CN25 - Homework 4

Matteo Mazzetti 0001161552

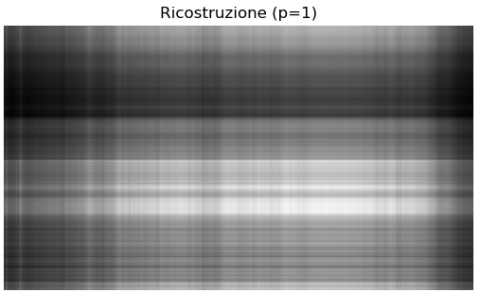
1 Compressione

In questo frammento di codice importiamo un'immagine(la pista di Spa-Francorchamps) in formato bmp e la comprimiamo dopo averla decomposta con SVD. I moduli python utilizzati nel codice che segue sono Numpy, Matplotlib, Skimage.color e dal modulo ProblemiInversi la funzione rel_err che si trova fra le utilities.

```
x=plt.imread("spaBMP.bmp")
x=rgb2gray(x)
nx,ny = x.shape
x=x/x.max()
U,S,Vt=np.linalg.svd(x)
p=50
if p>min(nx,ny):
    p=min(nx,ny)
U_p=U[:,:p]
S_p=np.diag(S[:p])
Vt_p=Vt[:p,:]
x_p = U_p @ S_p @ Vt_p
Er=utilities.rel_err(x_p,x)
c_p=((1/p)*min(nx,ny))-1
print(f"Errore relativo = {Er}")
print(f"Fattore di compressione = {c_p}")
```

Originale

Immagine	Errore relativo	Fattore di compressione	Dimensioni file
Originale	/	/	165KB
Compressa p=20	0.1974904789407892	71.95	116KB
Compressa p=100	0.13616883194908386	14.389999999999999	144KB
Compressa p=500	0.051427370290590114	2.878	156KB
Compressa p=1	0.40175778171716614	1439.0	74KB



2 Image deblur

In questa sezione ci poniamo l'obiettivo di rimuovere al meglio il rumore da un'immagine: calcoleremo la soluzione naive con Conjugate Gradient Least Squares (CGLS), mentre calcoleremo la soluzione regolarizzata con Tikhonov e con Total Variation.
Nel seguente modo sono stati importati i moduli per i frammenti di codice che seguiranno:

```
from ProblemiInversi import operators, solvers, utilities
import numpy as np
import matplotlib.pyplot as plt
from skimage.color import rgb2gray
```

Figure 1: Immagine originale senza rumore

2.1 CGLS

```
x=plt.imread("redbullBMP.bmp")
x = rgb2gray(x)
nx, ny = x.shape
x = x / x.max()
kernel = utilities.gaussian2d_kernel(k=7, sigma=2)
A = operators.ConvolutionOperator(kernel)
plt.imshow(kernel, cmap='hot')
plt.axis('off')
plt.show()
y = A(x)
y_delta = y + utilities.gaussian_noise(y, noise_level=0.05)
cgls_solver = solvers.CGLS(A)
x0 = np.zeros_like(x)
kmax = 30
tolf = 1e-8
tolx = 1e-8
x_cgls = cgls_solver.solve(y_delta, x0, kmax, tolf, tolx)
print('ER',utilities.rel_err(x_cgls,x))
print('PSNR',utilities.psnr(x_cgls,x))
print('SSIM',utilities.ssim(x_cgls,x))
```

A seguito dell'esecuzione di questo snippet e al variare dei parametri del kernel e del rumore possiamo costruire questa tabella riassuntiva dei test:

