Metodi diretti per matrici sparse

Alice Romagnoli - 829833
Andrei Gabriel Taraboi - 829904
Davide Rendina - 830730

Introduzione

Lo scopo del progetto presentato è quello di confrontare, in ambienti di programmazione open source e non, la risoluzione attraverso metodi diretti di sistemi lineari con matrici sparse di grandi dimensioni.

Introduzione

Gli ambienti di sviluppo/linguaggi di programmazione utilizzati sono:

- **Matlab**, come esempio di software proprietario
- **Gnu Octave**, in quanto software open source
- **Python**, come seconda alternativa open source

Tutti i confronti sono stati effettuati in ambienti Linux e Windows in termini di tempo, accuratezza, impiego della memoria e facilità d’uso.

MATLAB

Per la risoluzione del sistema lineare con matrici sparse, è stato ricavato il vettore b, in modo tale che la soluzione esatta fosse il vettore xe di soli 1.

Dopodiché è stata calcolata la soluzione del sistema (dati A e b), utilizzando l'operatore «\» e calcolando tempo d'esecuzione e memoria utilizzata.

Infine è stato calcolato l'errore relativo tra la soluzione esatta xe e quella trovata x.

```
%vettore xe di 1, della stessa dimensione di A
xe = ones(length(A), 1);

%calcolo della variabile b, usando l'equazione A*xe = b
disp("Calcolo A * xe ...")
b = A * xe;

%calcolo della soluzione x del sistema A*x = b (note A e b)
disp("Calcolo A \\ b")

tic; %inizio di time
    x = A\b;
    disp("Memory usage(Mb):") %memoria utilizzata
    memoria_usata = [memoria_usata, monitor_memory_whos];
tempo = toc; %fine di time

tempo_di_esecuzione = [tempo_di_esecuzione, tempo];

%calcolo dell'errore relativo tra x e xe
errore_relativo = norm(x-xe)/norm(xe);
error_relativo_matrice = [error_relativo_matrice, errore_relativo];
```

MATLAB

La documentazione di MATLAB fornisce una chiara spiegazione di come venga operata la risoluzione del sistema.

Prima di procedere con un approccio risolutivo, MATLAB esegue diversi controlli sulla matrice A in base ai quali decide come operare.

In particolare vengono effettuati uno o più dei seguenti controlli: se la matrice A è quadrata, diagonale, triangolare o a bande, triangolare permutata, se è hermitiana (simmetrica), se la diagonale è interamente positiva/negativa e se A contiene solo numeri reali.

GNU Octave

Lo stesso procedimento di risoluzione del sistema è stato poi eseguito in GNU Octave, il quale ha un comportamento molto simile a MATLAB.

```
%vettore xe di 1, della stessa dimensione di A
xe = ones(length(A), 1);

%calcolo della variabile b, usando l'equazione A*xe = b
disp("Calcolo A * xe ...")
b = A * xe;

%calcolo della soluzione x del sistema A*x = b (note A e b)
disp("Calcolo A \\ b")

tic; %inizio di time
    x = A\b;
    disp("Memory usage(Mb):") %memoria utilizzata
    memoria_usata =  monitor_memory_whos;
    disp(memoria_usata)
tempo = toc; %fine di time

%calcolo dell'errore relativo tra x e xe
errore_relativo = norm(x-xe)/norm(xe)
```

GNU Octave

Il procedimento dietro l'operatore «\» di Octave risulta essere un po' meno facilmente reperibile e comprensibile.

La serie di controlli effettuata sulla matrice A, prima di procedere alla risoluzione del sistema, risulta infatti essere più contorta e innestata.

I controlli effettuati risultano essere tuttavia pressoché gli stessi di MATLAB, benché eseguiti in ordine diverso.

Python

Utilizzando Python è necessario effettuare qualche passaggio in più rispetto ai software precedenti.

Ad esempio, il prodotto matriciale ritorna un array e bisogna quindi poi eseguire alcune conversioni per poter calcolare la soluzione del sistema.

Per calcolare la soluzione del sistema, la libreria scipy mette a disposizione il metodo «.spsolve()».

Infine è stato calcolato l'errore relativo tra la soluzione esatta xe e quella trovata x.

```
# vettore xe di 1, della stessa dimensione di A
xe = np.ones((int(A.get_shape()[1]), 1))
xe = csc_matrix(xe)  # conversione di xe in matrice

# calcolo della variabile b, usando l'equazione A*xe = b
print("Calcolo A*xe ... \n")
b = np.dot(A, xe)
# portiamo b alla forma necessaria per poter utilizzare .spsolve()
csc_matrix.sort_indices(b)   # indici della matrice sparsa ordinati
b = csc_matrix.toarray(b)    # creazione di un array colonna
b = np.concatenate(b)        # creazione di una matrice formata da una sola colonna

# calcolo della soluzione x del sistema A*x = b (note A e b)
print("Calcolo A \ b \n")
t = time.time()  # tempo iniziale
x = spsolve(A, b)
mempria_usata = psutil.Process(os.getpid()).memory_info().rss / 1024 ** 2
elapsed = time.time() - t  # calcolo tempo esecuzione

print(x, "runned in: ", elapsed)

xe = csc_matrix.toarray(xe)  # conversione in array
# rimodellazione di x in una matrice con una sola colonna e conseguente trasposizione
x = np.transpose(x).reshape((int(A.get_shape()[1]), 1))

# calcolo dell'errore relativo tra x e xe
errore_relativo = norm(array(x-xe), 2)/norm(array(xe), 2)
```

Python

Per quanto riguarda la documentazione della libreria scipy, in particolare per la funzione «.spsolve()», le informazioni risultano essere meno precise e dettagliate, riguardo l'effettivo procedimento utilizzato.

Viene utilizzata la fattorizzazione LU, ma non sono presenti ulteriori indicazioni a meno di guardare il codice sorgente.
Abbiamo infatti così notato che vengono effettuati dei controlli iniziali ma seguiti sempre comunque dalla fattorizzazione LU.

https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.linalg.spsolve.html

Specifiche tecniche

I software elencati descritti in precedenza sono stati fatti girare in due sistemi operativi: Ubuntu 18.04 LTS in macchina virtuale e Windows 10.

