# Elaborazione delle Immagini

Sara Angeretti
@Sara1798

Fabio Ferrario
@fefabo

2023/2024

# Indice

# Capitolo 1

# Introduzione a MatLab

## 1.1 Appunti video lezione 2020 da Moodle

### 1.1.1 Ambienti di lavoro

**Command Window e Workspace**

- Command Window: finestra di comando, dove andiamo effettivamente a scrivere i comandi.
- Workspace: variabili in memoria, vediamo le variabili che abbiamo istanziato, mi dà un feedback immediato di quello che ho fatto.

**Variabili**

```
>> pippo = 10
pippo =
    10
>>
```

Questo vuol dire che istanziare una variabile assegnandole anche un valore numerico, dopo che ho premuto invio mi viene creata in memoria la variabile che ho creato (lo vedo nel workspace che l'ho creata). Però scrivendo così mi viene anche stampato a schermo il comando che ha eseguito.
Nel workspace vedo che ho una variabile chiamata pippo che ha valore 10, size 1x1 e class double.
Questo vuol dire che ha dimensione 1x1, cioè che è una singola cella (MatLab lavora con dati che sono matrici). Inoltre, double è il valore base, di default, quasi tutto viene fatto in virgola mobile (ci possono essere forzature se necessario).

Ci sono operazioni più complesse però di semplici assegnamenti, per cui potrebbe tornare utile evitare di stampare a schermo, perché non è assolutamente necessario vedere live l'output del comando che viene eseguito. Perciò:

```
>> pluto = 10;
>>
```

### 1.1.2 Lista di comandi utili

**¿¿ clear:** pulisce il workspace, cancella tutte le variabili che ho istanziato.

**¿¿ clc:** pulisce il command window, cancella tutti i commenti che ho scritto.

**freccia su tastiera:** va a riprendere l'*history* dei miei comandi nella command window, che apre in una finestrella pop-up.

### 1.1.3 Operazioni matematiche

#### Addizione

```
>> a = 2;
>> b = 1;
>> c = a + b;
```

#### Sottrazione

```
>> a = 2;
>> b = 1;
>> c = a - b;

ans =

    1
```

#### Prodotto

```
>> a = 2;
>> b = 1;
>> c = a * b;

ans =

    2
```

**Divisione**

```
>> a = 2;
>> b = 1;
>> c = a / b;

ans =

    2
```

**Elevamento a potenza**

```
>> a = 2;
>> a^3;

ans =

    8
```

MatLab richiede che il valore di un'operazione venga associato ad una variabile. Se non la creo io, lo fa MatLab per me (qua la variabile "`ans`"). Visibile nel *Workspace*.
Questa variabile può essere riutilizzata.

```
>> a + ans;

ans =

    10
```

Però ocio che ad ogni operazione verrà sovrascritta.

**Altre operazioni**

Si può operare in tanti modi con gli scalari su MatLab; per vedere le operazioni possibili, si deve consultare la documentazione.

### 1.1.4 La documentazione

1. Tasto "*Help*" in "Resources"
2. In Command Window possiamo chiederlo direttamente a MatLab:

- ▷ comando "`help`", mi dà dei suggerimenti sull'ultima operazione eseguita.
- ▷ comando "`help max`", (dove "`max`" è un esempio di operazione che dà in output il massimo numero in un array di numeri dati in input) mi dice quale è la sintassi con tutte le sue alternative dell'operazione richiesta.
- ▷ comando "`lookfor max`", (dove "`max`" è un esempio di parte del nome di un'operazione che non ci ricordiamo nella sua interezza) mi dà una lista di possibili operazioni che potrebbero essere quella che stiamo cercando e che contengono il pezzo di nome che gli ho assegnato.

### 1.1.5 Gli array

**Sintassi e operazioni sui vettori**

**Vettori riga:** i vettori sono rappresentati come concatenazione di valori, secondo la sintassi:

```
>> v = [1,2,3,4,5,6];
>> v

v =

    1   2     3     4     5     6

>>
```

Nel workspace vedremo:

| Name | Value | Size | Class |
|---|---|---|---|
| `v` | `[1,2,3,4,5,6]` | `1x6` | `double` |

In questo caso è un **vettore riga**.
Le *parentesi quadre* si usano per **concatenare** i valori, le *virgole* per separare i valori in un *vettore riga*, i *punti e virgola* per separare i valori in un *vettore colonna*.