Test	kmax	Rumore	Kernel	Errore Relativo	PSNR	SSIM
A.	30	0.05	G(11, 3.5)	0.5526840988788954	8.99533240405611	0.11652658525104359
B.	30	0.10	G(11, 3.5)	1.150395334612094	2.6279295644348375	0.09340786062118511
C.	30	0.025	G(11, 3.5)	0.2633299693463085	15.434866028318428	0.18777693472990542
D.	30	0.05	G(7, 2)	0.6803169799003682	7.19064558859978	0.10283801810875974
E.	30	0.10	G(7, 2)	1.3903208494109554	0.9825710891374495	0.08145145125558878
F.	30	0.025	G(7, 2)	0.32686701980093863	13.557449732253279	0.1711491939706887
G.	50	0.05	G(11, 3.5)	0.8648559122005423	5.105996627512975	0.0980697768340959
H.	150	0.05	G(11, 3.5)	2.509269585430413	-4.146074652120268	0.07073271082152538

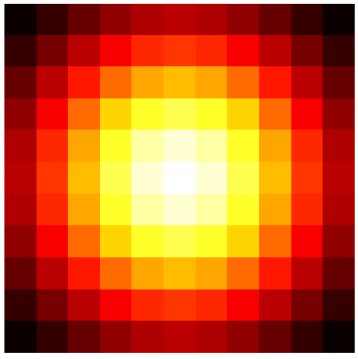


Figure 2: Kernel k=11 σ=3.5

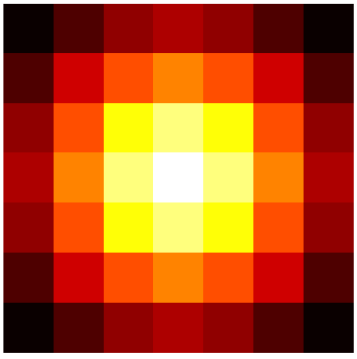


Figure 3: Kernel k=7 σ=2

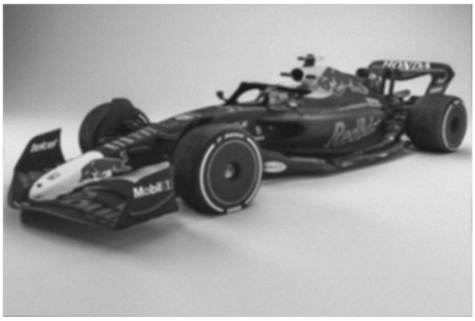


Figure 4: Immagine corrotta

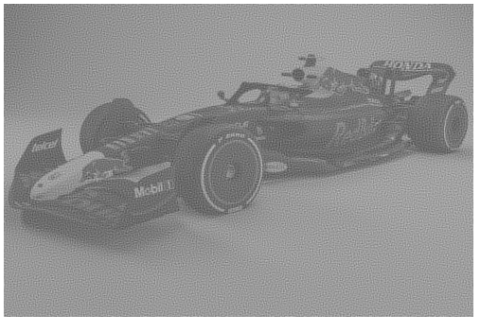


Figure 5: Ricostruzione A con CGLS

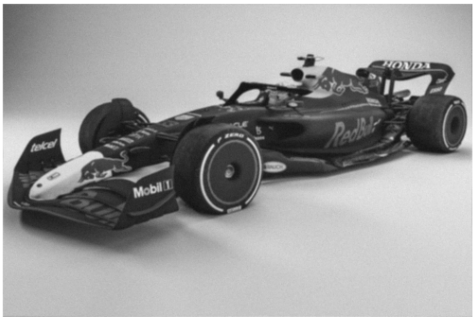


Figure 6: Immagine corrotta

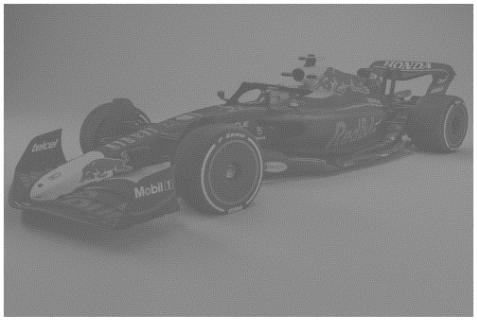


Figure 7: Ricostruzione D con CGLS

2.2 Tikhonov

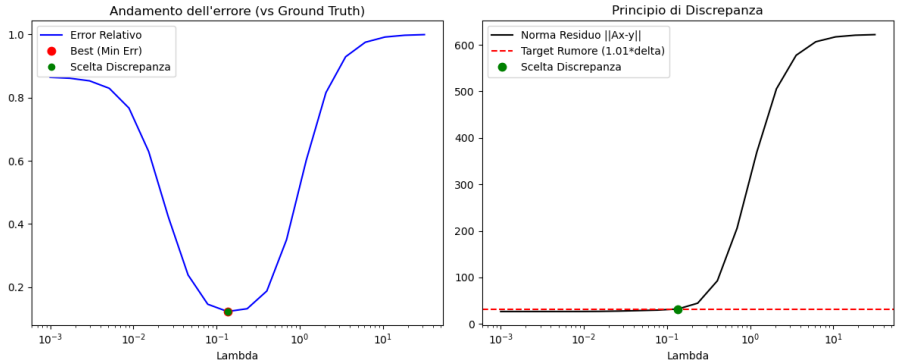
Nel codice ci serviremo di una griglia di valori λ (da 10^{-3} a $10^{\frac{3}{2}}$) e l'immagine che individueremo sarà quella tale per cui il λ utilizzato genererà l'errore relativo minore. Inoltre individueremo, attraverso il principio di massima discrepanza, un altro λ che minimizza l'errore assoluto fra l'immagine ground truth e la ricostruzione.

Test	**Rumore**	**Kernel**	**λ (Discrepanza)**	**Errore Relativo**	**PSNR**	**SSIM**
A.	0.05	G(11, 3.5)	0.13538761800225432	0.12317500740482173	22.0344	0.5570
B.	0.10	G(11, 3.5)	0.13538761800225432	0.14577992100848555	20.5709	0.3583
C.	0.025	G(11, 3.5)	0.07847599703514611	0.11730388991833358	22.4586	0.5409
D.	0.05	G(7, 2)	0.13538761800225432	0.1121742420035512	22.8470	0.5194
E.	0.10	G(7, 2)	0.13538761800225432	0.15364260338418387	20.1146	0.3018
