



Università degli Studi di Udine
Corso di Laurea in Tecnologie Web
e Multimediali

Corso di Immagini e Multimedialità
A.S. 2016/2017

Matlab
Prof. Vito Roberto

Relazione Finale

Esercizio 1.2

A partire dall'immagine sintetica della barra, effettuare le operazioni seguenti:

- Aggiungere rumore gaussiano a media nulla, con tre livelli diversi di varianza: 0.01, 0.1, 1. Visualizzare l' immagine sorgente e le tre immagini rumorose. Per la simulazione del rumore usare la funzione MATLAB imnoise()
- Rispetto all'immagine senza rumore presa come riferimento, calcolare l'errore quadratico medio MSE (funzione immse()); il rapporto segnale-rumore di picco (PSNR); il rapporto segnale rumore (SNR) per tutti i livelli di rumore simulato;
- Visualizzare i risultati in forma di tabelle (funzione table())

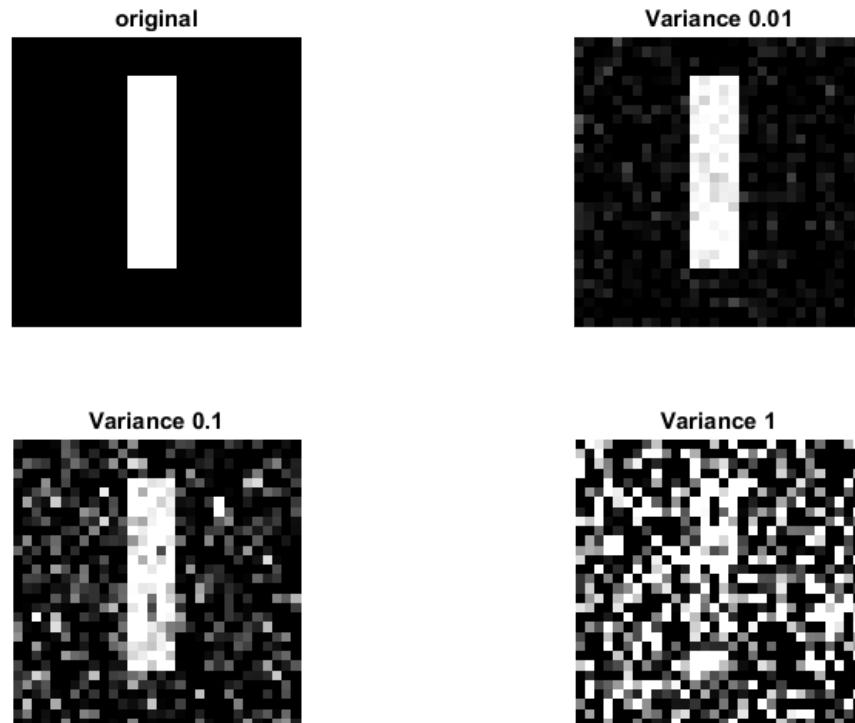
Codice:

- script

```
%%%%%%%%%%%%%
% Script to create artificial noise on an artificial image
% and then calculate Mean Squared Error, Peak Signal to
% Noise Ratio and Signal to Noise Ratio
%
% Silvia Gioia Florio, matr. 119328
% 11/04/2017 - Excercise 1.2
%%%%%%%%%%%%%
%% initialize synthetic image
%% initialize black image
img = zeros(30,30);
%% create a white rectangle
img(5:24, 13:17)=1.;
%
%% adding noise %%
% 0.01 variance noise
imgLowVar = imnoise (img, 'gaussian', 0, 0.01);
% 0.1 variance noise
imgMedVar = imnoise (img, 'gaussian', 0, 0.1);
% 1 variance noise
imgHighVar = imnoise (img, 'gaussian', 0, 1.);
%
%% Displaying images %%
figure
%% original image
subplot(2,2,1), imshow(img), title('original')
% 0.01 variance
subplot(2,2,2), imshow(imgLowVar), title('Variance 0.01')
% 0.1 variance
subplot(2,2,3), imshow(imgMedVar), title('Variance 0.1')
% 1 variance
subplot(2,2,4), imshow(imgHighVar), title('Variance 1')
%
%% Calculating mse, psnr and snr %%
% mse
mseLowVar = immse(img, imgLowVar);
mseMedVar = immse(img, imgMedVar);
mseHighVar = immse(img, imgHighVar);
% psnr
[psnrLowVar, snrLowVar] = psnr(img, imgLowVar);
[psnrMedVar, snrMedVar] = psnr(img, imgMedVar);
[psnrHighVar, snrHighVar] = psnr(img, imgHighVar);
%
%% Display values in table %%
mse = table(mseLowVar, mseMedVar, mseHighVar)
psnr = table(psnrLowVar, psnrMedVar, psnrHighVar)
snr = table(snrLowVar, snrMedVar, snrHighVar)
```

Output:

- *lightness:*



Con varianza a 1 l'immagine risulta irriconoscibile.

- *mse*

`mse =`

$$\begin{array}{ccc} \text{mseLowVar} & \text{mseMedVar} & \text{mseHighVar} \\ \hline 0.0045772 & 0.054545 & 0.26973 \end{array}$$

- *psnr*

`psnr =`

$$\begin{array}{ccc} \text{psnrLowVar} & \text{psnrMedVar} & \text{psnrHighVar} \\ \hline 23.394 & 12.632 & 5.6907 \end{array}$$

- *snr*

`snr =`

$$\begin{array}{ccc} \text{snrLowVar} & \text{snrMedVar} & \text{snrHighVar} \\ \hline 13.725 & 4.256 & 0.553 \end{array}$$

L'errore quadratico medio aumenta con l'aumentare del valore della varianza, mentre il rapporto segnale rumore e il rapporto segnale rumore di picco diminuiscono all'aumentare del valore della varianza (c'è meno informazione estraibile dall'immagine).

Osservazioni finali

Nell'esercizio è stato studiato cosa succede quando ad un'immagine inserita viene aggiunto del rumore Gaussiano a media nulla variando il valore della varianza.

Si è visto che con l'aumentare del valore della varianza, l'immagine diventa sempre più rumorosa, perché i pixel affetti da rumore diventano tanto più chiari o tanto più scuri rispetto all'originale. Infatti con varianza 1 un pixel nero (0) può diventare addirittura bianco ($0+1 = 1$, cioè bianco in double) e viceversa un pixel nero può diventare bianco. Con varianza 0.01, invece, il livello di grigio del pixel affetto da rumore può variare molto meno ($x \pm 0.01$).

Si è visto il funzionamento della funzione immse() per il calcolo dell'errore quadratico medio e della funzione psnr() per il calcolo del rapporto segnale rumore e il rapporto segnale rumore di picco. Queste funzioni hanno confermato matematicamente l'impressione che maggiore varianza causi la creazione di un'immagine con molta meno informazione.

Esercizio 2.1

Con riferimento alla Lezione 2.1 si consideri l'immagine ‘pout.tif’.

- Applicare all'immagine l'operatore 2 a) di variazione dell'intensità, con tre valori del parametro L: 30, 60, 90.
- Applicare l'operatore 2 b) con tre valori del parametro P: 0.5, 1.5, 2.
- Calcolare il negativo dell'immagine usando l'operatore 2 d).

Codice:

- script

```
%%%%%%%%%%%%%
% Script for applying punctual operator for
% lightness and contrast and for producing the
% negative of an image.
%
% Silvia Gioia Florio, matr. 119328
% Excercise 2.1
%%%%%%%%%%%%%
%
%% Punctual Operator: Lightness
%
% initialize image
img = imread('pout.tif');
% initialize lightness parameters
L1 = 30;
L2 = 60;
L3 = 90;
% apply operator
out1 = lightness(img, L1);
out2 = lightness(img, L2);
out3 = lightness(img, L3);
% display results
figure;
subplot(2,4,1), imshow(img), title('Original Image')
subplot(2,4,5), imhist(img)
subplot(2,4,2), imshow(out1), title('L = 30')
subplot(2,4,6), imhist(out1)
subplot(2,4,3), imshow(out2), title('L = 60')
subplot(2,4,7), imhist(out2)
subplot(2,4,4), imshow(out3), title('L = 90')
subplot(2,4,8), imhist(out3)
%
%% Punctual Operator: Contrast
%
% initialize image
img = imread('pout.tif');
% initialize contrast parameters
P1 = 0.5;
P2 = 1.5;
P3 = 2.;
% apply the operator
out4 = contrast(img, P1);
out5 = contrast(img, P2);
out6 = contrast(img, P3);
% display results
figure;
subplot(2,4,1), imshow(img), title('Original Image')
subplot(2,4,5), imhist(img),
subplot(2,4,2), imshow(out4), title('P = 0.5')
subplot(2,4,6), imhist(out4)
subplot(2,4,3), imshow(out5), title('P = 1.5')
```

```

subplot(2,4,7), imhist(out5)
subplot(2,4,4), imshow(out6), title('P = 2')
subplot(2,4,8), imhist(out6)
%
%% Negative of an image %%
%
% initialize image
img = imread('pout.tif');
% apply operator
out = negative(img);
% display result
figure
subplot(2,2,1), imshow(img), title('Original Image')
subplot(2,2,2), imhist(img)
subplot(2,2,3), imshow(out), title('Negative Image')
subplot(2,2,4), imhist(out)

```

- **funzioni**

- *lightness*:

```

function [ out ] = lightness(img, L)
%%%%%%%%%%%%%
% Function to add lightness punctual operator on an image.
%
% Silvia Gioia Florio, matr. 119328
% Excercise 2.1
%%%%%%%%%%%%%
%%% Initialization %%
% initialize output image
out = uint8(zeros(length(img),length(img(1)))); 
%
%% apply the operator %%
sizes = size(img);
for i=1:sizes(1)
    for j=1:sizes(2)
        out(i,j) = img(i,j)+L;
    end
end

```

- *contrast*:

```

function [ out ] = contrast(img, P)
%%%%%%%%%%%%%
% Function to add contrast punctual operator on an image.
%
% Silvia Gioia Florio, matr. 119328
% Excercise 2.1
%%%%%%%%%%%%%
%%% Initialization %%
% initialize output image
out = uint8(zeros(length(img),length(img(1)))); 
%
%% apply the operator %%
sizes = size(img);
for i=1:sizes(1)
    for j=1:sizes(2)
        out(i,j) = img(i,j)*P;
    end
end

```

- *negative*:

```

function [ out ] = negative(img)
%%%%%%%%%%%%%
% Function to return the negative of an image.
%
% Silvia Gioia Florio, matr. 119328
% Excercise 2.1
%%%%%%%%%%%%%
%%% Initialization %%
% initialize output image

```

```

out = uint8(zeros(length(img),length(img(1))));  

%% apply the operator %%  

sizes = size(img);  

for i=1:sizes(1)  

    for j=1:sizes(2)  

        a = double((img(i,j)))*(-1) + 254;  

        out(i,j) = uint8(a);  

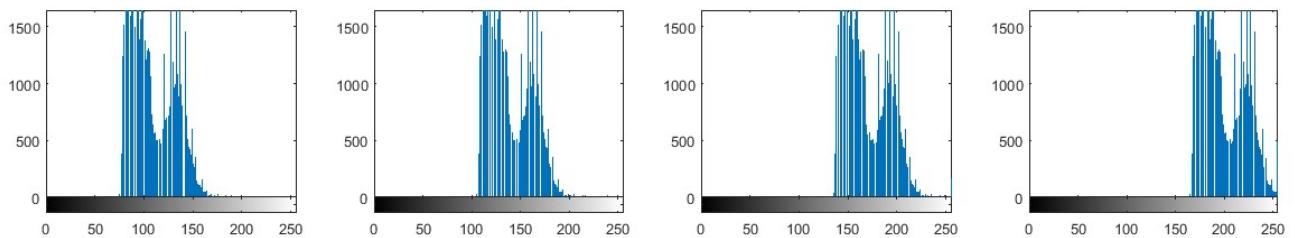
    end  

end

```

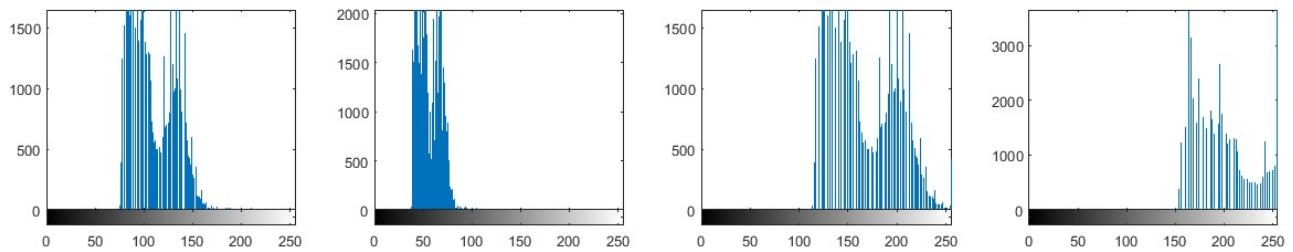
Output:

- *lightness:*



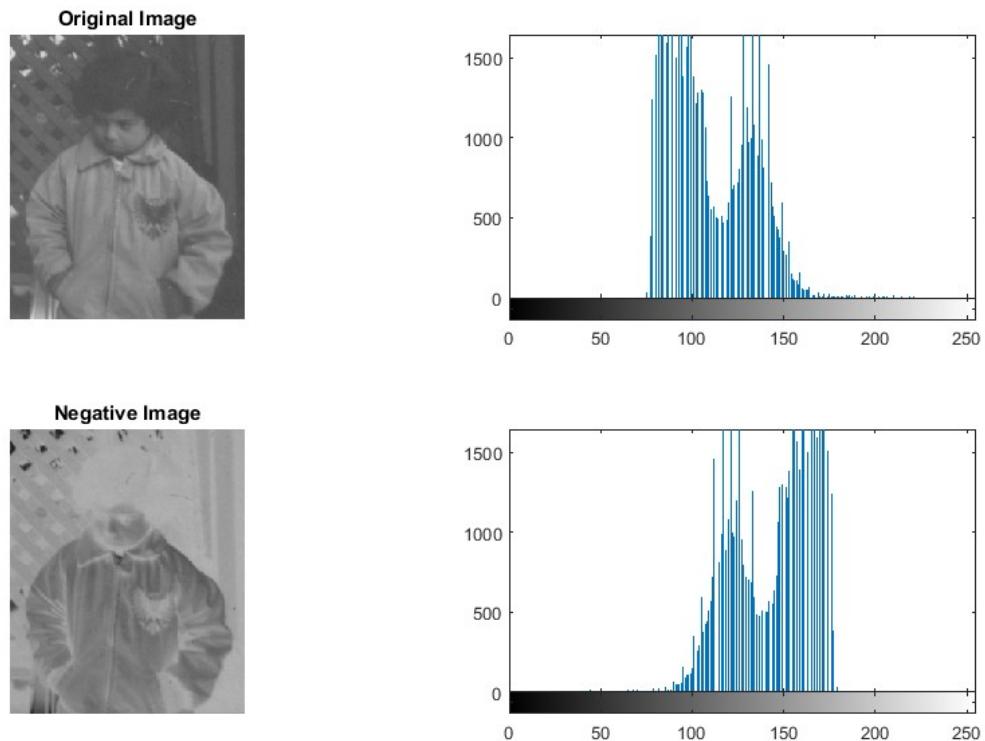
In questo esempio si vede chiaramente tramite l'istogramma come i valori nella scala di grigio mantengano per ogni valore di L lo stesso profilo, solamente shiftato verso valori più alti nella scala.

- contrast:



Si nota come per il valore minore di 1 l'immagine perda contrasto e i valori nella scala di grigio si stringano in una zona più ristretta della scala. Con il valore compreso tra 1 e 2, al contrario, i valori si spalmano in un'area più ampia della scala. Infine con P=2 la zona torna ad essere più ristretta, verosimilmente perché cerca di spingersi più avanti del massimo livello di bianco possibile.

- negative:



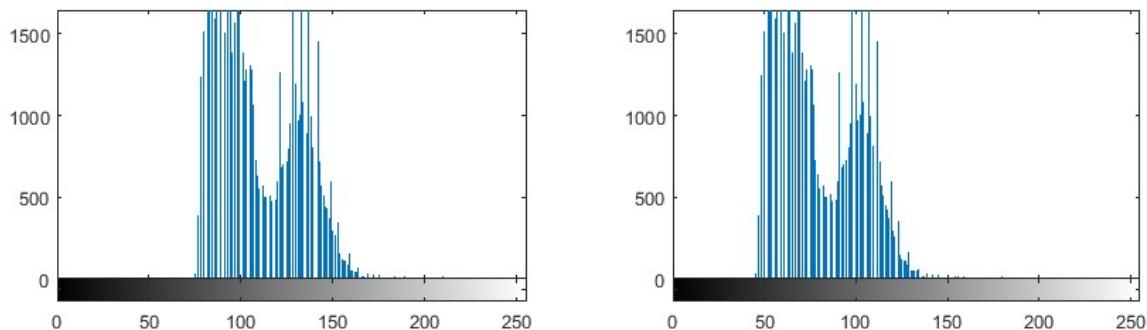
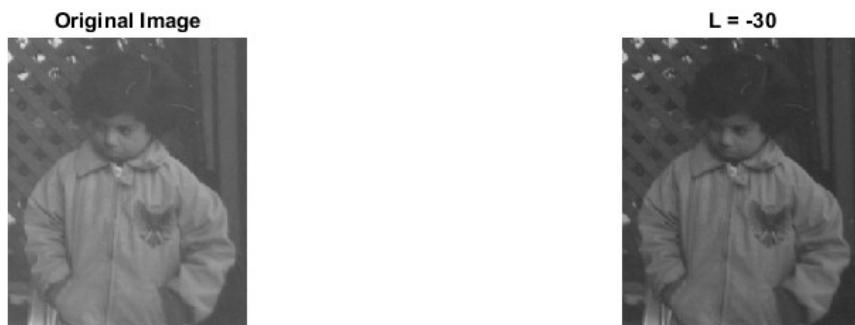
Osservando l'output dello script, si vede chiaramente che i profili dei due istogrammi sono speculari rispetto al valore mediano della scala di livelli di grigio.

Approfondimento: la somma per L negativo:

Per confermare la tesi secondo la quale con un valore negativo della L l'istogramma si sarebbe spostato a sinistra, ho scritto il seguente codice:

```
%% negative values for L %%
out4 = lightness(img, -30);
figure
subplot(2,2,1), imshow(img), title('Original Image')
subplot(2,2,3), imhist(img)
subplot(2,2,2), imshow(out4), title('L = -30');
subplot(2,2,4), imhist(out4)
```

E ottenuto il seguente output:



L'immagine si è effettivamente scurita e l'istogramma si è spostato verso valori più bassi.

Osservazioni finali:

Nello script di questo esercizio abbiamo applicato 3 operatori per la modifica delle immagini.

Abbiamo visto come la somma di una costante ad ogni valore di pixel di un'immagine porti ad una traslazione del suo istogramma a destra e al contempo ad un aumento della luminosità dell'immagine.

Abbiamo visto anche come il prodotto del valore di ogni pixel di un'immagine con una costante porti ad uno stiramento del suo istogramma o alla sua compressione a seconda del valore che sceglieremo come moltiplicatore e dunque ad un aumento o diminuzione del contrasto (e della chiarezza) dell'immagine rispettivamente.

Infine abbiamo visto come l'inversione del valore di ogni pixel di un'immagine cambi il suo istogramma nello speculare rispetto al punto mediano della scala di grigi e visivamente a produrre l'immagine negativa rispetto a quella di partenza.

Esercizio 2.2

Con riferimento alla Lezione 2.1(1.2?) si consideri l'immagine sintetica della barra come sorgente.

- Applicare all'immagine sorgente rumore additivo gaussiano a media nulla e varianza 0.1. Generare complessivamente cinque immagini affette dallo stesso tipo di rumore. Associare a ciascuna immagine il proprio istogramma.
- Calcolare la media pixel per pixel (stacking) delle cinque immagini con rumore.
- Stimare il MSE, rispetto all'immagine sorgente, di ciascuna delle immagini rumorose e dell'immagine calcolata al punto 2.

Codice:

- script

```
%%%%%%%%%%%%%
% script for analizing stacking effect of noisy
% images
%
% Silvia Gioia Florio, matr. 119328
% Excercise 2.2
%%%%%%%%%%%%%

%% initialize synthetic image
% initialize black image
img = zeros(30,30);
% create a white rectangle
img(5:24, 13:17)=1.;

%% adding noise %%
% initialize array of noisy images
noisy = zeros(30,30,5);
% iterate to produce five images
for i=1:5
    % 0.1 variance noise
    noisy(:,:,:,i) = imnoise (img, 'gaussian', 0, 0.1);
end

%% dispay noisy images with histogram %%
figure;
% loop to display noisy images
for i=1:5
    subplot(2,5,i);
    imshow(noisy(:,:,:,i), 'InitialMagnification', 'fit');
    subplot(2,5,5+i);
    imhist(noisy(:,:,:,i));
end

%% apply stacking %%
% call function
res = stacking(noisy);

%% display stacking result %%
% original image
figure;
subplot(2,3,1)
imshow(img,'InitialMagnification', 'fit');
title('Original Image')
subplot(2,3,4)
imhist(img)
% noisy image example
```

```

subplot(2,3,2)
imshow(noisy(:,:1),'InitialMagnification', 'fit');
title('Noisy Image Example')
subplot(2,3,5)
imhist(noisy(:,:1))
% stacked image
subplot(2,3,3)
imshow(res,'InitialMagnification', 'fit');
title('Image after Stacking')
subplot(2,3,6)
imhist(res)

%% compare images using MSE %%
noisyMse = [];
% loop to calculate mse of noisy images
for i=1:5
    noisyMse(i) = mse(img,noisy(:,:i));
end
% calculate stacked image mse
stackingMse = mse(img,res);

%% display results using tables %%
array2table(noisyMse)
table(stackingMse)

```

- funzione per lo stacking

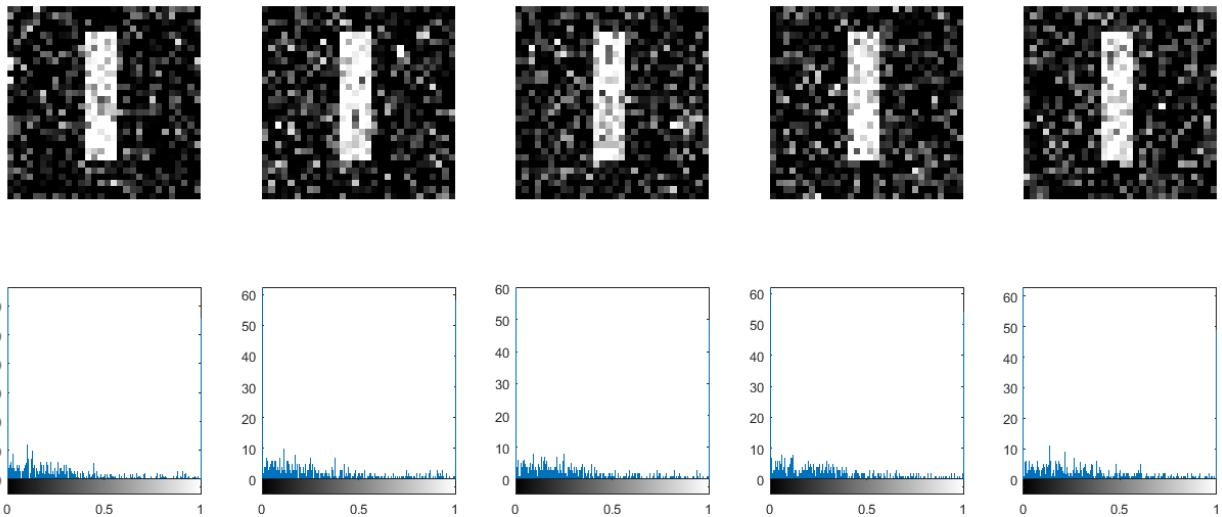
```

function [ out ] = stacking(noisyArray)
%%%%%%%%%%%%%
% Function to calculate mean of every pixel in
% an array of noisy images (stacking)
%
% Silvia Gioia Florio, matr. 119328
% Esercizio 2.2
%%%%%%%%%%%%%
%
%% Initialization %%
% number of images
sizes = size(noisyArray);
% initialize output image
out = zeros(sizes(1),sizes(2));
%
%% calculate mean %%
numOfImages = sizes(3);
for i=1:sizes(1)
    for j=1:sizes(2)
        % summation
        sum = 0;
        for k=1:sizes(3)
            sum = sum+noisyArray(i,j,k);
        end
        % mean
        out(i,j) = (1/numOfImages)*sum;
    end
end

```

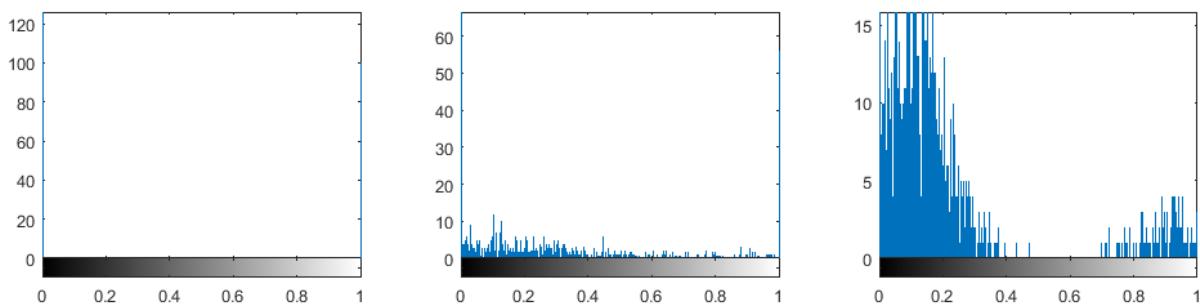
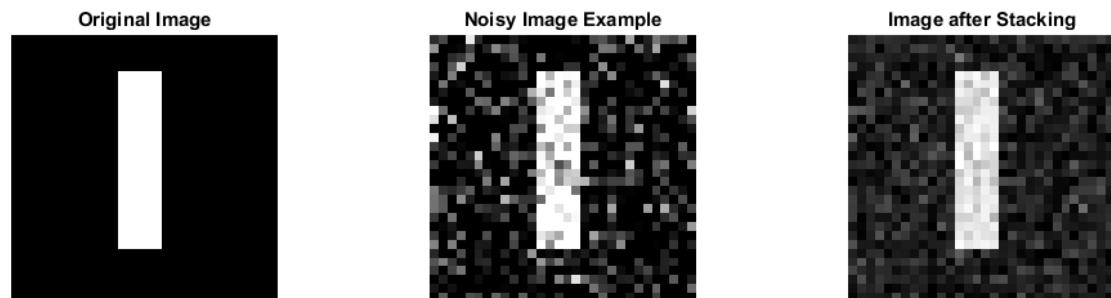
Output:

- 5 immagini con rumore:



Le 5 immagini sono tutte differenti, nonostante siano state create tramite la stessa funzione, questo perché il rumore gaussiano è casuale.

- Risultato dello stacking



Lo stacking prende in input una serie rumorosa della stessa immagine e inserisce nell'output per ogni pixel la media dei valori di quel pixel nelle immagini rumorose. Questo migliora l'immagine grazie alla natura CASUALE del rumore, che inserisce in alcuni pixel un valore più alto o più basso rispetto a quello effettivo, SENZA CAUSALITÀ, consentendo al valore atteso di essere molto vicino al valore originale.

- *MSE*

```
noisyMSE =
```

noisyMse1	noisyMse2	noisyMse3	noisyMse4	noisyMse5
0.048266	0.049314	0.048578	0.049547	0.048847

```
stackingMSE =
```

stackingMse
0.02232

Dall'output si vede come l'MSE dell'immagine creata con lo stacking sia meno della metà rispetto ad ogni singola immagine rumorosa

Osservazioni finali:

In questo esercizio abbiamo visto l'operatore di *stacking* e cercato di capire perché esso, a partire da più immagini rumorose dello stesso soggetto e senza effettivamente modificare alcuna immagine rumorosa, riesca ad approssimare abbastanza bene la figura originale del soggetto iniziale, ragionando sulla naturale casualità del rumore e sfruttandola a nostro vantaggio.

Siccome il rumore è casuale, se per ogni pixel prendo la media dei valori di quel pixel di ogni immagine rumorosa, allora otterrò un valore molto simile a quello originale, migliorando ogni singola immagine rumorosa.

Esercizio 2.3

Con riferimento alla Lezione 2.1 si considerino le immagini ‘moon.tif’ , da prendere come riferimento, e ‘eight.tif’ da considerare come maschera.

- usando la funzione MATLAB imresize() modificare le dimensioni della maschera in modo da renderle uguali a quelle del riferimento.
- Calcolare la combinazione delle due immagini applicando l’operatore 3 c) per cinque valori del parametro α : 0., 0.25, 0.5, 0.75, 1.

Tutte le immagini devono essere visualizzate separatamente assieme al rispettivo istogramma dei livelli di grigio.

Facoltativo: saranno valutati i due possibili sviluppi 3) e 4) qui di seguito.

- Le cinque immagini ottenute al punto 2. potrebbero essere concatenate (procedura cat() di MATLAB) in un collage, cioè un’immagine unica da visualizzare a sua volta;
- Il collage potrebbe essere composto da coppie di immagini diverse da quelle sopra individuate, allo scopo di ottenere risultati efficaci dal punto di vista espressivo.

Codice:

- script

```
%%%%%%%%%%%%%
% script to see the effects of opacity mask on
% an image with different values as alfa
%
% Silvia Gioia Florio, matr. 119328
% Excercise 2.3
%%%%%%%%%%%%%
%% initialize images %%
% read image moon.tif
img = imread('moon.tif');
% read eight.tif as mask
mask = imread('eight.tif');
% resize mask to fit image
sizes = size(img);
mask = imresize(mask,sizes);

%% display image and mask %%
figure
subplot(2,2,1), imshow(img), title('moon')
subplot(2,2,3), imhist(img), grid()
subplot(2,2,2), imshow(mask), title('eight (resized)')
subplot(2,2,4), imhist(mask), grid()

%% initialize opacity values %%
alfaNull = 0.;
alfaLow = 0.25;
alfaMed = 0.5;
alfaHigh = 0.75;
alfaFull = 1;

%% apply opacity mask %%
outNull = opacityMask(img,mask,alfaNull);
outLow = opacityMask(img,mask,alfaLow);
outMed = opacityMask(img,mask,alfaMed);
```

```

outHigh = opacityMask(img,mask,alfaHigh);
outFull = opacityMask(img,mask,alfaFull);

%% display results %%
figure
subplot(2,5,1), imshow(outNull), title('alpha = 0.')
subplot(2,5,6), imhist(outNull), grid()

subplot(2,5,2), imshow(outLow), title('alpha = 0.25')
subplot(2,5,7), imhist(outLow), grid()

subplot(2,5,3), imshow(outMed), title('alpha = 0.5')
subplot(2,5,8), imhist(outMed), grid()

subplot(2,5,4), imshow(outHigh), title('alpha = 0.75')
subplot(2,5,9), imhist(outHigh), grid()

subplot(2,5,5), imshow(outFull), title('alpha = 1')
subplot(2,5,10), imhist(outFull), grid()

%% concatenate output images %%
% concatenate
concat = cat(2, outNull, outLow, outMed, outHigh, outFull);
% display result
figure
imshow(concat);
title('5 opacity masks concatenated');

%% flip the image of the moon %%
% flip around X axis
flippedMoonX = flip(img);
% flip around Y axis
flippedMoonY = flip(img,2);
% flip around X and Y axis
flippedMoonXY = flip(flippedMoonX,2);
% create concatenated output images
% diamond
firstCol = cat(1, img, flippedMoonX);
secondCol = cat(1, flippedMoonY, flippedMoonXY);
diamond = cat(2, firstCol, secondCol);
% hourglass
firstCol = cat(1, flippedMoonX, img);
secondCol = cat(1, flippedMoonXY, flippedMoonY);
hourglass = cat(2, firstCol, secondCol);
% display results
figure
subplot(1,2,1), imshow(diamond), title('moon diamond')
subplot(1,2,2), imshow(hourglass), title('moon hourglass')

```

- funzione per la maschera

```

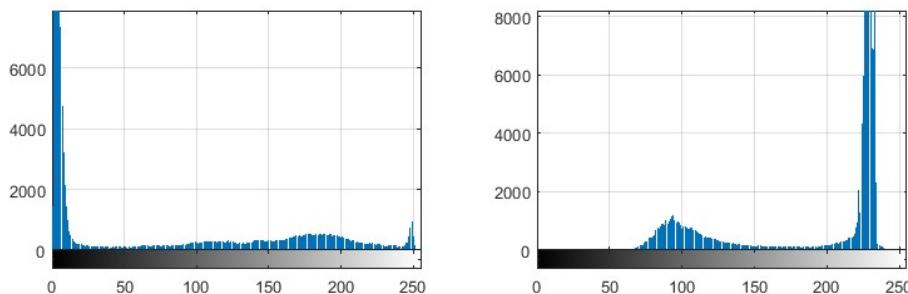
function [ out ] = opacityMask(img, mask, alpha)
%%%%%%%%%%%%%
% Function to apply opacity mask on an image
%
% Silvia Gioia Florio, matr. 119328
% Esercizio 2.3
%%%%%%%%%%%%%
%% Initialization %%
img = double(img);
mask = double(mask);
%
% apply operator
diff = mask - img;
opacity = diff * alpha;
% uint8 translation to show the image
out = uint8(img + opacity);

end

```

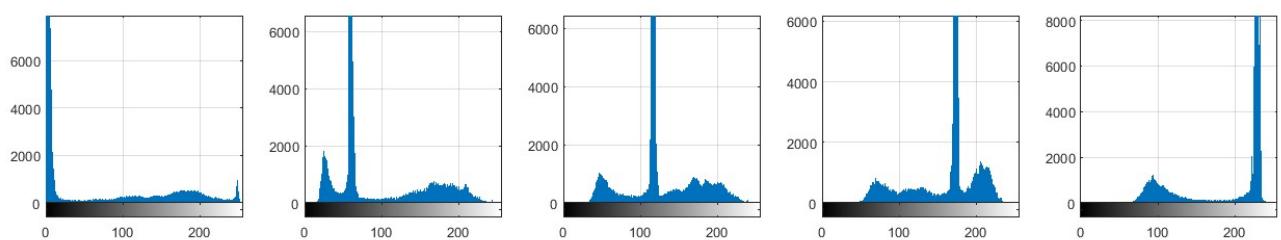
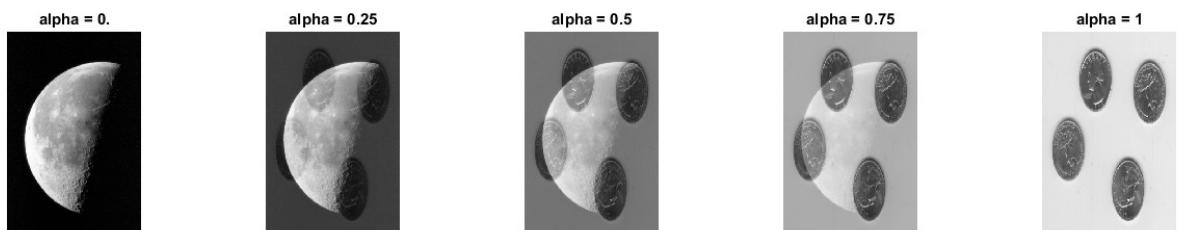
Output:

- Immagine sorgente e maschera:



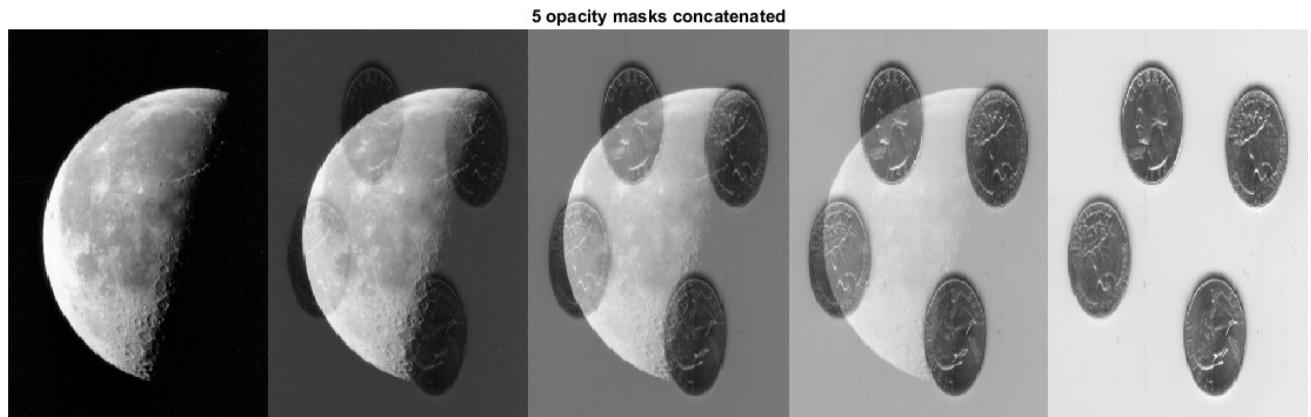
Le immagini scelte hanno un istogramma quasi speculare. Per la futura applicazione della maschera è importante il resize dell'immagine di destra.