**Vettori colonna:** se invece volessimo un **vettore colonna**:

```
>> w = [10;20;30;40];
>> w

w =

    10
    20
    30
    40

>>
```

Nel workspace vedremo

| Name | Value | Size | Class |
|---|---|---|---|
| `w` | `[10;20;30;40]` | `4x1` | `double` |

N.B.: sono ***indicizzabili***. Per indicizzare si usano le *parentesi tonde* e, cosa importantissima, *in MatLab* ***non esiste l'indice 0****, perciò si parte da 1 e non da 0.* Se si scrive lo 0 come indice, dà errore.
Se volessimo ad esempio il terzo elemento del vettore `v`:

```
>> v(3)

ans =

    3

>>
```

**Assegnare un valore:** questa indicizzazione la possiamo usare anche in assegnazione:

```
>> v(3) = 100;
>> v

v =

    1   2     100     4     5     6
```

```
>>
```

Posso lavorare anche su *serie di cellette* invece di una celletta singola:

```
>> v = (1:3);

ans =

1   2     100

>> v = (1:3);

ans =

    4     5     6

>>
```

Se volessi, senza sapere quanto è lungo un vettore (c'è un modo per saperlo, ma non lo vediamo ora), prendere tutti gli elementi da un certo indice fino alla fine, posso fare:

```
>> v(4:end)

ans =

    4     5     6

>>
```

**Scoprire la lunghezza di un vettore:** con il comando:

```
>> size(v)

ans =

    1   6
```

```
>>
```

Ovvero una riga e 6 colonne. La convenzione (è sempre così) è prima righe e poi colonne, perciò se volessi conoscere solo le righe:

```
>> size(v,1)

ans =

    1

>>
```

se volessi conoscere solo le colonne:

```
>> size(v,2)

ans =

    6

>>
```

**Somma:** come sappiamo da GAL, se sommo due vettori devono avere la stessa dimensione, altrimenti non posso sommarli.

**Prodotto matriciale:** come sappiamo da GAL, se moltiplico due vettori devono avere gli stessi indici interni, altrimenti non posso moltiplicarli. Es.: noi abbiamo `v 1x6` e `w 4x1`, non posso moltiplicarli perché $6 \neq 4$. Una cosa che potrei fare per ovviare il problema sarebbe selezionare un sotto vettore da `v` che abbia la stessa dimensione di `w`:

```
>> v(1:4)*w

ans =

    3210
```

```
>>
```

Per chi non ha ancora visto GAL, questo è un prodotto scalare, ma perché `ans` vale `3210`? Praticamente ho due matrici $m \times p$ e $p \times n$, la dimensione della matrice prodotto sarà $m \times n$. Quindi qua ho un vettore riga $1 \times 4$ e un vettore colonna $4 \times 1$, il prodotto sarà un vettore riga $1 \times 1$, che è quello che vediamo in `ans`. Inoltre il prodotto fra:

| una matrice | | | e una matrice | |
|---|---|---|---|---|
| $a_1$ | $a_2$ | $a_3$ | $b_1$ | $b_2$ |
| $a_4$ | $a_5$ | $a_6$ | $b_3$ | $b_4$ |
| | | | $b_5$ | $b_6$ |

mi dà una matrice $2 \times 2$ con i valori:

| | |
|---|---|
| $a_1 * b_1 + a_2 * b_3 + a_3 * b_5$ | $a_1 * b_2 + a_2 * b_4 + a_3 * b_6$ |
| $a_4 * b_1 + a_5 * b_3 + a_6 * b_5$ | $a_4 * b_2 + a_5 * b_4 + a_6 * b_6$ |

È quello che succede se:

```
>> z = v(1:4)

z =

    1    2    100    4

>> w*z

ans =

    10    20    1000    40
    20    40    2000    80
    30    60    3000    120
    40    80    4000    160

>>
```

**Prodotto puntuale** : .* (punto per punto), ovvero moltiplica elemento per elemento, pixel per pixel, lasciandoli al loro posto.

