



IMA 4509

Visual content analysis

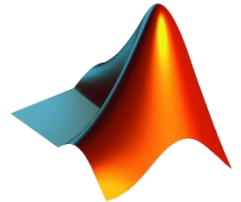
Image processing & analysis with MATLAB: an overview

Nicolas ROUGON
ARTEMIS Department



Motivations

■ MATLAB: an industry standard for rapidly testing new ideas & prototyping applications



- Interpreted programming language ▶ (rather) slow
- Easy-to-program: untyped C-like syntax operating on matrices
- Powerful API thanks to dedicated **toolboxes**
- C/C++ code generation (C/MEX files)
 - > compiled MATLAB routines ▶ faster execution
 - > standalone applications
- Extensive online resources: documentation, tutorials, examples, source code repository ([MATLAB CENTRAL](#))
- Huge literature:
 - > MATLAB references
 - > Scientific books using MATLAB as simulation language

Motivations

■ The **Image Processing Toolbox** for MATLAB

- Image display & exploration
- Spatial transformations & image registration
- Color space conversion
- Image statistics & arithmetic
- Basic image analysis
- Image enhancement & restoration
- Image filtering & transforms
- Mathematical morphology
- GUI tools
- Online documentation:

www.mathworks.fr/help/toolbox/images

Local MATLAB versions

■ 2 releases available on TSP **Unix** servers

■ MATLAB 2010a

- > main routine
(default)

/opt/matlab-disi-R2010a

/usr/local/bin/matlab

-> **/opt/matlab-disi-R2010a/bin/matlab**

■ MATLAB 2015a

- > main routine
- > **to be used in this course**

/opt/matlab-disi-R2015a

/opt/matlab-disi-R2015a/bin/matlab

(re)Initializing MATLAB

■ Clear display:

close all

> close all

// close all figure windows

■ Clear workspace:

clear

> clear

// clear all variables

> clear all

// clear all variables, functions, CMEX files...

> clear A1 A2 A3

// clear designated variables

Image IOs

■ Loading an image: `imread()`

```
> I = imread('the_image.jpg')           // filename
> I = imread('http://the_server.fr/the_image.jpg') // URL
```

- Supported image formats: BMP, GIF, JPEG, JPEG-2000, PBM/PGM/PPM, PNG, TIFF...

■ Writing an image: `imwrite()`

```
> imwrite(I, 'the_image.jpg',...)      // get format from extension
> imwrite(I, 'the_image', format,...)  // specify format
```

- Additional arguments: format-specific parameters

Getting image information

■ From file: `imfinfo()`

```
> imfinfo('the_image.jpg')
```

- Returns: file name image dimensions
 format # of bits per pixel
 modification date image type
 size (in bytes) (truecolor | grayscale | indexed)

■ From variable: `whos`

```
> whos
```

- Returns: size (in bytes) image dimensions
 storage class

DICOM images

■ Getting metadata from file: `dicominfo()`

```
> info = dicominfo('the_image.dcm')
```

■ Loading an image: `dicomread()`

```
> I = dicomread('the_image.dcm')
```

```
// from a DICOM file
```

```
> I = dicomread(info)
```

```
// from DICOM metadata
```

■ Writing an image: `dicomwrite()`

```
> dicomwrite(I, 'the_image.dcm')
```

```
// save image data only
```

```
> dicomwrite(I, 'the_image.dcm', info)
```

