Laboratorio 2 - MATLAB

Lanzarotti Raffaella

Indicizzazione di vettori

- `v = [1 3 5 7 9];`
- `v(2)` % accesso a un elemento
- `w = v';` % trasposta
- `a =v(1:3);` % accesso a un *blocco*
- `b =v(4:end);` % accesso a un *blocco*
- `c = v(1:2:end);` *% blocco non contiguo*
- `d = v(end:-2:1);` *% step negativo*
- `e = v([1 4 5]);` *%uso un vettore per %indicizzare un altro vettore*

Indicizzazione di matrici

- `A = [1 2 3; 4 5 6; 7 8 9];`
- `A(2, 3);` % accesso a un elemento
- `W = A';` % trasposta
- `c3 = A(:,3);` % accesso 3° colonna
- `r2 = A(2, :);` % accesso 2° riga
- `B = A(1:2,:);` *% blocco delle prime 2 % righe*
- `B2 = A(2:3,2:3);` *% blocco 2 x2*
- `B3 = A(2:3,end:-2:1);` *% step negativo*

Indicizzazione di matrici (cont.)

- `E = A([a b], [c d]);`
  `% seleziona gli elementi:`
  `% A(a,c),A(a,d),A(b,c),A(b,d)`

- `D = logical([1 0 0; 0 0 1; 0 0 0]);`
- `Atrue = A(D);` `% seleziona gli elem =1 di D`
- `V = A(:);` `% trasforma la matrice in`
  `% colonna, accodando le`
  `% colonne di A`
- `s = sum(A(:));` `% equivale a sum(sum(A));`

Le img in MATLAB

- Image Processing Toolbox: collezione di funzioni specializzate per operare su immagini
- Rappresentazione di un'immagine con una matrice (o più) di valori:
  - Ogni elemento corrisponde ad un singolo pixel
  - Le coordinate (i,j) di un pixel rappresentano (#riga,#colonna)
  - Il valore di un pixel può rappresentare dati differenti a seconda del tipo di immagine

Lettura e visualizzazione

- Lettura di un'immagine:

```
img_rgb = imread('trees.jpg');
```

- Visualizzazione:

```
figure
imshow(img_rgb);
```

- ...oppure:

```
figure
imshow('trees.jpg');
img_rgb = getimage;
```

Salvataggio

- Lettura di un'immagine in formato jpg:

```
img_rgb = imread('trees.jpg');
```

- Salvataggio e conversione in formato bitmap:

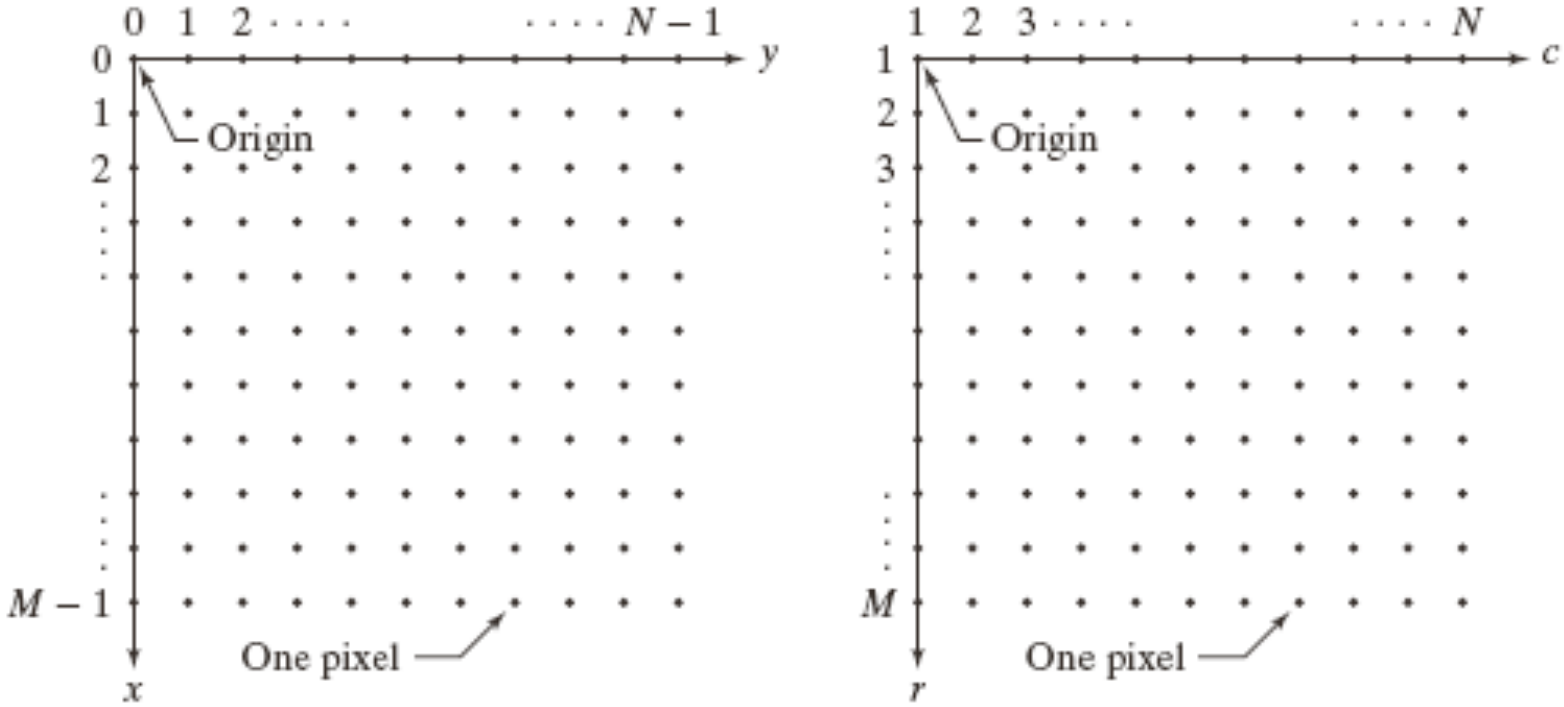
```
imwrite(img_rgb,'img_rgb.bmp',
'bmp');
```