**Valore massimo:** `max(v)` mi ritorna il valore massimo fra i numeri del vettore. Per visualizzare le varianti, basta scrivere `max` e aprire la parentesi tonda sinistra e in sovrimpressione uscirà la lista delle varianti. In questo caso:

```
>> max(v)

ans =

    100

>>
```

**Somma dei valori:** `sum(v)` mi ritorna il valore totale della somma dei numeri del vettore. Questo, come il comando precedente, ha un comportamento diverso se si tratta di vettori o di array. In questo caso:

```
>> sum(v)

ans =

    118

>>
```

**Creare matrici di numeri random:** `randi(r, l, c)` mi ritorna un array di `l` righe e `c` colonne di numeri scelti randomicamente in un range ampio `r`.

```
>> v = randi(5,1,10)

v =

    1   5   1   4   5   5   1   2   2   5

>>
```

**Trovare un valore uguale ad un numero dato in un array:** mi ritorna un vettore booleano che MatLab chiama *vettori logici* che mi dice dove la condizione che ho espresso è vera, ovvero il valore del vettore è uguale al numero dato quando c'è 1.

```
>> v == 5

ans =

    1x10 logical array

    0   1   0   0   1   1   0   0   0   1

>>
```

Non è male perché è una semplice riga di istruzione, mi evita cicli espliciti e altra roba.

**Trovare i valori maggiori di un numero dato:** mi ritorna un *vettore logico* che mi dice dove la condizione che ho espresso è vera.

```
>> v>3

ans =

    1x10 logical array

    0   1   0   1   1   1   0   0   0   1

>>
```

Sono operazioni molto utili. E combinabili!

```
>> sum(v>3)

ans =

    5

>>
```

Mi conta quanti sono gli 1 del vettore logico.
Posso usare variabili di appoggio:

```
>> x = v>3

x =

    0   1   0   1   1   1   0   0   0   1

>> v

v =

    1   5   1   4   5   5   1   2   2   5

>>
```

**Selezionare solo alcuni valori:** `>> v(x)`

```
ans =

    5   4   5   5   5

>>
```

Praticamente mi seleziona solo i valori di `v` che hanno 1 nel *vettore logico* `x`.
**Molto utile per creare le maschere!** Posso indicizzare il mio vettore anche usando un altro vettore di numeri:

```
>> v([1,3,4,6])

ans =

    1   1   4   5

>>
```

Questo invece mi ritorna i valori di `v` che mi interessano, che ho chiesto esplicitamente.

Posso selezionare solo i valori che hanno 1 nel vettore logico anche in un altro modo:

```
>> find(x==1)

ans =

    2   4   5   6   10

>>
```

Questo mi ritorna gli *indici* delle cellette di `x` che mi soddisfano questa condizione.
Posso ovviamente poi usarlo per ottenere i corrispondenti valori di `v`.

```
>> v(find(x==1))

ans =

    5   4   5   5   5

>>
```

Notare che mi produce lo stesso risultato della sua variante precedente, `v(x)`. Queste due istruzioni sono equivalenti.

### Sintassi e operazioni sulle matrici

Possiamo istanziare una matrice andando a memorizzare tutti i valori che vanno salvati nelle righe e nelle colonne. Le operazioni sono pressoché le stesse che abbiamo visto per i vettori, solo che ora abbiamo due dimensioni e quindi due indici, uno per le righe e uno per le colonne.

**Riempire una `3x2` a mano:** `>> a=[1,2;3,4;5,6]`

```
a =

    1   2
    3   4
    5   6
```

```
>>
```

**Trovare il valore di una specifica celletta:** ad esempio terza riga prima colonna,

```
>> a(3,1)

ans =

    5

>>
```

**Visualizzare un'intera riga:** ad esempio la seconda

```
>> a(2,1:end)

ans =

    3   4

>>
```

In alternativa, ":" indica "tutti gli elementi", quindi:

```
>> a(2,:)

ans =

    3   4

>>
```

**Visualizzare un'intera colonna:** funziona come con le righe, ad esempio la prima

```
>> a(:,1)

ans =

    1
    3
    5

>>
```

