

Progetto Metodi del Calcolo Scientifico

Matteo Colella - 794028

Matteo Costantini - 795125

Dario Gerosa - 793636

Obiettivo

- ▶ Utilizzo dell'implementazione della DCT2 in ambiente open source per l'osservazione degli effetti della compressione jpeg su immagini a livelli di grigio.
 - ▶ Prima parte: confronto dei tempi d'esecuzione della DCT2 implementata nella libreria scelta rispetto ad una nostra implementazione in ambiente open source.
 - ▶ Seconda parte: creazione di un software che applichi un'alterazione delle frequenze ad un'immagine a livelli di grigio scelta dall'utente tramite un'interfaccia grafica e ne visualizzi i risultati.

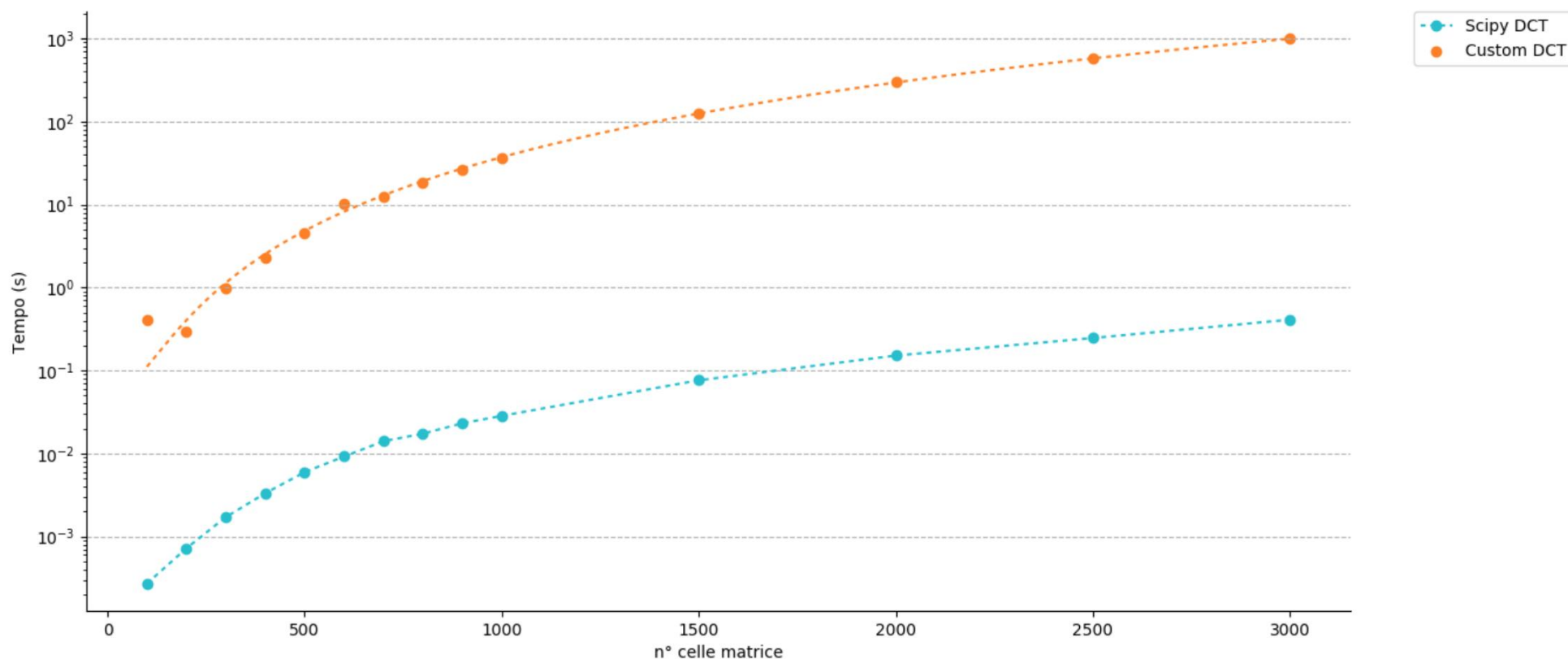
Python scipy

- ▶ Ecosistema open-source nato nel 2001 e composto da diverse librerie (NumPy, SciPy library, Matplotlib, Ipython, Sympy, Pandas)
- ▶ Utilizzato in matematica, scienze e ingegneria
- ▶ Composta da diversi package che offrono supporto per: clustering, trasformata di Fourier, interpolazione, algebra lineare, matrici sparse, programmazione lineare, trattamento di segnali ...
- ▶ Attivamente mantenuta (ultima release 5/10/18) e documentata (<https://www.scipy.org/docs.html>)
 - ▶ Sorgente: <https://github.com/scipy/scipy>

