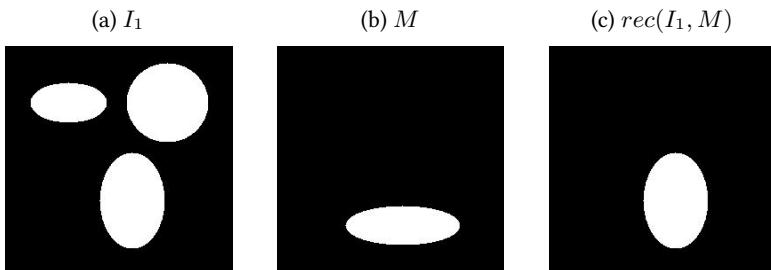
The operator of morphological reconstruction  $\rho$  is very powerful and largely used for practical applications. The principle is very simple. We consider two binary images:  $I_1$  (the studied binary image) and  $M$  (the marker image). The objective is to reconstruct the elements of  $I_1$  marked by  $M$  as illustrated in the Fig. 14.2.

To do this, we iteratively dilate the marker  $M$  while being included in  $I_1$  (see Eq.14.3). In order to guarantee this inclusion in  $A$ , we keep from each dilated set its intersection with  $I_1$ . The algorithm is stopped when the process dilation-intersection is equal to the identity transformation (convergence).

Let  $\delta_I^c(M) = \delta_{B_1}(M) \cap I$  be the dilation of the marker set  $M$  constrained to the set  $I$ . Then, the morphological reconstruction is defined as:

$$\rho_I(M) = \lim_{n \rightarrow \infty} \underbrace{\delta_I \circ \dots \circ \delta_I(M)}_{n \text{ times}} \quad (14.3)$$

Figure 14.2: Illustration of morphological reconstruction of  $I_1$  by  $M$ .



**Data:** image  $I$  and marker  $M$

**Result:** reconstructed image  $rec(I, M)$

```

 $r = area(M);$ 
 $s = 0;$ 
while  $r \neq s$  do
     $| \quad s = r;$ 
     $| \quad M = I \cap (M \oplus B_1);$ 
     $| \quad r = area(M);$ 
end
 $rec(I, M) = M;$ 

```

**Algorithm 5:** The algorithm of this morphological reconstruction



1. Implement the algorithm.
2. Test this operator with the images  $I_1$  and  $M$ .



The function `bwarea` evaluates the area of a 2D binary object.

## 14.4

## Operators by reconstruction



Using the reconstruction operator, implement the 3 following transformations:

1. removing the border objects,
2. removing the small objects,

3. closing the object holes.

Test these operators on the image  $I_2$ .



The morphological reconstruction function to use is `imreconstruct`.

## 14.5

## Cleaning of the image of cells



1. Threshold the image of cells (Fig. 14.1d).
2. Process the resulting binary image with the 3 cleaning processes of the previous question.



## 14.6. Matlab correction



Images given as examples are stored as binary images. The structuring elements used, square, line and disk, are defined with the MATLAB® function `strel`, and some parameters (of size and shape).

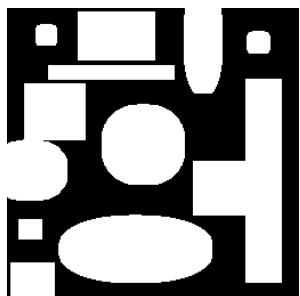


```
% read image
2 B=imread('B.bmp');
% display image
4 figure ; imshow(B,[]); title ('Original image');

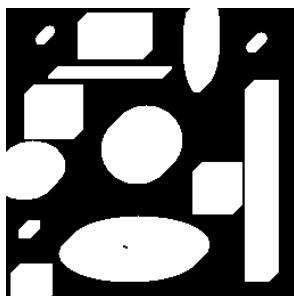
6 % Create structuring elements
    se1 = strel ('square' ,11) ;      % 15-by-15 square
8 se2 = strel ('line' ,11,45) ;     % line , length 15, angle 45 degrees
se3 = strel ('disk' ,5) ;          % disk , radius 15
```

### 14.6.1 Dilation

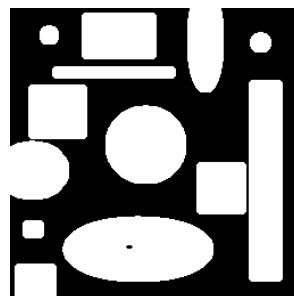
The dilation operation is illustrated in Fig.14.3.



(a) Square structuring element.



(b) Line structuring element.



(c) Disk structuring element.

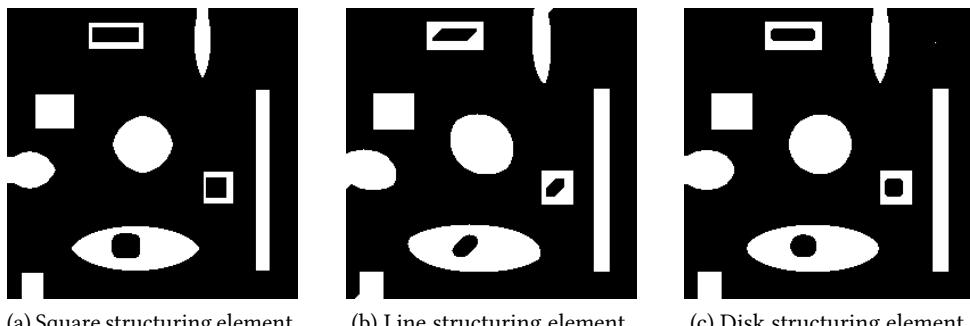
Figure 14.3: Dilation with different structuring elements.



```
1 figure ;
% dilation
3 dilateSquare = imdilate(B, se1);
dilateLine = imdilate(B, se2);
5 dilateDisk = imdilate(B, se3);
subplot (4,3,1) ; imshow(dilateSquare); title (' dilation , square');
7 subplot (4,3,2) ; imshow(dilateLine); title (' dilation , segment');
subplot (4,3,3) ; imshow(dilateDisk); title (' dilation , disk');
```

## 14.6.2 Erosion

The dual operation of dilation is the erosion, illustrated in Fig.14.4.



(a) Square structuring element. (b) Line structuring element. (c) Disk structuring element.

Figure 14.4: Erosion with different structuring elements.



```
% erosion
2 erodeSquare = imerode(B, se1);
3 erodeLine = imerode(B, se2);
4 erodeDisk = imerode(B, se3);
```

## 14.6.3 Opening and closing

Opening and closing are the combination of erosion and dilation in both orders, see illustration Fig.14.5.



```
% open
2 openSquare = imopen(B, se1);
3 openLine = imopen(B, se2);
4 openDisk = imopen(B, se3);
```



```
% close
2 closeSquare = imclose(B, se1);
3 closeLine = imclose(B, se2);
4 closeDisk = imclose(B, se3);
```

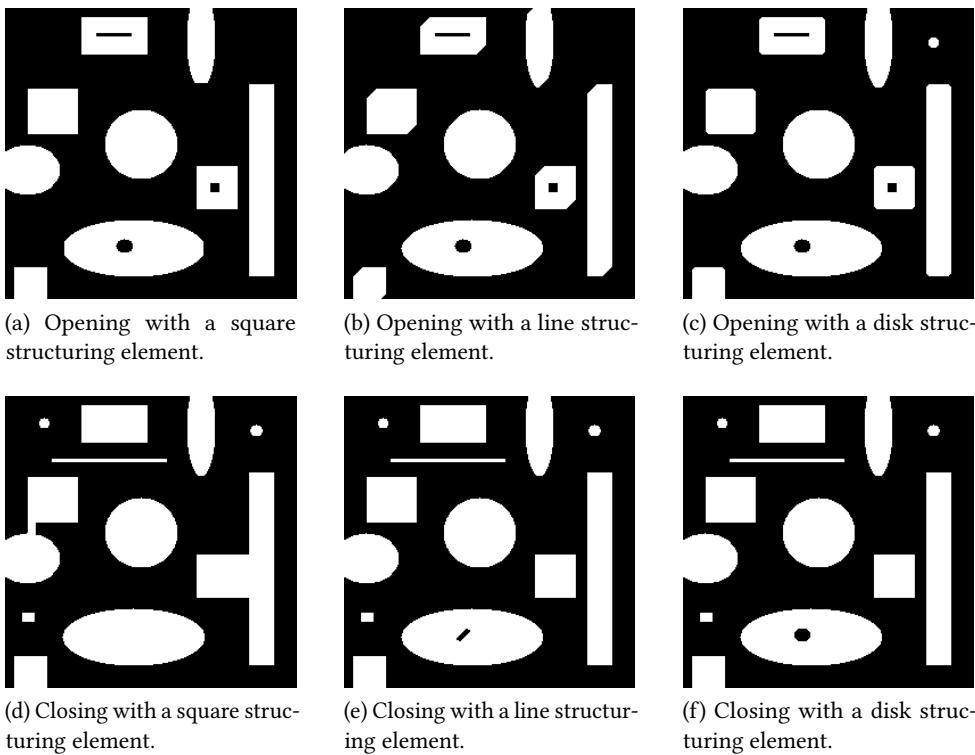


Figure 14.5: Opening and closing with different structuring elements.

The following code is used to illustrate the impact of the size of the structuring element.



```
%  
2 % opening with varying size  
figure;  
4 for i=1:12  
    sedisk(i)=strel('disk',i);  
6     openDisk = imopen(B, sedisk(i));  
    subplot(4,3, i); imshow(openDisk); title(strcat('opening, ', int2str(i)));  
8 end
```

#### 14.6.4 Morphological reconstruction

The morphological reconstruction is employed in a lot of applications, like removing objects touching the borders of the image, or removing small objects. The intersection of two sets is coded as the minimum of binary arrays. The result is illustrated in Fig.14.6.



```
function B=reconstruct(A,M)  
2 M=min(M,A);  
r=bwarea(M);  
4 s=0;  
se=strel('disk',1);  
6 while (r ~= s)  
    s=r;  
    M=min(A, imdilate(M, se));  
    r=bwarea(M);  
10 end  
B=M;
```

#### Remove border objects

In order to remove the objects that may touch the border, a marker of the border is defined. In practice, the extreme pixels are marked, then the reconstruction of the objects by this marker is performed, then removed from the original image. Notice the definition of a border with value 255, as it can handle unsigned 8 bits images (thanks to the `min` function). See Fig.14.7. If  $\mathcal{B}$  represents the border of the image (create an array of the same size as the image, with zeros everywhere and ones at the sides), then this operation is defined by:

$$\text{killBorders}(I) = I \setminus \rho_I(\mathcal{B})$$

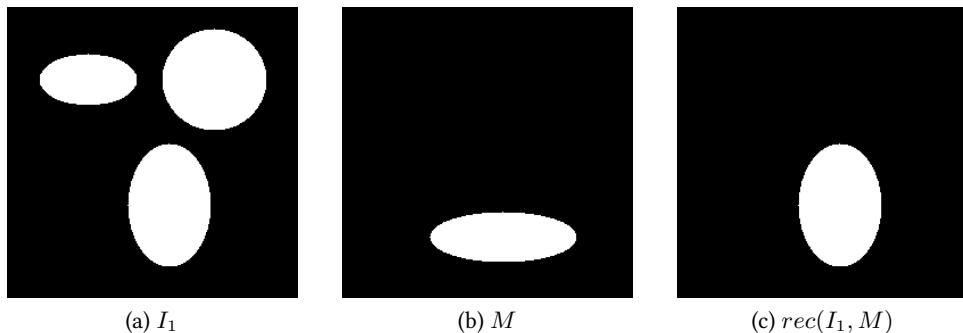


Figure 14.6: Illustration of morphological reconstruction of  $I_1$  by  $M$ .



```

function B=killBorders (A)
2 [m,n]=size(A);
M=zeros(m,n);
4 M (1,:)=255;
M(m,:)=255;
6 M (:,1)=255;
M (:,n)=255;
8 M=reconstruct(A,M);
B=A-M;

```

### Remove small objects

The principal is first to make an erosion with a given structuring element  $se$ , in order to suppress the objects smaller than  $se$ . Then, a reconstruction is performed in order to get the original objects. See Fig.14.7.

It consists in an erosion followed by a reconstruction. The structuring element used in the erosion defines the objects considered as “small”.

$$\text{killSmall}(I) = \rho_I(\varepsilon(I))$$



```

1 function B=killSmall (A,n)
se= strel ('disk' ,n);
3 M=imerode(A,se);
B=reconstruct(A,M);

```

### Close holes

In order to close holes, we work on the complementary set of  $A$  and try to isolate the background (that is supposed to touch the border of the image). See Fig.14.7. The operation is given by the following equation, with  $\mathcal{B}$  the border of image  $I$ , and  $X^C$  denoting the complementary of set  $X$ :

$$\text{removeHoles}(I) = \{\rho_{IC}(\mathcal{B})\}^C$$

```

function B=closeHoles(A)
2 Ac=imcomplement(A);
[m,n]=size(A);
4 M=zeros(m,n);
M(1,:)=255;
6 M(m,:)=255;
M(:,1)=255;
8 M(:,n)=255;
M=reconstruct(Ac,M);
10 B=imcomplement(M);
```

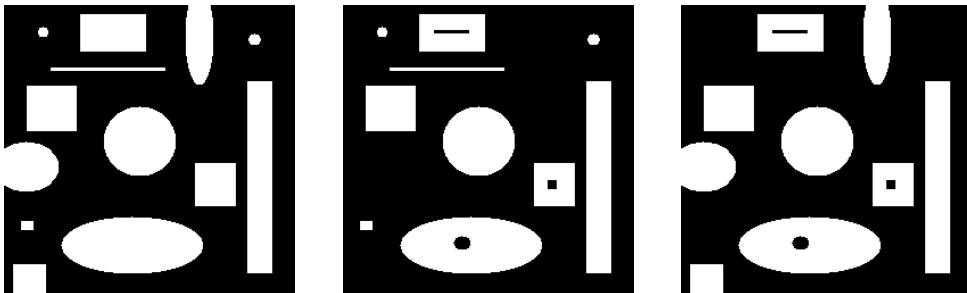


Figure 14.7: Illustration of the use of the morphological reconstruction.

### 14.6.5 Application on cells image

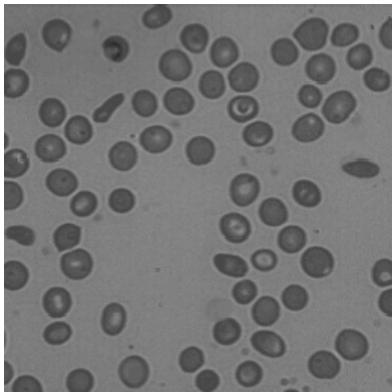
The application on the cell image is quite trivial: after thresholding (binarizing) the original image (the threshold value is chosen experimentally), we can close the holes, remove the small objects and the ones touching the borders (see Fig.14.8).

```

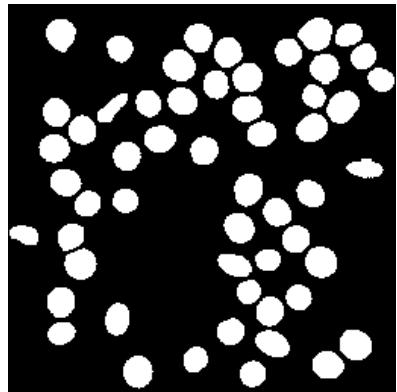
A=imread('cells.bmp');
2 C=A<98;
```



```
B=closeHoles(C);  
4 B=killBorders (B);  
B=killSmall (B,5);
```



(a) Original image of cells.



(b) Segmented image.

Figure 14.8: Segmentation of the image of cells.





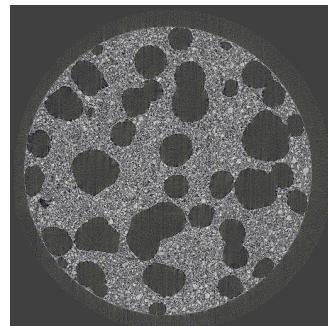
## 15 Morphological Geodesic Filtering

This tutorial aims to test different morphological filters and particularly the geodesic ones (using reconstruction) for gray-level images.

The different processes will be applied on the following images:



(a) Lena



(b) 2-D section of a cement paste  
(X-ray tomography)

### 15.1

## Morphological centre

The morphological centre is an auto-dual filter using a family of operators  $\{\psi_i\}_i$ :

$$C(f) = (f \vee (\wedge\{\psi_i(f)\})) \wedge (\vee\{\psi_i(f)\}) \quad (15.1)$$



- Implement this transformation with the family  $\{\gamma\phi\gamma, \phi\gamma\phi\}$  where  $\gamma$  denotes the morphological opening and  $\phi$  the morphological closing.
- Test this operator by varying the size of the structuring element.
- Add a 'salt and pepper' noise to the 'Lena' image and compare the morphological center with the median filtering.
- Try to reduce the noisy image 'cement paste'.

## 15.2 Alternate sequential filters (ASF)

ASF can be defined from a family of openings and closings:

$$N_i(f) = \gamma_i \phi_i \circ \gamma_{i-1} \phi_{i-1} \dots \gamma_2 \phi_2 \circ \gamma_1 \phi_1(f) \quad (15.2)$$

$$M_i(f) = \phi_i \gamma_i \circ \phi_{i-1} \gamma_{i-1} \dots \phi_2 \gamma_2 \circ \phi_1 \gamma_1(f) \quad (15.3)$$

where  $\gamma_k$  (resp.  $\phi_k$ ) denotes the opening (resp. closing) with a structuring element of size  $k$ .



- Implement these two operators.
- Test these transformations on the noisy image by varying the parameter  $i$  of the filter.

## 15.3 Reconstruction filters

The geodesic dilation of size 1 and  $n$  are respectively defined as:

$$\delta_f(g) = \wedge(\delta_{B_1}(g), f) \quad (15.4)$$

$$\delta_f^n(g) = \delta_f(\delta_f \dots (\delta_f(g))) \quad (15.5)$$

where  $\delta_{B_1}$  denotes the morphological dilation with a disk of radius 1 as structuring element. The opening ( $\gamma_k^{rec}(f)$ ) and closing ( $\phi_k^{rec}(f)$ ) by reconstruction are then defined as:

$$\gamma_k^{rec}(f) = \vee\{\delta_f^n(\epsilon_{B_k}(f)), n > 0\} \quad (15.6)$$

$$\phi_k^{rec}(f) = M - \gamma_k^{rec}(M - f) \quad (15.7)$$

where  $\epsilon_{B_k}$  denotes the morphological erosion with a disk  $B$  of radius  $k$  as structuring element.



- Implement these two operators  $\gamma_k^{rec}(f)$  and  $\phi_k^{rec}(f)$ .
- Test these transformations by varying the parameter  $k$  of the filter.
- Implement and test (on the noisy image) the filters of morphological center and ASF using the geodesic operators. Compare with the classical ones.



## 15.4. Matlab correction



### 15.4.1 Morphological center

The noisy image is obtained with the function imnoise.



```
A=double(imread('lena512.bmp'));
2 A=255*imnoise(A/255,'salt & pepper', 0.04);
```

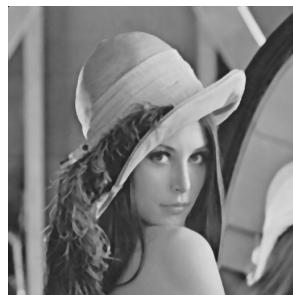
Following the definition, the morphological center is obtained with the following code, and illustrated in Fig.15.1:



```
n=1;
2 se=strel('disk',n);
coc=imclose(imopen(imclose(A,se),se),se);
4 oco=imopen(imclose(imopen(A,se),se),se);
cMin=min(oco,coc);
6 cMax=max(oco,coc);
B=min(max(A,cMin),cMax);
```



(a) Noisy image.



(b) Median filter of size  $5 \times 5$ .



(c) Morphological center.

Figure 15.1: Morphological center compared to the classical median filter.

### 15.4.2 Alternate sequential filters

The order of these filters are often chosen empirically. The results are illustrated in Fig.15.2.



```

1 function F = asf_n(I, order)
% Alternate Sequential Filter beginning by a closing
3 % I: original image
% order: order of the filter (number of loops)
5 F = I;
for i=1:order
7     se = strel ('disk', i);
    F = imclose(imopen(F, se), se);
9 end

```



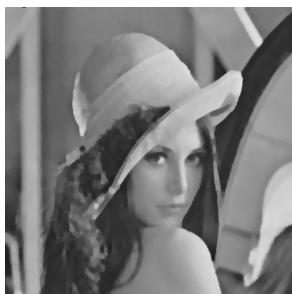
```

1 function F = asf_m(I, order)
% Alternate Sequential Filter beginning by an opening
3 % I: original image
% order: order of the filter (number of loops)
5 F = I;
for i=1:order
7     se = strel ('disk', i);
    F = imopen(imclose(F, se), se);
9 end

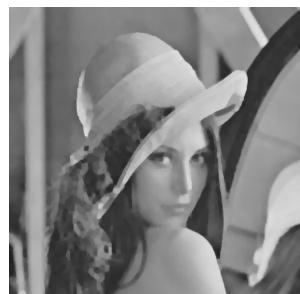
```



(a) Original image.



(b) ASF of order 3, starting with a closing operation (denoted N).



(c) ASF of order 3, starting with an opening operation (denoted M).

Figure 15.2: Alternate Sequential Filters compared to original image.

### 15.4.3 Geodesic reconstruction filters

These two functions are simply implemented using erosion and reconstruction operators. Notice the duality property, that is used to code closerec. In this example, 8 bits images (unsigned) are considered, and the results are illustrated in Fig.15.3.



```
function D=openrec(A,n)
2 B=imerode(A,strel('disk',n));
D=reconstruct(A,B);
```



```
1 function D=closerec(A,n)
% closerec and openrec are dual operators
3 D = 255-openrec(255-A, n);
```



(a) Original image.



(b) Opening by reconstruction.



(c) Closing by reconstruction.

Figure 15.3: Opening and closing by reconstruction.

#### 15.4.4 ASF by reconstruction

This is an example of a 3rd order alternate sequential filter, illustrated in Fig.15.4.



```
1 n1=1;
n2=2;
3 n3=3;
co1=closerec(openrec(A,n1),n1);
5 co2=closerec(openrec(co1,n2),n2);
co3=closerec(openrec(co2,n3),n3);
```

#### 15.4.5 Morphological center by reconstruction

Morphological center by reconstruction replaces the opening and closing operations by their equivalent by reconstruction (see Fig.15.5).



(a) Original image.



(b) ASF by reconstruction.



(c) Noisy image.

Figure 15.4: Alternate Sequential Filtering by reconstruction.



```
n=1;  
2 coc=closerec(openrec(closerec(A,n),n),n);  
oco=openrec(closerec(openrec(A,n),n),n);  
4 cMin=min(oco,coc);  
cMax=max(oco,coc);  
6 B=min(max(A,cMin),cMax);
```



(a) Original image.



(b) Center by reconstruction.



(c) Noisy image.

Figure 15.5: Morphological center by reconstruction.



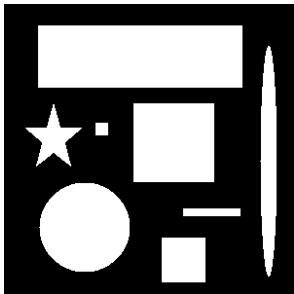
## 16 Morphological Attribute Filtering

This tutorial aims to test the attribute morphological filters for gray-level images. The objective of such filters is to remove the connected components of the level sets which do not satisfy a specific criterion (area, elongation, convexity...).

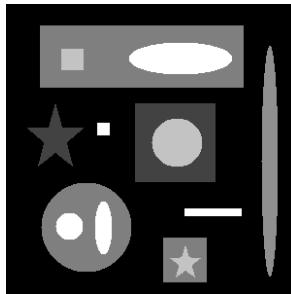
Fig.16.1 are presented some results of attribute filtering applied on the grayscale image.

Figure 16.1: Attribute filtering examples.

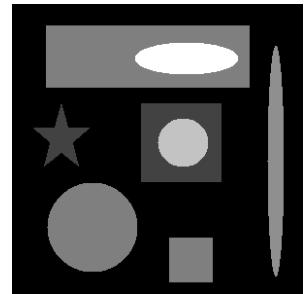
(a) Binary image.



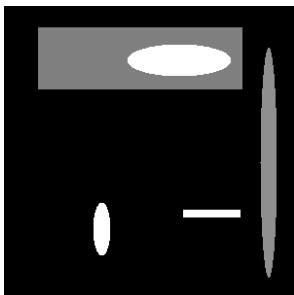
(b) Grayscale image.



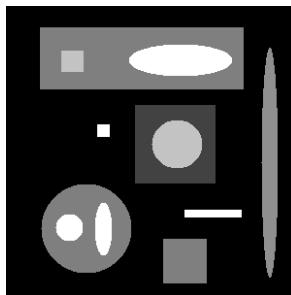
(c) Filtering by area.



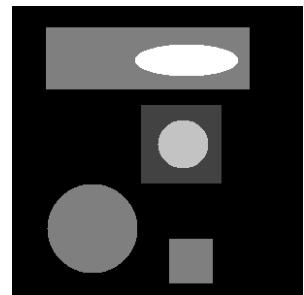
(d) Filtering by elongation.



(e) Filtering by convexity.



(f) Filtering objects larger than a square.



A *binary connected opening*  $\Gamma_x$  transforms a binary image  $f$  given a pixel  $x$ , by keeping the connected component that contains  $x$  and removing all the others.

*Binary trivial opening*  $\Gamma_T$  operates on a given connected component  $X$  according to an increasing criterion  $T$  applied to the connected component. If the criterion is satisfied, the connected component is preserved, otherwise it is removed according to:

$$\Gamma_T(X) = \begin{cases} X & \text{if } T(X) = \text{true} \\ \emptyset & \text{if } T(X) = \text{false} \end{cases} \quad (16.1)$$

In general, one or more features of the connected component, on which the filter is applied, are compared to a given threshold defined by the rule.

*Binary attribute opening*  $\Gamma^T$ , given an increasing criterion  $T$ , is defined as a binary trivial opening applied on all the connected components of  $F$ . This can be formally represented as:

$$\Gamma^T(F) = \bigcup_{x \in F} \Gamma_T(\Gamma_x(F)) \quad (16.2)$$

If the criterion  $T$  evaluated in the transformation is not increasing, e.g., when the computed attribute is not dependent itself on the scale of the regions (e.g. shape factor, orientation, etc.), the transformation also becomes not increasing. Even if the increasingness property is not fulfilled, the filter remains idempotent and anti-extensive. For this reason, the transformation based on a non-increasing criterion is not an opening, but a thinning. In this case, one speaks about a *binary attribute thinning*.

Attribute openings and thinnings introduced for the binary case can be extended to grayscale images employing the threshold decomposition principle. A grayscale image can be represented by thresholding the original image at each of its graylevels. Then, the binary attribute opening can be applied to each binary image and the *grayscale attribute opening* is given by the maximum of the results of the filtering for each pixel and can be mathematically represented as:

$$\gamma^T(f)(x) = \max\{k; x \in \Gamma^T(Th_k(f))\} \quad (16.3)$$

where  $Th_k(f)$  is the binary image obtained by thresholding  $f$  at graylevel  $k$ .

The extension to grayscale of the binary attribute thinning is not straightforward and not unique. For example, a possible definition of a *grayscale attribute thinning* can be analogously defined as Equation (3). However, other definitions are possible, according to the filtering rule considered in the analysis (max, min, subtractive, direct...). Again, if the criterion  $T$  is increasing (i.e. the transformation is an opening), all the strategies lead to the same result.

Attribute openings are a family of operators that also includes openings by reconstruction. If we consider a binary image with a connected component  $X$  and the increasing criterion "the size of the largest square enclosed by  $X$  must be greater than  $\lambda$ ", the result of the attribute opening is the same as applying an opening by reconstruction with a squared structuring element of size  $\lambda$ .

## 16.1 Binary attribute opening/thinning

We are going to implement and test some binary attribute filters according to different criteria (area, elongation, convexity).



- Regarding the area criterion, implement the associated binary attribute opening. Test on the binary image.
- Regarding the elongation criterion, implement the associated binary attribute thinning (non-increasing criterion). Test on the binary image.



Use the function `bwpropfilt`.

## 16.2

# Grayscale attribute opening/thinning

We are going to extend the binary filters to grayscale images.



- Implement the decomposition operator into binary sections.
- Conversely, implement the reconstruction operator from binary sections.
- Implement and test the grayscale attribute filters according to the following criteria: area, elongation, convexity.



## 16.3. Matlab correction



### 16.3.1 Binary attribute filtering

If  $I$  is a binary image, the different attribute are proposed in the following code (filtering small squares –  $25 \times 25$ , small objects, by elongation or convexity, respectively).



```

1 S= imreconstruct(imopen(I, strel ('square',25)), I));
2 A= bwareaopen(I, 1000);
E= bwpropfilt(I, 'eccentricity', [0.75 1]);
4 C= bwpropfilt(I, 'solidity', [0.75 1]);

```

### 16.3.2 Grayscale filtering

The grayscale filtering is the previous binary filtering process applied to all level-sets of the original image. The image is first decomposed into the level-sets.



```

A = double(imread('toy.png'));
2 A = A (:,:,1);

4 %% IMAGE DECOMPOSITION
[m,n] = size(A);
6 levelSets_init = false(m,n,256);
for s=0:255
8 levelSets_init (:,:, s+1) = A>=s;
end

```

Then, for each level, the binary set is filtered by some attribute, and the resulting image is reconstructed by taking the maximum value on all levels.



```

1 %% ATTRIBUTE FILTERING
levelSets_res1 = zeros(m,n,256);
3 levelSets_res2 = zeros(m,n,256);
levelSets_res3 = zeros(m,n,256);
5 levelSets_res4 = zeros(m,n,256);
for s=0:255
7 levelSets_res1 (:,:, s+1) = s*(imreconstruct(imopen(levelSets_init (:,:, s+1),
    ↪ strel ('square',25)), levelSets_init (:,:, s+1)));
levelSets_res2 (:,:, s+1) = s*bwareaopen(levelSets_init (:,:, s+1),1000);

```



```
9 levelSets_res3 (,:, s+1) = s* bwpropfilt( levelSets_init (,:, s+1), ' eccentricity '
    ↪ ,[0.75 1]);
10 levelSets_res4 (,:, s+1) = s* bwpropfilt( levelSets_init (,:, s+1), ' solidity '
    ↪ ,[0.75 1]);
11 end

13 %% IMAGE RECONSTRUCTION
14 B1 = max(levelSets_res1 ,[],3) ;
15 B2 = max(levelSets_res2 ,[],3) ;
16 B3 = max(levelSets_res3 ,[],3) ;
17 B4 = max(levelSets_res4 ,[],3) ;
```

Results are illustrated in Fig.16.1.





# 17 Morphological skeletonization

This tutorial aims to skeletonize objects with specific tools from mathematical morphology (thinning, maximum ball...).

The different processes will be applied on the following image:



## 17.1

### Hit-or-miss transform

The hit-or-miss transformation enables specific pixel configurations to be detected. Based on a pair of disjoint structuring elements  $T = (T^1, T^2)$ , this transformation is defined as:

$$\eta_T(X) = \{x, T_x^1 \subseteq X, T_x^2 \subseteq X^c\} \quad (17.1)$$

$$= \epsilon_{T^1}(X) \cap \epsilon_{T^2}(X^c) \quad (17.2)$$

where  $\epsilon_B(X)$  denotes the erosion of  $X$  using the structuring element  $B$ .



1. Implement the hit-or-miss transform.
2. Test this operator with the following pair of disjoint structuring elements:

$$T^1 = \begin{pmatrix} +1 & +1 & +1 \\ 0 & +1 & 0 \\ -1 & -1 & -1 \end{pmatrix}, \quad T^2 = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

where the points with  $+1$  (resp.  $-1$ ) belong to  $T^1$  (resp.  $T^2$ ).

## 17.2 Thinning and thickening

Using the hit-or-miss transform, it is possible to make a thinning or thickening of a binary object in the following way:

$$\theta_T(X) = X \setminus \eta_T(X) \quad (17.3)$$

$$\chi_T(X) = X \cup \eta_T(X^c) \quad (17.4)$$

These two operators are dual in the sense that  $\theta_T(X) = (\chi_T(X^c))^c$ .



1. Implement these two transformations.
2. Test these operators with the previous pair of structuring elements.

## 17.3 Topological skeleton

By using the two following pairs of structuring elements with their rotations ( $90^\circ$ ) in an iterative way (8 configurations are thus defined), the thinning process converges to a resulting object which is homothetic (topologically equivalent) to the initial object.

+1	+1	+1	0	+1	0
0	+1	0	+1	+1	-1
-1	-1	-1	0	-1	-1



1. Implement this transformation (the convergence has to be satisfied).
2. Test this operator and comment.

## 17.4 Morphological skeleton

A ball  $B_n(x)$  with center  $x$  and radius  $n$  is maximum with respect to the set  $X$  if there exists neither indice  $k$  nor centre  $y$  such that:

$$B_n(x) \subseteq B_k(y) \subseteq X$$

In this way, the morphological skeleton of a set  $X$  is constituted by all the centers of maximum balls. Mathematically, it is defined as:

$$S(X) = \bigcup_r \epsilon_{B_r(0)}(X) \setminus \gamma_{B_1(0)}(\epsilon_{B_r(0)}(X)) \quad (17.5)$$



1. Implement this transformation.
2. Test this operator and compare it with the topological skeleton.



## 17.5. Matlab correction



Please notice that when using boolean arrays in matlab, the notations  $1-X$  and  $\sim X$  are equivalent. When using uint8 arrays, verify that the values range into  $[0;1]$ .

### 17.5.1 Hit-or-miss transform

The hit-or-miss transform is illustrated in Fig.17.1.



(a) Original image.



(b) Hit or miss result (intensities are inverted).

Figure 17.1: Hit or miss illustration for a given orientation.



```

function B=hitormiss(X,T)
% X is the binary image (values 0 or 1)
% T is the structuring element
%
T1=(T == 1);
T2=(T == -1);
B=min(imerode(X,T1),imerode(~X,T2));

```

### 17.5.2 Thinning and thickening

Thinning and thickening are dual operations. The second function could make a call to the first one. The code elementary follows the definition. The illustration is presented in Fig.17.2.



```

1 function B=elementary_thinning(X,T)
% thinning function
3 % X: binary image
% T: structuring element
5 B=X-hitmiss(X,T);

```



```

1 function B=elementary_thickening(X,T)
% thickening function
3 % X: binary image
% T: structuring element
5 B = min(X, ~hitmiss (X,T));
% equivalent notation:
7 % B = X-hitmiss(X,T);

```



(a) Thinning.



(b) Thickening.

Figure 17.2: Thinning and thickening.

### 17.5.3 Skeletons

The pairs of structuring elements are defined like this, in the 8 directions:



```

1 TT=cell (1,8) ;
TT {1}=[-1,-1,-1;0,1,0;1,1,1];
3 TT {2}=[0,-1,-1;1,1,-1;0,1,0];
TT {3}=[1,0,-1;1,1,-1;1,0,-1];
5 TT {4}=[0,1,0;1,1,-1;0,-1,-1];
TT {5}=[1,1,1;0,1,0;-1,-1,-1];

```



```

7 TT {6}=[0,1,0;-1,1,1;-1,-1,0];
TT {7}=[-1,0,1;-1,1,1;-1,0,1];
9 TT {8}=[-1,-1,0;-1,1,1;0,1,0];

```

Thus, the thinning operation is coded as:



```

1 function B=thinning(A,TT)
B=A;
3 for i=1:length(TT)
    B=B-hitormiss(B,TT{i});
5 end

```

The topological skeleton is the iteration of the thinning with structuring elements in all 8 directions. It has the property of preserving the topology of the discrete structures, contrary to the morphological skeleton (see Fig.17.3. The morphological skeleton does not preserve the connexity of the branches, but it can be used to reconstruct the original image. Pay attention to the construction of structuring elements, which should be homothetic ( $B_r = \underbrace{B_1 \oplus \dots \oplus B_1}_{r \text{ times}}$ ).



```

1 % topological skeleton function
% X: binary image
3 % T: structuring element
function B=topological_skeleton(X,TT)
5 B2=X;
B=-B2;
7 while (isequal(B,B2)~=1)
    B=B2;
9 B2=thinning(B,T);
end

```



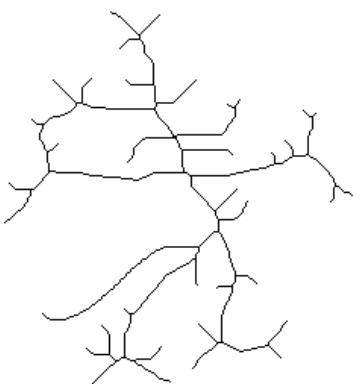
```

function S=morphological_skeleton(X)
2 % morphological skeleton function
% X: binary image
4 S=zeros( size(X));
strel_size =0; % size of structuring element
6 pred=true;
while pred
8     strel_size = strel_size +1;

```



```
E=imerode(X,strel('disk', strel_size));
10 if sum(E(:))==0
    pred=false;
12 end
S=max(S,E-imopen(E,strel('disk',1)));
14 end
```



(a) Topological skeleton.



(b) Morphological skeleton.

Figure 17.3: Skeletons. The topology is not preserved in the morphological skeleton, but it can be used to reconstruct the original image. The intensities are inverted in order to facilitate the display.



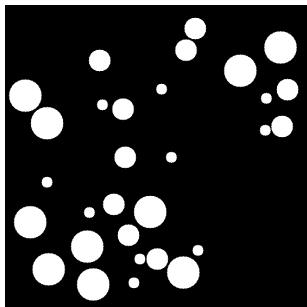
# ★ 18 Granulometry

This tutorial aims to compute specific image measurements on digital images. It consists in determining the size distribution of the 'particles' included in the image to be analyzed.

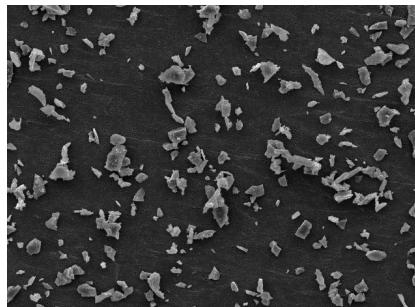
The image measurements will be realized on a simulated image of disks and an image of silicon carbide powder acquired by Scanning Electron Microscopy (SEM, Fig. 18.1).

Figure 18.1: These images are used for computing a granulometry of objects by mathematical morphology.

(a) Simulated image.



(b) Real image of powder, by SEM.



## 18.1

# Morphological granulometry

### 18.1.1 Introduction to mathematical morphology

The tutorial 14 introduces the basic operations of the mathematical morphology, as well as the morphological reconstruction. More informations can be found in [?].

