

# Metodi del Calcoli Scientifico

Progetto 2 / Parte 1

Compressione di immagini tramite la DCT

EDOARDO SILVA 816560

BRYAN ZHIGUI 816335

DAVIDE MARCHETTI 815990

A.A.: 2019/2020

## **Abstract**

Si vuole avere un confronto dei tempi d'esecuzione della DCT2 della libreria scelta con la nostra implementazione. I problemi da risolvere sono matrici quadrate generate casualmente.

I risultati ottenuti saranno riportati graficamente in una scala semi-logaritmica.

# 1 Librerie

## Numpy

Libreria open source che contiene diverse funzioni e metodi utili per il calcolo scientifico. In particolar modo per il calcolo vettoriale e di matrici multidimensionali in maniera efficiente e veloce.

## Scipy

Libreria open source che utilizza le funzionalità di Numpy per fornire un pacchetto di calcolo scientifico General-purpose. In particolare, in questo progetto si utilizza la estensione **Fast Fourier Transform** (`fftpack`) per lavorare con le DCT e DCT-multidimensionale(**DCT2**).

## Pandas

Fornisce una serie di strumenti per elaborare e manipolare dati in formato tabellare (`DataFrame`) interfacciandosi con le strutture dati native in Python.

## Matplotlib

Permette di generazione grafici di svariate tipologie. Il modulo `pyplot` fa da interfaccia alle API più a basso livello per la generazione di codice.

## Seaborn

Utilizzato principalmente per la creazione di grafici statistici. Agisce come un wrapper per la libreria `matplotlib` semplificandone alcune funzionalità per la creazione di grafici complessi.

### 1.1 Implementazione della DCT in Scipy

Dalla documentazione del modulo `scipy.fftpack`<sup>1</sup> viene referenziato un documento che descrive dal punto di vista matematico l'implementazione della DCT mono e multidimensionale nella libreria.

---

1

L'abstract del paper "*A fast cosine transform in one and two dimensions*" di Makhoul, J. (Feb. 1980)<sup>2</sup> riporta:

The discrete cosine transform (DCT) of an  $N$ -point real signal is derived by taking the discrete Fourier transform (DFT) of a  $2N$ -point even extension of the signal. It is shown that the same result may be obtained using only an  $N$ -point DFT of a reordered version of the original signal, with a resulting saving of  $1/2$ . If the fast Fourier transform (FFT) is used to compute the DFT, the result is a fast cosine transform (FCT) that can be computed using on the order of  $N \log_2(N)$  real multiplications. The method is then extended to two dimensions, with a saving of  $1/4$  over the traditional method that uses the DFT.

Difatti, i risultati proposti nella sezione 4 confermeranno l'effettiva efficienza di questo metodo rispetto all'implementazione da noi proposta, risultando sempre più evidente al crescere della dimensione della matrice da elaborare.

---

<sup>2</sup><https://ieeexplore.ieee.org/document/1163351>

## 2 Implementazione della DCT/DCTN

### 2.1 DCT

La funzione *DCT* prende in input un vettore  $V$  di dimensione  $N$  ed esegue le seguenti operazioni:

1. Viene pre-allocato un vettore  $C$  di dimensione  $N$  con componenti tutte pari a zero.
2. Per ogni elemento del vettore  $V$  di input:
  - (a) Viene calcolato il coefficiente  $C[k]$  nella base  $w_k$  del valore  $V[i]$
  - (b) Questo viene scalato per un certo valore  $\alpha_k$
3. Il vettore risultante  $C$  viene ritornato come vettore in output

**Listing 1:** Implementazione della DCT monodimensionale

```
1 def dct(V):
2     N = len(V)
3     c = np.zeros(N)
4     for k in range(N):
5         s = 0
6
7         for i in range(N):
8             s += V[i] * np.cos(k * np.pi * ((2*i+1) / (2*N)))
9
10        alpha = 1 / math.sqrt(N) if k == 0 else math.sqrt(2 / N)
11        c[k] = alpha * s
12
13    return c
```

### 2.2 DCT2

La implementazione della DCT2 sfrutta l'applicazione della DCT monodimensionale su colonne e righe per ottenere la scomposizione corretta.

Nell'algoritmo, l'input risulta essere una matrice quadrata *Mat* di dimensione  $N \times N$ . Inoltre, viene definita una matrice intermedia  $C$  inizializzata

a zero la quale sarà elaborata a partire dalla matrice in input per ottenere l'applicazione corretta della DCT2.

Inizialmente si effettua la *DCT* monodimensionale prima sui vettori colonna della matrice originale salvando i risultati ottenuti nella matrice  $C$  e, successivamente, sui vettori riga della stessa appena popolati.