**Matrice trasposta:** `>> a'`

```
ans =

    1   3   5
    2   4   6

>>
```

**Creare matrici di comodo:** utili per creare matrici su cui computare operazioni:

```
>> zeros(3,3)

ans =

    0   0   0
    0   0   0
    0   0   0

>> ones(5,5)

ans =

    1   1   1   1   1
    1   1   1   1   1
    1   1   1   1   1
    1   1   1   1   1
    1   1   1   1   1
```

```
>>
```

**Creare matrici randomiche:** abbiamo visto prima il comando rand, che mi crea numeri randomici double. Se voglio creare una matrice intera, posso usare il comando randi, che mi crea una matrice intera randomica.

```
>> m=rand(5,5)

m =

    0.4314  0.1361  0.8530  0.0760  0.4173
    0.9106  0.8693  0.6221  0.2399  0.0497
    0.1818  0.5797  0.3510  0.1233  0.9027
    0.2638  0.5499  0.5132  0.1839  0.9448
    0.1455  0.1450  0.4018  0.2400  0.4909

>>
```

**Creare matrici inverse:** `>> inv(m)`

```
ans =

   -9.6261   -0.7953   -70.5587    84.5354   -24.6859
    5.3676    1.3707    43.2380   -50.8728    13.6995
    7.3422    0.6401    42.8965   -51.9585    14.8125
   -1.0754    0.7834     6.1327   - 9.5106     7.8621
   -4.2154   -1.0759   -29.9599    37.1396   -10.6575

>>
```

**Determinante:** restituisce il suo valore

```
>> det(m)

ans =

    -0.0014
```

```
>>
```

**Diagonale:** riporta i valori posizionati sulla diagonale principale della matrice

```
>> diag(m)

ans =

    0.4314
    0.8693
    0.3510
    0.1839
    0.4909

>>
```

**Sottomatrici:** riporta solo i valori della sottomatrice scelta, ad esempio quella fra seconda e terza riga, terza e quarta colonna

```
>> m(2:3,3:4)

ans =
0.6221  0.2399  0.0497
0.1818  0.5797  0.3510  0.1233

>>
```

# Capitolo 2

# Laboratori Elaborazione delle Immagini 2023-34

## 2.1 Lab1 13 Ottobre 2023

### 2.1.1 Il negativo di un'immagine

Serve per rimappare un'immagine. Lo faccio sottraendo la prima immagine a 255 e asegnando il tutto ad una seconda immagine.

```
>> im1 = imread('path\img.format');
>> im2 = 255 - im1;
>> imshow(img2);
```

### 2.1.2 Mostrare a schermo due immagini

```
>> figure(i);
>> subplot(1,2,1);
>> imshow(im1);
>> subplot(1,2,2);
>> imshow(im2);
```

**figure;** : crea una finestra, quando ne serve più di una si aggiunge `(i)` con un indice a scelta.
Se non passiamo niente in input, MatLab indicizza da solo, altrimenti glielo diciamo noi l'indice.
Rendo attiva la finestra richiamandola con l'indice scelto.

**subplot(r, c, q);** : `subplot(n° righe, n° colonne, n° quadrante)`, rende attiva la `figure` e divide la finestra in riquadri (puoi fare più righe

($r > 1$) e più colonne ($c > 1$)) e `q` dice in quale quadrante (della riga scelta con `r`) mettere l'immagine.

Poi abbiamo creato `im3` come somma di `im1` e `im2`, cosa esce? Un'immagine tutta bianca, perché si sommano tutti i valori matematici e sarà tutto 255. Poi, per mostrarla a schermo, se faccio solo `imshow(im3)` andrà a sovrascrivere quella negativa, perché non ho specificato che serve una nuova figure e quindi usa l'ultima attiva. Serve `figure;` per averne una nuova.

### 2.1.3 Pulizia dello schermo

Ogni volta che eseguo lo script mi si aprono finestre di figure.

`close all;` : chiude in automatico tutte le finestre.

`clear all;` : cancella in automatico tutte le variabili.