- 5 livelli di opacità (5 diversi valori del parametro alfa)



Vediamo che, con l'aumentare del parametro alfa, diminuisce la trasparenza della maschera che diventa sempre più evidente sopra alla luna, fino a coprirla totalmente. Il parametro alfa gestisce quello che sarà il “peso” della maschera sopra all'immagine. Dall'output si nota come il picco nell'istogramma si sposti dal picco dell'immagine originale sempre più verso il picco della maschera all'aumentare di alfa.

- *Concatenazione*



La concatenazione crea un effetto di “fading” dalla luna alle monete.

- *Concatenazioni artistiche*



Per questo esercizio ho voluto sfruttare la forma della luna e la legge di chiusura della Gestalt per dare l'impressione che ci siano delle forme che effettivamente nell'immagine non ci sono.

Osservazioni finali:

Abbiamo visto l'applicazione di una maschera con vari livelli di opacità.

Pensando ad un'applicazione di tale tecnica, mi vengono in mente i frame di un video e mi sembra una tecnica adatta alle transizioni da una scena ad un'altra.

Esercizio 2.4

Con riferimento alla Lezione 2.2 si consideri l'immagine ‘pout.tif’ .

- Trasformare l'immagine in un'altra applicando la trasformazione lineare 1a) della lezione 2.2. Visualizzare le immagini d'ingresso e uscita assieme ai loro istogrammi.
- Visualizzare tramite plot() la dipendenza tra le scale di uscita e d'ingresso outputSc(lev) e inputSc(lev)
- script

```
%%%%%%%%%%%%%
% script to apply stretching on the histogram of
% an image to increase contrast
%
% Silvia Gioia Florio, matr. 119328
% Excercise 2.4
%%%%%%%%%%%%%
% initialize images %
% read image pout.tif
img = imread('pout.tif');
% apply transform
stretched = stretch(img);
% display images
figure
subplot(2,2,1), imshow(img), title('pout')
subplot(2,2,3), imhist(img), grid()
subplot(2,2,2), imshow(stretched), title('pout (stretched)')
subplot(2,2,4), imhist(stretched), grid()

% output scale and input scale dependency %%
img = double(img);
% initialize parameters
kmax = max(max(img));
fmin = min(min(img));
% input scale
inputScale = fmin:1.0:kmax;
% output scale
outputScale = (kmax / (kmax - fmin)) * (inputSc - fmin);
% display results
figure
plot(inputScale, outputScale), xlabel('Input Scale'), ylabel('Output Scale')
title('Input and Output scales mapping'), grid
```

- funzione per lo stretch

```
function [ out ] = stretch(img)
%%%%%%%%%%%%%
% Function for stretching the histogram of an
% image.
%
% Silvia Gioia Florio, matr. 119328
% Esercizio 2.4
%%%%%%%%%%%%%
% Initialization %
img = double(img);
% initialize parameters
kmax = max(max(img));
fmin = min(min(img));
% apply transform %
out = (kmax / (kmax - fmin)) * (img - fmin);
out = uint8(out);

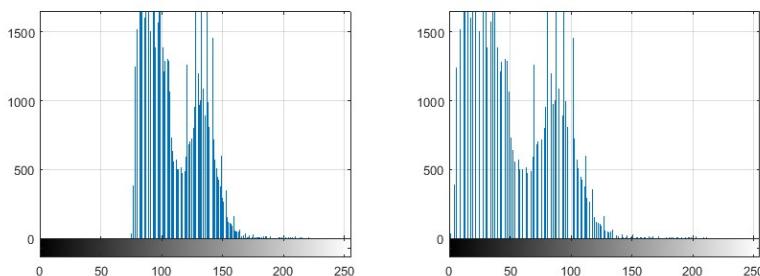
end
```

Output:

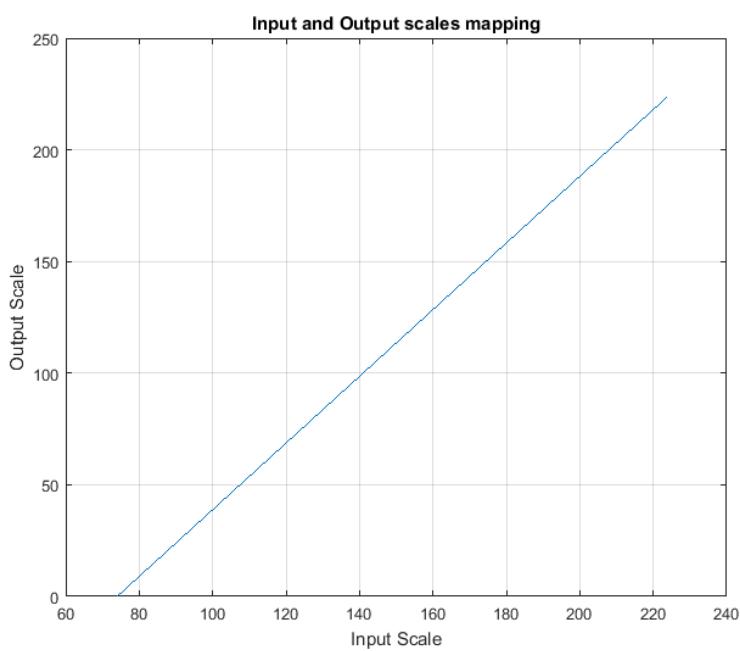
- Risultato dell'operazione



Con lo stretching dell'istogramma, valori lontani al suo interno lo diventano ancora di più, evidenziando maggiormente la differenza a livello visivo nell'immagine.



- Confronto tra input scale and output scale



Osservando il confronto tra le due scale si nota come una scala di valori che in input si sparge solamente in valori compresi circa tra 70 e 230 diventa molto più ampia, tra 0 e 230.

Osservazioni finali:

In questo esercizio abbiamo appreso come lo stiramento nei valori dell'istogramma porti ad un aumento nel contrasto di un'immagine poiché corrisponde ad un ulteriore allontanamento dei valori tra loro.

Mettendo in relazione la scala di input e quella di output abbiamo anche notato che lo stiramento mantiene lo stesso andamento dei valori, solamente ampiandone i confini e spalmando i valori nella nuova scala proporzionalmente all'originale (linea retta).

Esercizio 2.5

Con riferimento alla Lezione 2.2 si consideri l'immagine ‘moon.tif’ .

- Definire numericamente le due scale $\text{inputSc}(\text{lev})$ e $\text{outputSc}(\text{lev})$, $0 \leq \text{lev} \leq 1$; la $\text{inputSc}(\text{lev})$ ha due punti discontinuità in R_1, R_2 ; la $\text{outputSc}(\text{lev})$ è non-nulla solo in $[R_1 \ R_2]$; è nulla altrove in $[0 \ 1]$. L’andamento della funzione di trasformazione tra le scale è riportato in Figura 1. Effettuare il calcolo per i seguenti valori dei parametri: $R_1=0.4$; $R_2=0.6$; $S_2=1$; nel sottointervallo $[0 \ R_1]$ è $S_1=0$; nel sottointervallo $[R_2 \ 1]$ è $S_3=0$. Visualizzare tramite plot() di MATLAB la dipendenza tra $\text{outputSc}(\text{lev})$ e $\text{inputSc}(\text{lev})$. Visualizzare tramite plot() la dipendenza tra le scale di uscita e d’ingresso $\text{outputSc}(\text{lev})$ e $\text{inputSc}(\text{lev})$
- Trasformare i livelli di grigio dell’immagine d’ingresso $f(i,j)$ applicando le regole definite al precedente punto 1. L’implementazione può essere effettuata tramite istruzioni di condizione if/elseif. Visualizzare le immagini di ingresso e uscita dopo la trasformazione.
- Ripetere il calcolo e la visualizzazione per le bande di livelli di grigio seguenti: $[R_1 \ R_2] = [0, 0.2] ; [0.2 \ 0.4] ; [0.4 \ 0.6] \text{ già calcolato}; [0.6 \ 0.8] ; [0.8 \ 1.]$ Il parametro $S_2 = 1$. in tutti i sottointervalli precedenti. Al di fuori di questi è $S_1=S_3=0$ in tutti i casi.

Tutte le immagini devono essere visualizzate assieme al rispettivo istogramma dei livelli di grigio.

- script

```
%%%%%%%%%%%%%
% Script for seeing the results of thresholding on
% an image
%
% Silvia Gioia Florio, matr. 119328
% 30.03.2017 - Excercise 2.5
%%%%%%%%%%%%%
%
%% artificial input scale %%
% initialization
% input scale
inputScale = 0:1/255:1;
% parameters
r = [0.4, 0.6];
s = [0,1,0];
% apply transform
outputScale = thresholding(inputScale, r, s);
%
% plot dependency
figure;
plot(inputSc,outputSc,'r'), title('Isolating values between .4 and .6')
%
%% using moon.tif as input scale %%
% initialization
img = imread('moon.tif');
img = double(img)/255;
% apply transform
out = thresholding(img,r,s);
%
% display result
figure;
subplot(2,2,1), imshow(img), title('Original image')
subplot(2,2,3), imhist(img), grid
subplot(2,2,2), imshow(out), title('Isolating values between .4 and .6')
subplot(2,2,4), imhist(out), grid
%
%% changing values in r %%

```

```
% initialization
rs = [[0 .2]; [.2 .4]; [.4 .6]; [.6 .8]; [.8 1]];
length = size(rs);
length = length(1);
outs = zeros(537,358,5);
%
% apply transform
for i=1:length
    r1 = rs(i,1);
    r2 = rs(i,2);
    outs(:,:,:,i) = thresholding(img,[r1,r2],s);
end
%
% display results
figure;
for i=1:length
    subplot(2,5,i), imshow(outs(:,:,:,:,i)),
    if i~=5
        title(sprintf('values between %.d and %.d', 2*i-2, 2*i))
    else
        title(sprintf('values between %.d and 1.', 2*i-2))
    end
    subplot(2,5,5+i), imhist(outs(:,:,:,:,i)), grid
end
%
%% playing with values of r and s %%
% isolating more than one band %
% ends
rMore1 = [.2 .8];
sMore1 = [1 0 1];
imgMore1 = thresholding(img, rMore1, sMore1);
% 2 bands
rMore2 = [.2 .3 .7 .8];
sMore2 = [0 1 0 1 0];
imgMore2 = thresholding(img, rMore2, sMore2);
%
% display results
figure
subplot(2,2,1), imshow(imgMore1), title('ends isolated')
subplot(2,2,3), imhist(imgMore1), grid
subplot(2,2,2), imshow(imgMore2), title('Isolating btween .2 and .3 and between .7 and .8')
subplot(2,2,4), imhist(imgMore2), grid
%
% set r and s to sample image in less grayscale bands %
% initialization
% 16 grayscale bands (4 bits) %
r4bits = 1/16:1/16:(1-1/16);
s4bits = 0:1/15:1;
img4bits = thresholding(img, r4bits, s4bits);
% 8 grayscale bands (3 bits) %
r3bits = 1/8:1/8:(1-1/8);
s3bits = 0:1/7:1;
img3bits = thresholding(img, r3bits, s3bits);
% 4 grayscale bands (2 bits) %
r2bits = 1/4:1/4:(1-1/4);
s2bits = 0:1/3:1;
img2bits = thresholding(img, r2bits, s2bits);
%
% display results
figure
subplot(2,4,1), imshow(img), title('Original Image')
subplot(2,4,5), imhist(img), grid
subplot(2,4,2), imshow(img4bits), title('16 bands - 4 bits')
subplot(2,4,6), imhist(img4bits), grid
subplot(2,4,3), imshow(img3bits), title('8 bands - 3 bits')
subplot(2,4,7), imhist(img3bits), grid
subplot(2,4,4), imshow(img2bits), title('4 bands - 2 bits')
subplot(2,4,8), imhist(img2bits), grid
% closer look to original and 4-bit images
```

```

figure
subplot(1,3,1), imshow(img), title('Original Image')
subplot(2,3,3), imhist(img), title('Original Image histogram'), grid
subplot(1,3,2), imshow(img4bits), title('16 bands - 4 bits')
subplot(2,3,6), imhist(img4bits), title('16 bands histogram'), grid

```

- **funzione**

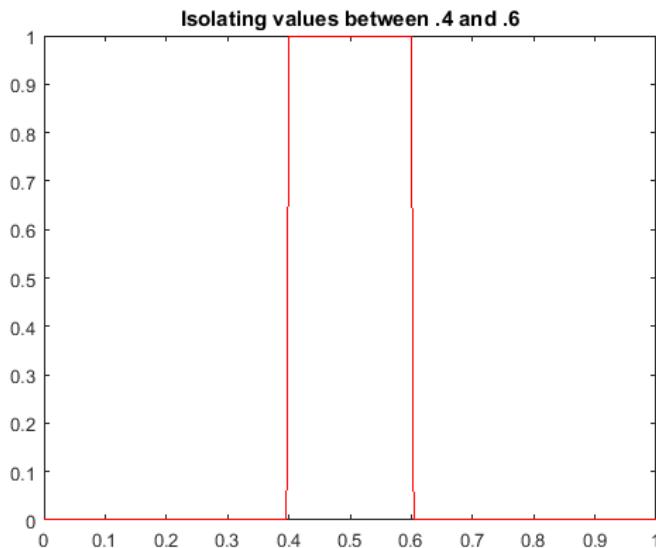
```

function [ out ] = thresholding(input, r, s)
%%%%%%%%%%%%%
% Function for slicing an image by thresholding.
% Input values are divided in bands by vector r
% and then normalized by vector s.
%
% Silvia Gioia Florio, matr. 119328
% 30.03.2017 - Excercise 2.5
%%%%%%%%%%%%%
%
%% Initialization %%
out = input;
% initialize parameters
sizeIn = size(input);
sizeR = size(r);
sizeS = size(s);
sizeR = sizeR(2);
sizeS = sizeS(2);
% errors
if(sizeR~=sizeS-1)
    disp('Incompatible sizes');
    return
end
%
%% apply operator %%
for i=1:sizeIn(1)
    for j=1:sizeIn(2)
        k=1;
        while(k<=sizeR && r(k)<input(i,j))
            k=k+1;
        end
        out(i,j) = s(k);
    end
end
end

```

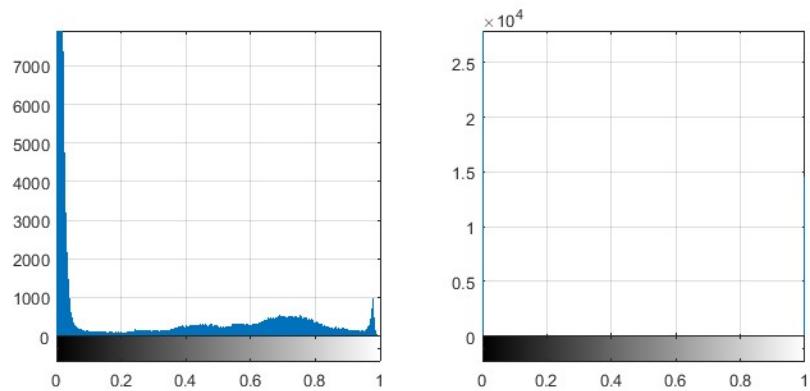
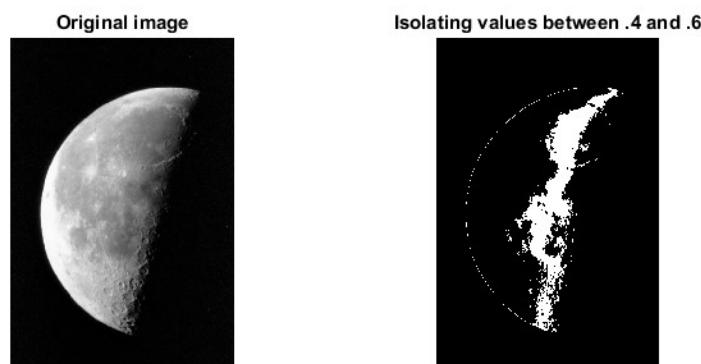
Output:

- Esempio di cosa fa la funzione



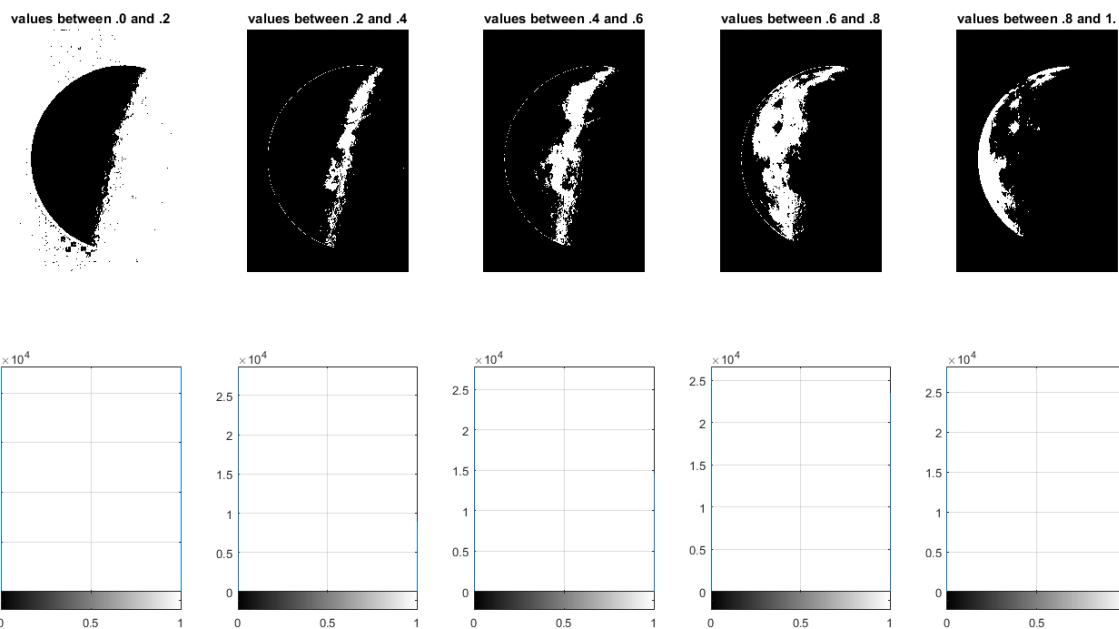
Si può vedere come la funzione selezioni solamente i valori tra .4 e .6 e li accentui a 1, mentre i valori al di fuori di quel range li annulli.

- Isolamento nell'immagine della luna



In questo esempio vediamo quali sono nell'immagine della luna i valori tra .4 e .6.

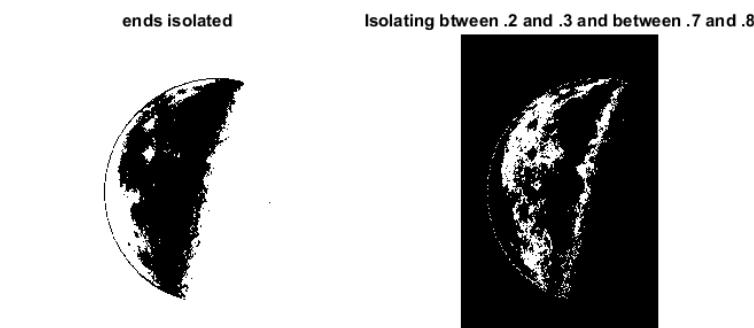
- Isolamento di bande diverse



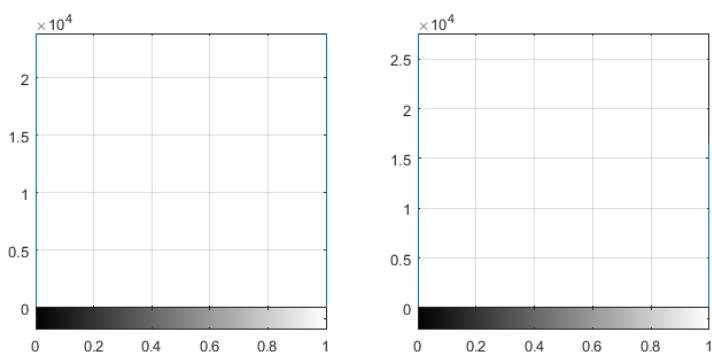
Questo procedimento mette in evidenza qual è la banda preponderante nell'immagine e qual è il peso di ogni banda all'interno dell'immagine. In questo caso la banda con il maggior numero di valori è quella tra 0 e .2, mentre peso minore ce l'ha quella tra .2 e .4.

Approfondimenti

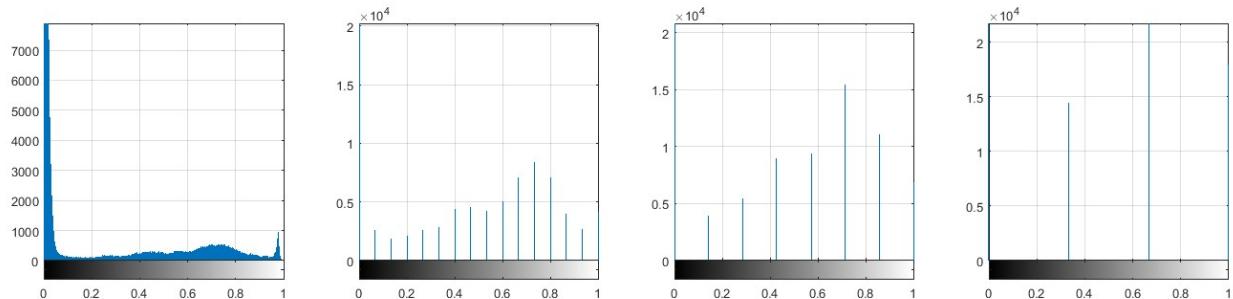
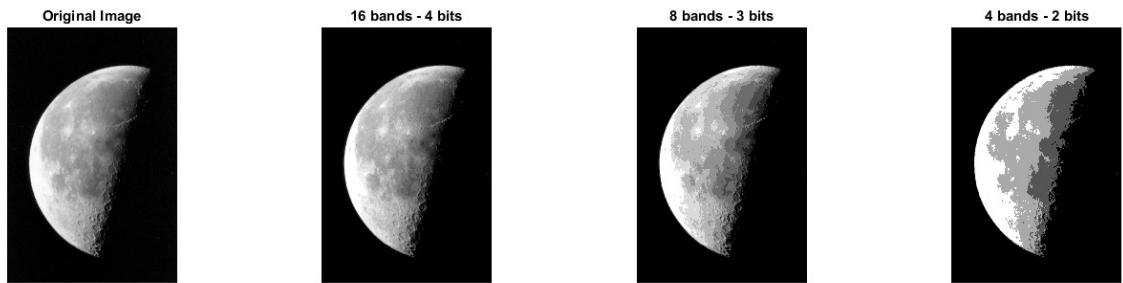
- Isolamento di due bande diverse



Ho voluto potenziare la funzione di modo che possa isolare più di una banda di un'immagine. Nell'esempio ho isolato gli estremi (tra 0 e .2 e tra .8 e 1) e due bande in mezzo.

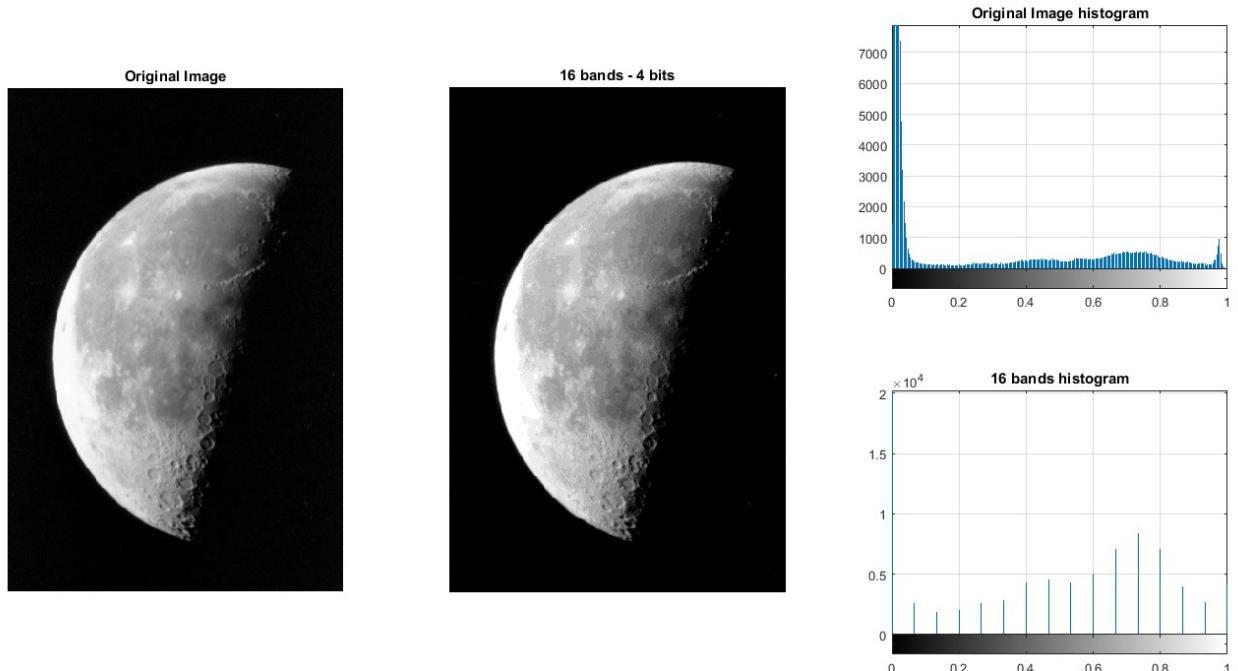


- *Campionamento e quantizzazione*



Ho utilizzato la nuova funzione per suddividere la scala di grigi in parti uguali e riassegnare i valori di grigio di modo che la scala sia molto ridotta e dunque usi meno bit per essere memorizzata. Ho provato con 16, 8 e 4 bande, rispettivamente 4, 3 e 2 bits.

- *Focus on: 256 bande e 16 bande*



Particolarmente mi ha stupito il risultato ottenuto campionando in sole 16 bande. La differenza con l'immagine originale è stata molto minore di quanto mi aspettassi.

Osservazioni finali:

In questo esercizio abbiamo studiato come isolare dei valori della scala di grigi inclusi in un determinato range, in modo da enfatizzare le zone in cui il livello di grigio appartiene all'intervallo di nostro interesse e capire qual è il peso di questo intervallo nell'immagine.

Si possono anche isolare più di un intervallo e assegnare ad ognuno un valore differente di modo, ad esempio, da campionare ulteriormente l'immagine

Esercizio 2.6

Con riferimento alla Lezione 2.2 si consideri l'immagine ‘westconcordaerial.png’.

- Visualizzare l'immagine sorgente. Convertire l'immagine in un'altra a livelli di grigio tramite la procedura `rgb2gray()`. Visualizzare l'immagine ottenuta e il relativo istogramma.
- Usare la procedure MATLAB `grayslice()` per partizionare l'immagine a livelli di grigio in 16 bande. Visualizzare l'immagine partizionata usando la colormap `jet()`.

Facoltativo. Navigare in Google Earth relativamente a una zona del Friuli-Venezia Giulia opp. del Veneto. Usare lo zoom per ottenere un'immagine più ravvicinata della zona che si è scelta. Effettuare le operazioni ai punti 1., 2. sopra specificati, eventualmente modificando il numero di bande opp. la colormap in modo da ottenere risultati visivamente utili.

Codice:

- script

```
%%%%%%%%%%%%%
% Script for turning an image into grayscale and
% see the effects using colormaps
%
% Silvia Gioia Florio, matr. 119328
% Excercise 2.6
%%%%%%%%%%%%%
%
%% West Concord %%
%
% initialize image
img = imread('westconcordaerial.png');
%
% display image
figure
imshow(img,'InitialMagnification', 'fit'), title('West Concord aerial')
%
%% convert image to grayscale %%
gray = rgb2gray(img);
%
% display grayscale image and its histogram
figure;
subplot(1,2,1), imshow(gray), title('grayscale image')
subplot(1,2,2), imhist(gray), title('grayscale image histogram'), grid;
%
%% divide in 16 gray levels %%
sliced = grayslice(gray, 16);
%
% display result
figure
imshow(sliced, jet(16)), colorbar, title('West Concord aerial - colormap jet')
%
%% Cervignano map %%
%
% initialize image
imgC = imread('cervignano-aerial.png');
%
% display image
figure
imshow(imgC,'InitialMagnification', 'fit')
title('Cervignano aerial')
%
%% convert image to grayscale %%
```

```

grayC = rgb2gray(imgC);
%
% display grayscale image and its histogram
figure;
subplot(1,2,1), imshow(grayC), title('grayscale image')
subplot(1,2,2), imhist(grayC), title('grayscale image histogram'), grid;
%
%% divide in 3 gray levels %%
slicedC = grayslice(grayC, 3);
%
% display result
figure
imshow(slicedC, pink(3)), colorbar, title('Dark red zones are parks and green zones of
the area');

```

Output:

- *West Concord*



Immagine aerea di un'area urbana.

- *West Concord in grayscale*

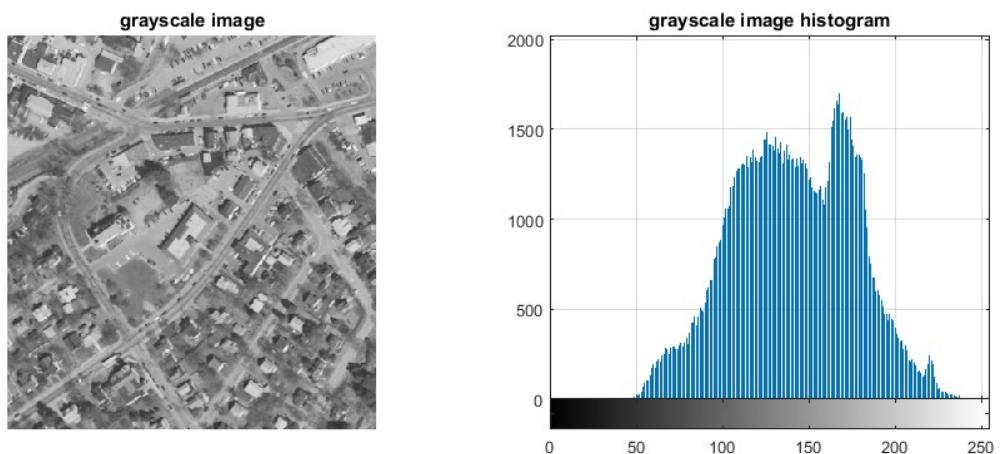
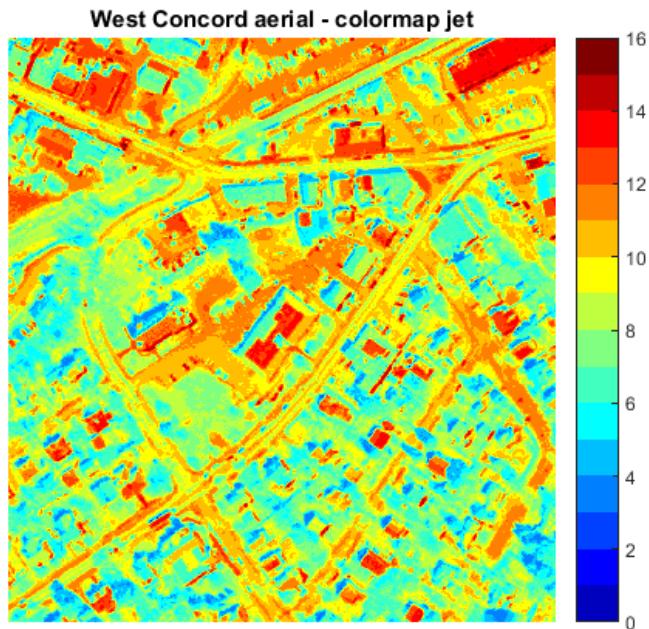


Immagine tradotta in scala di grigio

- West Concord colormap: jet



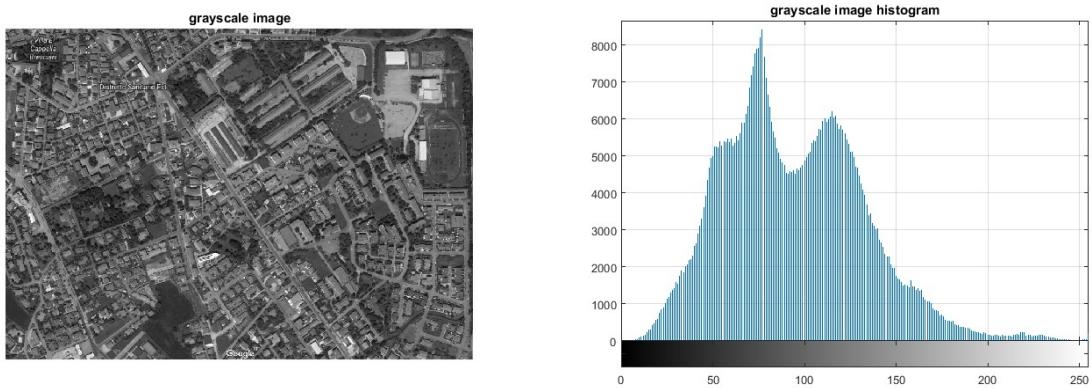
L'immagine in scala di grigi è stata divisa in 16 bande e tramite la colormap ad ogni banda è stato assegnato un certo colore.