With Matlab, the useful functions are imopen for the morphological opening, imreconstruct for the geodesic reconstruction. For counting the number of objects, one can use bweuler in the case of objects with no hole, or bwlabel for a labelling and counting algorithm.



### 18.1.2 Granulometry



- Load and visualize the image 'simulation' Fig.18.1.
- Generate  $n$  images by applying morphological openings (with reconstruction) to the binary image with the use of homothetic structuring elements of increasing size:  $1 < \dots < n$ .
- Compute the morphological granulometry expressed in terms of surface area density. It consists in calculating the specific surface area of the grains in relation with the size  $n$  of the structuring element.
- Compute the morphological granulometry expressed in terms of number density. It consists in calculating the specific number of grains in relation with the size  $n$  of the structuring element.
- Compare and discuss.



Homothetic structuring element can be defined as

`se = strel ('disk', i, 0)` for disks of size  $i$ .

## 18.2

### Real application



- Load and visualize the image 'powder' of Fig. 18.1b.
- Realize the image segmentation step. It can simply consist in a binarization by a global threshold, followed by hole filling. Noise can also be removed by mathematical morphology opening.
- Compute the morphological granulometry (surface area, number).
- What can you conclude?



## 18.3. Matlab correction



### 18.3.1 Morphological granulometry

The code is straightforward from the definition. It consists on a loop over the different sizes of the structuring element.



```
% read image
2 A=imread('simulation.bmp');
A=logical(double(A)/255);
4
% visualisation
6 figure ;imshow(A);
title ('Original simulated image');
```

Different structuring elements shapes can be used, the most classical one being the disk. In order to suppress small objects, the function imreconstruct is used (see tutorial 14).



```
% maximal radius size
2 N=35;

4 % array of areas and numbers
areas=zeros(N, 1);
6 number=zeros(N,1);
area0=sum(A(:));
8 nbre0=bweuler(A);
% loop over the different sizes
10 for i=0:N
    se = strel ('disk', i, 0); % structuring element
12    C = imopen(A, se); %
    C = imreconstruct(C,A); % suppress small objects
14    areas(i) = sum(C(:))/area0*100; % normalized area
    number(i)=bweuler(C)/nbre0*100;% Euler number
16 end
```

The results are displayed using the following commands, and reproduced in Fig. 18.2. The function `diff` is used to evaluate a discrete derivative.



```
% display the results
2 figure;
```



```

1 subplot(121); plot(0:N,areas,'-xr'); title('Granulometry');
4 hold on; plot(0:N,number,'-xb'); legend('area analysis','number analysis');
% finite difference analysis
6 diff_areas = -diff(areas);
diff_number = -diff(number);
8 subplot(122);
plot(0:N-1,diff_areas,'-xr'); title('Finite differences');
10 hold on; plot(0:N-1,diff_number,'-xb');
legend('area analysis','number analysis');

```

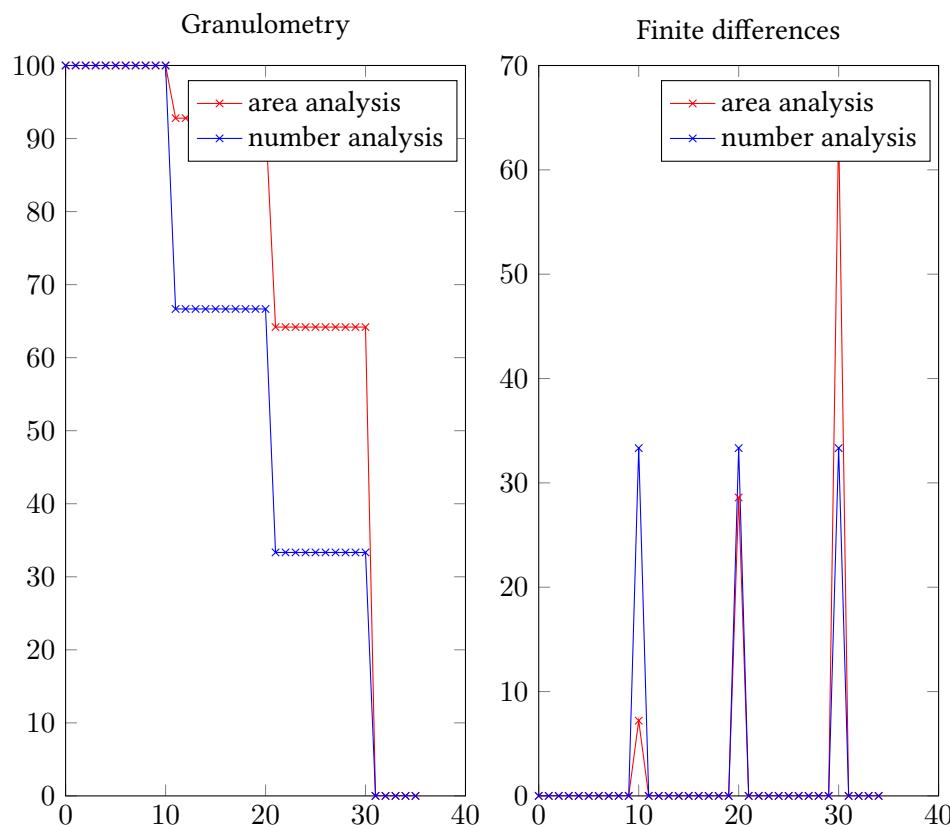


Figure 18.2: Granulometry and finite differences for the synthetic image of disks.

### 18.3.2 Real application

The code is exactly the same as the previous one, taking a binary image as input. The powder image is segmented using a threshold at value 74, and applying some filtering processes (see result in Fig. 18.3).



```
1 B=imread('poudre.bmp');
% threshold
3 imThresh=(B>74);
% fill holes
5 imHoles=imfill(imThresh,'holes');
% suppress small objects
7 se = strel('disk',1);
C = imopen(imHoles,se);
9 imSegmented=imreconstruct(C,imHoles);
% visualisation images
11 figure;
subplot(121);imshow(B,[]);colormap('gray'); title ('Original image of silicium');
13 subplot(122);imshow(imSegmented);colormap('gray'); title ('Segmented image');
```

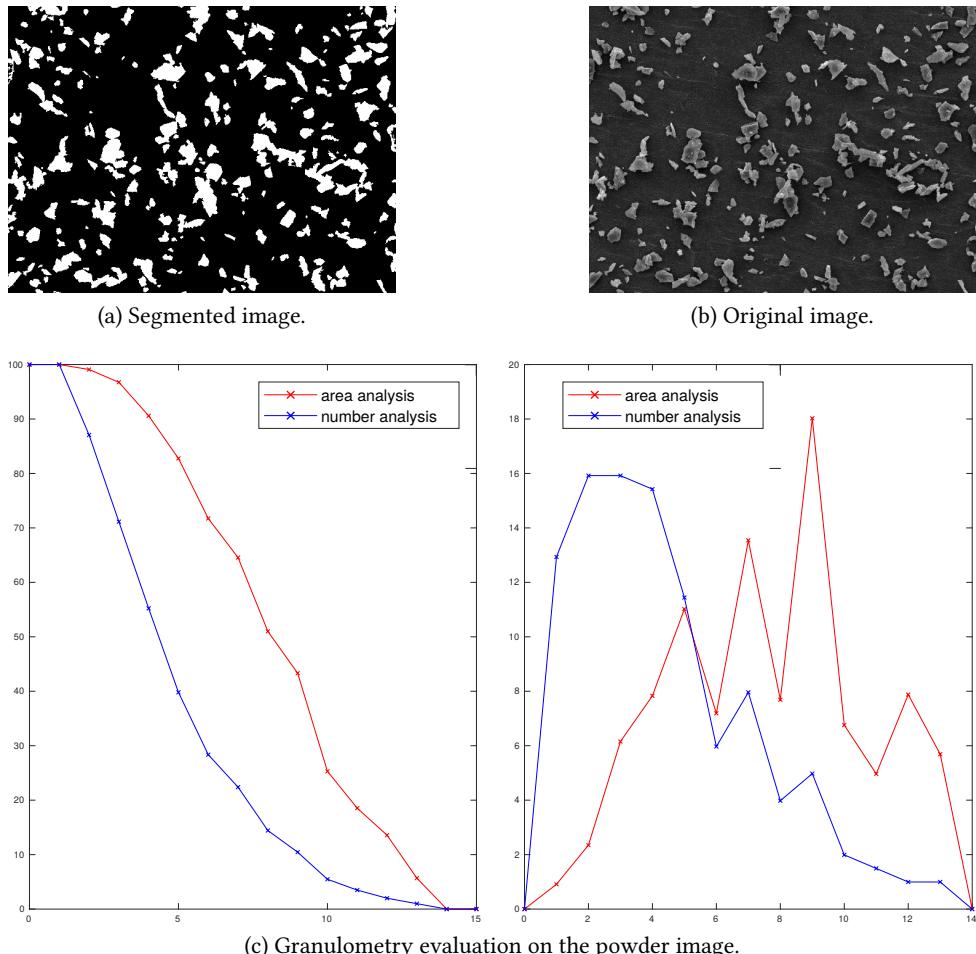


Figure 18.3: Illustration of grain analysis, in size and number, on a powder image.

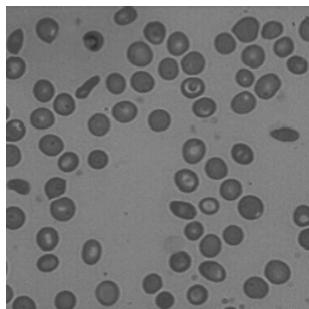
## **Part III    Registration and Segmentation**



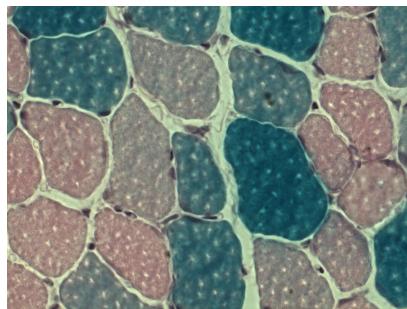
# \* 19 Histogram-based image segmentation

This tutorial aims to implement some image segmentation methods based on histograms (thresholding and “ $k$ -means” clustering).

The different processes will be applied on the following images:



(a) Cells.



(b) Muscle cells, author: Damien Freyssenet, University Jean Monnet, Saint-Etienne, France.

## 19.1 Manual thresholding

The most simple segmentation method is thresholding.



- Visualize the histogram of the grayscale image 'cells'.
- Make the segmentation with a threshold value determined from the image histogram.

## 19.2 k-means clustering

Let  $X = \{x_i\}_{i \in [1; n], n \in \mathbb{N}}$  be a set of observations (the points) in  $\mathbb{R}^d$ . The  $k$ -means clustering consists in partitioning  $X$  in  $k$  ( $k < n$ ) disjoint subsets  $\tilde{S}$  such that:

$$\tilde{S} = \arg \min_{S=\{S_i\}_{i \leq k}} \sum_{i=1}^k \sum_{x_j \in S_i} \|x_j - \mu_i\|^2 \quad (19.1)$$

where  $\mu_i$  is the mean value of the elements in  $S_i$ . The  $k$ -means algorithm is iterative. From a set of  $k$  initial elements  $\{m_i^{(1)}\}_{i \in [1; k]}$  (randomly selected), the algorithm iterates the following ( $t$ ) steps:

- Each element of  $X$  is associated to an element  $m_i$  according to a distance criterion (computation of a Voronoi partition):

$$S_i^{(t)} = \left\{ \mathbf{x}_j : \|\mathbf{x}_j - \mathbf{m}_i^{(t)}\| \leq \|\mathbf{x}_j - \mathbf{m}_{i^*}^{(t)}\|, \forall i^* \in [1; k] \right\} \quad (19.2)$$

- Computation of the new mean values for each class:

$$\mathbf{m}_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{\mathbf{x}_j \in S_i^{(t)}} \mathbf{x}_j \quad (19.3)$$

where  $|S_i^{(t)}|$  is the number of elements of  $S_i^{(t)}$ .

### 19.3

## Grayscale image, $k = 2$ in one dimension

The objective is to binarize image image 'cells', which is a grayscale image. The set  $X$  is defined by  $X = \{I(p)\}$ , for  $p$  being the pixels of the image  $I$ .



- Implement the algorithm proposed below (Alg. 6).
- Test this operator on the image 'cells'.
- Test another method of automatic thresholding (defined by Otsu in [?]).
- Compare the values of the thresholds (manual and automatic).

**Data:** Original image  $A$   
**Data:** Stop condition  $\varepsilon$   
**Result:** thresholded image

Initialize  $T_0$ , for example at  $\frac{1}{2}(\max(A) + \min(A))$ ;  
 $done \leftarrow False$ ;

```

while NOT done do
    Segment the image  $A$  with the threshold value  $T$ ;
    it generates two classes  $G_1$  (intensities  $\geq T$ ) and  $G_2$  (intensities  $< T$ );
    Compute the mean values, denoted  $\mu_1, \mu_2$ , of the two classes  $G_1, G_2$ ,
        respectively;
    Compute the new threshold value  $T_i = \frac{1}{2}(\mu_1 + \mu_2)$ ;
    if  $|T_i - T_{i-1}| < \varepsilon$  then
         $| done \leftarrow True$ 
    end
end
Segment the image with the estimated threshold value.
```

**Algorithm 6:** K-means algorithm for automatic threshold computation of grayscale images.



The Matlab function `graythresh` computes the automatic threshold by the method of Otsu.

## 19.4

# Simulation example, $k = 3$ in two dimensions

The objective is to generate a set of 2-D random points (within  $k = 3$  distinct classes) and to apply the  $k$ -means clustering for separating the points and evaluating the method (the classes are known!).



Write a function for generating a set of  $n$  random points around a point with coordinates  $(x, y)$ .



We will use the Matlab function `randn`.



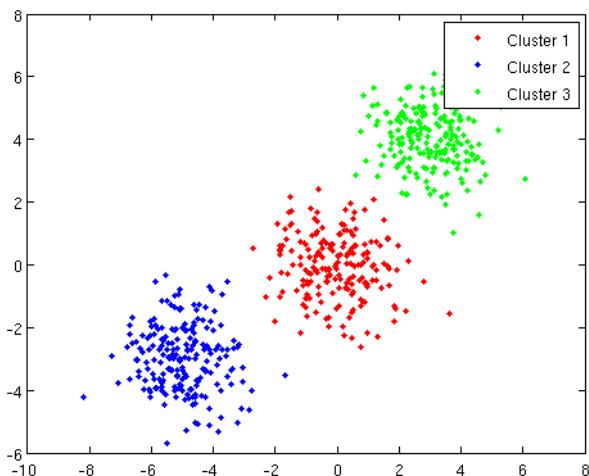
- Use this function to generate 3 set of points (in a unique matrix) around the points  $(0, 0)$ ,  $(3, 4)$  and  $(-5, -3)$ .

- Use the Matlab function kmeans for separating the points. The result is presented in Fig. 19.1.



Verify the utility of the option `replicate`.

Figure 19.1: Resulting clustering of random points.



## 19.5

## Color image segmentation using K-means: $k = 3$ in 3D

The  $k$ -means clustering is now used for segmenting the color image representing the muscle cells 'Tv16.png'.



Which points have to be separated? Transform the original image into a vector of size  $N \times 3$  (where  $N$  is the number of pixels) which represent the 3 components R, G and B of each image pixel.



The MATLAB<sup>®</sup> function `reshape` can perform the transformation.



- Visualize the 3-D map (histogram) of all these color intensities.
- Make the clustering of this 3-D map by using the K-means method.
- Visualize the corresponding segmented image.



## 19.6. Matlab correction



### 19.6.1 Manual thresholding

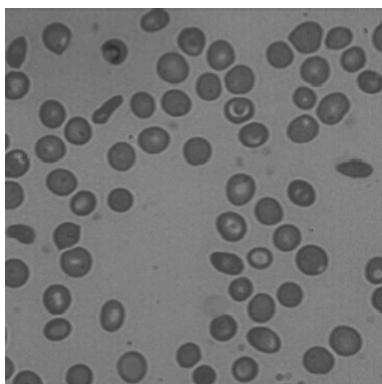
Choose manually a value, by observing the histogram.



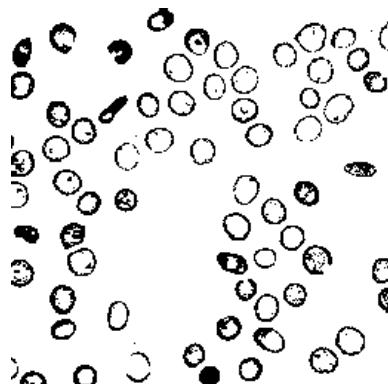
```

1 A=imread(' cells .bmp');
2 B=(A>80);
figure ;
3 subplot (1,2,1) ;imshow(A); title (' Original image');
4 subplot (1,2,2) ;imshow(B); title ('Manual threshold');

```



(a) Original image.



(b) Manual threshold.

Figure 19.2: Simple thresholding.

### 19.6.2 Grayscale image, $k = 2$ in 1D

The automatic threshold selection proposed can be coded as follows:



```

1 function [s,B]=autothresh(A)
% Automatic threshold of image A
3 % return values:
% s: threshold value
5 % B: thresholded (binary) image
7 % initialization of s
s=0.5*(min(A(:)) + max(A(:)));

```



```

9 done = false;

11 % iterate until convergence of s
12 while ~done
13     B=(A>=s);
14     sNext=0.5*(mean(A(B))+mean(A(~B)));
15     done=abs(s-sNext)<0.5; % convergence ?
16     s=sNext;
17 end

```

Then, display the different results with:



```

1 A=imread('cells.bmp');
% threshold determination
3 [s1,B]=autothresh(A);
s1
5
% Otsu (\ matlabregistered {} function)
7 s2=graythresh(A);
s2=255*s2
9 C=(A>=s2);

11 % display results
figure;
13 subplot (2,2,1) ;imshow(A);title ('Original image');
subplot (2,2,3) ;imshow(B);title ('Automatic threshold');
15 subplot (2,2,4) ;imshow(C);title ('Otsu threshold');

```

### Command window



```

1 s1 = 104.0398
s2 = 105.5000

```

### 19.6.3 Simulation example, $k = 3$ in 2D

The first function generates random points around a center. The objective is to retrieve the different clusters (aka classes). Depending on the distance and the distribution of the points, this could not yield to the expected result.



```

function Y=generation(n, x, y)
% Generates n random points (normal law) around point

```

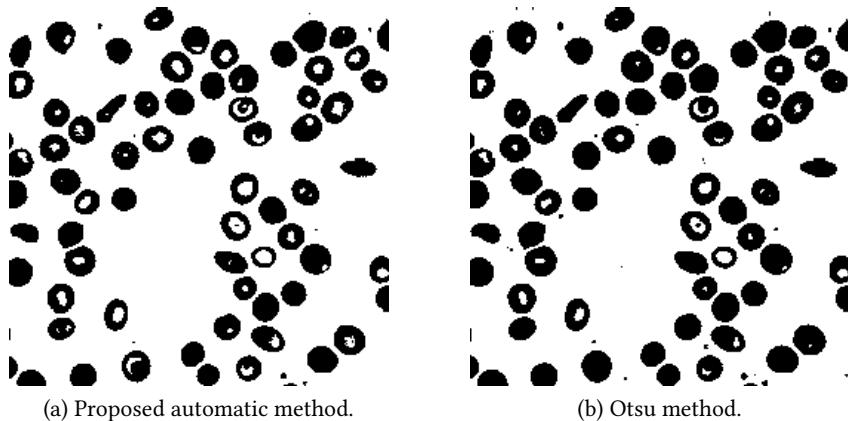


Figure 19.3



```
% of coordinates (x,y)
4 Y = randn(n,2)+ones(n,2)*[x 0; 0 y];
```



```
% Generate 3 point clouds
2 n=100;
X=[generation(n ,3,4) ; generation(n ,0,0) ; generation(n,−5,−3)];
4
% Classification
6 [idx, ctrs] = kmeans(X, 3, 'replicates ', 5);
8 % Display results
figure();
10 plot(X(idx ==1,1),X(idx ==1,2), 'r.', 'MarkerSize',12)
hold on
12 plot(X(idx ==2,1),X(idx ==2,2), 'b+', 'MarkerSize',12)
plot(X(idx ==3,1),X(idx ==3,2), 'g*', 'MarkerSize',12)
14 legend('Cluster 1','Cluster 2','Cluster 3')
```

#### 19.6.4 Color image segmentation by K-means: $k = 3$ in 3D

Each pixel of the color image is represented as a vector (3D point, with the RGB color values of the pixels). Then, the same method is performed. The result is represented in Fig.19.4.



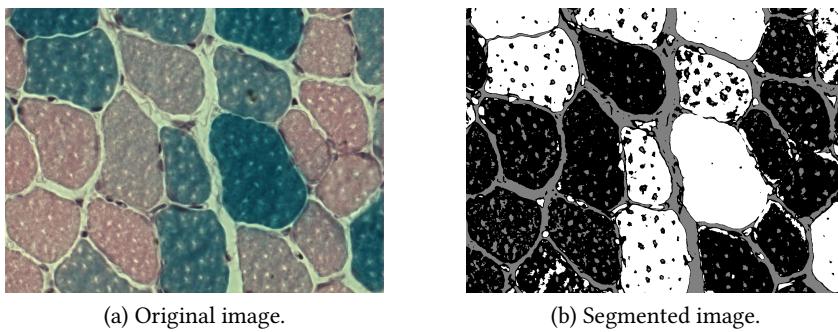
```
1 % Load image
I=imread('Tv16.png');
3 I=double(I);
figure
5 imshow(I/255);

7 % Segmentation
nCouleurs = 3; % number of clusters
9 nLignes = size(I,1);
nCols = size(I,2);
11 X = reshape(I, nLignes*nCols, 3);
13 [index centres] = kmeans(X, nCouleurs, 'distance', 'sqEuclidean', 'replicates', 3)
    ↪ ;
15
% 3D histogram (can be difficult to display, depending on machine)
17 id1=index==1;
id2=index==2;
19 id3=index==3;

21 figure
plot3(X(id1,1), X(id1,2), X(id1,3), 'r.')
23 hold on
plot3(X(id2,1), X(id2,2), X(id2,3), 'g.')
25 plot3(X(id3,1), X(id3,2), X(id3,3), 'b.')

27 % Label each pixel
labels = uint8(reshape(index, nLignes, nCols));
29 labels = imadjust(labels);

31 figure
subplot(121)
33 imshow(I/255);
subplot(122)
35 imshow(labels, []);
```



(a) Original image.

(b) Segmented image.

Figure 19.4: K-means segmentation applied on a color image. The segmentation is applied in the color space. The spatial informations of the image structures is lost.

# ★ 20 Segmentation by region growing

This tutorial proposes to program the region growing algorithm.

## 20.1 Introduction

The region growing segmentation method starts from a seed. The initial region first contains this seed and then grows according to

- a growth mechanism (in this tutorial, the  $N_8$  will be considered)
- an homogeneity rule (predicate function)

The algorithm is simple and barely only needs a predicate function:

```
Data: I: image
Data: seed: starting point
Data: queue: queue of points to consider
Result: visited: boolean matrix, same size as I
begin
    queue.enqueue( seed );
    while queue is not empty do
        p = queue.dequeue();
        foreach neighbor of p do
            if not visited(p) and neighbor verifies predicate then
                queue.enqueue( neighbor );
                visited( neighbor ) = true;
            end
        end
    end
    return visited
end
```

**Algorithm 7:**

The difficulty of the code lays in the presence of a queue structure. For the purpose of simplicity, it is advised to use the java class `java.util.LinkedList`, which is allowed within matlab:



```
% create the queue structure by a Java object
2 queue = java.util.LinkedList;
```



```

4 % test it
p = [1;2];
6 queue.add(p);

8 r = queue.remove()

10 % test if queue is empty
if queue.isEmpty()
12     % queue is empty
end

```

For picking up the coordinates of a pixel with the mouse in MATLAB<sup>®</sup>, you can use the `ginput` function:



```

I = double(imread('cameraman.tif'));
2 [Sx, Sy] = size(I);
imshow(I,[]);
4
% seed
6 [x, y]=ginput(1);
seed = round([y;x]); % beware of inversion of coordinates

```

## 20.2 Region growing implementation



The seed pixel is denoted  $s$ .

- Code the predicate function: for an image  $f$  and a pixel  $p$ ,  $p$  is in the same segment as  $s$  implies  $|f(s) - f(p)| \leq T$ .
- Code a function that performs region growing, from a starting pixel (seed).
- Try others predicate functions like:
  - pixel  $p$  intensity is close to the region mean value, i.e.:

$$|I(p) - m_s| \leq T$$

- Threshold value  $T_i$  varies depending on the region  $R_i$  and the intensity of the pixel  $I(p)$ . It can be chosen this way:

$$T_i = \left(1 - \frac{\sigma_i}{m_i}\right) \cdot T$$



## 20.3. Matlab correction



The tricky part of this code is that, for convenience, some Java code is used inside matlab. This can be really useful when using particular structures and objects like lists, queues, etc.



```

1 % needs an image I, gray level
% double is needed to perform comparison
3 I = double(imread('cameraman.tif'));
[Sx, Sy] = size(I);
5 imshow(I,[]);
% seed
7 [x, y]=ginput(1);
9 seed = round([y;x]); % beware of inversion of coordinates
11 I(seed(1), seed(2))
% create the queue structure by a Java object
13 queue = java.util.LinkedList;
15 % Visited matrix : result of segmentation
17 % this matrix will contain 1 if in the region,
% -1 if visited but not in the region,
% 0 if not visited.
19 visited = zeros(size(I));

```

The next code is used to compute the visited matrix and display it.



```

% Start of algorithm -----
2 queue.add(seed);
visited (seed(1), seed(2)) = 1;
4 tic
6 while ~queue.isEmpty()
    p = queue.remove();
8 % look at the pixel in a 8-neighborhood
10 r = p(1); % row
    c = p(2); % col
12 for i=max(1,r-1):min(Sx,r+1)
        for j=max(1,c-1):min(Sy,c+1)
14            if (visited (i,j)==0) % not visited yet
                if (predicate(I, [i j], seed, visited ))
16                % condition is fulfilled
                    visited (i, j) = 1;

```



```

18     queue.add([ i ; j ]) ; % add to visiting queue
19     else
20         visited ( i , j ) = -1;
21     end
22     end
23 end
24 end
25 toc
% end of the algorithm:
26 % the visited matrix contains the segmentation result
27
28 figure () ; imshow(visited ==1,[]);
29
30

```

Notice that values  $-1$  of the visited matrix avoid testing multiple times the same pixel. In the predicate function, the visited matrix is used in case of adapting the predicate to the current region. In the next case, the candidate pixel's graylevel is compared to the mean gray value of the region. The results are illustrated Fig.20.1.



```

1 function r = predicate2(I, p, seed, visited )
2 % threshold parameter
3 t = 10;
4
5 m = mean(I( visited ==1));
6 if abs(I(p(1), p(2)) - m) <= t
7     r = true;
8 else
9     r = false ;
10 end

```

Another predicate function would be:



```

1 function r = predicate3(I, p, seed, visited )
2 % threshold parameter
3 t = 10;
4
5 m = mean(I( visited ==1));
6 sigma = std(I( visited ==1));
7 if abs(I(p(1), p(2)) - m) <= t * (1-sigma/m)
8     r = true;
9 else
10    r = false ;
11 end

```



(a) Original image.



(b) First predicate function.



(c) Second predicate function.



(d) Third predicate function.

Figure 20.1: The segmentation result highly depend on the order used to populate the queue, on the predicate function and on the seed pixel. The seed pixel is chosen somewhere in the coat of the cameraman.

# ★★ 21 Hough transform and line detection

This tutorial introduces the Hough transform. Line detection operators are implemented.

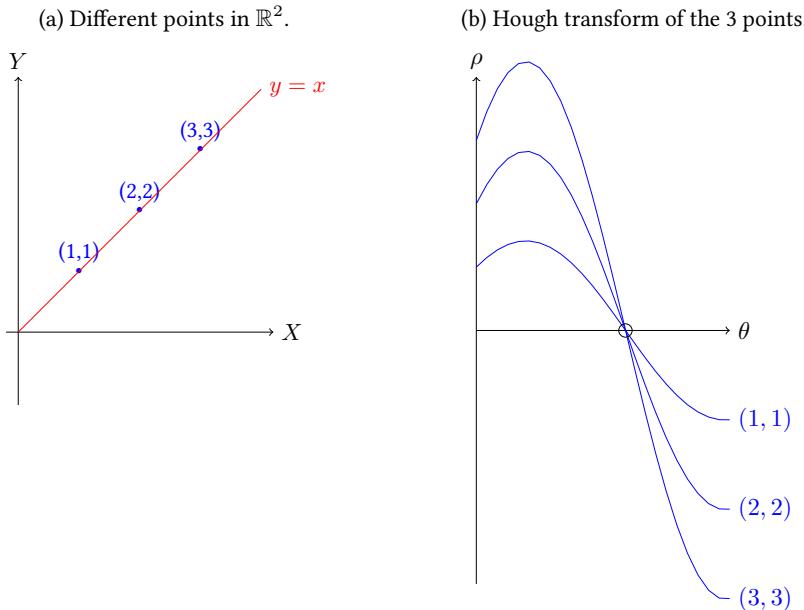
## 21.1 Introduction

This tutorial deals with line detection in an image. For a given point of coordinates  $(x, y)$  in  $\mathbb{R}^2$ , there exists an infinite number of lines going by this point, with different angles  $\theta$ . These lines are represented by the following equation:

$$\rho = x \cdot \cos(\theta) + y \cdot \sin(\theta).$$

Thus, for each point  $(x, y)$  (Fig. 21.1a) corresponds a curve parameterized by  $[\theta, \rho]$ , where  $\theta \in [0; 2\pi]$  (Fig. 21.1b). The intersection of these curves represents a line (in this case,  $y = x$ ).

Figure 21.1: Representation of the Hough transform.



## 21.2 Algorithm

The (general and simple) method for line detection is then:

1. Compute contours detections (get a binary image BW).
2. Apply the Hough transform on the contours BW.
3. Detect the maxima of the Hough transform.
4. Get back in the Euclidean space and draw the lines on the image.

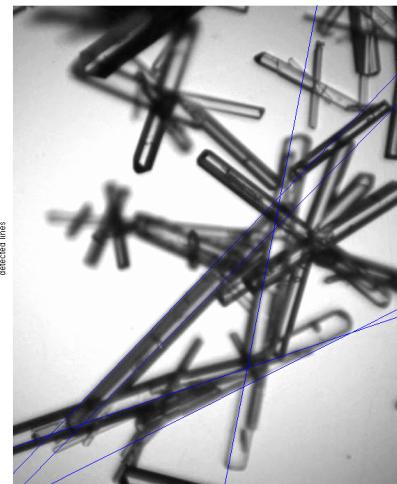
Results should look like in Fig. 21.2.

Figure 21.2: Lines detection via Hough transform.

(a) Hough transform and maxima detection. Angles  $\theta$  are represented in abscissa, pixels  $\rho$  are represented in ordinates. The detection of the absolute maxima of this images will lead to the lines.



(b) Line detection.



## 21.3

## Hough transform



Code a function that will transform each point of a binary image into a curve in the Hough space. For each curve, increment each pixel by one in the Hough space.

**21.4**

## Maxima detection



Use or code a function to detect maxima (regional maxima). For each maximum, keep only one point.

**21.5**

## Display lines



For each maximum, display the corresponding line above the original image.



## 21.6. Matlab correction



### 21.6.1 Contour detection

The first step is to perform contours detections. A classical method is employed here (see Fig.21.3, Canny edge detection). The important thing is to start by a binary image (binary set of points).



```

1 % Load an image
I = double(imread('TestPR46.png'));
3 I = I (:,:2) ; % keep grayscale image

5 %% performs contour detection
BW = edge(I, 'canny');

```



Figure 21.3: Canny edge detection.

### 21.6.2 Hough transform

This code does not make use of the MATLAB<sup>®</sup> function dedicated to line detection. The result is presented in Fig.21.2.

First, you can initialize the values. The size of the image is used to determine the maximal  $\rho$  value.



```

%% Hough transform
2 angular_sampling = 0.002; % angles in radians
[x, y] = size (BW);
4

```



```

rho_max = norm([x y]);
6 rho = -rho_max:1:rho_max;
theta = 0:angular_sampling:pi;
8 H = zeros(length(rho), length(theta));

```

Then, you loop over all the pixels  $(i, j)$ : in case of a True pixel  $(BW(i,j)==1$ , you transform it into a sinusoid function, and increase the rounded values in the  $H$  matrix for all discrete values of  $\theta$ .



```

% performs Hough transform
2 for i = 1:x
    for j = 1:y
        if BW(i, j)
            for theta_index = 1:length(theta)
6                th = theta(theta_index);
                r = i * cos(th) + j * sin(th);
8                rho_index = round(r + length(rho)/2);
                H(rho_index, theta_index) = H(rho_index, theta_index) + 1;
10           end
            end
12       end
end

```

### 21.6.3 Maxima detection

#### Basic maxima detection

This version of maxima detection is very simple. However, it does not handle the neighborhood (it has the drawbacks of a basic threshold). One could look at h-maxima operators in order to get blobs instead of points. The threshold value can be tuned to find a given number of lines.



```

1 %% maxima detection
difference = 50;
3 M = max(H(:));
maxima = H > (M - difference);
5
% find the peaks
7 [indices_rho_peaks, indices_theta_peaks] = find(maxima);

```

### Enhanced maxima detection

The MATLAB® version of the maxima detection gives cleaner maxima. Each peak, described by a coordinate  $\rho, \theta$ , corresponds to a line in the original image.



```
1 peaks = houghpeaks(H, 5);
2 indices_rho_peaks = peaks (:,1) ;
3 indices_theta_peaks = peaks (:,2) ;
```

The following code displays the results in the Hough space.



```
1 rho_peaks = rho(indices_rho_peaks);
2 theta_peaks = theta( indices_theta_peaks );
3
4 imshow(H,[]), hold on
5 title ('Hough Transform');
6 xlabel ('\theta ( radians )');
7 ylabel ('\rho ( pixels )');
8 plot( indices_theta_peaks , indices_rho_peaks , 'r*' );
```

### 21.6.4 Lines retrieval

From the coordinates  $\rho, \theta$ , it is easy to compute and display the different detected lines.



```
%% Find hough lines
1 x= 1: size(I, 2);
2 figure , imshow(I,[]), hold on
3 for i=1:length(rho_peaks)
4     y = (rho_peaks(i) - x* cos(theta_peaks(i)) )/ sin(theta_peaks(i));
5     plot(y, x);
6 end
7 title ('detected lines')
```



## 22 Active contours

This tutorial aims at introducing the active contours (a.k.a. snakes) method as originally presented in [?]. It is a segmentation method based on the optimization of a contour that will converge to a specific object.

### 22.1

### Definition

A snake is a parametric curve  $v(s)$  with  $s \in [0; 1]$ . The energy functional is represented by:

$$E_{\text{snake}} = \int_0^1 E_{\text{int}}(v(s)) ds + \int_0^1 E_{\text{ext}}(v(s)) ds.$$

The internal energy is detailed in Eq. 22.1. The first order derivation constrains the length of the curve, the second order derivation constrains the curvature. The parameters  $\alpha$  and  $\beta$  a priori depend on  $s$ , but for simplicity, constant values will be taken.

$$E_{\text{int}}(v(s)) = \frac{1}{2} \left( \underbrace{\alpha(s)|v'(s)|^2}_{\text{length}} + \underbrace{\beta(s)|v''(s)|^2}_{\text{curvature}} \right) \quad (22.1)$$

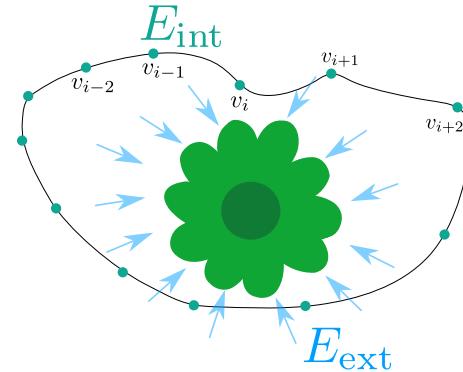
The snake will be attracted by some edges points from the external energy (the image to be segmented). The energy functional will be in a local minimum: it is shown that the Euler-Lagrange equation is satisfied:

$$-\alpha v^{(2)} + \beta v^{(4)} + \nabla E_{\text{ext}} = 0$$

with  $v^{(2)}$  and  $v^{(4)}$  denoting the 2nd and 4th order derivatives. To solve this equation, the gradient descent method is employed: the snake is now transformed into a function of the position  $s$  and the time  $t$ , with  $F_{\text{ext}} = -\nabla E_{\text{ext}}$

$$\frac{\partial s}{\partial t} = \alpha v^{(2)} - \beta v^{(4)} + F_{\text{ext}}$$

Figure 22.1: Illustration of the active contours segmentation method. Two energies are at stake: internal energies depend only on the snake shape and control points, external energies are related to the image properties.



## 22.2 Numerical resolution

The spatial derivatives are approximated with the finite difference method, and the snake is now composed of  $n$  points,  $i \in [1; n]$ :

$$\begin{aligned} x^{(2)} &= x_{i-1} - 2x_i + x_{i+1} \\ x^{(4)} &= x_{i-2} - 4x_{i-1} + 6x_i - 4x_{i+1} + x_{i+2} \end{aligned}$$

The gradient descent method can be written as, with  $\gamma$  being the time step and controls the convergence speed:

$$\begin{aligned} \frac{x_t - x_{t-1}}{\gamma} &= A \cdot x_t + f_x(x_{t-1}, y_{t-1}) \\ \frac{y_t - y_{t-1}}{\gamma} &= A \cdot y_t + f_y(x_{t-1}, y_{t-1}) \end{aligned}$$

with

$$A = \begin{bmatrix} -2\alpha - 6\beta & \alpha + 4\beta & -\beta & 0 & \dots & 0 & -\beta & \alpha + 4\beta \\ \alpha + 4\beta & -2\alpha - 6\beta & \alpha + 4\beta & -\beta & 0 & \dots & 0 & -\beta \\ -\beta & \alpha + 4\beta & -2\alpha - 6\beta & \alpha + 4\beta & -\beta & 0 & \dots & 0 \\ 0 & -\beta & \alpha + 4\beta & -2\alpha - 6\beta & \alpha + 4\beta & -\beta & 0 & \dots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & -\beta & \alpha + 4\beta & -2\alpha - 6\beta & \alpha + 4\beta & -\beta & 0 \\ 0 & \dots & 0 & -\beta & \alpha + 4\beta & -2\alpha - 6\beta & \alpha + 4\beta & -\beta \\ -\beta & 0 & \dots & 0 & -\beta & \alpha + 4\beta & -2\alpha - 6\beta & \alpha + 4\beta \\ \alpha + 4\beta & -\beta & 0 & \dots & 0 & -\beta & \alpha + 4\beta & -2\alpha - 6\beta \end{bmatrix}$$

The forces  $f_x$  and  $f_y$  are the components of  $F_{\text{ext}}$ . For example for an image  $I$ , if  $*$  denotes the convolution and  $G_\sigma$  a gaussian kernel of standard deviation  $\sigma$ :

$$F_{\text{ext}} = -\nabla(\|\nabla(G_\sigma * I)\|)$$



- Generate a binary image (of size  $n \times n$  pixels, with values 0 and 1) containing a disk (of a radius  $R$ ).
- Generate the initial contour as an ellipse, with the same center as the disk.
- Generate the pentadiagonal matrix  $A$ . The different parameters can be  $\alpha = 10^{-5}$  and  $\beta = 0.05$ .
- Program the iterations and visualize the results. For example,  $\gamma = 200$  and 1000 iterations may give an idea of the parameters to use.