Python scipy

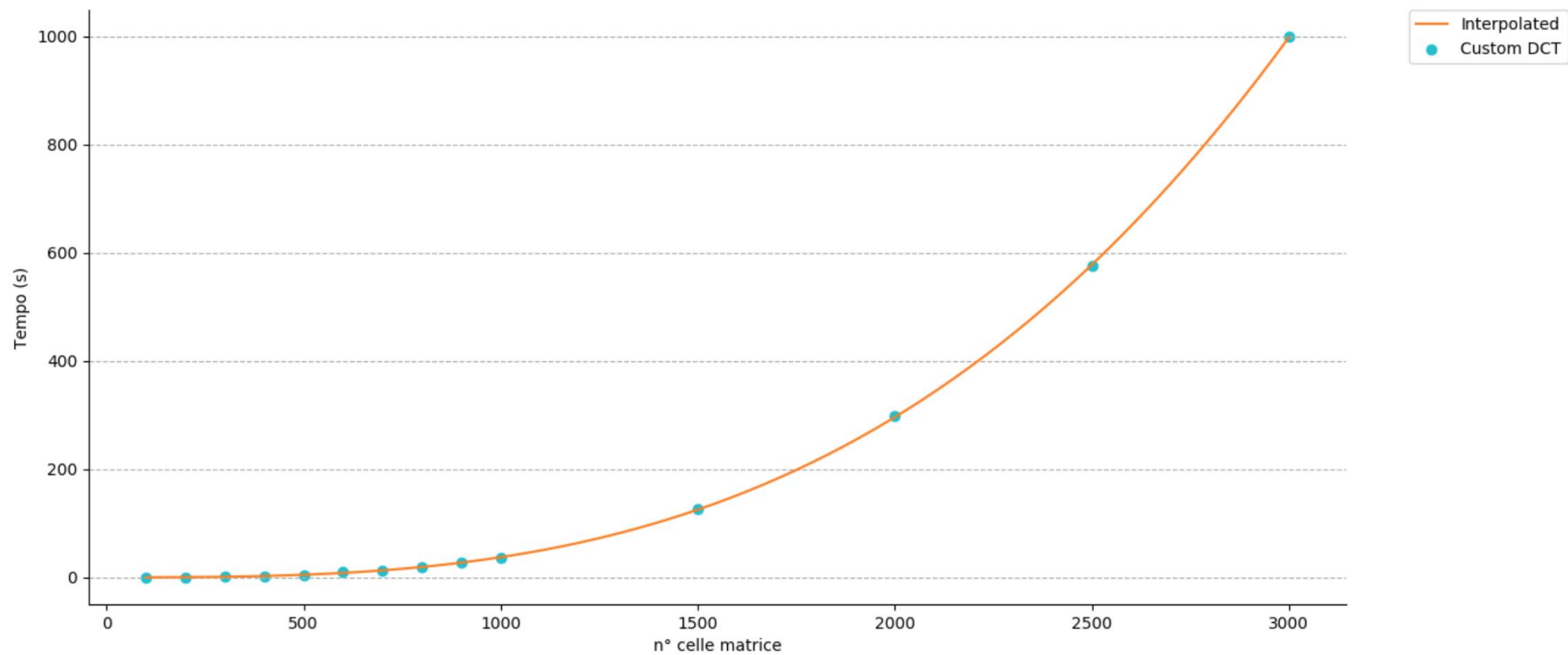
- ▶ In particolare abbiamo utilizzato:
 - ▶ `scipy.fftpack`: pacchetto contenente le implementazioni delle trasformate di Fourier
 - ▶ `dct()`: calcola la Discrete Cosine Transform di un array `x`
 - ▶ `idct()`: calcola la Inverse DCT di un array `x`
- ▶ `numba`: libreria per la compilazione just-in-time di codice Python, tramite annotazioni
- ▶ `opencv`: libreria open source per l'elaborazione delle immagini, utilizzata per il caricamento delle immagini.
- ▶ `PyQt5`: libreria per la creazione dell'interfaccia grafica

Parte 1 - Confronto tempi esecuzione

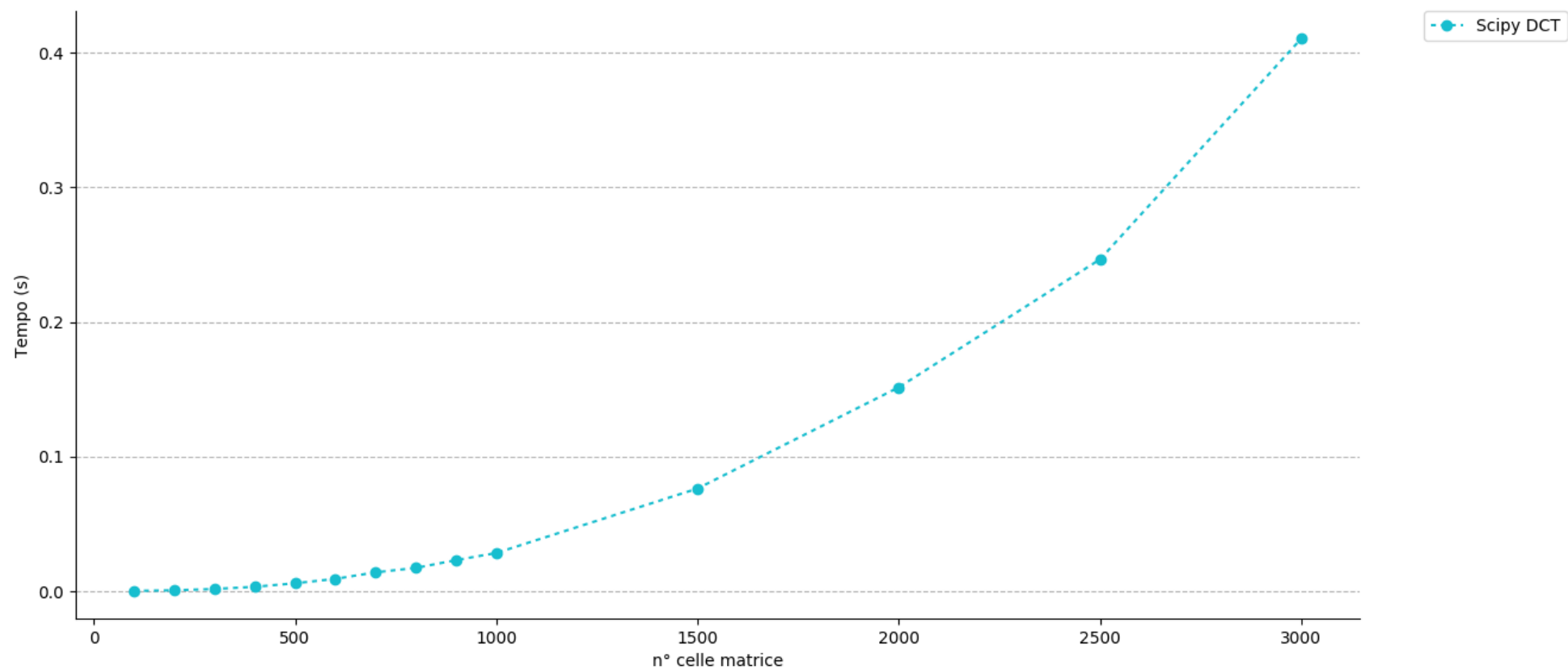


Abbiamo potuto verificare che i tempi d'esecuzione della custom-dct sono effettivamente proporzionali a N^3 , mentre quelli della scipy-dct sono più bassi e proporzionali a N^2 .