### 2.1.4 Modificare size immagini

```
imresize(clouds, size(moon))
```

### 2.1.5 Le maschere binarie

Mi individuano una zona di interesse (in questo caso "dove vivono le nuvole") da togliere poi all'immagine originale.
Come? `treeshold` mi ritorna un vettore logico, che posso usare come maschera binaria. Per ora la soglia la troviamo a mano.
Abbiamo provato T = 0.5, ma non bastava. Abbiamo abbassato a 0.25, che poteva andare per quanto riguarda la zona di interesse, ma abbiamo abbassato ancora a 0.125 che mi ha preso ancora un po' di pixel.
Spoiler alert: alla fine abbiamo rimesso 0.25 perché la maschera aveva preso troppi pixel scuri e quando si zoommava sulla foto finale era tutta rovinata con bordi scuri etc.
Abbiamo provato 0 ed è diventato troppo, ha mostrato un effetto quadrettato che mi mostra artefatti/rumore che è tipico della compressione JPEG.
0.125 va più che bene, poi la usiamo per togliere un pezzetto di immagine da quella della Luna.

$1 - mask_clouds$ mi ritorna la maschera binaria invertita, ovvero dove c'è 1 mette 0 e viceversa.

.∗ moltiplicazione punto punto, dove nella maschera invertita ho 0, si azzera il pixel corrispondente nell'immagine.

Se volessi una maschera più smooth? Così che si nasconda il passaggio tra nuvola e sfondo della nuvola? Applico un filtro di smooth sull'immagine delle nuvole. Non tolgo totalmente il rumore, ma lo riduco localmente.

#### Immagini double

Prima mi sono dimenticata di segnare che spesso serve avere le immagini in forma double, si fa tipo

```
moon = im2double(imread("Lab1\moon.jpg"));
```

usando quindi il comando

```
im2double();
```

che converte l'immagine da *uint8* in *double*.

### 2.1.6 Gli istogrammi

`imhist();` mi ritorna l'istogramma dell'immagine.
Dall'istogramma dell'immagine del cavallo vedo che è **ben contrastata**, ovvero uso tutti i valori del range di valori dispobile.
`histeq();` mi ritorna l'immagine equalizzata.

### 2.1.7 Gamma correction

`imadjust();` mi ritorna l'immagine con la gamma corretta.
È un mapping non lineare, che mi permette di rimappare i valori di un'immagine.

### 2.1.8 Compiti a casa - 1

Sort of assignment, sono 4 e danno fino a 2 punti in più all'esame (quindi se ne consegno 2 è solo 1 punto in più). Per dire, il primo chiede di scrivere una funzione `myhistogram` che calcola l'istogramma di una immagine a livelli di grigio. Noi sappiamo che un istogramma cumulativo (uint8 tra 0 e 255 monocanale) è un vettore che va da 0 a 255 e mi conta le occorrenze di ogni valore.

## 2.2 Lab2 20 Ottobre 2023

### 2.2.1 Es1

Es.1, su immagine `contrast.jpg`.
Punto `a1.`: "Caricate l'immagine 'contrast.jpg' in una variabile im e scalatela tra 0 e 1." vuol dire `imread` e portarla in double con `im2double`.
Punto `b1.`: `crop` mi prende una sottomatrice della mia immagine, che va da $(x_1, x_2)$ a $(y_1, y_2)$, ovvero:`crop = img`$(x_1 : x_2, y_1 : y_2)$
Punto `c1.`: soglia T per trovare i valori scuri.
Ci applichiamo la statistica di **media**, bisogna saper capire quale statistica è meglio.
Ma MatLab opera per colonne; posso linearizzarlo, facendo `crop(:)`, così facendo la media avrà un solo valore.
$media + -k * deviazionestandard$ mi dà un intorno al picco dell'istogramma.
Abbiamo parzialmente automatizzato la soglia.
Punto `d1.`: due maschere light e dark che contengono rispettivamente le regioni chiare e scure di im usando la tecnica delle maschere.

### 2.2.2 Es2

#### Filtri lineari

Sono filtri spaziali che possono essere espressi con funzioni lineari. Maschera di convoluzione (?) che prende l'intorno di un pixel e fa la media pesata di tutti i pixel del vicinato. Abbiamo visto a lezione diversi filtri, i più diffusi quelli di smoothing che riducono il rumore (variazione pseudorandomica dell'immagine).