## 22.3. Matlab correction



### 22.3.1 Binary image generation

A circle is generated via the `meshgrid` function (see Fig.22.2).



```
n = 512+256; % size of image
2 R =200; % radius of circle
[X, Y] = meshgrid(-n/2:n/2-1, -n/2:n/2-1);
4 I = double(X.^2+Y.^2 >= R.^2);
```

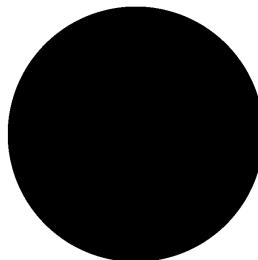


Figure 22.2: The active contour will converge toward this circle.

### 22.3.2 Initial contour

The choice of the initial contour is crucial in this method. The parameters used in this example ensure the convergence of the snake. To emphasize the convergence, an ellipse is chosen as the initial contour (see Fig.22.3).



```
% ellipse formula
2 x = n/2 + 300 * cos (0:1:2* pi);
y = n/2 + 150 * sin (0:1:2* pi);
4
% in order to be consistent with the following code, one has to take the
% → transposed vector.
6 x = x';
```



```
y = y';
```

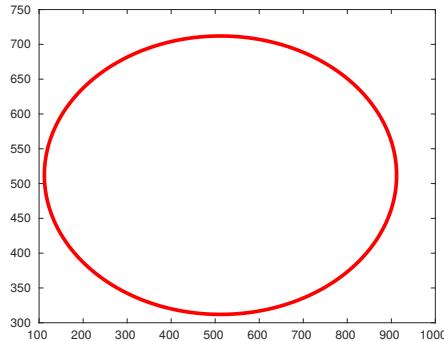


Figure 22.3: Initialization of the active contour.

The different parameters are defined by:



```
1 alpha = .00001;
2 beta = .05;
3 gamma = 200;
iterations = 1000;
```

### 22.3.3 Matrix construction

This is maybe the hardest part of this code, with the use of the [spdiags](#) function.



```
N = length(x);
2 X = [-beta alpha+4*beta -2*alpha-6*beta alpha+4*beta -beta -beta alpha+4*beta -
       ↪ beta alpha+4*beta];
B = repmat(X, N, 1);
4 A = full ( spdiags(B, [-2 -1 0 1 2 N-2 N-1 -N+2 -N+1], N, N));
AA = eye(N) - gamma*A;
```

Be aware that in MATLAB®, for efficiency reasons, the invert of the matrix can be written as [inv\(A\)\\*y=A\y](#).

### 22.3.4 External forces



```
% define convolution kernels
2 hgauss = fspecial ('gaussian', 100, 30);
hprewitt = fspecial ('prewitt');

4 % gaussian filter
G = imfilter (I, hgauss, 'replicate');

% gradient (prewitt) and its norm
8 Fy= imfilter (G, hprewitt, 'symmetric');
Fx= imfilter (G, hprewitt, 'symmetric');

10 G = sqrt(Fx.^2+Fy.^2);

12 % orientation of previous gradient
Fy= imfilter (-G, hprewitt, 'symmetric');
14 Fx= imfilter (-G, hprewitt, 'symmetric');
```

### 22.3.5 Display results

To enhance the role of the external forces, the arrows showing the force are displayed (`quiver` function).



```
imshow(I,[])
2 hold on
plot([x;x(1)], [y; y(1)], 'g', 'linewidth', 3);

4 %% display arrows for external forces
6 step=20;
subx = 1:step:size(I,1);
8 suby = 1:step:size(I,2);
[Xa, Ya] = meshgrid(subx, suby);
10 quiver(Xa, Ya, Fx(subx, suby), Fy(subx,suby));
```

### 22.3.6 Convergence algorithm



```
h = waitbar(0, 'snake converging ... ');
2 % iterations
for index = 1:iterations,
    % interpolate values of forces
```

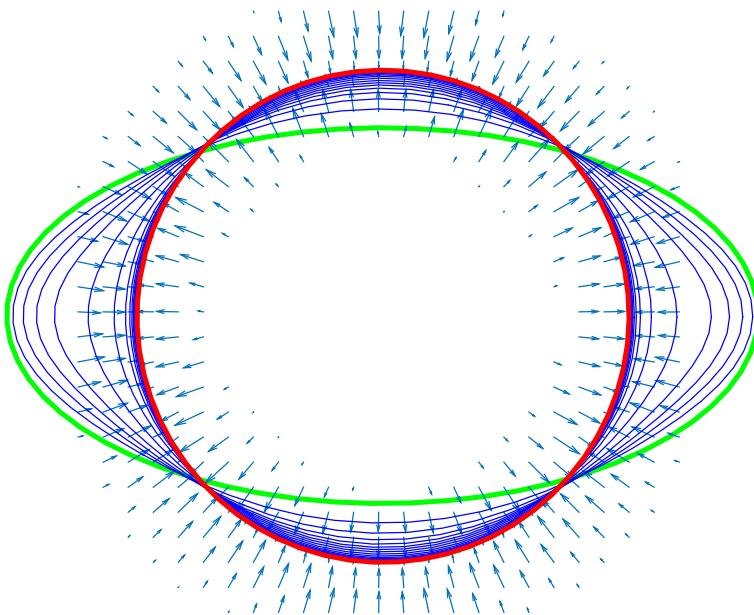


Figure 22.4: Result of the snake converging toward the disk, after 1000 iterations with the proposed parameters. The green ellipse is the original snake, the red snake shows the final result. Blue snakes are intermediate results.



```

fex = interp2(Fx, x, y, 'linear');
6   fey = interp2(Fy, x, y);

8   x = AA\ (x+gamma*fex);
y = AA\ (y+gamma*fey);
10 % display
if mod(index,10)==0
12   plot ([x;x(1)], [y;y(1)], 'b');
end
14 waitbar(index/ iterations );
end
16 plot ([x;x(1)], [y;y(1)], 'r', 'linewidth', 3);
close (h)

```

The results are displayed in Fig.22.4.

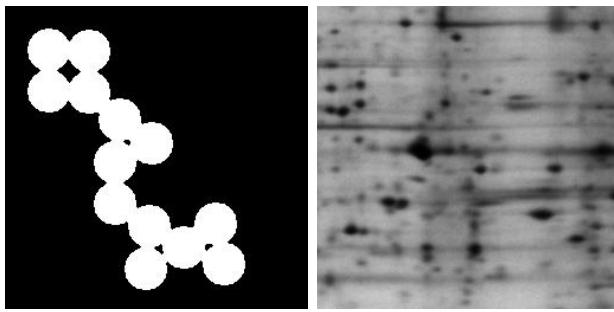




## 23 Watershed

This tutorial aims to study the watershed transform for image segmentation. In image processing, an image can be considered as a topographic surface. If we flood this surface from its minima and if we prevent the merging of the water coming from different sources, we partition the image into two different sets: the catchment basins separated by the watershed lines.

The different processes will be applied on the following images:



### 23.1

## Watershed and distance maps

The objective is to individualize the disks by disconnecting them with the distance map.



- Calculate the distance map of the image 'circles'.
- Take the complementary of this distance map and visualize its minima.
- Calculate the watershed transform of the inverted distance map.
- Subtract the watershed lines to the original image.



The distance map can be calculated by `bwdist` on a binary image, the local minima are evaluated by `imregionalmin`.

## 23.2

# Watershed and image gradients



- Calculate the Sobel gradient of the image 'gel'.
- Visualize the minima of the image gradient.
- Apply the watershed transform on the gradient image.

The watershed transform, applied in a direct way, leads to an over-segmentation. To overcome this limitation, the watershed operator can be applied on a filtered image.



- Smooth the original image with a low pass filter. To stay in the mathematical morphology field, you can use an alternate morphological filter (opening followed by closing). A Gaussian filter is also a good solution.
- Calculate the gradient operator on the filtered image.
- Calculate the corresponding watershed.



The mathematical morphology function of opening and closing are `imopen` and `imclose`.

## 23.3

# Constrained watershed by markers

In order to individualize the image spots, we have to determine the internal and external markers for the constrained watershed.



- Calculate the gradient (Sobel) of the filtered image.
- Calculate the internal markers (minima of the filtered image) and external (watershed of the filtered image) as minima of the gradient image.
- Calculate the corresponding watershed.

- Superimpose the watershed lines of the resulting segmentation to the original image.



`imgradient` returns the gradient magnitude of an image. `imimposemin` is used to impose minima to an image in order to perform watershed segmentation.



## 23.4. Matlab correction



### 23.4.1 Watershed and distance map

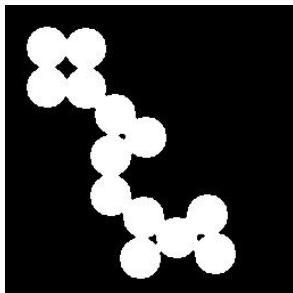
This method is a classical way of performing the separation of some objects by proximity or influence zones. It is illustrated in Fig.23.1.



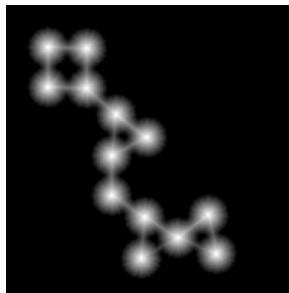
```

1 A=imread('circles.tif');
2 % distance map
3 dist =bwdist(~A);
4 % watershed
5 waf=watershed(imcomplement(dist));
6 waf=(waf==0);
7 % separation of the grains
8 B=A & ~waf;

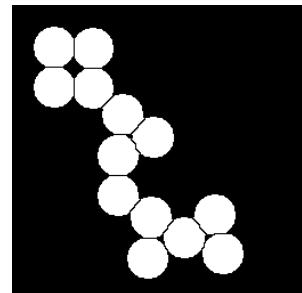
```



(a) Original image.



(b) Distance map.



(c) Separation of the grains.

Figure 23.1: Steps of the separation of the grains.

### 23.4.2 Watershed and image gradients

The gradient image amplifies the noise. Thus, the watershed operator directly applied to the gradient of the image produces an over-segmented image (see Fig.23.1).



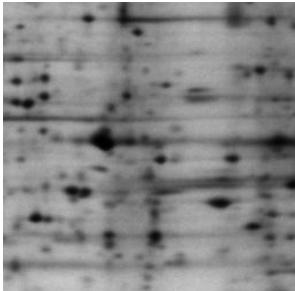
```

1 % read grayscale image
2 A=imread('gel.jpg');
3 % gradient
4 gradient=sobel(A);
5 rm=imregionalmin(gradient);
6 % watershed

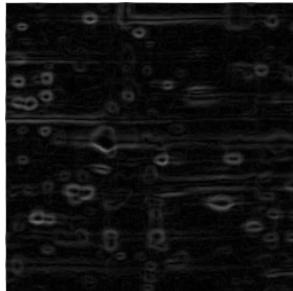
```



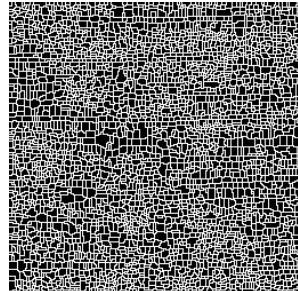
```
wat=watershed(gradient);
8 wat=(wat==0);
```



(a) Original image.



(b) Amplitude of the gradient (Sobel).



(c) Watershed segmentation.

Figure 23.2: Performing the watershed on the gradient image is not a good idea.

In fact, this method produces as many segments as there are minima in the gradient image Fig.23.3.

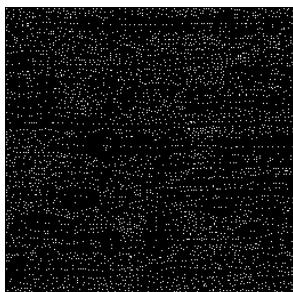


Figure 23.3: Local minima of the gradient image.

### Solution: filtering the image

Before evaluating the gradient, the image is filtered. The number of minima is lower and this leads to a less over-segmented image (Fig.23.4).



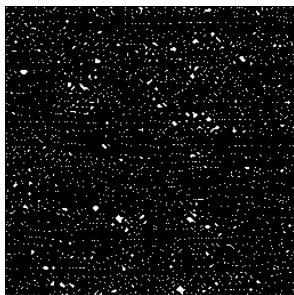
```
A=imread('gel.jpg');
2 % filtering
se = strel ('disk',2);
4 AA=imopen(A,se);
f=imclose(AA,se);
```



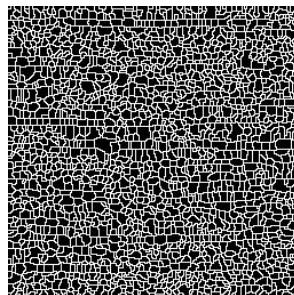
```

6 % gradient
gradient=imgradient(f);
8 rm=imregionalmin(gradient);
% watershed
10 wat=watershed(gradient);
wat=(wat==0);

```



(a) Minima of the gradient after filtering the image.



(b) Watershed segmentation.

Figure 23.4: Even if the gradient is performed on the filtered image, there is still a high over-segmentation.

### 23.4.3 Watershed constrained by markers

The watershed can be constrained by markers: the markers can provide the correct number of regions. This method imposes both the background (external markers) and the objects (internal markers). The results are illustrated in Fig.23.5. The ultimate erosion of the internal markers is used to disconnect these markers from the external markers.

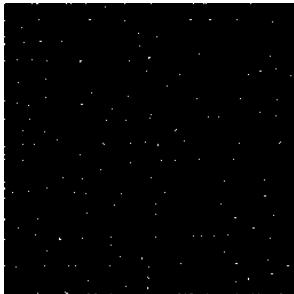


```

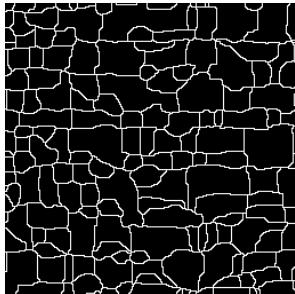
1 % filtering
se = strel ('disk',2);
3 AA=imopen(A,se);
f=imclose(AA,se);
5 % internal markers (of the objects)
rm=imregionalmin(f);
7 rm = bwulterode(rm);
% external markers: watershed of filtered image
9 % background of the objects
watf=watershed(f);
11 watf=(watf==0);

```

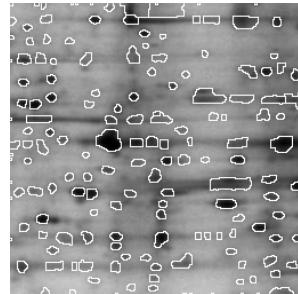
```
13 % constrain the minima  
gradient=imgradient(f);  
15 mie=imimposemin(gradient, rm | watf);  
minima=max(rm, watf);  
17 % watershed constraint  
watc=watershed(mie);  
19 watc=(watc==0);
```



(a) Internal markers (blobs).



(b) External (background) markers .



(c) Final segmentation.

Figure 23.5: Watershed segmentation by markers.



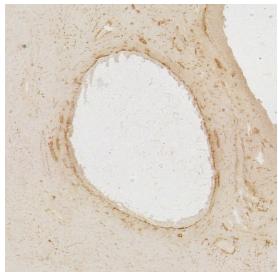
## ★ 24 Segmentation of follicles

This practical work aims to investigate image segmentation with a direct application to ovarian follicles. The overall objective is to extract and quantify the granulosa cells and the vascularization of each follicle included in an ewe's ovary.

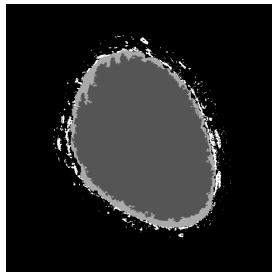
The image to be processed is a 2D histological image of an ewe's ovary acquired by optical microscopy in Fig.24.1. The presented image contains one entire follicle (the white region and its neighborhood) and a part of a second one (right-upper corner). The follicle is composed of different parts shown in: antrum, granulosa cells and vascularization. The theca is the ring region around the antrum where the follicle is vascularized.

Figure 24.1: Different parts of the follicles, to be segmented.

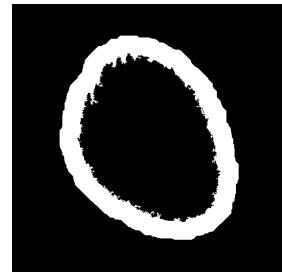
(a) Original image with one entire follicle (white region and its neighborhood).



(b) Antrum (dark gray),  
granulosa cells (light gray)  
and vascularization (white)  
of the follicle.



(c) Theca.



### 24.1 Vascularization



- Load and visualize the image.
- Extract the antrum of the follicle.
- Extract the vascularization (inside a ring around the antrum).

## 24.2 Granulosa cells



Which kind of processing could be suitable for extracting the granulosa cells?

## 24.3 Quantification



Provide some geometrical measurements of the different entities of the follicle (antrum, vascularization, granulosa cells).



## 24.4. Matlab correction



### 24.4.1 Vascularization

#### Antrum segmentation

The first step consists in the segmentation of the antrum by thresholding the blue component. Some post-processes are used to remove artifacts such as holes. This is illustrated in Fig. 24.2.



```

1 % image reading
A = imread(' follicule .bmp');
3 % image visualization
figure ;imagesc(A); title (' Original image');
5 % antrum segmentation
B = A (:,:,3) ;
7 antrum = (B>220);
antrum = bwselect(antrum ,300,300,8) ;
9 antrum = imfill (antrum,'holes');
figure ;colormap gray;imagesc(antrum); title (' Antrum');

```

#### Theca segmentation

The second step provides the segmentation of the theca which is extracted as a spatial region (corona) adjacent to and outside the antrum. The width of the corona is selected by the user (expert).



```

se = strel (' disk ',40) ;
2 theque = imdilate (antrum,se);
theque = theque - antrum;
4 figure ;colormap gray;imagesc(theque); title ('Theca');

```

#### Vascularization segmentation

The final step extracts the vascularization of the considered follicle. Pixels belonging to the vascularization are considered to have a low blue component and they should also be included in the theca of the follicle.

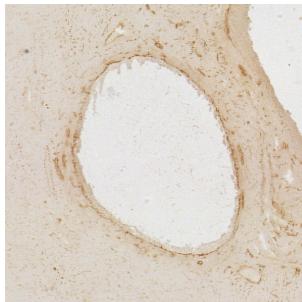


```

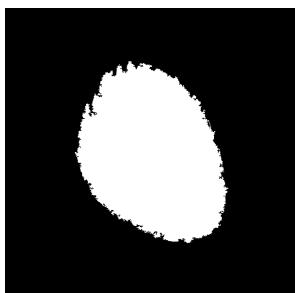
vascularisation = (B<140);
2 vascularisation = min( vascularisation ,theque);
figure ; colormap gray; imagesc( vascularisation ); title (' Vascularisation ');

```

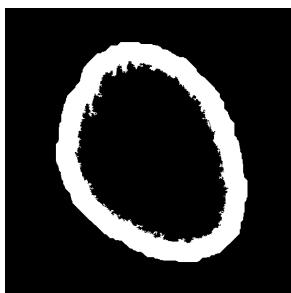
## Results



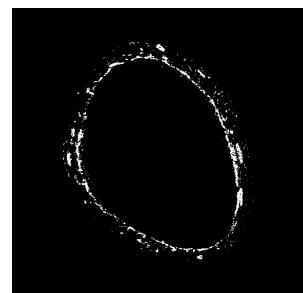
(d) Original image.



(e) Antrum.



(f) Theca.



(g) Vascularization.

Figure 24.2: Extraction of the different components of the follicle.

### 24.4.2 Granulosa cells

#### First solution

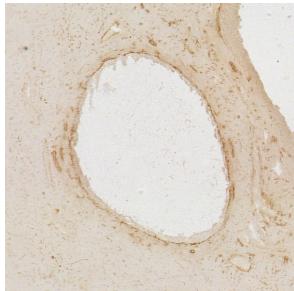
The granulosa cells have a low contrast, so it is difficult to use thresholding techniques. But we know they are localized between the antrum and the vascularization. Nevertheless the vascularization is not a closed region outside the antrum. Therefore, the proposed solution consists in first trying to close the vascularization region and taking the corona between this region and the antrum. The result is shown in Fig. 24.3.



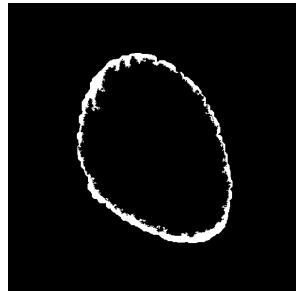
```

1 dil = 1-imclose( vascularisation , strel ('disk',10));
dil = bwselect( dil , 300, 300, 4);
3 granulosa = dil - antrum;
figure; colormap gray; imagesc(granulosa); title ('Cellules de granulosa');

```



(a) Original image.



(b) Granulosa cells.

Figure 24.3: Extraction of the granulosa cells of the follicle.

### Second solution

This first solution is not really robust. The closing of the vascularization region is not really accurate. A more robust solution consists in using deformable models to get the corona between the vascularization and the antrum. But this kind of method is out of the scope of this tutorial.

#### 24.4.3 Quantification

The quantification is easy to process. In addition, we can represent the different extracted components of the follicle in false colors.



```

all = antrum + 2*granulosa + 3* vascularisation ;
2 figure; colormap jet; imagesc(all); title ('Antrum, vascularization and granulosa
↔ cells ');
follicule = antrum + theque;
4 qVascularisation = bwarea( vascularisation )/bwarea( follicule )
qGranulosa = bwarea(granulosa)/bwarea( follicule )

```

This quantification gives:

```
Command window ▾  
1 qVascularisation = 0.0448  
qGranulosa = 0.0912
```



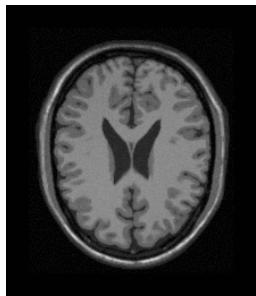
## 25 Image Registration

This tutorial aims to implement the Iterative Closest Point (ICP) method for image registration. More specifically, we are going to estimate a rigid transformation (translation + rotation without scaling) between two images.

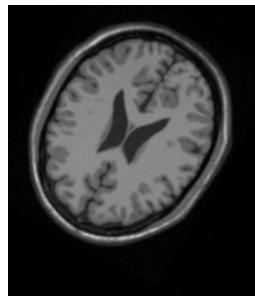
The different processes will be applied on T1-MR images of the brain in Fig.25.1.

Figure 25.1: Original images.

(a) brain1



(b) brain2



### 25.1

## Transformation estimation

A classical method in image registration first consists in identifying and matching some characteristic points by pairs. Thereafter, the transformation is estimated from this list of pairs (displacement vectors).

### 25.1.1 Preliminaries

Pairs of points are first manually selected.



- Read and visualize the two MR images 'brain1' and 'brain2' (moving and source images).
- Manually select a list of corresponding points.



`cpselect` is a graphical interface dedicated to manual selection of pairs of points.

### 25.1.2 Rigid transformation

With this list of pairs  $(p_i, q_i)_i$ , this tutorial proposes to estimate a rigid transformation between these points. It is composed of a rotation and a translation (and not scaling). For doing that, we make a Least Squares (LS) optimization, which is defined as follows. The parameters of the rotation  $R$  and the translation  $t$  minimize the following criterion:

$$C(R, t) = \sum_i \|q_i - R.p_i - t\|^2$$

#### Calculation of the translation :

The optimal translation is characterized by a null derivative of the criterion:

$$\frac{\partial C}{\partial t} = -2 \sum_i (q_i - R.p_i - t)^T = 0 \Leftrightarrow \sum_i q_i - R. \left( \sum_i p_i \right) = N.t$$

where  $N$  denotes the number of matching pairs. By denoting  $\bar{p} = \frac{1}{N} \sum_i p_i$  and  $\bar{q} = \frac{1}{N} \sum_i q_i$  the barycenters of the point sets and by changing the geometrical referential:  $p'_i = p_i - \bar{p}$  et  $q'_i = q_i - \bar{q}$ , the criterion can be written as:

$$C'(R) = \sum_i \|q'_i - R.p'_i\|^2$$

The estimated rotation  $\hat{R}$  will provide the expected translation:

$$\hat{t} = \bar{q} - \hat{R}.\bar{p} \quad (25.1)$$

#### calculation of the rotation by the SVD method:

We will use the following theorem: Let  $U.D.V^T = K$  a singular decomposition of the correlation matrix  $K = q'^T.p'$ , for which the singular values are sorted in the increasing order. The minimum of the criterion:  $C(R) = \sum_i \|q'_i - R.p'_i\|^2$  is reached by the matrix :

$$\hat{R} = U.S.V^T \quad (25.2)$$

with  $S = \text{DIAG}(1, \dots, 1, \det(U). \det(V))$ .



- Code a function that takes as parameters the pairs of points, and returns the rigid transformation elements of rotation  $\hat{R}$  and translation  $\hat{t}$  as defined in Eqs.25.1 and 25.2.
- Apply the transformation to the moving image.

- Visualize the resulting registered image.



- See [svd](#) function.
- See [imwarp](#) and [affine2d](#) for transformation construction and application.
- See [imshowpair](#) for displaying the registration results.

## 25.2

# ICP-based registration

When the points are not correctly paired, it is first necessary to reorder them before estimating the transformation. In this way, the ICP (iterative Closest Points) consists in an iterative process of three steps: finding the correspondence between points, estimating the transformation and applying it. The process should converge to the well registered image.



To simulate the mixing of the points, randomly shuffle them selected on the first image and perform the registration with the previous method.



See [randperm](#).



1. From the list of the characteristic points  $p$  of the image 'brain1', find the nearest neighbors  $q$  in the image 'brain2'. Be careful to the order of the input arguments.
2. Estimate the transformation  $T$  by using the LS minimization coded previously.
3. Apply this transformation to the points  $p$ , find again the correspondence between these resulting points  $T(p)$  and  $q$  and estimate a new transformation. Repeat the process until convergence.
4. Visualize the resulting registered image.



Look at [dsearchn](#) for nearest neighbor search.

## 25.3

# Automatic control points detection

The manual selection of points may be fastidious. Two simple automatic methods for detecting control points are the so-called Harris corners detection or Shi-Tomasi corner detector.



The function `detectHarrisFeatures` returns the corners, their coordinates can be retrieved by `corners.selectStrongest(nb_points)`.



Replace the manual selection in the two previous parts of this tutorial.

Notice that the automatic points detection does not ensure to give the same order in the points of the two images. More generally, other salient points detectors do not give the same number of points, and thus the algorithms have to remove outliers (non matching points). This case is not taken into account in this tutorial.



## 25.4. Matlab correction



### 25.4.1 Transformation estimation based on corresponding points

#### Image visualization

The following code is used to display the images (see Fig.25.2).



```

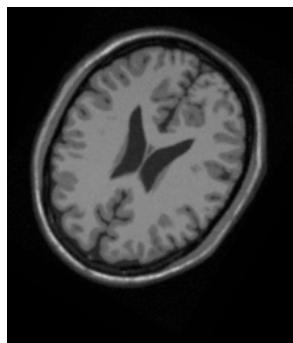
A = double(imread('Brain1.bmp'));
2 B = double(imread('Brain2.bmp'));

4 figure
subplot(131);viewImage(A);title ('moving image');
6 subplot(132);viewImage(B);title ('source image')
subplot(133),imshowpair(A,B),title ('superimposition');

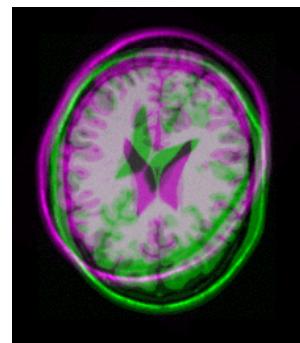
```



(a) Moving image.



(b) Source image.



(c) Superimposition.

Figure 25.2: Initial images.

If you want to save the fusion of both images, you can extract a frame, convert it into an image and save it:



```

1 figure; imshowpair(Atrans,B);
frame = getframe();
3 imageFusion = frame2im(frame);
imwrite(imageFusion,'imageFusionReg.png');
5 close;

```

## Manual selection of corresponding points

You just have to use the Matlab function `cpselect`:



```
[A_points, B_points] = cpselect (A/255,B/255);
```

## Transformation estimation

The rigid transformation is estimated from the corresponding points by the following function (take care of the confusion between  $x$  and  $y$  coordinates, which is handled by `flplr` or `flipud`):



```
function [R, t] = rigid_registration (data1, data2)
% Rigid transformation estimation between n pairs of points
% This method finds a rotation R and a translation t
% data1 : array of size nx2
% data2 : array of size nx2
%
% Convert pixels coordinates into x,y coordinates
data1_inv = flplr (data1);
data2_inv = flplr (data2);
%
% computes barycenters, and recenters the points
m1 = mean(data1_inv);
m2 = mean(data2_inv);
data1_inv_shifted = data1_inv - m1;
data2_inv_shifted = data2_inv - m2;
%
% Evaluates SVD
K = data2_inv_shifted * data1_inv_shifted ;
[U,~,V] = svd(K);
%
% Computes Rotation
S = eye(2);
S (2,2) = det(U)*det(V);
R = U*S*V';
%
% Computes Translation
t = flipud ( m2' - R*m1' );
```

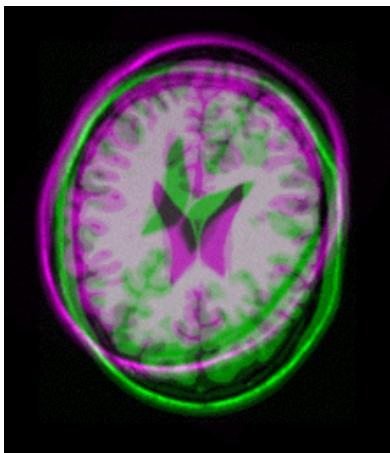
Then, you can apply this function to the manually selected points:



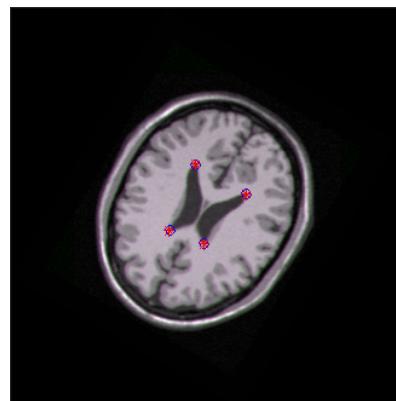
```
1 %% Transformation estimation
% If coming from the previous cpselect method, the points will give an
3 % almost perfect transformation.
[R, t] = rigid_registration (A_points, B_points);
5
% if you want to evaluate the angle of rotation:
7 % angle_rotation = acos(R(1,1))*180/pi*sign(R(1,2))
xform = [R ,0;0; t ',1];
9 tform_rigid = affine2d (xform);

11 % Matlab solution to evaluate the transformation. Notice that this is not
% exactly the same transformation, as the previous one is rigid, and this
13 % one allows a scaling factor.
%tform_rigid= fitgeotrans (A_points,B_points,' nonreflectivesimilarity ');
15
% Transformation
17 [Atrans, Rtrans] = imwarp(A, imref2d(size (A)), tform_rigid);

19 % Transform of the control points
A_points_trans = R * flipud (A_points') + flipud (t);
```



(a) Without registration.



(b) With registration.

Figure 25.3: Result of the registration for the manually selected control points.

The result is good, because the manual selection of the points is almost perfect.

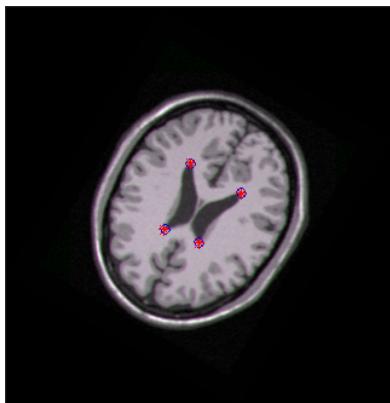
## 25.4.2 ICP registration

### Random permutation of points

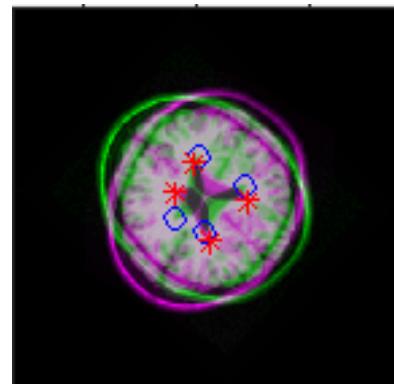
The following code randomly shuffles the points of the first vector. The result is of course a wrongly registered image, see Fig.25.4.



```
p = randperm(length(A_points));
2 A_points = A_points(p,:);
```



(a) Matching pairs of points.



(b) Permutation of the points.

Figure 25.4: Result of the registration for when the control points are not found in the same order.

### ICP

The previous operations simulates a general non-manual selection of the control points: there is no reason to finding the points by matching pairs. Thus, a reordering is necessary. This proposition implies that the number of points is exactly the same in order to perform the registration process, and that these are matching points (every point has a matching point in the other image). The ICP method (see code for `icp_transform`) reorders the points via a nearest neighbor rule.



```
function tform = icp_transform(dataA, dataB)
2 % Find a transform between A and B points
% with an ICP ( Iterative Closest Point) method.
4 % dataA and dataB are of size nx2, with the same number of points n
% returns a tform affine2d object
6
7 data2A = dataA;
8 data2B = zeros( size (data2A));
```



```
tform=affine2d(eye(3));
10
nb_loops=10;
12 data_loop = cell(nb_loops,1);
data_loop{1} = data2A;
14
for loop =2: nb_loops
16 % search for closest points and reorganise array of points accordingly
    [corr,~] = dsearchn(dataB, data2A);
18 for j = 1:length(corr)
        data2B(j ,:) = dataB(corr(j) ,:);
    end
20

22 % find rigid registration with reordered points
[R_loop, t_loop] = rigid_registration (data2A, data2B);
24 tform_loop = affine2d ([R_loop ,[0;0]; t_loop ',1]);
26
[X,Y]=tform_loop.transformPointsForward(data2A (:,1) ,data2A (:,2) );
data2A=[X,Y];
28 tform = affine2d (tform_loop.T * tform.T);

30 data_loop{loop} = data2A;
end
```

This function is then applied to the manually selected points, in random order.

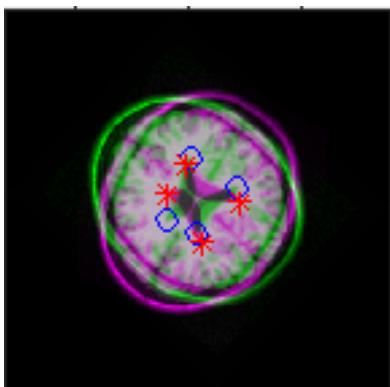


```
1 %% ICP-based image registration
% uses iterative control points algorithm, with reorganisation of points
3 % case with manual selection of points
% remember that A_points have been shuffled randomly
5 tform = icp_transform(A_points, B_points);

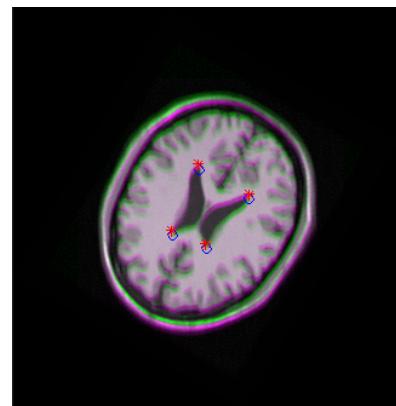
7 [Atrans, Rtrans] = imwarp(A, imref2d(size(A)), tform);
[X,Y]=tform.transformPointsForward(A_points (:,1) ,A_points (:,2) );
9 A_points_trans = [Y, X]';
```

### Automatic extraction of corner points

Generally, the points are automatically detected, and thus, there is no warranty that they are found in the same order, nor that each pair of point correspond to matching points (some points –called outliers– need to be eliminated to compute the correct transformation). In this tutorial, we do not address the problem of outliers. Please notice that with these parameters and images, by chance, the same points are detected in the correct order.



(a) Random shuffle of points and direct rigid transformation estimation.



(b) ICP registration on the same points.

Figure 25.5: Result of the registration for when the control points are not found in the same order. The ICP algorithm reorders the points and gives a good result.



```

1 %% Automatic extraction of corner points
% Unfortunately, this is not possible to have an interactive selection of
3 % the control points.
% Harris corners detection does not ensure to give the same exact points,
5 % in the same order.
cornersA = detectHarrisFeatures (A, 'FilterSize ', 7);
7 cornersB = detectHarrisFeatures (B, 'FilterSize ', 7);

9 nb_points = 4;
figure
11 subplot (121); imshow(A,[]); title ('moving image');
hold on; plot(cornersA. selectStrongest (nb_points))
13 subplot (122); imshow(B,[]); title ('source image');
hold on;
15 plot(cornersB. selectStrongest (nb_points));

```

You can pass the detected points to the rigid registration method by using:



```

1 A_points = cornersA. selectStrongest (nb_points). Location);

```

The Fig.25.6 displays the 20 strongest points for both images. Notice that there is no correspondance in these points. If you apply the previous code with these 20 points, there is a high chance that the registration will be wrong.