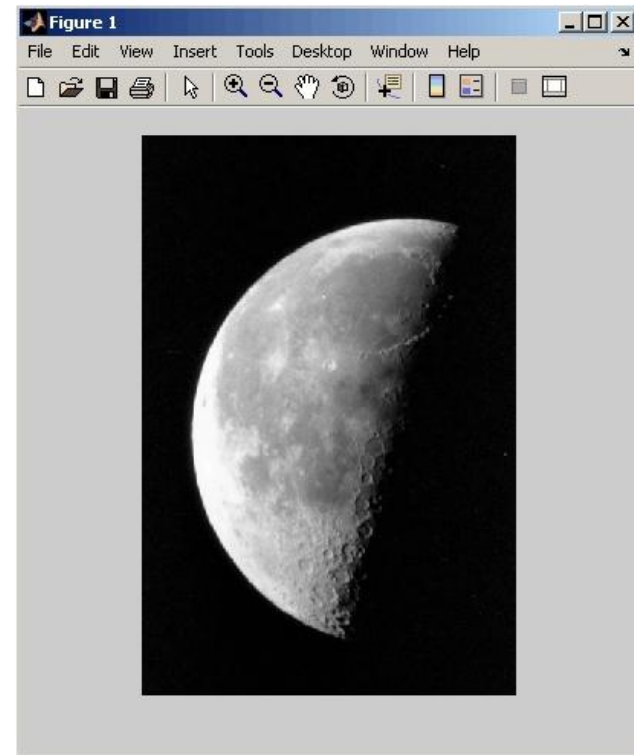
```
// save image & metadata
```

Displaying images

- In a figure window:

```
> I = imread('the_image.jpg');  
imshow(I)
```

`imshow()`



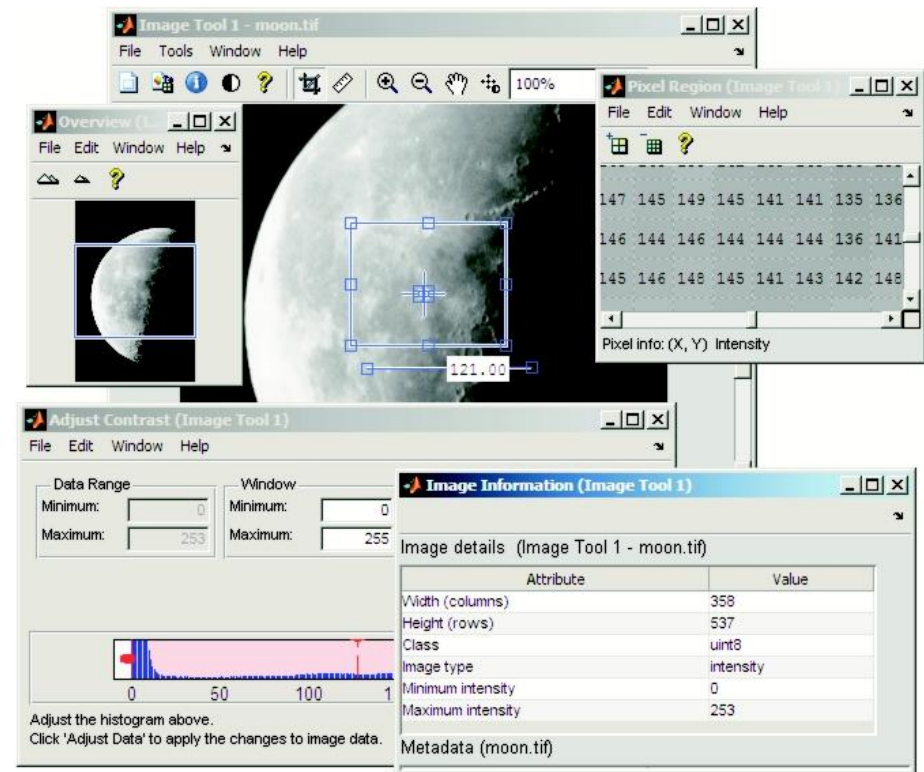
Displaying images

■ An integrated image viewer:

`imshow()`

```
> I = imread('the_image.jpg');  
imshow(I)
```

- Image information
- Current pixel value
- Region pixel values
- Zoom / Pan
- Crop
- Global contrast transform
- ...