- Cervignano map



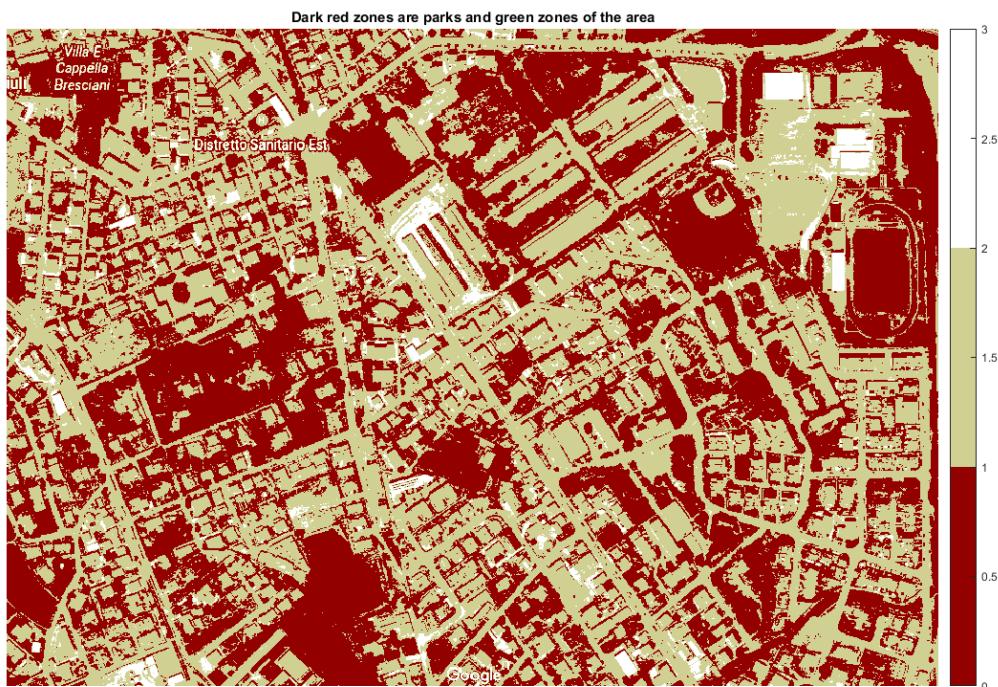
Per la parte facoltativa ho voluto inquadrare la zona di Cervignano dove ho vissuto gran parte della mia vita.

- Cervignano: grayscale



Nell'immagine in bianco e nero ho cercato di capire cosa fosse bello evidenziare e ho deciso di far vedere le zone verdi, più scure nell'immagine in bianco e nero.

- Cervignano: colormap pink



Essendo il grigio dei parchi molto simile a quello delle strade, ma un po' più scuro ho dovuto eseguire una divisione molto secca in sole 3 bande e cercare una colormap che evidenziasse molto solo la parte più scura. La parte rosso scuro, alla fine, rappresenta abbastanza bene la presenza di parchi e aree verdi nella zona.

Osservazioni finali:

Nello script abbiamo visto come la suddivisione di un'immagine in bande di grigio può essere utile per evidenziare determinati aspetti di un'immagine. Tali aspetti possono essere ulteriormente enfatizzati con l'utilizzo di una mappa di colore opportuna.

Esercizio 2.7

Con riferimento alla Lezione 2.2 si consideri l'immagine ‘cameraman.tif’ .

- Selezionare gli otto piani di bit usando la procedura MATLAB bitget(). Visualizzare gli otto piani di bit dell'immagine.
- Passo di verifica. Partendo dal piano del bit 1 (il meno significativo), generare i n-1 piani di bit successivi moltiplicando il piano precedente per il fattore 2 dove n=2,...,8 Verificare, calcolando le differenze, che i piani così generati coincidano con quelli ricavati al punto 1.
- Ricostruire l'immagine originale a partire dai suoi piani di bit calcolati al punto 2. Verificare, calcolando la differenza, che l'immagine originale e quella ricostruita coincidono. Ogni immagine deve essere associata al proprio istogramma dei livelli di grigio.
- Effettuare ricostruzioni parziali dell'originale comando soltanto alcuni piani di bit, e valutare quantitativamente la perdita di informazione tramite stima del MSE nelle situazioni seguenti: (a) immagine approssimata dal solo piano del bit 8 (il più significativo); (b) comando soltanto i piani 7,8; (c) i piani 6,7,8; (d) i piani 5,6,7,8. Riportare i risultati tramite table().

Codice:

- script

```
%%%%%%%%%%%%%
% Script for understanding weight single bits
% in defining the color of an image
%
% Silvia Gioia Florio, matr. 119328
% Esercizio 2.7
%%%%%%%%%%%%%
%
% initialization
% image
img = imread('cameraman.tif');
% outputs
len = size(img);
bits           = zeros(len(1,1), len(1,2), 8);
weigthedBits   = bits;
weigthedBitsNorm = bits;
diffBits       = bits;
diffBitsInt    = bits;
%
% calculations
for i=1:8
    % bitget
    bits(:,:,:,i) = bitget(img,i, 'uint8');
    % weighted bits
    weigthedBits(:,:,:,i) = bits(:,:,:,i) * 2^(i-1);
    % normalization
    weigthedBitsNorm(:,:,:,i) = weigthedBits(:,:,:,i) / 255;
    weigthedBitsNorm(:,:,:,i) = im2bw(weigthedBitsNorm(:,:,:,i), 0);
    % difference
    diffBits(:,:,:,i) = weigthedBitsNorm(:,:,:,i) - bits(:,:,:,i);
end
%
% display results
% bits
```

```

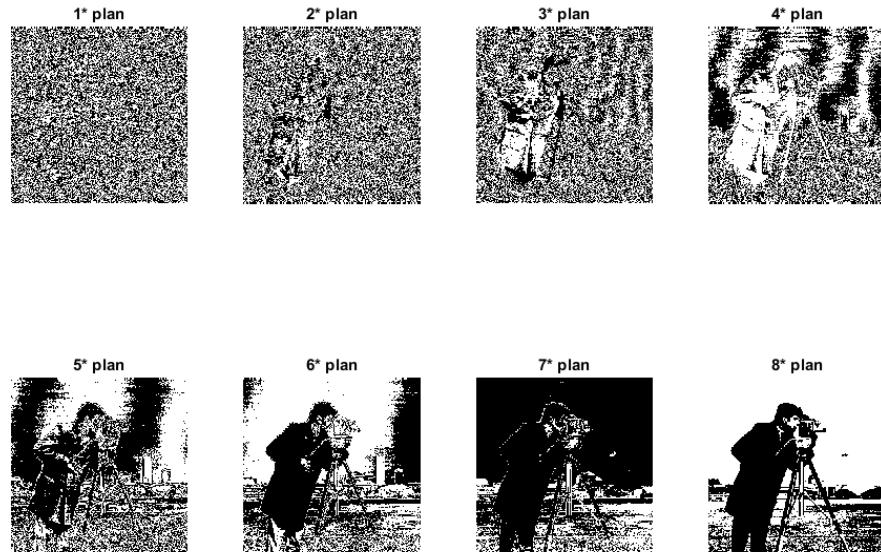
figure
for i=1:8
    subplot(2,4,i), imshow(bits(:,:,:,i)), title(sprintf('%d* plan', i))
end
% weighted bits
figure
for i=1:8
    subplot(2,4,i), imshow(weigthedBits(:,:,:,i)), title(sprintf('%d* plan', i))
end
% difference
figure
for i=1:8
    subplot(2,4,i), imshow(diffBits(:,:,:,i)), title(sprintf('%d* plan', i))
end
%
%% return to the original %%
%
% initialization
% sum of bits plans
bitsSum = sum(weigthedBits, 3);
% difference between original and reconstructed
diffImg = double(img)-bitsSum;
%
% display results
figure();
subplot(2,3,1), imshow(img), title('Original Image')
subplot(2,3,4), imhist(img), grid
subplot(2,3,2), imshow(uint8(bitsSum)), title('Reconstructed Image')
subplot(2,3,5), imhist(uint8(bitsSum)), grid
subplot(2,3,3), imshow(uint8(diffImg)), title('Difference')
subplot(2,3,6), imhist(uint8(diffImg)), grid
%
%% partial reconstructions %%
%
% initialization
% 8th plan
img8 = weigthedBits(:,:,:8);
img8 = uint8(img8);
mse8 = immse(img, img8);
% 8th and 7th plans
img78 = sum(weigthedBits(:,:,:7:8), 3);
img78 = uint8(img78);
mse78 = immse(img, img78);
% 8th, 7th, 6th
img678 = sum(weigthedBits(:,:,:6:8), 3);
img678 = uint8(img678);
mse678 = immse(img, img678);
% 8th, 7th, 6th and 5th
img5678 = sum(weigthedBits(:,:,:5:8), 3);
img5678 = uint8(img5678);
mse5678 = immse(img, img5678);
%
% display results
% figure
figure
subplot(2,5,1), imshow(img), title('Original Image')
subplot(2,5,6), imhist(img), grid
subplot(2,5,2), imshow(img8), title('8th')
subplot(2,5,7), imhist(img8), grid
subplot(2,5,3), imshow(img78), title('8th and 7th')
subplot(2,5,8), imhist(img78), grid
subplot(2,5,4), imshow(img678), title('8th, 7th and 6th')
subplot(2,5,9), imhist(img678), grid
subplot(2,5,5), imshow(img5678), title('8th, 7th, 6th and 5th')
subplot(2,5,10), imhist(img5678), grid
% mse table
imgDescription = {'Piano 8';'Piani 7+8';'Piani 6+7+8';'Piani 5+6+7+8'};
imgMSE = [mse8;mse78;mse678;mse5678];
MSETable = table(imgDescription, imgMSE);
MSETable.Properties.VariableNames{'imgDescription'} = 'bit_plans';

```

```
MSETable.Properties.VariableNames{'imgMSE'} = 'mse';
MSETable
```

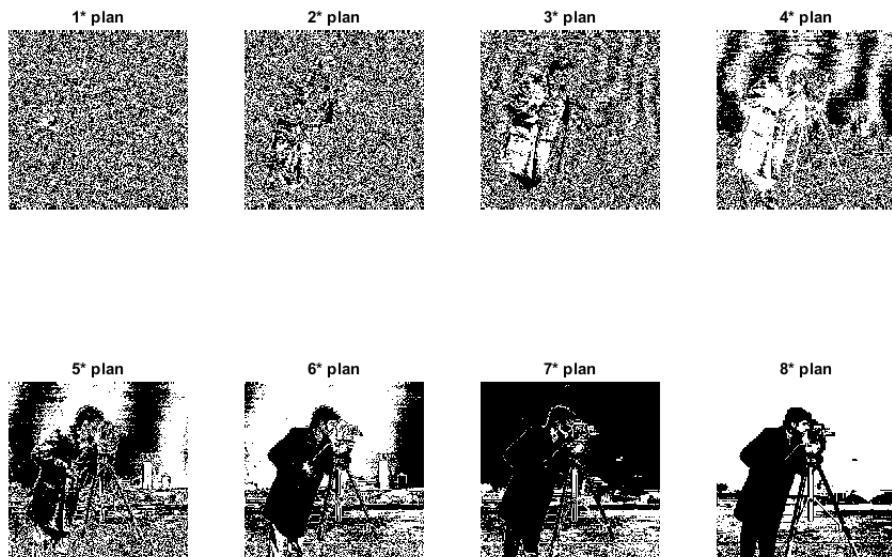
Output:

- *Piani di bit*

Si

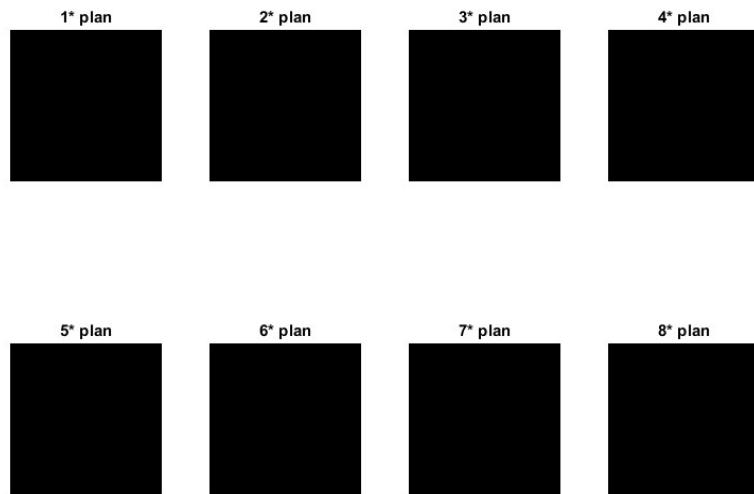
nota come al salire della significatività del bit, aumenta anche il peso che esso porta nel trasmettere informazione nell'immagine.

- *Piani pesati*



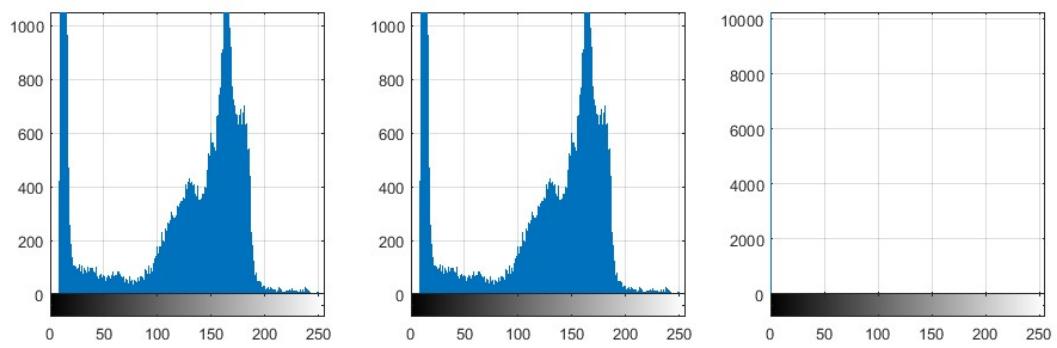
I piani di bit dell'esercizio precedente sono stati moltiplicati per il loro peso all'interno del numero di 8 bit. Il risultato è lo stesso.

- *Differenze*



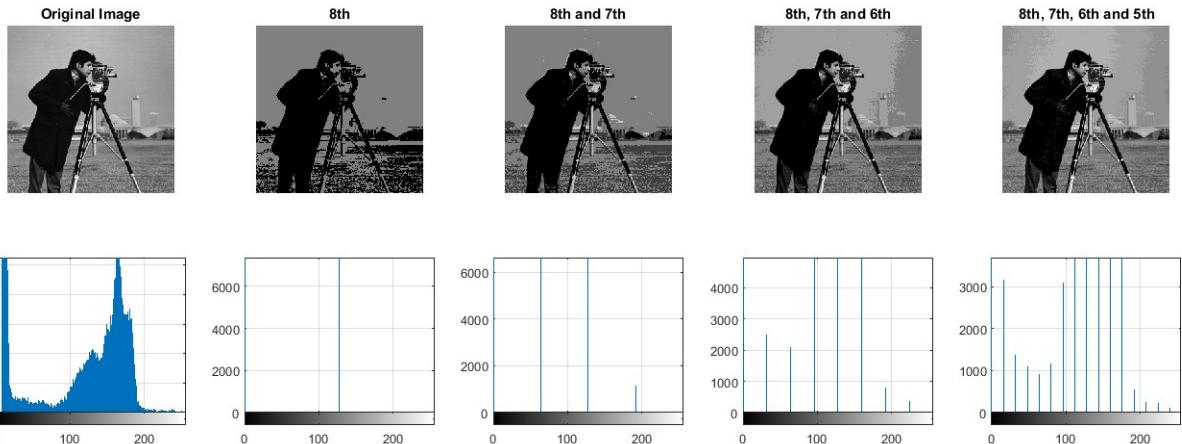
I piani di bit ottenuti sono uguali, la loro differenza è infatti uguale a 0.

- *Ricostruzione*



L'immagine in mezzo è stata derivata dalla somma dei piani di bit estrapolati al punto 2. Nel processo non c'è stata alcuna perdita di informazione, infatti la differenza delle due immagini è 0.

- Somme parziali



Si possono anche effettuare delle somme parziali di piani di bit per avere un'idea più chiara del peso dell'informazione trasportata da un determinato piano.

- MSE

bit_plans	mse
'Piano 8'	2924.7
'Piani 7+8'	1263.3
'Piani 6+7+8'	286.54
'Piani 5+6+7+8'	82.156

Il calcolo dell'errore quadratrico medio evidenzia come all'aggiunta di un piano di informazione esso più che dimezza. In questo caso un grande salto ce l'abbiamo aggiungendo il sesto piano.

Osservazioni finali:

In questa esercitazione abbiamo cercato di capire qual è la quantità di informazione rilevante per l'occhio umano che viene trasportata da ogni singolo bit del numero a 8 bit che rappresenta il pixel, cioè qual è il peso di ogni bit nell'immagine e abbiamo capito che il bit più significativo è effettivamente il più significativo. L'immagine, infatti, risulta riconoscibile anche solo guardando il piano di quel singolo bit. L'aggiunta di piani successivi, in ogni caso, contribuisce a diminuire molto l'errore quadratrico medio.

Esercizio 2.8

E' data l'immagine sintetica della scacchiera (checkerboard), che deve essere rappresentata in forma binaria, di dimensioni complessive 256x256. Applicare alla scacchiera rumore additivo gaussiano a media nulla, una volta con varianza 0,1 (rumore basso); un'altra con varianza 1 (rumore alto).

- Visualizzare l'immagine sorgente della scacchiera e le due immagini con rumore, assieme ai rispettivi istogrammi dei livelli di grigio;
- Effettuare il filtraggio per la riduzione del rumore tramite filtro di media (average) con maschera 3x3. Visualizzare le seguenti immagini: originale; affetta da rumore basso; affetta da rumore alto; immagine filtrata dal rumore basso; filtrata dal rumore alto; tutti gli istogrammi relativi;
- Il filtraggio è stato utile a ridurre il rumore, sia basso che alto? Come si può stimare quantitativamente l'efficacia del filtraggio? Applicare la stima e calcolare l'efficacia del filtraggio. Riportarla in una tabella;
- Ripetere i calcoli effettuando il filtraggio solo nel caso di rumore basso, con lo stesso operatore di media (average) e maschere di dimensione 5x5 e 7x7; visualizzare le immagini filtrate e i rispettivi istogrammi;
- Confrontare il risultato dei filtri ottenuti con le tre finestre: 3x3, 5x5, 7x7, a rumore basso. Riportare le stime in una tabella. Quale finestra ha dato il risultato migliore?
- Riportare i risultati dei confronti nelle osservazioni finali.

Usare le seguenti primitive MATLAB: checkerboard(), imnoise(), fspecial(), imfilter(), immse(), table().

Facoltativo: ripetere i calcoli effettuati ai punti 4 e 5 applicandoli anche al caso di rumore alto. Ricavare la nuova tabella e le rispettive conclusioni.

Codice:

- script
- ```
%%%%%
% Script for different noise filtering
%
% Silvia Gioia Florio, matr. 119328
% 04.04.2017 - Excercise 2.8
%%%%%
%
%% add noise %%
% define initial checkerboard
img = double(checkerboard(32)>0.5);
%
% show checkerboard and relative histogram
figure;
% checkerboard
subplot(3,2,1)
imshow(img);
title('checkerboard');
% histogram
subplot(3,2,2)
```

```

imhist(img)
grid()
title('checkerboard - histogram');
%
% initialize low and high noises
% low
low = imnoise (img, 'gaussian', 0.0, 0.1);
histLow = imhist(low);
% high
high = imnoise (img, 'gaussian', 0.0, 1);
histHigh = imhist(high);
%
% show noisy checkerboard and relative histogram
% low-noisy
subplot(3,2,3)
imshow(low);
title('noise low');
% low-noisy histogram
subplot(3,2,4)
imhist(low);
grid()
title('noise low - histogram');
% high-noisy
subplot(3,2,5)
imshow(high);
title('noise high');
% high-noisy histogram
subplot(3,2,6)
imhist(high);
grid()
title('noise high - histogram');
%
%% filtering %%
%
% filter the noisy checkerboards with average mask
filter = fspecial('average');
filteredLow = imfilter(low, filter);
filteredHigh = imfilter(high, filter);

% display differences
figure;
% low noisy filtered image and its histogram
% image
subplot(2,2,1)
imshow(filteredLow);
title('noise low - filtered');
% histogram
subplot(2,2,2)
imhist(filteredLow);
grid()
title('noise low - filtered: histogram');
% low noisy filtered image and its histogram
% image
subplot(2,2,3)
imshow(filteredHigh);
title('noise high - filtered');
% histogram
subplot(2,2,4)
imhist(filteredHigh);
grid()
title('noise high - filtered: histogram');
%
%% esteem %%
%
% initialization
mseNoiseLow = immse(img, low);
mseNoiseHigh = immse(img, high);

mseFilteredLow = immse(img, filteredLow);
mseFilteredHigh = immse(img, filteredHigh);

```

```
mseNoise = [mseNoiseLow; mseNoiseHigh];
mseFiltered = [mseFilteredLow; mseFilteredHigh];

table(mseNoise, mseFiltered)
%
%% masks Low Noise %%
%
% initialization
filter = fspecial('average', 5);
filteredLow5 = imfilter(low, filter);

figure;

subplot(2,2,1)
imshow(filteredLow5);
title('noise low - filtered 5x5');
subplot(2,2,2)
imhist(filteredLow5);
grid()
title('filtered 5x5: histogram');

filter = fspecial('average', 7);
filteredLow7 = imfilter(low, filter);

subplot(2,2,3)
imshow(filteredLow7);
title('noise low - filtered 7x7');
subplot(2,2,4)
imhist(filteredLow7);
grid()
title('filtered 7x7: histogram');
%
%% masks comparison %%
mse3x3Low = immse(img, filteredLow);
mse5x5Low = immse(img, filteredLow5);
mse7x7Low = immse(img, filteredLow7);

table(mse3x3Low, mse5x5Low, mse7x7Low)

%
%% masks high noise %%
%
% initialization
filter = fspecial('average', 5);
filteredHigh5 = imfilter(high, filter);

figure;

subplot(2,2,1)
imshow(filteredHigh5);
title('noise high - filtered 5x5');
subplot(2,2,2)
imhist(filteredHigh5);
grid()
title('filtered 5x5: histogram');

filter = fspecial('average', 7);
filteredHigh7 = imfilter(high, filter);

subplot(2,2,3)
imshow(filteredHigh7);
title('noise high - filtered 7x7');
subplot(2,2,4)
imhist(filteredHigh7);
grid()
title('filtered 7x7: histogram');
%
%% mask comparison %%
```

```

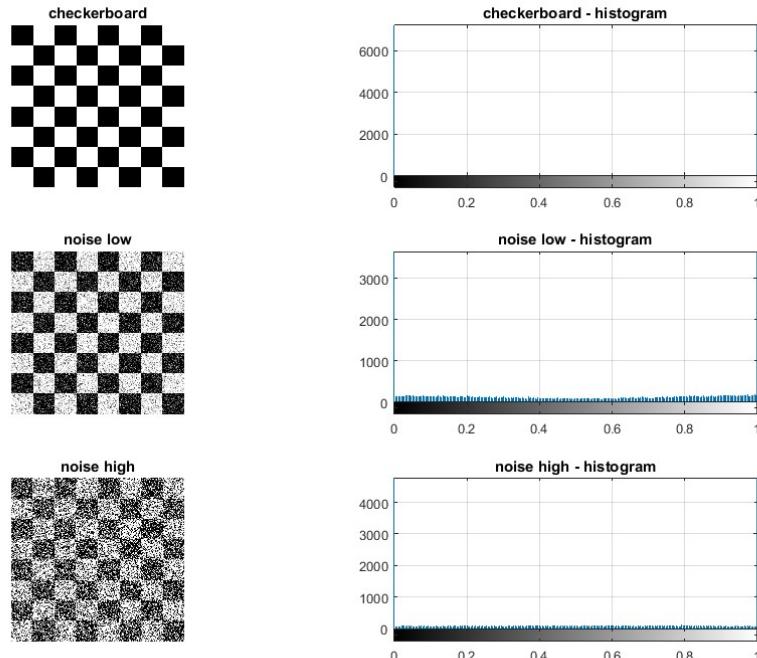
mse3x3High = immse(img, filteredHigh);
mse5x5High = immse(img, filteredHigh5);
mse7x7High = immse(img, filteredHigh7);

table(mse3x3High, mse5x5High, mse7x7High)

```

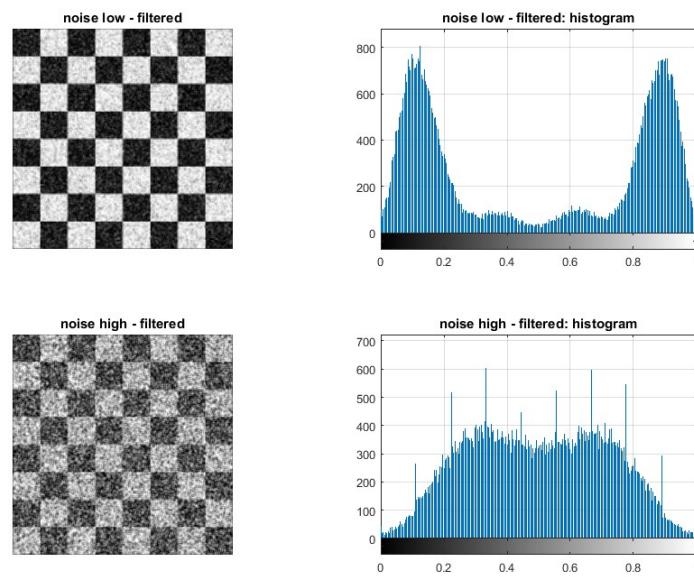
**Output:**

- *Scacchiera rumorosa*



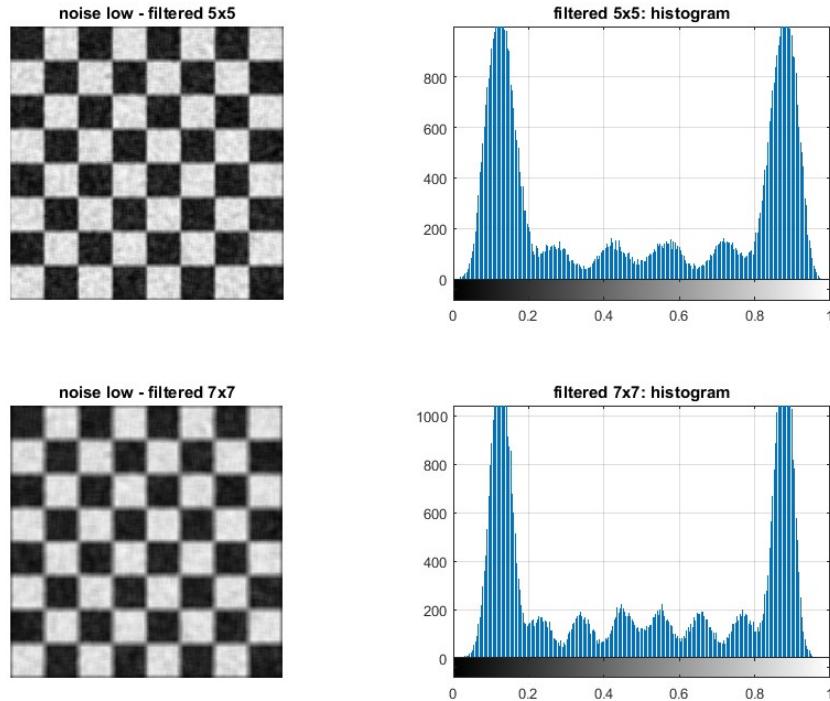
*Dagli istogrammi si vede come il rumore sparga i valori all'interno della scala di grigi, anche partendo da un'immagine fortemente binaria.*

- *Filtro di media con finestra 3x3*



*I filtri cercano di ripristinare i due picchi agli estremi. Nel secondo caso non è molto efficace.*

- Filtro di media  $5 \times 5$  e  $7 \times 7$  sull'immagine poco rumorosa



Si nota come allargando la finestra in questo caso sembra esserci un miglioramento ulteriore dell'immagine, con un ulteriore assottigliarsi e alzarsi dei due picchi alle estremità

- mse

`mseNoise`

---

0.049331

0.2572

`mse3x3Low`    `mse5x5Low`    `mse7x7Low`

---

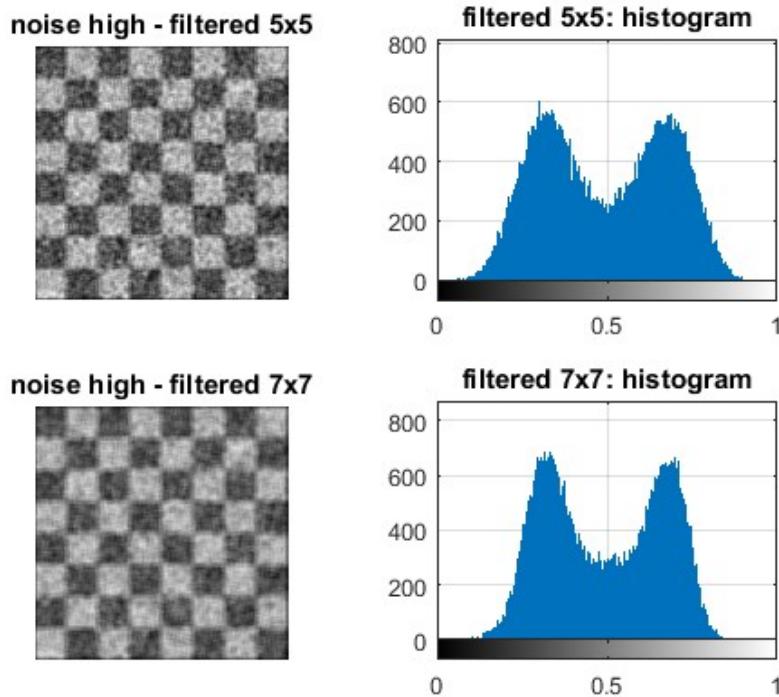
0.034089

0.043143

0.053372

Il calcolo dell'errore quadratico medio, però, ci dice che il risultato migliore ce l'ha in realtà la finestra più piccola e che la finestra  $7 \times 7$  addirittura aumenta l'errore rispetto all'immagine rumorosa

- Filtro di media  $5 \times 5$  e  $7 \times 7$  sull'immagine molto rumorosa



Anche in questo caso più la finestra è grande più si alzano i due picchi, in questo caso non proprio agli estremi, poiché il rumore ha reso l'immagine molto grigia invece che bianca e nera.

- mse

`mseNoise`

---

0.049331

0.2572

`mse3x3High`    `mse5x5High`    `mse7x7High`

---

0.12752

0.12469

0.12969

In questo caso nessuna delle finestre peggiora l'errore quadratico medio, ma la finestra più efficace è quella  $5 \times 5$ , anche se gli errori rimangono molto alti e molto vicini tra loro

### Osservazioni finali:

In questa esercitazione abbiamo iniziato ad analizzare il filtraggio del rumore. Abbiamo visto come aumentare la dimensione della finestra del filtro non vuol dire migliorare il filtraggio, ma neanche vuol dire peggiorarlo. Analizzando due diversi casi, infatti abbiamo visto come due finestre differenti possono essere adatte a rumori di diversa intensità.



## Esercizio 2.9

E' data l'immagine 'text.png'.

- Usando la procedura imtool() individuare una regione rettangolare dell'immagine che corrisponda al carattere 'e'. Ritagliare la sottoimmagine corrispondente usando la procedura imcrop(). Visualizzare l'immagine sorgente e la sottoimmagine ritagliata.
- Calcolare la correlazione incrociata normalizzata con la procedura normxcorr2() ; visualizzare la correlazione rappresentandola in livelli di grigio (assieme al proprio istogramma) e in grafica 3d tramite la procedura surf()
- Binarizzare l'immagine della correlazione tramite sogliatura (thresholding), usando la im2bw() con valore di soglia threshold=0.8.
- Presentare i risultati nelle osservazioni finali.

### Facoltativo

- Ripetere il calcolo della correlazione (passo 2) avendo ruotato di 90° l'immagine ritagliata del carattere 'e'.
- Ripetere i passi 3, 4 con i nuovi risultati.