Parte 1 - Tempi di esecuzione custom



Parte 1 - Tempi di esecuzione scipy



Codice Parte 1

```
def custom_dct(array):  
    r_array = np.zeros(array.size)  
    N = array.size  
  
    for u in range(N):  
        somma = 0  
        a = math.sqrt(1. / N) if u == 0 else math.sqrt(2. / N)  
  
        for x, cell in enumerate(array):  
            somma += cell * math.cos((u * math.pi * (2 * x + 1)) / (2 * N))  
  
        r_array[u] = a * somma  
  
    return r_array  
  
def custom_dct2(mat):  
    r_mat = np.zeros(mat.shape)  
    r_mat = np.apply_along_axis(custom_dct, axis=1, arr=mat)  
    r_mat = np.apply_along_axis(custom_dct, axis=0, arr=r_mat)  
  
    return r_mat
```


Parte 2 - Esempio Beta < 1

$d = 300$
Beta = 0

Immagine originale

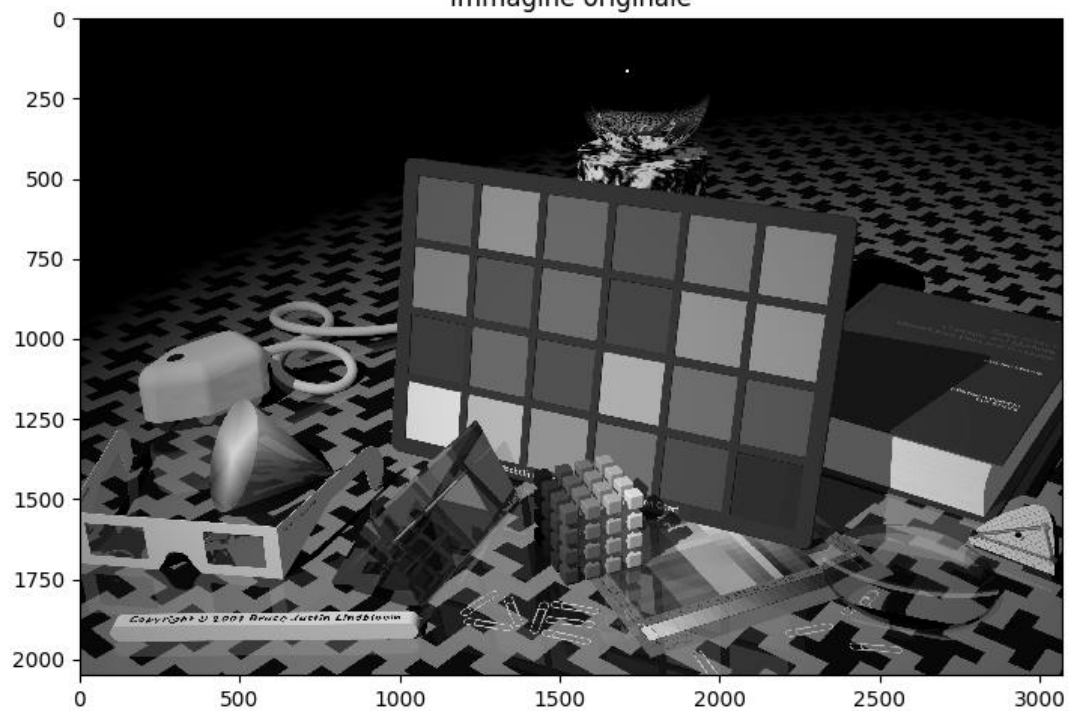
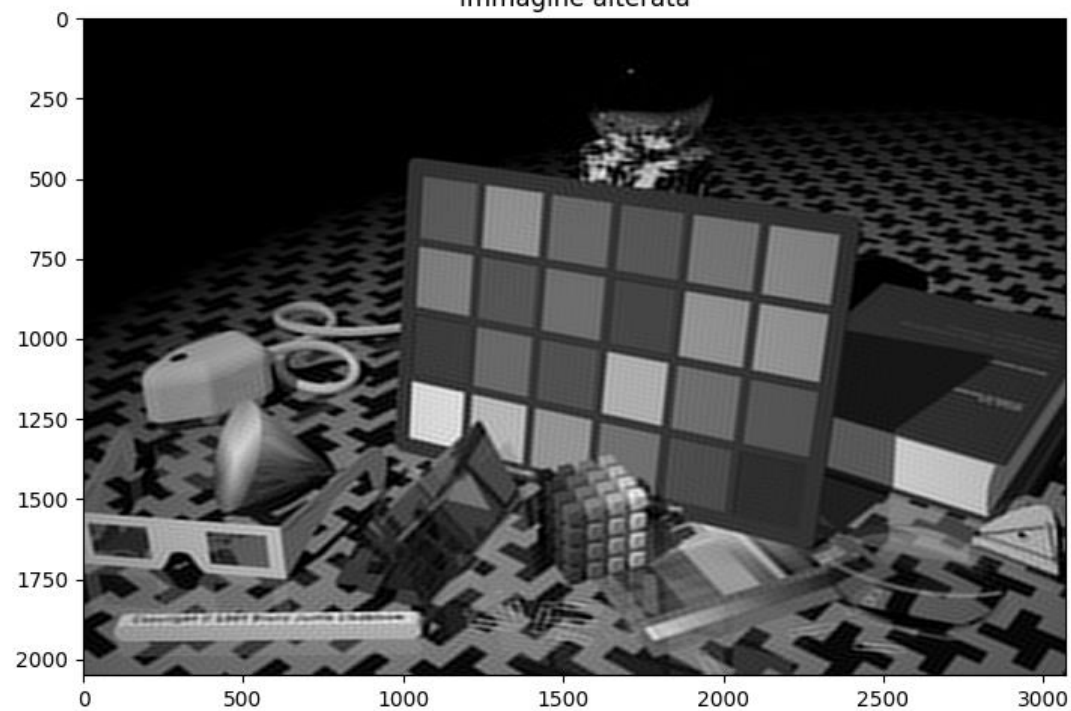
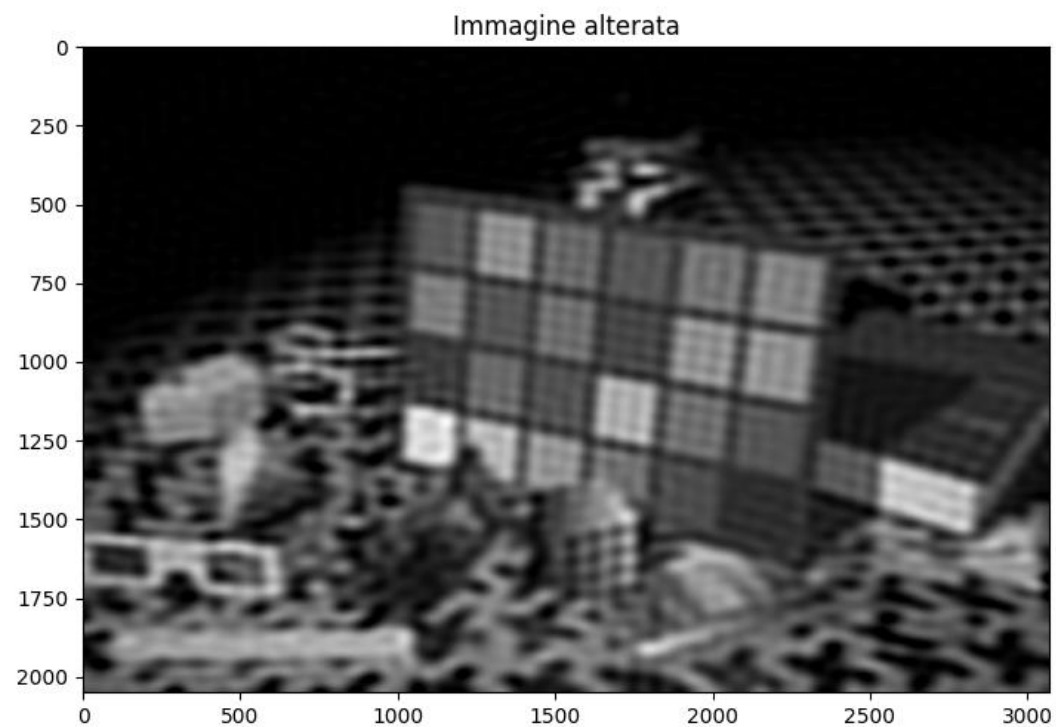
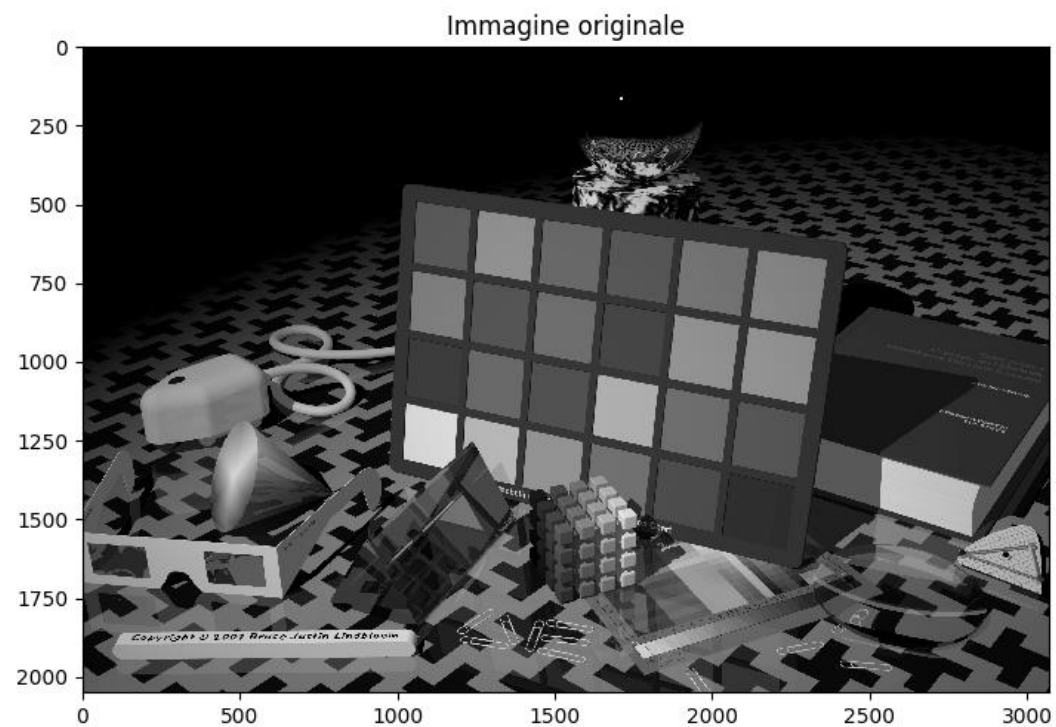