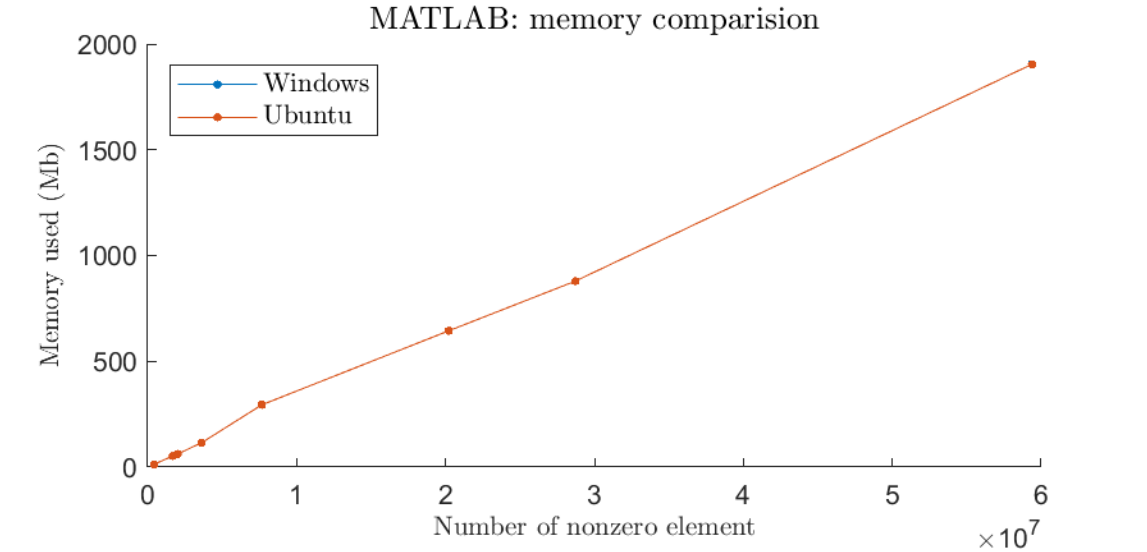
Per quanto riguarda Windows 10, sono state messe a disposizione 16Gb di RAM, Ryzen-5 3600.

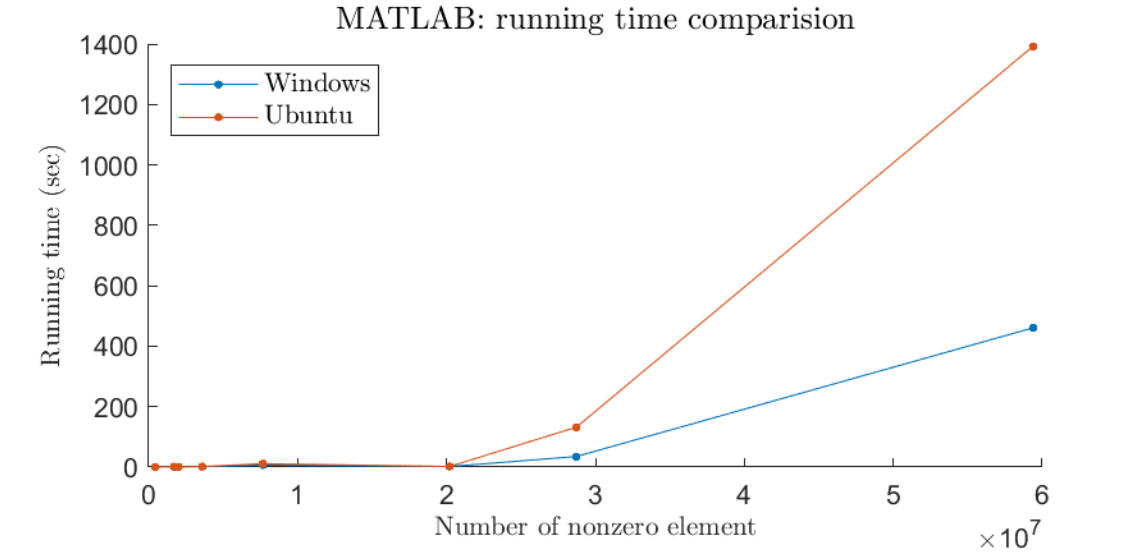
Invece, per la macchina virtuale, abbiamo messo a disposizione lo stesso processore ma solamente 11Gb di RAM.