### Codice:

- script

```
%%%%%%%%%%%%%
% Script for understanding cross correlation
% effects
%
% Silvia Gioia Florio, matr. 119328
% 04/04/2017 - Excercise 2.9
%
%%%%%%%%%%%%%
%
%% crop
%
% initialization
img = imread('text.png');
crop = imcrop(img, [29 11 12 12]);
%
% display result
figure;
subplot(1,2,1), imshow(img), title('text.png')
subplot(1,2,2), imshow(crop), title('cropped "e"')

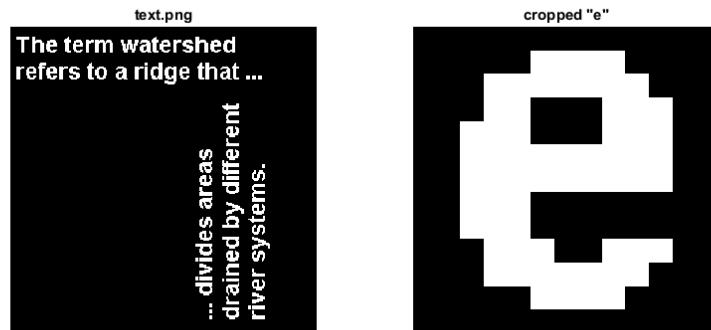
%% cross correlation
%
% calculate correlation
res = normxcorr2(crop,img);
%
% display results
figure
%image
subplot(1,3,1), imshow(res), title('image after correlation'),
%histogram
subplot(1,3,2), imhist(res), title('histogram'), grid
%3d representation
subplot(1,3,3), surf(res), title('3d colored surface')

%% Thresholding %%
bwres = im2bw(res,0.8);
```

```
%
% display results
figure
%image
imshow(bwres, 'InitialMagnification', 'fit'), title('image after correlation, sliced at .
.8')
```

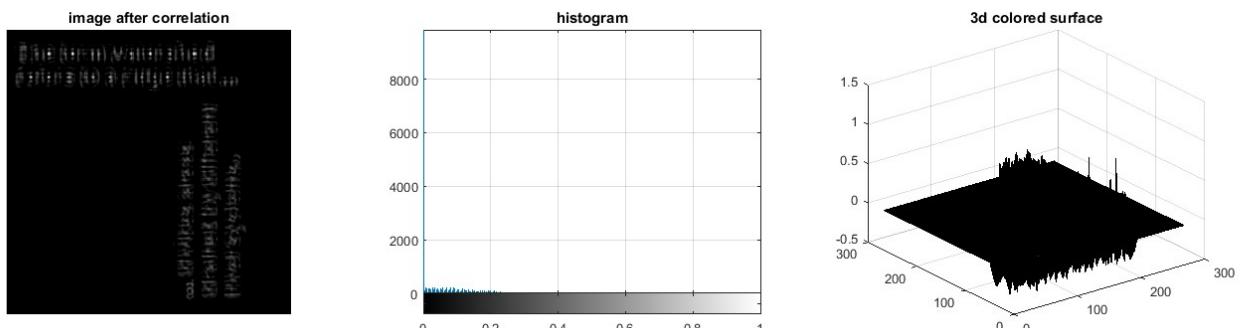
**Output:**

- *Image and crop*



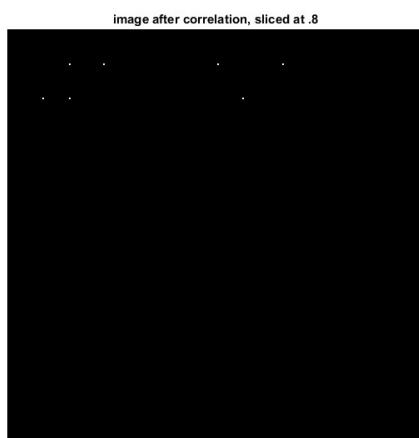
*Abbiamo preso una delle 'e' della frase e l'abbiamo ritagliata.*

- *Correlazione*



*Dopo aver effettuato la correlazione incrociata, essa sembra aver generato del rumore piuttosto casuale.*

- *Risultato binarizzato*



*Dopo aver binarizzato l'immagine, però, esce l'informazione importante. I punti bianchi sono esattamente dove erano le 'e' nel testo.*

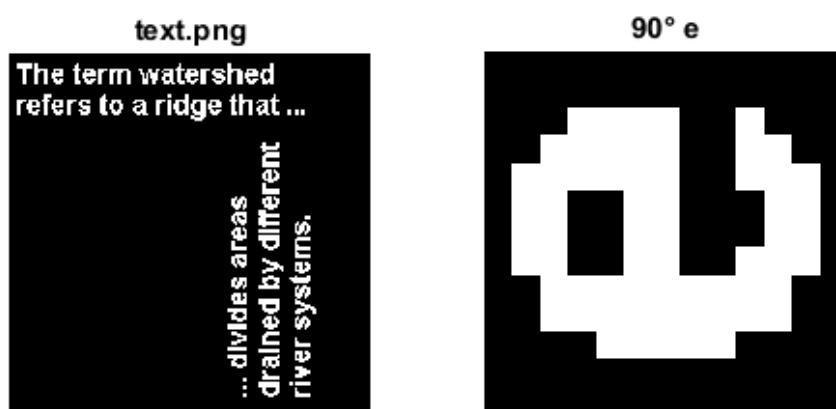
## Facoltativo: immagine ruotata di 90°

- script

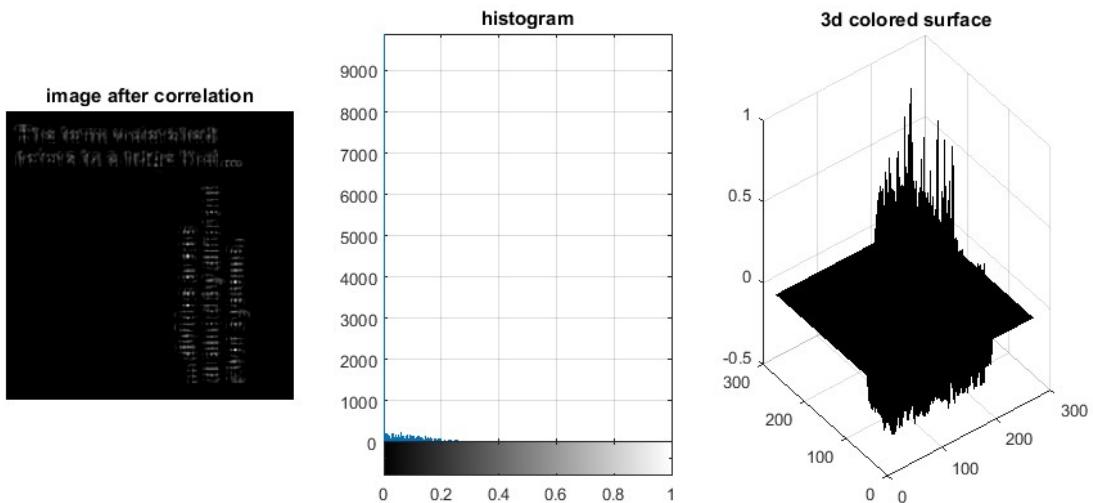
```
%% crop
% initialization
img = imread('text.png');
crop = imcrop(img, [29 11 12 12]);
crop = imrotate(crop, 90);
%
% display result
figure;
subplot(1,2,1), imshow(img), title('text.png')
subplot(1,2,2), imshow(crop), title('90° e')
%
%% cross correlation
% calculate correlation
res = normxcorr2(crop,img);
%
% display results
figure
%image
subplot(1,3,1), imshow(res), title('image after correlation'),
%histogram
subplot(1,3,2), imhist(res), title('histogram'), grid
%3d representation
subplot(1,3,3), surf(res), title('3d colored surface')
%
%% Thresholding %%
bwres = im2bw(res,0.8);
%
% display results
figure
%image
imshow(bwres, 'InitialMagnification', 'fit'), title('image after correlation, sliced at .8')
```

## Output:

- Immagine e 'e' ruotata

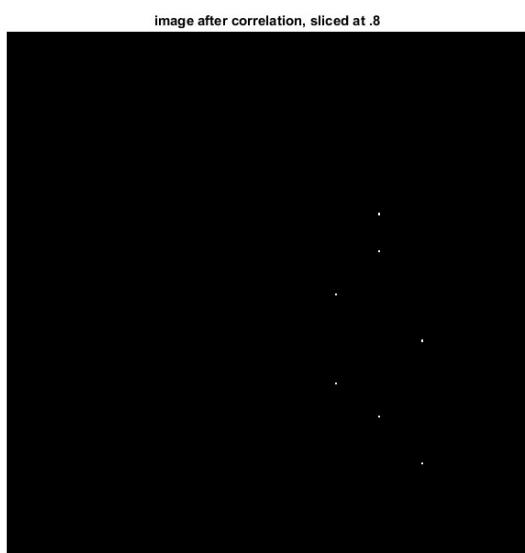


- *Correlazione*



*Anche in questo caso non sembra essere uscita dell'informazione dalla correlazione.*

- *Risultato binarizzato*



*Ecco che, nuovamente, dopo la sogliatura i puntini bianchi rimasti si trovano dove erano le 'e' ruotate di 90° all'interno del testo.*

### Osservazioni finali:

L'esercitazione ha spiegato l'utilizzo della correlazione incrociata per il riconoscimento di un determinato pattern all'interno di un'immagine.

È importante notare che per utilizzare al meglio questa procedura è necessario effettuare un'operazione di binarizzazione scegliendo adeguatamente il livello entro cui effettuare la sogliatura.

## Esercizio 2.10

Data l'immagine ‘eight.tif’, simulare la presenza di rumore casuale additivo nelle forme: gaussiana a media nulla, varianza 0.01 e 0.1 (basso e alto rumore rispettivamente).

### Analisi del rumore

- Visualizzare l'immagine senza rumore (ideale); l'immagine a basso e alto rumore di entrambi i tipi. Visualizzare il profilo dei livelli di grigio lungo una riga significativa (scanline) nelle cinque immagini. Usare la procedura imtool() in modalità interattiva per individuare le coordinate-pixel iniziale e finale di un segmento dell'immagine, e la procedura improfile() da codice.
- Considerare le immagini con rumore basso-alto di entrambi i tipi. Rispetto all'immagine originale presa come riferimento, stimare: il MSE usando la immse(); il rapporto segnale-rumore di picco (psnr) usando la procedura psnr(). Riportare i risultati in una tabella.

### Filtraggio gaussiano del rumore gaussiano

- Applicare a ciascuna delle due immagini con rumore gaussiano, tre filtri anch'essi gaussiani con finestra 3x3 e tre valori di deviazione standard: 0.1, 0.5, 1.
- Stimare il MSE; la varianza e il psnr delle immagini dopo i filtraggi.
- Recuperare le stime di MSE e psnr calcolate al punto 2. Confrontare le stime prima e dopo il filtraggio, per ciascun valore di deviazione standard del filtro, e riportare i dati in una nuova tabella. Stabilire quale dei tre filtri ha dato il risultato migliore e riportarlo nelle osservazioni finali.

### Facoltativo

Ripetere i calcoli ai passi 4, 5, 6 per le immagini con rumore ‘sale e pepe’ basso e alto.

### Codice:

- script

```
%%%%%
% Script for the analysis of noise and gaussian
% filtering.
%
% Silvia Gioia Florio, matr. 119328
% 11/04/2017 - Esercizio 2.10
%%%%%
%% profiling %%
%
% initialize image "eight.tif"
img = imread('eight.tif');
img = double(img)/255;
%
% initialize noises %
%
% variance 0.01 - low noise
gaussLow = imnoise(img,'gaussian', 0, 0.01);
% variance 0.1 - high noise
gaussHigh = imnoise(img,'gaussian', 0, 0.1);
%
% initialize coordinates vectors %
```

```

x = [19 300];
y = [96 96];
%
% display images and profiles %
figure;
% original image
subplot(3,2,1);
imshow(img);
title('eight.tif');
% original image profile
subplot(3,2,2)
improfile(img,x,y), grid on;
title('eight.tif profile (19,96)-(300,96)');
% low noise image
subplot(3,2,3);
imshow(gaussLow);
title('low gaussian noise');
% low noise image profile
subplot(3,2,4)
improfile(gaussLow,x,y), grid on;
title('low gaussian: profile (19,96)-(300,96)');
% high noise image profile
subplot(3,2,5);
imshow(gaussHigh);
title('high gaussian noise');
% high noise image profile
subplot(3,2,6)
improfile(gaussHigh,x,y), grid on;
title('high gaussian: profile (19,96)-(300,96)');
%
%% mse %%
% calculate mse %
mseNoiseLow = immse(img, gaussLow);
mseNoiseHigh = immse(img, gaussHigh);
%
% calculate peak signal-to-noise ratio %
psnrNoiseLow = psnr(img, gaussLow);
psnrNoiseHigh = psnr(img, gaussHigh);
%
% create table %
MSE = [mseNoiseLow;mseNoiseHigh];
PSNR = [psnrNoiseLow;psnrNoiseHigh];

table(MSE, PSNR, 'RowNames', {'NoiseLow', 'NoiseHigh'})
%
%% filtering %%
% initialize filters %
gauss01=fspecial('gaussian',3, 0.1);
gauss05=fspecial('gaussian',3, 0.5);
gauss1=fspecial('gaussian',3, 1);
%
% apply filters low noise %
imgFG01Low= imfilter(gaussLow,gauss01);
imgFG05Low= imfilter(gaussLow,gauss05);
imgFG1Low= imfilter(gaussLow,gauss1);
%
% output low noise %
% noisy images
figure
subplot(3,3,1), imshow(gaussLow)
subplot(3,3,4), imshow(gaussLow)
subplot(3,3,7), imshow(gaussLow)
%filters mesh
subplot(3,3,2), mesh(gauss01)
subplot(3,3,5), mesh(gauss05)
subplot(3,3,8), mesh(gauss1)
% filtered images
subplot(3,3,3), imshow(imgFG01Low), title('gaussian filter std deviation 0.01');
subplot(3,3,6), imshow(imgFG05Low), title('gaussian filter std deviation 0.05');
subplot(3,3,9), imshow(imgFG1Low), title('gaussian filter std deviation 0.1');

```

```

%
% apply filters high noise %
imgFG01High= imfilter(gaussHigh,gauss01);
imgFG05High= imfilter(gaussHigh,gauss05);
imgFG1High= imfilter(gaussHigh,gauss1);
%
% output %
figure
% noisy images
subplot(3,3,1), imshow(gaussHigh)
subplot(3,3,4), imshow(gaussHigh)
subplot(3,3,7), imshow(gaussHigh)
%
% filters mesh
subplot(3,3,2), mesh(gauss01)
subplot(3,3,5), mesh(gauss05)
subplot(3,3,8), mesh(gauss1)
%
% filtered images
subplot(3,3,3), imshow(imgFG01High), title('gaussian filter std deviation 0.01');
subplot(3,3,6), imshow(imgFG05High), title('gaussian filter std deviation 0.05');
subplot(3,3,9), imshow(imgFG1High), title('gaussian filter std deviation 0.1');
%
%% mse esteem filters %%
%
% calculate mse %
mseFG01High = immse(img, imgFG01High);
mseFG05High = immse(img, imgFG05High);
mseFG1High = immse(img, imgFG1High);

mseFG01Low = immse(img, imgFG01Low);
mseFG05Low = immse(img, imgFG05Low);
mseFG1Low = immse(img, imgFG1Low);
%
% calculate peak signal-to-noise ratio
psnrFG01High = psnr(img, imgFG01High);
psnrFG05High = psnr(img, imgFG05High);
psnrFG1High = psnr(img, imgFG1High);

psnrFG01Low = psnr(img, imgFG01Low);
psnrFG05Low = psnr(img, imgFG05Low);
psnrFG1Low = psnr(img, imgFG1Low);
%
% construct table %
LowFilterMse = [mseFG01Low; mseFG05Low; mseFG1Low];
LowFilterPsnr = [psnrFG01Low; psnrFG05Low; psnrFG1Low];
HighFilterMse = [mseFG01High; mseFG05High; mseFG1High];
HighFilterPsnr = [psnrFG01High; psnrFG05High; psnrFG1High];

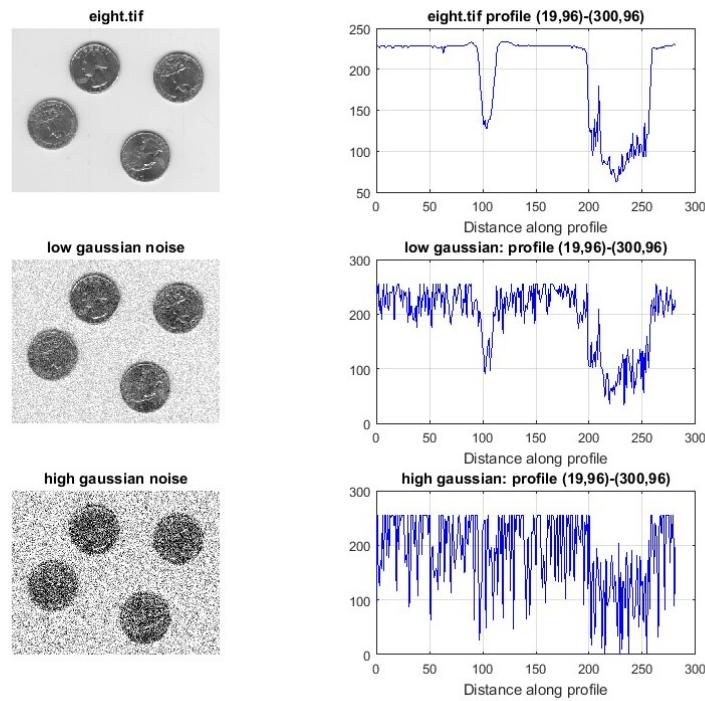
table(LowFilterMse, LowFilterPsnr, HighFilterMse, HighFilterPsnr, 'RowNames',
{'stdv_01', 'stdv_05', 'stdv_1'})
%
%% mse comparison %%
% initialize variables %
LowNoiseMse = [MSE(1); MSE(1); MSE(1)];
LowNoisePsnr = [PSNR(1); PSNR(1); PSNR(1)];

HighNoiseMse = [MSE(2); MSE(2); MSE(2)];
HighNoisePsnr = [PSNR(2); PSNR(2); PSNR(2)];
%
% display table %
LowNoiseTable = table(LowFilterMse, LowNoiseMse, LowFilterPsnr, LowNoisePsnr,
'RowNames', {'stdv_01', 'stdv_05', 'stdv_1'})
HighNoiseTable = table(HighFilterMse, HighNoiseMse, HighFilterPsnr, HighNoisePsnr,
'RowNames', {'stdv_01', 'stdv_05', 'stdv_1'})

```

**Output:**

- *Immagini rumorose e profilo*



*Più l'immagine è rumorosa più il profilo è irregolare e seghettato perché il cambio da un valore ad un altro è più frequente.*

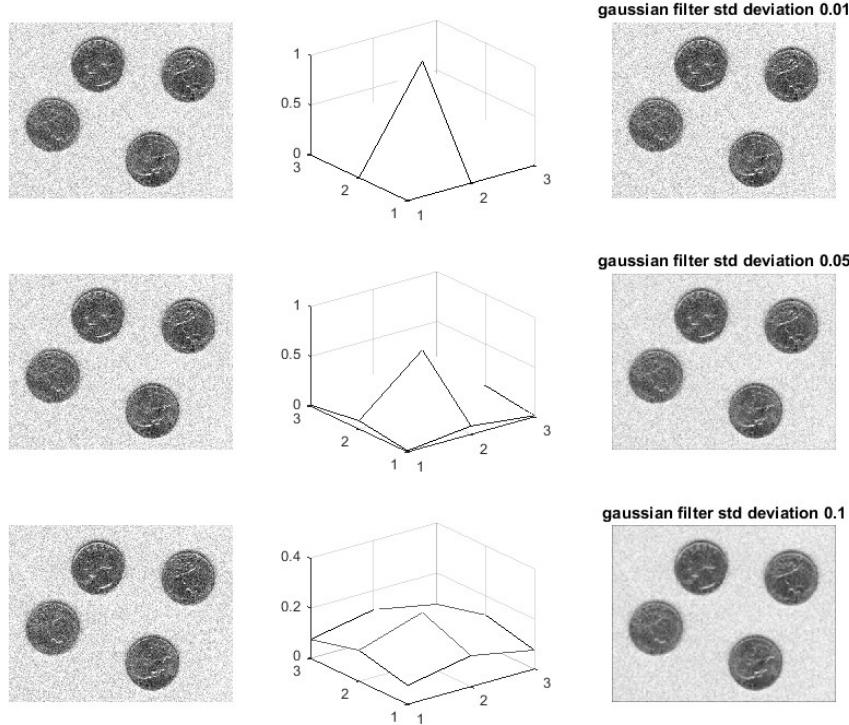
- *Mse e Psnr delle immagini rumorose*

| MSE   | PSNR  |
|-------|-------|
| ————— | ————— |

|           |           |        |
|-----------|-----------|--------|
| NoiseLow  | 0.0082909 | 20.814 |
| NoiseHigh | 0.059439  | 12.259 |

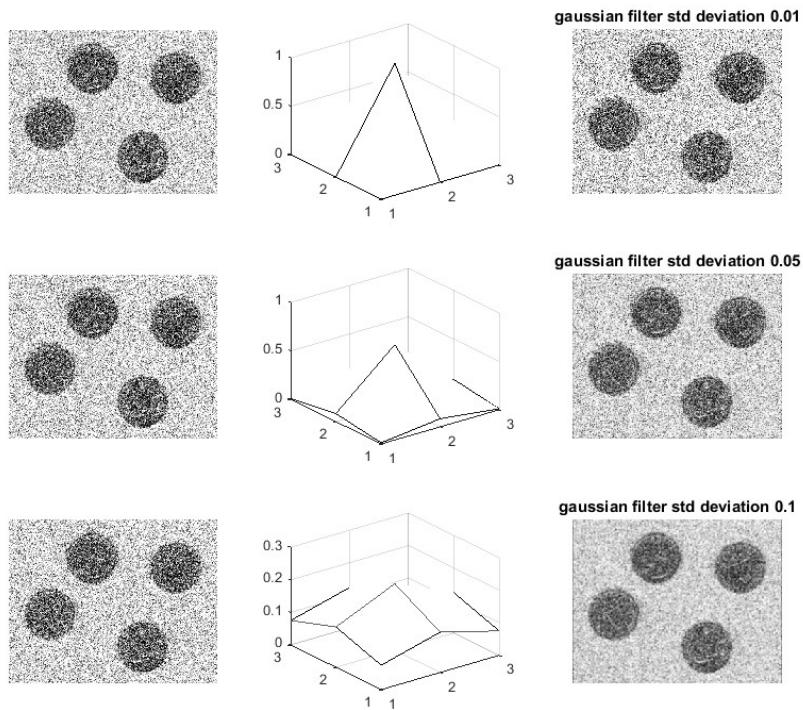
*I due rumori non sono molto alti rispetto a quelli che abbiamo visto nell'esercizio 2.8.*

- *Filtro gaussiano per il rumore basso*



*Il filtro con deviazione standard 0.1 sembra comportarsi molto meglio degli altri due. Il filtro con deviazione standard 0.01 non sembra, invece, essere molto efficace.*

- *Filtro gaussiano per il rumore alto*



*Anche in questo caso il più efficace sembra il filtro con deviazione standard 0.1.*

- *Mse e Psnr dopo il filtraggio*

|         | LowFiltereMse | LowFilterPsnr | HighFilterMse | HighFilterPsnr |
|---------|---------------|---------------|---------------|----------------|
| stdv_01 | 0.0082909     | 20.814        | 0.059439      | 12.259         |
| stdv_05 | 0.0037401     | 24.271        | 0.027888      | 15.546         |
| stdv_1  | 0.0026873     | 25.707        | 0.013679      | 18.639         |

Come si supponeva prima, il filtro con deviazione standard è quello che risulta essere migliore, con il minor MSE e il maggior PSNR per entrambe le immagini rumorose.

- *Mse e Psnr a confronto con l'immagine rumorosa*

- *Immagine poco rumorosa*

|         | LowFiltereMse | LowNoiseMse | LowFilterPsnr | LowNoisePsnr |
|---------|---------------|-------------|---------------|--------------|
| stdv_01 | 0.0082909     | 0.0082909   | 20.814        | 20.814       |
| stdv_05 | 0.0037401     | 0.0082909   | 24.271        | 20.814       |
| stdv_1  | 0.0026873     | 0.0082909   | 25.707        | 20.814       |

- *Immagine molto rumorosa*

|         | HighFilterMse | HighNoiseMse | HighFilterPsnr | HighNoisePsnr |
|---------|---------------|--------------|----------------|---------------|
| stdv_01 | 0.059439      | 0.059439     | 12.259         | 12.259        |
| stdv_05 | 0.027888      | 0.059439     | 15.546         | 12.259        |
| stdv_1  | 0.013679      | 0.059439     | 18.639         | 12.259        |

Come supposto in precedenza, notiamo che il filtro con deviazione standard 0.01 non è per nulla efficace e lascia invariato sia l'errore quadratico medio che il rapporto segnale rumore di picco.

### Osservazioni finali:

In questo esercizio abbiamo lasciato invariata la finestra del filtro e abbiamo lavorato con filtri gaussiani di diversa varianza.

Abbiamo visto come una varianza troppo piccola non ha alcun effetto sull'immagine, mentre sembra che al crescere della varianza aumenti l'efficacia del filtro.

## Esercizio 2.11

E' data l'immagine 'circuit.tif'.

- Aggiungere rumore gaussiano a media nulla e varianza 0.01 (basso) e 0.1 (alto rumore); aggiungere rumore 'sale e pepe' con parametro 0.1 (basso) e 0.4 (alto). Visualizzare l'immagine originale e quelle con rumore;
- Effettuare il filtraggio delle quattro immagini rumorose tramite due filtri: un filtro gaussiano 3x3 con deviazione standard 0.5; un filtro mediano (funzione medfilt2()), anche questo con maschera 3x3;
- Confrontare l'effetto dei due filtri sulle quattro immagini rumorose e riportare i risultati in tabella: quale filtro dà le prestazioni migliori a basso/alto rumore? Riportare le osservazioni nelle conclusioni.

Codice:

- script

```
%%%%%
% Script for the analysis of noise and gaussian
% filtering.
%
% Silvia Gioia Florio, matr. 119328
% 11/04/2017 - Esercizio 2.11
%%%%%
%
% initialize image
img = imread('circuit.tif');
img = double(img)/255;
%
%% noise %%
% initialize noisy images %
% gaussian
gaussLow = imnoise(img, 'gaussian', 0, 0.01);
gaussHigh = imnoise(img, 'gaussian', 0, 0.1);
% salt and pepper
sepLow = imnoise(img, 'salt & pepper', 0.1);
sepHigh = imnoise(img, 'salt & pepper', 0.4);
%
% display results %
figure;
% original
subplot(3,2,1), imshow(img), title('circuit.tif');
% gaussian noise
subplot(3,2,3), imshow(gaussLow), title('gaussian std dev 0.01');
subplot(3,2,4), imshow(gaussHigh), title('gaussian std dev 0.1');
% salt & pepper noise
subplot(3,2,5), imshow(sepLow), title('s&p noise density 0.1');
subplot(3,2,6), imshow(sepHigh), title('s&p noise density 0.4');
%
%% filtering %%
% initialize filters %
% gaussian
gauss = fspecial('gaussian', 3, 0.5);
%
% apply filters %
% median
gaussHighMF = medfilt2(gaussHigh);
gaussLowMF = medfilt2(gaussLow);
sepHighMF = medfilt2(sepHigh);
sepLowMF = medfilt2(sepLow);
% gaussian
```

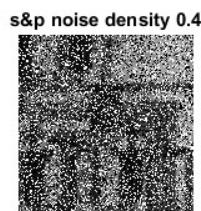
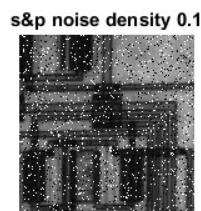
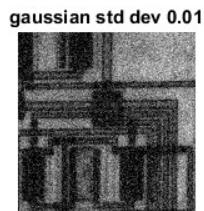
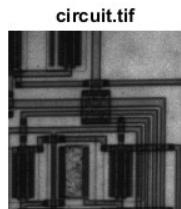
```

gaussHighGF = imfilter(gaussHigh, gauss);
gaussLowGF = imfilter(gaussLow, gauss);
sepHighGF = imfilter(sepHigh, gauss);
sepLowGF = imfilter(sepLow, gauss);
%
% display images %
figure;
% gaussian noisy images
% first column, original
subplot(2,4,1), imshow(img), title('circuit.tif');
subplot(2,4,5), imshow(img)
% noisy
subplot(2,4,2), imshow(gaussLowMF), title('gaussian low');
subplot(2,4,6), imshow(gaussHighMF), title('gaussian high');
% median filters
subplot(2,4,3), imshow(gaussLow), title('median filter');
subplot(2,4,7), imshow(gaussHigh)
% gaussian filters
subplot(2,4,4), imshow(gaussLowGF), title('gaussian filter');
subplot(2,4,8), imshow(gaussHighGF),
%
% salt & pepper noisy images
figure;
% first column, original
subplot(2,4,1), imshow(img), title('circuit.tif');
subplot(2,4,5), imshow(img)
% noisy
subplot(2,4,2), imshow(sepLowMF), title('salt & pepper low');
subplot(2,4,6), imshow(sepHighMF), title('salt & pepper high');
% median filters
subplot(2,4,3), imshow(sepLow), title('median filter');
subplot(2,4,7), imshow(sepHigh)
% gaussian filters
subplot(2,4,4), imshow(sepLowGF), title('gaussian filter');
subplot(2,4,8), imshow(sepHighGF);
%
%% comparison %%
% calculate mse %
% noisy
mseNoiseLowG = immse(img, gaussLow);
mseNoiseHighG = immse(img, gaussHigh);
mseNoiseLowS = immse(img, sepLow);
mseNoiseHighS = immse(img, sepHigh);
% gaussian filter
mseGFLowG = immse(img, gaussLowGF);
mseGFFhighG = immse(img, gaussHighGF);
mseGFLowS = immse(img, sepLowGF);
mseGFFhighS = immse(img, sepHighGF);
% median filter
mseMFLowG = immse(img, gaussLowMF);
mseMFHighG = immse(img, gaussHighMF);
mseMFLowS = immse(img, sepLowMF);
mseMFHighS = immse(img, sepHighMF);
%
% initialize variables %
LowGaussian = [mseNoiseLowG; mseGFLowG; mseMFLowG];
HighGaussian = [mseNoiseHighG; mseGFFhighG; mseMFHighG];
LowSaP = [mseNoiseLowS; mseGFLowS; mseMFLowS];
HighSaP = [mseNoiseHighS; mseGFFhighS; mseMFHighS];
%
% table %
table(LowGaussian, HighGaussian, LowSaP, HighSaP, 'RowNames', {'Noise', 'GaussianFilter', 'MedianFilter'})

```

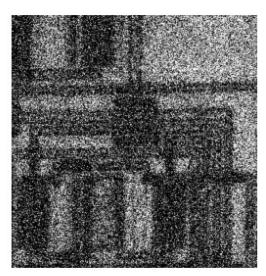
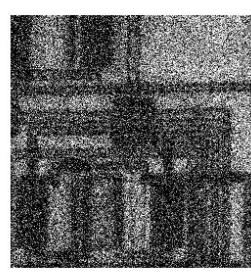
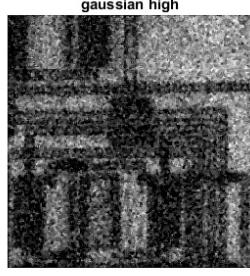
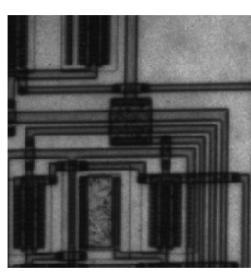
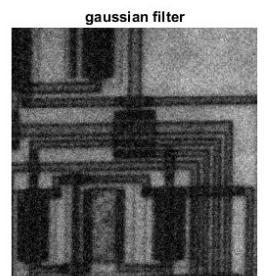
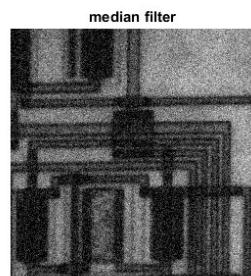
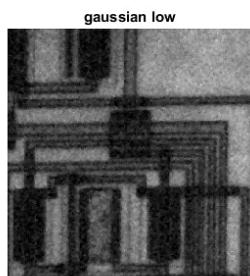
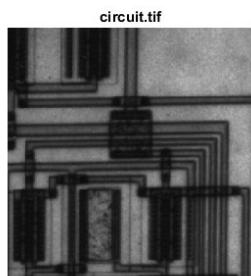
**Output:**

- *Immagini rumorose*



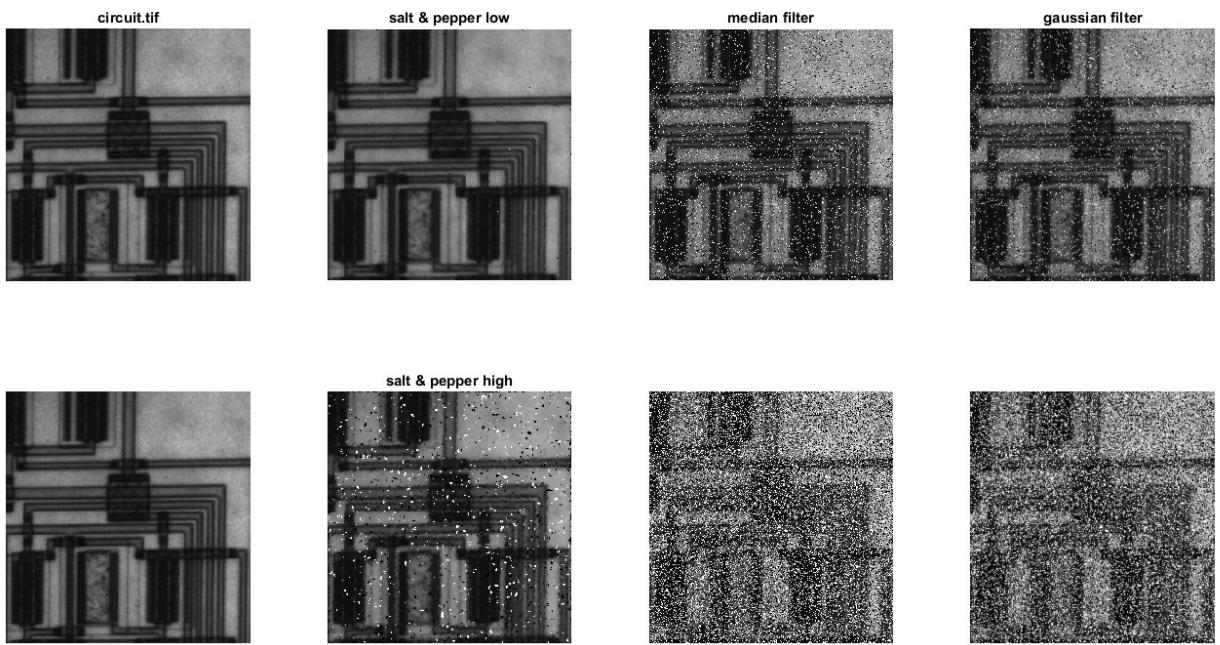
*Il rumore “sale e pepe” modifica di molto il valore del pixel dove agisce, portandolo all'estremo (bianco o nero), mentre il rumore gaussiano è più diffuso, ma meno incisivo.*

- *Filtro gaussiano e mediano sul rumore gaussiano*



*I filtri gaussiani sembrano essere più efficaci sul rumore gaussiano rispetto ai filtri mediani*

- *Filtro gaussiano e mediano sul rumore “sale e pepe”*



*Sembrerebbe il filtro gaussiano ad essere più effi cace anche nel caso di rumore “sale e pepe”.*

- *Mse*

|                | LowGaussian | HighGaussian | LowSaP     | HighSaP  |
|----------------|-------------|--------------|------------|----------|
| Noise          | 0.0091408   | 0.067352     | 0.032682   | 0.13134  |
| GaussianFilter | 0.0038416   | 0.029437     | 0.013999   | 0.061351 |
| MedianFilter   | 0.0019989   | 0.015289     | 0.00039447 | 0.01426  |

*I numeri nella tabella smentiscono le assunzioni fatte in precedenza. Dai dati emerge che il filtro mediano è in realtà più effi cace in entrambi i casi*

### Osservazioni finali:

Nello script di questo esercizio abbiamo visto come si comportano filtri diversi su tipi di rumore diversi e ne è emerso che il filtro mediano è molto efficace, in ogni immagine analizzata si è rivelato più efficace del filtro gaussiano (a varianza 0.5).