F.	0.025	G(7, 2)	0.07847599703514611	0.10650540270011072	23.2974	0.5295

Il codice è riportato nella pagina successiva.

```
x=plt.imread("redbullBMP.bmp")
x = rgb2gray(x)
nx, ny = x.shape
x = x / x.max()
kernel = utilities.gaussian2d_kernel(k=7, sigma=2)
A = operators.ConvolutionOperator(kernel)
y = A(x)
noise = utilities.gaussian_noise(y, noise_level=0.025)
y_delta = y + noise
norm_noise = np.linalg.norm(noise)
zeros = np.zeros_like(y)
ybar = np.vstack((y, zeros))
ybar_delta = np.vstack((y_delta, zeros))
L = operators.Identity()
x0 = np.zeros_like(x)
kmax = 50
tolf = 1e-8
tolx = 1e-8
lambdaVals=np.logspace(-3,1.5,20)
relErrs=[]
residueNorms=[]
reconstructedImgs=[]
print(f"Ricerca di lambda che minimizzi l'errore usano i possibili valori:{lambdaVals}")
print(f"Norma del rumore (Target Discrepanza): {norm_noise }")
for i,lmbda in enumerate(lambdaVals):
    M = operators.TikhonovOperator(A, L, lmbda)
    cgls_tik_solver = solvers.CGLS(M)
    x_TIK = cgls_tik_solver.solve(ybar_delta, x0, kmax, tolf, tolx)
    reconstructedImgs.append(x_TIK)
    relErr=utilities.rel_err(x_TIK,x)
    relErrs.append(relErr)
    current_residual = np.linalg.norm(A(x_TIK) - y_delta)
    residueNorms.append(current_residual)
bestIndex=np.argmin(relErrs)
bestLmbda=lambdaVals[bestIndex]
bestImg=reconstructedImgs[bestIndex]
bestError=relErrs[bestIndex]
print(f"Miglior lambda (Min Error) = {bestLmbda } con Error = {bestError }")
tau = 1.01
target_discrepancy = tau * norm_noise
diffFromTarget = np.abs(np.array(residueNorms) - target_discrepancy)
bestIndexDisc = np.argmin(diffFromTarget)
bestLmbdaDisc = lambdaVals[bestIndexDisc]
bestImgDisc = reconstructedImgs[bestIndexDisc]
bestResidDisc = residueNorms[bestIndexDisc]
print(f"Miglior lambda (Discrepanza) = {bestLmbdaDisc } con Residuo = {bestResidDisc } (Target: {targ
print("\nMetriche per soluzione da Discrepanza:")
print('ER',utilities.rel_err(bestImgDisc,x))
print('PSNR',utilities.psnr(bestImgDisc,x))
print('SSIM',utilities.ssim(bestImgDisc,x))
```