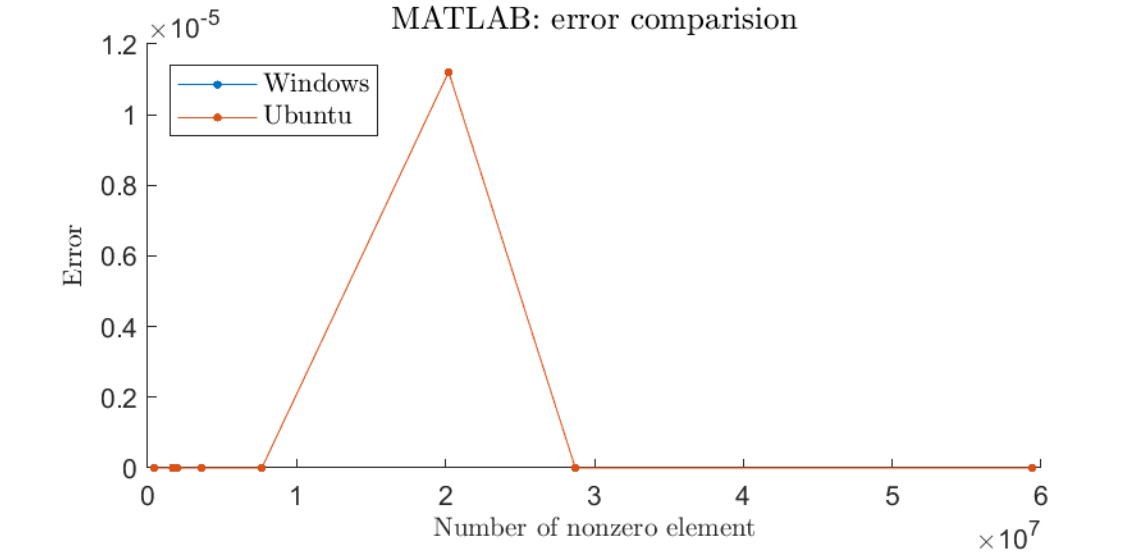
Risultati MATLAB

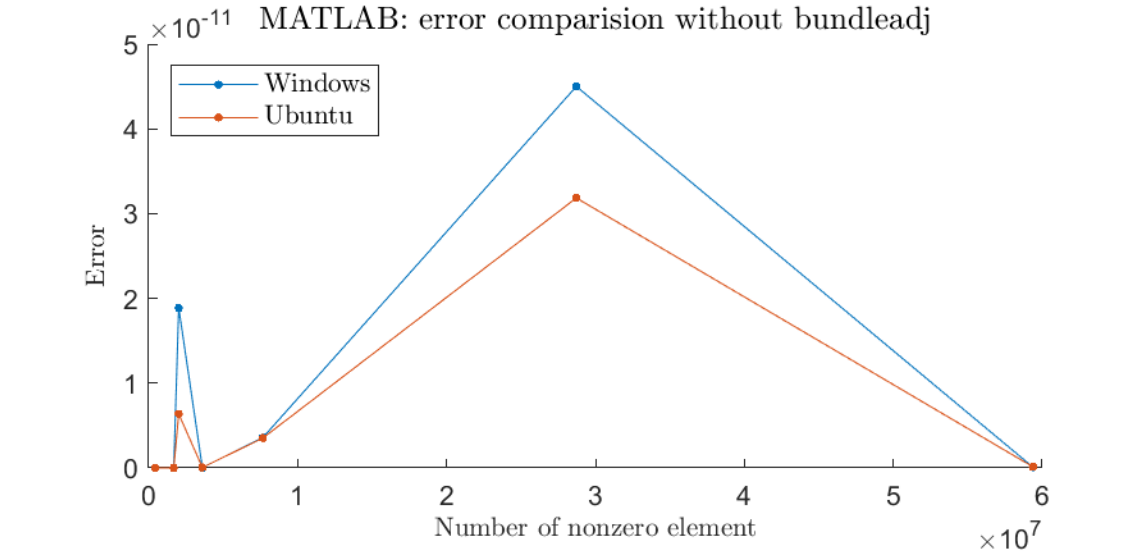
Nome	Numero zeri	Windows			Ubuntu		
		Memoria (Mb)	Tempo (secondi)	Accuratezza	Memoria (Mb)	Tempo (secondi)	Accuratezza
GT01R	4.3091e+05	14	0.16	3.8099e-15	14	0.18	3.3928e-15
ns3Da	1.6796e+06	52	0.86	9.389e-16	52	1.41	9.2082e-16
TSC_OPF_1047	2.0128e+06	62	0.38	1.8877e-11	62	0.53	6.3652e-12
ifiss	3.5999e+06	114	1.40	3.3311e-14	114	1.83	5.4347e-14
G3_circuit	7.6608e+06	294	6.75	3.5748e-12	294	11.25	3.5228e-12
Bundle_adj	2.0208e+07	644	1.84	1.121e-05	644	1.97	1.121e-05
nd24k	2.8716e+07	879	34.95	4.5006e-11	879	131.93	3.1846e-11
Hook_1498	5.9374e+07	1904	460.66	9.0569e-14	1904	1392.00	1.2814e-13