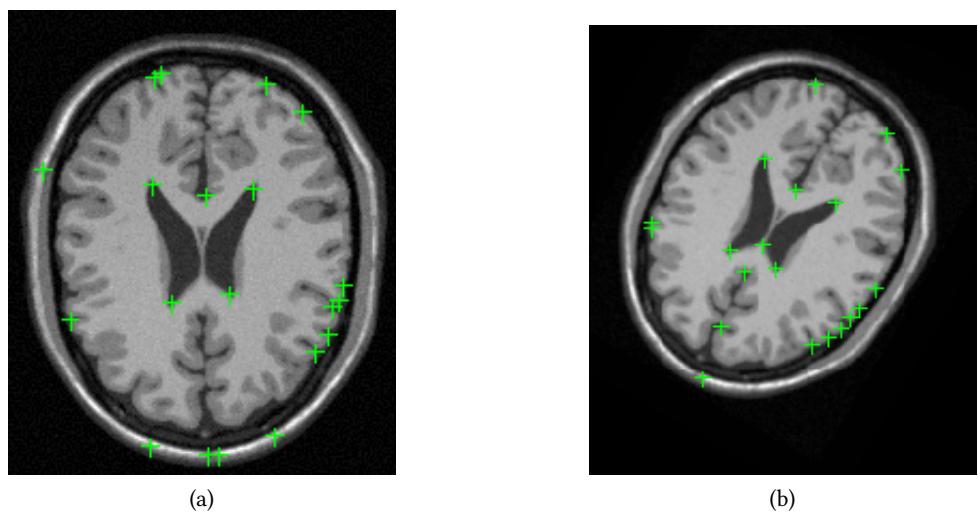


Figure 25.6: Harris corners detection. 20 points are represented.



## **Part IV Stochastic Analysis**





## 26 Stochastic Geometry / Spatial Processes

This tutorial aims to simulate different spatial point processes.

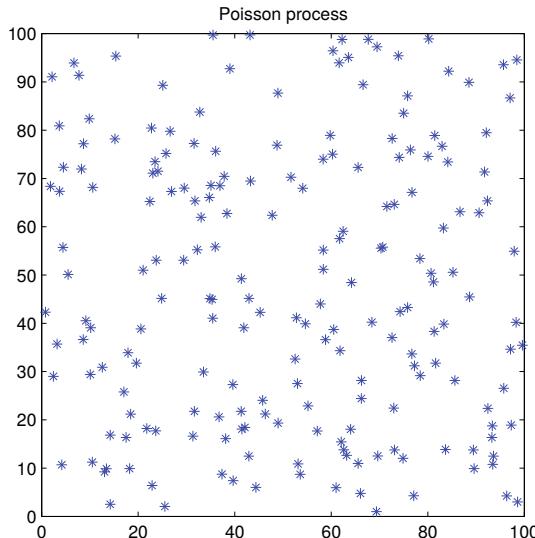
### 26.1

### Poisson processes

This process simulates a conditional set of  $point_{nb}$  points in a spatial domain  $D$  defined by the values  $x_{min}, x_{max}, y_{min}, y_{max}$ . In order to simulate a non conditional Poisson point process of intensity  $\lambda$  within a domain  $S$ , it is necessary to generate a random number of points according to a Poisson law with the parameter  $\lambda S$  :  $point_{nb} = poisson(\lambda S)$  (i.e. the number of points is a random variable following a Poisson Law).

The coordinates of each point follow a uniform distribution.

Figure 26.1: Conditional Poisson point process, with 200 points.



1. Simulate a conditional Poisson point process on a spatial domain  $D$  with a fixed number of points (see Fig. 26.1).
2. Simulate a Gaussian distribution of points around a given center.

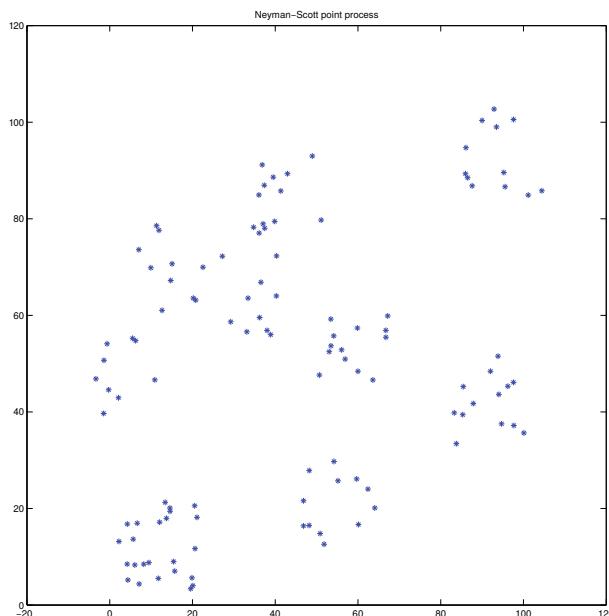


The function `poissrnd` can be used to generate a random variable following a Poisson law. Use `rand` for generating uniform distribution random variables.

## 26.2 Neyman-Scott processes

This process simulates aggregated sets of points within a spatial domain  $D$  defined by the values  $x_{min}, x_{max}, y_{min}, y_{max}$ . For each aggregate ( $n_{par}$ ), we first generate the random position of the 'parent' point. Then, the 'children' points are simulated in a neighborhood (within a square box of size  $r_{child}^2$ ) of the 'parent' point. The number of points for each aggregate is either fixed or randomized according to a Poisson law of parameter  $n_{child}$  (see Fig. 26.2).

Figure 26.2: Neyman-Scott point process with  $h_{child} = 10$  and  $n_{par} = 10$



1. Simulate a process of 3 aggregates with 10 points.
2. Simulate a process of 10 aggregates with 5 points.

## 26.3

# Gibbs processes

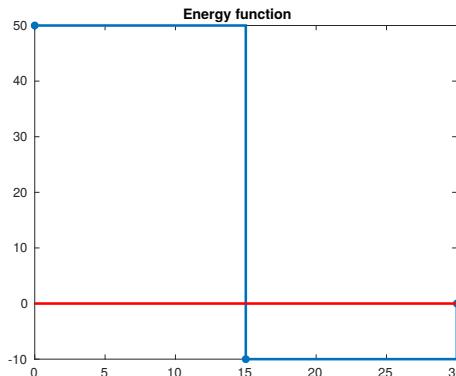
The idea of Gibbs processes is to spatially distribute the points according to some laws of interactions (attraction or repulsion) within a variable range.

Such a process can be defined by a cost function  $f(d)$  that represents the cost associated to the presence of 2 separated points by a distance  $d$  (see Fig. 26.3). For a fixed value  $r$ , if  $f(d)$  is negative, there is a high probability to find 2 points at a distance  $d$  (attraction). Conversely, if  $f(d)$  is positive, there is a weak probability to find 2 points at a distance  $d$  (repulsion).

- Code such a function, with prototype `function e=f(d)` or `def energy(d):`, where

$$f(d) = \begin{cases} 50 & \text{if } 0 < d \leq 15 \\ -10 & \text{if } 15 < d \leq 30 \\ 0 & \text{otherwise} \end{cases}$$

Figure 26.3: Energy function  $f$ .

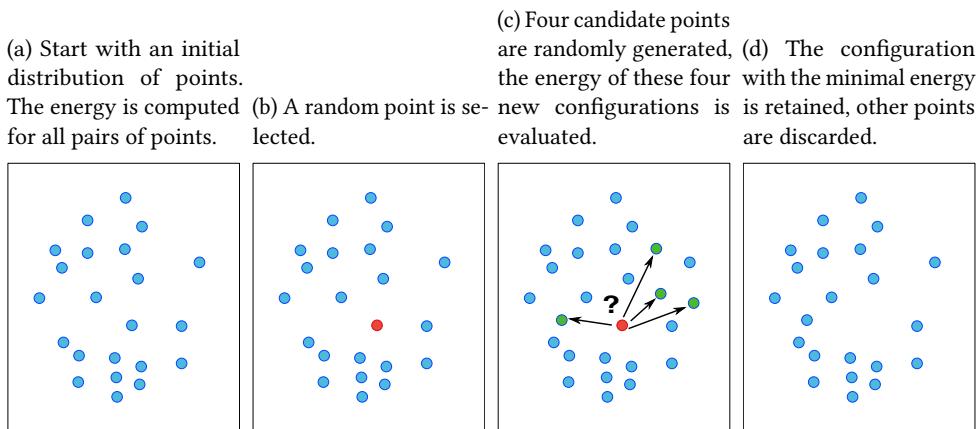


The generation process reorganizes an initial Poisson point process within the spatial domain according to a specific cost piecewise constant function. The reorganization consists in a loop of  $nb_{iter}$  iterations. For each iteration, we calculate the total energy:

$$e = \sum_{(i,j), i \neq j} f(\text{dist}(x_i, x_j)) \quad (26.1)$$

The objective is to reduce this energy iteratively. At each iteration step, we try to replace a point by four (for example) other random points and we calculate the energy for each new configuration. The initial point is either preserved or replaced (by one of the four points) according to the configuration of minimal energy (see Fig. 26.4).

Figure 26.4: Illustration of iterative construction of the Gibbs point process.



1. Code a function with the following prototype (the function *energyFunction* is previously noticed *f*). It computes the energy between all the points present in an array of coordinates  $[x, y]$  (the points that do not move) and point  $[x_k, y_k]$  (the new point). The *energyFunction* converts a distance to an 'energy', reflecting attractivity or repulsivity.



```
function e = energy(energyFunction, x, y, xk, yk)
```

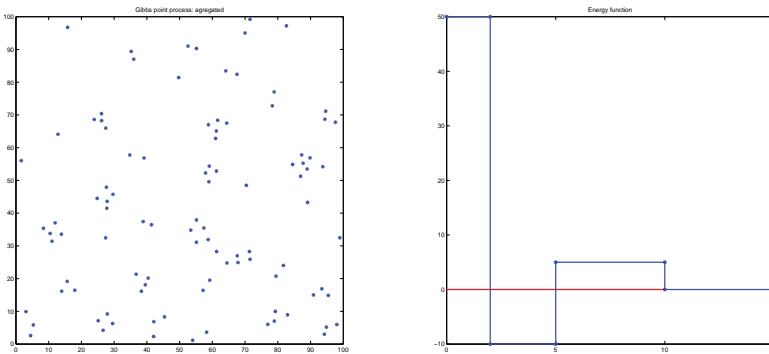
2. Simulate a regular point process by choosing a specific energy function.
3. Change the cost function and simulate a few aggregated point processes (see Fig. 26.5).



Informations

Use the function *pdist*, which computes pairwise distances of all points. You may also consider the *pdist2* function.

Figure 26.5: Gibbs aggregated point process and its energy function.



## 26.4 Ripley function

The Ripley function  $K(r)$  characterizes the spatial distribution of the points. For a Poisson process of density  $\lambda$ ,  $\lambda K(r)$  is equal to the mean value of the number of neighbors at a distance lower than  $r$  to any point. In the case where the process is not known ( $\lambda?$ ), the Ripley function has to be estimated (approximated) with the unique known realization. It is the first estimator of  $K(r)$ :

$$K(r) = \frac{1}{\lambda} \frac{1}{N} \sum_{i=1}^N \sum_{j \neq i} k_{ij} \quad (26.2)$$

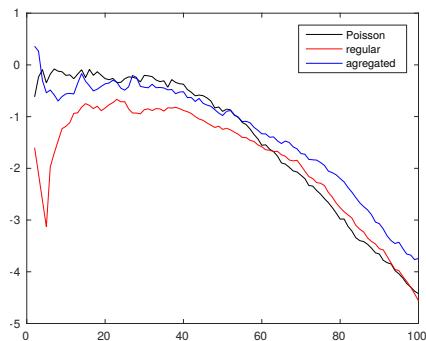
where  $N$  is the number of points in the studied domain  $D$ ,  $\lambda = N/D$  is the estimator of the density of the process and  $k_{ij}$  takes the value 1 if the distance between the point  $i$  and the point  $j$  is lower than  $r$ , and 0 in the other case.

We denote:

$$L(r) = \sqrt{K(r)/\pi} \quad (26.3)$$



1. Code a function to compute  $K$  (`function K=ripley(points, box, r)`), with `points` being the considered points, `box` the simulation domain, and `r` the distance (or an array of all distances).
2. Calculate the Ripley function for an aggregated point process, a Poisson point process and a regular point process.
3. Display the value  $L(r) - r$  as in Fig 26.6.

Figure 26.6: Ripley's function  $L(r) - r$ .

## 26.5 Marked point processes

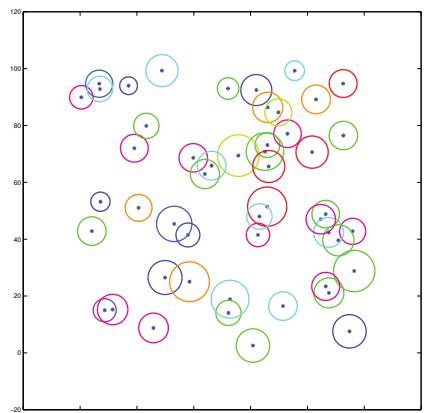
To simulate a complex point process, it can be useful to associate several random variables (marks) for each point.



1. Simulate a disk process, where the points (disk centers) are defined according to a Poisson process and the radii are selected with a Gaussian law.
2. Add a second mark for allocating a color to each disk (uniform law).

An example result is shown in Fig. 26.7

Figure 26.7: A Poisson point process is used to generates the center of the circles. The radii are chosen according a Gaussian law, and the color according a uniform law.





## 26.6. Matlab correction



### 26.6.1 Poisson and other classical processes

To generate a conditional Poisson process, the following command is used:



```
n=200;
2 [x, y] = semi_alea(n, 0, 0, 100, 100);
plot(x,y,'*'); title ('Conditional Poisson process');
4 axis square
```

The simulation window is given as a parameter, as well as the number of points. This is illustrated in Fig.26.1



```
function [x,y]=semi_alea(point_nb,xmin,xmax,ymin,ymax)
2 % Conditional Poisson Point process
% uniform distribution
4 % point_nb is the number of points
% xmin, xmax, ymin, ymax define the domain
6 x = xmin + (xmax-xmin)*rand(point_nb, 1);
y = ymin + (ymax-ymin)*rand(point_nb, 1);
```

To generate a point cloud normally distributed around the point  $(a, b)$ , one can use the Matlab function `randn`. This is illustrated in Fig.26.8.



```
% Normal distribution , centered around (a,b)
2 a=0;
b=0;
4 x = a + 50 * randn(200,1);
y = b + 50 * randn(200,1);
6 figure();
plot(x, y, '*'); title ('Std normal distribution ');
8 axis square
```

### 26.6.2 Neyman-Scott processes

The Neyman-Scott process is a way of simulating clustered processes located around root points issued from a root point process.

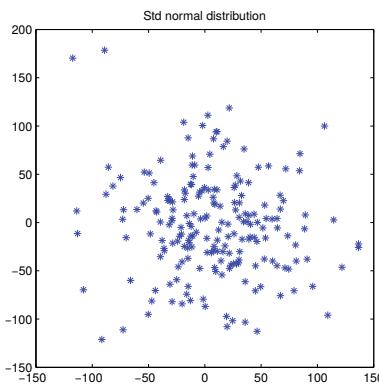


Figure 26.8: Normal standard distribution of the point process around coordinates  $(0, 0)$ .



```

1 function [x,y]=semi_NS(nRoot, xmin, xmax, ymin, ymax, lambdaS, rSon)
2 % Neyman–Scott process simulation
3 % i.e. aggregate of processes
4 % nRoot : number of aggregates
5 % lambdaS : number of points, lambda is a density, S is the spatial support area
6 % rSon : radius around aggregate (points are distributed in a square)

7 % should generate the number of sons
8 n = poissrnd(lambdaS, nRoot, 1);
9
10 % Allocation of memory
11 nb = sum(n(:));
12 x=zeros(nb,1);
13 y=zeros(nb,1);

14 indice=1;
15 % loop over all aggregates
16 for i=1:nRoot
17     xr=xmin+randi(xmax-xmin); % root point
18     yr=ymin+randi(ymax-ymin);
19     for j=1:n(i)
20         dx=randi(2*rSon)-rSon;
21         dy=randi(2*rSon)-rSon;
22
23         x(indice)=xr+dx;
24         y(indice)=yr+dy;
25         indice = indice + 1;
26     end
27 end

```

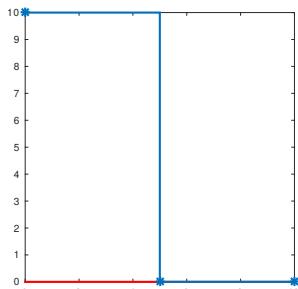
The result is given by the following command and illustrated in Fig.26.2.



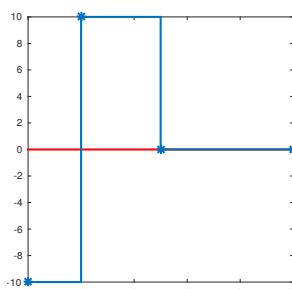
```
1 [x,y]=semi_NS (3,0,100,0,100,20,10) ;
plot(x,y,'*'); title ('Neyman–Scott point process');
```

### 26.6.3 Gibbs point processes

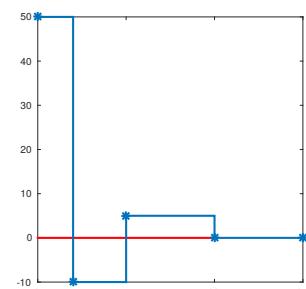
The Gibbs point process uses the definition of an energy in order to attract or repulse points. First of all, the energy function is coded with the following function stairsEnergy. Its results are illustrated in Fig.26.9.



(a) Energy function  $f$  for regular distribution.



(b) Energy function  $f$  for aggregated distribution.



(c) Energy function  $f$  for aggregated distribution.

Figure 26.9: Different energy functions.

The following code is used to evaluate the energy. The different steps are passed as arrays of values.



```
function e = stairsEnergy (distance , steps , energy)
2 % This function returns e with same size as distance .
% e takes the value given in variable energy according to the steps
4 % distance : vector of all distances between points
% steps    : steps for energy function (unit: distance )
6 % energy   : value of energy for each step
e = zeros( size (distance )) ;
8
prev_step = 0;
10 for i=1:length(steps)
    e(distance >=prev_step & distance<steps(i)) = energy(i);
12 prev_step=steps(i);
end
```

To generate the Gibbs process, the following code can be used. It is illustrated in Fig.26.10.



```

1 figure
2 subplot(241)
3 [x1,y1]=semi_alea (100,0,100,0,100) ;
4 plot(x1,y1,'*'); title ('Poisson point process');
5 axis square

7 subplot(242)
8 steps = [0,5,10];
9 energy= [10,0,0];
10 functionEnergy = @(x)stairsEnergy(x,steps, energy);
11 [x2,y2]=semis_inter (100,0,100,0,100,2000, functionEnergy);
12 plot(x2,y2,'*'); title ('Gibbs point process: regular');
13 axis square
14 subplot(246)
15 plot (0:10, zeros (1,11) , 'r', 'linewidth' ,2); hold on;
16 stairs (steps, energy,'*—', 'linewidth' ,2); title ('Energy function')
17 axis square

19 subplot(243)
20 steps = [0,2,5,10];
21 energy= [-10,10,0,0];
22 functionEnergy = @(x)stairsEnergy(x,steps, energy);
23 [x3,y3]=semis_inter (200,0,100,0,100,200, functionEnergy);
24 plot(x3,y3,'*'); title ('Gibbs point process: aggregated');
25 axis square
26 subplot(247)
27 plot (0:10, zeros (1,11) , 'r', 'linewidth' ,2); hold on;
28 stairs (steps,energy,'*—', 'linewidth' ,2); title ('Energy function')
29 axis square

31 subplot(244)
32 steps = [0,2,5,10,15];
33 energy= [50,-10,5,0,0];
34 functionEnergy = @(x)stairsEnergy(x,steps, energy);
35 [x4,y4]=semis_inter (100,0,100,0,100,200, functionEnergy);
36 plot(x4,y4,'*'); title ('Gibbs point process: aggregated');
37 axis square
38 subplot(248)
39 plot (0:10, zeros (1,11) , 'r', 'linewidth' ,2); hold on;
40 stairs (steps, energy,'*—', 'linewidth' ,2); title ('Energy function')
41 axis square

```

The principle of the algorithm is to iteratively add one point that minimizes the energy after several trials. In order to speed up the process, notice that only one point is moved, and it is thus sufficient to only compute the distances from this point to all others.

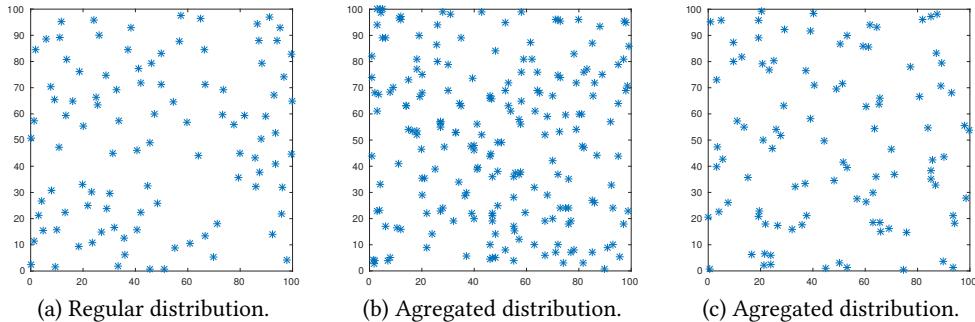


Figure 26.10: Gibbs process with different energy functions.



```

1 function [x,y]=semis_inter(point_nb, xmin,xmax, ymin,ymax, nbiter, energyFunction)
% simulation of a Gibbs point process
3 % point_nb is the number of points to plot
% xmin and xmax define the X domain of points
5 % ymin and ymax define the Y domain of points
% nbiter is the number of iterations , i.e. the number of times a point will be
    ↪ given a try to move
7 %
% energyFunction is an optional argument defining a function that takes a distance
    ↪ as a parameter and returns an energy (see example at the end of this
    ↪ code, with the default value of this function).
9 if nargin < 7
    energyFunction = @exampleEnergyFunction;
11 end

13 rng(0)
% Start with a poisson point process
15 [x,y]=semi_alea(point_nb,xmin,xmax,ymin,ymax);
%plot(x, y, 'b*'); hold on
17 % The variable indices is used to select a point
indices = 1:length(x);

19 for i=1:nbiter
    % choose a random point among previous process
    j=randi(length(x));
23
    % all points except jth
25 x2 = x(indices~=j);
    y2 = y(indices~=j);

27 % compute the energy associated to the point j
e1 = energyFromPoint(energyFunction, x, y, x(j), y(j));

31 % try to minimize energy with new points

```



```

for m=1:10
    % new point
    [xx,yy] = semi_alea (1, xmin, xmax, ymin, ymax);
35
    e2 = energyFromPoint(energyFunction, x2, y2, xx, yy);
    % the new point is kept if energy is less than previous configuration
    if e2<e1
        x(j)=xx;
        y(j)=yy;
        e1=e2;
41
    end
43    end
end % end for
45 end % end function

```

The next functions evaluate the total energy of the point process. The main argument is the set of all distances  $D$  between all pairs of points.



```

1 function e = energy(energyFunction, D, k)
% compute the energy between all the distances D and the point k
3 %
% if xx and yy are provided, this new point tries to replace the jth point
5 dist = sqrt(D(k,:).^2); % Euclidean distance
ee = energyFunction(dist );
7 e = sum(ee);

9 end

```



```

1 function e = energyNewPoint(energyFunction, x, y, xx, yy)
% compute the energy between all points and the new point
3 dist = sqrt ((x-xx).^2+(y-yy).^2);
ee = energyFunction(dist );
5 e = sum(ee);
end

```



```

function e = exampleEnergyFunction(distance)
2 % Example of energy function
% Takes a distance as a parameter
4 % Returns value of energy.
%

```

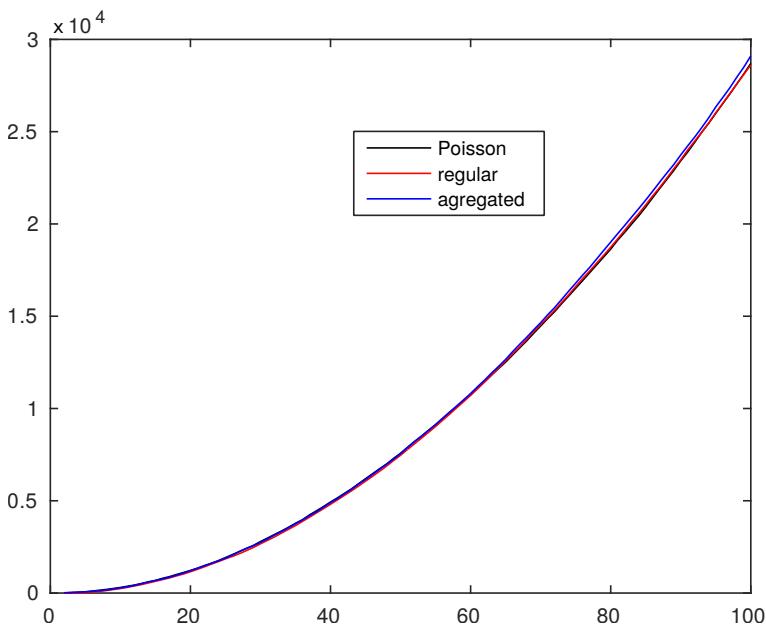


Figure 26.11: Ripley's K function for 3 different processes.



```

6 % A negative energy means that the points are attracted .
7 % A positive energy means a repulsion of the points .
8 e=zeros( size( distance ) );
9 e( distance <10 ) = 10;
10 e( distance <20 ) = -50;
11 end

```

## 26.6.4 Ripley functions

The code for the ripley function uses the MATLAB® function histcounts for efficiency. The bottleneck in algorithm is the computation of all pairs distances, efficiently done by the pdist function. Notice that this function is biased because points in border of window are counted as points in the center. This could be corrected by the use of pdist2. The results are illustrated in Fig.26.11 and Fig.26.6.



```

function [K, L, vals]=ripley (x,y,xmin,xmax,ymin,ymax,edges)
2 % Ripley K and L functions
3 % x and y define the points abscisses and ordinates
4 % xmin xmax ymin ymax define the domain
5 % edges is a vector containing the distances ( typically , r=1:100 for example)

```



```

6 % K: K function
% L: L function
8 % vals : values of radius

10 % number of points
nb_points=length(x);

12 % area
area = (xmax-xmin)*(ymax-ymin);

14 % computes the distances between all pairs of points
d = pdist ([x,y]);

16 % histcounts (d, edges, 'Normalization', 'cumcount');
h = histcounts (d, edges, 'Normalization', 'cumcount');

18 % count mean number of points
20 % multiply by 2 because each distance is only counted once in g
K = 2*h/nb_points;

22 % estimates density
24 densite=nb_points/area;
K=K/densite;

26 L = sqrt (K/pi);

28 vals = edges(1:end-1)+diff(edges);

30 end % of function
32

```

The code to generate these results is:



```

r =1:100;
2 xmin=0; xmax=1000; ymin=0; ymax=1000;
nb_points=2000;
4 [x,y] = semi_alea(nb_points, xmin, xmax, ymin, ymax);
[k1,l1 , vals]=ripley (x,y,xmin, xmax, ymin, ymax, r);

6 steps = [0,5,10];
8 energy= [10,0,0];
functionEnergy = @(x)stairsEnergy(x, steps , energy);
10 [x2,y2]=semis_inter(nb_points, xmin, xmax, ymin, ymax, 2000, functionEnergy);
[k2,l2 , vals]=ripley (x2,y2, xmin, xmax, ymin, ymax, r);

12

14 steps = [0,2,5,10];
energy= [-10,10,0,0];
16 functionEnergy = @(x)stairsEnergy(x, steps , energy);
[x3,y3]=semis_inter(nb_points, xmin, xmax, ymin, ymax, 2000, functionEnergy);
18 [k3,l3 , vals]=ripley (x3,y3, xmin, xmax, ymin, ymax, r);

```



```
20 % figures
21 figure
22 plot(vals,l1-vals,'k-');hold on;
23 plot(vals,l2-vals,'r-');hold on;
24 plot(vals,l3-vals,'b-');hold on;
25 legend('Poisson','regular','aggregated');
26 title ('Ripley L functions');

27 figure
28 plot(vals,k1,'k-'); hold on
29 plot(vals,k2,'r-');
30 plot(vals,k3,'b-');

31 legend('Poisson','regular','aggregated');
32 title ('Ripley K function');
```

## 26.6.5 Marked point process



```
1 figure
2 subplot(121)
3 % generates Poisson process
4 nb_points=50;
5 xmin=0;
6 xmax=100;
7 ymin=0;
8 ymax=100;
9 [x,y]=semi_alea(nb_points,xmin, xmax, ymin, ymax);
10 plot(x,y, '*'); title ('Poisson process');
11 axis square
12 axis([-20,120,-20,120]);

13 % Generates radii with mean mu and stddev sigma
14 sigma=1;
15 mu = 5;
16 r = sigma*randn(length(x), 1) + mu;
17
18 % generates plot with disks
19 subplot(122)
20 plot(x,y, '*');
21 hold on
22 theta =0:0.01:2* pi;
23 xx = zeros(length(theta), 1);
24 yy = xx;
25
```



```
% the second mark (color) can be generated as this
28 nb_colors = 10;
    m2 = randi(nb_colors, nb_points, 1);
30 cmap = hsv(nb_colors);

32 for i=1:length(x)
    xx=x(i)+r(i)*cos(theta);
34     yy=y(i)+r(i)*sin(theta);
        plot(xx,yy,'LineWidth', 2, 'Color', cmap(m2(i),:));
36 end

38 axis square
axis([-20,120,-20,120]);
```

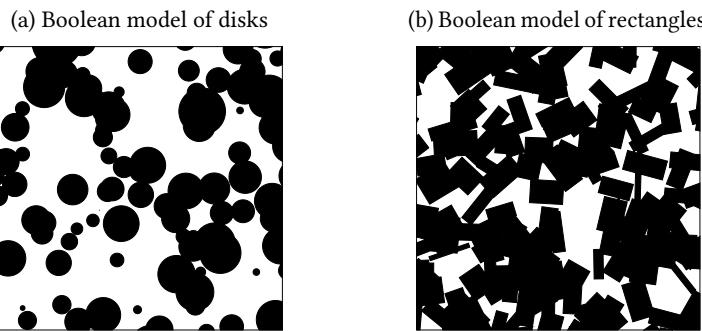
An example result is shown in Fig.26.7

## ★★★ 27

# Boolean Models

This tutorial aims to study a classical model coming from stochastic geometry: the Boolean model. The first objective is to simulate some realizations of a Boolean model of 2-D disks representing a population of overlapped particles. Thereafter, the geometrical characteristics of the individual disks (from a statistical point of view) will be analyzed.

Figure 27.1: Illustration of 2-D Boolean models observed in a squared window  $W$ .



## 27.1

# Simulation of a 2-D Boolean model

Let  $\{x_i; i \in \mathbb{N}\}$  be a random collection of points in  $\mathbb{R}^2$  forming a stationary Poisson point process with intensity  $\gamma > 0$ . Let  $Z_0, Z_1, Z_2, \dots$  be independent, identically distributed random 2-D convex bodies (nonempty, compact, convex sets) with distribution  $\mathbb{Q}$ , which are independent of the point process  $\{x_i; i \in \mathbb{N}\}$ . The random points  $x_1, x_2, \dots$  are the germs and the random sets  $Z_1, Z_2, \dots$  are the grains of the Boolean model. The random set  $Z_0$  is called the typical grain. The union of the translated grains:

$$Z = \bigcup_{i=1}^{\infty} (Z_i + x_i) \quad (27.1)$$

is a random closed set, which is called the stationary Boolean model with intensity  $\lambda$  and grain distribution  $\mathbb{Q}$ . The random collection  $X = \{Z_1 + x_1, Z_2 + x_2, \dots\}$  of the shifted grains is the particle process underlying the Boolean model.

Figure 27.1 shows a realization of two different Boolean models  $Z$  observed in a compact and convex observation window  $W$ .

We are going to simulate some realizations of a Boolean model of 2-D disks in a squared observation window. The final simulated model will be represented as a discrete binary image.



- Generate a 2-D discrete observation window  $W$  of size  $500 \times 500$ .
- Generate the random germs that follows a Poisson law with intensity  $\lambda = 100/(500 * 500)$ . Take care of the edge effects (a grain with a germ outside the observation window could intersect it!)
- Generate the random grains (as disks). The disk radius will follow a uniform distribution  $\mathcal{U}(10, 50)$ .
- Vizualize the realization.



Look at the MATLAB® function `poissrnd` and `randi`.  
The `meshgrid` function can be used to generate the disks.

## 27.2 Geometrical characterization of a 2-D Boolean model

Assume we observe  $Z$  in a compact, convex observation window  $W$  with positive volume (as shown in Figure 27.1). Our aim is to extract distributional information from the geometric properties of the sample  $Z \cap W$ . Thus, we assume we can measure geometric functionals like the perimeter of the sample  $Z \cap W$  which is a finite union of convex bodies (a polyconvex set). By a geometric functional  $\phi$  we mean a real-valued functional defined on the space of polyconvex sets with specific additional properties. Important examples of geometric functionals are the Minkowski functionals  $W_\nu$  that are related in the 2-D space to the measures of area ( $A$ ), perimeter ( $P$ ) and Euler number ( $\chi$ ):

$$W_0 = A \quad (27.2)$$

$$W_1 = P/2 \quad (27.3)$$

$$W_2 = \pi\chi \quad (27.4)$$

The boundary of the observation window  $W$  has a disturbing effect. It is therefore of advantage to assume a sufficiently large observation window and to consider only limits as the window tends to infinity. This motivates the introduction of the density (or specific value) of the Boolean model for a geometric functional  $\phi$ . The density of  $\phi$  is the combined spatial and probabilistic mean value:

$$\bar{\phi}(Z) = \lim_{r \rightarrow \infty} \frac{\mathbb{E}[\phi(Z \cap rW)]}{W_0(rW)} \quad (27.5)$$

The crucial problem when studying a Boolean model is, that the particles overlap and can therefore not be observed individually.

For this purpose, the Miles formula gives relationships between the global Minkowski densities and the Minkowski densities of the particle. For isotropic Boolean models and by using the densities of the particle process  $X$ :

$$\bar{\phi}(X) = \gamma \mathbb{E}[\phi(Z_0)] \quad (27.6)$$

Miles formulas express the observable Minkowski densities  $\bar{W}_\nu(Z)$  in terms of the Minkowski densities  $\bar{W}_\nu(X)$  and can be inverted.

For example in 2-D:

$$\bar{W}_0(Z) = 1 - e^{-\bar{W}_0(X)} \quad (27.7)$$

$$\bar{W}_1(Z) = e^{-\bar{W}_0(X)} \bar{W}_1(X) \quad (27.8)$$

$$\bar{W}_2(Z) = e^{-\bar{W}_0(X)} (\bar{W}_2(X) - \bar{W}_1(X)^2) \quad (27.9)$$



- Generate different realizations of the previous Boolean model and compute the Minkowski densities of  $Z$  (by using the functions done in the tutorial "Integral Geometry").
- Compute the theoretical Minkowski densities of  $Z$  by using the Miles formulas.
- Compare the computed and theoretical values.



## 27.3. Matlab correction



### 27.3.1 Simulation of a 2-D Boolean model

The process for simulating the proposed Boolean model of 2-D disks consists in four steps:

- Generate the random number of points (by using the intensity parameter of the Poisson distribution). In order to avoid edge effects, one consider a larger window than the observation window to generate the disks. Indeed, a germ outside the observation window can generate a disk that intersects the observation window.
- Generate the random locations of the germs (random coordinates from a uniform distribution)
- Generate the random size of the disks (random radius from a probability distribution)
- Generate the union of disks (by using the dilation of the germs with a disk of the corresponding radius as structuring element)

Here is the global function for generating such a Boolean model as a binary image:



```

function Z = BooleanModel(Wsize, Gamma, RadiusParam)
2 % Generates a boolean model of disks in 2D. The number of disks is chosen
    % according to a Poisson law of parameter lambda = Gamma*areaW, where
    % areaW is the area of the window defined by Wsize.
%
4 % Wsize: size of the window (2x1 array)
% Gamma: Parameter to get the density for the Poisson law
6 % RadiusParam: [min, max] values of radii
% returns: boolean array of size Wsize(1)xWsize(2)
8
    edgeEffect = 2*max(RadiusParam)+100;
10 WsizeExtended = [Wsize(1)+2*edgeEffect , Wsize(2)+2*edgeEffect ];
    % nb of points
12 nf = WsizeExtended(1);
    nc = WsizeExtended(2);
14 areaW = nf*nc;
    nbPoints = poissrnd(Gamma*areaW);
16 % germs
    x = randi(nf, nbPoints);
18 y = randi(nc, nbPoints);
    % grains
20 r = randi(RadiusParam, nbPoints);

```



```

Z = false (nf,nc);
22
% union of grains
24 [X, Y] = meshgrid(1:nf, 1:nc);
for i = 1:nbPoints
26 Z = Z | ((X-x(i)).^2+(Y-y(i)).^2) <= r(i)^2;
end
28 Z = Z(edgeEffect +1: edgeEffect +Wsize(1), edgeEffect +1: edgeEffect +Wsize(2));

```

When executing this function with the following parameters:



```

% parameters
2 Wsize = [500 500];
Gamma = 100/Wsize(1)/Wsize(2);
4 RadiusParam = [10 30]; % uniform law between 20 and 50

6 % generation
Z = BooleanModel(Wsize, Gamma, RadiusParam);
8
% visualization
10 imshow(Z);

```

We get a realization of this Boolean model as a binary image in Fig.27.2.

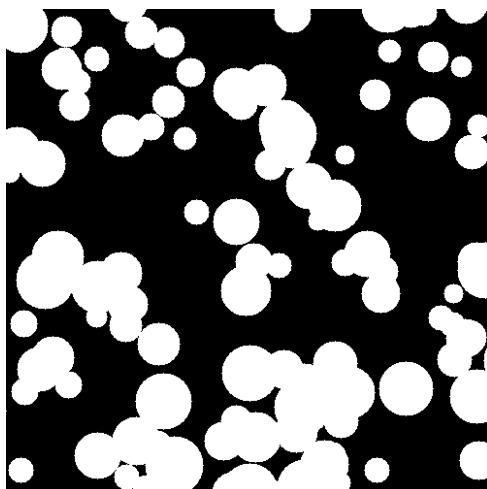


Figure 27.2: Boolean model of disks, with  $Wsize = [500, 500]$  and  $RadiusParam = [10, 30]$ .