Visualizziamo nella prossima pagina gli output del codice con i parametri del test A.



2.3 Total Variation

Anche per il metodo con TV utilizzeremo una λ-grid con valori da 10^{-5} a $10^{-\frac{1}{2}}$. Il codice, dove crechiamo solo l'immagine che minimizzi l'errore relativo, ci permette di riempire la tabella riportata qui:

Test	**Rumore**	**Kernel**	**Miglior λ**	**Errore Relativo**	**PSNR**	**SSIM**
A.	0.05	G(11, 3.5)	$5.1348329074375493 \cdot 10^{-5}$	0.11999160983407642	22.2619	0.6577
B.	0.10	G(11, 3.5)	0.002335721469090121	0.12652859911462186	21.8011	0.6189
C.	0.025	G(11, 3.5)	$2.9763514416313192 \cdot 10^{-5}$	0.116269875723798	22.5355	0.7581
D.	0.05	G(7, 2)	0.0013538761800225433	0.10440763764620617	23.4702	0.6848
E.	0.10	G(7, 2)	0.0069519279617756054	0.11318126695706472	22.7694	0.6423
F.	0.025	G(7, 2)	$8.858667904100833 \cdot 10^{-5}$	0.09738255026904434	24.0752	0.7527

```
x=plt.imread("redbullBMP.bmp")
x = rgb2gray(x)
nx, ny = x.shape
x = x / x.max()
kernel = utilities.gaussian2d_kernel(k=7, sigma=2)
A = operators.ConvolutionOperator(kernel)
y = A(x)
y_delta = y + utilities.gaussian_noise(y, noise_level=0.025)
gd_tv_solver = solvers.GDTotalVariation(A, beta=1e-3)
x0 = np.zeros_like(x)
kmax = 30
tolf = 1e-8
tolx = 1e-8
lambdaVals=np.logspace(-5,-0.5,20)
errs=[]
reconstructedImgs=[]
print(f"Ricerca di lambda che minimizzi l'errore usano i possibili valori:{lambdaVals}")
for i,lmbda in enumerate(lambdaVals):
    x_TV, obj_val, grad_norm = gd_tv_solver.solve(y_delta, lmbda, x0, kmax, tolf, tolx)
    relErr=utilities.rel_err(x_TV,x)
    errs.append(relErr)
    reconstructedImgs.append(x_TV)
bestIndex=np.argmin(errs)
bestLmbda=lambdaVals[bestIndex]
bestImg=reconstructedImgs[bestIndex]
bestError=errs[bestIndex]
print(f"Miglior lambda = {bestLmbda} con bestError = {bestError} bestIndex = {bestIndex}")
print('ER',utilities.rel_err(bestImg,x))
print('PSNR',utilities.psnr(bestImg,x))
print('SSIM',utilities.ssim(bestImg,x))
```