Risultati MATLAB





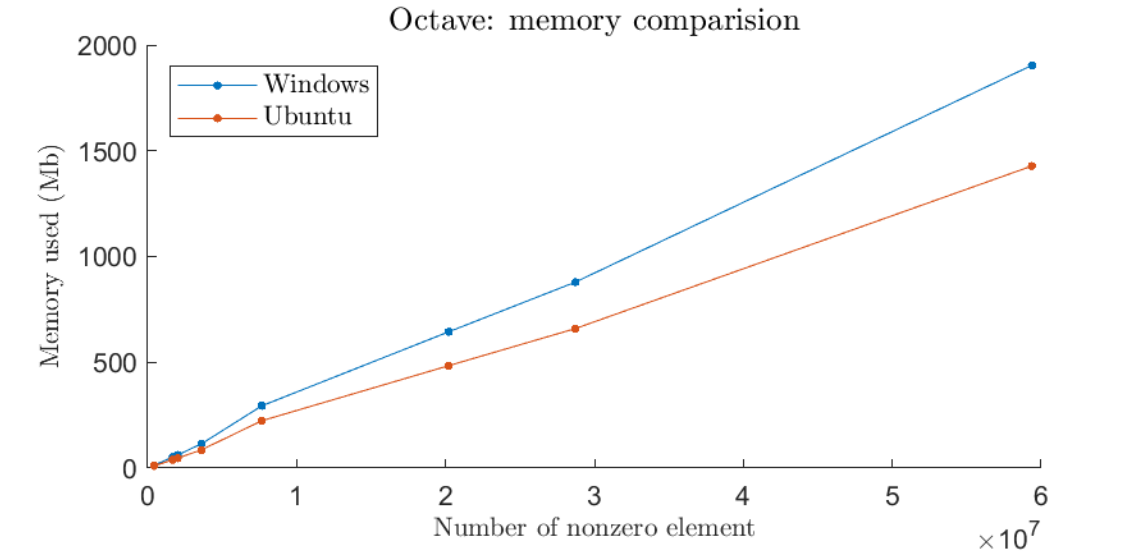


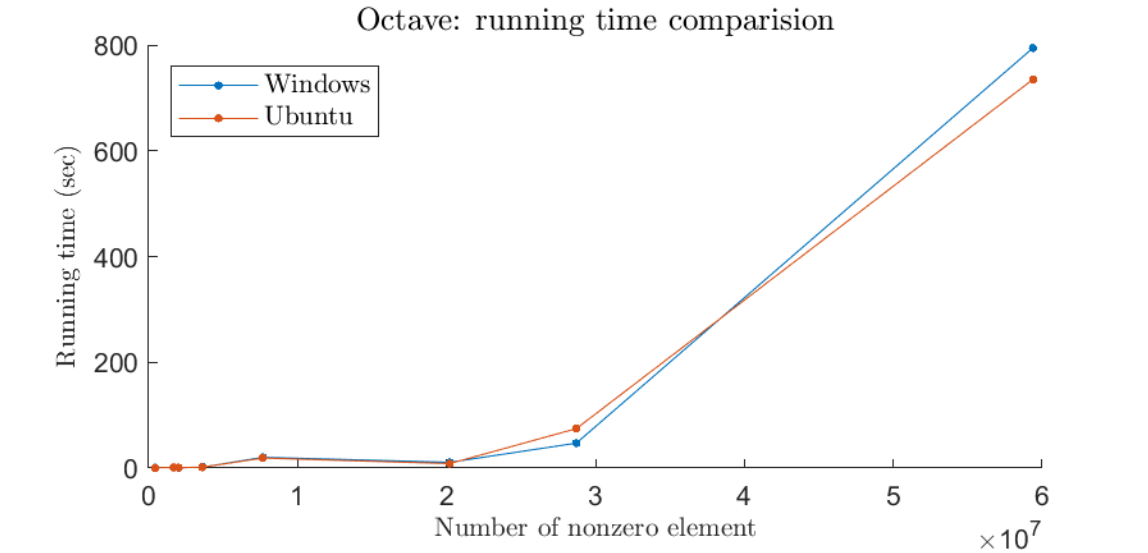


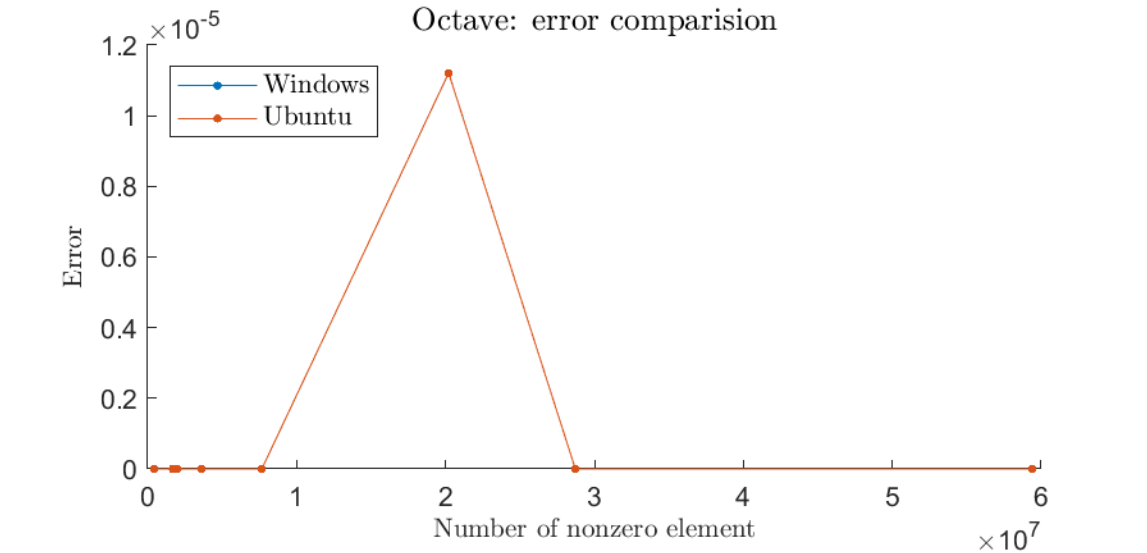
Risultati GNU Octave

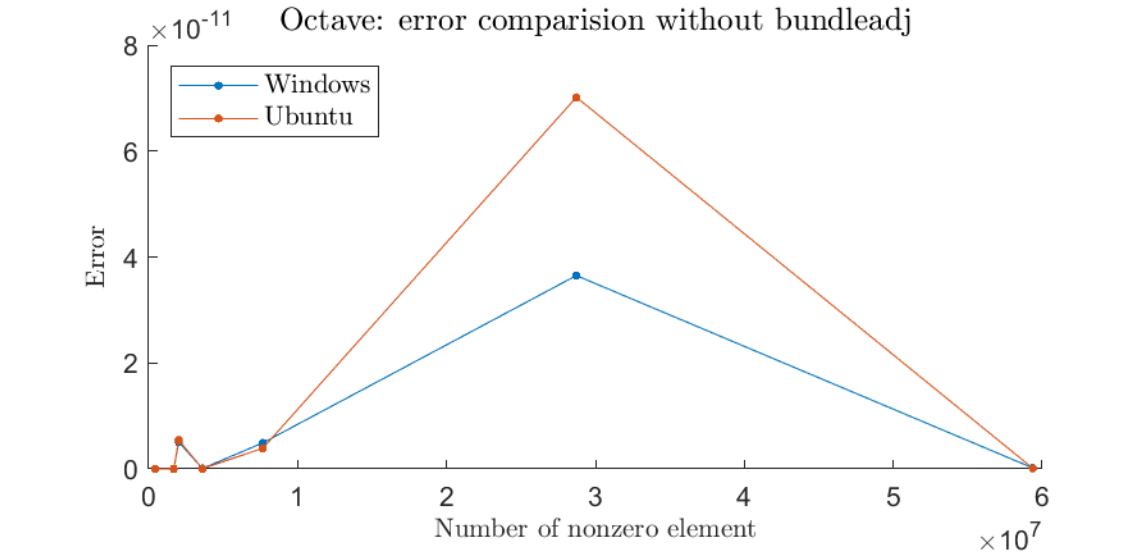
Nome	Numero zeri	Windows			Ubuntu		
		Memoria (Mb)	Tempo (secondi)	Accuratezza	Memoria (Mb)	Tempo (secondi)	Accuratezza
GT01R	4.3091e+05	14	0.72	4.4302e-15	10	0.62	2.4502e-15
ns3Da	1.6796e+06	52	1.20	9.8788e-16	39	1.32	9.1822e-16
TSC_OPF_1047	2.0128e+06	62	0.75	5.0719e-12	46	0.97	5.4378e-12
ifiss	3.5999e+06	114	1.74	8.8396e-14	86	1.62	5.7476e-14
G3_circuit	7.6608e+06	294	20.32	4.9098e-12	223	18.97	3.8989e-12
Bundle_adj	2.0208e+07	644	10.87	1.121e-05	484	8.48	1.121e-05
nd24k	2.8716e+07	879	47.16	3.6502e-11	659	74.62	7.0163e-11
Hook_1498	5.9374e+07	1904	794.28	1.7144e-13	1428	734.81	1.1002e-13