## Esercizio 2.12

Generazione di profili di intensità per simulare modelli di edge e filtri in una dimensione.

- Generare un profilo di intensità (intensity profile) che simuli edge di tipo rampa (ramp-edge) e tetto (roof-edge). Generare un secondo profilo di intensità che simuli un edge di tipo gradino (step-edge); visualizzare i due segnali ottenuti;
- Aggiungere rumore gaussiano a media nulla basso/alto ai due profili di intensità ottenuti al punto (a). Il livello basso di rumore deve essere l'1% della massima intensità, il livello alto il 20%. Visualizzare i risultati;
- Generare due profili di filtri unidimensionali, rispettivamente alle somme — maschera [1 1] - e alle differenze, maschera [-1 1]. Visualizzare i risultati dei due segnali ottenuti.

**Codice:**

- script

```
%%%%%%%%%%%%%
% Script for generating intensity profiles.
%
% Silvia Gioia Florio, matr. 119328
% 11/04/2017 - Excercise 2.12
%%%%%%%%%%%%%
%
% ramp and roof %
% initialize sequence
seqRR = [0 0 0 4 8 12 0 0 0 12 0 0 0];
% calling intensity profiling function
nsample = 100;
profileRR = IntProfile(seqRR, nsample);
% step edge %
% initialize sequence
seqSE = [0 0 0 4 4 0 0 0 8 8 0 0 0];
% calling intensity profiling function
nsample = 100;
profileSE = IntProfile(seqSE, nsample);
%
% visualization %
figure;
subplot(2,1,1), plot(profileRR), title('ramp and roof intensity profile'), grid();
subplot(2,1,2), plot(profileSE), title('step-edge intensity profile'), grid();
%
%% noises %%
% calculate noises %
% ramp and roof low noise
noiseRRLow = sigNoise(profileRR, 0.01);
% ramp and roof high noise
noiseRRHigh = sigNoise(profileRR, 0.2);
% step-edge low noise
noiseSELow = sigNoise(profileSE, 0.01);
% step-edge high noise
noiseSEHigh = sigNoise(profileSE, 0.2);
%
% visualization %
figure;
% ramp and roof
subplot(2,2,1), plot(noiseRRLow), title('ramp and roof low noise'), grid();
subplot(2,2,2), plot(noiseRRHigh), title('ramp and roof high noise'), grid();
% step-edge
subplot(2,2,3), plot(noiseSELow), title('step-edge low noise'), grid();
subplot(2,2,4), plot(noiseSEHigh), title('step-edge high noise'), grid();
```

```

%% masks %%
% initialize masks
maskSum = [1 1];
maskDiff = [1 -1];
% initialize filters - profiles
filterProfileRRSum = conv(profileRR, maskSum);
filterProfileRRDiff = conv(profileRR, maskDiff);
filterProfileSESum = conv(profileSE, maskSum);
filterProfileSEDiff = conv(profileSE, maskDiff);
%
% visualize filters %
figure;
subplot(2,2,1), plot(filterProfileRRSum), title('filter ramp and roof sum mask'), grid()
subplot(2,2,2), plot(filterProfileRRDiff), title('filter ramp and roof difference mask'), grid()
subplot(2,2,3), plot(filterProfileSESum), title('filter step-edge sum mask'), grid()
subplot(2,2,4), plot(filterProfileSEDiff), title('filter step-edge difference mask'), grid()

```

- funzione: IntProfile

```

function [intensProfile] = intProfile(seq, nsample)
%%%%%%%%%%%%%
% Function for intensity profiling an image.
%
% Silvia Gioia Florio, matr. 119328
% 11/04/2017 - Esercizio 2.11
%%%%%%%%%%%%%
%
%% Initialization %%
intensProfile = [];
nseq = length(seq);
for i = 1:nseq
 intensProfile = [intensProfile seq(i)*ones(1,nsample)];
end

```

- funzione: sigNoise

```

function noise = sigNoise(signal, variance)
%%%%%%%%%%%%%
% Script for noise addition to a signal.
%
% Silvia Gioia Florio, matr. 119328
% 11/04/2017 - Esercizio 2.11
%%%%%%%%%%%%%
%
%% initialize variables
% size of the input signal
sizeS = size(signal);
% maximum intensity value in the input signal
maxS = max(signal);
if sizeS(2)>1
 maxS = max(maxS);
end
% output
noise = [];

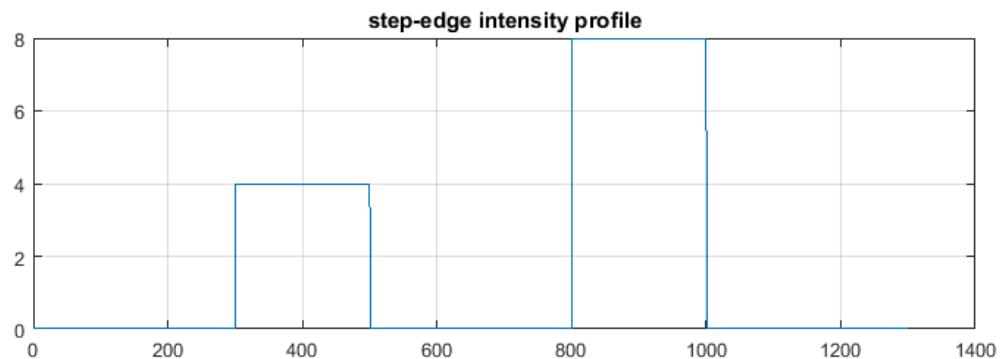
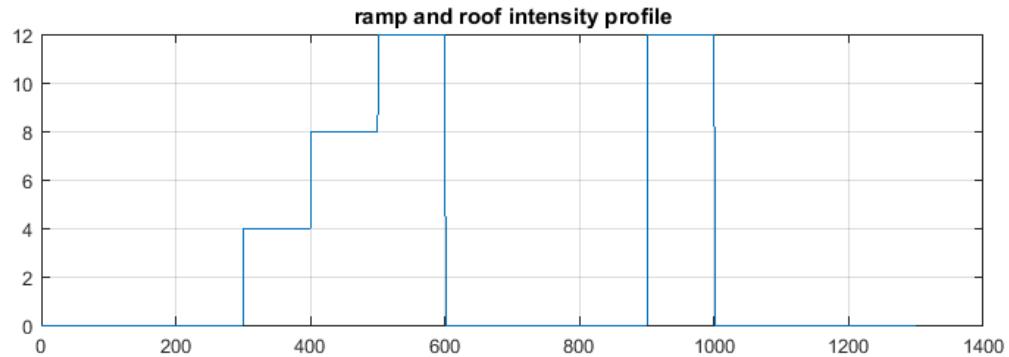
%% calculate variance value
varianceS = maxS*variance;
%% initialize random noise vector
noiseBase = randn(sizeS);
noise = noiseBase*varianceS;

%% calculate noised signal
noise = signal + noise;

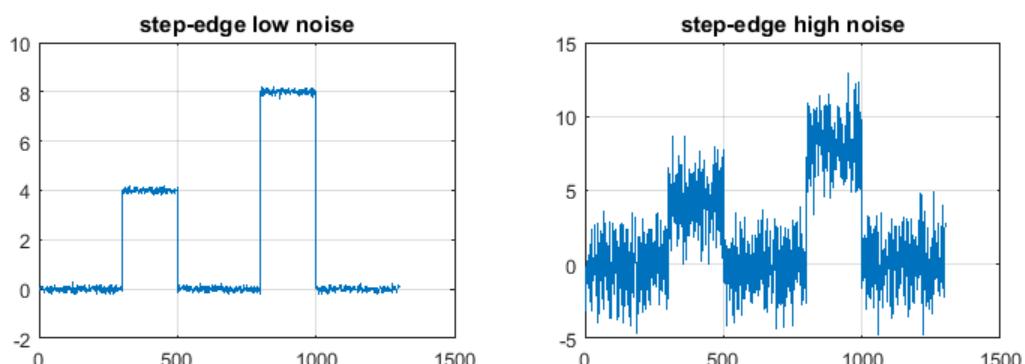
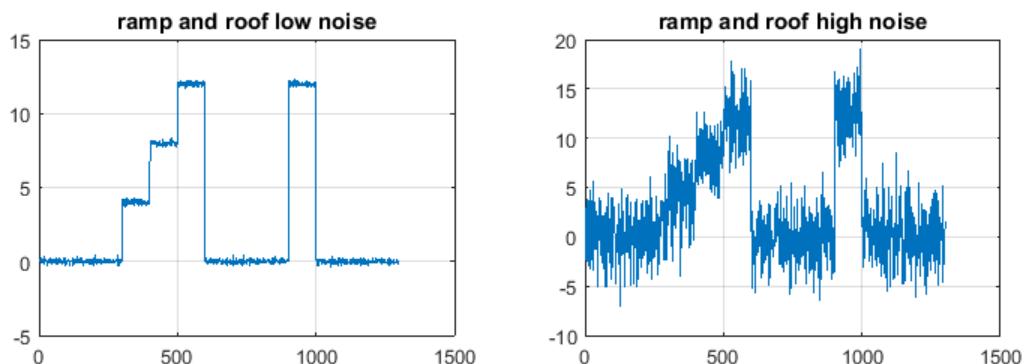
```

**Output:**

- *Ramp and Roof and Step Edge intensity profiles*

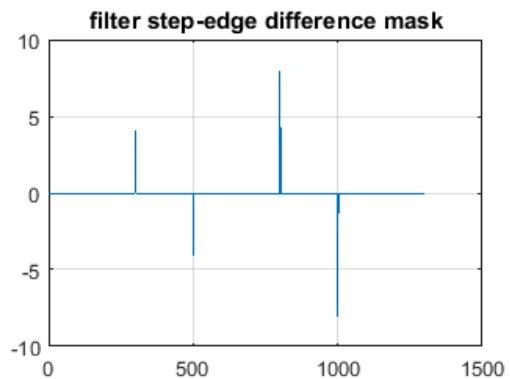
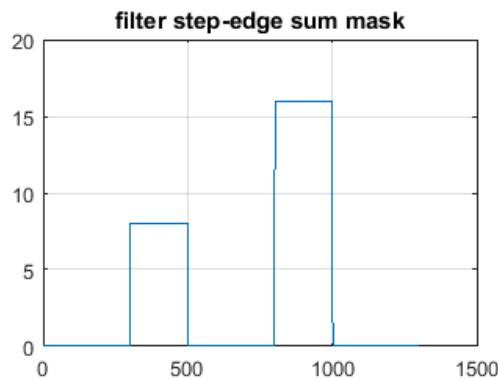
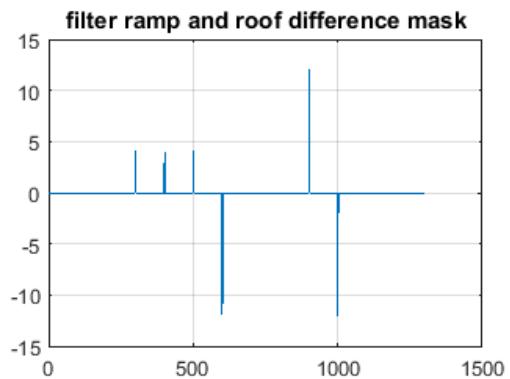
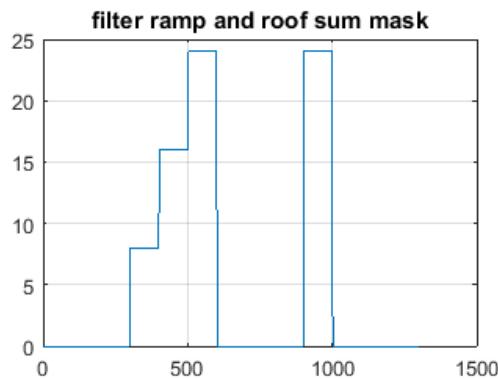


- Aggiunta di rumore basso (varianza 0.01) e alto (varianza 0.2)



*Il rumore alto è molto alto, da rendere quasi impossibile distinguere i gradini.*

- *I profili delle maschere*



### Osservazioni finali:

Nell'esercitazione abbiamo visto qual è il profilo di una maschera alle differenze e confrontata con una maschera alle somme. Abbiamo capito che la maschera alle differenze punta a rilevare i cambi di intensità in un'immagine e può essere utile per il rilevamento automatico di contorni e figure.

## Esercizio 2.13

Analisi della risposta a profili di intensità in una dimensione da parte di filtri alle somme e alle differenze.

- Filtrare i due segnali generati all'esercizio 2.12(a) senza rumore con i filtri alle somme e alle differenze generati al punto 2.12(c). Osservare i risultati e riportare le osservazioni nelle conclusioni;
- Filtrare i segnali con rumore basso/alto con filtri alle somme e alle differenze (punto precedente). Osservare i risultati e riportare le osservazioni;
- Effettuare un doppio filtraggio, applicando ai segnali senza rumore e con rumore filtrati alle differenze, un ulteriore filtro alle differenze. Osservare e discutere i risultati;
- Applicare ai segnali senza rumore e con rumore filtrati alle differenze, un ulteriore filtro alle somme. Osservare e discutere i risultati.

Utilizzare per i filtraggi la procedura MATLAB conv per la convoluzione in una dimensione.

Codice:

- script

```
%%%%%
% Segmentazione analysis of sum and differences filters response to
% one-dimensional intensity profiles
%
% Florio Silvia, mat. 119328
% 27-04-17, excercise 2.13
%%%%%
%
%% intensity profiles %%
% initialization
nsample = 100;
% ramp and roof profile
seqRR = [0 0 0 4 8 12 0 0 0 12 0 0 0];
sigRR = IntProfile(seqRR, nsample);
% step edge profile
seqS = [0 0 4 4 0 0 0 8 8 0 0 0];
sigS = IntProfile(seqS, nsample);
%
% noisy profiles %
% ramp and roof
sigRRLow = sigNoise(sigRR, 0.01);
sigRRHigh = sigNoise(sigRR, 0.2);
% step edge
sigSLow = sigNoise(sigS, 0.01);
sigSHigh = sigNoise(sigS, 0.2);
%
% unidimensional filters initialization
maskSum = [1 1];
maskDiff = [1 -1];
%
% filtering %
% ramp and roof
sigRRSum = conv(sigRR, maskSum);
sigRRDiff = conv(sigRR, maskDiff);
% step edge
sigSSum = conv(sigS, maskSum);
sigSDiff = conv(sigS, maskDiff);
%
% display results %
figure
```

```
% ramp and roof
subplot(2,3,1), plot(sigRR), grid, title('ramp and roof')
subplot(2,3,2), plot(sigRRSum), grid, title('ramp and roof - sum filter')
subplot(2,3,3), plot(sigRDiff), grid, title('ramp and roof - difference filter')
% step edge
subplot(2,3,4), plot(sigS), grid, title('step edge')
subplot(2,3,5), plot(sigSSum), grid, title('step edge - sum filter')
subplot(2,3,6), plot(sigSDiff), grid, title('step edge - difference filter')

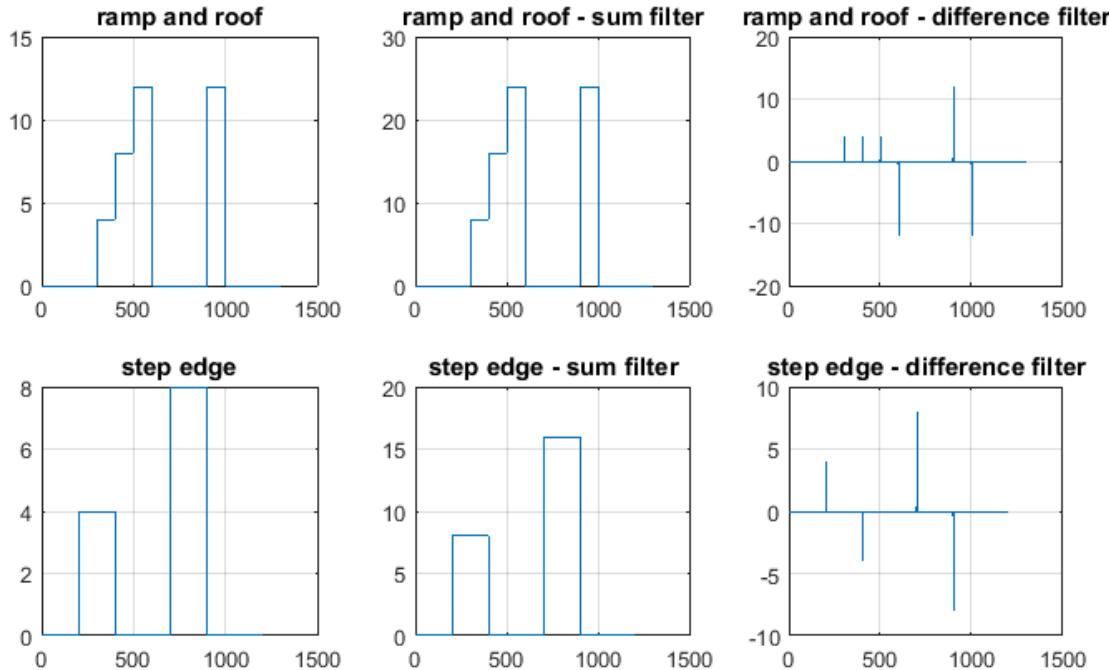
%
%% filtering noisy profiles %%
% ramp and roof
sigRRLowSum = conv(sigRRLow, maskSum);
sigRRHighSum = conv(sigRRHigh, maskSum);
sigRRLowDiff = conv(sigRRLow, maskDiff);
sigRRHighDiff = conv(sigRRHigh, maskDiff);
% step edge
sigSLowSum = conv(sigSlow, maskSum);
sigSHighSum = conv(sigSHigh, maskSum);
sigSLowDiff = conv(sigSlow, maskDiff);
sigSHighDiff = conv(sigSHigh, maskDiff);
%
% display results %
% ramp and roof low noise
figure
subplot(2,3,1), plot(sigRR), grid, title(sprintf('ramp and roof - no noise'))
subplot(2,3,2), plot(sigRRSum), grid, title(sprintf('ramp and roof - no noise\nsum filter'))
subplot(2,3,3), plot(sigRDiff), grid, title(sprintf('ramp and roof - no noise\ndifference filter'))
subplot(2,3,4), plot(sigRRLow), grid, title(sprintf('ramp and roof - low noise'))
subplot(2,3,5), plot(sigRRLowSum), grid, title(sprintf('ramp and roof - low noise\nsum filter'))
subplot(2,3,6), plot(sigRRLowDiff), grid, title(sprintf('ramp and roof - low noise\ndifference filter'))
% ramp and roof high noise
figure
subplot(2,3,1), plot(sigRR), grid, title(sprintf('ramp and roof - no noise'))
subplot(2,3,2), plot(sigRRSum), grid, title(sprintf('ramp and roof - no noise\nsum filter'))
subplot(2,3,3), plot(sigRDiff), grid, title(sprintf('ramp and roof - no noise\ndifference filter'))
subplot(2,3,4), plot(sigRRHigh), grid, title(sprintf('ramp and roof - high noise'))
subplot(2,3,5), plot(sigRRHighSum), grid, title(sprintf('ramp and roof - high noise\nsum filter'))
subplot(2,3,6), plot(sigRRHighDiff), grid, title(sprintf('ramp and roof - high noise\ndifference filter'))
% step edge
figure
subplot(2,3,1), plot(sigS), grid, title(sprintf('step edge - no noise'))
subplot(2,3,2), plot(sigSSum), grid, title(sprintf('step edge - no noise\nsum filter'))
subplot(2,3,3), plot(sigSDiff), grid, title(sprintf('step edge - no noise\ndifference filter'))
subplot(2,3,4), plot(sigSlow), grid, title(sprintf('step edge - low noise'))
subplot(2,3,5), plot(sigSlowSum), grid, title(sprintf('step edge - low noise\nsum filter'))
subplot(2,3,6), plot(sigSlowDiff), grid, title(sprintf('step edge - low noise\ndifference filter'))

figure
subplot(2,3,1), plot(sigS), grid, title(sprintf('step edge - no noise'))
subplot(2,3,2), plot(sigSSum), grid, title(sprintf('step edge - no noise\nsum filter'))
subplot(2,3,3), plot(sigSDiff), grid, title(sprintf('step edge - no noise\ndifference filter'))
subplot(2,3,4), plot(sigSHigh), grid, title(sprintf('step edge - high noise'))
subplot(2,3,5), plot(sigSHighSum), grid, title(sprintf('step edge - high noise\nsum filter'))
subplot(2,3,6), plot(sigSHighDiff), grid, title(sprintf('step edge - high noise\ndifference filter'))
```

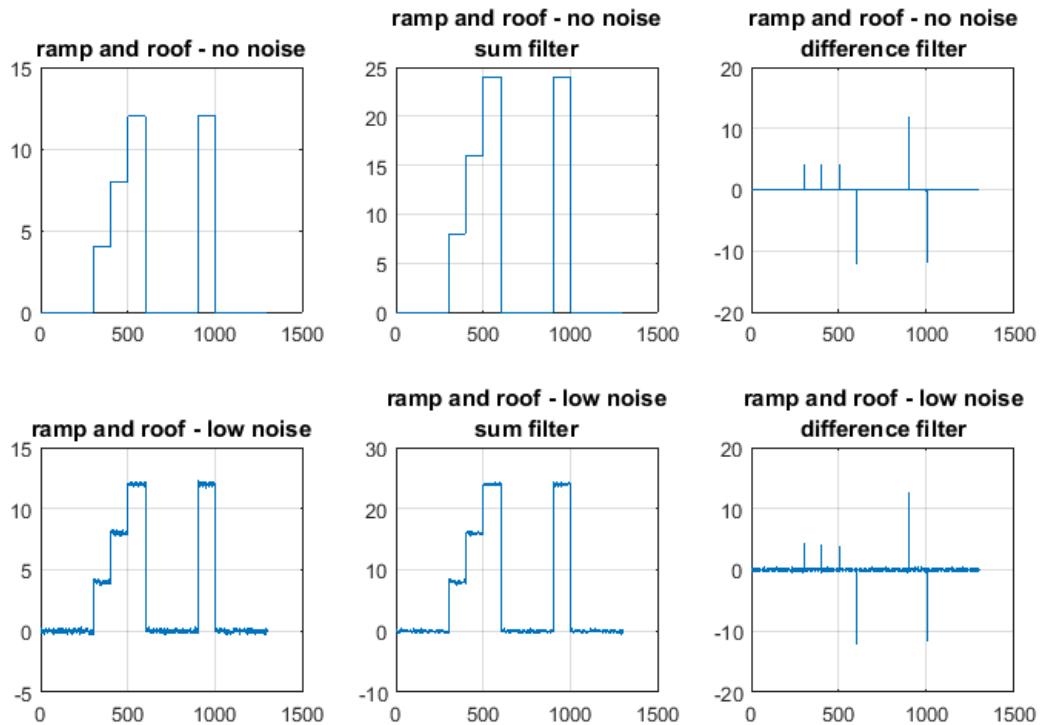
```
%
%% double difference filtering %%
% no noise %
sigRRDoubleDiff = conv(sigRRDiff, maskDiff);
sigSDoubleDiff = conv(sigSDiff, maskDiff);
%
% display results %
figure
% ramp and roof
subplot(2,2,1), plot(sigRR), grid, title(sprintf('ramp and roof - no noise'))
subplot(2,2,2), plot(sigRRDoubleDiff), grid, title(sprintf('ramp and roof - no
noise\ndouble difference filter'))
% step edge
subplot(2,2,3), plot(sigS), grid, title(sprintf('step edge - no noise'))
subplot(2,2,4), plot(sigSDoubleDiff), grid, title(sprintf('step edge - no noise\ndouble
difference filter'))
%
% noisy profiles %
% ramp and roof
sigRRLowDoubleDiff = conv(sigRRLowDiff, maskDiff);
sigRRHighDoubleDiff = conv(sigRRHighDiff, maskDiff);
% step edge
sigSLowDoubleDiff = conv(sigSLowDiff, maskDiff);
sigSHighDoubleDiff = conv(sigSHighDiff, maskDiff);
%
% display results %
figure
% ramp and roof
subplot(2,2,1), plot(sigRRLowDoubleDiff), grid, title(sprintf('ramp and roof - low
noise\ndouble difference filter'))
subplot(2,2,2), plot(sigRRHighDoubleDiff), grid, title(sprintf('ramp and roof - high
noise\ndouble difference filter'))
% step edge
subplot(2,2,3), plot(sigSLowDoubleDiff), grid, title(sprintf('step edge - low
noise\ndouble difference filter'))
subplot(2,2,4), plot(sigSHighDoubleDiff), grid, title(sprintf('step edge - high
noise\ndouble difference filter'))
```

**Output:**

- Visualizzazione dei profili dei filtri

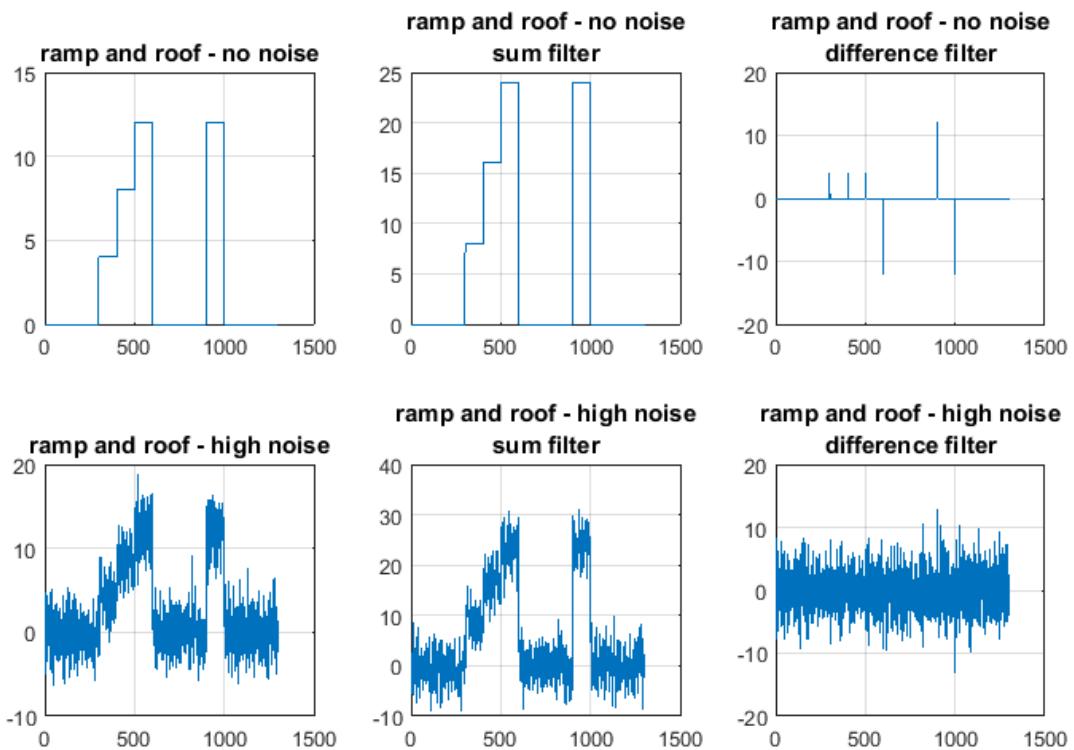


- Ramp and Roof, rumore basso, filtro alle somme e alle differenze



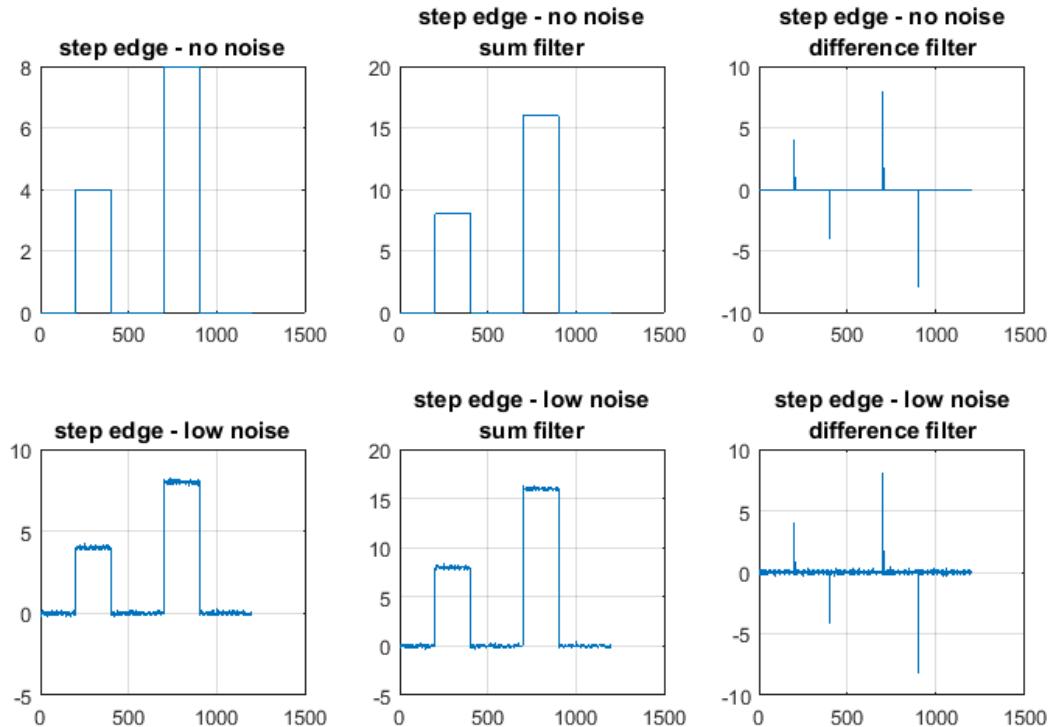
Si nota come un filtro alle somme miри a diminuire il rumore, mentre un filtro alle differenze miри ad individuare i cambi di intensità.

- Ramp and Roof, rumore alto, filtro alle somme e alle differenze



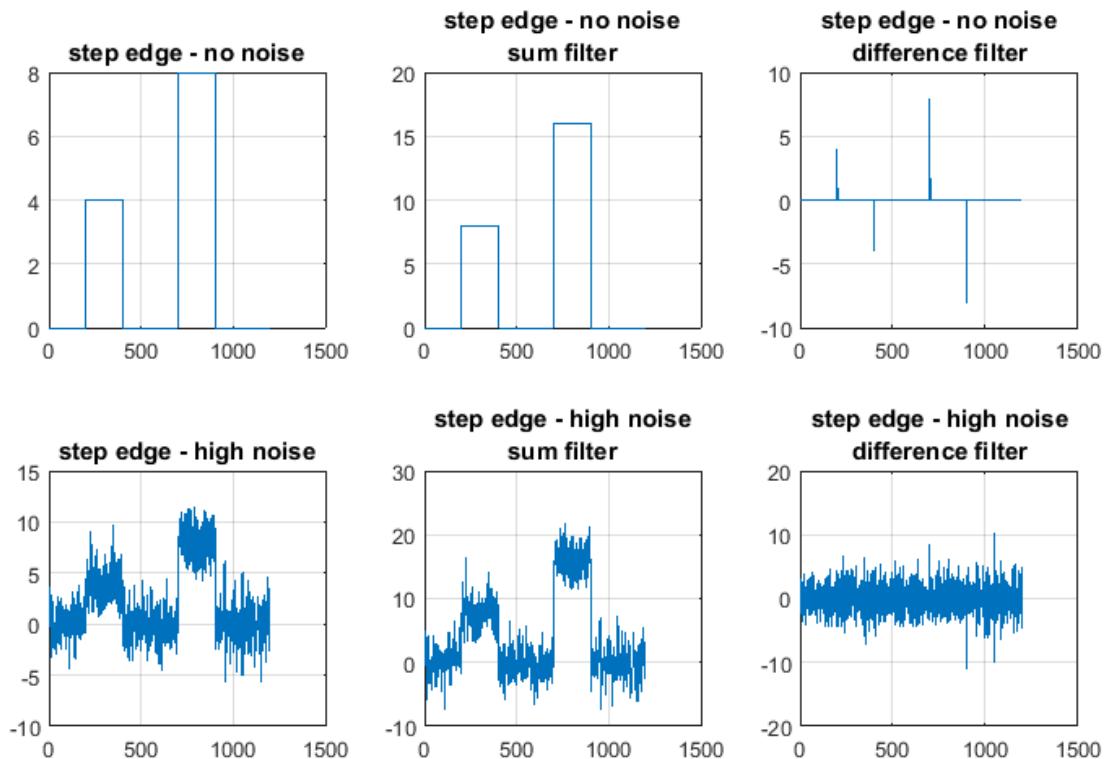
Se il rumore è tanto alto, purtroppo, il filtro alle differenze non è molto efficace.

- Step Edge, rumore basso, filtro alle somme e alle differenze



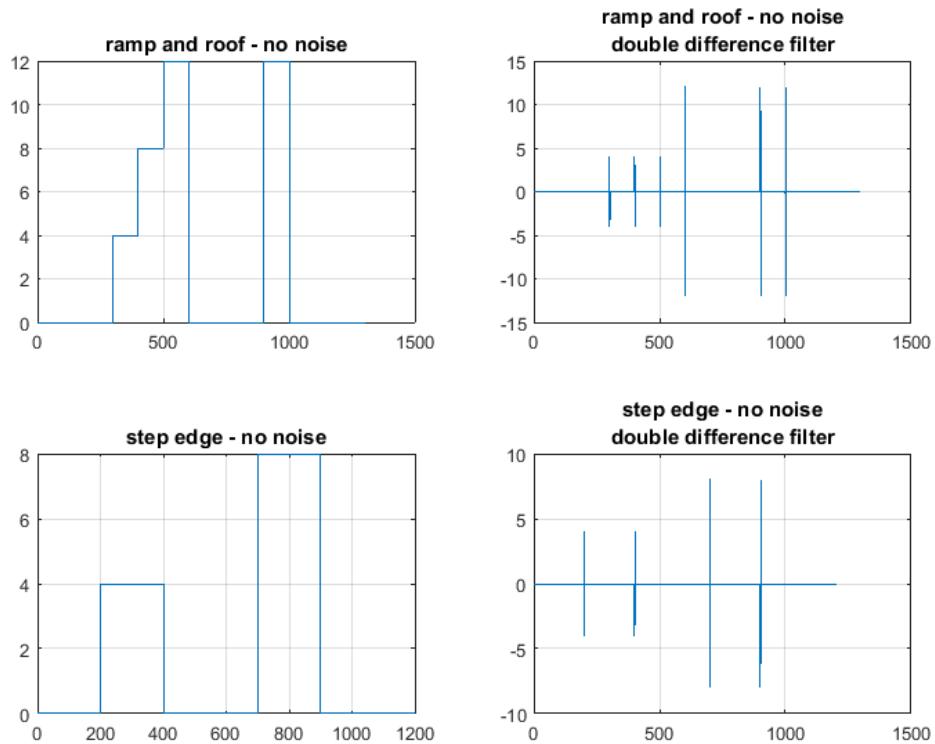
Anche in questo caso il filtro alle differenze riesce ad individuare facilmente i cambi di intensità.