Sistema di riferimento

- Un pixel in un'immagine m x n viene indicizzato con due valori (i,j):
  - i = # di riga da alto a basso (1,2,3, …, m)
  - j = # di colonna da sx a dx (1,2,3, …, n)

a b

FIGURE 2.1 Coordinate conventions used (a) in many image processing books, and (b) in the Image Processing Toolbox.



Esempio su img

- `f = imread('lumaca.jpg');`
- Es: Immagine capovolta verticalmente:
  - `fp = f(end:-1:1, :);`
- Es: Immagine capovolta orizzontalmente
  - `fp = f(:, end:-1:1);`
- Es: estrarre il quarto quadrante dell'img
  -
```
Q = f( round(end/2):end,
       round(end/2):end);
```

Matrici standard

- `zeros(M,N)`
- `ones(M,N)`
- `true(M,N)`
- `false(M,N)`
- `magic(M,N)`
- `rand(M,N)`
- `randn(M,N)`

Data class in MATLAB

- *double, single:*
  floating point (8 o 4 byte per elemento)
- *uint8, uint16, uint32*:
  interi senza segno (8,16 o 32 bit ➔ 1,2,4 byte)
- *int8, int16, int32:*
  interi con segno (8,16 o 32 bit ➔ 1,2,4 byte)
- *char*:
  carattere (2 byte per elemento)
- *logical*:
  valori logici 0 o 1 (1 byte per elemento)

Conversioni tra data class

- **B = data_class_name(A);**
- Es:
  - se A è uint8 ➔ B = double(A)
    - mantiene gli stessi valori
  - Se A è double ➔ B = uint8(A)
    - I valori <0 vengono portati a 0
    - I valori >255 vengono portati a 255
    - Le parti decimali vengono troncate
  - Se A è uint8 ➔ B = logical(A)
    - I valori diversi da 0 vengono posti a 1
    - I valori = 0 restano a 0 (ma 0 logico)

Conversioni tra classi e tipi (1/4)

- **B =im2uint8(A)** ➔ effettua le scalature necessarie per riconoscere B come un'img valida di tipo uint8
- Es:

```
f = [-0.5  0.5
      0.75  1.5]
g = im2uint8(f);
```

Conversioni tra classi e tipi (2/4)

- **B =mat2gray(A)** →
  input: numerico o logico,
  output: double scalato nel range [0 1])

- vedi anche:

  B =mat2gray(A, [Amin, Amax])

Conversioni tra classi e tipi (3/4)

- **B =im2double(A) ➔**
  - output di tipo double.
  - Se input non double, allora l'output è scalato nel range [0 1] (dividendo i valori per 255).
  - Se l'input è double, output invariato (per averlo nel range [0 1] usare mat2gray

- Es:

```
h = uint8([25 50; 128 200]);
g = im2double(h);
```

Conversioni tra classi e tipi (4/4)

- **g =im2bw(f, T) ➔**
  - f: img in input, T: soglia (default: 0.5)
  - output di tipo logical: 0 per i pixel di *f* con intensità < T 1 gli altri
- Es: `f = [1 2;3 4];`
  vogliamo un'img binaria t.c. 1 e 2 diventino 0 e 3 e 4 diventino 1:

```
g  = mat2gray(f);
gb = im2bw(g);
```

`gb=im2bw(mat2gray(f));`
NESTED

Oppure

```
gb = f > 2;
```

Aritmetica tra img (overflow/underflow/rounding)