Displaying images

■ As a topographic surface: `surf()`

```
> I = imread('the_image.jpg');  
figure, surf(double(I(1:8:end,1:8:end))), zlim([0,255]);  
set(gca, 'ydir', 'reverse');
```

- `surf()` operates on **double** data
- `surf()` uses a reference frame with origin at the upper-left corner and **upward-pointing** y-axis
- `zlim()` sets z-axis limits

Displaying images

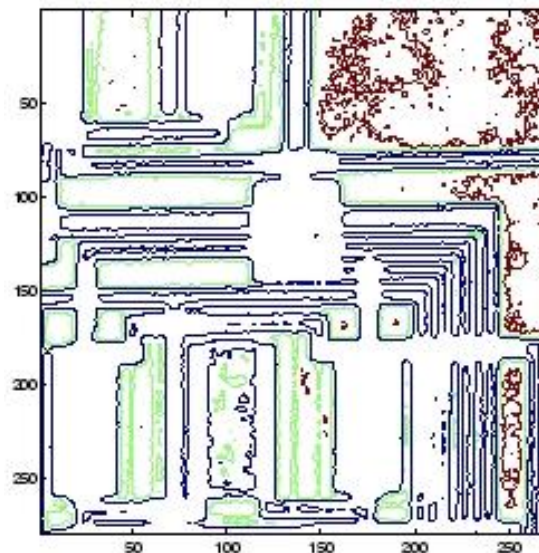
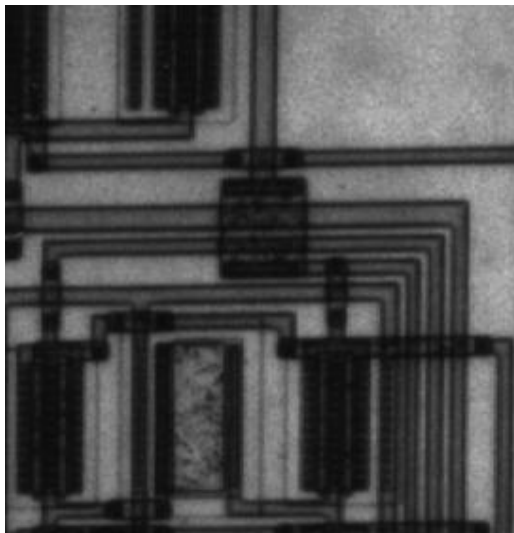
■ Image level lines:

`imcontour()`

> `imcontour(I)`

> `imcontour(I, nb_lines)` // equally spaced values

> `imcontour(I, values)` // specified values



> `I = imread('circuit.tif');`
> `imcontour(I,3)`

Displaying images

■ Line intensity profile:

`improfile()`

> `improfile`

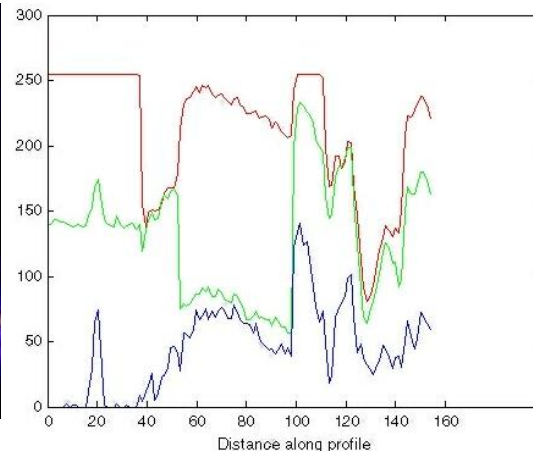
// interactive line definition

> `improfile(I, xi, yi)`

// line definition from end points

■ Arguments

xi, yi: vectors of x,y coordinates of end points
(n lines \blacktriangleright $2n$ -dimensional vectors)



```
> I = imread('peppers.png');  
imshow(I)  
improfile
```