- Step Edge, rumore alto, filtro alle somme e alle differenze



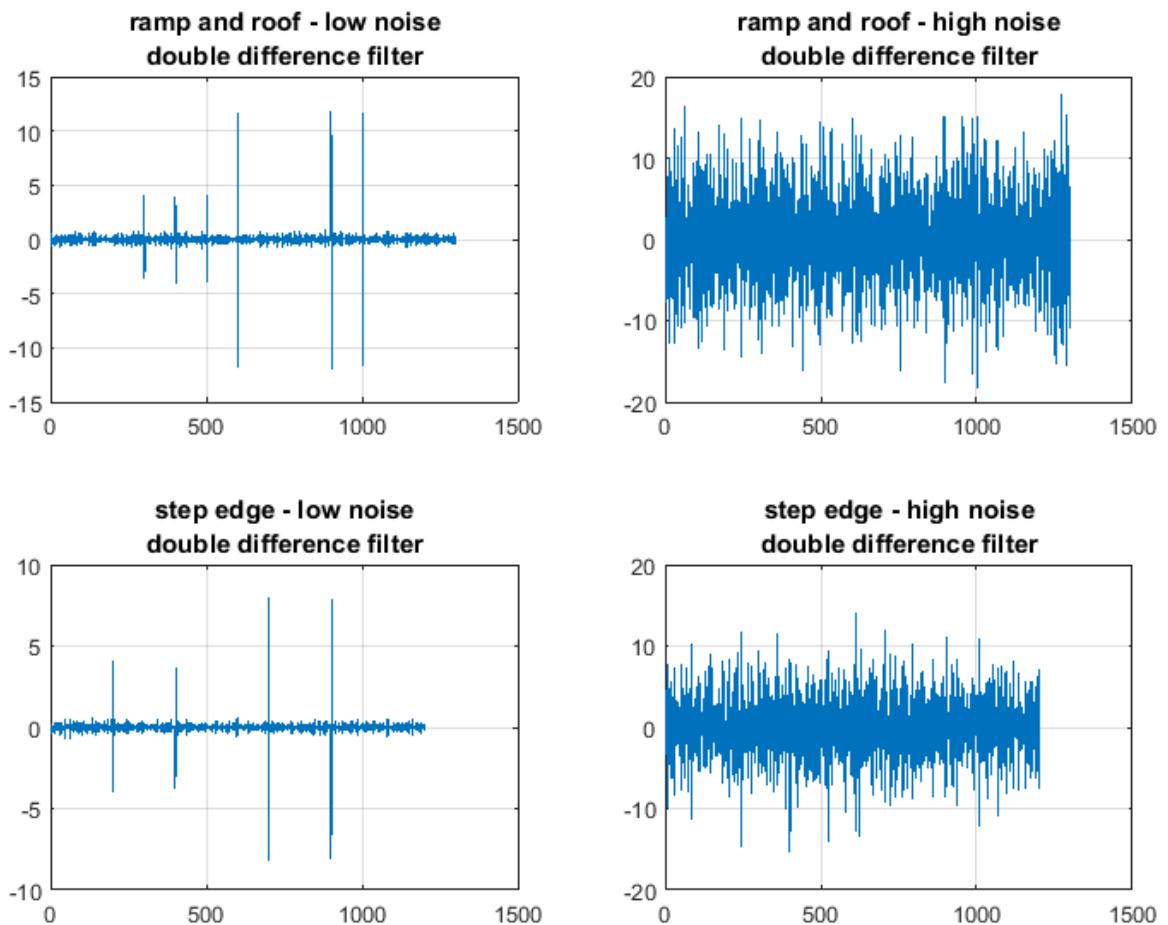
Come prima, l'alto grado di rumore impedisce al filtro di individuare i cambi di intensità.

- Visualizzazione profili dei filtri a doppia differenza



La maschera del filtro a doppia differenza evidenzia gli edge con una linea verticale che si estende sia in alto che in basso (non solo in un lato, come la singola differenza).

- *Applicazione dei filtri a doppia differenza*



*Il filtro a doppia differenza enfatizza ancora di più il cambio di intensità, ma ancora non basta per rilevare le variazioni con un rumore di varianza 0.2*

#### Osservazioni finali:

Nello script abbiamo visto come si comportano le maschere alle somme e alle differenze su input rumorosi. Abbiamo capito che la maschera alle somme punta ad un miglioramento dell'immagine cercando di ridurre il rumore presente nell'immagine, mentre quella alle differenze, come già detto, punta a rilevare i cambi di intensità all'interno del profilo.

C'è da notare che nel rumore molto alto, mentre la maschera alle somme continua a funzionare discretamente, la maschera alle differenze non riesce più ad avere effetto.

## Esercizio 2.14

Considerare l'immagine sintetica della barra.

- Calcolare la mappa delle ampiezze e delle direzioni di edge sull'immagine senza rumore effettuando un filtraggio derivativo tipo gradiente di Roberts e di Sobel. Osservare i risultati e riportare le osservazioni nelle conclusioni;
- Sull'immagine della barra con rumore gaussiano additivo con deviazione standard  $\text{stdev}=0.1$ , ripetere il calcolo delle mappe di edge con filtri di Roberts e Sobel. Riportare le osservazioni nelle conclusioni;
- Considerare l'immagine ‘blobs.png’ a cui aggiungere rumore gaussiano additivo con  $\text{stdev}=0.1$ . Effettuare il calcolo al punto (b) dei gradienti di Roberts e Sobel — soltanto le ampiezze di edge. Riportare le osservazioni;
- Considerare l'immagine ‘circuit.tif’. Calcolare la mappa delle ampiezze e delle direzioni degli edge tramite gradiente di Sobel operando sulla ‘circuit.tif’ senza aggiunta di rumore;
- Considerare l'immagine ‘circuit.tif’. Effettuare il calcolo delle ampiezze di edge con i gradienti di Roberts e Sobel. Sulle immagini delle ampiezze effettuare una sogliatura tramite la `im2bw()` ; visualizzare le mappe sogliate (binarizzate) affiancate usando la `imshowpair(..., ..., 'montage')`;

Per il filtraggio di Roberts si deve definire le maschere e usare la sola `imfilter()` di MATLAB; per il filtraggio di Sobel si deve usare le procedure `fspecial()` e `imfilter()`. Per visualizzare le mappe delle direzioni degli edge candidati usare la funzione `quiver()`. Non si devono usare le funzioni `imgradient()` e `edge()` di MATLAB.

Codice:

- script

```
%%%%%
% Script for the analysis of directions of edges
% on Sobel and Roberts filters.
%
% Silvia Gioia Florio, matr. 119328
% 11/04/2017 - Excercise 2.14
%%%%%
%
%% Roberts and Sobel filters %%
% initialize synthetic image %
% initialize black image
img = zeros(30,30);
% create a white rectangle
img(5:24, 13:17)=1.;
%
% initialize Roberts and Sobert matrixes %
% roberts
robX = [-1,0;0,1];
robY = [0,-1;1,0];
% sobel
sobX = fspecial('sobel');
sobY = transpose(sobX);
% apply filters %
% roberts
imgFRX = imfilter(img,robX);
imgFRY = imfilter(img,robY);
% sobel
imgFSX = imfilter(img, sobX);
```

```

imgFSY = imfilter(img, sobY);
%
% display results %
% roberts
figure
subplot(2,3,1), imshow(img), title('Original Image')
subplot(2,3,2), imshow(imgFRY, 'InitialMagnification', 'fit'), title('Roberts Filter First Derivative')
subplot(2,3,3), imshow(imgFRX, 'InitialMagnification', 'fit'), title('Roberts Filter Second Derivative')
subplot(2,3,5), mesh(imgFRY), title('Roberts First Derivative Mesh')
subplot(2,3,6), mesh(imgFRX), title('Roberts Second Derivative Mesh')
%sobel
figure
subplot(2,3,1), imshow(img), title('Original Image')
subplot(2,3,2), imshow(imgFSX, 'InitialMagnification', 'fit'), title('Sobel Filter First Derivative')
subplot(2,3,3), imshow(imgFSY, 'InitialMagnification', 'fit'), title('Sobel Filter Second Derivative')
subplot(2,3,5), mesh(imgFSX), title('Sobel First Derivative Mesh')
subplot(2,3,6), mesh(imgFSY), title('Sobel Second Derivative Mesh')
%
% amplitudes and directions map %
% roberts
ampRob = abs(imgFRX) + abs(imgFRY);
% sobel
ampSob = abs(imgFSX) + abs(imgFSY);
%
% display result %
figure
subplot(1,3,1), imshow(ampRob), title('Roberts Amplitudes Map')
subplot(1,3,2), mesh(ampRob), title('Roberts Amplitudes Map Mesh')
subplot(1,3,3), quiver(imgFRX, imgFRY), title('Roberts Directions Map')
figure
subplot(1,3,1), imshow(ampSob), title('Sobel Amplitudes Map')
subplot(1,3,2), mesh(ampSob), title('Sobel Amplitudes Map Mesh')
subplot(1,3,3), quiver(imgFSX, imgFSY), title('Sobel Directions Map')
%
%% experiment %%
% diagonal figure %
imgDia = zeros(30,30);
for i=5:20
 for j=0:5
 imgDia(i+j, i+1) = 1;
 end
end
%
% apply filters %
% roberts
imgDiaFRX = imfilter(imgDia, robX);
imgDiaFRY = imfilter(imgDia, robY);
% sobel
imgDiaFSX = imfilter(imgDia, sobX);
imgDiaFSY = imfilter(imgDia, sobY);
%
% display results %
% roberts
figure
subplot(2,3,1), imshow(imgDia), title('Original Image')
subplot(2,3,2), imshow(imgDiaFRY, 'InitialMagnification', 'fit'), title('Roberts Filter First Derivative')
subplot(2,3,3), imshow(imgDiaFRX, 'InitialMagnification', 'fit'), title('Roberts Filter Second Derivative')
subplot(2,3,5), mesh(imgDiaFRY), title('Roberts First Derivative Mesh')
subplot(2,3,6), mesh(imgDiaFRX), title('Roberts Second Derivative Mesh')
%sobel
figure
subplot(2,3,1), imshow(imgDia), title('Original Image')
subplot(2,3,2), imshow(imgDiaFSX, 'InitialMagnification', 'fit'), title('Sobel Filter First Derivative')

```

```

subplot(2,3,3), imshow(imgDiaFSY, 'InitialMagnification', 'fit'), title('Sobel Filter
Second Derivative')
subplot(2,3,5), mesh(imgDiaFSX), title('Sobel First Derivative Mesh')
subplot(2,3,6), mesh(imgDiaFSY), title('Sobel Second Derivative Mesh')
%
% amplitudes and directions map %
% roberts
ampRobDia = abs(imgDiaFRX) + abs(imgDiaFRY);
% sobel
ampSobDia = abs(imgDiaFSX) + abs(imgDiaFSY);
%
% display result %
figure
subplot(1,3,1), imshow(ampRobDia), title('Roberts Amplitudes Map')
subplot(1,3,2), mesh(ampRobDia), title('Roberts Amplitudes Map Mesh')
subplot(1,3,3), quiver(imgDiaFRX, imgDiaFRY), title('Roberts Directions Map')
figure
subplot(1,3,1), imshow(ampSobDia), title('Sobel Amplitudes Map')
subplot(1,3,2), mesh(ampSobDia), title('Sobel Amplitudes Map Mesh')
subplot(1,3,3), quiver(imgDiaFSX, imgDiaFSY), title('Sobel Directions Map')
%
%% noisy bar %%
% initialization %
imgNoise = sigNoise(img, 0.1);
% apply filters %
% roberts
imgNFRX = imfilter(imgNoise, robX);
imgNFRY = imfilter(imgNoise, robY);
% sobel
imgNFSX = imfilter(imgNoise, sobX);
imgNFSY = imfilter(imgNoise, sobY);
%
% display results %
% roberts
figure
subplot(2,3,1), imshow(imgNoise), title('Noisy Image')
subplot(2,3,2), imshow(imgNFRY, 'InitialMagnification', 'fit'), title('Roberts Filter
First Derivative')
subplot(2,3,3), imshow(imgNFRX, 'InitialMagnification', 'fit'), title('Roberts Filter
Second Derivative')
subplot(2,3,5), mesh(imgNFRY), title('Roberts First Derivative Mesh')
subplot(2,3,6), mesh(imgNFRX), title('Roberts Second Derivative Mesh')
%sobel
figure
subplot(2,3,1), imshow(imgNoise), title('Noisy Image')
subplot(2,3,2), imshow(imgNFSX, 'InitialMagnification', 'fit'), title('Sobel Filter First
Derivative')
subplot(2,3,3), imshow(imgNFSY, 'InitialMagnification', 'fit'), title('Sobel Filter
Second Derivative')
subplot(2,3,5), mesh(imgNFSX), title('Sobel First Derivative Mesh')
subplot(2,3,6), mesh(imgNFSY), title('Sobel Second Derivative Mesh')
%
% amplitudes and directions map %
% roberts
ampNRob = abs(imgNFRX) + abs(imgNFRY);
% sobel
ampNSob = abs(imgNFSX) + abs(imgNFSY);
%
% display result %
figure
subplot(1,3,1), imshow(ampNRob), title('Roberts Amplitudes Map')
subplot(1,3,2), mesh(ampNRob), title('Roberts Amplitudes Map Mesh')
subplot(1,3,3), quiver(imgNFRX, imgNFRY), title('Roberts Directions Map')
figure
subplot(1,3,1), imshow(ampNSob), title('Sobel Amplitudes Map')
subplot(1,3,2), mesh(ampNSob), title('Sobel Amplitudes Map Mesh')
subplot(1,3,3), quiver(imgNFSX, imgNFSY), title('Sobel Directions Map')
%
%% blobs %%
% initialization %

```

```

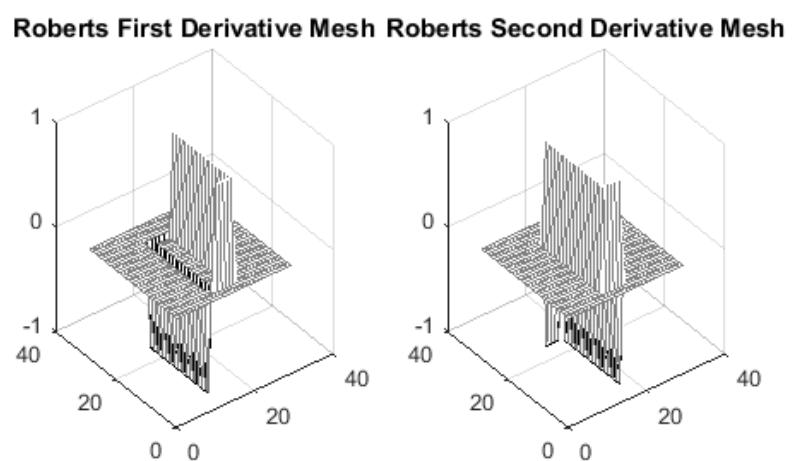
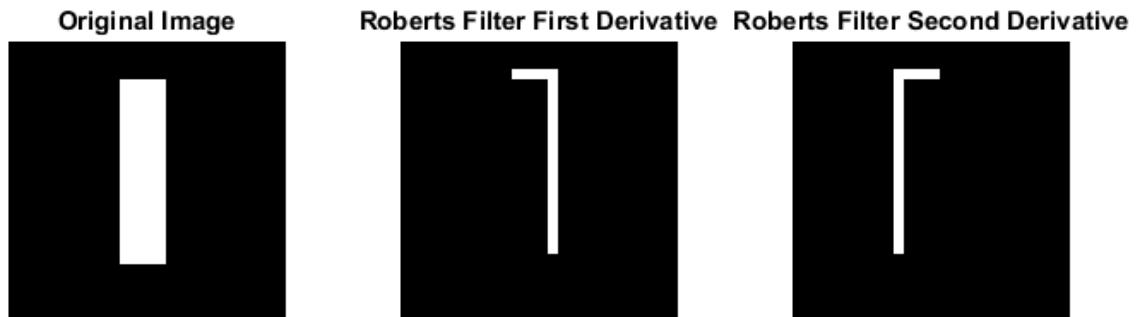
blobs = imread('blobs.png');
blobsNoise = sigNoise(blobs, 0.1);
% apply filters %
% roberts
blobsNoiseFRX = imfilter(blobsNoise, robX);
blobsNoiseFRY = imfilter(blobsNoise, robY);
% sobel
blobsNoiseFSX = imfilter(blobsNoise, sobX);
blobsNoiseFSY = imfilter(blobsNoise, sobY);
%
% amplitudes and directions map %
% roberts
ampNRob = abs(blobsNoiseFRX) + abs(blobsNoiseFRY);
% sobel
ampNSob = abs(blobsNoiseFSX) + abs(blobsNoiseFSY);
%
% display result %
figure
subplot(2,3,1), imshow(blobs, 'InitialMagnification', 'fit'), title('Original Image')
subplot(2,3,4), imshow(blobsNoise, 'InitialMagnification', 'fit'), title('Noisy Image')
subplot(2,3,2), imshow(ampNRob), title('Roberts Amplitudes Map')
subplot(2,3,5), mesh(ampNRob), title('Roberts Amplitudes Map Mesh')
subplot(2,3,3), imshow(ampNSob), title('Sobel Amplitudes Map')
subplot(2,3,6), mesh(ampNSob), title('Sobel Amplitudes Map Mesh')
%
%% circuit %%
circuit = imread('circuit.tif');
% apply filters %
% roberts
circuitFRX = imfilter(circuit, robX);
circuitFRY = imfilter(circuit, robY);
% sobel
circuitFSX = imfilter(circuit, sobX);
circuitFSY = imfilter(circuit, sobY);
%
% amplitudes and directions map %
% roberts
ampRobCircuit = abs(circuitFRX) + abs(circuitFRY);
% sobel
ampSobCircuit = abs(circuitFSX) + abs(circuitFSY);
%
% display result %
figure
subplot(2,2,1), imshow(circuit, 'InitialMagnification', 'fit'), title('Original Image')
subplot(2,2,2), imshow(ampRobCircuit), title('Roberts Amplitudes Map')
subplot(2,2,4), mesh(ampRobCircuit), title('Roberts Amplitudes Map Mesh')
subplot(2,2,3), quiver(circuitFRX, circuitFRY), title('Roberts Directions Map')
figure
subplot(2,2,1), imshow(circuit, 'InitialMagnification', 'fit'), title('Original Image')
subplot(2,2,2), imshow(ampSobCircuit), title('Sobel Amplitudes Map')
subplot(2,2,4), mesh(ampSobCircuit), title('Sobel Amplitudes Map Mesh')
subplot(2,2,3), quiver(circuitFSX, circuitFSY), title('Sobel Directions Map')
%
% focus on directions maps %
figure
subplot(1,2,1), quiver(circuitFRX, circuitFRY), title('Roberts Directions Map')
subplot(1,2,2), quiver(circuitFSX, circuitFSY), title('Sobel Directions Map')
%
%% binarization %%
% display amplitudes maps %
figure
subplot(2,3,1), imshow(circuit, 'InitialMagnification', 'fit'), title('Original Image')
subplot(2,3,2), imshow(ampRobCircuit), title('Roberts Amplitudes Map')
subplot(2,3,5), mesh(ampRobCircuit), title('Roberts Amplitudes Map Mesh')
subplot(2,3,3), imshow(ampSobCircuit), title('Sobel Amplitudes Map')
subplot(2,3,6), mesh(ampSobCircuit), title('Sobel Amplitudes Map Mesh')
%
% binarization %
ampRobBin = im2bw(ampRobCircuit, .12);
ampSobBin = im2bw(ampSobCircuit,.4);

```

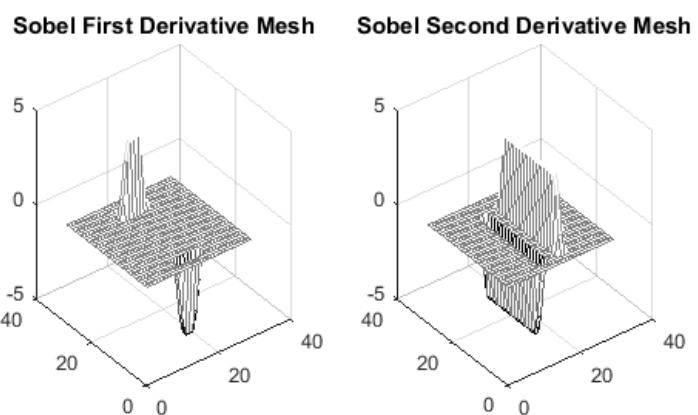
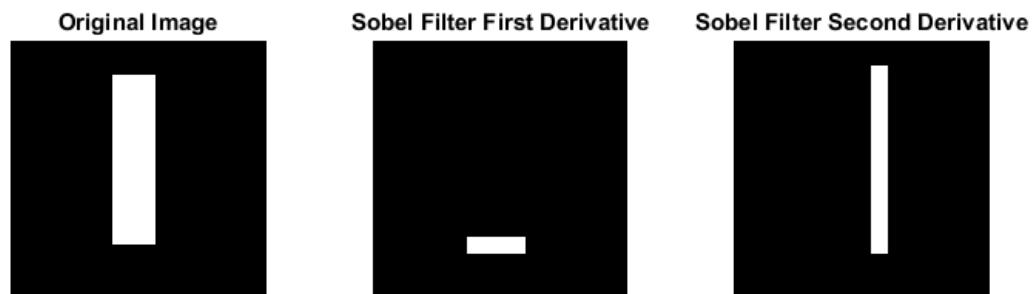
```
%
% display results %
figure
subplot(2,1,1), imshowpair(ampRobCircuit, ampRobBin, 'montage'), title(sprintf('Roberts
Aplitudes Maps\nThreshold 0.12'))
subplot(2,1,2), imshowpair(ampSobCircuit, ampSobBin, 'montage'), title(sprintf('Sobel
Aplitudes Maps\nThreshold 0.4'))
```

**Output:**

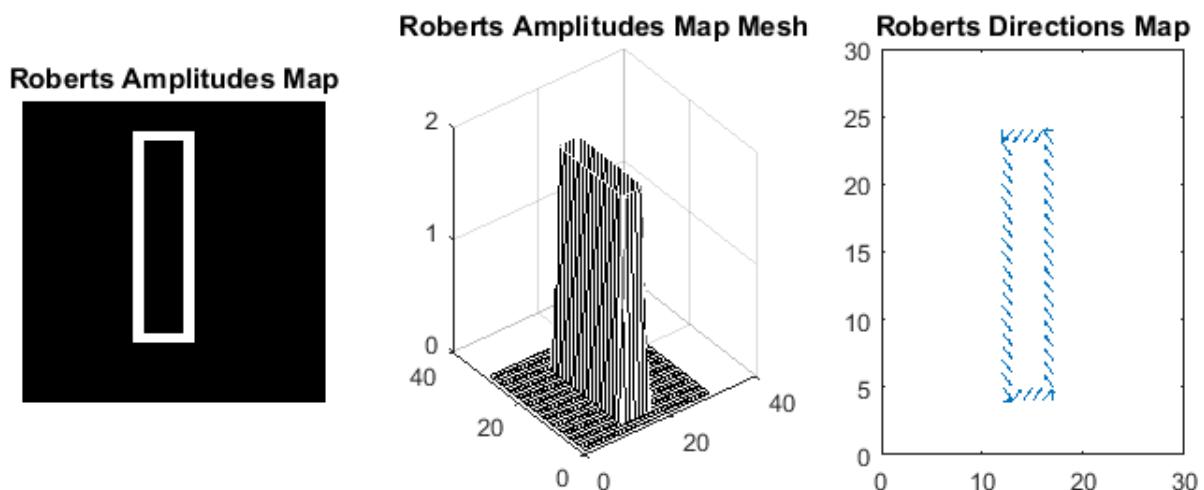
- Rettangolo Bianco - derivata prima e seconda del filtro di Roberts



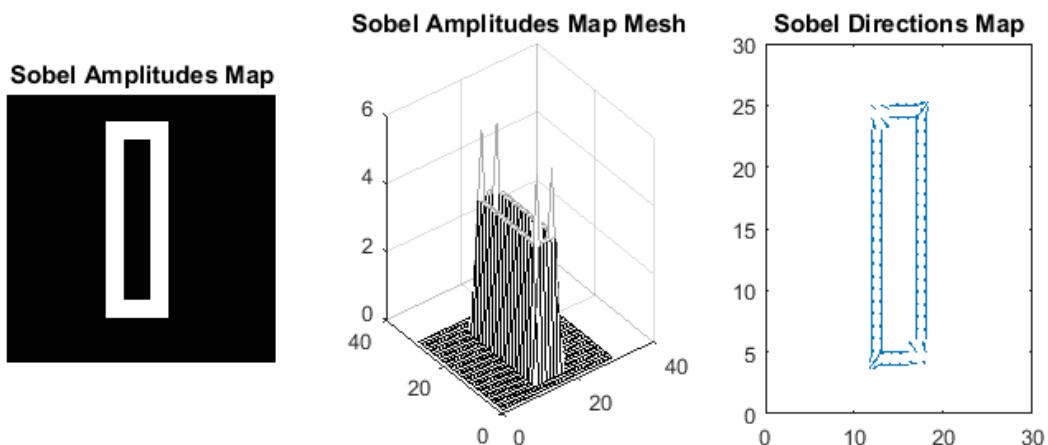
- Rettangolo bianco - derivata prima e seconda del filtro di Sobel



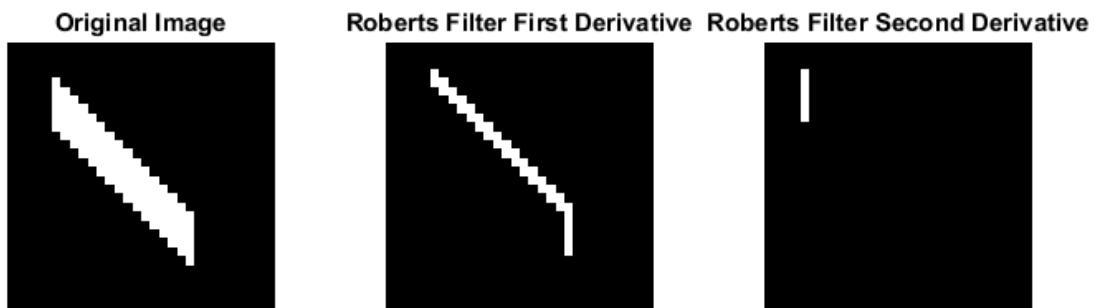
- Rettangolo bianco - mappa delle ampiezze e delle direzioni – filtro di Roberts



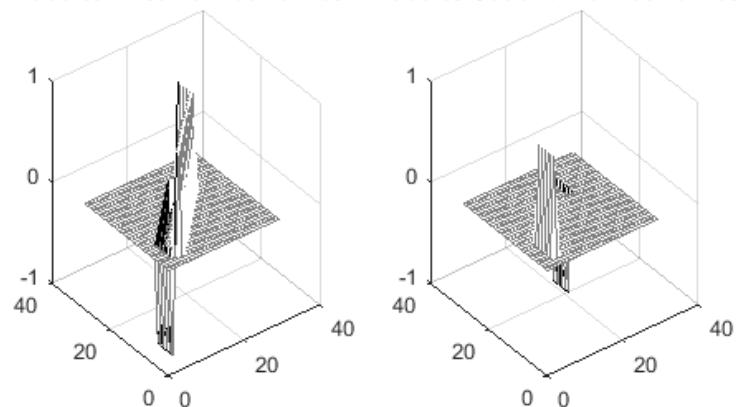
- Rettangolo bianco - mappa delle ampiezze e delle direzioni — filtro di Sobel



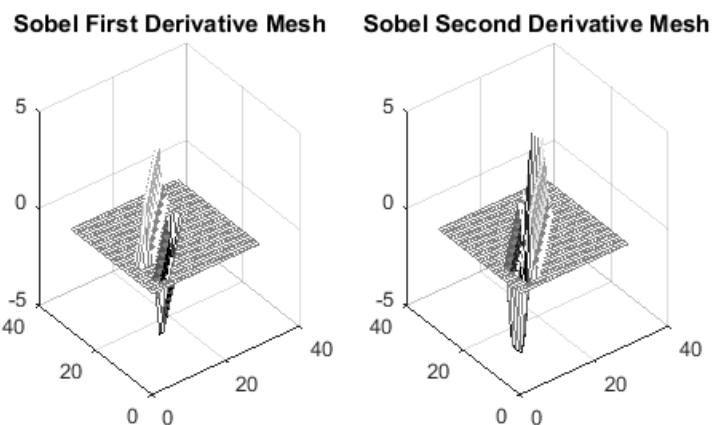
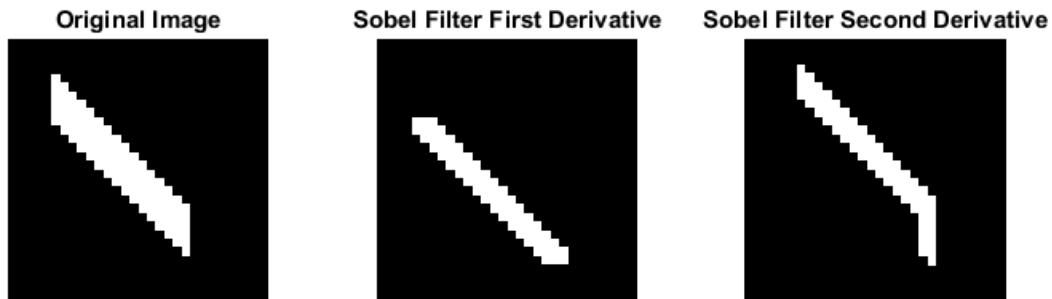
- Rettangolo obliquo — derivata prima e seconda filtro di Roberts



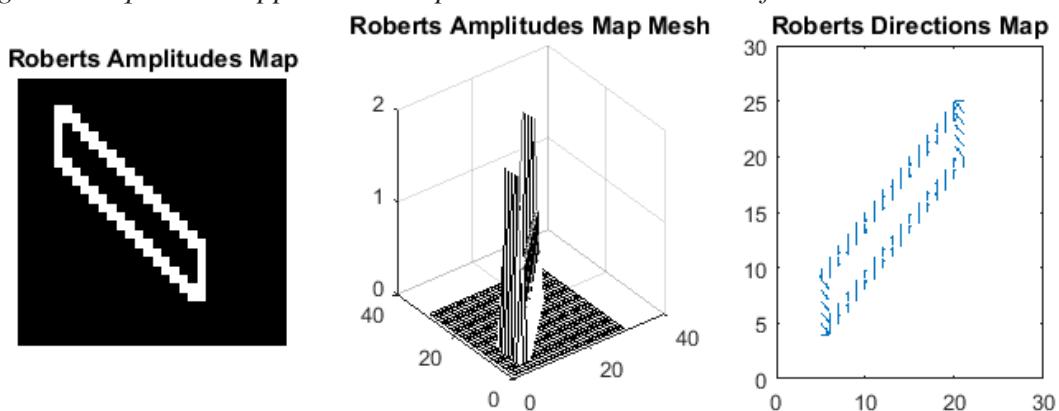
**Roberts First Derivative Mesh** **Roberts Second Derivative Mesh**



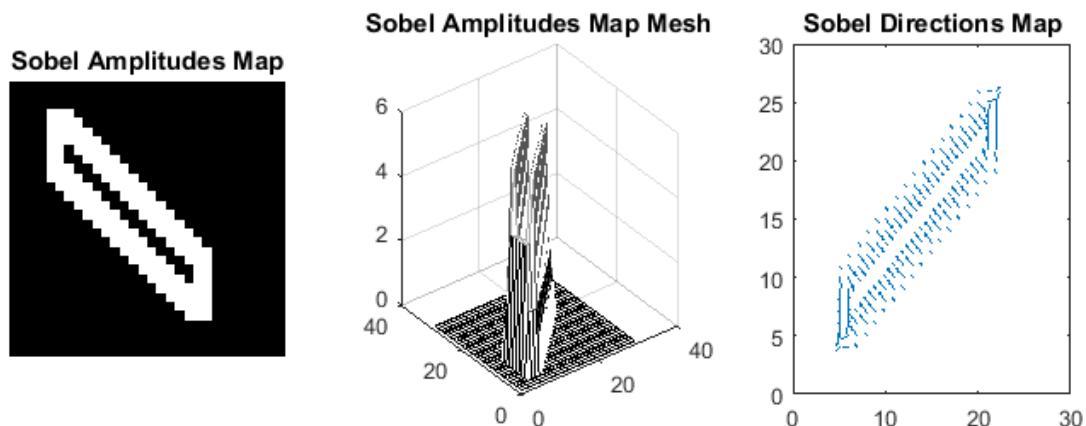
- Rettangolo obliquio – derivata prima e seconda filtro di Sobel



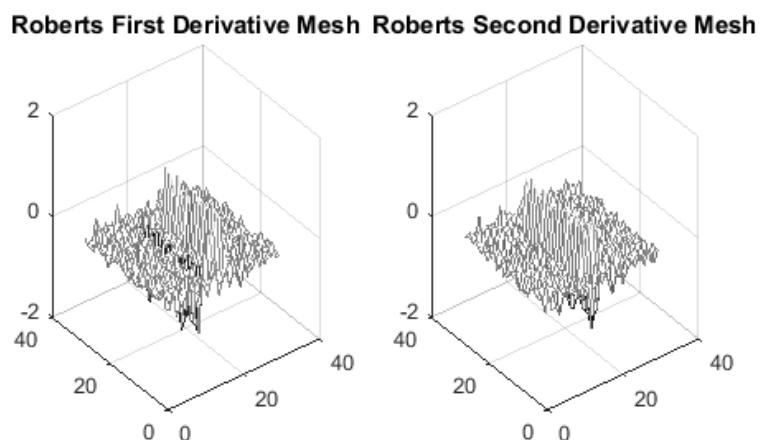
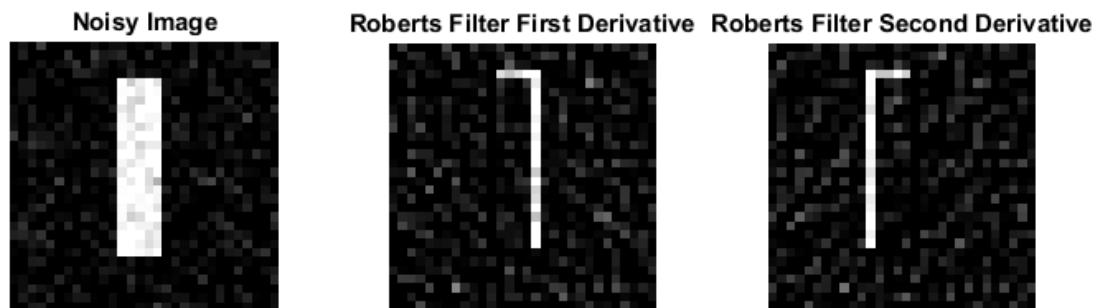
- Rettangolo obliquio – mappa delle ampiezze e delle direzioni filtro di Roberts