Immagine alterata

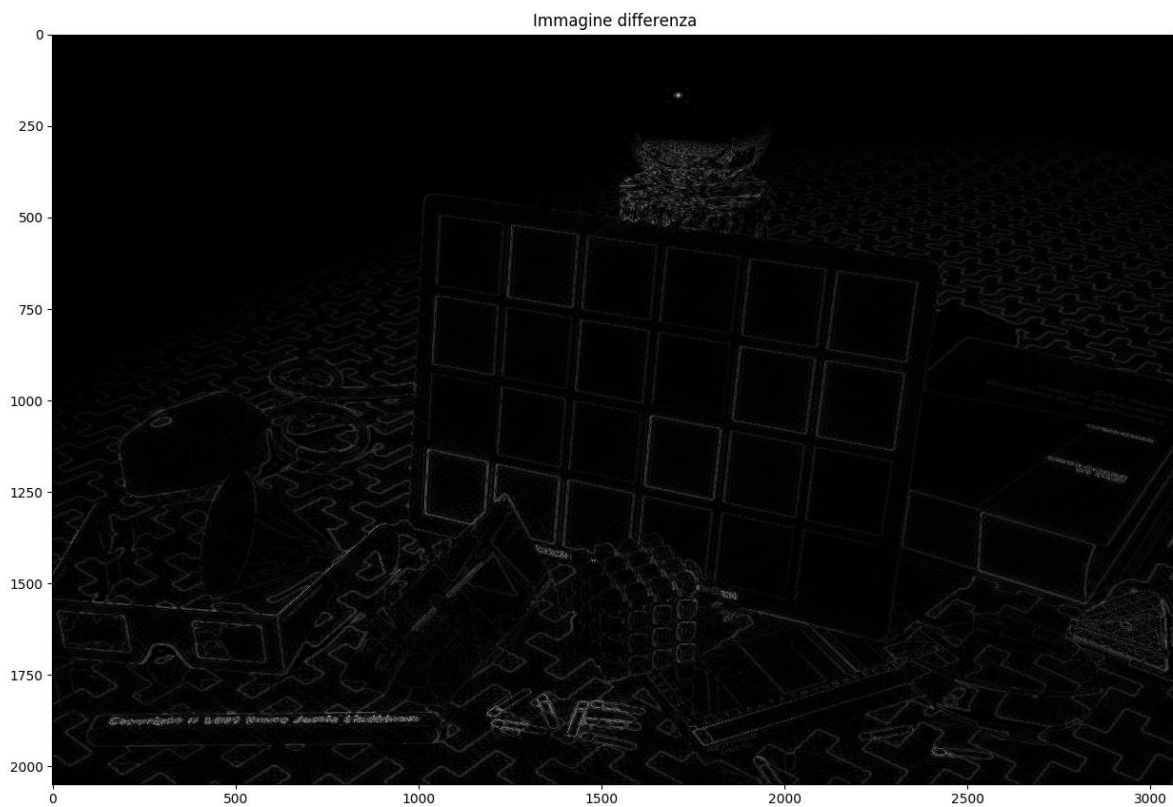


Parte 2 - Esempio Beta < 1

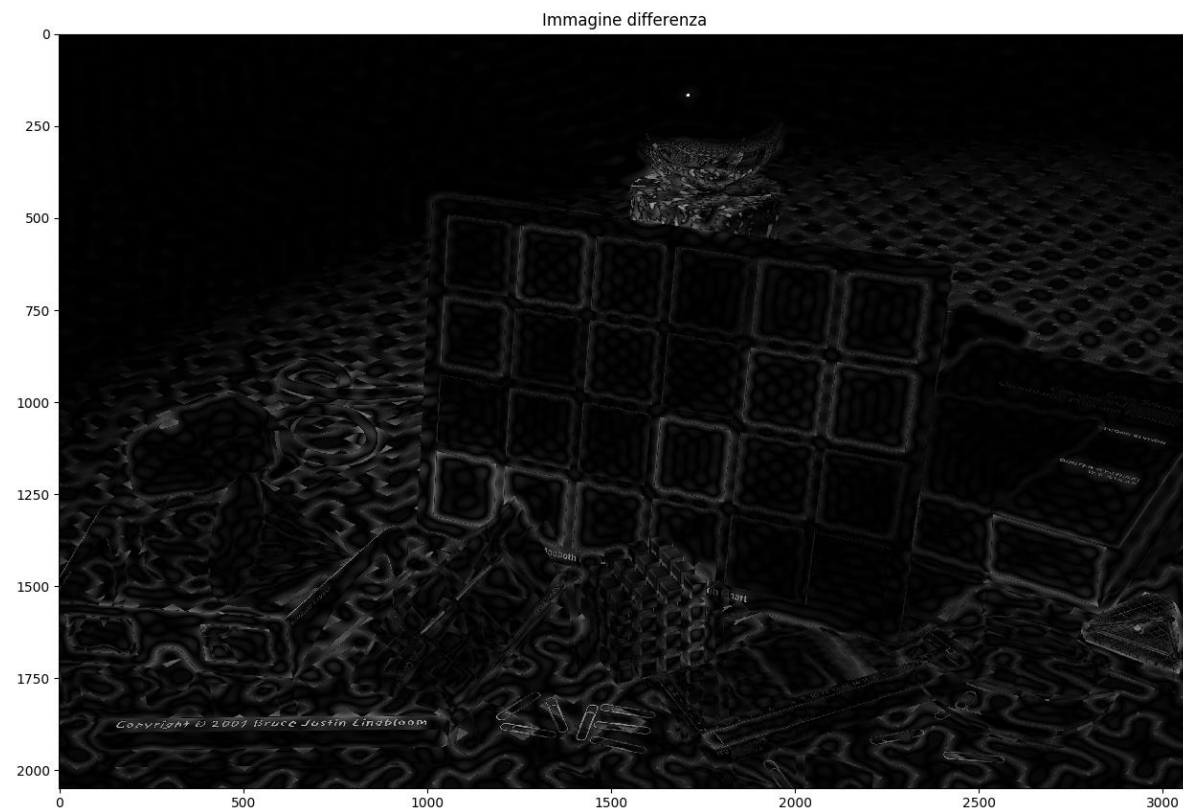
$d = 100$
Beta = 0