### 27.3.2 Geometrical characterization of a 2-D Boolean model

We can use the following function to compute the Minkowski functionals of the Boolean model (see the tutorial on Integral Geometry):



```

function [Area, Perimeter, EulerNb4, EulerNb8] = MinkowskiFunctionals(X)
2
% Neighborhood configuration
4 F = [0 0 0; 0 1 4; 0 2 8];
XF = conv2(double(X),F, 'same');
6 h = hist(XF(:,16));
%bar(0:15,h);
8
% Computation of the functionals
10 f_intra = [0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1];
e_intra = [0 2 1 2 1 2 2 2 0 2 1 2 1 2 2 2];
12 v_intra = [0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1];
EulerNb8 = sum(h.*v_intra - h.*e_intra + h.*f_intra);
14 f_inter = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0];
e_inter = [0 0 0 1 0 1 0 2 0 0 0 1 0 1 0 2];
16 v_inter = [0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1];
EulerNb4 = sum(h.*v_inter - h.*e_inter + h.*f_inter);
18 Area = sum(h.*f_intra);
Perimeter = sum(-4*h.*f_intra + 2*h.*e_intra);

```

So we can estimate the Minkowski densities on different realizations of the Boolean model:



```

1 % use of the Tutorial "Integral Geometry"
% computation of the Minkowski densities on different realizations
3 nbRealizations = 100;
W = zeros(nbRealizations,3);
5 areaWsize = Wsize(1)*Wsize(2);
for i = 1:nbRealizations
7 [Z] = BooleanModel(Wsize, Gamma, RadiusParam);
[area, per, ~, chi8] = MinkowskiFunctionals(Z);
9 W(i,:) = [area, per/2, chi8*pi]/areaWsize;
clear area per chi4 chi8;
11 end

```

Thereafter, we can compare the estimated Minkowski mean densities of the Boolean model with the theoretical ones (by using the known parameters of the different probability distributions of this Boolean model):



```
1 % mean densities (estimated)
W = mean(W,1);
3
% mean densities ( theoretical ) by using Miles formulas
5 rMean = (RadiusParam(1)+RadiusParam(2))/2;
AreaMean = pi*rMean^2;
7 PerMean = 2*pi*rMean;
W_X = Gamma * [AreaMean,PerMean/2,pi];
9 W_th(1) = 1 - exp(-W_X(1));
W_th(2) = exp(-W_X(1)) * W_X(2);
11 W_th(3) = exp(-W_X(1)) * (W_X(3)-W_X(2)^2);

13 % comparison
error_W0 = abs(W(1)-W_th(1))/W_th(1)
15 error_W1 = abs(W(2)-W_th(2))/W_th(2)
error_W2 = abs(W(3)-W_th(3))/W_th(3)
```

Here are the results for 100 specific realizations:

Command window

```
errorW0 = 0.0701
2 errorW1 = 0.2437
errorW2 = 0.4396
```

The errors can be large due to the bias estimation of the Minkowski densities within an observation window (specifically for the perimeter and the Euler number). But you can use unbiased estimators which can be found in the literature. Note that the Miles formulas can be inverted to estimate the Minkowski functionals of the typical grain from the Minkowski mean densities of the Boolean model.





## 28 Geometry of Gaussian Random Fields

This tutorial aims at simulating Gaussian random fields and analyzing their geometry.

### 28.1 Introduction

Let  $\phi$  be a real-valued stationary Gaussian Random Field (GRF) on  $\mathbb{R}^2$ .  $\forall p \in \mathbb{R}^2$ ,  $\phi(p)$  defines a random variable. The family  $(\phi(p), p \in \mathbb{R}^2)$  consists of identically distributed random variables on a probability space  $\Omega, \mathcal{F}, P$  that satisfies for any subset of  $\{p_1, \dots, p_n \in \mathbb{R}^2\}$  and  $\alpha_k \in \mathbb{R}$  that the random variable  $\sum_{k=1}^n \alpha_k \phi(p_k)$  is normally distributed. The reader could find more informations in [?, ?, ?].

Thus,  $\phi$  is completely characterized by its mean and its covariance:

$$m = \mathbb{E}(\phi(p)) = \mathbb{E}(\phi(0)), \quad p \in \mathbb{R}^2 \quad (28.1)$$

$$C(p) = \mathbb{E}(\phi(0)\phi(p)) - m^2 = \mathbb{E}(\phi(q)\phi(p+q)) - m^2, \quad p, q \in \mathbb{R}^2 \quad (28.2)$$

In this tutorial, we will assume  $m = 0$ . The goal is to construct  $\phi$  for a given covariance  $C$ .

### 28.2 Simulation

For more details on algorithms simulating GRFs in  $\mathbb{R}^d$ , see [?].

#### 28.2.1 Gaussian white noise random field

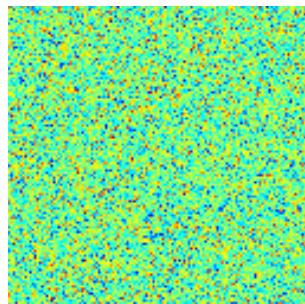


- Generate a white noise on  $\mathbb{R}^2$ . Choose an  $n \times n$  size. Display it on the screen (see for example Fig. 28.1).
- Modify the variance and observe the results.



Use `randn` and `imagesc` or `imshow`.

Figure 28.1: Gaussian White Noise, with  $m = 128$  and  $\sigma = 30$  for display purposes.



## 28.2.2 Gaussian Random Field

Let  $W$  be an independent centered white noise random fields.  $\mathcal{F}$  is the Fourier Transform. The gaussian random field  $\phi$  is defined by:

$$p \in \mathbb{R}^2, \phi(p) = \mathcal{F}^{-1}(\hat{\phi})(p)$$

and

$$\hat{\phi}(k) = \sqrt{\mathcal{F}(C)(k)\mathcal{F}(W)}$$



- Generate a 2D Gaussian covariance function , see Fig. 28.2a (we recall the definition,  $u^2 = x^2 + y^2$ ):

$$C(u) = e^{-\frac{u^2}{2\sigma^2}}$$

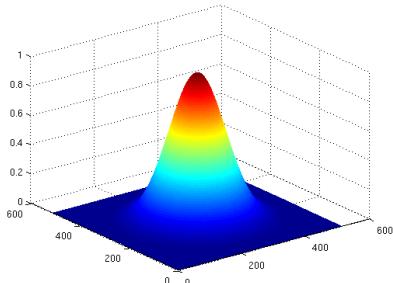
- Generate the Gaussian random field (see Fig. 28.2b). Test with different values of  $\sigma$ .



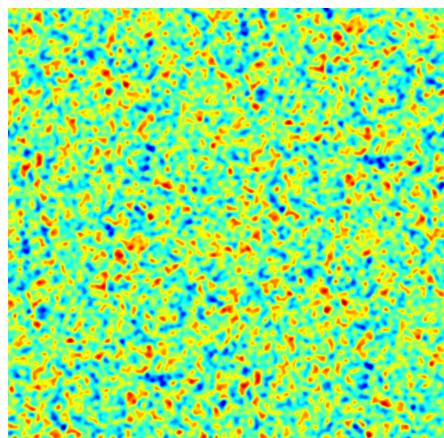
Use `meshgrid` for efficiency.

Figure 28.2: Covariance and Gaussian random field examples.

(a) Generation of a 2D Gaussian function.



(b) Generation of a 2D Gaussian random field with Gaussian covariance.



## 28.3 Geometry

### 28.3.1 Excursion set

Let  $Y(x)$ ,  $x \in \mathbb{R}^2$ , be a stationary real-valued random field. An excursion set, denoted by  $E_h(Y, S)$ , of  $Y$  inside a compact subset  $D \subset \mathbb{R}^2$  above a level  $h$ , is defined as:

$$E_h(Y, D) = \{p \in D : Y(p) \geq h\}$$

In  $D \subset \mathbb{R}^2$ , an excursion set is a binary image.

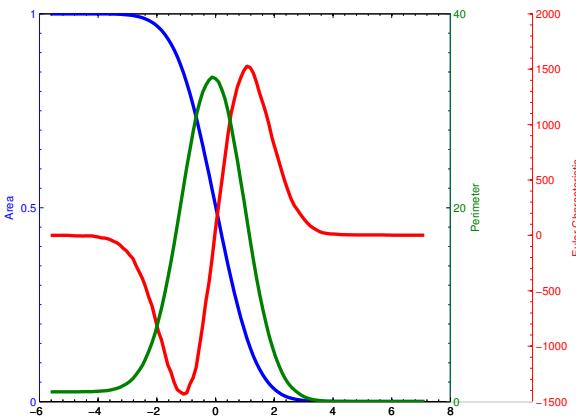


Represent, for each level  $h$ , the area  $A$ , the perimeter  $P$  and the Euler number  $\chi$  of  $E_h(Y, D)$  (see Fig. 28.3). You can have a look at the tutorial about integral geometry for perimeter, area and Euler number computation.



`bwperim(levelset)`, `bwarea` and `bweuler( levelset )` can be useful. Use default values for the neighborhood.

Figure 28.3: Example of computation of Area, Perimeter and Euler number of level sets  $E_h$ .



### 28.3.2 Analytical values

We define three values, area  $A$ , perimeter  $C$  (Contour length) and Euler number  $\chi$ , as functions of the level  $h$ , with:

$$A(h) = \mathcal{L}_2(D)\rho_0(h) \quad (28.3)$$

$$C(h) = 2 \left( \mathcal{L}_1(D)\rho_0(h) + \frac{\pi}{2} \mathcal{L}_2(D)\rho_1(h) \right) \quad (28.4)$$

$$\chi(h) = \mathcal{L}_0(D)\rho_0(h) + \mathcal{L}_1(D)\rho_1(h) + \mathcal{L}_2(D)\rho_2(h) \quad (28.5)$$

with  $\mathcal{L}_0(D) = \chi(D) = 1$ ,  $\mathcal{L}_1(D)$  is half the boundary length of the rectangle  $D$  and  $\mathcal{L}_2(D)$  is its area.



- First, compute the following values:

$$\rho_0(h) = \int_h^\infty \frac{1}{\sqrt{2\pi}} e^{-u^2/2} du \quad (28.6)$$

$$\rho_1(h) = \frac{\sqrt{\lambda}}{2\pi} e^{-h^2/2} \quad (28.7)$$

$$\rho_2(h) = \frac{\lambda}{(2\pi)^{2/3}} e^{-h^2/2} h \quad (28.8)$$

$$\lambda = \frac{1}{\sigma^2} \quad (28.9)$$

- Then, compute the analytical values of  $A$ ,  $C$  and  $\chi$ .
- Compare the analytical values to the empirical ones.



Use [erf](#).



## 28.4. Matlab correction



### 28.4.1 White noise

The white noise is generated with the following code. This is illustrated in Fig.28.4.



```

%% white noise simulation
1 close all
2 n = 128;
3
4 W = randn(n);
5 imagesc(W);
6
7 imwrite_rf(W, 'wn.png');
8

```

In order to save a grayscale matrix as a colored image, the following code is necessary.



```

function I = imwrite_rf(R, name)
1 % function that write a random field in a color image
2 %
3 % R: random field
4 % name: name of the image
5 %
6 % I: color image rgb
7
8 R = R - min(R(:));
9 R = 255 * R / max(R(:));
10 I = ind2rgb(uint8(R), jet(256));
11
12 imwrite(I, name);

```

### 28.4.2 Gaussian Random Field

The Gaussian function that will serve as a covariance function is generated via the given code. Pay attention to the discretization grid: the FFT implies that functions are periodic, and in order to do that, the discretization must be set between  $[-N/2 : N/2[$ .

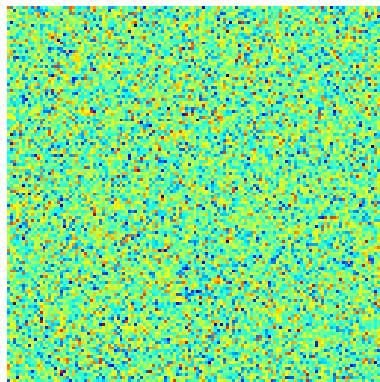


Figure 28.4: White noise.



```

% gaussienne
2 N = 512;
N = 2^nextpow2(N); % force power of two
4 [Y, X] = meshgrid(-N/2:N/2-1, -N/2:N/2-1);

6 % covariance generation
sigma = 10;
8 Cmat = exp(-(X.^2+Y.^2)/(2*sigma^2));
figure()
10 h=surf(Cmat);
set(h, 'edgecolor', 'none')

```

The Gaussian Random Field can be generated via the formula already presented.



```

1 % Fourier domain
% gaussian complex white noise
3 W=randn(N);

5 % covariance matrix in the Fourier domain
% ensure real values, although this is theoretically true
7 Cf = real(fft2(Cmat));

9 % ensure positive values, some numerical approximations can give negative values
Cf = sqrt(max(zeros(size(Cf)), Cf));
11
% phi_hat is the fourier transform of the gaussian random field
13 phi_hat = Cf.* fft2(W);

15 % we take the real part (should be real, but due to numerical
% approximations ...)

```



```

17 G = real( ifft2 (phi_hat));
19 % verify statistical properties
20 m = mean(G(:));
21 s = std(G(:));
23 figure()
24 imagesc(G);
25 imwrite_rf(G, 'grf.png');

```

### 28.4.3 Minkowski functionals

The Minkowski functionals are illustrated in Fig.28.5. The code follows.

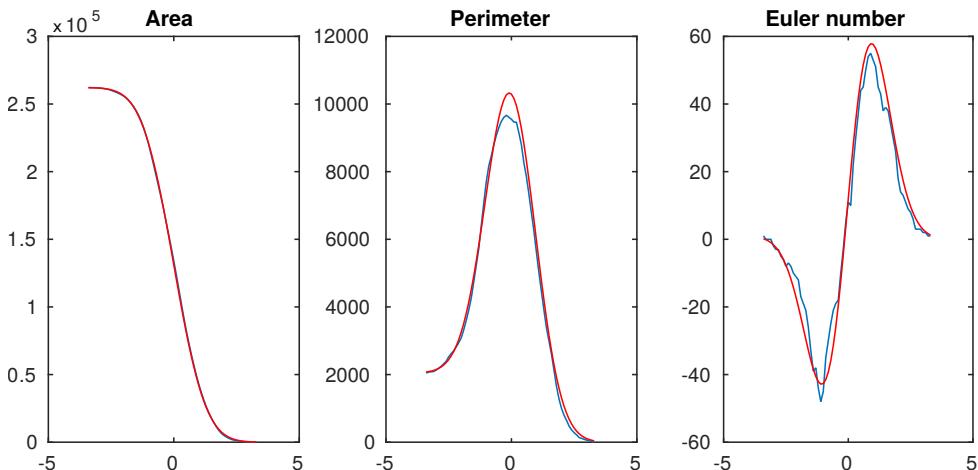


Figure 28.5: Illustration of the simulated and analytical values of the Minkowski functionals of the level sets of the Gaussian Random Field.



```

1 %
2 hmin = min(G(:));
3 hmax = max(G(:));
4 H = hmin :.1: hmax;
5 %
6 A = zeros(length(H), 1);
7 P = zeros(length(H), 1);
8 E = zeros(length(H), 1);
9 %
10 % analytical values
11 %

```



```
rho_0 = zeros(length(H), 1);
13 rho_1 = zeros(length(H), 1);
rho_2= zeros(length(H), 1);
15 lambda = 1/(2* sigma^2);
17 for i = 1:length(H)
19 levelSet = G>=H(i);
A(i) = bwarea(levelSet );
21 P(i) = bwarea(bwperim(levelSet ,4) );
E(i) = bweuler(levelSet ,4) ;
23
% analytic
25 rho_0(i) = 1/2* erfc (H(i)/ sqrt (2) );
rho_1(i) = sqrt (lambda)*exp(-H(i)^2/2)/(2* pi) ;
27 rho_2(i) = lambda /(2* pi) ^ (3/2) * exp(-(H(i)^2)/2)*H(i);
end
29 Aa = N^2*rho_0;
Pa = 4*N*rho_0 + pi*N^2*rho_1;
31 Ea = rho_0 + 2*N*rho_1 + N^2*rho_2;

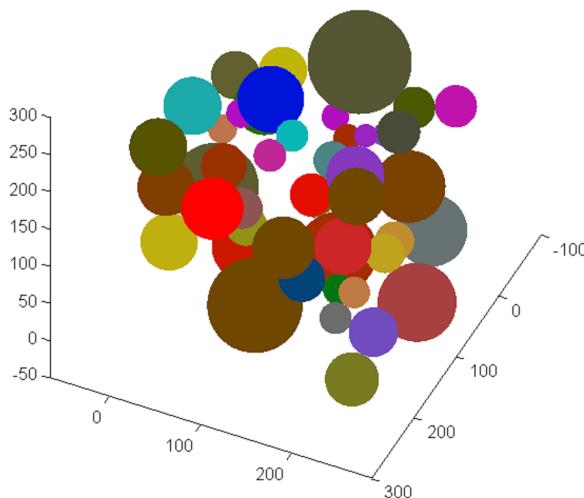
33 %----- display results
figure ;
35 subplot(131); plot(H, A); hold on;
plot(H, Aa, 'r'); title ('Area');
37 subplot(132); plot(H, P); hold on;
plot(H, Pa, 'r'); title ('Perimeter');
39 subplot(133); plot(H, E); hold on;
plot(H, Ea, 'r'); title ('Euler number');
```





## 29 Stereology and Bertrand's paradox

This tutorial introduces the problems of stereological measurements, based on simple probes (points, lines...). In a second part, the Bertrand's paradox is explored in the case of the analysis of the distribution of chord lengths of disks and spheres. This tutorial uses the notation that can be found in [?] (among others).



### 29.1

## Classical measurements of stereology

Let start by some definitions. The stereology is based on some measures, that can be seen as samples, called probes. A probe can be a point, a line, a curve, a plane, a surface... These probes allow us to estimate global geometrical properties through partial measures. Very simple and practical probes are now presented and used. The notation  $\langle \cdot \rangle$  denotes an expected count for some normalized value.

Probe name	Notation	Definition
Point count	$\langle P_P \rangle$	Fraction of points in phase
Line intercept count	$\langle P_L \rangle$	Number of intersections per length
Area fraction	$\langle A_A \rangle$	Fraction of area in intersection

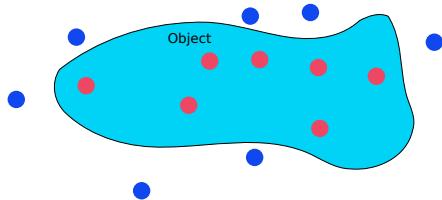
### 29.1.1 Probes

The measures are performed with the following definitions:

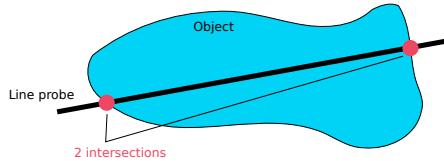
- $\langle P_P \rangle$ : the count of the points that lie in the phase, normalized by the total number of points.
- $\langle P_L \rangle$ : the count of the number of intersection of lines and the surface of the phase, normalized by the total length of the lines (in  $m^{-1}$ , see Fig.29.1).  $L_A$  is the ratio between the perimeter of the phase and the total area.
- $\langle A_A \rangle$ : in a 3D object, the probes are planes and  $\langle A_A \rangle$  is the area of the intersections of the planes and the phase, normalized by the total area of the planes.

Figure 29.1: Probes examples: points and lines.

(a) Evaluation of  $P_P$ : count the number of points that lie in the objects, normalized by the total number of points.



(b) Evaluation of  $P_L$ : count the number of intersection of some lines with the surface of the phase (object). Then, perform a ratio with the total length of the lines.



- Generate a 2-D binary image containing a population of disks.
- Estimate its area fraction by using points as probe population ( $\langle P_P \rangle = A_A$ ).
- Estimate its length per area by using lines as probe population ( $\langle P_L \rangle = \frac{2}{\pi} L_A$ ).
- Load the 3-D image and verify the following relation:  $\langle A_A \rangle = V_V$  with  $V_V$  being the volume fraction.

## 29.2

# Random chords of a disk, and Bertrand's paradox

The goal of this exercise is to simulate the distribution of random chord lengths on a disk. In the field of process engineering, optical particle sizers provide chord

length distributions of objects that are considered as spheres. These developments are not only theoretical, they have a practical use in laboratories or in industrial reactors.

### 29.2.1 Bertrand's paradox

From Wikipedia<sup>1</sup>, the Bertrand paradox goes as follows: Consider an equilateral triangle inscribed in a circle. Suppose a chord of the circle is chosen at random. What is the probability that the chord is longer than a side of the triangle?

Bertrand provided three different methods in order to evaluate this probability. These gave three different values.

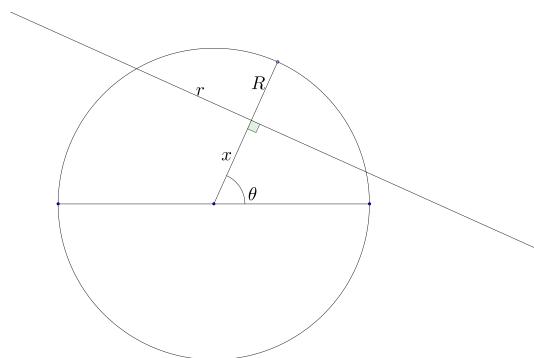
The objective is to evaluate the distribution of the chord lengths for two of these methods. The bertrand's paradox lays on the fact that the question is ill-posed, and the choice "at random" must be clarified. The reader will find more details in the different citations.

### 29.2.2 Random radius

The intersection of a random line with a disk of radius  $R$  is a segment whose half length  $r$  is linked to the distance  $x$  between the segment and the center of the disk (Eq. 29.1 and Fig. 29.2).

$$r = \sqrt{R^2 - x^2} \quad (29.1)$$

Figure 29.2: Relation between the radius of the chord and the distance to the center of the disk.



As presented in Fig. 29.2,  $\theta$  is randomly chosen in  $[0; 2\pi[$  (uniform law), and  $x$  is randomly chosen in  $[0; R]$  (uniform law).

---

<sup>1</sup>[https://en.wikipedia.org/wiki/Bertrand\\_paradox\\_\(probability\)](https://en.wikipedia.org/wiki/Bertrand_paradox_(probability))



Repeat the simulation of  $r$  by the so-called random radius method a large number of times ( $N = 1e7$ ), and compute the probability density function of the distribution.

This method is in agreement with the analytical results.

### 29.2.3 Random endpoints

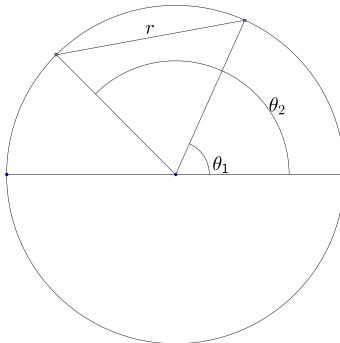
In this second case, two random angles  $\theta_1$  and  $\theta_2$  are randomly chosen (uniform law in  $[0; 2\pi]$ ). This defines two points and thus a chord (see Fig 29.3).



Make the simulation for the “random endpoints” and compare it to the analytical values and to the first simulation. Notice that this simulation gives different values (search for the Bertrand paradox for more details).

This method is NOT in agreement with the analytical results.

Figure 29.3: Second method for simulating a random chord of the disk.



### 29.2.4 Analytical values

The probability to obtain a chord of half length  $x$  between  $a$  and  $b$  is given by Eq. 29.2:

$$\mathbb{P}(x \in [a; b]) = \int_a^b \frac{\rho}{R\sqrt{R^2 - \rho^2}} d\rho \quad (29.2)$$



By discretizing the interval  $[0; R]$ , compute the analytical value of the probability density function of the distribution of the radii with the Eq. 29.2.

## 29.3

# Random sections of a sphere and a plane

The two previous strategies can be applied on the sphere. What we are looking for are radii of the intersection of the sphere with a random plane.

### 29.3.1 First simulation: random radius

To find a random plane  $\mathcal{P}$  intersecting a sphere  $\mathcal{S}$  of radius  $R$ , one have to choose a direction  $\vec{u}$  (i.e. a point on the sphere), find a point  $P$  between the latter and the center  $O$  of the sphere, and consider the plane  $\mathcal{P}$  that is orthogonal to the direction  $\vec{u}$  and passing at this point  $P$  (Fig. 29.4).



Code the following method:

- A random point on the sphere is chosen with the following method (see [?]): let  $x$ ,  $y$  and  $z$  be 3 Gaussian random variables, the point on the sphere is defined by the normed vector  $\vec{u}$ , such that:

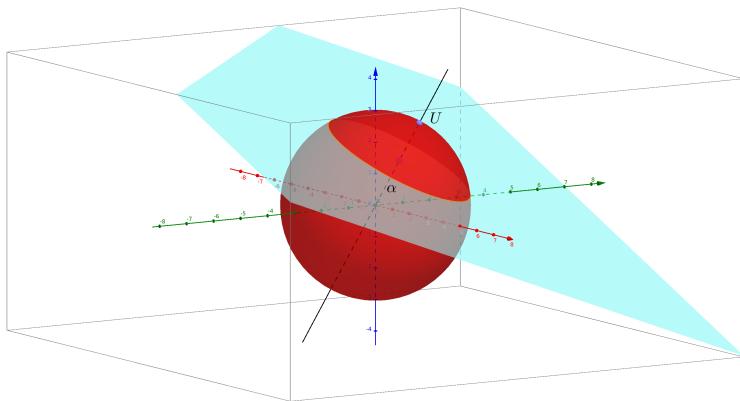
$$\vec{u} = \frac{1}{\sqrt{x^2 + y^2 + z^2}} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \quad (29.3)$$

- The point  $P$  is chosen as  $\vec{OP} = \alpha \vec{u}$ , with  $O$  being the center of the sphere, and  $\alpha$  being a uniform random variable in  $[0; R]$ . The plane  $\mathcal{P}$  is orthogonal to  $\vec{u}$ , passing at  $P$  (see Fig. 29.4).

Notice that this simulation is equivalent to the first presented case (random radius) of the random chords of a disk, and that the choice of the random point on the sphere is useless.

The method that consists to choose a point by two angles does not provide a uniform distribution of the points on the sphere.

Figure 29.4: First simulation method of a random intersection of a sphere and a plane.



### 29.3.2 Second simulation: 3 random endpoints

The random plane  $\mathcal{P}$  is defined by 3 random points laying on the sphere (see Eq. 29.3).



- Analytically, find the distance between the center of the sphere  $O$  and the plane  $\mathcal{P}$ .
- Simulate a high number of intersections and find numerically the distribution of the radii of these intersection disks.

### 29.3.3 Third simulation: 2 random endpoints

Choose 2 random points laying on the sphere (see Eq. 29.3) and evaluate their half distance.



Simulate a high number of couple of points and evaluate the distribution of their half distances.

The distribution of the length of the chords on a sphere is linear, and is different from the distribution of the radii resulting from the intersection of a random plane and the sphere.

### 29.3.4 Comparison



Compare the 3 results and comment.



## 29.4. Matlab correction



### 29.4.1 Classical measurements of stereology

#### Population of disks

To generate a given number of overlapping disks, the following procedure can be used. Result is shown in Fig. 29.5.



```
% generates population of disks
2 function I = popDisks(nb_disks, S, Rmax)
% nb_disks: number of disks
4 % S: size of the spatial support
% Rmax: maximum radius of disks
6 % I: result , binary image of size SxS

8 I = zeros(S);

10 positions = randi(S, nb_disks, 2);
radii      = rand(nb_disks, 1) * Rmax;
12
13 [X, Y] = meshgrid(1:S, 1:S);
14 for i=1:nb_disks

16     I2 = ((X-positions(i,1)).^2 + (Y-positions(i,2)).^2) <= radii(i).^2;
I = I | I2;
18 end
```

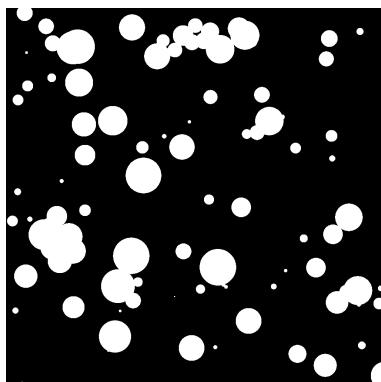


Figure 29.5: Random population of disks in a rectangle.

### Area fraction

The 2-D binary image containing the population of disks is loaded or generated by the previous method:



```
A=double(imread('disks.bmp'))/255;
```

Then the points are randomly chosen and the measures are taken



```

1 % Points where the measures are performed
nbPoints=1000;
3 P = randi(size(A,1), nbPoints, 2);

5 % Do the measures
probes=0;
7 for i=1:nbPoints
    if A(P(i,1), P(i,2)) == 1
9        probes = probes + 1;
    end
11 end

13 % the area of the objects can be measured by the ratio of probes/nbPoints
PP = probes/nbPoints;
15 disp(['PP: ' num2str(c)])

```

This value has to be compared to the real number of pixels:



```

1 AA = bwarea(A)/(size(A,1) * size(A,2));
disp(['AA: ' num2str(r)])

```

### Command window



```

>>PP: 0.369
2 >>AA: 0.36576

```

### Length per area

This evaluation highly depends on the result of the perimeter evaluation that depends on the connectivity chosen. The Fig.29.6 illustrates the different steps for evaluating  $P_L$ .

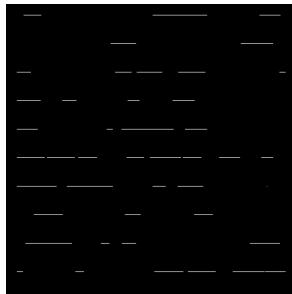


```
% create an image with lines
2 probe=zeros(512,512);
probe(20:50: end-20,20:end-20)=1;
4 lines=probe.*A;

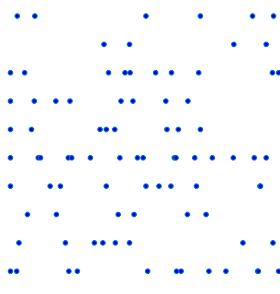
6 % detect number of segments
points=abs(conv2(lines,[1 -1 0], 'same'));
8
disp(['pi*PL/2 : ' num2str(sum(points(:))/sum(lines(:))*pi/2)])
10 disp(['LA: ' num2str(sum(bwperim(A))/bwarea(A))])
```



(a) Probe lines.



(b) Intersection with binary set.



(c) Location of intersection with the surface of the binary set.

Figure 29.6: Probe lines.

### Command window

```
>>pi*PL/2 : 0.078406
2 >>LA: 0.067719
```

### Volume fraction

This measure is more precise and is the equivalent of the area fraction.



```
spheres = load('spheres.mat');
2 VV = sum(spheres.A(:)) / ( size(spheres.A,1) * size(spheres.A,2) * size(spheres.A
    ↪ ,3));
probe = zeros(size(spheres.A));
4 probe(10:50: end-10, 10:end, 10:end) = 1;
```



```

6 s = sum(probe());
probe = probe .* spheres.A;
8 AA= sum(probe(:)) / s;
disp(['VV=' num2str(VV)])
10 disp(['AA=' num2str(AA)])

```

### Command window

```

>>VV=0.050185
2 >>AA=0.052769

```

## 29.4.2 Random chords of a disk



Note on the random uniform generation: MATLAB® `rand` generates a number between  $]0;1[$  (open interval). This should not be a problem in this simulation.

### First case: random radius

To find the probability,  $N$  values  $x$  are randomly chosen between 0 and  $R$ . The formula  $r = \sqrt{R^2 - x^2}$  yields to the half length  $r$  of the chord. After discretizing the interval  $[0; R]$  in  $nBins$ , the number of values  $x$  in each bin is counted (with MATLAB® `histcounts` function). The results are presented in red in Fig. 29.7.



```

N = 1e7; % nb of points
2 nBins = 1000; % histogram number of bins
R = 1; % radius of the disk
4
% take a random (uniform law) number between 0 and R and compute the
% half length of the chord
d=R * rand(N, 1);
8 radii = sqrt(R^2 - d.^2);
probaSimu = histcounts(radii, nBins, 'normalization', 'probability');

```

### Second case: random endpoints

It appears that this method does not follow the analytical solution, and thus should be avoided.



```

1 % 2nd method: take 2 points A and B uniformly on the disk,
% compute their half length
3 thetas = 2*pi * rand(N,2);

5 dX = diff (R * cos( thetas ),1,2) ;
dY = diff (R * sin( thetas ),1,2);
7 radii = 1/2 * sqrt (dX.^2 + dY.^2);
probaSimu2 = histcounts (radii , nBins, 'normalization' , 'probability');

```

### Analytical values

The analytical values are computed like this: define discretization of the interval  $[0; R]$  and approximate the integral on each segment. The results are presented in Fig. 29.7, with black dots. To simplify the display of the values, a second sampling (defined by the variable step) is used.



```

step = 0.1;
2 r2 = 0:step:R;
probaReal = 1/R* r2 ./ sqrt (R^2 - r2.^2);

```

In order to get the probability density function, you have to perform the approximation of the integral (here, by the so-called rectangle method):

$$\int_a^b f(t)dt \approx (b-a) \cdot f\left(\frac{b-a}{2}\right)$$

which is coded by:



```

1 probaReal = probaReal * R/nBins; % approximation of the integral

```

### Results

To display the results, the (simple) following method is proposed (see Fig. 29.7):



```

1 figure ; hold on
r = 0:R/nBins:R*(1-1/nBins);
3 plot(r, probaSimu, 'r', 'linewidth', 2);
plot(r2, probaReal, 'ko', 'linewidth', 3);

```



```
5 plot(r, probaSimu2, 'b', 'linewidth', 2);
legend({'random radius' 'Analytical values' '2 random angles'});
```

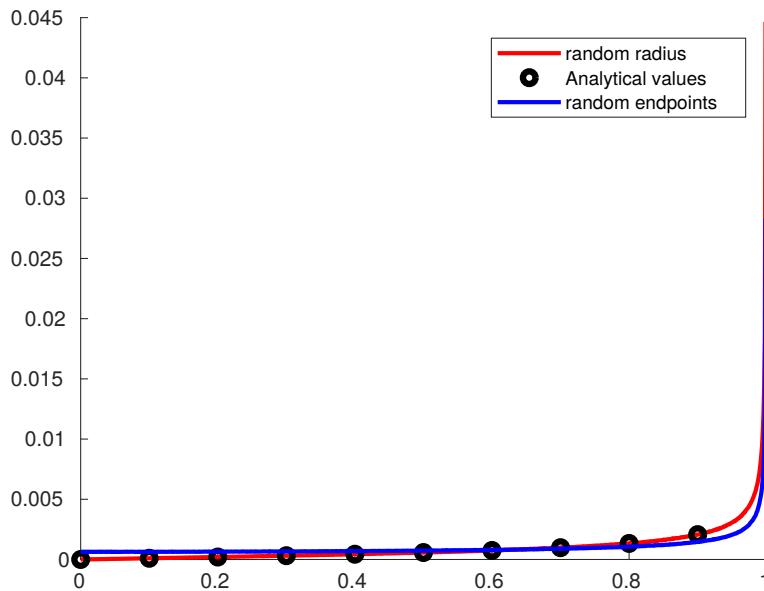


Figure 29.7: Probability of disks chord length simulations in 2-D, equivalent to the radii distribution of the disk, intersections of the sphere and a random plane in 3-D.

#### 29.4.3 Random planes intersecting a sphere

##### Random radius

This method is equivalent to the first case of the disk chord. The 3D property of the sphere is not used and thus the code is strictly equivalent to the 2D case.



```
N = 1e6; % number of points
2 R = 1; % radius of the sphere
nBins = 1000;
4
radii = 0:R/nBins:R*(1-1/nBins);
6
% first simulation: equivalent to a disk
8 x = R*rand(N, 1);
r = sqrt(R^2 - x.^2);
10 proba = histcounts(r(r<=R), nBins, 'normalization', 'probability');
```

### Definition of a random intersecting plane by 3 endpoints

Let  $n_1, n_2$  and  $n_3$  be 3 points on the sphere. These points define the plane  $\mathcal{P}$ . The distance between the center of the sphere  $O$  and the plane  $\mathcal{P}$  is given by the relation:

$$d(0, \mathcal{P}) = \frac{|\vec{n} \cdot \vec{u}|}{\|\vec{n}\|}$$

with  $\vec{u} = \vec{n}_2 - \vec{n}_1$ ,  $\vec{v} = \vec{n}_3 - \vec{n}_1$ , and  $\vec{n} = \vec{u} \wedge \vec{v}$  the normal vector to the plane. The results are presented in Fig. 29.8.

This is a case of Bertrand's paradox: the definition of randomness is not good in the present case.



```
% n1, n2 and n3 are 3 arrays of points
1 n1 = randn(N,3);
2 mynorm = sqrt(sum(n1.^2,2));
3 n1 = R* n1 ./ repmat(mynorm,1,3);

5 n2 = randn(N,3);
6 mynorm = sqrt(sum(n2.^2,2));
7 n2 = R* n2 ./ repmat(mynorm,1,3);

9 n3 = randn(N,3);
10 mynorm = sqrt(sum(n3.^2,2));
11 n3 = R* n3 ./ repmat(mynorm,1,3);

13 % u and v are vectors that belong to the plane
14 u=n2-n1;
15 v=n3-n1;
16 % n: normal vector to the plane
17 n=cross(u,v);

19 % x: distance from the center of the sphere to the plane
20 x = dot(n, n1, 2) ./ sqrt(sum(n.^2, 2));
21

23 % r: radius of the intersection sphere/plane
24 r = sqrt(R^2 - x.^2);

26 proba = histcounts(r, nBins, 'normalization', 'probability');
27 plot( radii , proba, 'b');
```

### Use of 2 random points

This situation presents the random choice of two points on the sphere, and the computation of their distance. This produces a linear probability (see Fig.29.8).

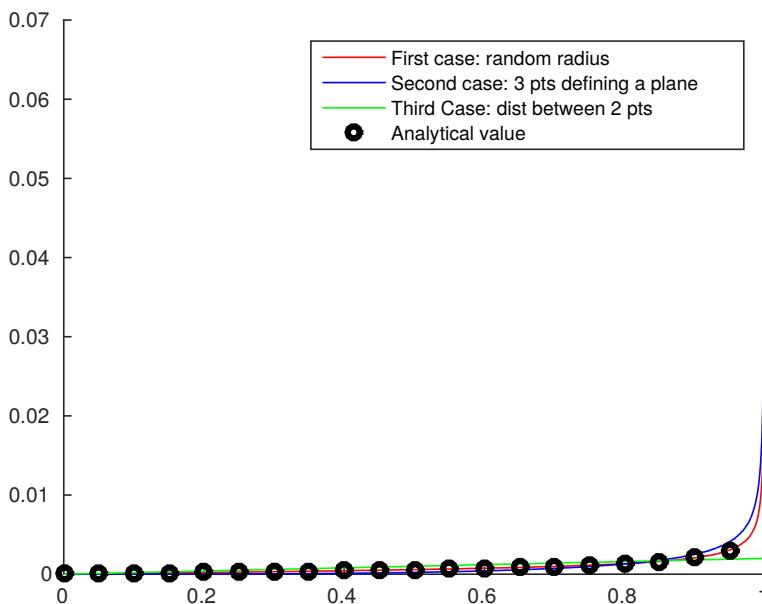


Figure 29.8: Case of the sphere intersecting a random plane.



```

1 % first points
n1 = randn(N,3);
3 mynorm = sqrt(sum(n1.^2,2));
n1 = R* n1 ./ repmat(mynorm,1,3);
5
% second points
7 n2 = randn(N,3);
mynorm = sqrt(sum(n2.^2,2));
9 n2 = R* n2 ./ repmat(mynorm,1,3);

11 % evaluate half Euclidean distance
r = sqrt (sum((n1-n2).^2, 2)) /2;
13
% probability
15 proba = histcounts (r, nBins, 'normalization', 'probability');

```

The Bertrand's paradox is illustrated by the fact that “at random” can provide several different interpretations. The objective here is to focus on the computational choices that can be made in order to provide random chords or random points on a sphere.