#### I comandi

`fspecial("filtro", valore)`: funzione che permette di generare diversi tipi di filtri.
In un filtro di media voglio fare la media di `n` valori, se ne ho 25 peso i valori per $\frac{1}{25}$.
`imfilter(input, filtro)`: applica un filtro ad un input.
Più aumento il filtro (5 -¿ 11 -¿ 21) più l'immagine si sfoca ma si crea un bordo perché dove non esistono i pixel vengono messi a 0 (quindi la convoluzione sui bordi mi crea bordi neri).
Ci sono diverse opzioni per evitare ciò: replica i valori, o pescali da in giro, o boh.
Ora dobbiamo creare delle immagini che siano la differenza tra immagine

originale e filtrata in valore assoluto. Ma sono array di valori, non usiamo imshow. Usiamo `imagesc` (image scaling) che tratta gli array come fossero immagini e al tempo stesso le tratta con colori fasulli in modo da vedere le differenze. Però spiattella le immagini, perciò aggiungo `axis image` che mi mantiene le proporzioni. `colorbar` mi aggiunge una scala di valori per capire quanto è grande la differenza e dove, in base ai colori fasulli che mi ha messo. E dove lo vedo? Dove ci sono alte frequenze, ovvero nei dettagli, che è dove opera il filtro di smoothing.
Quindi questi valori evidenziati dalle immagini colorati mi dicono **dove ho perso informa**
cosa che fanno i filtri di smoothing (sono da tonare adeguatamente).
Tutto dipende dalla forza del rumore: se è poco evidente posso limitare la forza del filtro di smoothing.

### 2.2.3 Es3

Copio e incollo l'es2 perché riciclo il codice in quanto cambio solo il filtro che uso.

#### Il filtro gaussiano

Due parametri: dimensione spaziale del filtro e la $\sigma$.
La sigma calcola quanto sono dispersi i valori della gaussiana rispetto alla sua media.
Essendo distribuzione di probabilità, la sua area?boh senti la registrazione.
I pixel più vicini a quello da calcolare hanno più peso rispetto a quelli più lontani.
Come faccio filtro spaziale su distribuzione gaussiana? Vado a campionarla. Ha aperto la F3, poi la F5, poi la F7. Nella 5 la sottomatrice centrale è praticamente quella del 3. Stessa cosa per F7. Va da sé che anche F9. Che è succ? MatLab ha preso una funzione gaussiana di valori discreti e ha campionato la gaussiana, trovato certi valori utili.
Una volta trovata la forma della gaussiana (sigma) bisogna tonare la gaussiana: perciò non ha senso andare troppo oltre un certo valore (in questo caso 5 perché i pixel intorno sono praticamente a 0).
Quando creo un filtro gaussiano a due parametri, la dimensione spaziale del filtro e la sigma, questi due parametri non sono indipendenti ma sono legati tra loro. Di solito, si la dimensione di un filtro Gaussiano si ricava dalla seguente formula che garantisce di avere tutti i valori utili del filtro:

$$N = 1 + 2 \times [2.5 \times \sigma]$$

`figure, surf(filtro)`: mi fa vedere la forma del filtro.
Il filtro gaussiano è molto meglio del filtro di smooth per smoothare l'immagine perché è tonabile. A parità di dimensione, il gaussiano rovina meno l'immagine del filtro di smooth.

### 2.2.4 Es4

Parte dicendo "Create un filtro Gaussiano G di dimensione 21x21 con la Sigma adatta." questo lo faccio con la formula di prima

$$N = 1 + 2 \times [2.5 \times \sigma]$$

assegnando 21 a $N$ e risolvendo per $\sigma$. Ottengo 4, il minimo sindacale.

### 2.2.5 Es5

Cosa succede con immagini (Lawrence1) dove c'è palesemente del rumore gaussiano? Rumore distribuito ed evidente ma che non ci dà tanto fastidio.
La riduzione del rumore dipende sempre dal task finale: per dire va bene per ambito cinematografico (la pellicola crea naturalmente questo tipo di rumore quindi è perfetta) mentre per tipo boha ltri scopi mi dà fastidio e lo voglio togliere e avrebbe completamente senso.
Lawrence2 invece salt&pepper, ovvero rumore impulsivo.
S&P è un rumore che non può essere rimosso da un filtro normale, è randomico quindi non segue una legge di distribuzione (perciò i filtri lineari non sono utilizzabili). Il rumore gaussiano può essere rimosso da un filtro gaussiano (o di media, ma gaussiano meglio).