Parte 2 - Esempio Beta < 1



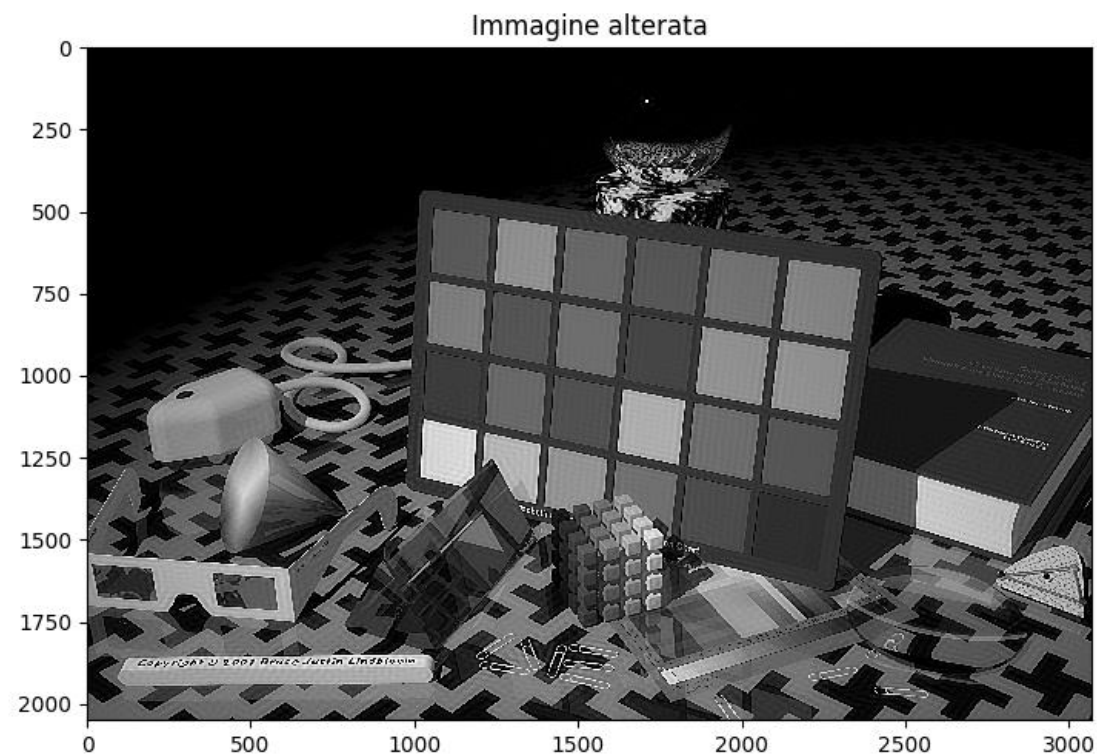
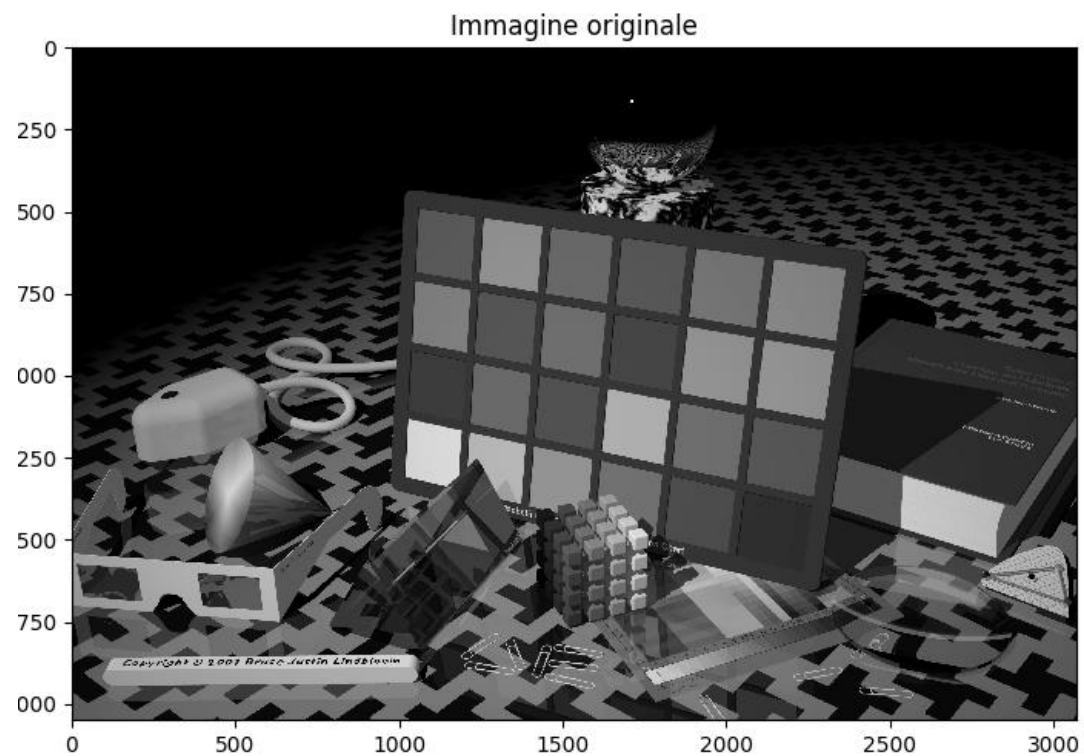
$d=300$
 $\text{Beta} = 0$