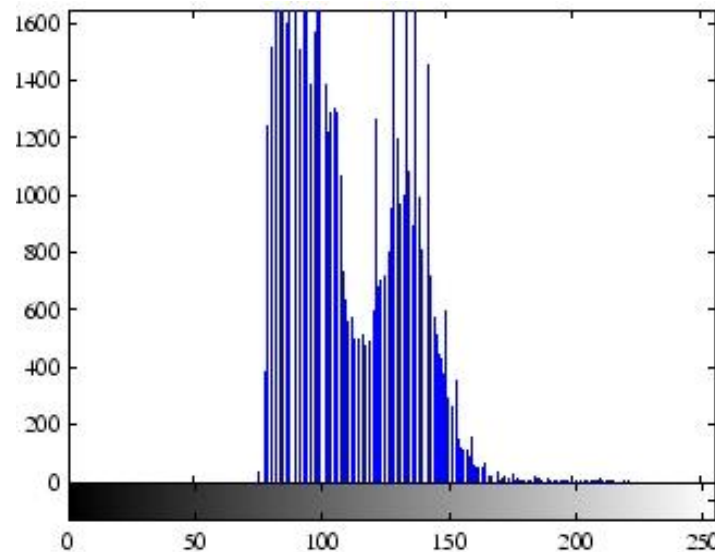
Image histogram

■ Display image histogram: `imhist()`

> `imhist(I)`

> `imhist(I, nb_bins)` // specify # of bins (default: 64)

> `[counts, x] = imhist(I)` // get histogram counts & bin locations



> `I = imread('pout.tif');`
`imshow(I);`
`figure, imhist(I)`

Image histogram

■ (Robust) histogram stretching: `imadjust()`

- > `J = imadjust(I)` // 1% of data saturated at lower/upper bounds
- > `J = imadjust(I, [low_in,high_in],[low_out,high_out])`
// input/output intensity ranges in [0,1]



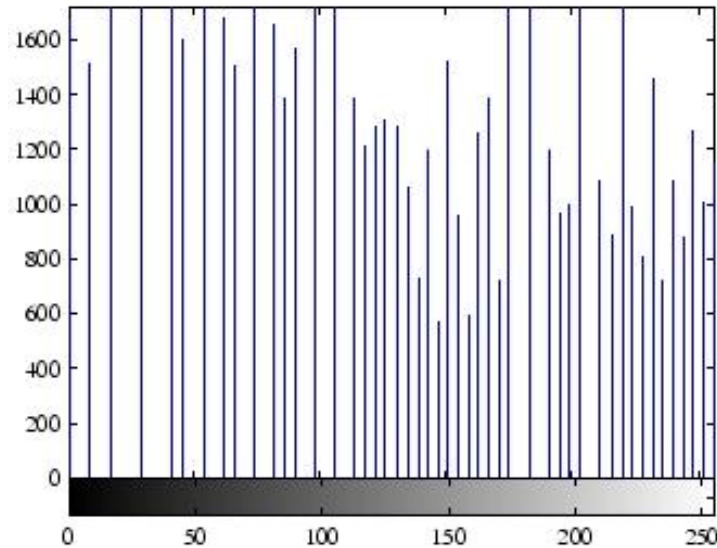
- > `I = imread('pout.tif');`
`imshow(I);`
`J = imadjust(I);`
`figure, imshow(J)`

Histogram transforms

■ Histogram equalization: `histeq()`

```
> J = histeq(I)
```

```
> J = histeq(I, nb_bins) // predefined # of bins
```



```
> I = imread('pout.tif');  
J = histeq(I);  
imshow(J);  
figure, imhist(J);
```

Histogram transforms

- Histogram specification:

`histeq()`

```
> hJ = imhist(J)
```

```
l_J = histeq(I, hJ)
```

```
// specified target histogram
```

- Optional arguments: # of bins

Image threshold

■ Image binarization:

`im2bw()`

> `BW = im2bw(I, level)`

■ Arguments

`I` : grayscale / color image

(color images are first converted to grayscale)