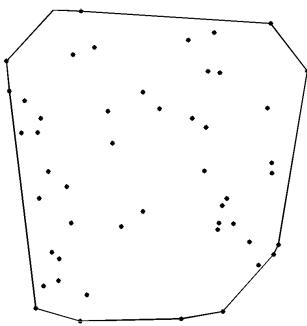




## 30 Convex Hull

This tutorial aims to determine the convex hull of a set of 2D points with a simple and classical algorithm. This tool is largely used in computational geometry and image modeling. This tutorial is widely inspired of the Wikipedia page [https://en.wikipedia.org/wiki/Graham\\_scan](https://en.wikipedia.org/wiki/Graham_scan).

Figure 30.1: Convex Hull example.



### 30.1

## Graham scan

The Graham scan is a method of computing the convex hull of a finite set of points in the plane with time complexity  $O(n \log n)$ ,  $n$  is the number of points. It is an evolution of the Gift wrapping algorithm ( $O(nh)$ ,  $h$  is the number of points in the hull) in the sense that it avoids evaluating all pairs of angles by first sorting the points.

### 30.1.1 Lowest y-coordinate point

The first step, as in the gift wrapping algorithm, is to find the point with the lowest y-coordinate. If two points exist in the set, choose the one with the lowest  $x$ -coordinate: it is denoted  $P$ . This step obviously takes  $O(n)$ .



Use the `min` function.

### 30.1.2 Sort by angle

Next, the set of points must be sorted in increasing order of the angle they and the point  $P$  make with the  $x$ -axis.

This is the limiting step, it takes  $O(n \log n)$ . Notice that the cosine of the angle is a decreasing function between 0 and 180 degrees, and will thus avoid to evaluate the angle itself. The sorted set of points is denoted  $\mathcal{S}$  (it does not contain  $P$ ).

### 30.1.3 Check angles: left or right turn?

From the star-like shape issued from the sorting algorithm, construct a list  $\mathcal{L}$  of points as:  $\mathcal{L} = \{P, \mathcal{S}, P\}$ .

Then, for each triplet of consecutive points  $(P_i, P_{i+1}, P_{i+2})$  of  $\mathcal{L}$ , check if the angle  $\widehat{P_i P_{i+1} P_{i+2}}$  is a right turn or a left turn. In case of a right turn, remove  $P_{i+1}$  from the list  $\mathcal{L}$ . Process the entire list this way.

### 30.1.4 Left or right turn?

Again, determining whether three points constitute a “left turn” or a “right turn” does not require computing the actual angle between the two line segments, and can actually be achieved with simple arithmetic only. Consider the cross product of the vectors  $\overrightarrow{P_i P_{i+1}}$  and  $\overrightarrow{P_i P_{i+2}}$ .

```
Procedure ccw( $p_1, p_2, p_3$ )
|   return  $(p_2.x - p_1.x) * (p_3.y - p_1.y) - (p_2.y - p_1.y) * (p_3.x - p_1.x)$ ;
```

Three points are a counter-clockwise turn if  $ccw > 0$ , clockwise if  $ccw < 0$ , and collinear if  $ccw = 0$  because  $ccw$  is a determinant that gives the signed area of the triangle formed by  $p_1, p_2$  and  $p_3$ .

### 30.1.5 Convex hull algorithm

This pseudo-code shows a different version of the algorithm, where points in the hull are pushed into a new list instead of removed from  $\mathcal{L}$ .

```

Data:  $n$ : number of points
Data:  $\mathcal{L}$ : sorted list of  $n + 1$  elements
Data: First and last elements are the starting point  $P$ .
Data: All other points are sorted by polar angle with  $P$ .
stack will denote a stack structure, with push
and pop functions.
Data: stack.push( $\mathcal{L}(1)$ )
Data: stack.push( $\mathcal{L}(2)$ )
for  $i = 3$  to  $n + 1$  do
    while stack.size  $\geq 2$  AND ccw(stack.secondlast, stack.last,  $\mathcal{L}(i)) < 0$ 
        do
            | stack.pop();
        end
    stack.push( $\mathcal{L}(i)$ );
end
```

In this pseudo-code, stack.secondlast is the point just before the last one in the stack. When coding this algorithm, you might encounter problems with floating points operations (collinearity or equality check might be a problem).



1. Generate a set of random points.
2. Implement and apply the algorithm, and visualize the result.



## 30.2. Matlab correction



### 30.2.1 Graham scan algorithm

The convex hull function starts by giving the points as parameters:



```
function Q=conv_hull(PointsOri)
2 % PointsOri: original point set
```

This function follows the stated algorithm (Graham Scan).

- find lowest y-axis point
- sort points by their angle
- proceed in this order and check left or right turn

For convex hull and other computational geometry algorithms, robustness must be handled with special care. Floating points operations may be really tricky and the following code is not ensured to work for all cases.



```
% very naive precision handling
2 Points=round(PointsOri .*1000) /1000;
```

The first step is to get the starting point.



```
% get points with minimal y-coordinate
2 PP = sortrows(Points, [2 1]);
P = PP (1,:);
```

Then, the points are sorted by the opposite of the cosine of the angle, because cosine is decreasing on the interval  $[0; \pi]$ . This step has complexity  $O(n \log n)$ .



```
1 Points=PP(2:end, :);
3 % sort by cosinus of angle
adj_side=Points (:,1) -P(1);
5 opp_side=Points (:,2) -P(2);
```



```

hypotenuse=sqrt(adj_side.^2+opp_side.^2);
7 cosinus=adj_side ./ hypotenuse;
[~, ind]=sort(-cosinus); % cos is decreasing
9 Points=[P;Points(ind ,:) ;P];

```

Finally, the sorted points allow to construct the convex hull by testing the orientation of the turn of the hull (see Fig.30.2).



```

1 % compute convex hull in this order
Q=[];
3 Q=push(Q,Points (1,:) );
Q=push(Q,Points (2,:) );
5 for i=3: size (Points ,1)
    while (length(Q)>=2 && CrossProduct(Q,Points(i ,:)) < 0)
7         Q=pop(Q);
        end
9     Q=push(Q,Points(i,:) );
end

```

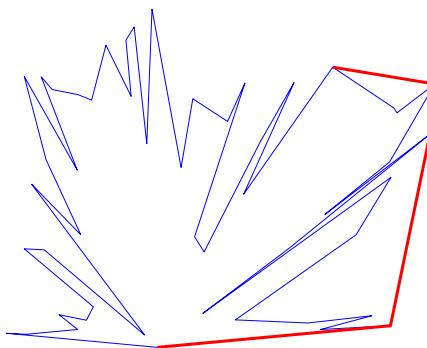


Figure 30.2: Graham scan illustration while constructing the convex hull.

### 30.2.2 Useful functions

These 3 functions are used to push and pop a point in a list, and to compute the cross-product.



The MATLAB® function `cross` could have been used.



```

1 function Z=push(Q,point)
2     % point into a list
3     Z=[Q,point];
4 end

```



```

1 function Z=pop(Q)
2     % pop last element
3     Z=Q(1:end-1,:);
4 end

```

This cross-product function first extract the last two points of the hull, and check if there is a left-turn or a right-turn to go to the point  $p_3$ .



```

1 function p=CrossProduct(Q, p3)
2     % cross product
3     % Q : list of points (hull)
4     % p3 : point
5     p1=Q(end-1,:);
6     p2=Q(end,:);
7     p=(p2(1) - p1(1))*(p3(2) - p1(2)) - (p3(1) - p1(1))*(p2(2) - p1(2));
8 end

```

### 30.2.3 Simple tests

For 5 points:



```

1 Points=[1 1 3 4 3;
2         2 -4 -4 1 0];
3 Points=Points';
4 plot(Points (:,1),Points (:,2),'*')
5 axis equal
6 Q=conv_hull(Points);
7 hold on;
8 plot(Q (:,1),Q (:,2),'r')

```

For a few random points:



```
n=50;  
2 Points2=rand(n,2);  
  
4 figure  
plot(Points2 (:,1) ,Points2 (:,2) ,'*')  
6 axis equal  
Q=conv_hull(Points2);  
8 hold on;  
plot(Q (:,1) ,Q (:,2) , 'r')
```

The results are illustrated in Fig.30.3.

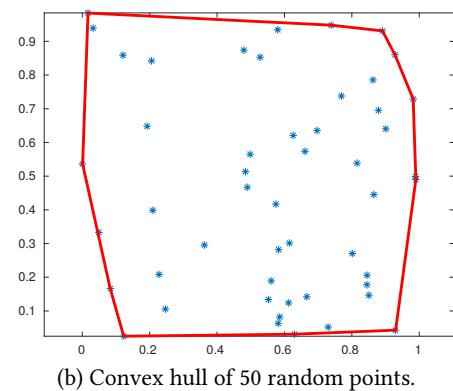
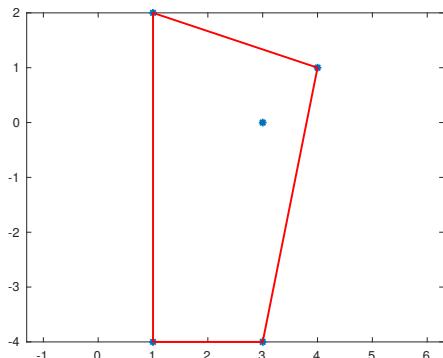


Figure 30.3: Illustration of the convex hull computation.

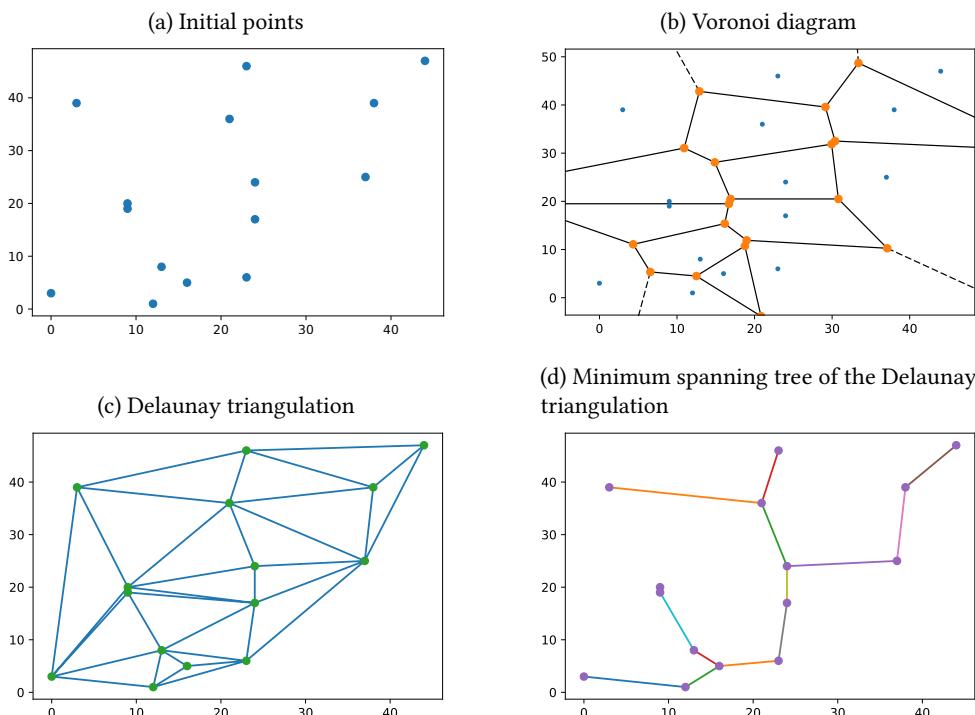




# 31 Voronoï Diagrams and Delaunay Triangulation

This tutorial aims to spatially characterize a spatial point pattern by using some tools of computational geometry: the Voronoï diagram, the Delaunay triangulation and the Minimum Spanning Tree (MST), illustrated in Fig.31.1. For biomedical issues, this point pattern analysis can help the biologists to classify different populations of cells.

Figure 31.1: Random point pattern and some geometrical structures used to characterize it.



## 31.1 Voronoï and Delaunay

A voronoï diagram, in 2D, is defined as a partition of the plane into cells  $R_k$  according to a distance function  $d$  and a set of seeds (germs)  $P_k$ .

$$R_k = \{x \in X \mid d(x, P_k) \leq d(x, P_j) \text{ for all } j \neq k\}$$

The Delaunay graph is the dual graph that links the germs of the neighboring Voronoi cells.

### 31.1.1 Random tesselation

Follow these instructions to generate a random tesselation:



1. Generate a random point process.
2. Determine the Delaunay triangulation.
3. Determine the Voronoi diagram.



Use the MATLAB® functions delaunayTriangulation and voronoiDiagram.

### 31.1.2 Characterization of the Voronoi diagram

This basic approach characterizes the set of the cells. With the help of the Voronoi diagram, it is possible to make the two following measurements, Area Disorder (AD) and Round Factor Homogeneity (RFH), defined by:

$$AD = 1 - \frac{1}{1 + \frac{\sigma(A)}{\mu(A)}} \quad (31.1)$$

$$RFH = 1 - \frac{\sigma(RF)}{\mu(RF)} \quad (31.2)$$

where  $A$  and  $RF$  are calculated on the regions  $R_k$  of the Voronoi diagram.  $\mu$  and  $\sigma$  are the mean and standard deviation of the areas of the Voronoi cells. The circularity (RF) of a polygon can be defined as the ratio between its area and the area of the disk of an equivalent perimeter.



- Code these measurements with the following prototypes:



```
function ad = AD(V, R)
% computes AD (area disorder) parameters
% V: Vertices of the Voronoi diagram
% R: Regions of the Voronoi diagram
```



```
1 def AD(vor):
# takes a voronoi diagram to compute area disorder
```

In order to evaluate the area of each Voronoi cell, transform each cell to a polygon.

- Represent the couple  $(ad, rfh)$  in a graph, which gives a characterization of the Voronoi diagram.



See polyarea for evaluating the area of a polygon.

### 31.1.3 Characterization of the Delaunay graph

If  $L$  denotes the set of the edge lengths of the Delaunay triangulation, the mean and the standard deviation of  $L$  can also give informations on the graph.



- Compute and display in a graph the point of coordinates  $(\mu(L), \sigma(L))$ , with  $\mu$  representing the mean and  $\sigma$  the standard deviation.

## 31.2 Minimum spanning tree

Definition from Wikipedia: a minimum spanning tree (MST) or minimum weight spanning tree is a subset of the edges of a connected, edge-weighted undirected graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight. That is, it is a spanning tree whose sum of edge weights is as small as possible.

One of the methods to compute the MST is the Kruskal algorithm.



It is implemented in Matlab via the function `minspantree` (introduced in MATLAB® 2015b) or `graphminspantree`.



- Compute the MST.
- Compute  $(\mu(L^*), \sigma(L^*))$  where  $L^*$  denotes the set of the edge lengths of the MST.

### 31.3

## Characterization of various point patterns



1. Generate  $n$  condition Poisson point processes of 100 points each. For each realization, calculate the parameters  $(AD, RFH)$ ,  $(\mu(L), \sigma(L))$  and  $(\mu(L^*), \sigma(L^*))$ . Display these  $n$  points in a 2D diagram in order to analyze the robustness of the quantification.
2. Generate 3 different point processes with regular, uniform and Gaussian dispersion. Display the different diagrams. Which one is the most discriminant?



## 31.4. Matlab correction



### 31.4.1 Voronoi, Delaunay and Minimum Spanning Tree

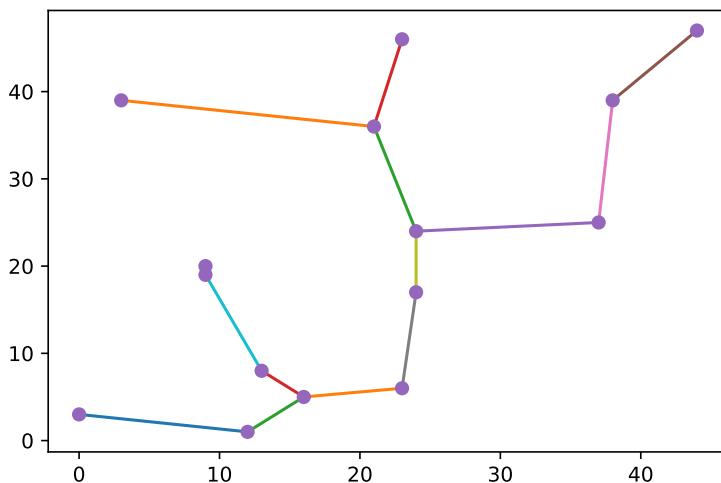


Figure 31.2: Delaunay triangulation, Voronoi Diagram and Minimum spanning tree of a point process of 15 points with a uniform distribution.

Let  $P$  be a point process, generated by:



```

P = rand(15,2);
2
% display
4 plot(P(:,1), P(:,2), 'g*');
axis square

```

#### Delaunay Triangulation



```

1 DT = delaunayTriangulation(P);
tripplot(DT)

```

## Voronoi Diagram



```
[V, R] = DT.voronoiDiagram();
2 % display
voronoi(DT);
```

## Minimum spanning tree



```
1 % From Delaunay Triangulation, compute edges distances
edges = DT.edges;
3 P1 = P(edges (:,1) , :);
P2 = P(edges (:,2) , :);
5 d = sqrt(sum((P1-P2).^2,2));

7 % directed graph as a sparse matrix
DG = sparse(edges (:,1) , edges (:,2) , d, size(P,1) , size(P,1));
9 ST = graphminspantree(DG');
[i, j, s] = find(ST);
11
% display MST
13 for k = 1:length(i)
    plot([P(i(k),1) ; P(j(k),1)], [P(i(k),2) ; P(j(k),2)], 'r',
'linewidth', 2);
15 end
axis square
```

### 31.4.2 Quantification

From the previous code, the quantification is performed via the following commands:



```
ad=AD(V,R);
2 rfh=RFH(V,R);
delaunay_parameter=[mean(d) std(d)];
4 mst_parameter=[mean(s) std(s)];
```

with



```

function sol = AD(V, R )
2 % computes AD (Area Disorder) parameter
% V: Vertices from Voronoi
4 % R: Regions of Voronoi

6 s=[];
for i = 1:length(R)
8 if all(R{i}~=1) % If at least one of the indices is 1,
% then it is an open region
10 s=[s;polyarea(V(R{i},1),V(R{i},2))];
end
12 end
sol=1-(1/(1+(std(s)/mean(s))));
14 end

```

and



```

function sol = RFH( V, R )
2 % compute RFH parameter (Round Factor Homogeneity)
% V: Vertices from Voronoi
4 % R: Regions of Voronoi diagram

6 r=[];
for i = 1:length(R)
8 if all(R{i}~=1) % If at least one of the indices is 1,
% then it is an open region
10 l=length(R{i});
surface=polyarea(V(R{i},1),V(R{i},2));

14 xv=V(R{i},1);
yv=V(R{i},2);
16 perimetre=norm([xv(1),yv(1)]-[xv(l),yv(l)]);
for k = 1:(l-1)
18 perimetre=perimetre+norm([xv(k),yv(k)]-[xv(k+1),yv(k+1)]);
end
20 r=[r;4*pi* surface /(perimetre*perimetre)];

22 end
end
24 sol=1-(std(r)/mean(r));
26 end

```

The results are displayed with the next command in Fig. 31.3



```

1 figure
axis ([0 1 0 1]);
3 axis square
hold on;
5 l=msigd(1);
c=msigd(2);
7 plot (l,c, 'r*');
text (l+.02, c, '(\sigma_d,\mu_d)');
9
plot (ad,rfh, 'g*');
11 text (ad+.02, rfh, '(AD,RFH)');

13 l=msigmst(1);
c=msigmst(2);
15 plot (l,c, 'b*');
text (l+.02, c, '(\sigma_{MST},\mu_{MST})');

17 legend ({'Delaunay Characterization', 'Voronoi Characterization', 'MST
    ↪ Characterization'})

```

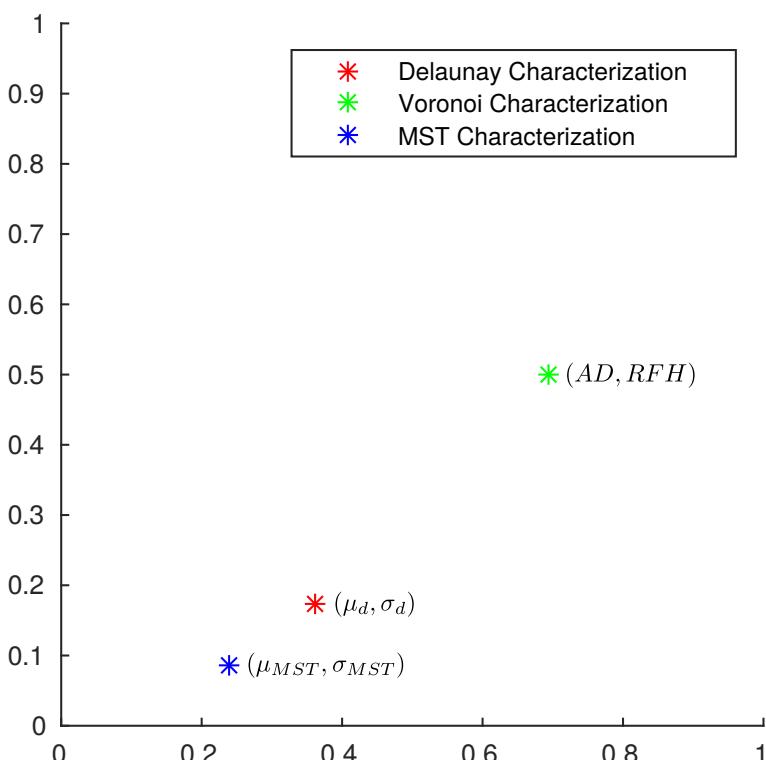


Figure 31.3: Characterization of the spatial point processes.

### 31.4.3 Characterization of various point patterns

#### Regular Point Processes



```
function [x,y] = disregular (n, length)
2 % Regular distribution of points
% n: number of points
4 % length: window size
c=floor( sqrt(n));
6 [x2,y2]=meshgrid(0:(length/c):length ,0:( length/c):length );
x=x2(:)-length/2;
8 y=y2(:)-length/2;
end
```

#### Uniform Point Processes



```
1 function [x,y] = disalea( n, length )
% uniform distribution of n points
3 % n: number of points
% length: window size
5 x1=rand(n,1);
y1=rand(n,1);
7 x=x1.*length-length/2;
y=y1.*length-length/2;
9 end
```

#### Gaussian Point Processes



```

1 function [X,Y] = disgauß(n, length)
% generate a gaussian point process, centered in 0,0, with sigma=1;
3 % n: number of points
% length: window size
5 X=0 + randn(n,1);
Y=0 + randn(n,1);
7 % cut on window
X1=(-length/2<X<length/2);
9 X=X1.*X;
Y1=(-length/2<Y<length/2);
11 Y=Y1.*Y;
end

```

## Characterization

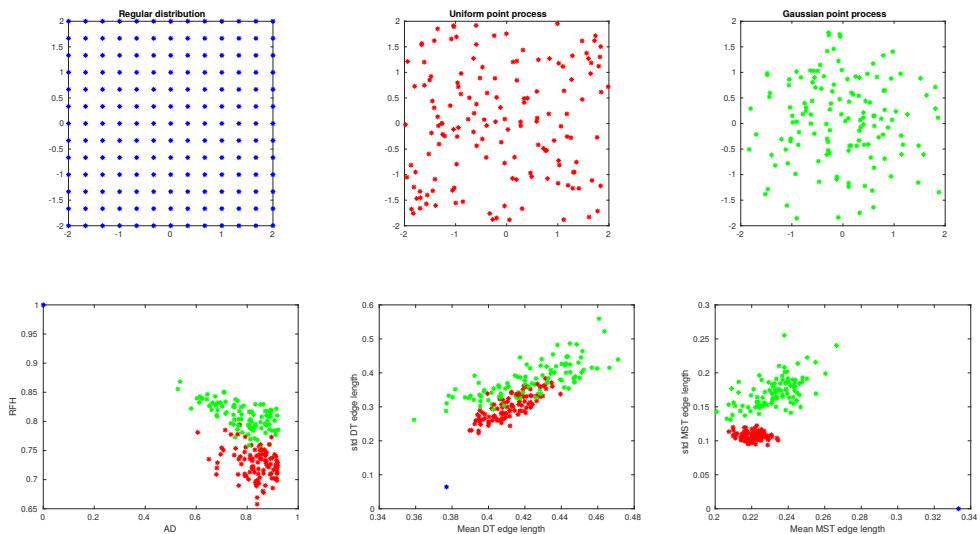


Figure 31.4: Characterization of different spatial point processes, with regular, uniform and gaussian distribution.

The following script generates the Fig.31.4.



```

figure
2 [x,y]=disregular (150,4) ; subplot (231) ; plot (x,y, 'b*') ; axis equal; title ('Regular
↔ distribution ') ; axis([-2 2 -2 2]);
4 [x,y]=disalea (150,4) ; subplot (232) ; plot (x,y, 'r*') ; axis equal; title ('Uniform point
↔ process ') ; axis([-2 2 -2 2]);

```



```
[x,y]=disgauss(150, 4); subplot(233); plot(x,y,'g*'); axis equal; title('Gaussian
    ↪ point process'); axis([-2 2 -2 2]);
6 hold on

8 % generate 100 different processes
% display the results
10 [ ad, rfh, msigd, msigmst ] = analysis([x y], 0);
    subplot(234);
12 axis([0 1 0 1]);
    plot(ad, rfh, 'b*'); xlabel('AD'); ylabel('RFH'); hold on
14 subplot(235);
    axis([0 1 0 1]);
16 plot(msigd(1), msigd(2), 'b*'); xlabel('Mean DT edge length'); ylabel('std DT
    ↪ edge length'); hold on
    subplot(236);
18 axis([0 1 0 1]);
    plot(msigmst(1), msigmst(2), 'b*'); xlabel('Mean DT edge length'); ylabel('std
    ↪ DT edge length'); hold on
20
for i=1:100
22 [x,y]=disalea(150,4);
    [ ad, rfh, msigd, msigmst ] = analysis([x y], 0);
24 subplot(234);
    plot(ad, rfh, 'r*');
26 subplot(235);
    plot(msigd(1), msigd(2), 'r*');
28 subplot(236);
    plot(msigmst(1), msigmst(2), 'r*');

30
[x,y]=disgauss(150, 8);
32 [ ad, rfh, msigd, msigmst ] = analysis([x y], 0);
    subplot(234);
34 plot(ad, rfh, 'g*');
    subplot(235);
36 plot(msigd(1), msigd(2), 'g*');
    subplot(236);
38 plot(msigmst(1), msigmst(2), 'g*');

40 end
```



## **Part V Image Characterization and Pattern Analysis**



This tutorial aims to characterize objects by measurements from integral geometry.

The different processes will be applied on the following synthetic image:

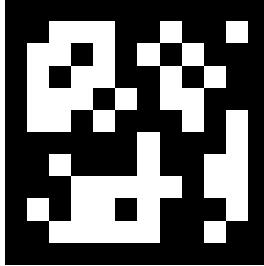


Figure 32.1:  $X$

## 32.1 Cell configuration

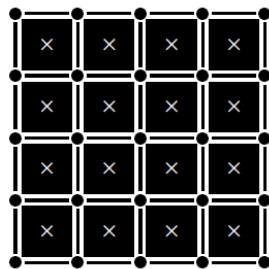
The spatial support of an image can be covered by cells associated with pixels. A cell (square of interpixel distance size) is composed of 1 face, 4 edges and 4 vertices. Either a cell is centered in a pixel (intrapixel cell) or a cell is constructed by connecting pixels (interpixel cell). The following figure shows the two possible representations of an image with 16 pixels:

Figure 32.2: Pixels representation.

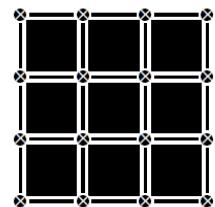
(a) Image.

$\times$	$\times$	$\times$	$\times$
$\times$	$\times$	$\times$	$\times$
$\times$	$\times$	$\times$	$\times$
$\times$	$\times$	$\times$	$\times$

(b) Intrapixel cells.



(c) Interpixel cells.



We respectively denote  $f^{intra}$  (resp.  $f^{inter}$ ),  $e^{intra}$  (resp.  $e^{inter}$ ) and  $v^{intra}$  (resp.  $v^{inter}$ ) the number of faces, edges and vertices for the intrapixel (resp. interpixel) cell configuration. Using these two configurations, the measurements from integral

geometry (area  $A$ , perimeter  $P$ , Euler number  $\chi_8$  or  $\chi_4$ ) can be computed as:

$$A = f^{intra} = v^{inter} \quad (32.1)$$

$$P = -4f^{intra} + 2e^{intra} \quad (32.2)$$

$$\chi_8 = v^{intra} - e^{intra} + f^{intra} \quad (32.3)$$

$$\chi_4 = v^{inter} - e^{inter} + f^{inter} \quad (32.4)$$



1. Count manually the number of faces, edges and vertices of the image  $X$  for the two configurations (Intra- and Inter-pixel) of Fig. 32.1.
2. Deduce the measurements from integral geometry (Eq. 32.1-32.4).

## 32.2 Neighborhood configuration

In order to efficiently calculate the number of vertices, edges and faces of the object, the various neighborhood configurations (of size 2x2 pixels) of the original binary image  $X$  are firstly determined. Each pixel corresponds to a neighborhood configuration  $\alpha$ . Thus, sixteen configurations are possible, presented in Tab. 32.1.

Table 32.1: Neighborhood configurations.

	0 0	0 0	0 1	0 1	0 0	0 0	0 1	0 1
	0 0	0 1	0 0	0 1	1 0	1 1	1 0	1 1
$\alpha$	0	1	2	3	4	5	6	7
	1 0	1 0	1 1	1 1	1 0	1 0	1 1	1 1
	0 0	0 1	0 0	0 1	1 0	1 1	1 0	1 1
$\alpha$	8	9	10	11	12	13	14	15

Thereafter, each configuration contributes to a known number of vertices, edges and faces (Tab. 32.2). To determine the neighborhood configurations of all the pixels, an efficient algorithm effective consists in convolving the image  $X$  by a mask  $F$ , whose values are powers of two, and whose origin is the top-left pixel:

$$F = \begin{pmatrix} 1 & 4 \\ 2 & 8 \end{pmatrix}$$

The resulting image is  $X * F$ . Notice that this image  $X$  has no pixel touching the borders, your code should ensure this (for example by padding the array with zeros). In this way, the histogram  $h$  of  $X * F$  gives the distribution of the neighborhood configurations from image  $X$ . And each configuration contributes to a known

number of vertices, edges and faces:

$$v = \sum_{\alpha=0}^{15} v_\alpha h(\alpha) \quad (32.5)$$

$$e = \sum_{\alpha=0}^{15} e_\alpha h(\alpha) \quad (32.6)$$

$$f = \sum_{\alpha=0}^{15} f_\alpha h(\alpha) \quad (32.7)$$

The following table gives the values of  $v_\alpha$ ,  $e_\alpha$  and  $f_\alpha$  for each cell configuration.



1. Compute the distribution of the neighborhood configurations from image  $X$ .
2. Deduce the number of vertices, edges and faces for each cell representation and compare these values with the previous (manually computed) results.

Table 32.2: Contributions of the neighborhood configurations to the computation of  $v$ ,  $e$  and  $f$

intrapixel cells																
$\alpha$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$f_\alpha$	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
$e_\alpha$	0	2	1	2	1	2	2	2	0	2	1	2	1	2	2	2
$v_\alpha$	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
interpixel cells																
$\alpha$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$f_\alpha$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
$e_\alpha$	0	0	0	1	0	1	0	2	0	0	0	1	0	1	0	2
$v_\alpha$	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

## 32.3

## Crofton perimeter

The Crofton perimeter could be computed form the number of intercepts in different random directions. In discrete case, only the directions  $0, \pi/4, \pi/2$  and  $3\pi/4$  are considered; they are selected according to the desired connexity.

The number of intercepts are denoted  $i_0, i_{\pi/4}, i_{\pi/2}$  and  $i_{3\pi/4}$  for the orientation angles  $0, \pi/4, \pi/2$  and  $3\pi/4$  respectively.

In this way, the Crofton perimeter (in 4 and 8 connexity) is defined in discrete case as:

$$P_4 = \frac{\pi}{2} (i_0 + i_{\pi/2}) \quad (32.8)$$

$$P_8 = \frac{\pi}{4} \left( i_0 + \frac{i_{\pi/4}}{\sqrt{2}} + i_{\pi/2} + \frac{i_{3\pi/4}}{\sqrt{2}} \right) \quad (32.9)$$

These perimeter measurements can be computed from the neighborhood configurations of the original image:

$$P_4 = \sum_{\alpha=0}^{15} P_\alpha^4 h(\alpha) \quad (32.10)$$

$$P_8 = \sum_{\alpha=0}^{15} P_\alpha^8 h(\alpha) \quad (32.11)$$

with the following weights  $P_\alpha^4$  and  $P_\alpha^8$  of these linear combinations (Tab. 32.3):

Table 32.3: Weights for the computation of the Crofton perimeter

$\alpha$	0	1	2	3	4	5	6	7
$P_\alpha^4$	0	$\frac{\pi}{2}$	0	0	0	$\frac{\pi}{2}$	0	0
$P_\alpha^8$	0	$\frac{\pi}{4} \left( 1 + \frac{1}{\sqrt{2}} \right)$	$\frac{\pi}{4\sqrt{2}}$	$\frac{\pi}{2\sqrt{2}}$	0	$\frac{\pi}{4} \left( 1 + \frac{1}{\sqrt{2}} \right)$	0	$\frac{\pi}{4\sqrt{2}}$
$\alpha$	8	9	10	11	12	13	14	15
$P_\alpha^4$	$\frac{\pi}{2}$	$\pi$	0	0	$\frac{\pi}{2}$	$\pi$	0	0
$P_\alpha^8$	$\frac{\pi}{4}$	$\frac{\pi}{2}$	$\frac{\pi}{4\sqrt{2}}$	$\frac{\pi}{4\sqrt{2}}$	$\frac{\pi}{4}$	$\frac{\pi}{2}$	0	0



Evaluate the perimeters  $P_4$  and  $P_8$  with the previous formula on Fig.32.1



## 32.4. Matlab correction



### 32.4.1 Cell configuration

For intrapixels representation, the following values are (manually) counted:

$$f^{intra} = 50$$

$$e^{intra} = 158$$

$$v^{intra} = 107$$

For interpixels representation, the following values are (manually) counted:

$$f^{inter} = 4$$

$$e^{inter} = 42$$

$$v^{inter} = 50$$

Then, it is easy to compute the following values:

$$A = f^{intra} = 50$$

$$P = -4f^{intra} + 2e^{intra} = 116$$

$$\chi_8 = v^{intra} - e^{intra} + f^{intra} = -1$$

$$\chi_4 = v^{inter} - e^{inter} + f^{inter} = 12$$

### 32.4.2 Neighborhood configuration

The configuration is computed using the convolution function `conv2`.



```

F = [1 4; 2 8];
2 XF = conv2(double(X),F, 'same');
h = hist (XF(:,16));
4 bar (0:15, h);

```

Then, the functionals are computed with the following lines. One should get the same values as previously counted.



```

f_intra = [0 1 0 1 0 1 0 1 0 1 0 1 0 1];
2 e_intra = [0 2 1 2 1 2 2 0 2 1 2 1 2 2];
v_intra = [0 1 1 1 1 1 1 1 1 1 1 1 1 1];
4 EulerNb8 = h*v_intra' - h*e_intra' + h*f_intra'

```



```
f_inter = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1];  
6 e_inter = [0 0 0 1 0 1 0 2 0 0 0 1 0 1 0 2];  
v_inter = [0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1];  
8 EulerNb4 = h*v_inter' - h*e_inter' + h*f_inter'  
Area = h*f_intra'  
10 Perimeter = -4*h*f_intra' + 2*h*e_intra'
```

### 32.4.3 Crofton perimeter

The Crofton perimeter is computed with the same strategy in 2 and 4 directions, respectively:



```
Perimeter4 = [0 pi/2 0 0 0 pi/2 0 0 pi/2 pi 0 0 pi/2 pi 0 0];  
2 P4 = h*Perimeter4'  
Perimeter8 = [0 pi /4*(1+1/( sqrt (2))) pi /(4* sqrt (2)) pi /(2* sqrt (2)) 0 pi /4*(1+1/(  
    ↪ sqrt (2))) 0 pi /(4* sqrt (2)) pi /4 pi/2 pi /(4* sqrt (2)) pi /(4* sqrt (2)) pi /4  
    ↪ pi/2 0 0];  
4 P8 = sumh*Perimeter8'
```

The obtained values are:

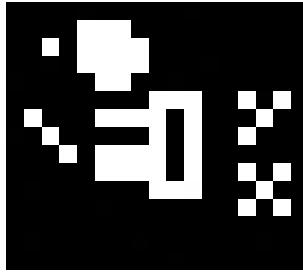
#### Command window

```
P4 = 91.1062  
2 P8 = 77.7640
```

The objective of this tutorial is to classify all foreground points of a binary image according to their topological significance: interior, isolated, border.... The reader can refer to [?] for more details.

The different processes will be realized on the following binary image.

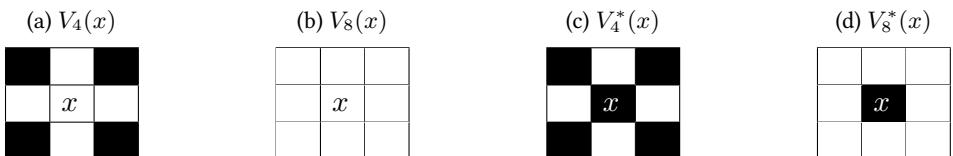
Figure 33.1: Test



### Preliminary definitions:

- $y$  is 4-adjacent to  $x$  if  $|y_1 - x_1| + |y_2 - x_2| \leq 1$ .
- $y$  is 8-adjacent to  $x$  if  $\max(|y_1 - x_1|, |y_2 - x_2|) \leq 1$ .
- $V_4(x) = \{y : y \text{ is 4-adjacent to } x\}; V_4^*(x) = V_4(x) \setminus \{x\}$ .
- $V_8(x) = \{y : y \text{ is 8-adjacent to } x\}; V_8^*(x) = V_8(x) \setminus \{x\}$ .