### 2.2.6 Es6

Facciamo noi, vediamo come l'uso di filtri spaziali semplici possono tornare utili per cose tricky.

### 2.2.7 Es7

Combinazione di filtri lineari = filtro lineare.
La convoluzione è un filtro lineare.
Riprendiamo es4 e out diventa = $glamour@(I + G) * 0.5$; combino op di convoluzione.

## 2.3 Gli assignment

nearest neighboor campiono le righe e le colonne e replico qualche valore, il problema è decidere quale.
fare a mano il filtro di convoluzione. Anche abusando di cicli for. Nel caso filtro fuori dall'immagine, assumo valore 0 o scelgo un altro meccanismo (tipo scelgo solo una porzione di immagine su cui riesco a lavorare quinid output immagine più piccola).

terzo, filtro non lineare. Non matrice di pesi ma algoritmo. Per ogni pixel e intorno dare in output il massimo fra i valori dell'intorno. Non lineare perché serve un algoritmo vero e proprio e non ci sono funzioni lineari.

chapterLab3

## 2.4 Filtro mediano

### 2.4.1 Es.1

Per ogni pixel da processare, prende l-intorno lo analizza e valuta il filtro da applicare.
Non lineare.
Perfetto per rimuovere rumore s&p. Non genera nuovi valori nell'immagine ma rigenera quelli già presenti.
Rimane un pixel di rumore, come mai? Ale ha detto la risposta ma non mi ricordo.
Il f.m. è un buon filtro, ma attenzione: a seconda della dimensione aumento o diminuisco la probabilità di beccare un filtro di rumore.
Con 5x5 invece di 3x3 sparisce il pixel bianco.
Ad aumentare il filtro spariscono i pixel residui di rumore ma si perdono dettagli dell'immagine.
Filtro troppo alti me la rendono "cartoonizzata", mantiene i bordi ma sfoca le singole regioni.

## 2.5 Immagini a colori

### 2.5.1 Spazio RGB

#### Es.2

Non sono array **bidimensionali** ma **tridimensionali**.
Un canale colore è la risposta di un filtro che seleziona una certa regione della

lunghezza d'onda del visibile (ovvero quanto il dispositivo di acquisizione è stato sensibile nell'acquisizione delle lunghezze elettromagnetiche). Queste risposte sono convertite in numeri.
Es.: rosso, quanto è presente l'onda elettromagnetica della regione che corrisponde al rosso? Si comporta (***in ogni singolo canale***) come il livello di grigio, perché mi calcola l'intensità del colore di quel canale (banalmente, noi lo traduciamo con "quanto colore c'è").
Come facciamo a prendere ogni singolo canale? "Facciamo a fette" l'immagine.

```
R = balls(:,:,1);
G = balls(:,:,2);
B = balls(:,:,3);
```

Perché nella foto del matrimonio il piattino nel canale G è grigio scuro e non bianco? Perché i colori non sono esattamente così come li percepiamo noi, il verde eventualmente lì non è verde puro e quindi il canale G non è prominente in quella regione rispetto agli altri colori.
NB: il bianco non è assenza di informazione, ma esattamente il contrario.
Perché il canale blu non è tutta nera? Perché il blu è contenuto in altri colori, tipo bianco del tovagliolo, grigio del tavolo, ... Invece c'è assenza totale di blu ad es. dove c'è l'arancione per dire.
TIPICA DOMANDA DEGLI SCRITTI.
Ricorda:

**nero puro:** assenza di tutta l'informazione del colore.

**bianco puro:** presenza di tutta l'informazione del colore.

**grigio:** presenza di informazione colore di tutti e tre i canali ma in quantità equa.

### 2.5.2 Spazio HSV

Ma non basta RGB? Dipende dall'utilizzo che ne facciamo. Danno informazioni che da RGB non posso avere. Ce ne sono decine di spazi colori, noi a teoria ne abbiamo visti alcuni.
HSV codifica dello spazio colore che dà informazioni su tinta, saturazione e valore.