Risultati GNU Octave





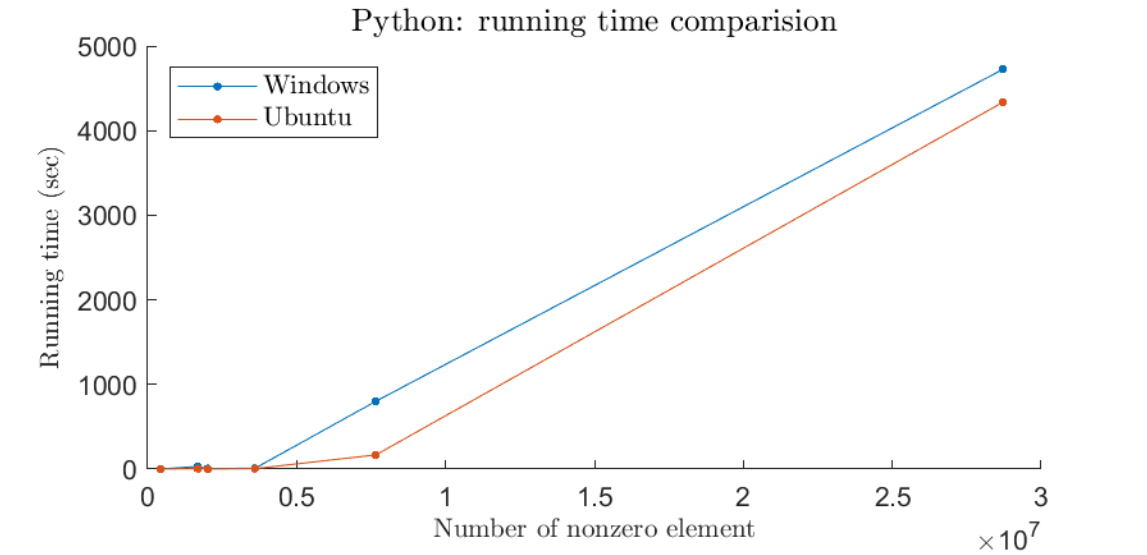


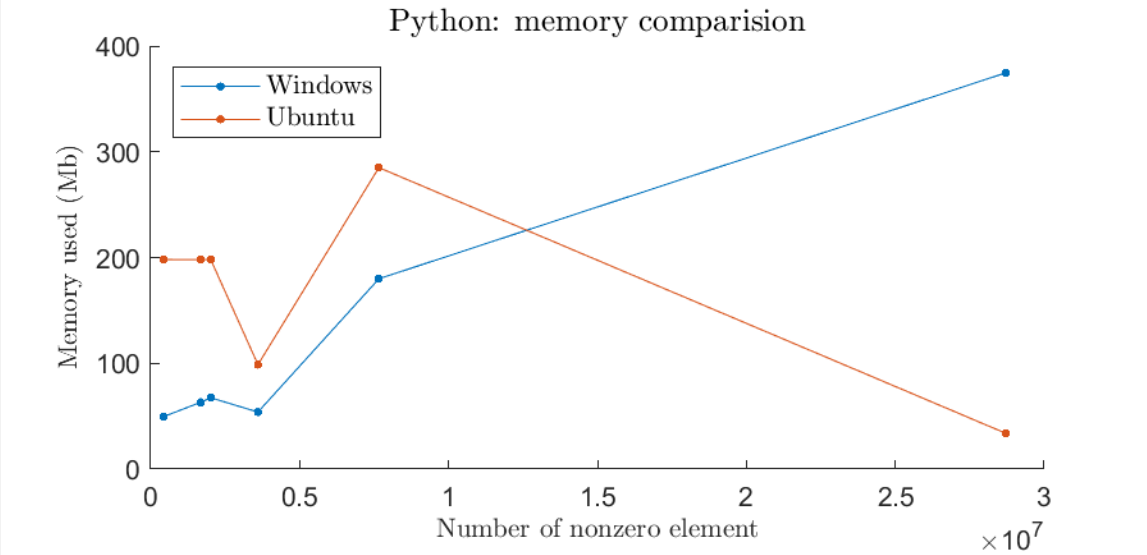


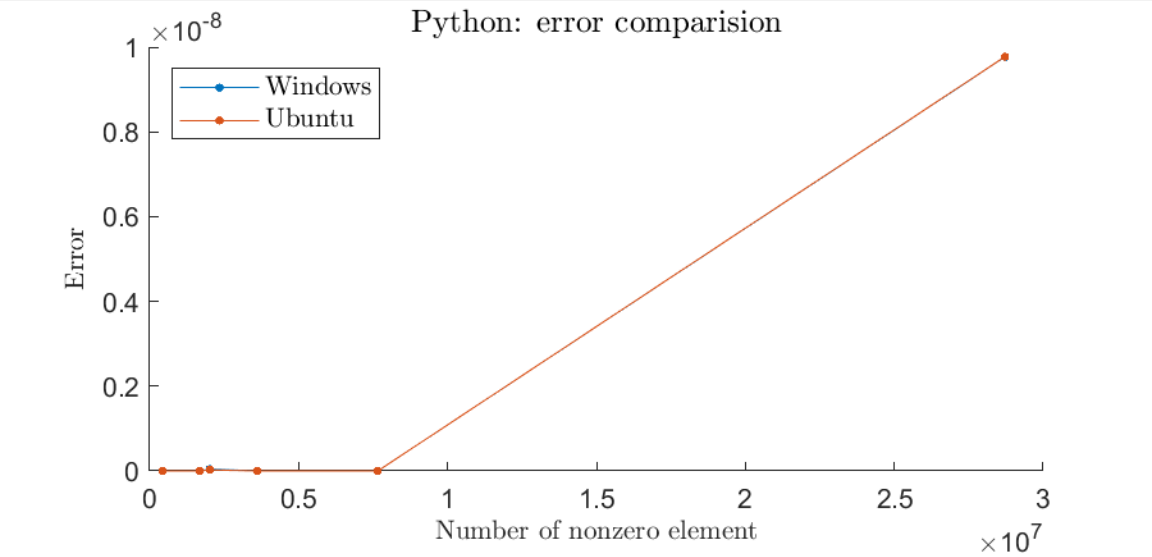
Risultati Python

Nome	Numero zeri	Windows			Ubuntu		
		Memoria (Mb)	Tempo (secondi)	Accuratezza	Memoria (Mb)	Tempo (secondi)	Accuratezza
GT01R	4.3091e+05	50	1.33	1.0046e-12	198	1.53	6.3836e-13
ns3Da	1.6796e+06	63	30.60	9.1691e-15	198	5.30	7.6691e-15
TSC_OPF_1047	2.0128e+06	67	7.32	4.1e-11	198	1.28	2.4244e-11
ifiss	3.5999e+06	54	8.79	4.8475e-14	99	5.53	3.8862e-14
G3_circuit	7.6608e+06	180	801.35	8.7327e-13	285	167.32	8.7526e-13
Bundle_adj	2.0208e+07	6	12085	0.022471	-	-	-
nd24k	2.8716e+07	375	4724.3	9.7688e-09	34	4335.40	9.7728e-09
Hook_1498	5.9374e+07	-	-	-	-	-	-