Figure 33.2: Different neighborhoods. By convention, pixels in white are of value 1, in black of value 0.



- a n-path is a point sequence  $(x_0, \dots, x_k)$  with  $x_j$  n-adjacent to  $x_{j-1}$  for  $j = 1, \dots, k$ .
- two points  $x, y \in X$  are n-connected in  $X$  if there exists a n-path  $(x = x_0, \dots, x_k = y)$  such that  $x_j \in X$ . It defines an equivalence relation.
- the equivalence classes of the previous binary relation are the n-components of  $X$ .

## 33.1 Connectivity numbers

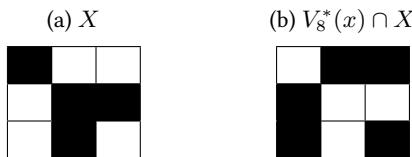
Let  $Comp_n(X)$  be the number of n-components ( $n = 4$  or  $n = 8$  within the selected topology  $V_4$  or  $V_8$ ) of the set  $X$  of foreground points (object). We define the following set:

$$CAdj_n(x, X) = \{C \in Comp_n(X) : C \text{ is } n\text{-adjacent to } x\}$$

We select the 8-connectivity for the set  $X$  of foreground points (object) and the 4-connectivity for the complementary  $\bar{X}$ .

Warning: the definition of  $CAdj_n$  introduces the n-adjacency to the central pixel  $x$ . In the case of the following configuration (Fig.33.3),  $T_8 = 2$ ,  $\bar{T}_8 = 2$  and  $TT_8 = 3$ .

Figure 33.3: The pixel in the bottom right corner is not C-adjacent-4 to  $x$ .



1. Create a function for determining the connectivity number:  
 $T_8(x, X) = \#CAdj_8(x, V_8^*(x) \cap X),$
2. Create a function for determining the connectivity number:  
 $\bar{T}_8(x, X) = \#CAdj_4(x, V_8^*(x) \cap \bar{X}),$
3. Create a function for determining the number:  
 $TT_8(x, X) = \#(V_8^*(x) \cap X).$

Test these functions on some foreground points of the image 'test'.



Use the functions `bwlabel` and `bwlabeled`.

## 33.2 Topological classification of binary points

From the connectivity numbers  $T_8(x, X)$ ,  $\bar{T}_8(x, X)$  and  $TT_8(x, X)$ , it is possible to classify a foreground point  $x$  within the binary image  $X$  according to its topological

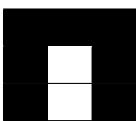
signification:

$T_8(x, X)$	$\bar{T}_8(x, X)$	$TT_8(x, X)$	Type
0	1		isolated point
1	0		interior point
1	1	$> 1$	border point
1	1	1	end point
2	2		2-junction point
3	3		3-junction point
4	4		4-junction point

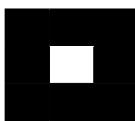
The following Fig. 33.4 shows the classification of 4 points.

Figure 33.4: Points configurations.

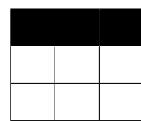
(a) end



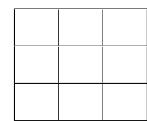
(b) isolated



(c) border



(d) interior



With the help of this table, classify the points of the image 'test'.



### 33.3. Matlab correction



#### 33.3.1 Toplogical description

The connectivity numbers are computed in the following way:



```

function [T8,T8c,TT8]=nc(A)
2 % A : block 3x3, binary

4 % complementary set of A
invA=1-A;

6
% neighborhoods
8 V8=ones(3,3);
V8_star=[1 1 1;1 0 1;1 1 1];
10 V4=[0 1 0;1 1 1;0 1 0];

12 % intersection is done by the min operation
X1=min(V8_star,A);
14 TT8=sum(X1(:));
 [~, T8] = bwlabeln(X1,8);

16
% The C-adj-4 might introduce some problems if a pixel is not 4-connected
18 % to the central pixel
X2=min(V8,invA);
20 Y=min(X2,V4);
X=imreconstruct(Y,X2,4);
22 [~, T8c]=bwlabeln(X,4);

```

They are here applied on the original 'test' image at the specific point with coordinates (2, 5):



```

1 % reading image
A=imread('test.bmp');
3 A=double(A (:,:,1) );
A=A/255;
5 figure ;
x=[2 5];
7 X=A(x(1)-1:x(1)+1,x(2)-1:x(2)+1);
subplot (1,2,1) ;viewImage(A); title ('original image');
9 subplot (1,2,2) ;viewImage(X); title ('3x3 window centered on the point (2,5)');
% connectivity numbers
11 [nc1,nc2,nc3]=nc(X)

```

The current 3x3 window is represented in Fig.33.5.

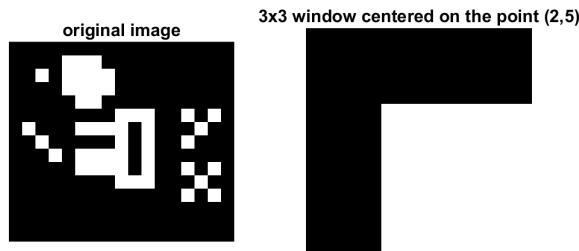


Figure 33.5: Representation of the current 3x3 window.

giving the following connectivity numbers:

**Command window**

```

1 nc1 = 1
2 nc2 = 1
3 nc3 = 3

```

### 33.3.2 Topological classification of binary points

The following function gives the classification of the current point according to the 3 connectivity numbers. Note that the topological description of a point is given by a value between 1 and 7.



```

1 function y=nc_type(n);
2 [a,b,c]=nc(n);
3 if (a==0) y=1;end % isolated point
4 if ((a==1) && (b==1) && (c>1)) y=5;end % border point
5 if (b==0) y=7;end % interior point
6 if ((a==1) && (b==1) && (c==1)) y=6;end % end point
7 if (a==2) y=2;end % 2-junction point
8 if (a==3) y=3;end % 3-junction point
9 if (a==4) y=4;end % 4-junction point

```

Consequently, each point of a binary image can now be topologically classified:



```

1 % for the whole image
2 [m,n]=size(A);
3 B=zeros(m,n);
4 for i=2:m-1
5     for j=2:n-1

```



```
if A(i,j)> 0
    X=A(i-1:i+1,j-1:j+1);
    B(i,j)=nc_type(X);
end
end
end
disp('Point classification :');
13 disp('1 : isolated points');
14 disp('2 : 2-junction points');
15 disp('3 : 3-junction points');
16 disp('4 : 4-junction points');
17 disp('5 : border points');
18 disp('6 : end points');
19 disp('7 : interior points');

21 subplot (3,3,1) ;viewImage(A); title (' original image');
22 subplot (3,3,2) ;viewImage(B); title (' point classification ');
23 subplot (3,3,3) ;viewImage(B==5); title (' border points ');
24 subplot (3,3,4) ;viewImage(B==7); title (' interior points ');
25 subplot (3,3,5) ;viewImage(B==1); title (' isolated points ');
26 subplot (3,3,6) ;viewImage(B==6); title (' end points ');
27 subplot (3,3,7) ;viewImage(B==2); title (' 2-junction points ');
28 subplot (3,3,8) ;viewImage(B==3); title (' 3-junction points ');
29 subplot (3,3,9) ;viewImage(B==4); title (' 4-junction points ')
```

with the following result:

Note that the following function `viewImage` has been used to display the different images of this tutorial:



```
1 function viewImage(A)
2 B=double(A);
3 mmax=max(max(B));
4 mmin=min(min(B));
5 if (mmax == mmin) B=0;
6 else B=uint8(255*(B-min(min(B))/(max(max(B))-min(min(B))));
```

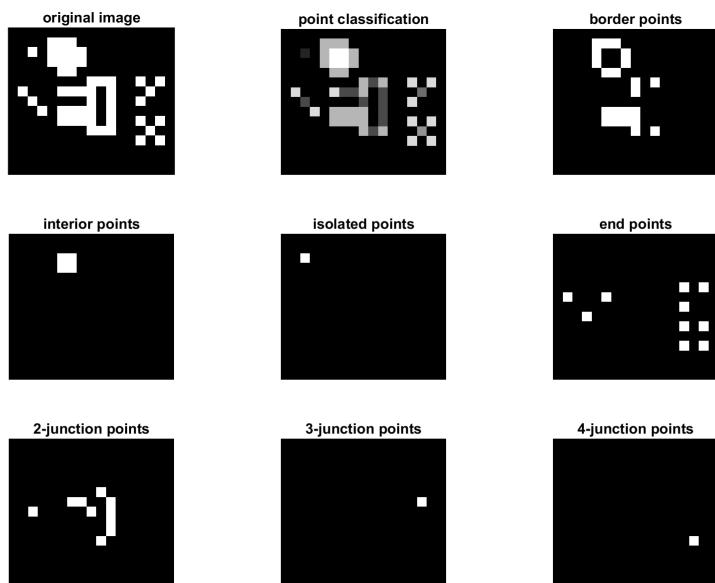


Figure 33.6: Topological classification of the image points.

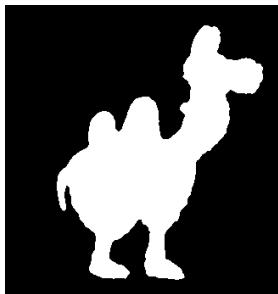


This tutorial aims to characterize objects by geometrical and morphometrical measurements.

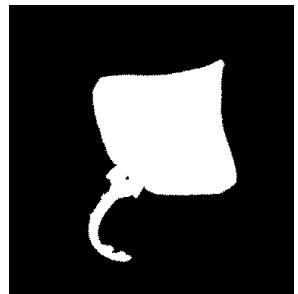
The different processes will be applied on synthetic images as well as images from the Kimia database [?, ?]:



(a) Bat.



(b) Camel.



(c) Ray.

## 34.1 Perimeters

We are going to calculate the perimeter using the Crofton formula. This formula consists in integrating the intercept number of the object with lines of various orientation and positions. Its expression in the 2-D planar case is given by:

$$P(X) = \pi \int \chi(X \cap L) dL$$

where the Euler-poincaré characteristic  $\chi$  is equal to the number of connected components of the intersection of  $X$  with a line  $L$ .

In the discrete case, the Crofton formula can be estimated by considering the intercept numbers for the horizontal  $i_0$ , vertical  $i_{\pi/2}$  and diagonal orientations  $i_{\pi/4}$  and  $i_{3\pi/4}$  as:

$$P(X) = \pi \times \frac{1}{4} \left( i_0 + \frac{1}{\sqrt{2}} i_{\pi/4} + i_{\pi/2} + \frac{1}{\sqrt{2}} i_{3\pi/4} \right)$$



1. Calculate the intercept number of a binary object from the Kimia database with lines oriented in the following four directions:  $0, \pi/4, \pi/2, 3\pi/4$ .
2. Deduce the value of the Crofton perimeter.
3. Compare the result with the classical perimeter functions.

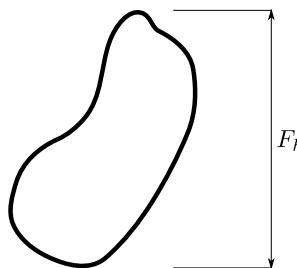


See bwperim.

## 34.2 Feret Diameter

The Feret diameter (a.k.a. the caliper diameter) is the length of the projection of an object in one specified direction Fig.34.1.

Figure 34.1: Feret diameter of the object in horizontal direction.



1. Calculate the projections in different directions of a binary object from the Kimia database.
2. Deduce the minimum, maximum and mean value of the Feret diameters.

## 34.3 Circularity

We want to know if the object  $X$  is similar to a disk. For that, we define the following measurement (circularity criterion):

$$\text{circ}(X) = \frac{4\pi A(X)}{P(X)^2}$$

where  $A(X)$  and  $P(X)$  denote the area and perimeter of the object  $X$ .



1. Show that the circularity of a disc is equal to 1.
2. Generate an array representing an object as a discrete disc.
3. Calculate its circularity and comment the results.



Use [meshgrid](#).

## 34.4

## Convexity

We want to know if the object  $X$  is convex. For that, we define the following measurement:

$$\text{conv}(X) = \frac{A(X)}{A(\text{CH}(X))}$$

where  $\text{CH}(X)$  denotes the filled convex hull of the object  $X$ .



1. Compute the convex hull of a pattern from the Kimia database.
2. Evaluate the area of the filled convex hull.
3. Deduce the convexity of the pattern.



See [convhull](#) and [poly2mask](#).



## 34.5. Matlab correction



### 34.5.1 Perimeters

The convolution is used here as an easy way of getting the borders of the object in one direction, i.e. counting the number of intercepts. This method is efficient because, for one given direction, a high number of lines are considered.



```
% load the image
2 I = imread('camel-5.png');
I=double(I)>0; % ensure binarization
4 imshow(I)

6 % defines an orientation , computes the number of intercepts by convolution
h = [-1 1];
8 b = abs(conv2(double(I), h, 'same'));
n1 = sum(b(:))/2;
10 b = abs(conv2(double(I), h', 'same'));
12 n2 = sum(b(:))/2;

14 % diagonal orientations :
h = [1 0; 0 -1];
16 b = abs(conv2(double(I), h, 'same'));
n3 = sum(b(:))/2;
18 h = [0 1; -1 0];
b = abs(conv2(double(I), h, 'same'));
20 n4 = sum(b(:))/2;

22 % crofton evaluation and comparison
perim_Crofton = pi/4 * sum( [n1 n2 n3/sqrt(2) n4/sqrt(2) ] )
24 perim_usual = sum(sum(bwperim(I)))
```

The b matrix obtained for intercept evaluation is illustrated in Fig.34.2. The results obtained are:

#### Command window

```
perim_Crofton =
2      1.3055e+03

4 perim_usual =
1182
```

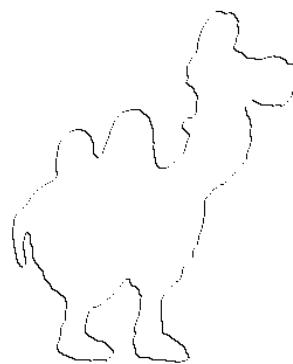


Figure 34.2: Illustration of the intercept number evaluation in the first diagonal direction.

### 34.5.2 Feret diameter

The code consists in rotating the image and evaluating the projected diameter. The rotation function can interpolate the pixel, a binarization step is thus required. The function `sum` indirectly performs the projection as it sum all values in one direction only.



```
% discrete orientations
2 deg = 1:180;
for i = 1:length(deg)
    4 I2 = imrotate(I,deg(i), 'nearest');
    I3 = sum(I2)>0;
    6 diameter(i) = sum(I3);
end
8
diamFeret_min = min(diameter)
10 diamFeret_max = max(diameter)
diamFeret_mean = mean(diameter)
```

For the camel image, the Feret diameters are:

**Command window**

```

1 diamFeret_min =
  184
3
5 diamFeret_max =
  326
7 diamFeret_mean =
  267.1944

```

### 34.5.3 Circularity

For a disk, the perimeter is  $\pi \cdot D$  and the surface is  $\pi \cdot \frac{D^2}{4}$ .

In order to generate a binary image containing a disk, one simple way is to use the formula:  $(x - x_0)^2 + (y - y_0)^2 \leq R^2$ . This gives with matlab, for an image of size  $1000 \times 1000$ , a circle of radius 400 centered in (500, 500):



```

I = zeros (1000,1000) ;
2 [X,Y] = meshgrid (1:1000,1:1000) ;
I = sqrt ((X-500).^2+(Y-500).^2) < 400;

```

The perimeters are evaluated with the code proposed earlier. The results are show that the Crofton perimeter is more precise in this example.

**Command window**

```

1 >> 2*pi*400
ans =
  2.5133e+03
3
5 >> perim_Crofton =
  2.5113e+03
7
9 >> perim_usual =
  2260
11
13 >> circ_Crofton =
  1.0015
15 >> circ_usual =
  1.2366

```

### 34.5.4 Convexity

In order to compute the convexity criterion, the convex hull is computed. For this purpose, all the pixels of the objects are converted into points. The polygon is then transformed into a binary image. The resulting image of the convex hull is presented in Fig.34.3.



```
1 I = imread('camel-5.png');
I = double(I)>0;
3
dim = size(I);
5 D1 = dim(1);
D2 = dim(2);
7
[Y,X] = find(I==1);
9 CH = convhull(X,Y);
XCH = X(CH);
11 YCH = Y(CH);
I_convhull = poly2mask(XCH,YCH,D1,D2);
13
figure
15 subplot(121);imshow(I);
subplot(122);imshow(I_convhull)
17
convexity = sum(I(:))/sum(I_convhull(:))
```

#### Command window

```
convexity = 0.6439
```

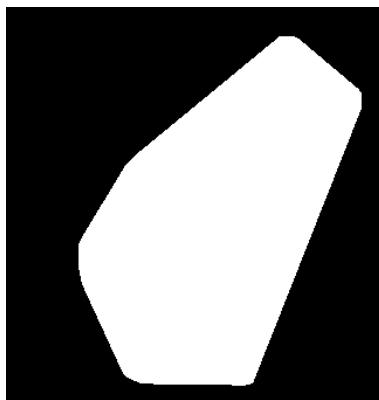


Figure 34.3: Convex hull of the camel object.



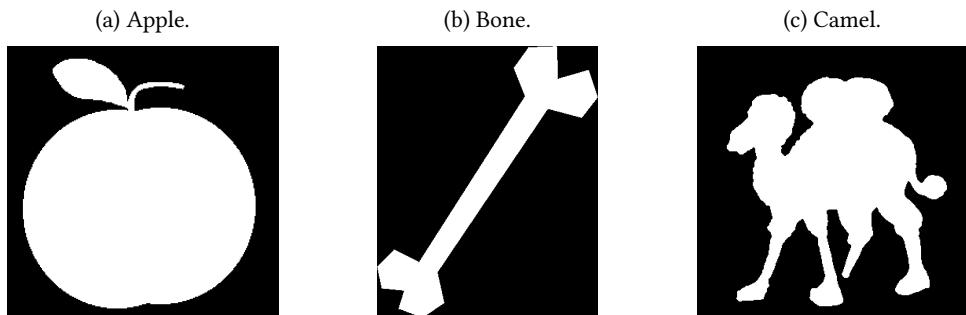
The objective is to study some shape diagrams and the possibility to define properties that may be useful in order to distinguish the different objects.

Shape diagrams are representations of single shapes (connected compact sets, see [?, ?, ?]) as points in the 2-D unit square plane. They are based on inequalities between 6 geometrical measurements: area  $A$ , perimeter  $P$ , radius of the inscribed circle  $r$ , radius of the circumscribed circle  $R$ , minimum Feret diameter  $\omega$  and maximum Feret diameter  $d$ . In this way, the morphometrical functionals used in the different shape diagrams are normalized ratios of such geometrical functionals. The following table shows the morphometrical functionals for non-convex sets:

Geometrical functionals	Inequalities	Morphological functionals
$r, R$	$r \leq R$	$r/R$
$\omega, R$	$\omega \leq 2R$	$\omega/2R$
$A, R$	$A \leq \pi R^2$	$A/\pi R^2$
$d, R$	$d \leq 2R$	$d/2R$
$r, d$	$2r \leq d$	$2r/d$
$\omega, d$	$\omega \leq d$	$\omega/d$
$A, d$	$4A \leq \pi d^2$	$4A/\pi d^2$
$R, d$	$\sqrt{3}R \leq d$	$\sqrt{3}R/d$
$r, P$	$2\pi r \leq P$	$2\pi r/P$
$\omega, P$	$\pi\omega \leq P$	$\pi\omega/P$
$A, P$	$4\pi A \leq P^2$	$4\pi A/P^2$
$d, P$	$2d \leq P$	$2d/P$
$R, P$	$4R \leq P$	$4R/P$
$r, A$	$\pi r^2 \leq A$	$\pi r^2/A$
$r, \omega$	$2r \leq \omega$	$2r/\omega$

Table 35.1: Morphometrical functionals.

Figure 35.1: The different processes will be applied on images from the Kimia database [?, ?].



## 35.1 Geometrical functionals



Code functions in order to evaluate the different parameters:

- the area, Crofton perimeter and Feret diameters have been already presented in tutorial 32;
- the radius of the inscribed circle can be defined from the ultimate erosion of a set.



The function `bwdist` computes the distance map of a binary image.

## 35.2 Morphometrical functionals



Code and evaluate some of the morphometrical functionals listed in the table 35.1. Note that each of them has a physical meaning, e.g.  $\frac{4\pi A}{P^2}$  (circularity),  $\frac{4A}{\pi d^2}$  (roundness),  $2\omega/P$  (thinness).

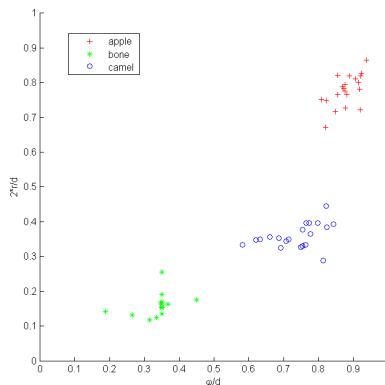
## 35.3

# Shape diagrams



- Visualize the different shape diagrams for all the images (from the Kimia database) within the three classes 'apple', 'bone' and 'camel'. The Fig.35.2 illustrates the result for the shape diagram ( $x = 2r/d$ ,  $y = P/\pi d$ ).
- Which shape diagram is the most appropriate for the discrimination of such objects?

Figure 35.2: Example of a shape diagram.



## 35.4

# Shape classification



- Use a K-means clustering method for automatic classification of such shapes.
- Propose a method to quantify the classification accuracy for each shape diagram.



## 35.5. Matlab correction



### 35.5.1 Geometrical functionals

The Crofton perimeter and Feret diameters have been already computed in the tutorial 32 For computing the perimeter, we use the regionprops function.



```
% classical perimeter
1 if (~ islogical (I))
    perim = regionprops(I>100, 'Perimeter');
4 else
    perim = regionprops(I, 'Perimeter');
6 end
p = perim.Perimeter;
```

The Crofton is evaluated with the use of 4 directions.



```
1 I=double(I);
  inter = zeros (4,1) ;
3 for i=1:4
    I2=imrotate(I, 45*(i-1));
5 h=[1 -1];
  I3=conv2(I2, h, 'same')>0;
7 inter (i) = sum(I3 (:));
end
9
% formula of the Crofton perimeter
11 crofton = pi /4*( inter (1)+inter (3) + ( inter (2)+inter (4))/sqrt (2));
```

The Feret diameter corresponds to the projected length in a given direction. The following code uses a angular step of 30 degrees. It computes the maximum and the minimum at the same time.



```
1 function [d,D] = feret (I)
3 d=max(size(I));
D=0;
5 for a=0:30:179
    I2 = imrotate(I, a);
7 F=max(I2);
  mesure=sum(F>0);
9
```



```

11 if (mesure<d)
12     d=mesure;
13 end
14 if (mesure>D)
15     D=mesure;
16 end
17 end

```

The radius of the inscribed circle can be computed as:



```

function r=InscribedCircleRadius (I)
2 dm = bwdist(~(I>0));
r=max(dm(:));

```

## 35.5.2 Morphometrical functionals

The different morphometrical functionals can be easily computed as ratios of geometrical functionals.

## 35.5.3 Shape diagrams

The following process gives 3 shape diagrams for the whole Kimia image database:



```

1 name={'apple-' 'bone-' 'camel-'};
2 couleur={'r+' 'g*' 'bo'};
3 format=' bmp';
4 indices =1:20;
5
6 % number of diagrams:
7 N_diag=3;
8 x=zeros(N_diag, length(name)*length(indices));
9 y=zeros(N_diag, length(name)*length(indices));
10
11 xlabs={'\omega/d' '\omega/d' '2*r/d'};
12 ylabs={'2*r/d' '4A/(pi*d^2)' 'P/(pi*d)' };
13
14 % computation of the functionals for all the images
15 for n=1:length(name)
16     for i=indices
17         I = imread([name{n} num2str(i) format]);
18         [omega d] = feret (I);
19

```



```

21 [ crofton , P ] = perimetres(I);
r = rayonInscrit (I);
stats = regionprops(I>0, 'Area');
ii = (n-1)*length(indices)+i;
x(1, ii )=omega/d;
x(2, ii )=omega/d;
x(3, ii )=2*r/d;
27
y(1, ii )=2*r/d;
29 y(2, ii )=4* stats .Area/(pi*d^2);
y(3, ii )=P/(pi*d); % values > 1, non-convex objects
31
% progress bar
33 waitbar (((n-1)*length(indices)+i)/(length(name)*length(indices)));
end
35 end

```

The visualization is done via the following code and displayed in Fig.35.3.



```

1 % visualization of the 3 diagrams with a specific color for each category
for j=1:N_diag
2   figure(); hold on
3   for i=1:length(name)
4     i1=length(indices)*(i-1)+1;
5     i2=length(indices)*i;
6     plot(x(j,i1:i2), y(j,i1:i2),[ couleur{i} ]);
7
8   end
9   legend(name);
10  xlabel(xlabs{j});
11  ylabel(ylabs{j});
13 end

```

### 35.5.4 Shape classification

The MATLAB® function kmeans is used for classifying the shapes into 3 classes:



```

X1=x(1,:);
2 X2=x(2,:);
X3=x(3,:);
4 Y1=y(1,:);
Y2=y(2,:);
6 Y3=y(3,:);

```

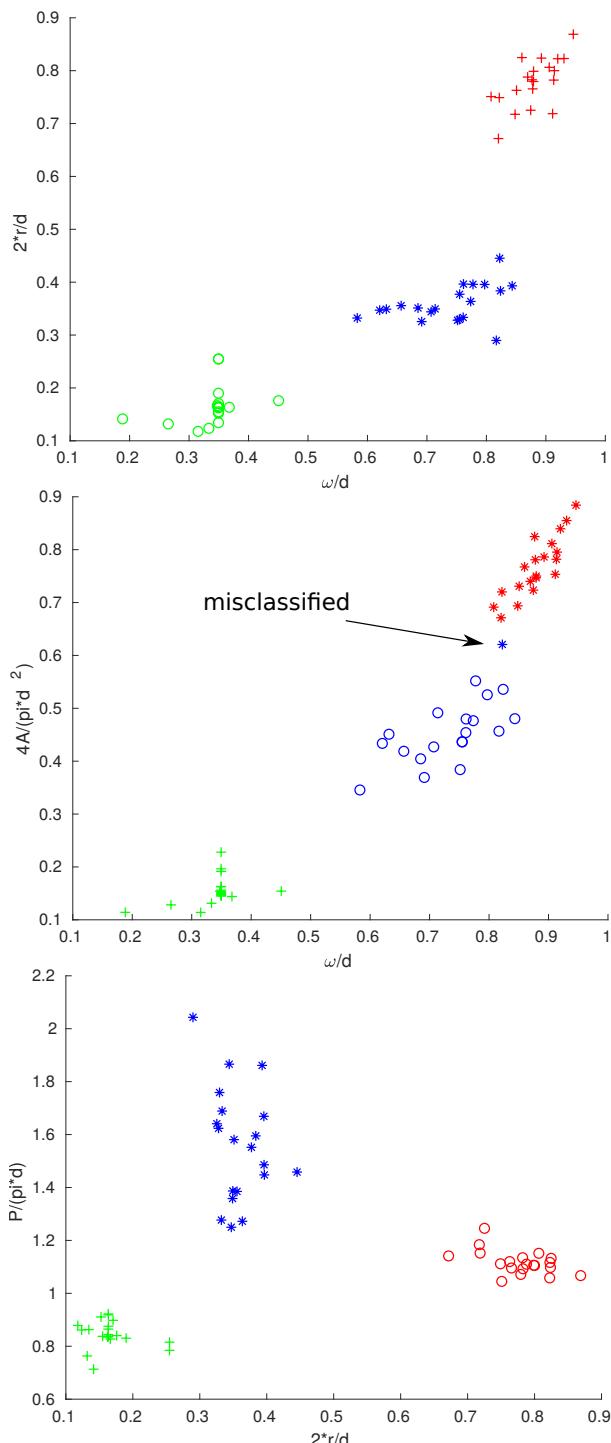


Figure 35.3: Three shape diagrams for the three binary images from the Kimia database, and their classification by the kmeans algorithm. The color denotes the real class, and the shape denotes the results of the kmeans algorithm.



```
1 idx1 = kmeans([X1',Y1'],3);  
8 idx2 = kmeans([X2',Y2'],3);  
idx3 = kmeans([X3',Y3'],3);
```

The classification accuracy for each shape diagram is then computed as a percentage of correctly positioned shapes.



```
1 accuracy1 = (sum(idx1(1:20)==mode(idx1(1:20))) + sum(idx1(21:40)==mode(idx1(21:40))  
    ↪ )) + sum(idx1(41:60)==mode(idx1(41:60)))/60*100  
accuracy2 = (sum(idx2(1:20)==mode(idx2(1:20))) + sum(idx2(21:40)==mode(idx2(21:40))  
    ↪ )) + sum(idx2(41:60)==mode(idx2(41:60)))/60*100  
3 accuracy3 = (sum(idx3(1:20)==mode(idx3(1:20))) + sum(idx3(21:40)==mode(idx3(21:40))  
    ↪ )) + sum(idx3(41:60)==mode(idx3(41:60)))/60*100
```

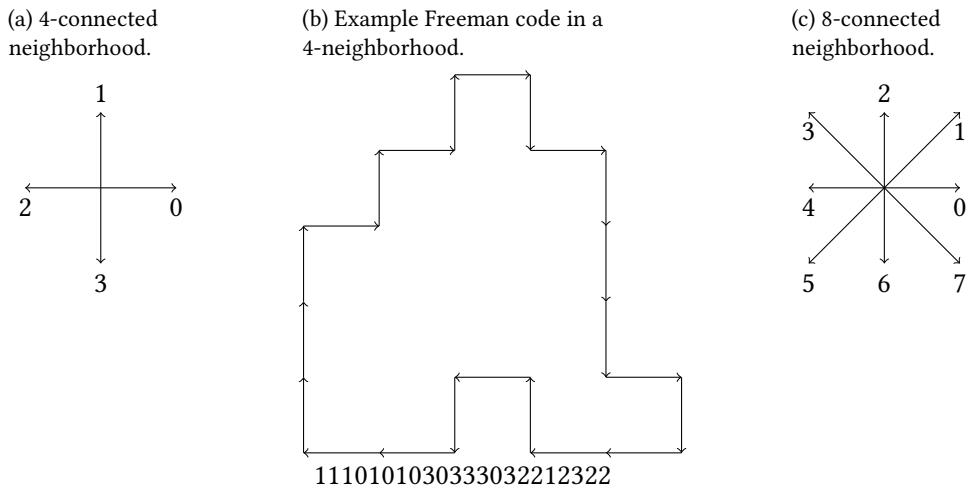
with the following results. This quantifies the error done in second diagram (Fig.35.3).

### Command window

```
1 accuracy1 = 100  
accuracy2 = 98.3333  
3 accuracy3 = 100
```

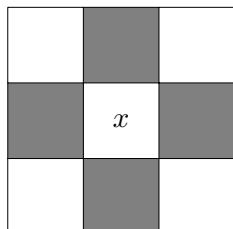
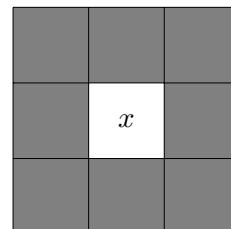
This tutorial is focused on shape representation by the Freeman chain code. This code is an ordered sequence of connected segments (of specific sizes and directions) representing the contour of the shape to be analyzed. The direction of each segment is encoded by a number depending on the selected connectivity (Fig. 36.1). In this example, the contour of the shape and its Freeman code are given in 4-connectivity.

Figure 36.1: Freeman chain code examples.



### Notations:

- $y$  is 4-adjacent to  $x$  if  $|y_1 - x_1| + |y_2 - x_2| \leq 1$ .
- $y$  is 8-adjacent to  $x$  if  $\max(|y_1 - x_1|, |y_2 - x_2|) \leq 1$ .
- $N_4(x) = \{y : y \text{ is 4-adjacent to } x\}; N_4^*(x) = N_4(x) \setminus \{x\}$ .
- $N_8(x) = \{y : y \text{ is 8-adjacent to } x\}; N_8^*(x) = N_8(x) \setminus \{x\}$ .

(a)  $V_4(x)$ (b)  $V_8(x)$ 

## 36.1 Shape contours

Before determining the Freeman chain code of the shape, it is necessary to extract its contour.



1. Generate or load a simple shape as a binary image  $A$ .
2. Extract its contour  $C_4(A)$  ou  $C_8(A)$  according to the 4-connectivity or the 8-connectivity, respectively:

$$x \in C_4(A) \Leftrightarrow \exists y \in N_8(x) \quad y \in {}^cA$$

$$x \in C_8(A) \Leftrightarrow \exists y \in N_4(x) \quad y \in {}^cA$$



Informations

You can use the function `bwperim` with the appropriate connectivity number.

## 36.2 Freeman chain code

From the shape contours, the Freeman chain code can be calculated.



1. From the binary array of pixels, extract the first point belonging to the shape (from left to right, top to bottom).



You can use `find` to locate the first point.

2. From this initial point, determine the Freeman chain code  $c$  (counter-clockwise direction) using the  $N_4$  or  $N_8$  connectivity.

## 36.3

# Normalization

The Freeman chain code is depending on the initial point and is not invariant to shape rotation. It is then necessary to normalize this code.

1. The first step consists in defining a differential code  $d$  from the code  $c$  :

$$d_k = c_k - c_{k-1} \pmod{4 \text{ or } 8}$$

In this example,  $d = 3003131331300133031130$ ;

2. The second step consists in normalizing the code  $d$ . We have to extract the lowest number  $p$  from all the cyclic translations of  $d$ .

In this example,  $p = 0013303113030031313313$ .

This Freeman chain code  $p$  is then independant from the initial point and invariant by rotations of angles  $k * \pi/2$  radians ( $k \in \mathbb{Z}$ ).



- Code above steps 1 and 2. Prototypes of functions are given.
- Validate your code by rotating the shape of  $3\pi/2$  radians.



```
function d=codediff(fcc,conn)
```

 Use `circshift` for circularly shifting the freeman code array. Use `polyval` for transforming the array of values into a number, although rounding errors might occur, due to numerical approximation of floating numbers.

## 36.4 Geometrical characterization

### 36.4.1 Perimeter

From the Freeman chain code, it is possible to make different geometrical measurement of the shape.



Calculate the perimeter of the shape (take care of the diagonals).

### 36.4.2 Area

The area is evaluated by the following algorithm:

- The area is initialized to 0 and the parameter  $B$  is initialized to 0.
- For each iteration on the Freeman chain code  $c$ , the area and the parameter value  $B$  are incremented with the following rules:

8-code	area	$B$
0	- $B$	0
1	- $B-0.5$	1
2	0	1
3	$B+0.5$	1
4	$B$	0
5	$B-0.5$	-1
6	0	-1
7	- $B+0.5$	-1



Calculate the area of the shape in 8-connectivity. Note that the area does not correspond to the total number of pixels belonging to the shape.



## 36.5. Matlab correction



### 36.5.1 Shape contours

To generate a simple object, here is an example:

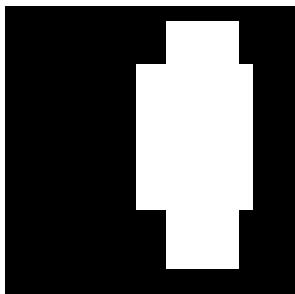


```
A=zeros(20,20);
2 A (5:14,10:17) =1;A (2:18,12:16) =1;
```

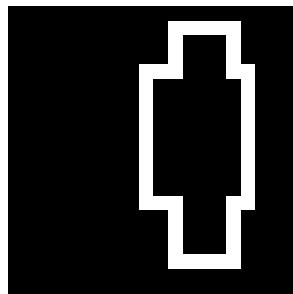
The contours are computed in 4- or 8-connectivity, see Fig.36.2.



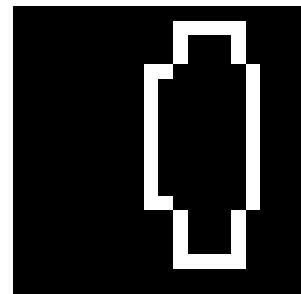
```
% compute the contours
2 contours8 = bwperim(A, 4);
contours4 = bwperim(A, 8);
```



(c) Sample object.



(d) Contour in 4-connectivity.



(e) Contour in 8-connectivity.

Figure 36.2: Simple object and its contours in 4- or 8-connectivity.