`level` : normalized threshold in $[0,1]$

Histogram-based segmentation

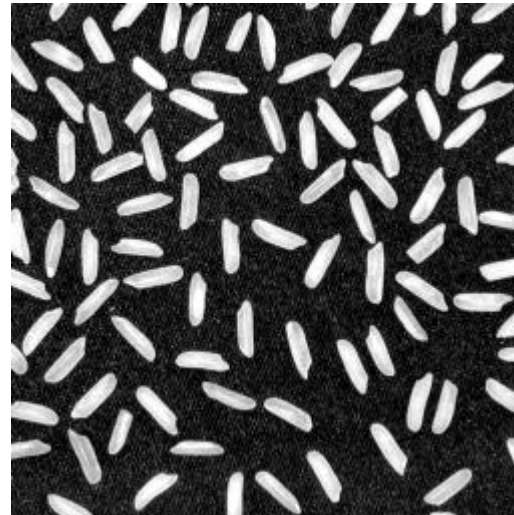
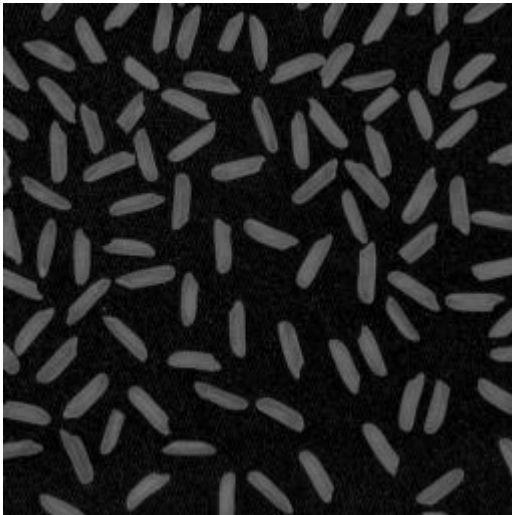
■ Histogram threshold:

```
> level = graythresh(I)
```

```
graythresh()
```

```
// normalized threshold in [0,1]
```

```
// using Otsu's method
```



```
> I = imread('rice.png')
```

```
> J = imadjust(I)
```

```
> level = graythresh(J);  
BW = im2bw(J,level)
```

Image quantization

■ Image quantization:

> $J = \text{imquantize}(I, \text{levels})$

> $J = \text{imquantize}(I, \text{levels}, \text{values})$

■ Arguments

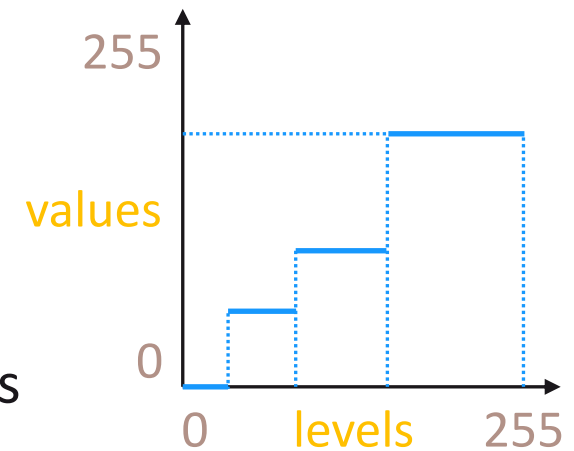
I : grayscale / color image

levels : (1xN)-vector of quantization levels

values : (1xN)-vector of quantization values

default: $[1..N+1]$

imquantize()



Label map visualization

■ Label to RGB map conversion: `label2rgb()`

> `I = label2rgb(L)`

> `I = label2rgb(L, map)`

■ Arguments

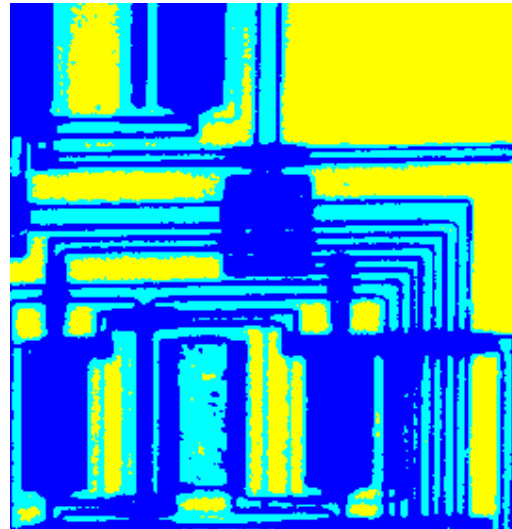
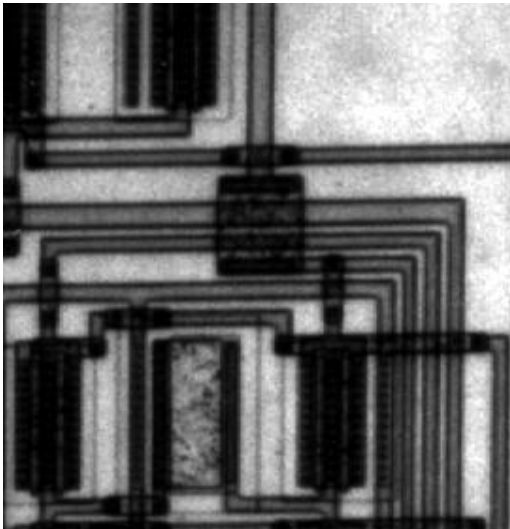
L : label matrix

map : (Nx3) matrix | MATLAB predefined colormap
(see [colormap](#))