**Es.3**

YCbCr
Y canale informazione luminanza, ovvero la quantità di luce che c'è in un

punto, quanto è forte quel pixel in quella posizione.
Cb dice quanto è blu il colore.
Cr dice quanto è rosso il colore.
Cb e Cr dicono contemporaneamente quanto è blu vs quanto verde c'è. Parlano di un rapporto fra regioni.
`imshow` considera quello che gli do come un'immagine di livelli di grigio o RGB. Quindi reinterpreta i valori che gli passo con colori sfalsati. Noi però abbiamo YCbCr, quindi l'immagine non ha senso. I singoli canali invece vanno bene.
Blu verso giallognolo, rosso verso verde.
Si possono processare le immagini indipendentemente dall'intensità della luce. Nell'equazione (prodotto matrice per vettore) il primo vettore, a sinistra del +, è l'offset. Ho tutto traslato in un range positivo (lo 0, l'assenza di colore, è pura Y e Cr e Cb a 128, perché da una parte andiamo sul blu e dall'altra sul giallo).

HSV
Trasformazione non lineare di RGB.
H (angolo, codificata come cerchio) - tinta - percezione del colore.
S (valore lineare) - saturazione - quanto è "pieno".
V (valore lineare) - value - intensità luminosa.

Big big nope: non fare istogramma per H. Il valore a 0 e a 360 sono uguali (siamo in un cerchio), bisognerebbe fare un istogramma circolare `(?!)`.

### 2.5.3 Rumore nelle immagini a colore

**Es.4**

Spacchetto nei tre canali poi ricostruisco l'immagine con i canali filtrati tramite il comando:

```
R2 = imfilter(R,GF);
...
im = cat(3,R2,G2,B2);
```

3 dimensioni perché vuole in quante direzioni e quali.
Posso fare direttamente

```
face3 = imfilter(face,GF);
```

Perché le immagini a colori vengono filtrate per canale in automatico.

**Es.5**

Rumore spike, sono valori molto netti. Rumore s&p colorato.
Applichiamo filtro mediano (quello che si usa nel caso di s&p). Problema: per matlab è bidimensionale il filtro mediano. Dobbiamo spacchettare nei canali.
Plot-twist: esiste `medfilt3` che fa il filtro mediano in 3 dimensioni.

**Es.6**

Non ho scritto niente.

**Es.7**

Filtri di edge, lineari, nella convoluzione otteniamo sia val pos che neg perché approssimano le derivate.
A noi interessa di più l'effetto finale.
Esiste la funzione `edge` che applica un filtro di gradiente e una sogliatura e mi ritorna un'immagine binaria con i bordi.
Il filtro di Prewitt è un filtro derivativo del 1 ordine (derivata prima).
Il filtro di Sobel è un filtro derivativo del 2 ordine (derivata seconda).
Così soglia automatica:

```
im = im2double(imread('mondrian.jpg'));
gray = rgb2gray(im);
ep = edge(gray, "prewitt");
es = edge(gray, "sobel");
figure,
subplot(2,2,1), imshow(im), title('A colori')
subplot(2,2,2), imshow(gray), title('B&N')
subplot(2,2,3), imshow(ep), title('Prewitt')
subplot(2,2,4), imshow(es), title('Sobel')
```

Giocando con la soglia:

```
im = im2double(imread('mondrian.jpg'));
gray = rgb2gray(im);
ep = edge(gray, "prewitt");
es = edge(gray, "sobel");
figure,
subplot(2,2,1), imshow(im), title('A colori')
subplot(2,2,2), imshow(gray), title('B&N')
subplot(2,2,3), imshow(ep), title('Prewitt')
subplot(2,2,4), imshow(es), title('Sobel')
```

Canny invece è un algoritmo, non filtro lineare, e difatti è il migliore finora. QUANDO SI FA IL PROCESSING DELLE IMMAGINI, ATTENZIONE ALL'ORDINE CON CUI SI FANNO LE COSE! Prima si toglie il rumore, poi si fanno i fltri. Prima i filtri, poi gli edge. Eventualmente, poi tolgo gli edges che non mi interessano.