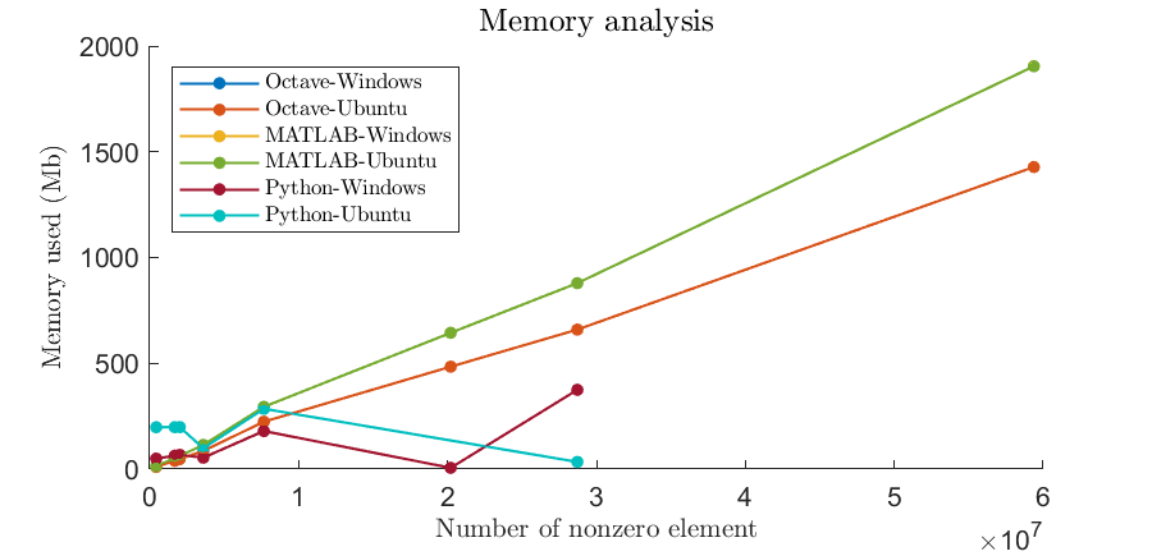
Risultati Python

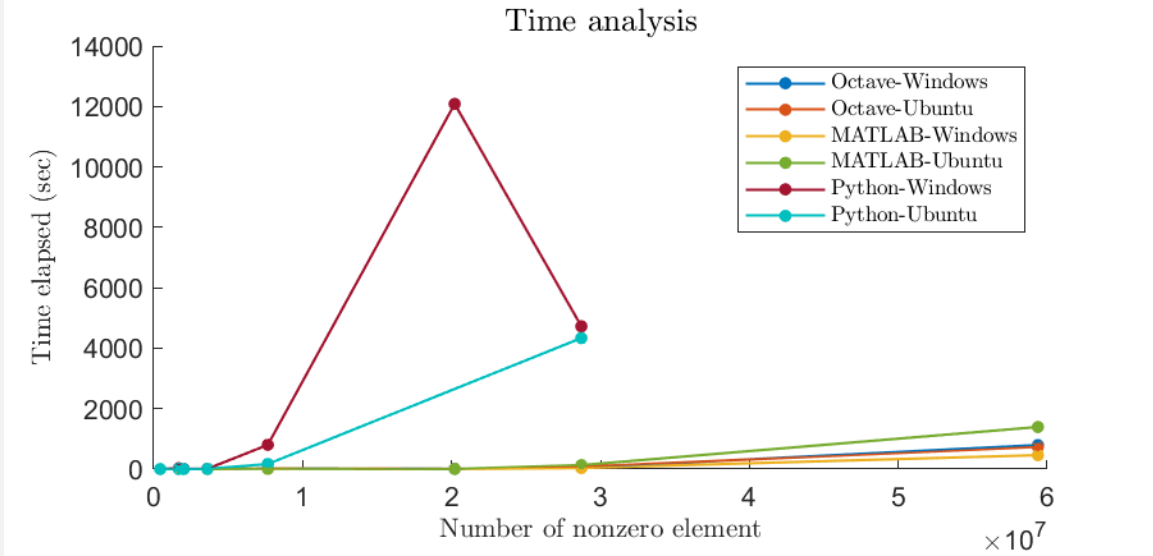


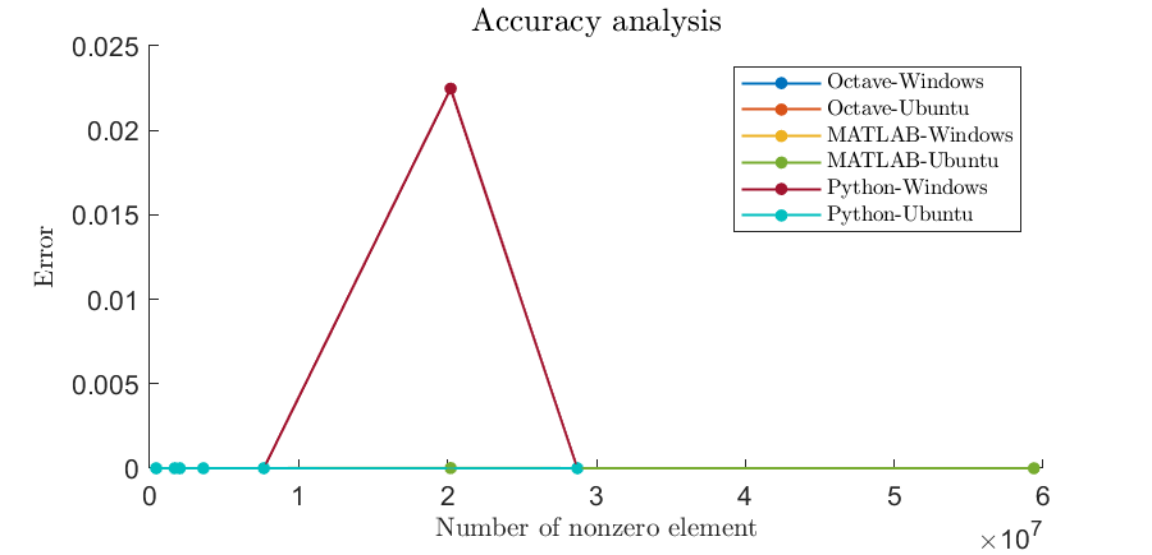




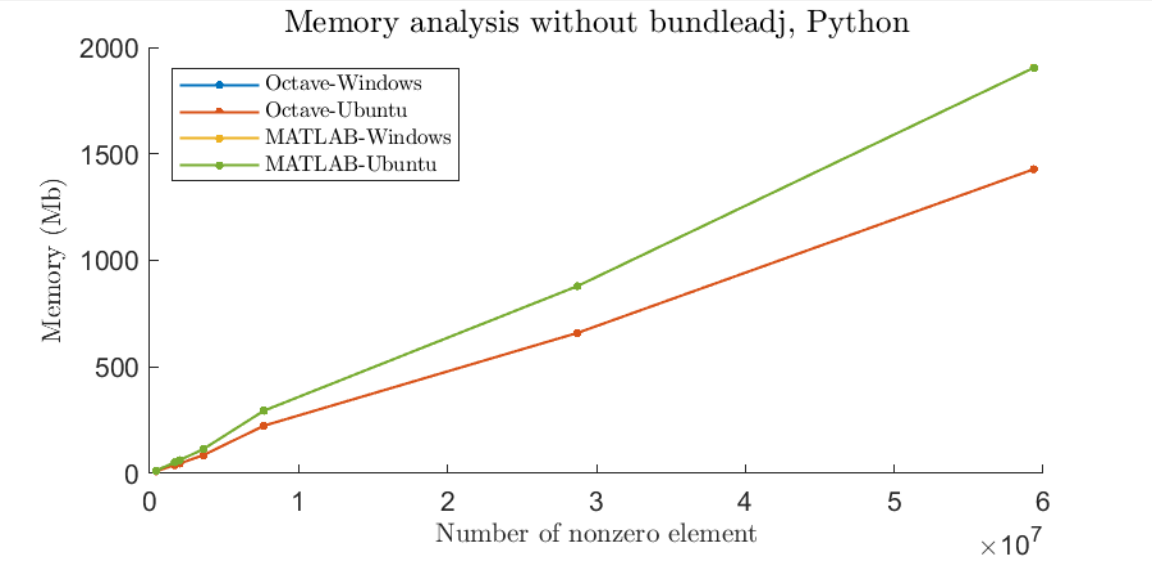
Risultati aggregati

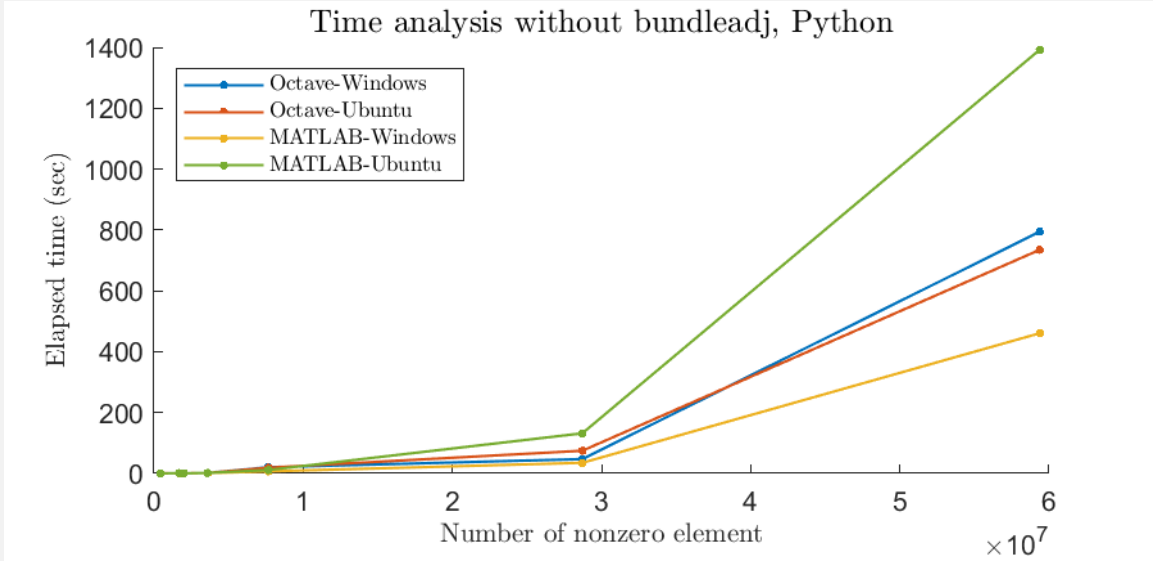


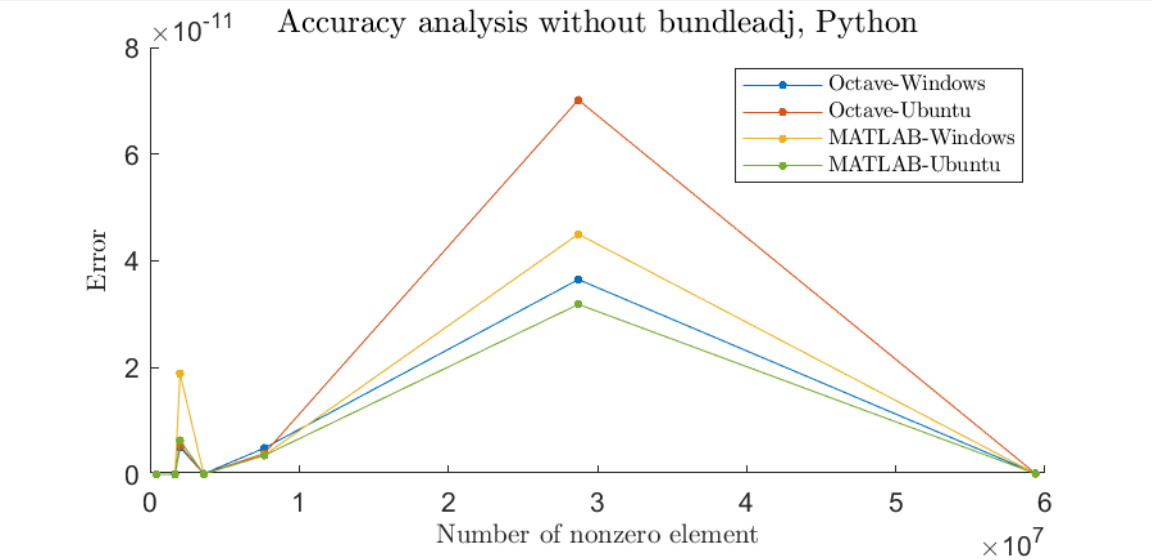




Risultati aggregati







Osservazioni Python

È in generale uno tra i linguaggi più semplici da utilizzare, partendo dall'installazione la quale richiede pochi secondi e non è necessario software dedicato. È possibile inoltre utilizzare anche il semplice prompt dei comandi.
Purtroppo, per quanto riguarda le librarie utilizzate durante il confronto, la documentazione messa a disposizione non è sufficiente ai fini di capire più precisamente quale sia l'effettivo procedimento eseguito dalle funzionalità messe a disposizione.

Facilità d'uso	Documentazione
Semplice	Carente

Osservazioni MATLAB

Risulta facile da utilizzare soprattutto grazie alla documentazione messa a disposizione da MathWorks, consultabile in maniera gratuita, dettagliata e ricca di contenuti. Matlab gode inoltre di un'ampia community sempre disponibile per risolvere eventuali dubbi e rendere noti possibili errori. Questo comporta anche un continuo aggiornamento del software per riparare possibili errori segnalati dagli utenti.

Facilità d'uso	Documentazione
Semplice	Completa e ben strutturata

Osservazioni GNU Octave