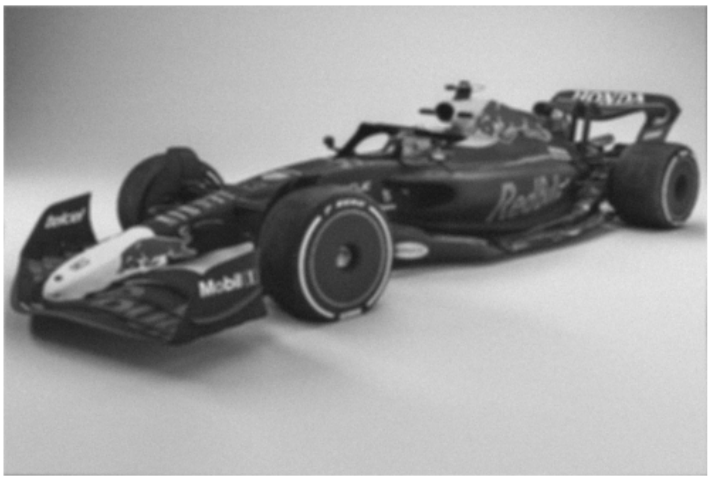


Figure 8: Immagine corrotta

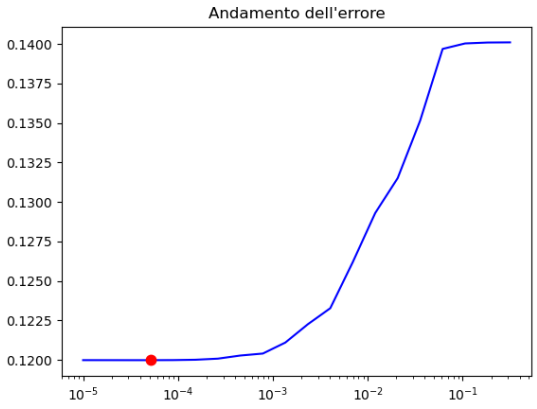


Figure 9: Test A.

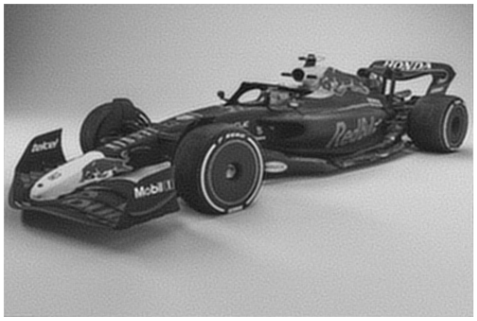


Figure 10: Ricostruzione A con Total Variation

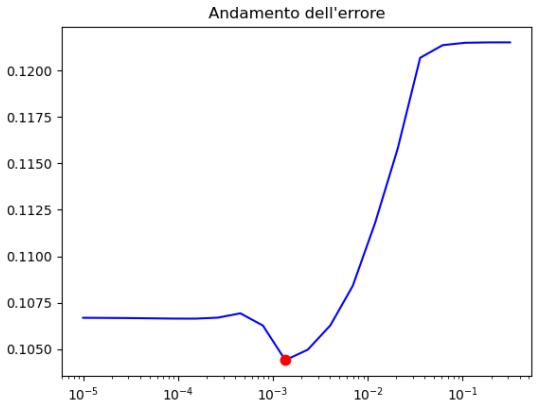


Figure 11: Test D.

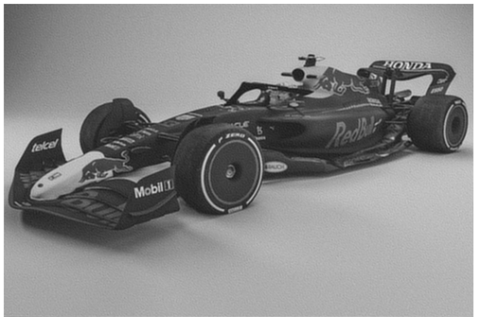


Figure 12: Ricostruzione D con Total Variation