Histogram-based segmentation

■ Multilevel histogram threshold: `multithresh()`

```
> levels = multithresh(I, N) // (1xN)-vector of thresholds  
// using multilevel Otsu's method
```



```
> levels = multithresh(I, 2);  
I_seg = imquantize(I, levels);  
RGB = label2rgb(I_seg);  
imshow(RGB);
```

```
> I = imread('circuit.png');  
imshow(I);
```

Image type conversions

- **Image to double:** `im2double()`

> `J = im2double(I)`

- **Image to 8-bit integers:** `im2uint8()`

> `J = im2uint8(I)` // unsigned integers

- **Image to 16-bit integers:** `im2int16()`
`im2uint16()`

> `J = im2int16(I)` // signed integers

> `J = im2uint16(I)` // unsigned integers

Image noise

■ Noisy image synthesis: `imnoise()`

> `J = imnoise(I, type, parameters)`

<code>type</code>	<code>parameters</code>	<code>default</code>	
<code>'gaussian'</code>	<code>mean, variance</code>	<code>0, 0.01</code>	
<code>'localvar'</code>	<code>local_variance</code>		<code>// variance map</code>
<code>'poisson'</code>	<code>-</code>	<code>-</code>	
<code>'salt & pepper'</code>	<code>density</code>	<code>0.05</code>	
<code>'speckle'</code>	<code>variance</code>	<code>0.04</code>	

Image noise

■ Noisy image synthesis:

`imnoise()`



```
> I = imread('eight.tif');  
    imshow(I);
```

```
> J = imnoise(I, 'salt & pepper', 0.02);  
    figure, imshow(J);
```

Image linear filtering

■ Create predefined kernel: `fspecial()`

> `H = fspecial(type)`

> `H = fspecial(type, parameters)` // specify filter parameters

type	parameters	default	
'average'	hsize	[3,3]	
'disk'	radius	5	
'gaussian'	hsize, sigma	[3,3] , 0.5	
'laplacian'	alpha	0.2	// (3x3) Laplacian
'prewitt'	-	-	
'sobel'	-	-	
'log'	hsize, sigma	[5,5] , 0.5	

Image linear filtering

■ Apply linear filter:

`imfilter()`

> `J = imfilter(I, H)`

> `J = imfilter(I, H, bcond)`

// specify boundary conditions

■ Arguments

H : filter kernel

bcond = 'symmetric' | 'replicate' | 'circular' | value (default: 0)

Image linear filtering

■ Apply linear filter:

imfilter()



```
> I = imread('cameraman.tif');  
imshow(I);
```

```
> H = fspecial('disk', 10);  
J = imfilter(I,H, 'symmetric');  
figure, imshow(J);
```

Image nonlinear filtering

■ 2D median filter:

`medfilt2()`

> `J = medfilt2(I)`

> `J = medfilt2(I, [m n])`

// specify neighborhood size

// default : [3 3]

■ 3D median filter:

`medfilt3()`

> `J = medfilt3(I)`

> `J = medfilt3(I, [m n q])`

// specify neighborhood size

// default : [3 3 3]

Image nonlinear filtering

■ 2D median filter:

medfilt2()



```
> I = imread('cameraman.tif');  
J = imnoise(I,'salt & pepper',0.02);  
imshow(J);
```

```
> K = medfilt2(J);  
figure, imshow(K);
```

