Informazioni consegna

- Aggiungere i componenti del proprio gruppo in questo form (https://forms.office.com/r/1WKBx4YgSH).
- **I gruppi possono essere composti da massimo 3 persone. Nel caso si intenda fare il progetto da soli bisogna comunque aggiungere il proprio nome nel form.**
- Per la consegna è necessario caricare su Virtuale la relazione, il notebook Colab e le immagini generate.
- La consegna deve essere effettuata da un qualsiasi membro del gruppo e verrà automaticamente attribuita ai restanti componenti.
- Per i dettagli sulle tempistiche e le scadenze si faccia riferimento a Virtuale.

Deblur Immagini

Il problema di deblur consiste nella ricostruzione di un immagine a partire da un dato acquisito mediante il seguente modello:

$$b = Ax + \eta$$

dove b rappresenta l'immagine corrotta, x l'immagine originale che vogliamo ricostruire, A l'operatore che applica il blur Gaussiano ed η il rumore additivo con distribuzione Gaussiana di media 0 e deviazione standard σ.

Funzioni di blur

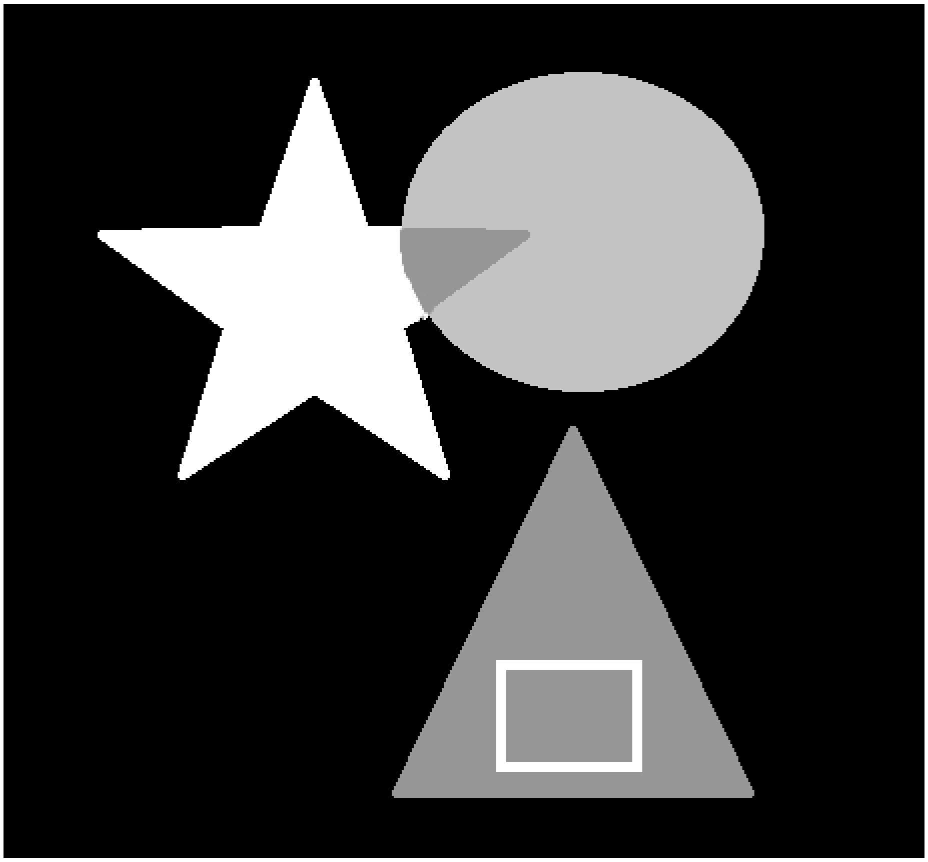
Le seguenti funzioni servono per applicare il blur di tipo gaussiano ad un'immagine.

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
from skimage import data, metrics
from scipy import signal
from numpy import fft

# Create a Gaussian kernel of size kernlen and standard deviation s
igma
def gaussian_kernel(kernlen, sigma):
    x = np.linspace(- (kernlen // 2), kernlen // 2, kernlen)
    # Unidimensional Gaussian kernel
    kern1d = np.exp(- 0.5 * (x**2 / sigma))
    # Bidimensional Gaussian kernel
    kern2d = np.outer(kern1d, kern1d)
    # Normalization
    return kern2d / kern2d.sum()

# Compute the FFT of the kernel 'K' of size 'd' padding with the ze
ros necessary
# to match the size of 'shape'
def psf_fft(K, d, shape):
    # Zero padding
    K_p = np.zeros(shape)
    K_p[:d, :d] = K