$d=100$
 $\text{Beta} = 0$

Parte 2 - Esempio Beta > 1

$d = 300$
Beta = 2



Parte 2 - Esempio Beta > 1

$d = 100$
Beta = 2

Immagine originale

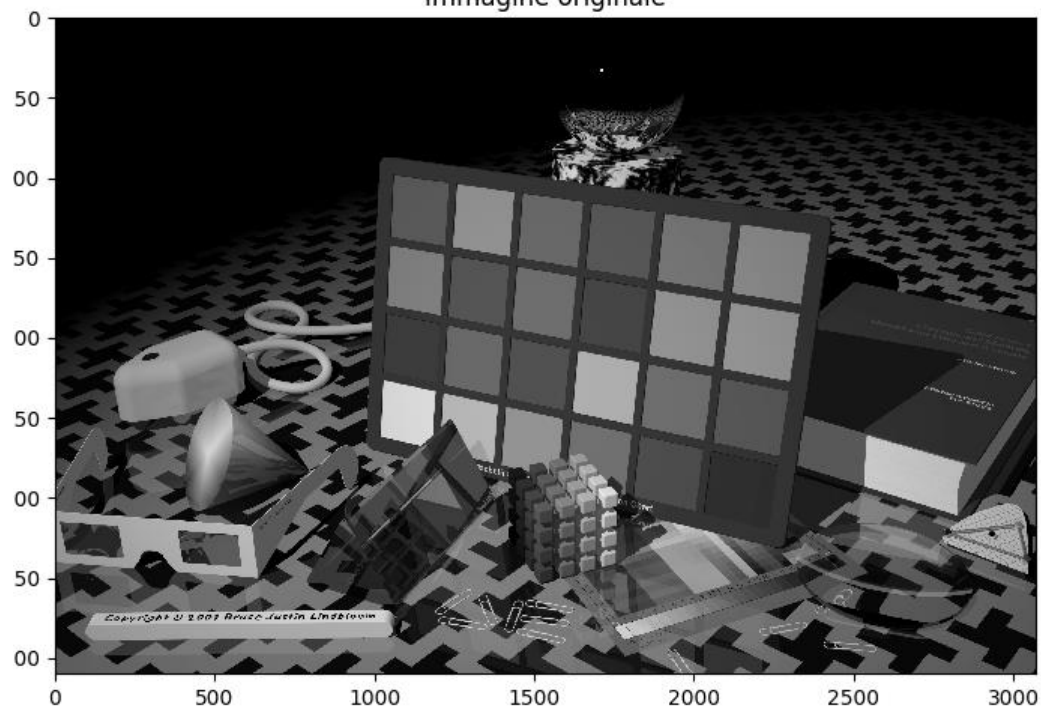
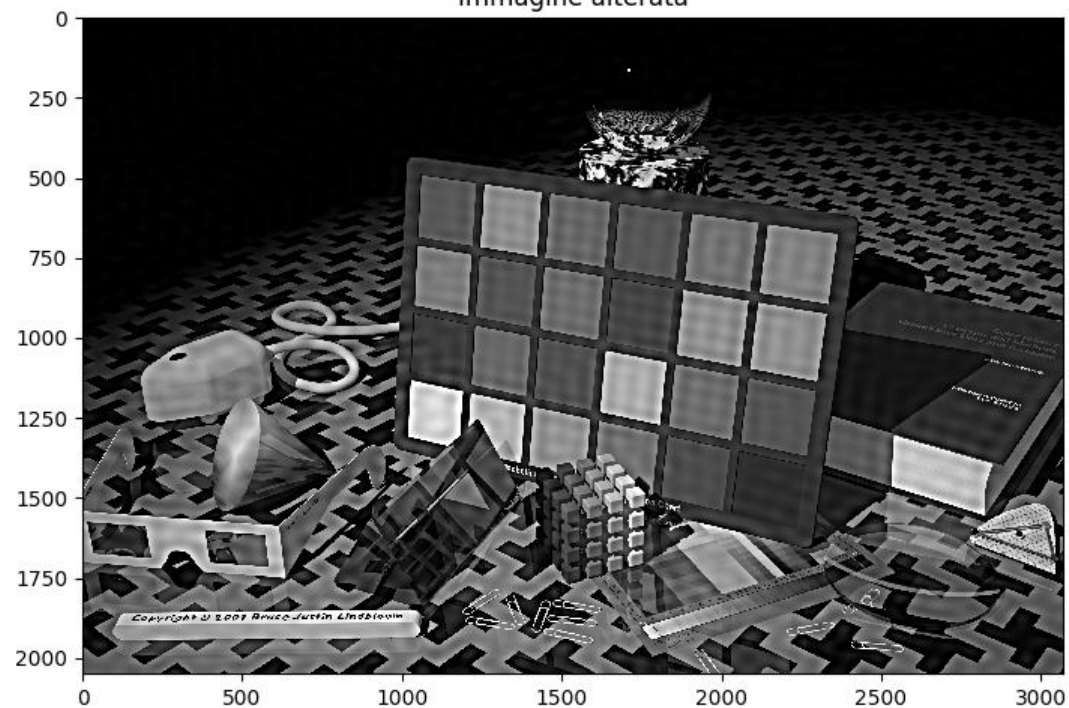
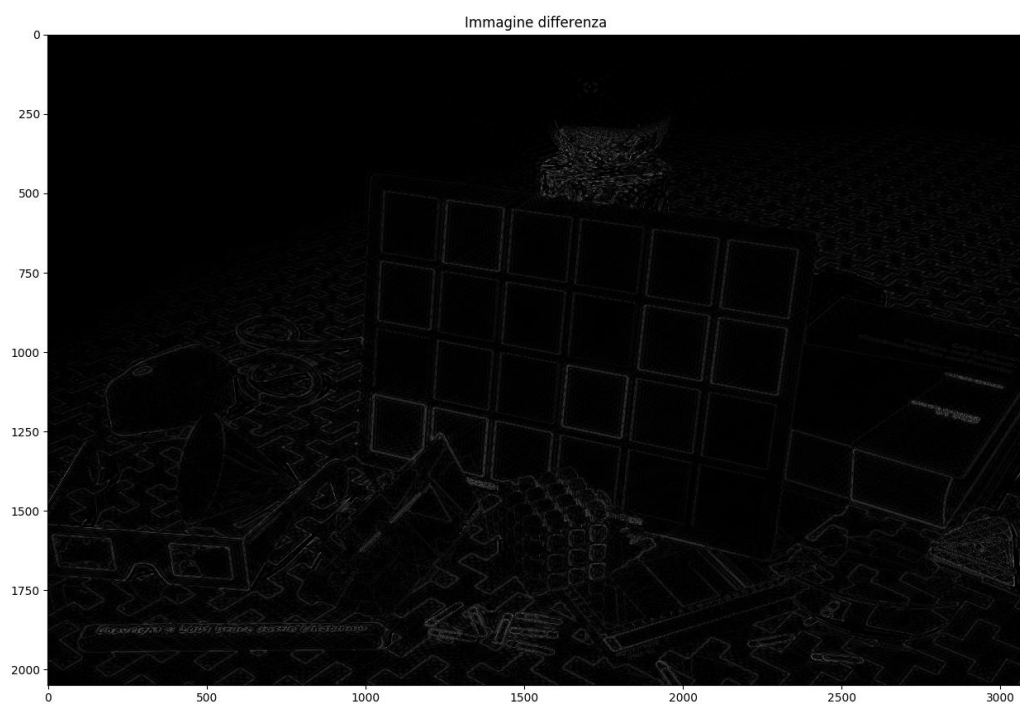