Image nonlinear filtering

■ 2D order-statistic filter: `ordfilt2()`

> `J = ordfilt2(I, order, domain)`

■ Arguments

domain : numeric / logical binary matrix = neighborhood
e.g. `ones(N)` | `true(N)`

order : index in the sorted neighborhood pixel list
e.g. `domain = true(N)`

`order = 1` > minimum

`order = (NxN+1)/2` > median

`order = NxN` > maximum

Image enhancement

■ Image sharpening:

`imsharpen()`

> `J = imsharpen(I, Name, Value, ...)` // unsharp masking

■ Arguments

Name

default Value

'Radius'

1

// Gaussian kernel std deviation

'Amount'

0.8

// sharpening parameter

standard range [0 2]

Edge detection

■ Differential edge detection:

`edge()`

- > `BW = edge(I, method, thresh,...)` // fixed threshold
- > `BW = edge(I, method,...)` // adaptive threshold
- > `[BW, thresh] = edge(I, method,...)` // get threshold value

■ Arguments

`method` = 'roberts' | 'prewitt' | 'sobel' | 'log' | 'canny'

`thresh` : threshold value / range

■ Optional arguments

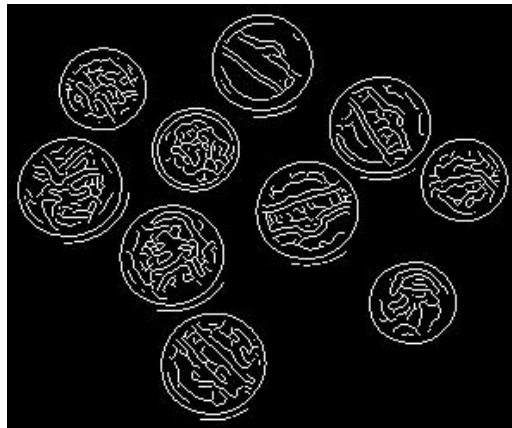
`sigma` : variance of LoG filter / Canny-Deriche parameter

`options` = 'nothinning' | 'thinning' (default)

`direction` = 'horizontal' | 'vertical' | 'both' (default)

Edge detection

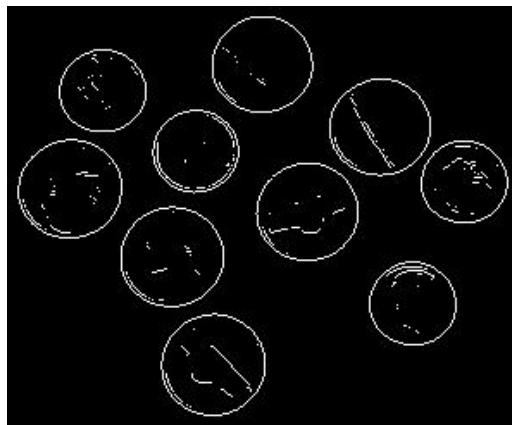
■ Differential edge detection:



edge()

```
> thresh = 40;  
BW = edge(I, 'sobel', thresh);  
figure, imshow(BW)
```

```
> I = imread('coins.png');  
imshow(I)
```



```
> thresh = [120,140];  
BW = edge(I, 'canny', thresh);  
figure, imshow(BW)
```

Corner detection

■ Differential corner detection:

`corner()`

> `C = corner(I)`

> `C = corner(I, method)`

// specify corner detector

> `C = corner(I, N,...)`

// specify maximum # of points

■ Arguments

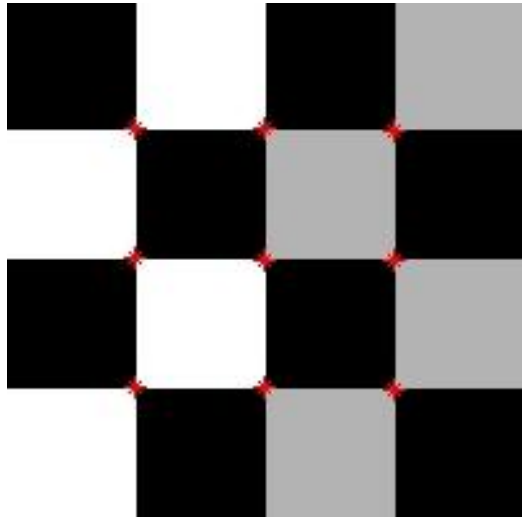
`method` = 'MinimumEigenValue' | 'Harris' (default)