GNU Octave, seppur risulta essere un software gratuito, offre la maggior parte delle funzionalità offerte da MATLAB, in maniera chiaramente non sempre performante. Anche in questo caso il software è facile da utilizzare, e riporta anch'esso una ampia documentazione, purtroppo non sempre chiara e strutturalmente più complessa rispetto a quella offerta dal team di MathWorks.
Essendo un software open source, sono i singoli utenti a correggere gli errori all'interno del software: questo corrisponde a un aggiornamento non sempre continuo per migliorare possibili errori.

Facilità d'uso	Documentazione
Semplice	Parzialmente completa, non ben strutturata

Conclusioni

Dall'analisi effettuata e dai dati raccolti, possiamo effettuare alcuni ragionamenti:

- **Python** è risultato altamente inadeguato sia a livello di documentazione che di performance.
- **MATLAB** si è confermato molto performante e solido, anche grazie a un'ottima documentazione disponibile.
- **Octave** si è rivelato un’alternativa valida a MATLAB, con valori non molto distanti e anche migliori. Anche la documentazione e community sono altrettanto valide, anche se non ai livelli di MATLAB.

Considerando le combinazioni SO e linguaggio, possiamo affermare che l'acquisto di una sola licenza Windows (che potrebbe risultare utile anche in altri contesti) combinato con l'utilizzo di Octave sia la scelta migliore, in quanto garantisce velocità e accuratezza adeguati e una documentazione e supporto buoni.