### 36.5.2 Freeman chain code

First point of the shape

The important thing is to find one point in the contour. The Freeman chain code is sensitive to this choice, but several methods can transform this code so that this first choice does not have any importance.



```

1 function [x,y]= firstPoint (A)
% locates first non zero point of the contour A
3 [r,c]=find (A);
x=r(1); y=c(1);

```

### Command window

```

>>[r0,c0]= firstPoint (A)
2
r0 =
4      5
c0 =
6      10

```

### Freeman chain code



```

function code=freeman(A,r0,c0,conn)
2 % freeman code of a contour.
% A : contour
4 % r0, c0 : coordinates of 1st point
% conn: connectivity
6
B=A;
8 stop=0; % stop condition
point0=[r0,c0];
10 if (conn==8)
    lut=[1 2 3; 8 0 4; 7 6 5];
12 else
    lut=[0 2 0; 8 0 4; 0 6 0];
14 end

16 % be careful that these LUTs consider coordinates
% from left to right, top to bottom
18 % 0
% 0+----- y
20 % |
% |
22 % |
% x|
24 %
lutx=[-1 -1 -1 0 1 1 1 0];
26 luty=[-1 0 1 1 0 -1 -1];
lutcode=[3 2 1 0 7 6 5 4];

```



```

28
nbrepoints=sum(B(:));
30 code=[];
point=point0;
32
for indice = 1:nbrepoints
    B(point(1),point(2))=0;
    window=B(point(1)-1:point(1)+1,point(2)-1:point(2)+1);
    window=window.*lut;
    index=max(window(:));
    if (index==0) % no more points ? should link to first point
        B(point0(1),point0(2))=1;
        window=B(point(1)-1:point(1)+1,point(2)-1:point(2)+1);
        window=window.*lut;
        index=max(window(:));
        B(point0(1),point0(2))=0;
    end
44
% compute coordinates of new point
point=[point(1)+lutx(index),point(2)+luty(index)];
48
% add code
50 code(indice) = lutcode(index);

52 end

```

### Command window

```

z4= 6 6 6 6 6 6 6 6 0 0 6 6 6 6 0 0 0 0 0 2 2 2 2 0 2 2 2 2 2 2 2 2 4 2 2 2 4 4
4 4 6 6 6 4 4
2
z8= 6 6 6 6 0 0 0 7 0 0 0 0 0 0 0 0 0 0 1 0 0 2 2 2 2 4 4 3 2 4 4 4 4 4 4 4 4 4 6 5
4 4 4

```

### 36.5.3 Normalization

#### Differential code



```

1 function d=codediff(fcc,conn)
% fcc : freeman chain code
3 % conn: connectivity
sr=circshift(fcc,1);
5 d=fcc-sr;

```



```
d = mod(d, conn);
```

## Normalization



```
function code=freeman_normalization(fcc)
2 % find the lowest number constituted among all the cyclic translations of
% fcc: freeman code
4 L = length(fcc);
C = zeros(L);
6 for i=1:L
    C(i,:) = circshift (fcc , i);
8 end

10 % search for minimum number
% evaluates the value of the number formed by the array, thus, use polyval
12 % in decimal basis for getting this number
D= zeros(L, 1);
14 for i=1:L
    D(i) = polyval(C(i,:), 10);
16 end

18 [~, ind]=min(D);
code=C(ind(1),:);
```

The differential code is evaluated in d8, the normalization gives shapenumber8:



```
[r0,c0]= firstPoint (A);
2 z8=freeman(contours8,r0,c0,8) ;
d8=codediff(z8,8) ;
4 shapenumber8=freeman_normalization(d8);
```

### Command window

```
d8 = 2 0 0 0 2 0 0 7 1 0 0 0 0 0 0 0 0 1 7 0 2 0 0 0 2 0 7 7 2 0 0 0 0 0 0 0 0 2 7
      7 0 0
2 shapenumber8 = 0 0 0 0 0 0 0 0 1 7 0 2 0 0 0 2 0 7 7 2 0 0 0 0 0 0 0 2 7 7 0 0 2
      0 0 0 2 0 0 7 1
```

## Validation

This validation shows the effect on a different starting point.



```
% check for another starting point
2 r0=9;c0=10;
z8=freeman(contours8,r0,c0,8);
4 d8=codediff(z8,8);
shapenumber8_startchanged=freeman_normalization(d8);
6 disp('* validation test for starting point changed')
if (shapenumber8-shapenumber8_startchanged)
8 disp(' error: different freeman code')
else
10 disp(' OK: same freeman code')
end
```

Another test is to verify the result after a rotation. To prevent discretization problems, we use 90 degrees and take the transpose of the matrix.



```
1 % check for rotation by 90 deg
contours8rot=contours8';
3
figure ;imshow(contours8rot);
5 [r0,c0]= firstPoint (contours8rot);
7 z8=freeman(contours8rot,r0,c0,8);
d8=codediff(z8,8);
9 shapenumber8_rotated=freeman_normalization(d8);
disp('* validation test after rotation')
11 if (shapenumber8-shapenumber8_rotated)
    disp(' error: different freeman code')
13 else
    disp(' OK: same freeman code')
15 end
```

The same code should be found:

### Command window



```
1 * validation test for starting point changed
OK: same freeman code
3 * validation test after rotation
OK: same freeman code
```

### 36.5.4 Geometrical characterization

#### Perimeter for 8-connectivity

We first need to extract the codes in the diagonal directions and apply a  $\sqrt{2}$  factor, then add the number of codes in vertical and horizontal directions.



```

1 nb_diag=mod(z8, 2);
2 nb_diag=sum(nb_diag(:));
nb = length(z8)-nb_diag;
4 perimeter = nb_diag * sqrt(2) + nb
stats = regionprops(A,'Perimeter')

```

#### Command window

```

1 perimeter =
43.6569
3 stats =
struct with fields:
5 Perimeter: 41.5900

```

#### Area for 8-connectivity



```

1 area=0;
B=0;
3 lutB=[0 1 1 1 0 -1 -1 -1];
for i=1:length(z8)
5 lutArea=[-B -(B+0.5) 0 (B+0.5) B (B-0.5) 0 -(B-0.5)];
area=area+lutArea(z8(i)+1);
7 B=B+lutB(z8(i)+1);
end
9 disp(['Area by freeman code: ' num2str(area)]);
disp(['Number of pixels: ' num2str(sum(A(:)))] )

```

#### Command window

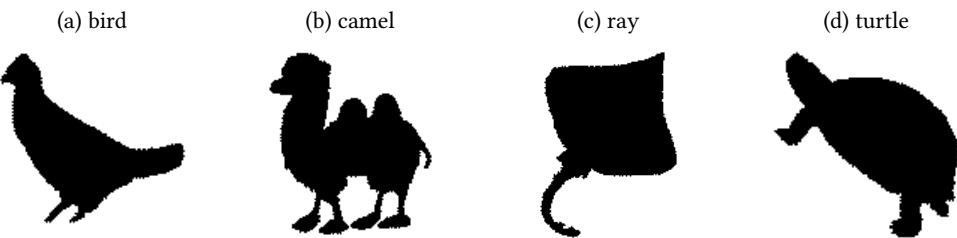
```

1 Area by freeman code: 93
2 Number of pixels: 115

```

The objective of this tutorial is to classify images by using machine learning techniques. Some images, as illustrated in Figure 37.1 of the Kimia database will be used [?, ?].

Figure 37.1: The different processes will be applied on images from the Kimia database. Here are some examples.



## 37.1 Feature extraction

The image database used in this tutorial is composed of 18 classes. Each class contains 12 images. All these 216 images come from the Kimia database. In order to classify these images, we first have to extract some features of each binary image.



1. For each image in the database, extract a number of different features, denoted  $nbFeatures$ .
2. Organize these features in an array of  $nbFeatures$  lines and 216 columns. In this way, each column  $i$  represents the different features of the image  $i$ .



You can use the Matlab function `regionprops` to extract some features such as area, eccentricity, perimeter, solidity...

## 37.2 Image classification

In order to classify the images, we are going to use neural networks. Pattern recognition networks are feedforward networks that can be trained to classify inputs according to target classes. The inputs are the features of each image. The target data are the labels, indicating the class of each image.



1. Build the target data, representing the class of each image. The target data for pattern recognition networks should consist of vectors of all zero values except for a 1 in element  $i$ , where  $i$  is the class they are to represent. In our example, the target data will be an array of 18 lines and 216 columns.
2. The database will be divided into a training set (75%) and a test set (25%).
3. Run the training task and classify the test images.
4. Show the classification confusion matrix as well as the overall performance.
5. Try to run the same process. What can you conclude?
6. Try to change the parameters of the network to improve the classification performance.



Look at the Matlab function `patternnet` to make the classification. Look at the field `divideParam` of your pattern network for performing the division.



### 37.3. Matlab correction



#### 37.3.1 Feature extraction

We make a loop on the whole database to extract some features of each image. The 9 features used here are: area, convex area, eccentricity, equivalent diameter, extent, major axis length, minor axis length, perimeter and solidity. All these parameters are defined in the documentation of the Matlab function `regionprops`.



```
% 18 classes of 12 images
1 folderImages = './images_Kimia216/';
2 classes = {'bird', 'bone', 'brick', 'camel', 'car', 'children', ...
3     'classic', 'elephant', 'face', 'fork', 'fountain', ...
4     'glass', 'hammer', 'heart', 'key', 'misk', 'ray', 'turtle'};
5 nbClasses = length(classes);
6 nbImages = 12;
7
8 features = [];
9 targets = zeros(nbClasses,nbClasses*nbImages);
10
11 for i=1:nbClasses
12     for k=1:nbImages
13         nameImage = strcat(folderImages, classes{i}, '-', num2str(k), '.bmp');
14         currentImage = imread(nameImage);
15         currentImage = currentImage==0;
16         s = regionprops(currentImage, 'Area', 'ConvexArea', ...
17             'Eccentricity', 'EquivDiameter', 'Extent', ...
18             'MajorAxisLength', 'MinorAxisLength', ...
19             'Perimeter', 'Solidity');
20         sArray = [s.Area;s.ConvexArea;s.Eccentricity, ...
21             s.EquivDiameter;s.Extent, ...
22             s.MajorAxisLength;s.MinorAxisLength, ...
23             s.Perimeter;s.Solidity];
24         features = [features, sArray (:,1)];
25     end
26     targets (i ,(i-1)*nbImages+[1:nbImages]) = 1;
27 end
```

Note that in the same time, the target array (required in the following) is built within this loop. It represents the true classes of the objects.

#### 37.3.2 Classification

The network is created with 10 hidden layers. We used a training set of 75% of the database and 25% for the test set.



```
% create the network
2 hiddenLayerSize = 10;
net = patternnet(hiddenLayerSize);
4
% set up the division of data
6 net.divideParam.trainRatio = 75/100;
%net.divideParam.valRatio = 20/100;
8 net.divideParam.testRatio = 25/100;
```

Now the network is trained and tested:



```
% train the network
2 [net, tr] = train(net, features, targets);

4 % test the network
outputs = net(features);
```

The overall performance as well as the confusion matrix is here computed:



```
1 % overall performance
[ c, cm, ind, per ] = confusion( targets, outputs );
3 perf = 1 - c

5 % confusion matrix
figure; plotconfusion( targets, outputs );
```

and the result is:

### Command window

```
perf = 0.9444
```

Note that if we run the same code, the result can be different since the initialization of the optimization process (used for the training task) is different. For example, by running again, we get the following result:

### Command window

```
1 perf = 0.9167
```

Confusion Matrix																			
Output Class	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	
	12 5.6%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	1 0.5%	92.3% 7.7%
	0 0.0%	12 5.6%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	100% 0.0%
	0 0.0%	0 0.0%	11 5.1%	0 0.0%	0 0.0%	1 0.5%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	91.7% 8.3%
	0 0.0%	0 0.0%	0 0.0%	10 4.6%	0 0.0%	0 0.0%	2 0.9%	0 0.0%	0 0.0%	1 0.5%	0 0.0%	1 0.5%	71.4% 28.6%						
	0 0.0%	0 0.0%	1 0.5%	0 5.6%	12 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	92.3% 7.7%
	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 5.6%	12 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	100% 0.0%
	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	10 4.6%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	100% 0.0%
	0 0.0%	0 0.0%	2 0.9%	0 0.0%	0 0.0%	0 0.0%	10 4.6%	0 0.0%	33.3% 16.7%										
	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	12 5.6%	0 0.0%	92.3% 7.7%									
	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	12 5.6%	0 0.0%	100% 0.0%								
	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	12 5.1%	0 0.0%	100% 0.0%							
	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	12 5.1%	0 0.0%	100% 0.0%						
	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	11 5.6%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	92.3% 7.7%
	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	11 5.6%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	100% 0.0%
	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	11 5.1%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	100% 0.0%
	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	11 5.1%	0 0.0%	0 0.0%	0 0.0%	91.7% 8.3%
	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	11 5.1%	0 0.0%	100% 0.0%
	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	11 5.1%	0 0.0%
	100% 0.0%	100% 8.3%	91.7% 16.7%	33.3% 0.0%	100% 16.7%	100% 16.7%	33.3% 0.0%	100% 0.0%	94.4% 5.6%										

Figure 37.2: Confusion matrix of the classification result.

The aim of this tutorial is to develop a simple Harris corner detector. This is the first step in pattern matching, generally followed by a feature descriptor construction, and a matching process.

## 38.1

# Corner detector and cornerness measure



Use `imgradientxy` and `imgaussfilt` with a scale parameter  $\sigma$  that will constrain the size of the window  $W$ .

### 38.1.1 Gradient evaluation

The Harris corner detector is based on the gradients of the image,  $I_x$  and  $I_y$  in x and y directions, respectively.



Apply a Sobel gradient in both directions in order to compute  $I_x$  and  $I_y$ .

### 38.1.2 Structure tensor

The structure tensor is defined by the following matrix. The coefficients  $\omega$  follow a gaussian law, and each summation represents a gaussian filtering process.  $W$  is an operating window.

$$M = \begin{bmatrix} \sum_{(u,v) \in W} \omega(u,v) I_x(u,v)^2 & \sum_{(u,v) \in W} \omega(u,v) I_x(u,v) I_y(u,v) \\ \sum_{(u,v) \in W} \omega(u,v) I_x(u,v) I_y(u,v) & \sum_{(u,v) \in W} \omega(u,v) I_y(u,v)^2 \end{bmatrix} = \begin{bmatrix} M_1 & M_2 \\ M_3 & M_4 \end{bmatrix}$$



- Evaluate  $M_1$  to  $M_4$  for each pixel of the image.

### 38.1.3 Cornerness measure

The cornerness measure  $C$ , as proposed by Harris and Stephens, is defined as follows for every pixel of coordinates  $(x, y)$ :

$$C(x, y) = \det(M) - K\text{trace}(M)^2$$

with  $K$  between 0.04 and 0.15.



Compute  $C$  for all pixels and display it for several scales  $\sigma$ .

## 38.2

### Corners detection

A so-called Harris corner is the result of keeping only local maxima above a certain threshold value. You can use the checkerboard image for testing, or load the sweden road sign image Fig.38.1.



```
I = imread('sweden_road.png');
```

Figure 38.1: Sweden road sign to be used for corner detection.



- Evaluate the extended maxima of the image.
- Only the strongest values of the cornerness measure should be kept.  
Two strategies can be employed in conjunction:

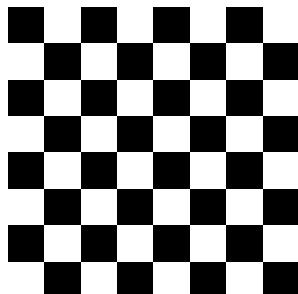
- Use a threshold value  $t$  on  $C$ : the choice of this value is not trivial, and it strongly depends on the considered image. An adaptive method would be preferred.
  - Keep only the  $n$  strongest values.
- The previous operations are affected by the borders of the image. Thus, eliminate the corner points near the borders.
- The detected corners may contain several pixels. Keep only the centroid of each cluster.



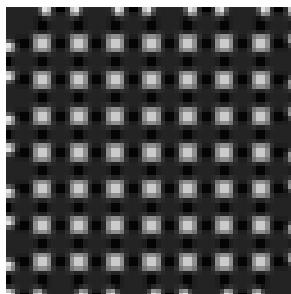
### 38.3. Matlab correction



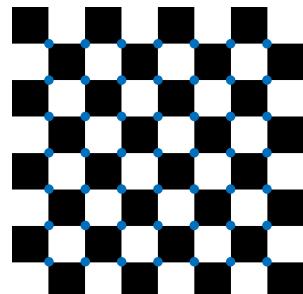
#### 38.3.1 Cornerness measure



(a) Checkerboard.



(b) Cornerness measure.



(c) Harris corner points.

Figure 38.2: Cornerness measure for the checkerboard image. The next step is to locate the corners by thresholding the measure, extracting the local maxima, eliminating points near the edges...

The first step is to compute the gradient in both x and y directions.



```

1 % first of all , compute gradient in x and y directions
hx = [-1 0 1; -2 0 2; -1 0 1];
3 Ix = conv2(I, hx, 'same');
Iy = conv2(I, hx', 'same');
```

Or more simply:



```
[Ix, Iy] = imgradientxy(I);
```

Then, the coefficients of the matrix are computed.



```

1 % compute the different coefficients of the Harris detector
M1 = Ix.^2;
3 M2 = Ix.*Iy;
M4 = Iy.^2;
```

In case of using a scale parameter, these coefficients should be filtered (for example via a gaussian filter).



```
% apply scale
2 M1 = imgaussfilt (M1, sigma);
M2 = imgaussfilt (M2, sigma);
4 M4 = imgaussfilt (M4, sigma);
```

Finally, the cornerness measure is evaluated.



```
% cornerness measure evaluation and display
2 C = (M1.*M4 - M2.^2) - K *(M1+M4).^2;
```

The cornerness measure is displayed in Fig.38.2.

### 38.3.2 Corners detection

In order to keep only the strongest corner points, a threshold value  $t$  is applied on  $C$ . This value is really depending on the considered image, thus such a global threshold is not generally a good idea. One would probably prefer a h-maximum or equivalent operator. For the purpose of this tutorial, we will keep this strategy.



```
C(C<t)=0;
```

The local maxima are then extracted.



```
1 corners = imextendedmax(C2, 3);
```

In the case of the checkerboard image, corner points near the edges are detected. These may be removed.



```
1 corners = imclearborder(corners, 8);
```

The result is a binary image, where some clusters of points are the corner points. To keep only one point per cluster, the centroid of each is detected. The final result is presented in Fig.38.2c.



```
1 stats = regionprops(corners, 'Centroid');  
3 if ~isempty(stats)  
    pts = cat(1, stats.Centroid);  
5 else  
    pts = [];  
7 end
```

### 38.3.3 Road sign image application

In this case, the values  $t = 10^7$  and  $\sigma = 3$  are used. The result is illustrated in Fig.38.3.



(a) Cornerness measure.



(b) Corner points.

Figure 38.3: Harris corner detection with scale  $\sigma = 3$  and threshold value  $t = 10^7$ .

This tutorial aims to study a texture descriptor named 'Local Binary Patterns'. The first objective is to implement this descriptor. Thereafter, digital images of textures will be classified using this descriptor and the k-means algorithm.

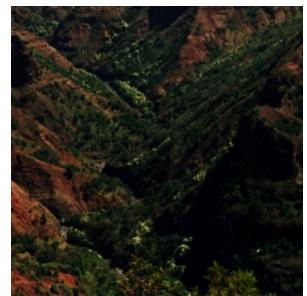
The different processes will be applied on this kind of texture images:



(a) metal image



(b) sand image



(c) ground image

## 39.1 Local Binary Patterns

The Local Binary Patterns (LBP) descriptor is a simple yet very efficient texture operator which labels the pixels of an image by thresholding the neighborhood of each pixel and considers the result as a binary number. Due to its discriminative power and computational simplicity, LBP texture operator has become a popular approach in various applications. It can be seen as a unifying approach to the traditionally divergent statistical and structural models of texture analysis. Perhaps the most important property of the LBP operator in real-world applications is its robustness to monotonic gray-scale changes caused, for example, by illumination variations. Another important property is its computational simplicity, which makes it possible to analyze images in challenging real-time settings.

The LBP feature vector, in its simplest form, is created in the following manner:

- For each pixel, compare the pixel to each of its 8 neighbors (on its left-top, left-middle, left-bottom, right-top, etc.). Follow the pixels along a circle, i.e. clockwise or counter-clockwise.
- Where the center pixel's value is greater than the neighbor's value, write "1". Otherwise, write "0". This gives an 8-digit binary number (which is usually converted to decimal for convenience).

- Compute the histogram of the frequency of each "number" occurring (i.e., each combination of which pixels are smaller and which are greater than the center).
- Normalize the histogram.

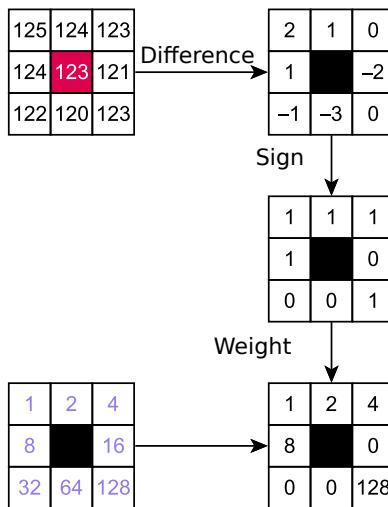


Figure 39.1: Local binary pattern. From wikipedia, author Xiawi, CC-By-SA.



- Code a function for computing the Local Binary Patterns.
- Test this operator on a texture image from the given database.



Consider the function `hiscounts` for histogram computation.

## 39.2 Classification of texture images

The objective is to classify the texture images from the given database by using the LBP descriptor.



1. Calculate the LBP descriptor for each image of the database.
2. Compare the descriptors for each class of images.

3. Compute the distance between each pair of images in order to get a dissimilarity matrix. Comment the result.
4. Use the k-means algorithm to classify the images of the database into three classes ( $k = 3$ ).



See `kmeans`.



### 39.3. Matlab correction



#### 39.3.1 LBP computation

Each pixel is given a specific 8 bits value according to a code as follows. The parfor loop is used for speed.



```
% binary code for pixel description
2 code = [1 2 4; 8 0 16; 32 64 128];

4 % loop over all pixels
parfor i=2:m-1
6   for j=2:n-1
8     w = A(i-1:i+1,j-1:j+1);
      w = (w >= A(i,j));
      w = w.*code;
10    B(i,j) = sum(w(:));
      end
12 end
```

Then, all values (except for border values) are summarized in the histogram.



```
B = B(2:end-1,2:end-1);
2 h = histcounts(B(:, ), 256, 'normalization', 'probability');
```

For the first image of sand, the histogram is shown in Fig.39.2.

#### 39.3.2 Classification

For all images of the same family, the LBP are computed and represented in the same graph. The histograms really look similar (see Fig.39.3). The following code is used for the “sand” family.

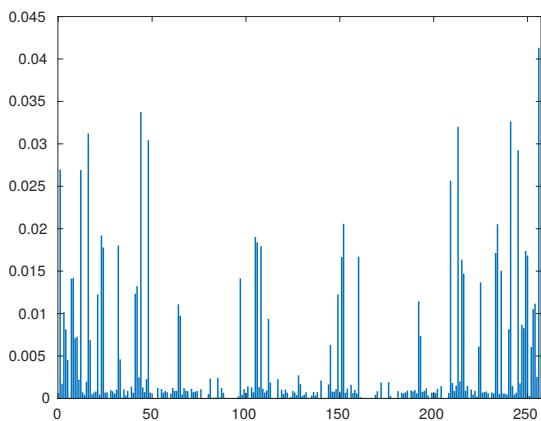


```
rep='images/';
2 name_image = 'Sand';
LBP_Sand = cell (1,4);

4 for i = 1:4
6   A = imread(strcat(rep, name_image, '.', num2str(i), '.bmp'));
     A = rgb2gray(A);
8   LBP_Sand{i} = LBP(A);
```



(a) Texture image.



(b) LBP of texture.

Figure 39.2: Illustration of the Local Binary Pattern computed on an entire image.



```

end
10
figure ;
12 hold on;
corange = {[1 0 0],[1 0.2 0], [1 0.4 0], [1 0.6 0]};
14 for i=1:4
    plot(LBP_Sand{i}, 'color ',corange{i}) ;
16 end

```

A distance criterion is used to compare the different histograms: the classical SAD (Sum of Absolute Differences) gives a numerical values. All pairs of distances are concatenated in a matrix, displayed as an image in Fig.39.4.



```

LBP = [LBP_Terrain,LBP_Metal,LBP_Sand];
2 dist = zeros (12,12) ;
parfor i=1:2
4     X(i ,:) = LBP{i}'; % for later K-means computation
        for j=1:12
6         dist (i ,j) = sum(abs(LBP{i}-LBP{j})) ;
        end
8 end
10 figure ;
11 imagesc(dist );
12 colormap('gray');

```

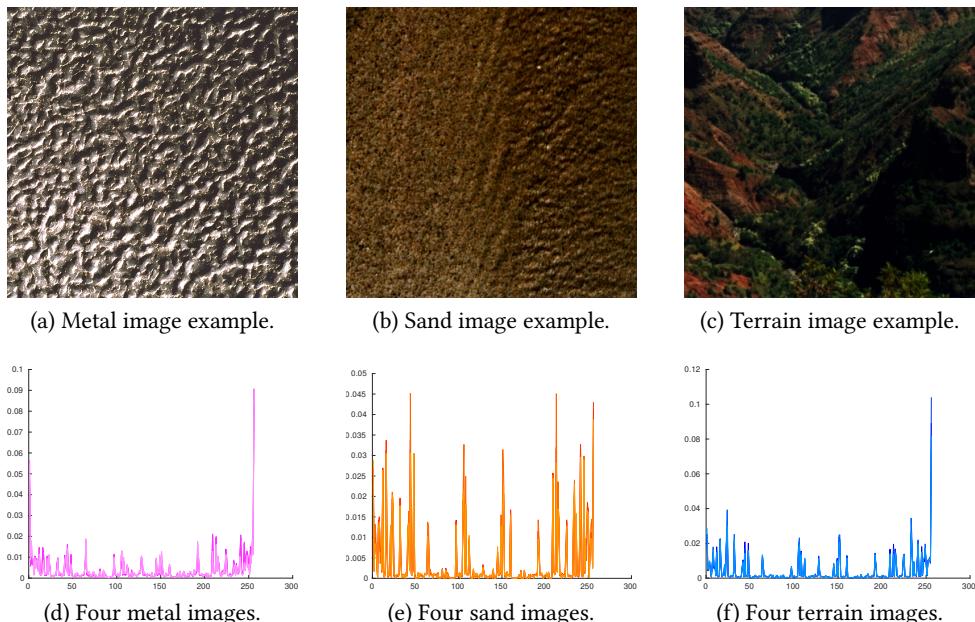


Figure 39.3: Illustration of the LBP of 4 images of each family. The histograms are almost equivalent, which shows that this descriptor can be employed to discriminate between the different families.

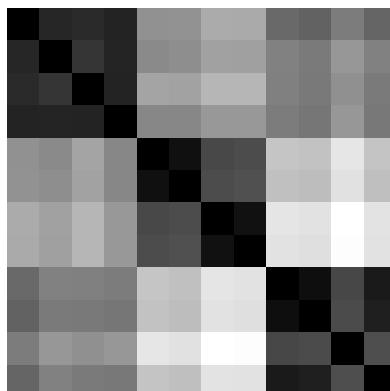


Figure 39.4: Sum of Absolute Differences between the different LBP histograms of each image. 3 families of 4 textures are represented here, terrain images are in the first part, metal images in the second and sand images in the last. Black represents 0 distance and white is 1 (highest distance, the values are normalized).

The kmeans algorithm uses such a distance, and we can verify that the clustering process works as expected. The result is presented in the next box.



```
% labels 1–4, 5–8, 9–12 should be equal, respectively  
2 label = kmeans(X,3,'replicates ',5)
```

### Command window

```
% the 3 families are well classified  
2 label = 3 3 3 3 2 2 2 1 1 1
```



## **Part VI Exams**



All printed documents allowed. Send your code via the campus website (zip files if several files are to be sent).

## 40.1

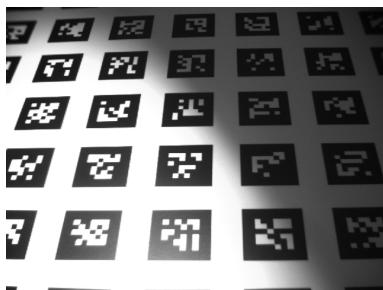
### Segmentation (13 points)



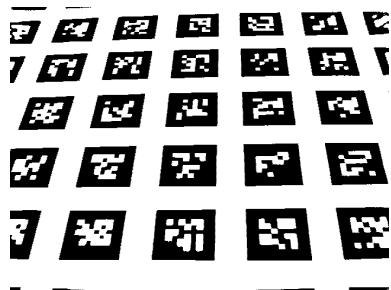
- Segment the image.

Figure 40.1: Test image

(a) Image to be segmented, from [?].



(b) Segmented image



## 40.2

### Integral image (7 points)

If the original image is denoted  $f$ , the integral image  $I$  is defined by the following equation, for all pixels of coordinates  $(x, y)$ :

$$I(x, y) = f(x, y) + I(x - 1, y) + I(x, y - 1) - I(x - 1, y - 1)$$



- Code a function that computes this integral image. The prototype of this function must be `function II=int_image(I)`.

- Code a function that computes the local average of the image  $f$ . Notice that:

$$\sum_{x=x_1}^{x_2} \sum_{y=y_1}^{y_2} f(x, y) = I(x_2, y_2) - I(x_2, y_1 - 1) \\ - I(x_1 - 1, y_2) + I(x_1 - 1, y_1 - 1)$$

- Compare it to a mean filter.



The mean filter can be coded with the MATLAB<sup>®</sup> functions `imfilter` and `fspecial` (result and computation time).

## 41.1

# Basic notions

### 41.1.1 Image characterisation



- Cite the names of the major image file formats and their main differences. What is DICOM?
- Define sampling in numerical images. Define the image resolution.

### 41.1.2 Sensors



- What is a Bayer filter?
- Explain the principles of demosaicing.

### 41.1.3 Image processing



- Define the operation of histogram equalization.
- What is the difference with histogram stretching?



- From the mathematical definition of the derivative, explain the construction of the gradient operator, and then of the Laplacian.
- Cite some method for contours detection, and list their pros and cons.

## 41.2 Open question



The Fig. 41.1 shows a human retina (eye fundus). Propose a method for segmenting:

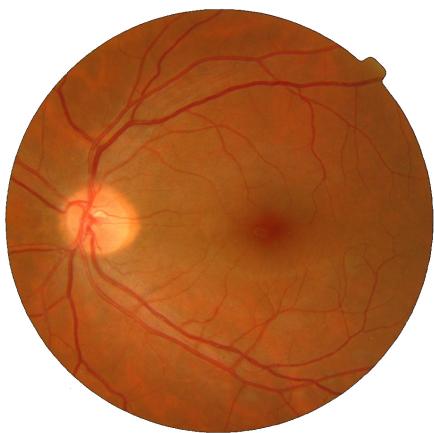
- the vessels,
- the optical nerve (bright disk).

Justify your choices. Indicate all elements that seem important and the way to deal with them. Look carefully at the shapes and intensities of objects you need to segment.

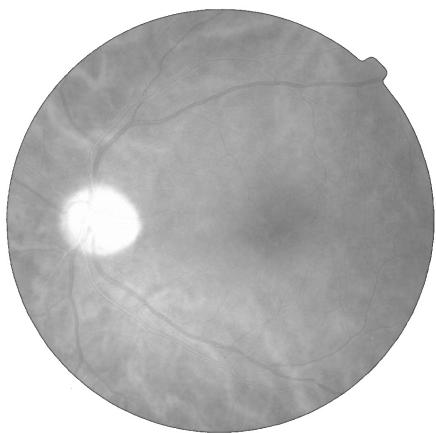
For your record, ophthalmologists need to measure the diameter and the circularity of the optical nerve, as well as the length and the number of vessels (branches, tortuosity...).

Figure 41.1: Human retina.

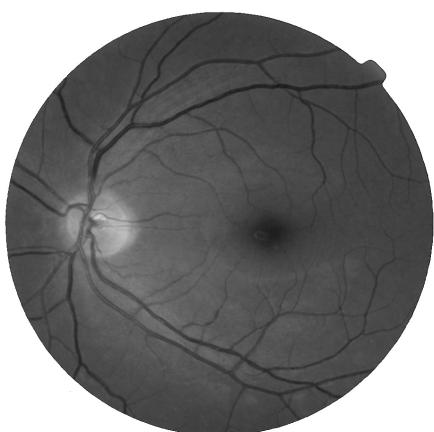
(a) Color image.



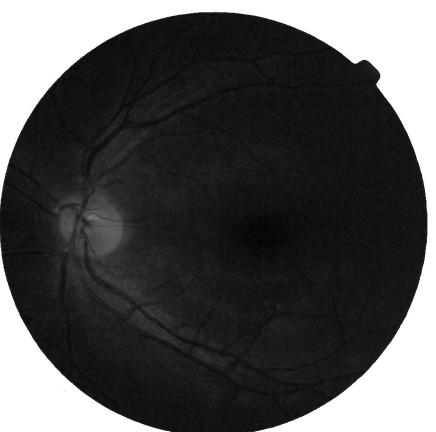
(b) Red channel.



(c) Green channel.



(d) Blue channel.





## 42.1 Warm-up questions

### 42.1.1 Thresholding methods



Cite different methods in order to find a threshold value to binarize a grayscale image. Explain the Otsu's method. What are its limits?

### 42.1.2 Image filtering



- What is the difference between a rank filter and a linear filter?
- Cite the name of a filter of each type.
- In case of a salt and pepper noise present in the image, what type of filter would you require to restore it? Why?
- For the latter, precise its limits and propose a way to improve the result.

### 42.1.3 Retina images



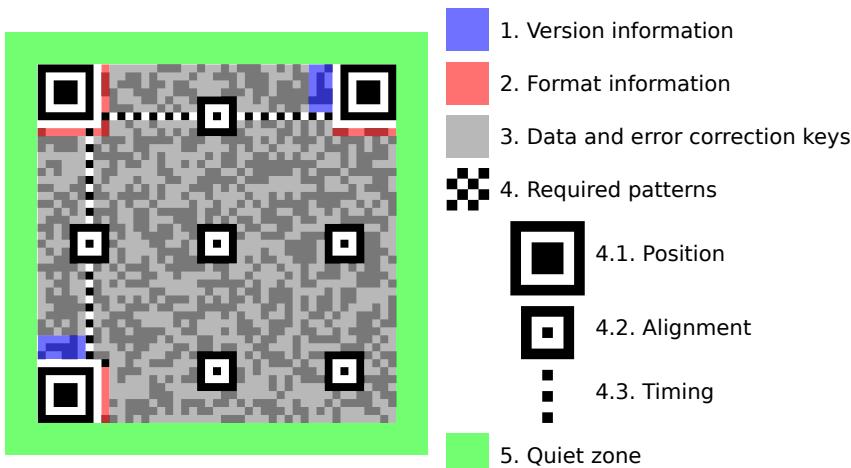
As a project, you coded a method for retina vessels segmentation. Explain the principles of the algorithm.

**42.2****Open question: QR code**

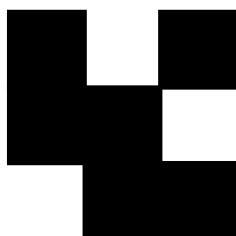
A QR code is a binary code represented as a 2D matrix. A scanner is in charge of reading it (via a camera) and translating it into the actual code. The structure of a QR code is represented in Fig.42.1.

Figure 42.1: Conversion of a binary pattern into a binary matrix.

(a) Structure of a QR code, Wikipedia, author: Bobmath, CC-By-SA



(b) Code.



(c) Matrix.

1	0	1
1	1	0
0	1	1



Describe the different steps required to perform the acquisition and geometric transformation of the QRcode into a binary 2D matrix representing the code (1 value for 1 square, see Fig.42.1).

- List different situations that may complicate the image acquisition and analysis.

- Propose some (image processing) solutions in order to deal with these situations. Explain them, give the necessary details describing the methods.

Figure 42.2: Different QR codes. The image acquisition and analysis must be tolerant to different observation conditions... Images from Collectif Raspouteam <http://raspouteam.org/>. Other images from unknown sources.



# Index

- Aliasing, 14, 21
- Characterization, 349, 358, 367, 375
  - Circularity, 369
  - Convexity, 369
  - Diameter, 368
  - Perimeter, 367
- Classification, 376, 401
- Color
  - Chromaticity diagram, 127
  - Color Matching Functions, 127
  - Color Spaces, 125
- Computational Geometry
  - Convex Hull, 315, 369
  - Delaunay Triangulation, 323
  - Minimum Spanning Tree, 323
  - Voronoi Diagram, 323
- Features, 375
  - Detection, 384
  - Harris Corner Detector, 393
  - Local Binary Patterns, 399
- File formats, 18
- Filtering
  - Choquet, 136
  - Convolution, 14
  - Degenerate Diffusion, 150
  - High-pass, 15, 27
  - Linear Diffusion, 148
  - Low-pass, 14, 24
  - Non Linear Diffusion, 148
  - Partial Differential Equations, 147
- Fourier Transform, 59, 89
  - 2D FFT, 48
  - Application, 49
  - Filtering, 48
- Inverse, 48, 91
- Frameworks
  - Color Logarithmic Image Processing, 125
  - General Adaptive Neighborhood Image Processing, 135
  - Logarithmic Image Processing, 117
  - Freeman Chain Code, 357
- Geometry
  - Crofton Perimeter, 338
  - Integral Geometry, 335
- Gradient
  - Prewitt, 15
  - Sobel, 15
- Heat Equation, 148
- Histogram, 12, 19
  - Definition, 37
  - Equalization, 37
  - Matching, 38
- Image
  - Display, 12
  - Load, 12
- Laplacian, 15, 27
- Mathematical Morphology, 261, 301
  - Alternate Sequential Filters, 274
  - Attribute Filters, 279
  - Dilation, 262
  - Erosion, 262
- General Adaptive Neighborhood, 138
- Geodesic Filtering, 273
- Granulometry, 307
- Hit-or-miss Transform, 285

- Reconstruction, 262, 274
- Skeleton, 285
- Topological Skeleton, 285
- Watershed, 293
- Multiscale, 157
  - Filtering, 159
  - Kramer & Bruckner, 159
  - Pyramid, 157
- Noise
  - Exponential, 75
  - Gaussian, 75
  - Salt & Pepper, 75
  - Uniform, 75
- Point Spread Function, 89
- Quantization, 13, 21
- Random Fields, 233
- Registration, 381
- Restoration
  - Adaptive median filter, 77
  - Blind deconvolution, 94
  - Deconvolution, 89
  - Lucy-Richardson filter, 94
  - Median filter, 76
  - Van Cittert filter, 93
  - Wiener filter, 92
- Segmentation, 301
  - Active Contours, 199
  - Distance Map, 293
  - Histogram, 175
  - Hough Transform, 191
  - K-means, 176, 351, 401
  - Otsu thresholding, 176
  - Region Growing, 185
  - Threshold, 175
  - Watershed, 293
  - Watershed by Markers, 294
- Shape From Focus, 107
  - SML, 109
  - Tenengrade, 110
  - Variance, 109
- Variance of Tenengrad, 110
- Spatial Processes, 209
  - Boolean Models, 225
  - Characterization, 212, 225
  - Gibbs Point Process, 211
  - Marked Process, 213
  - Miles Formula, 227
  - Neyman-Scott Point Process, 210
  - Poisson Point Process, 209
  - Ripley Functions, 212
- Stereology, 243
  - Random Chords, 245
  - Random Planes, 247
- Tomography, 165
  - Backprojection, 166
  - Filtered Backprojection, 167
- Topology
  - Classification, 343
  - Neighborhood, 335, 341, 357
- Wavelets
  - Definition, 59

**Image processing tutorials with python** This book is a collection of tutorials and exercises given at MINES Saint-Etienne as part of the Master's Degree in Science and Executive Engineering ("Ingénieur Civil des Mines – ICM"). In recent years, project-based learning has been used to illustrate theoretical concepts with real and concrete applications.

Whether you are in the early years of your university studies, in preparatory classes for the French Grandes Ecoles or in an engineering school, or even as a teacher, this book is made for you. You will find a large number of tutorials, classified by field, to familiarize yourself with the theoretical concepts of image processing and analysis.

Go to <http://iptutorials.science> to download the complete codes in python.

**Yann GAVET** He graduated from Mines Saint-Etienne with a Master's Degree in Science and Executive Engineering ("Ingénieur Civil des Mines - ICM") in 2001, obtained a Master of Science in 2004 and defended his PhD thesis in 2008. He teaches signal-processing, image-processing and pattern-recognition as well as C programming at Master's level.

**Johan DEBAYLE** He received his master of science and PhD thesis in 2002 and 2005. He is the head of the master MISPA of MINES Saint-Étienne, and teaches signal-processing, image-processing and pattern-recognition at Master's level.



Une école de l'IMT