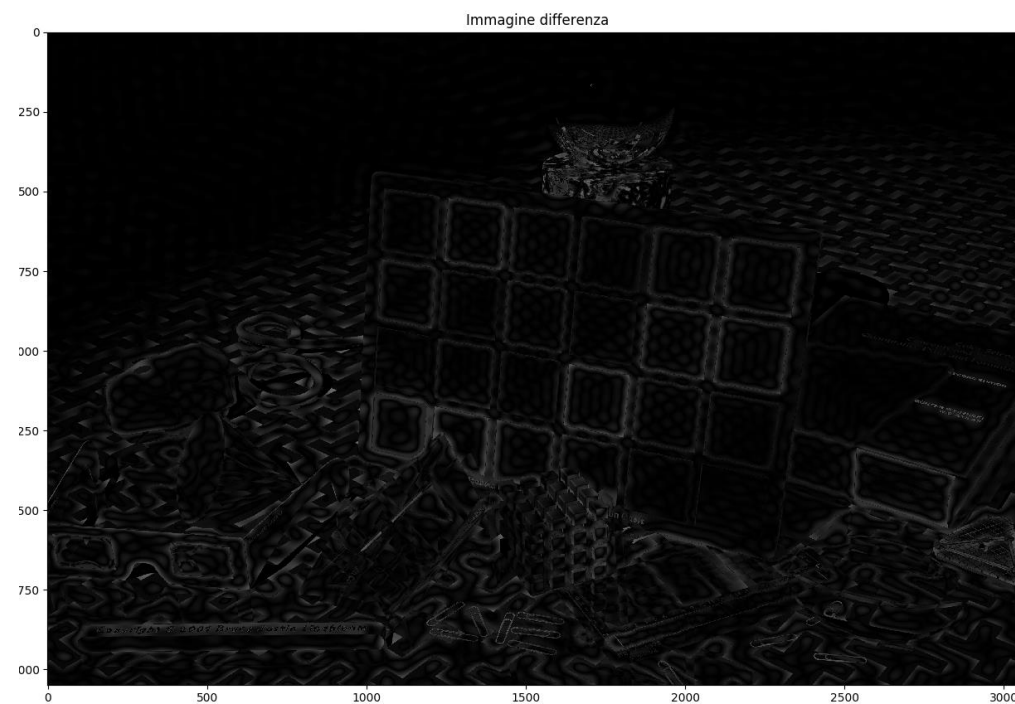
Immagine alterata



Parte 2 - Esempio Beta > 1



$d=300$
 $\text{Beta} = 2$



$d=100$
 $\text{Beta} = 2$

Codice Parte 2

```
def alter_freq(self):  
    # Applicazione dct  
    d_img = dct(dct(self.img.T, norm='ortho').T, norm='ortho')  
  
    # modifica frequenze  
    for i, row in enumerate(d_img):  
        for j, col in enumerate(row):  
            if i + j >= self.d:  
                d_img[i, j] *= self.beta  
  
    # Applicazione inversa dct e arrotondamento  
    i_img = round_image(idct(idct(d_img.T, norm='ortho').T, norm='ortho'))  
  
def round_image_(self, pixel):  
    if pixel > 255:  
        return 255  
    elif pixel < 0:  
        return 0  
    else:  
        return round(pixel)  
round_image = np.vectorize(self.round_image_)
```