**Listing 2:** Implementazione della DCT multidimensionale (DCT2)

```
1 def dct2(Mat):
2     N, M = Mat.shape
3
4     C = np.zeros((N, M), dtype='float')
5     for j in range(M):
6         C[:, j] = dct(Mat[:, j])
7
8
9     for i in range(N):
10        C[i, :] = dct(C[i, :])
11
12    return C
```

### 3 Esecuzione

Il fulcro del programma consiste in una funzione che accetta in input tre parametri:

`offset`: da che dimensione generare le matrici

`how_many`: numero di matrici da generare e confrontare

`step`: di quanto incrementare la dimensione della matrice ad ogni iterazione

La funzione esegue quindi i seguenti step:

1. Genera una matrice quadrata `problem` di dimensione `size` con valori randomici in un range `[0, 255]`.
2. Applica la DCTN di tipo 2 con norma ortogonale tenendo traccia del tempo di esecuzione
3. Salva dimensione e tempo impiegato per l'esecuzione `scipy` in un vettore `matrices`
4. Applica la DCT2 implementata come descritto nella sezione 2.2 tenendo traccia del tempo di esecuzione.
5. Memorizza dimensione e tempo di esecuzione dell'implementazione manuale nello stesso vettore.

Una volta terminato il ciclo, il vettore `matrices` conterrà `2×how_many` elementi: i risultati da visualizzare su grafico per il confronto.

**Listing 3:** Funzione per il confronto tra implementazioni

```
1 def performance_test(offset, how_many, step=1):
2     matrices = []
3
4     for index in range(0, how_many*step, step):
5         size = index + offset
6         problem = np.random.randint(0, 255, size=(size, size))
7
8         tic = time.perf_counter()
9         dctn(problem, type=2, norm='ortho')
```

```
10     toc = time.perf_counter()
11
12     matrices.append({
13         'size': size,
14         'duration': float(toc - tic),
15         'type': 'scipy'
16     })
17
18     tic = time.perf_counter()
19     custom_dct.dct2(problem)
20     toc = time.perf_counter()
21
22     matrices.append({
23         'size': size,
24         'duration': float(toc - tic),
25         'type': 'implemented'
26     })
27
28     return matrices
```

## 4 Risultati

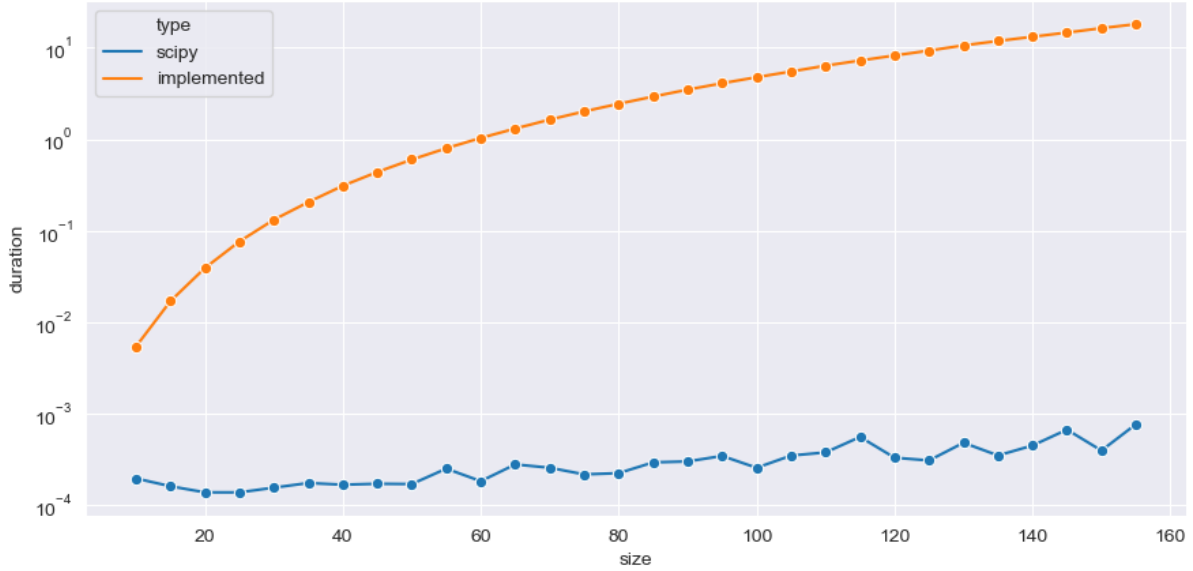


Figura 1: Confronto tra DCT2 scipy e custom

## 5 Conclusioni

L'implementazione proposta dal documento citato nella sezione 1.1 risulta quindi confermata dal confronto riportato in fig. 1.

Dall'analisi grafica possiamo notare che la DCT2 implementata manualmente segue un andamento riconducibile ad un  $O(N^3)$ , invece l'implementazione utilizzata in `scipy` delinea un andamento altalenante tra  $O(N \log N)$  e  $O(N^2)$ .

Naturalmente, ci si aspettava che la nostra implementazione risultasse decisamente più lenta dell'implementazione FFT, tuttavia, il risultato ottenuto in termini numerici risulta equiparabile con entrambi i metodi.