- Matlab supporta operazioni aritmetiche standard tra immagini, quali **somma**, **sottrazione**, **moltiplicazione** e **divisione**
- Il risultato di un'operazione aritmetica più comportare
  - Un valore esterno al range rappresentabile con il tipo di dato del pixel → forzato al valore estremo
    - (uint8) 340 → 255
    - (uint8) -34 → 0
  - un valore frazionario → arrotondamento
    - (uint8) 105.5 → 106

Interpolazione in Matlab (1/4)

- Riduzione di un'immagine

```
img_gray_reduced = imresize(img_gray,
0.25);
```

- Ingrandimento con interpolazione Nearest-Neighbor

```
img_gray_enlarged =
imresize(img_gray_reduced, 4,
'nearest');
```

Interpolazione in Matlab (2/4)

- Ingrandimento con interpolazione Bilineare

```
img_gray_enlarged =
imresize(img_gray_reduced, 4,
'bilinear');
```

- Ingrandimento con interpolazione Bicubica

```
img_gray_enlarged =
imresize(img_gray_reduced, 4,
'bicubic');
```

- Ridimensionamento con interpolazione Bicubica

```
img_gray_resized = imresize(img_gray,
[200 500], 'bicubic');
```

Interpolazione in Matlab (3/4)

- Rotazione con interpolazione N-N e dimensioni output costanti

```
img_gray_rotated = imrotate(img_gray, 35, 'crop');
```

- Rotazione con interpolazione N-N

```
img_gray_rotated = imrotate(img_gray, 35);
```

- Rotazione con interpolazione Bilineare

```
img_gray_rotated = imrotate(img_gray, 35, 'bilinear');
```

Interpolazione in Matlab (4/4)

- Rotazione con interpolazione Bicubica

```
img_gray_rotated = imrotate(img_gray,
35, 'bicubic');
```

Esercizio 1

- Scrivere la funzione:

  `function [subImg, varSubImg] = imagePartition(img, n)`

  input:

  -img: un'immagine a livelli di grigio

  - n: il numero di partizioni verticali e orizzontali (equispaziate) che vogliamo realizzare.

  output:

  - Visualizzi le sotto-parti avvalendosi della f. subplot
  - subImg: sotto-img con varianza massima
  - varSubImg: la varianza massima stessa

- Si scriva anche un **`main`** che:
  - apra un'img [Hint: usa imagein1 per esempio],
  - chieda il n. di partizioni orizzontali e verticali,
  - chiami la funzione **`imagePartition`**
  - visualizzi la sotto-img restituita in output

Esercizio 2

- Data la sotto-img restituita dalla funzione “**imagePartition**”, riportarla alle dimensioni originarie usando le 3 possibili interpolazioni.
- Confrontare qualitativamente i risultati

Esercizio 3 (extra)

- si scriva (su carta) una singola matrice 2x2 T che, se applicata a vettori [x, y]:
  - per prima cosa scali la direzione x di un fattore 1.0 e la direzione y di un fattore 0.5 e
  - e poi applichi una rotazione di 45° CCW
  - Hint: si pensi alle due matrici di scaling S e di rotazione R separatamente e poi le si moltiplichi opportunamente per ottenere T
- partendo dallo script **`imageTransform`**, sostituire la matrice T identità del testo con la matrice da voi trovata per applicare la trasformazione all'immagine.
- ESEGUITE e "giocate" con i parametri…

Esercizio 4 (extra)

- Aprire un'immagine e convertirla a livelli di grigio e in formato double.

A. avvalendosi della funzione MATLAB **`svd`** estrarre i primi 10 valori singolari e plottarli (in ascissa si avrà il loro ranking). Cosa potete notare da questo plot?
   Hint: il comando **`diag`** vi potrebbe essere utile.

B. Verifica che si può ricostruire l'immagine di partenza e visualizzarla usando le tre matrici prodotte dal comando **`svd`** (nota che il comando **`svd`** ritorna *V* e non *V'*).

C. Ottenere la compressione usando solo i primi *k* valori singolari e i corrispondenti vettori singolari destri e sinistri. Sia *k=10, 50 e 100*. Ricostruire e visualizzare l'immagine compressa per i tre distinti valori di *k*. Cosa si può osservare?

D. Invece di trasmettere l'immagine originale, si può applicare la compressione **`svd`** e trasmettere solo i primi *k* valori singolari e i corrispondenti vettori singolari destri e sinistri. Questo è conveniente per piccoli valori di *k*. Data un'immagine *nxm* determinare il massimo valore di *k* entro il quale si ha un vantaggio di compressione spaziale e stampare un messaggio a video che indichi tale estremo superiore.

E. Invece di fissare *k* a priori, calcolarlo in modo tale da catturare il 95% dell'energia, espressa dai valori singolari nella matrice S.
   Hint: il comando **`cumsum`** vi potrebbe essere utile.