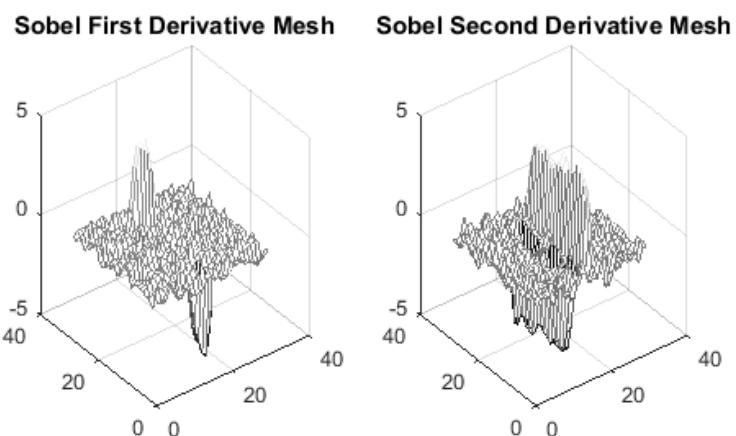
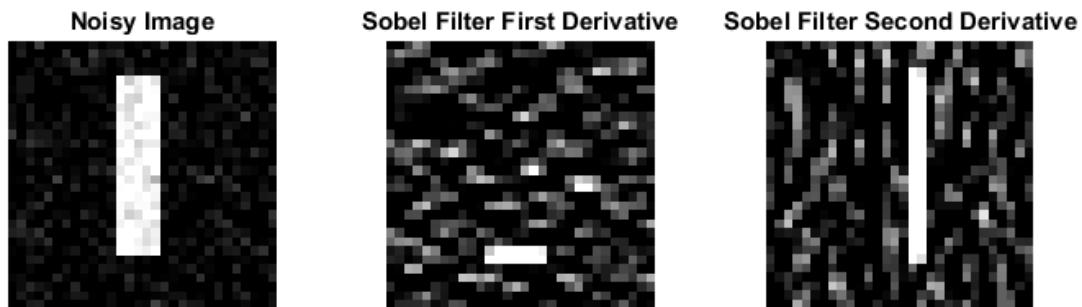
- Rettangolo obliquo — mappa delle ampiezze e delle direzioni filtro di Sobel



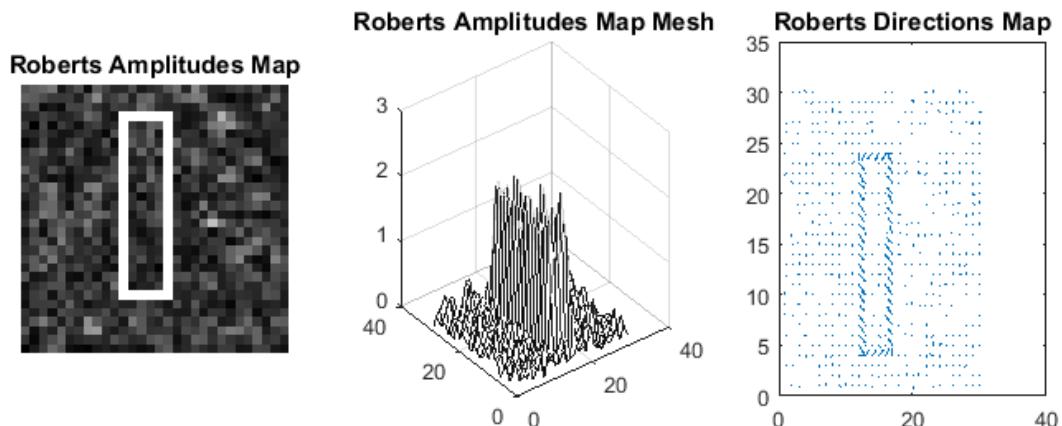
- Rettangolo rumoroso — derivata prima e seconda filtro di Roberts



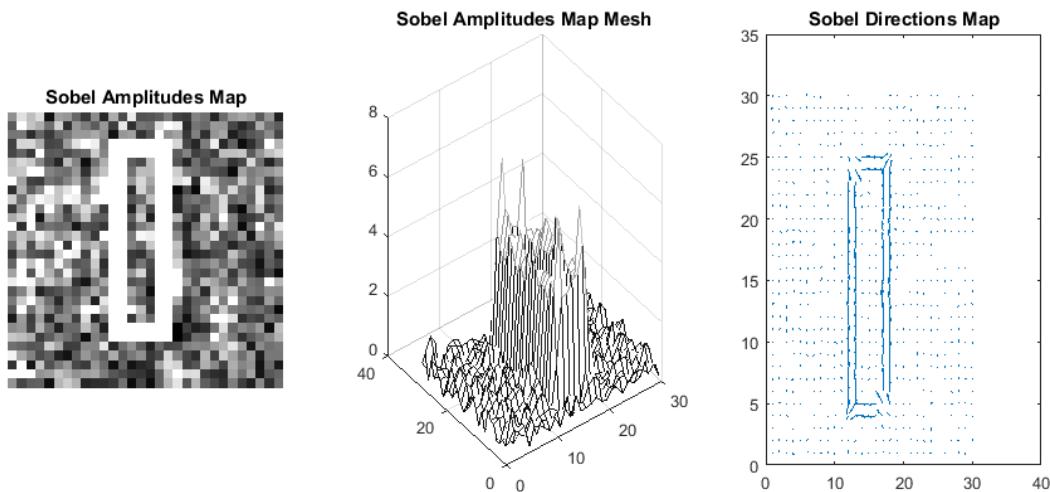
- Rettangolo rumoroso — derivata prima e seconda filtro di Sobel



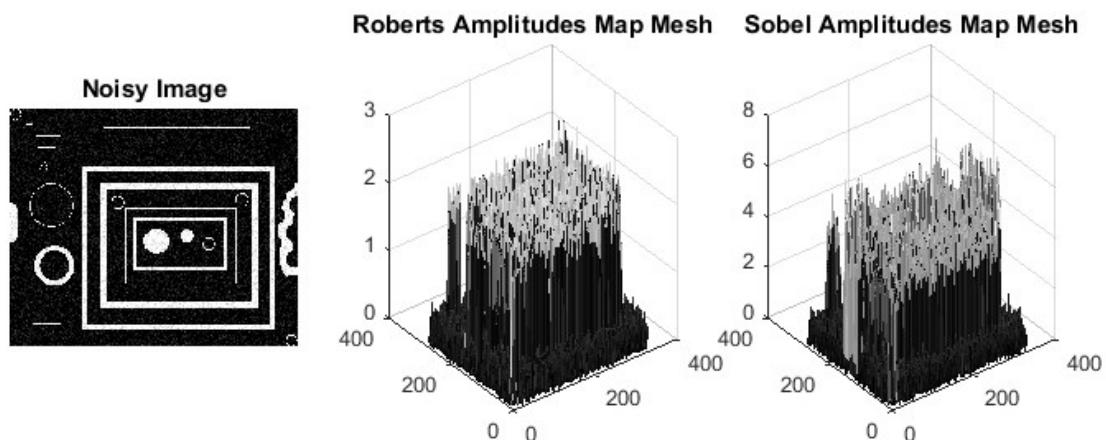
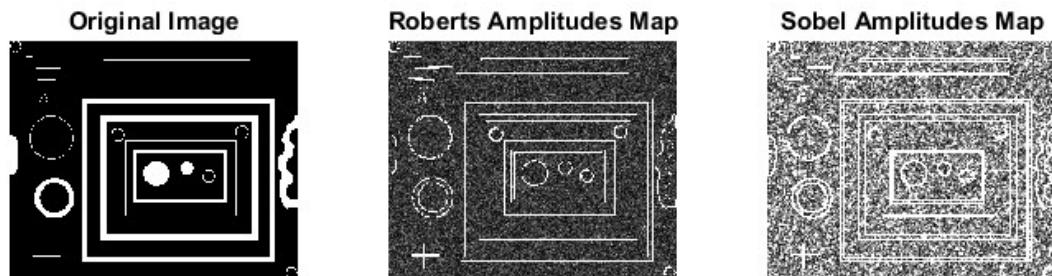
- Rettangolo rumoroso — mappe delle ampiezze e delle direzioni filtro di Roberts



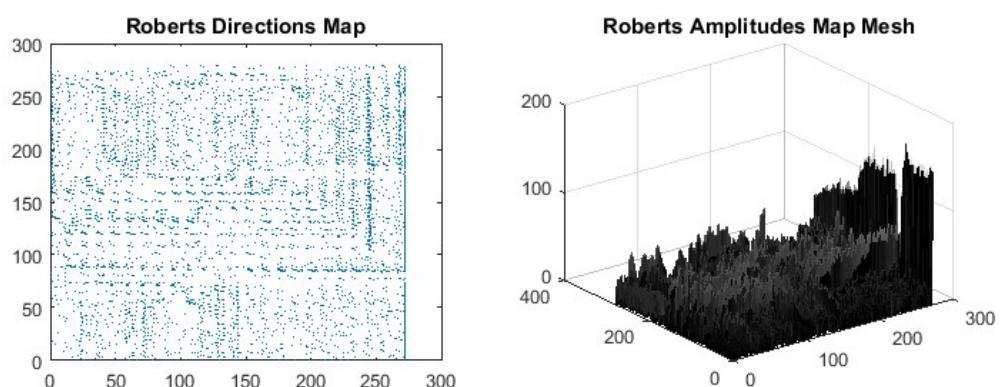
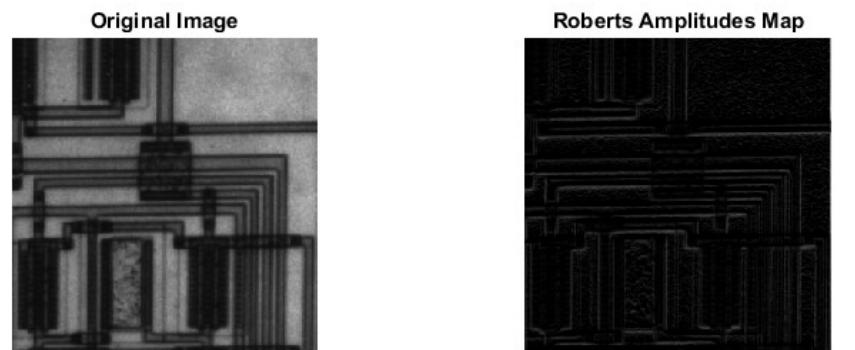
- Rettangolo rumoroso — mappe delle ampiezze e delle direzioni filtro di Sobel



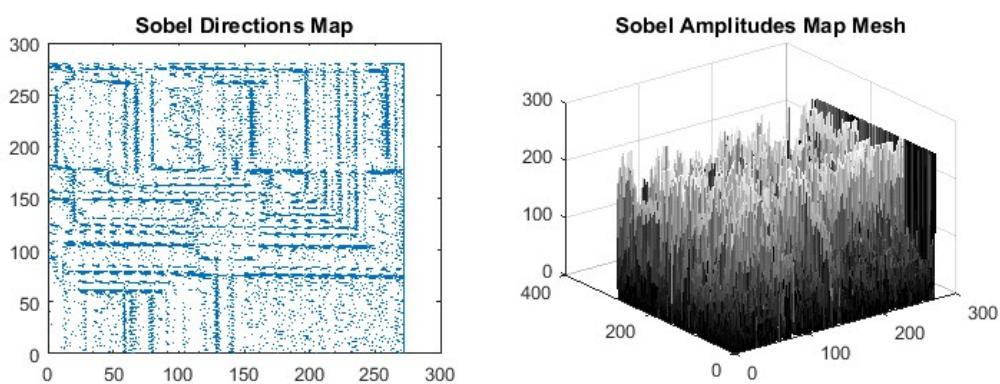
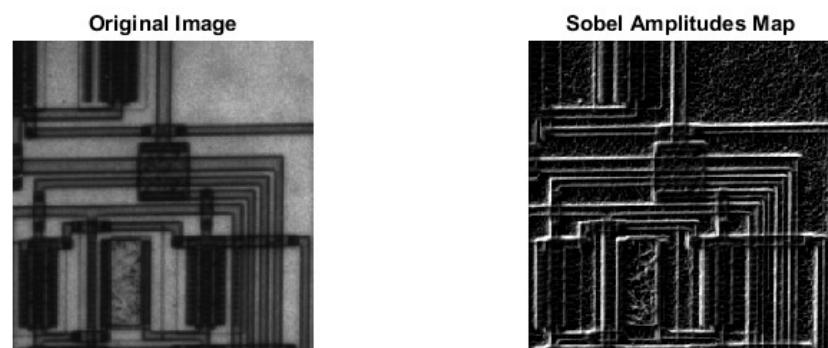
- Blobs — mappa delle ampiezze filtri di Roberts e Sobel



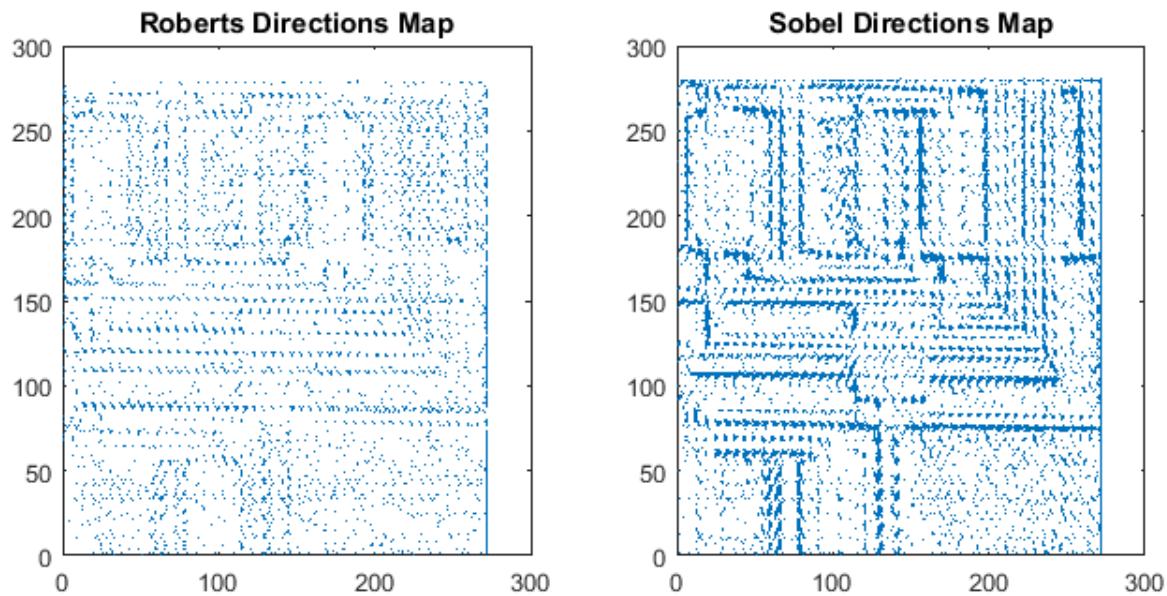
- Circuit — mappa delle ampiezze e delle direzioni filtro di Roberts



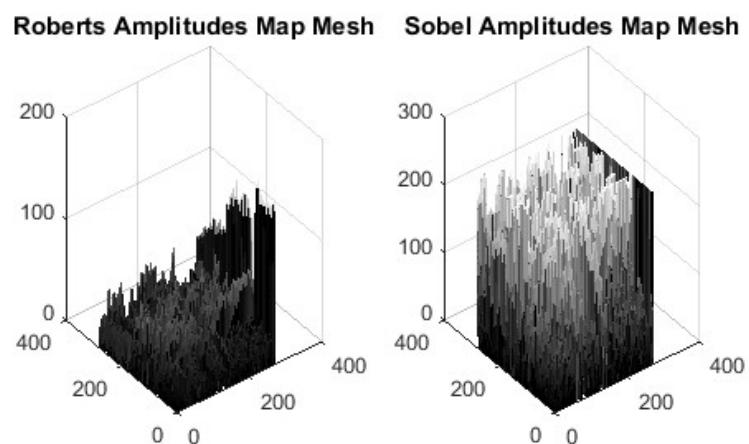
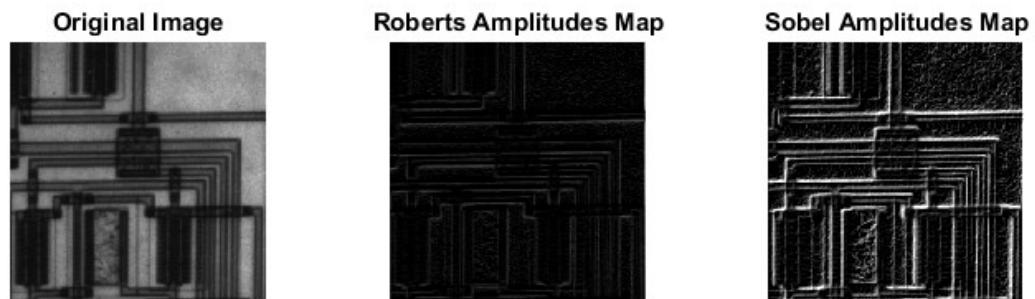
- Circuit — mappa delle ampiezze e delle direzioni filtro di Sobel



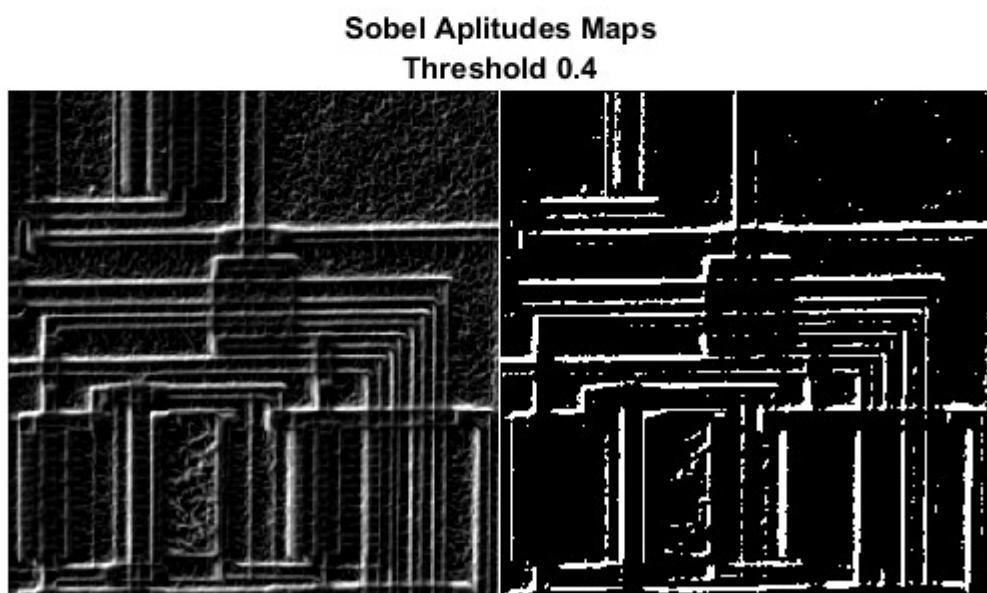
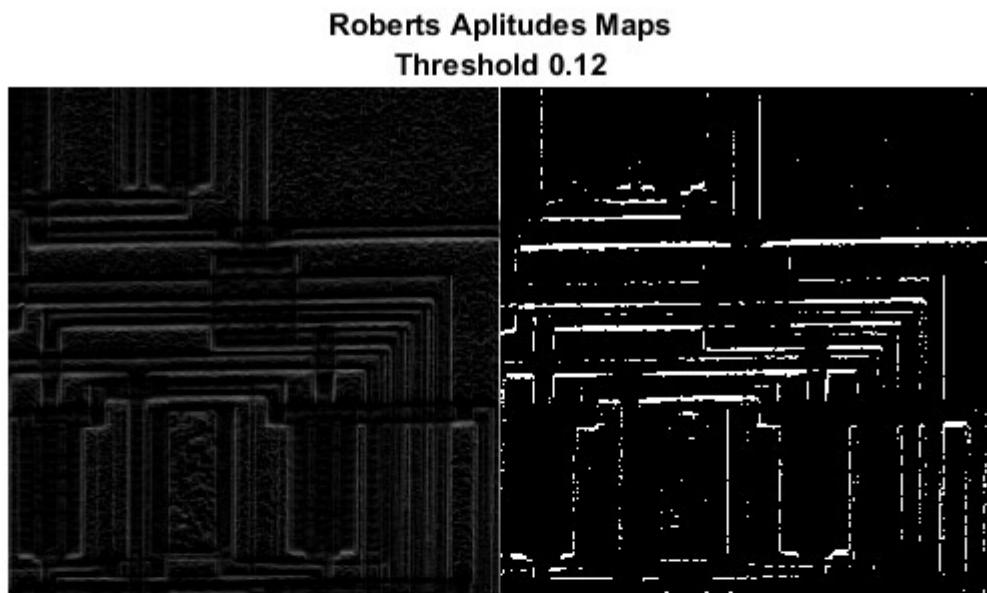
- Focus on: circuit — le mappe delle direzioni



- Circuit — le mappe delle ampiezze



- Circuit — mappe delle ampiezze binarizzate



#### Osservazioni finali:

Nello script abbiamo osservato come si utilizzano i filtri di Roberts e Sobel per rilevare gli edge di una forma all'interno di un'immagine e come estrarre la mappa delle ampiezze degli edge e la mappa delle direzioni degli edge.

Si nota come il filtro di Roberts sia più preciso nell'individuare gli edge, ma sia meno preciso nelle mappe delle ampiezze e soprattutto in quelle delle direzioni, dove serve una sogliatura molto bassa per rilevare gli edge con la controindicazione di far rientrare nell'immagine anche del rumore.

## Esercizio 2.15

Effettuare l'estrazione di mappe di edge usando la funzione edge() di MATLAB.

- Filtraggio di Sobel. Usare l'immagine ‘blobs.png’ con rumore come nell'esercizio 2.14. Calcolare le ampiezze di edge usando la edge() di MATLAB. Recuperare le ampiezze di edge calcolate all'esercizio 2.14 usando fspecial() e imfilter(). Visualizzare le due ampiezze assieme usando la imshowpair(). Stimare le prestazioni dei due algoritmi usando il MSE. Riportare le osservazioni nelle conclusioni.
- Filtraggio LoG (Laplaciano di Gaussiana). Filtrare l'immagine ‘coins.png’ usando la funzione edge() con i parametri di default. Effettuare un filtraggio analogo sull'immagine ‘mri.tif’. Riportare le osservazioni nelle conclusioni.
- Effettuare tre filtri LoG della ‘mri.tif’, rispettivamente con il parametro sigma = 0.5 ; 2 ; 3 della gaussiana. Riportare le osservazioni nelle conclusioni.

Codice:

- script

```
%%%%%
% Script for visualizing effects of Laplacian
% filter.
%
% Silvia Gioia Florio, matr. 119328
% 11/04/2017 - Excercise 2.15
%%%%%
%% Sobel (and Roberts) edges amplitudes %%
% initialization %
% reading image
img = imread('blobs.png');
img = double(img);
% adding noise
noisy = sigNoise(img, 0.1);
%
% calculate edges %
% function edge()
edgesSob = edge(noisy, 'sobel');
edgesSob = double(edgesSob);
edgesRob = edge(noisy, 'roberts');
edgesRob = double(edgesRob);

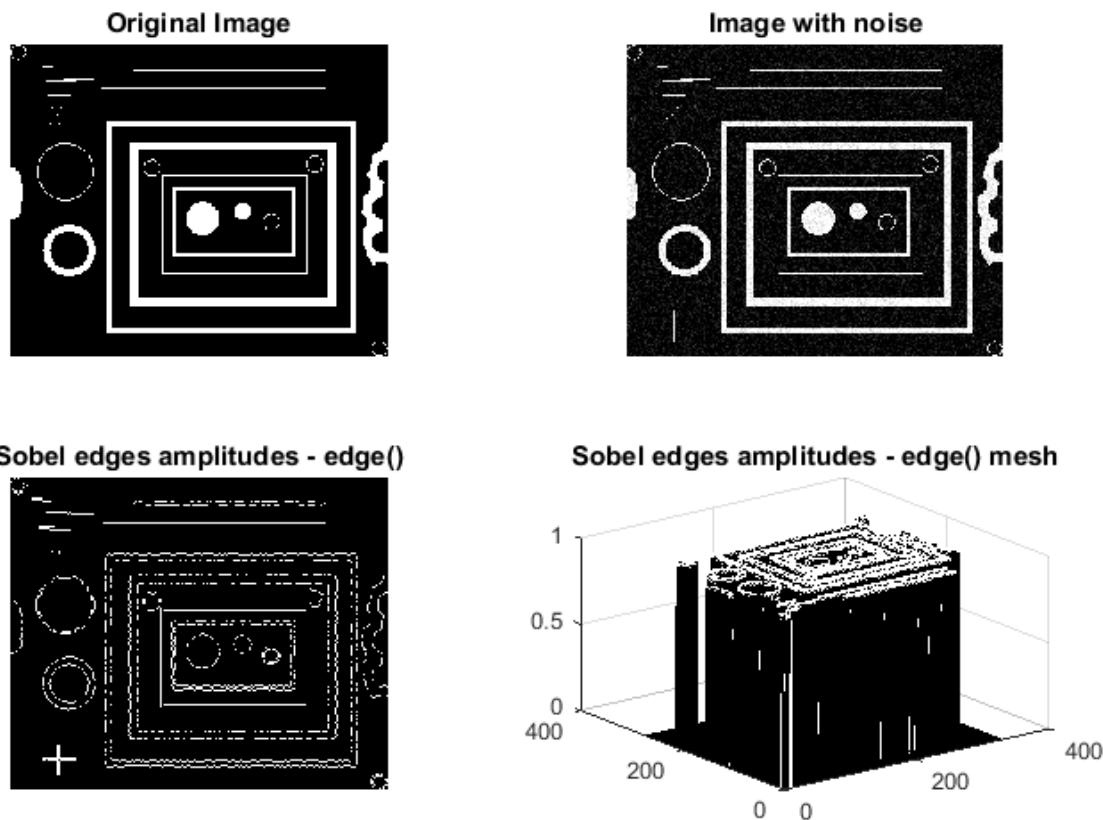
%
% display results %
% sobel
figure
subplot(2,2,1), imshow(img, 'InitialMagnification', 'fit'), title('Original Image')
subplot(2,2,2), imshow(noisy, 'InitialMagnification', 'fit'), title('Image with noise')
subplot(2,2,3), imshow(edgesSob), title('Sobel edges amplitudes - edge()')
subplot(2,2,4), mesh(edgesSob), title('Sobel edges amplitudes - edge() mesh')
% roberts
figure
subplot(2,2,1), imshow(img, 'InitialMagnification', 'fit'), title('Original Image')
subplot(2,2,2), imshow(noisy, 'InitialMagnification', 'fit'), title('Image with noise')
subplot(2,2,3), imshow(edgesRob), title('Roberts edges amplitudes - edge()')
subplot(2,2,4), mesh(edgesRob), title('Roberts edges amplitudes - edge() mesh')
%
% fspecial and imfilter %
% sobel %
% matrixes
sobX = fspecial('sobel');
sobY = transpose(sobX);
```

```
% apply
blobsNoiseFSX = imfilter(noisy, sobX);
blobsNoiseFSY = imfilter(noisy, sobY);
% amplitudes
ampNSob = abs(blobsNoiseFSX) + abs(blobsNoiseFSY);
%
% roberts %
% matrixes
robX = [-1,0;0,1];
robY = [0,-1;1,0];
% apply
blobsNoiseFRX = imfilter(noisy,robX);
blobsNoiseFRY = imfilter(noisy,robY);
% amplitudes
ampNRob = abs(blobsNoiseFRX) + abs(blobsNoiseFRY);
%
% display results %
% sobel
figure
subplot(2,2,1), imshow(img, 'InitialMagnification', 'fit'), title('Original Image')
subplot(2,2,2), imshow(noisy, 'InitialMagnification', 'fit'), title('Image with noise')
subplot(2,1,2), imshowpair(edgesSob, ampNSob, 'montage'), title('Sobel edges amplitudes
maps - L: edge(), R:fspecial() and imfilter()')
% roberts
figure
subplot(2,2,1), imshow(img, 'InitialMagnification', 'fit'), title('Original Image')
subplot(2,2,2), imshow(noisy, 'InitialMagnification', 'fit'), title('Image with noise')
subplot(2,1,2), imshowpair(edgesRob, ampNRob, 'montage'), title('Roberts edges amplitudes
maps - L: edge(), R:fspecial() and imfilter()')
%
% evaluation %
% calculate mse
mseSobEdge = immse(img, edgesSob);
mseSobFilter = immse(img, ampNSob);
mseRobEdge = immse(img, edgesRob);
mseRobFilter = immse(img, ampNRob);
% create table
% create rows
edge_function = [mseSobEdge; mseRobEdge];
fspecial_imfilter = [mseSobFilter; mseRobFilter];
table(edge_function, fspecial_imfilter, 'RowNames', {'Sobel', 'Roberts'})
%
%% Laplacian of Gaussian filter %%
% initialization %
% coins
coins = imread('coins.png');
coinsD = double(coins);
% mri
mri = imread('mri.tif');
mriD = double(mri);
% filtering
edgeCoins = edge(coinsD, 'log');
edgeMri = edge(mriD, 'log');
%
% display results %
% coins
figure
subplot(1,2,1), imshow(coins), title('coins.png')
subplot(1,2,2), imshow(edgeCoins), title('Laplacian on coins.png')
% mri
figure
subplot(1,2,1), imshow(mri), title('mri.tif')
subplot(1,2,2), imshow(edgeMri), title('Laplacian on mri.tif')
%
%% Changing sigma parameter %%
% initialization %
sigma05 = 0.5;
sigma2 = 2.;
sigma3 = 3.;
```

```
% apply filter %
edgeMri05 = double(edge(mriD, 'log', [], sigma05));
edgeMri2 = double(edge(mriD, 'log', [], sigma2));
edgeMri3 = double(edge(mriD, 'log', [], sigma3));
%
% display results
% sigma = 0.5
figure
subplot(2,2,1), imshow(mri), title('mri.tif')
subplot(2,2,2), imshow(edgeMri05), title('Laplacian with sigma = 0.5')
subplot(2,2,4), mesh(edgeMri05), title('Mesh')
% sigma = 2
figure
subplot(2,2,1), imshow(mri), title('mri.tif')
subplot(2,2,2), imshow(edgeMri2), title('Laplacian with sigma = 2')
subplot(2,2,4), mesh(edgeMri2), title('Mesh')
% sigma = 3
figure
subplot(2,2,1), imshow(mri), title('mri.tif')
subplot(2,2,2), imshow(edgeMri3), title('Laplacian with sigma = 3')
subplot(2,2,4), mesh(edgeMri3), title('Mesh')
```

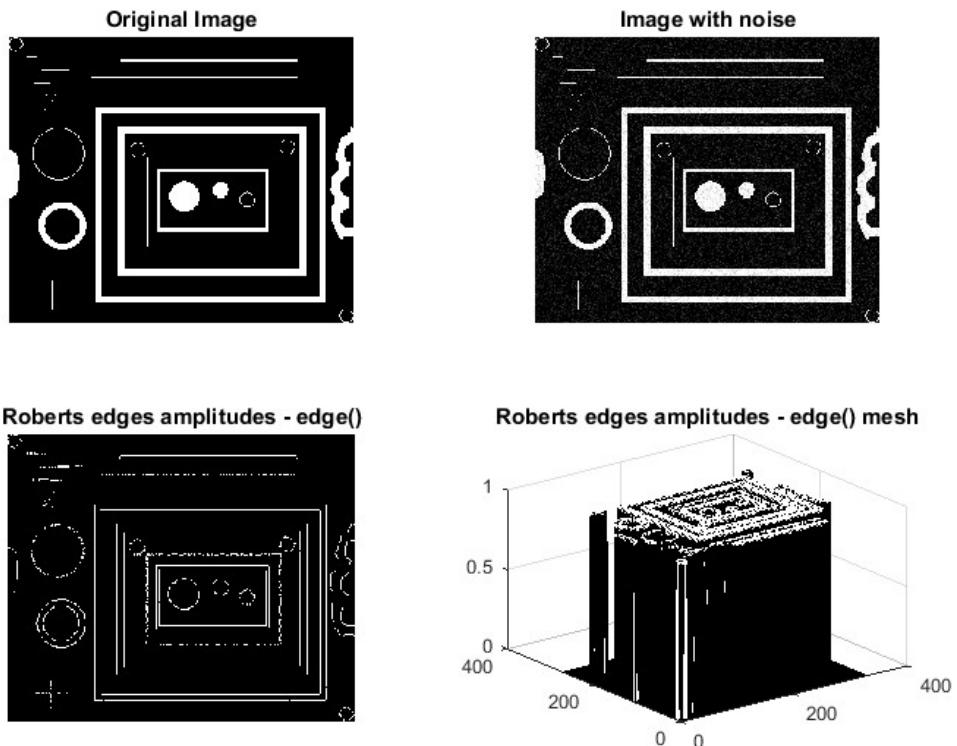
**Output:**

- *Mappa delle ampiezze degli edge del filtro di Sobel con la funzione edge()*



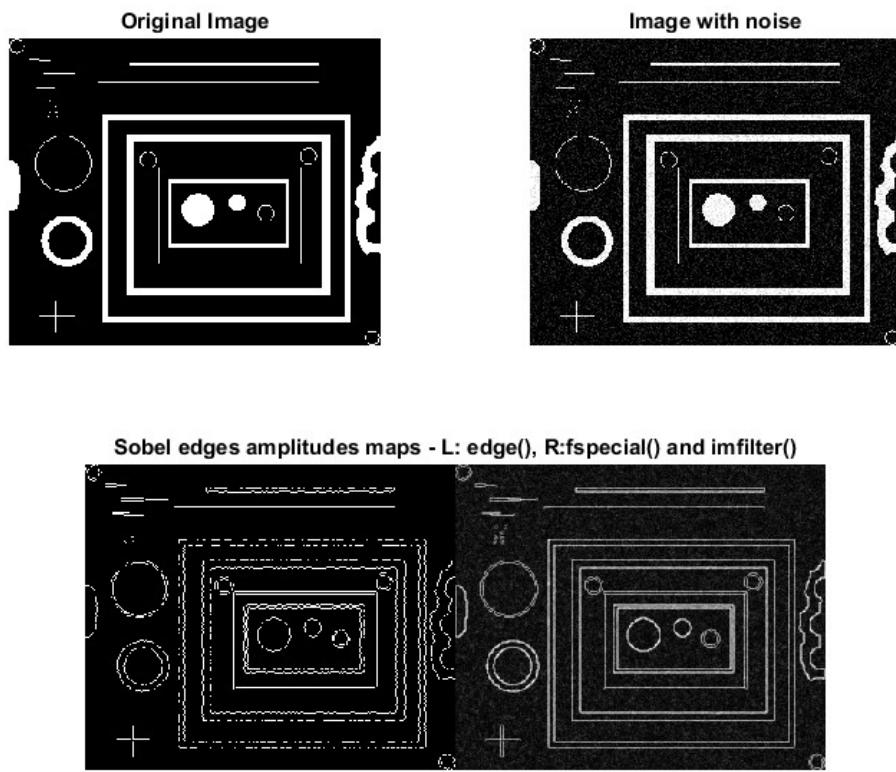
*Con i parametri di default le forme si riconoscono, anche se gli edge non vengono rilevati nel loro complesso.*

- Mappa delle ampiezze degli edge del filtro di Roberts con la funzione `edge()`



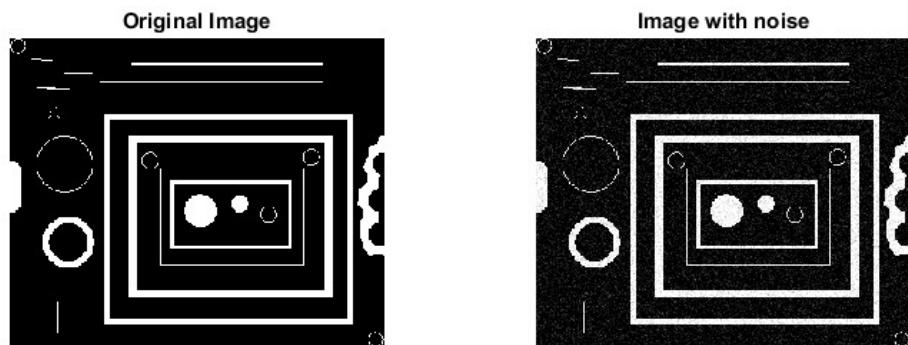
*Il filtro di Roberts, con la funzione `edge()`, risulta meno efficace del filtro di Sobel.*

- Confronto tra la funzione `edge()` e `fspecial()` + `imfilter()` - filtro di Sobel

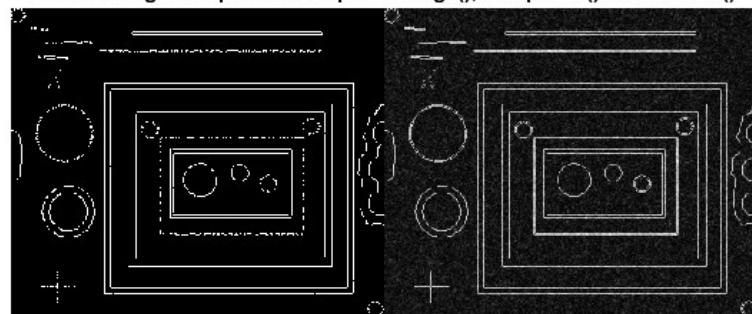


*Nell'immagine a destra gli edge sono più completi, ma è rimasto anche molto rumore.*

- Confronto tra la funzione `edge()` e `fspecial() + imfilter()` - filtro di Sobel



Roberts edges amplitudes maps - L: `edge()`, R:`fspecial()` and `imfilter()`



Nell'immagine a destra vengono rilevati più edge, sempre mantenendo molto rumore.