    # Shift
    p = d // 2
    K_pr = np.roll(np.roll(K_p, -p, 0), -p, 1)

    # Compute FFT
    K_otf = fft.fft2(K_pr)
    return K_otf

# Multiplication by A
def A(x, K):
  x = fft.fft2(x)
  return np.real(fft.ifft2(K * x))

# Multiplication by A transpose
def AT(x, K):
  x = fft.fft2(x)
  return np.real(fft.ifft2(np.conj(K) * x))
```

Generazione dataset

Generare un set di 8 immagini 512×512 in formato `png` in scala dei grigi che contengano tra i 2 ed i 6 oggetti geometrici, di colore uniforme, su sfondo nero.



1) Generazione immagini corrotte

Degradare le immagini applicando, mediante le funzioni riportate nella cella precedente, l'operatore di blur con parametri

- $\sigma = 0.5$ dimensione 5×5
- $\sigma = 1$ dimensione 7×7
- $\sigma = 1.3$ dimensione 9×9

ed aggiungendo rumore gaussiano con deviazione standard $(0, 0.05]$

In [ ]:

2) Soluzione naive

Una possibile ricostruzione dell'immagine originale x partendo dall'immagine corrotta b è la soluzione naive data dal minimo del seguente problema di ottimizzazione:

$$x^* = \arg\min_x \frac{1}{2} ||Ax - b||_2^2$$

La funzione f da minimizzare è data dalla formula $f(x) = \frac{1}{2}||Ax - b||_2^2$, il cui gradiente ∇f è dato da $\nabla f(x) = A^T Ax - A^T b$.

Utilizzando il metodo del gradiente coniugato implementato dalla funzione `minimize` calcolare la soluzione naive.

In [ ]:

3) Regolarizzazione

Per ridurre gli effetti del rumore nella ricostruzione è necessario introdurre un termine di regolarizzazione di Tikhonov. Si considera quindi il seguente problema di ottimizzazione.

$$x^* = \arg\min_x \frac{1}{2} ||Ax - b||_2^2 + \frac{\lambda}{2} ||x||_2^2$$

La funzione f da minimizzare diventa $f(x) = \frac{1}{2}||Ax - b||_2^2 + \frac{\lambda}{2}||x||_2^2$ il cui gradiente ∇f è dato da $\nabla f(x) = A^T Ax - A^T b + \lambda x$.

Utilizzando il metodo del gradiente coniugato implementato dalla funzione `minimize` ed il metodo del gradiente implementato a lezione, calcolare la soluzione del precendente problema di minimo regolarizzato per differenti valori di λ.

In [ ]:

4) Variazione Totale (Facoltativo)

Un altra funzione adatta come termine di regolarizzazione è la Variazione Totale. Data u immagine di dimensioni $m \times n$ la variazione totale TV di u è definit come:

$$TV(u) = \sum_i^n \sum_j^m \sqrt{||\nabla u(i,j)||_2^2 + \epsilon^2}$$

Per calcolare il gradiente dell'immagine ∇u usiamo la funzione `np.gradient` che approssima la derivata per ogni pixel calcolando la differenza tra pixel adiacenti. I risultati sono due immagini della stessa dimensione dell'immagine in input, una che rappresenta il valore della derivata orizzontale `dx` e l'altra della derivata verticale `dy`. Il gradiente dell'immagine nel punto (i, j) è quindi un vettore di due componenti, uno orizzontale contenuto in `dx` e uno verticale in `dy`.

Come nei casi precedenti il problema di minimo che si va a risolvere è il seguente:

$$x^* = \arg\min_x \frac{1}{2}||Ax - b||_2^2 + \lambda TV(u)$$

il cui gradiente ∇f è dato da

$$\nabla f(x) = (A^T Ax - A^T b) + \lambda \nabla TV(x)$$

Utilizzando il metodo del gradiente implementato a lezione, calcolare la soluzione del precendente problema di minimo regolarizzato per differenti valori di λ.

Per risolvere il problema di minimo è necessario anche calcolare il gradiente della variazione totale che è definito nel modo seguente

$$\nabla TV(u) = -div\left(\frac{\nabla u}{\sqrt{||\nabla u||_2^2 + \epsilon^2}}\right)$$

$$div(F) = \frac{\partial F_x}{\partial x} + \frac{\partial F_y}{\partial y}$$

$div(F)$ è la divergenza del campo vettoriale F, nel nostro caso F ha due componenti dati dal gradiente dell'immagine ∇u scalato per il valore $\frac{1}{\sqrt{||\nabla u||_2^2+\epsilon^2}}$.

Per calcolare la divergenza bisogna calcolare la derivata orizzontale $\frac{\partial F_x}{\partial x}$ della componente x di F e sommarla alla derivata verticale $\frac{\partial F_y}{\partial y}$ della componente y di F. Per specificare in quale direzione calcolare la derivata con la funzione `np.gradient` utilizziamo il parametro `axis = 0` per l'orizzontale e `axis = 1` per la verticale.

In [ ]:

```
eps = 1e-2

# Variazione totale
def totvar(x):
  # Calcola il gradiente di x
  dx, dy = np.gradient(x)
  n2 = np.square(dx) + np.square(dy)

  # Calcola la variazione totale di x
  tv = np.sqrt(n2 + eps**2).sum()
  return tv

# Gradiente della variazione totale
def grad_totvar(x):
  # Calcola il numeratore della frazione
  dx, dy = np.gradient(x)

  # Calcola il denominatore della frazione
  n2 = np.square(dx) + np.square(dy)
  den = np.sqrt(n2 + eps**2)

  # Calcola le due componenti di F dividendo il gradiente per il de
nominatore
  Fx = dx / den
  Fy = dy / den

  # Calcola la derivata orizzontale di Fx
  dFdx = np.gradient(Fx, axis=0)

  # Calcola la derivata verticale di Fy
  dFdy = np.gradient(Fy, axis=1)

  # Calcola la divergenza
  div = (dFdx + dFdy)

  # Restituisci il valore del gradiente della variazione totale
  return -div
```

In [ ]:

```
```

Relazione

1. Riportare e commentare i risultati ottenuti nei punti 2. 3. (e 4.) su un immagine del set creato e su altre due immagini in bianco e nero (fotografiche/mediche/astronomiche)
2. Riportare delle tabelle con le misure di PSNR e MSE ottenute al variare dei parametri (dimensione kernel, valore di sigma, la deviazione standard del rumore, il parametro di regolarizzazione).
3. Calcolare sull'intero set di immagini medie e deviazione standard delle metriche per alcuni valori fissati dei parametri.
4. Analizzare su 2 esecuzioni le proprietà dei metodi numerici utilizzati (gradiente coniugato e gradiente) in termini di numero di iterazioni, andamento dell'errore, della funzione obiettivo, norma del gradiente.