`N` : maximum # of corner points (default: 200)

■ Optional arguments: detector-specific parameters

Corner detection

■ Differential corner detection:



corner()

```
> I = checkboard(50,2,2);  
C = corner(I);           // Harris detector (default)  
imshow(I);  
hold on                 // overwrite on image  
plot(C(:,1), C(:,2), 'r*');
```

Active contours

■ Active contour segmentation:

`activecontour()`

> `BW = activecontour(I, mask)`

> `BW = activecontour(I, mask, n)` // specify max # of iterations

> `BW = activecontour(I, mask, method)` // specify AC method

■ Arguments

`I` : grayscale image

`mask` : binary initialization mask

`n` : maximum # iterations (default: 100)

`method` = 'Chan-Vese' (default) | 'edge' (geodesic AC)

Active contours

■ Active contour segmentation: `activecontour()`

> `BW = activecontour(__, Name, Value)` // specify hyperparameters

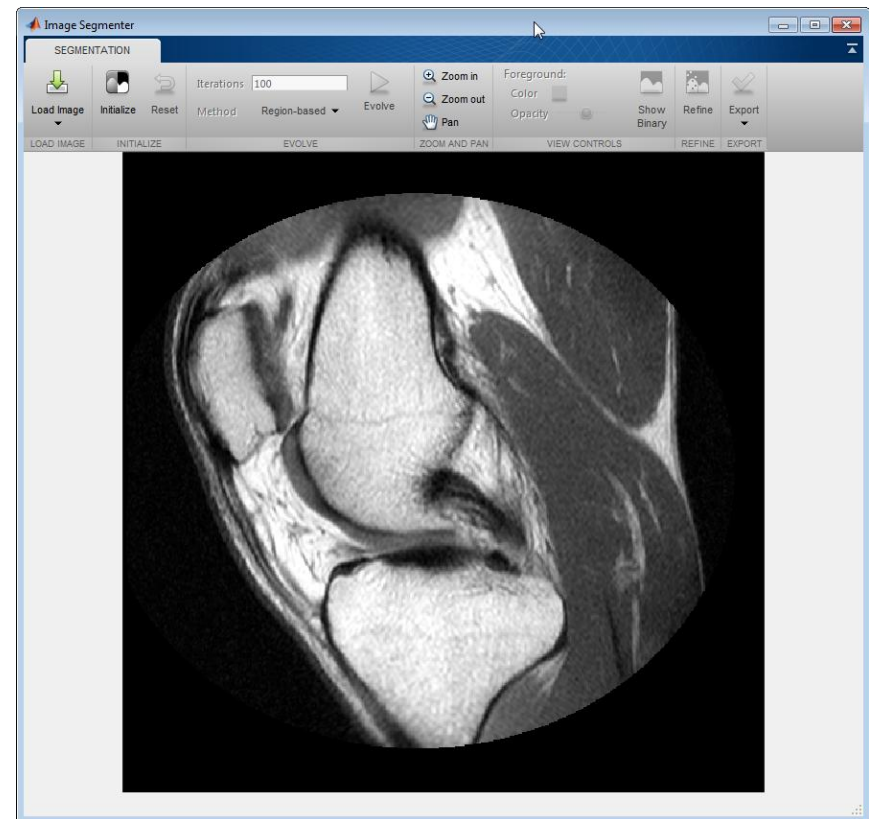
Name	Interpretation	default	
'ContractionBias'	pressure > 0 shrink < 0 expand	0	0.3
'SmoothFactor'	edge map smoothing	0	1.0
		'Chan-Vese'	'edge'

Active contours

■ Integrated AC segmentation tool: imageSegmenter

> ImageSegmenter

- Image IOs
- Initialization (external | interactive | image-based | grid based)
- GUI for `activecontour()`
- Segmentation assessment & post-processing





IMA 4509

Visual content analysis

Image processing & analysis with MATLAB: an overview

Nicolas ROUGON

ARTEMIS Department