- Confronto tramite MSE

| edge_function | fspecial_imfilter |
|---------------|-------------------|
| Sobel         | 0.17868           |
| Roberts       | 0.15616           |

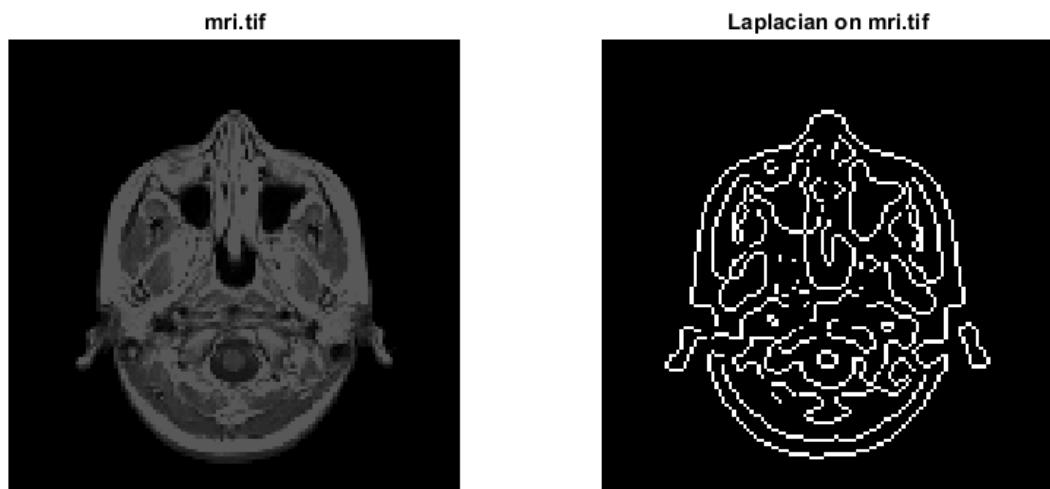
|         |        |
|---------|--------|
| Sobel   | 2.3353 |
| Roberts | 0.3024 |

L'MSE dimostra che la funzione `edge()` è più efficace

- Filtro Laplaciano di Gaussiana sull'immagine delle monete

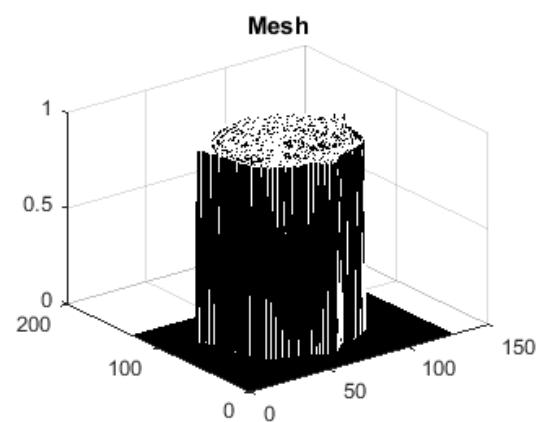
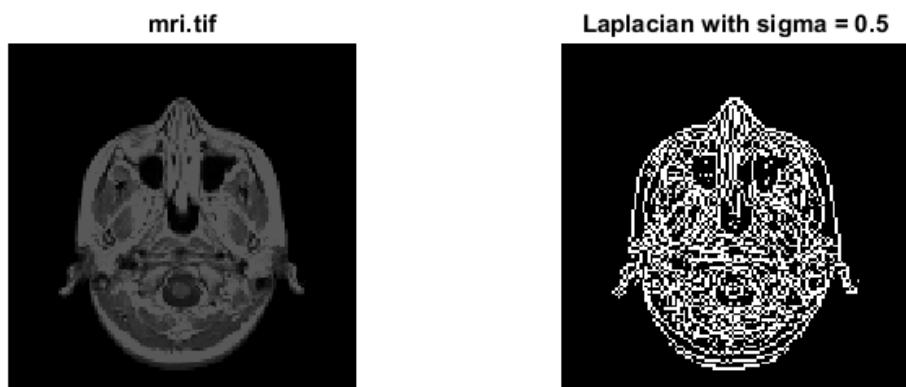


- Filtro Laplaciano di Gaussiana sull'immagine della risonanza magnetica

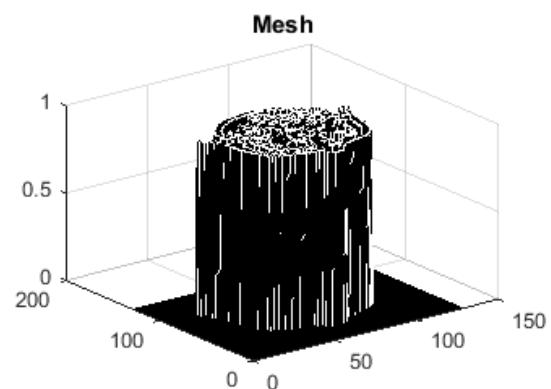
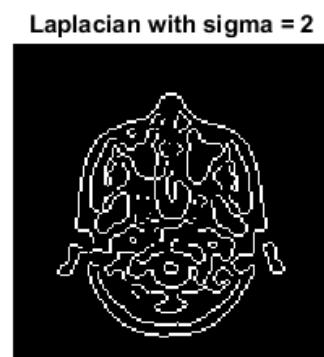
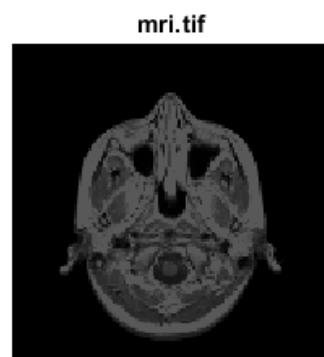


Con il parametro di default vengono individuati i contorni delle parti più chiare della risonanza.

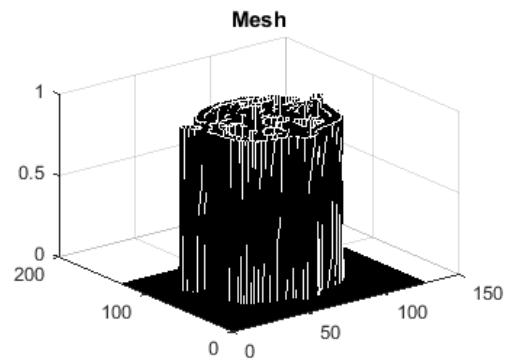
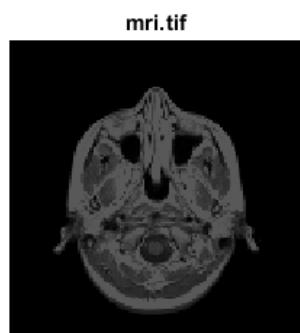
- Laplaciano con  $\sigma = 0.5$



- Laplaciano con sigma = 2



- Laplaciano con sigma = 3



**Osservazioni finali:**

Nello script è stata visualizzata la risposta della funzione edge() nella modalità filtro di Sobel, quella filtro di Roberts e infine nella modalità filtro Laplaciano di Gaussiana.

Per quanto riguarda i filtri di Sobel e Roberts, il calcolo dell'MSE ha rilevato una maggior efficacia da parte della funzione edge().

Dal punto di vista visivo i contorni rilevati da fspecial() + imfilter() sono più completi, ma allo stesso tempo, quest'ultimo procedimento riporta anche molto rumore.

Per quanto riguarda il filtro Laplaciano di Gaussiana è stato prima visto l'effetto con i parametri di default dove l'output ha mostrato che solo i contorni delle forme con valori di grigio più alti venivano rilevati. È stato poi variato il valore del parametro sigma per capirne l'influenza e si è visto che più il valore è alto, meno contorni vengono rilevati. Con il parametro a 0.5, si è generata un'immagine molto dettagliata, mentre con il valore 3 è stata rilevata la forma generale, ma per quanto riguarda i dettagli i contorni rilevati sono stati solo quelli delle forme molto chiare.

## Esercizio 2.16

Rilevamento di punti isolati e segmenti rettilinei.

- Definire un'immagine sintetica costituita da soli tre punti isolati; visualizzarla.
- Applicare un filtro derivativo di Laplace usando fspecial() e imfilter() e ottenere la mappa delle ampiezze di edge; visualizzarla anche in modalità grafica 3D, cioè mesh().
- Invertire l'immagine filtrata (negativo-positivo) ed eliminare i valori negativi mediante sogliatura usando im2bw(); visualizzare la nuova immagine anche in grafica.
- Definire una nuova immagine sintetica costituita da due segmenti di retta che si incrociano.
- Ripetere il filtraggio e le operazioni effettuate ai punti (b),(c).
- Riportare nelle conclusioni le osservazioni sui risultati ottenuti ai punti (c), (e).

Codice:

- script

```
%%%%%%%%%%%%%
% Script for visualize isolated points and
% straight lines.
%
% Silvia Gioia Florio, matr. 119328
% 11/04/2017 - Esercizio 2.16
%%%%%%%%%%%%%
%
%%
%%% synthetic image with three white points %%%
%%
%% Initialize Image %%
% initialize black image
img = zeros(50,50);
% set three single points to white
img(25,30) = 1.;
img(34,12) = 1.;
img(3,21) = 1.;
% display result
imshow(img, 'InitialMagnification', 'fit');
title('Three White Points');

%% Laplace filter %%
% initialize filter
lap = fspecial('laplacian');
% filtering image
filteredImage = imfilter(img,lap);
% display results
figure;
subplot(2,2,1);
imshow(img);
title('Original Image');
subplot(2,2,2);
mesh(img);
title('Original Mesh');
grid();
subplot(2,2,3);
imshow(filteredImage);
title('Filtered Image');
subplot(2,2,4);
mesh(filteredImage);
title('Filtered Mesh');
grid();

%% Inversion and negative removal %%
```

```
% inversion
negativeFilteredImage = filteredImage*(-1);
% negativeRemoval
negativeRemoved = im2bw(negativeFilteredImage, 0);
% display results
figure;
subplot(3,2,1)
imshow(img);
title('Original Image');
subplot(3,2,2);
mesh(img);
title('Original Mesh');
grid();
subplot(3,2,3);
imshow(filteredImage);
title('Filtered Image');
subplot(3,2,4);
mesh(filteredImage);
title('Filtered Mesh');
grid();
subplot(3,2,5);
imshow(negativeRemoved);
title('Invert and Negative Removal Image');
subplot(3,2,6);
mesh(double(negativeRemoved));
title('Invert and Negative Removal Mesh');
grid();
%%
%%%% synthetic image with two crossed segments %%%
%%
%% Initialize Image %%
% initialize black image
img = zeros(50,50);
% set two crossed segments to white
img(19:45,30) = 1.;
img(28,12:40) = 1.;

% display result
imshow(img, 'InitialMagnification', 'fit');
title('Two Crossed Lines');

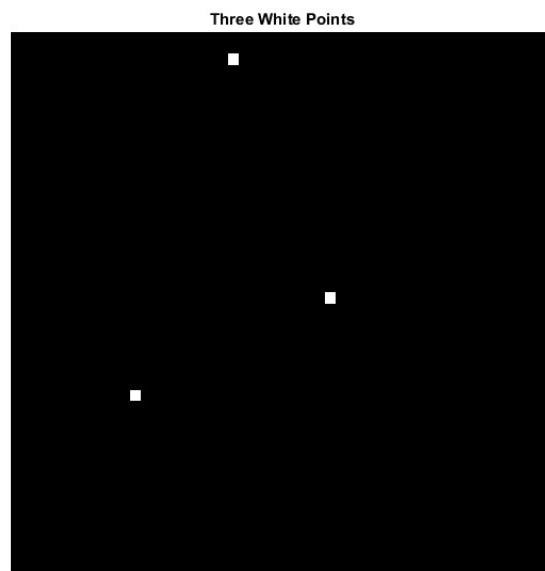
%% Laplace filter %%
% initialize filter
lap = fspecial('laplacian');
% filtering image
filteredImage = imfilter(img,lap);
% display results
figure;
subplot(2,2,1);
imshow(img);
title('Original Image');
subplot(2,2,2);
mesh(img);
title('Original Mesh');
grid();
subplot(2,2,3);
imshow(filteredImage);
title('Filtered Image');
subplot(2,2,4);
mesh(filteredImage);
title('Filtered Mesh');
grid();

%% Inversion and negative removal %%
% inversion
negativeFilteredImage = filteredImage*(-1);
% negativeRemoval
negativeRemoved = im2bw(negativeFilteredImage, 0);
% display results
figure;
subplot(3,2,1);
```

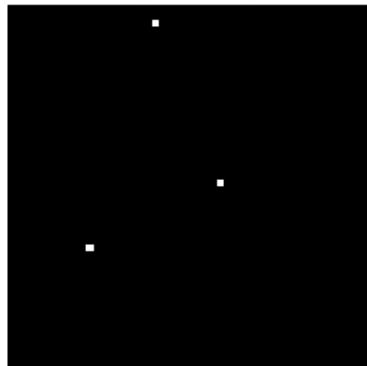
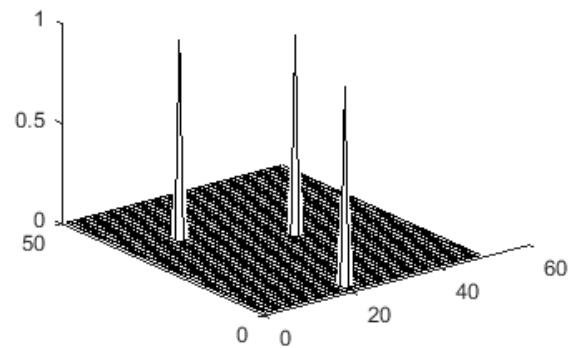
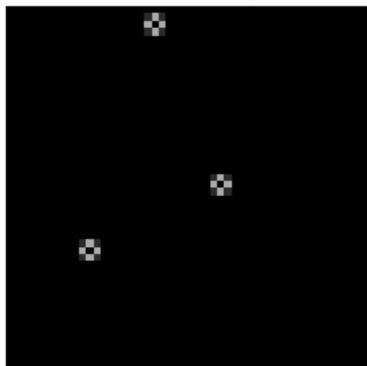
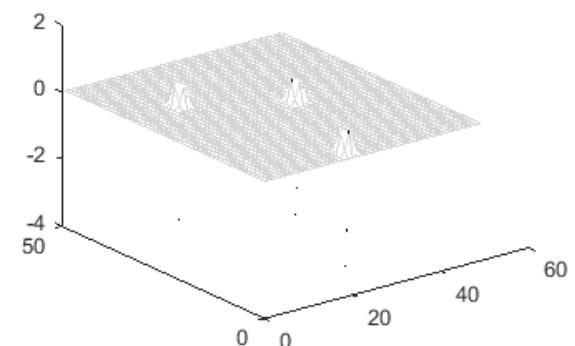
```
imshow(img);
title('Original Image');
subplot(3,2,2);
mesh(img);
title('Original Mesh');
grid();
subplot(3,2,3);
imshow(filteredImage);
title('Filtered Image');
subplot(3,2,4);
mesh(filteredImage);
title('Filtered Mesh');
grid();
subplot(3,2,5);
imshow(negativeRemoved);
title('Invert and Negative Removal Image');
subplot(3,2,6);
mesh(double(negativeRemoved));
title('Invert and Negative Removal Mesh');
grid();
```

**Output:**

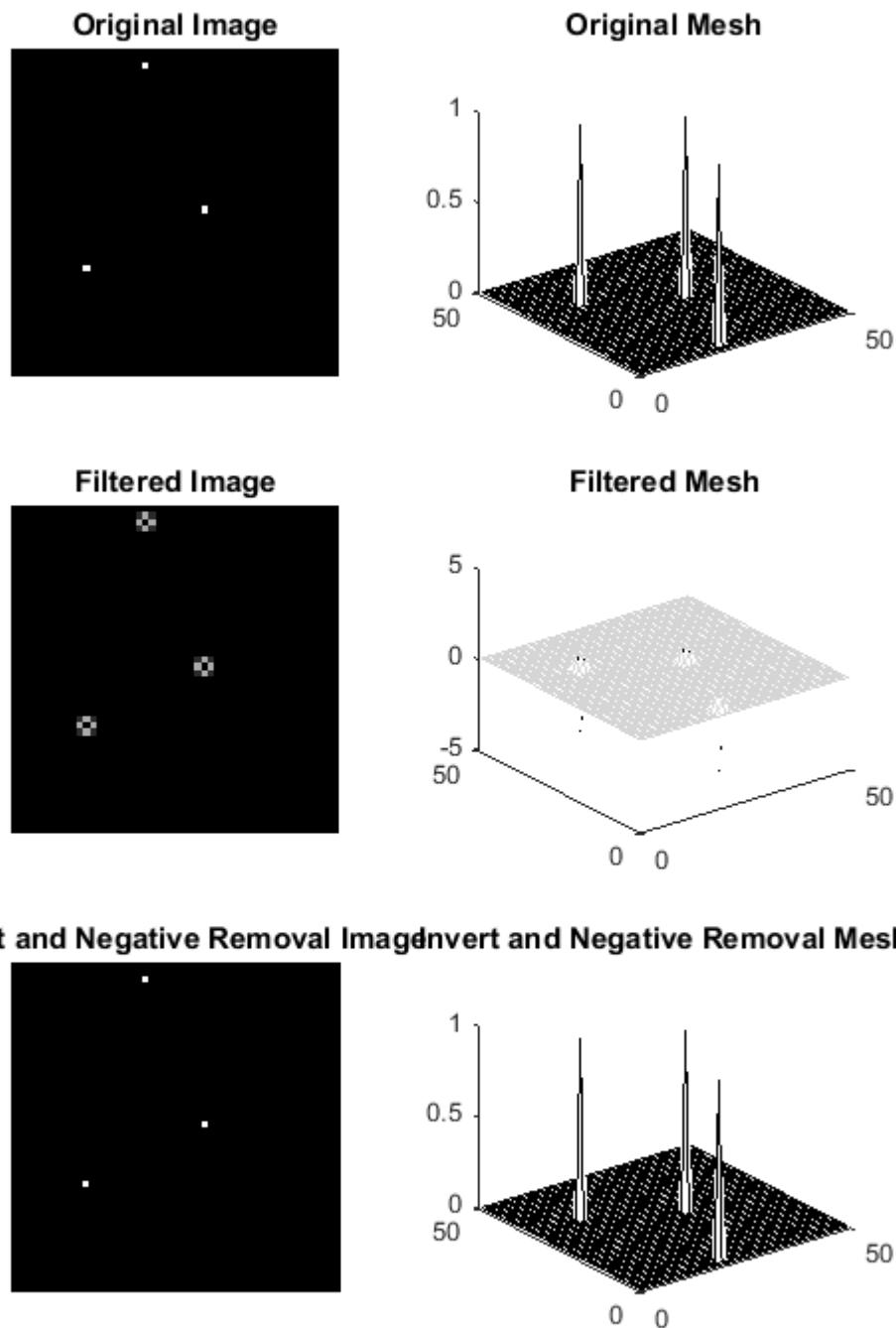
- *Immagine con i tre punti bianchi*



- *Filtro Laplaciano*

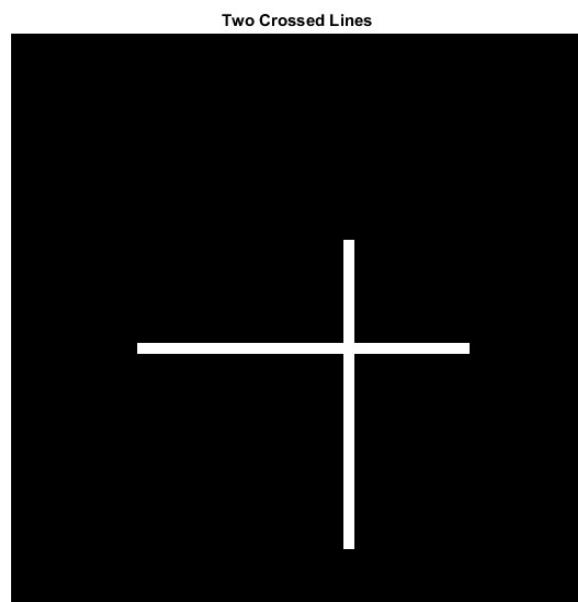
**Original Image****Original Mesh****Filtered Image****Filtered Mesh**

- *Inversione e rimozione dei negativi*

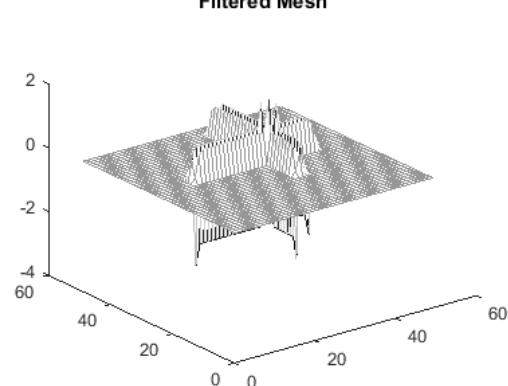
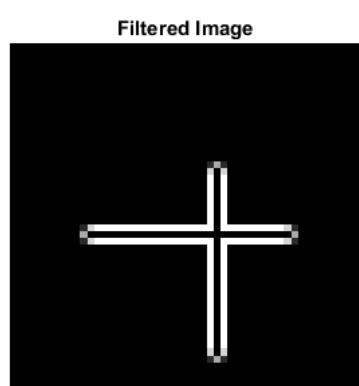
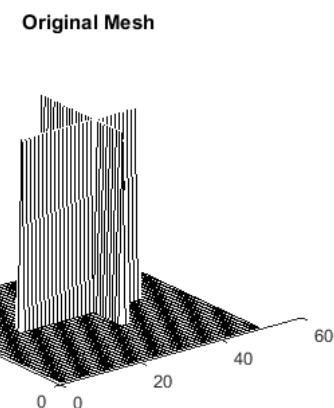
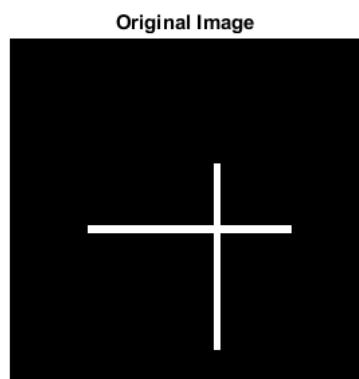


Si nota come si è tornati all'immagine originale.

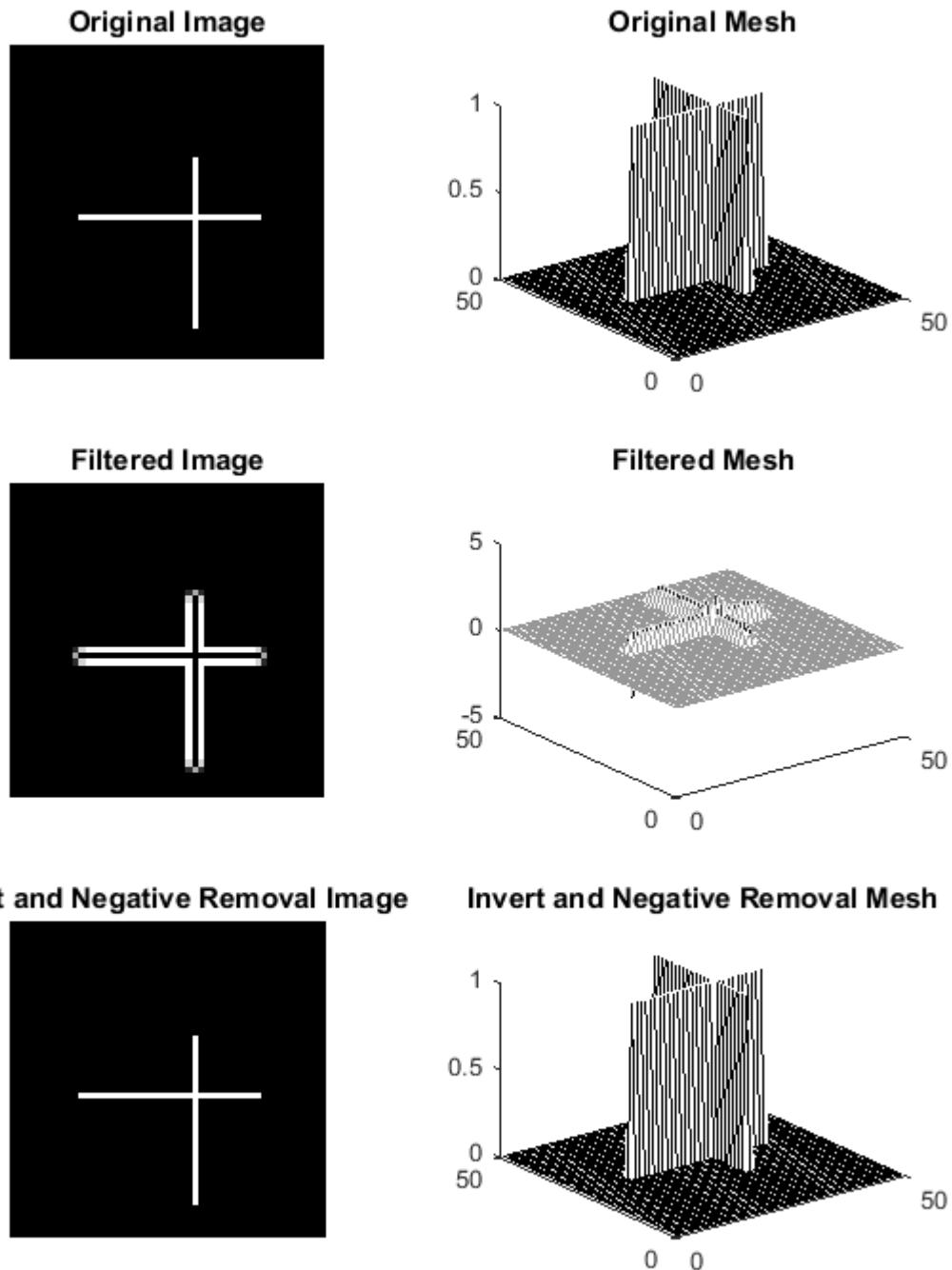
- *Immagine con due linee incrociate*



- *Filtro Laplaciano*



- Inversione e rimozione dei negativi



Anche in questo caso l'immagine risultante è uguale all'originale.

#### Osservazioni finali:

Nello script si è visto come il filtro Laplaciano crea un contorno sulle forme, grazie al quale tramite un'inversione si riesce a tornare all'immagine originale.

## Esercizio 2.17

Definire un'immagine sintetica costituita da almeno due rettangoli pieni non sovrapposti (barre).

- Studiare la documentazione MATLAB sulle procedure bwconncomp() per l'individuazione di regioni connesse nell'immagine - e regionprops() per la stima di caratteristiche geometriche (features) delle regioni individuate.
- Applicare nell'ordine prima la bwconncomp(), poi la regionprops() all'immagine per estrarre i centroidi, i perimetri, le aree dei rettangoli che vi sono rappresentati.
- Presentare i risultati numericamente per mezzo di una table();
- Sovrapporre all'immagine di partenza i centroidi calcolati e visualizzare i risultati.

Codice:

- script

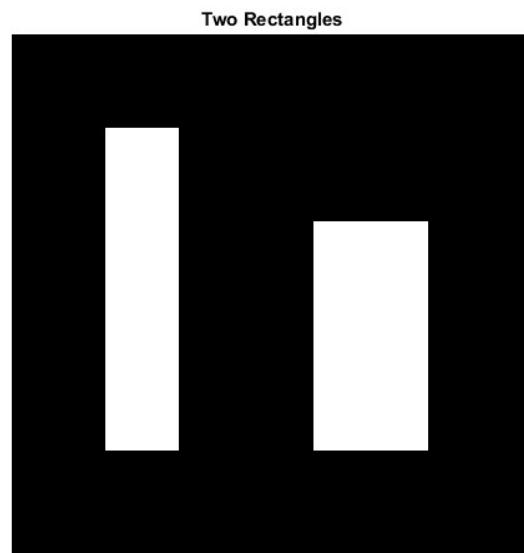
```
%%%%%
% Script for extracting area, perimeter and
% other informations from a synthetic image
%
% Silvia Gioia Florio, matr. 119328
% 11/04/2017 - Excercise 2.17
%%%%%
%
%% synthetic image with two rectangles %%
% Initialize Image %
% initialize black image
img = zeros(50,50);
% set two rectangles to white
img(19:40,30:40) = 1.;
img(10:40,10:16) = 1.;
% threshold image
imgBW = im2bw(img, 0.7);
% display result
imshow(img, 'InitialMagnification', 'fit');
title('Two Rectangles');

%% Centroids, Perimeters and Areas %%
% calculate centroids, perimeters and areas
conncomp = bwconncomp(imgBW);
props = regionprops('table', imgBW, 'Centroid', 'Perimeter', 'Area')
conn_comp = struct2table(conncomp)

%% Visualize Centroids
% visualize centroid locations superimposes to the
% binary image
figure
imshow(imgBW, 'InitialMagnification', 'fit')
hold on
plot(props.Centroid(:,1), props.Centroid(:,2), 'r*')
hold off
title('Centroids');
```

**Output:**

- *Immagine dei due rettangoli*



- *bwconncomp()*

```
conn_comp =
```

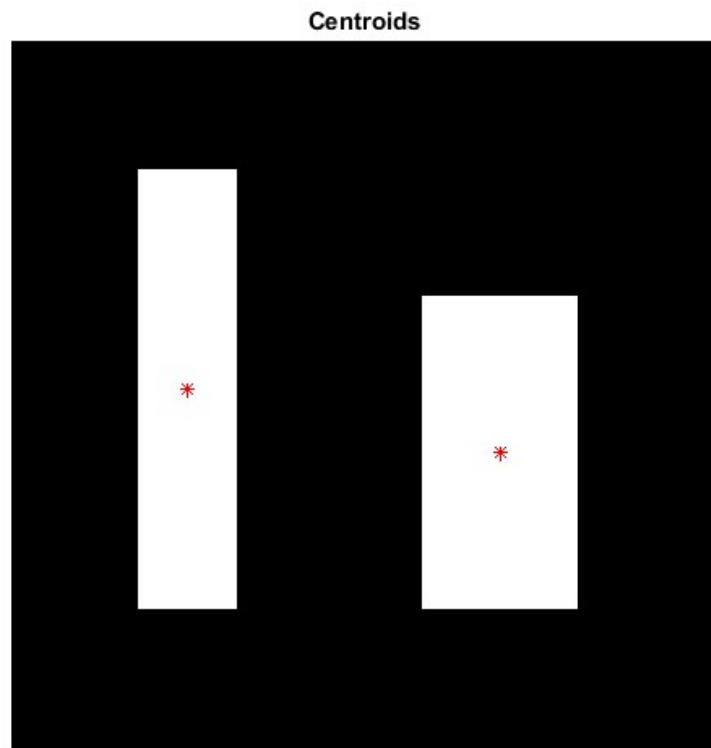
| <i>Connectivity</i> | <i>ImageSize</i> | <i>NumObjects</i> | <i>PixelIdxList</i>           |
|---------------------|------------------|-------------------|-------------------------------|
| 8                   | [50 50]          | 2                 | [217x1 double] [242x1 double] |

- *regionprops()*

```
props =
```

| <i>Area</i> | <i>Centroid</i> | <i>Perimeter</i> |
|-------------|-----------------|------------------|
| 217         | [13 25]         | 70.196           |
| 242         | [35 29.5]       | 60.396           |

- *Centroidi*



#### Osservazioni finali:

Nello script è stato approfondito il funzionamento della funzione regionprops() per avere informazioni di carattere geometrico sulle forme presenti nell'immagine.

In particolare è stato visto il calcolo dei centroidi per rilevare il centro della figura.