Compressione di immagini

Alice Romagnoli - 829833
Andrei Gabriel Taraboi - 829904
Davide Rendina - 830730

Introduzione

Lo scopo di questo progetto è stato lo studio della DCT per effettuare la compressione di immagini. In particolare, questo progetto si divide in due parti:

- Implementazione della DCT2 e confronto dei risultati con librerie che implementano la FFT.
- Creazione di un software ad hoc che ci permetta di visualizzare concretamente il funzionamento della DCT2 applicata alle immagini.

Parte 1
Implementazione e confronto della DCT

Introduzione

La prima parte ha riguardato l'implementazione della DCT e il confronto con una libreria open source che implementa la FFT. In particolare, sono state effettuate le seguenti operazioni:

1. Sviluppo della funzione per il calcolo della DCT.
2. Esecuzione della DCT tramite le funzioni della libreria open source e la funzione sviluppata da noi, con raccolta dei tempi di esecuzione.
3. Analisi dei tempi di esecuzione raccolti attraverso grafici.

Tecnologia utilizzata

Per l'implementazione è stato scelto di utilizzare il linguaggio Python e *scipy,* la libreria open source di algoritmi e vari strumenti matematici. In particolare è stato utilizzato il modulo *scipy.fft*, che permette, tra le altre cose, di effettuare la DCT2 utilizzando la FFT. Questo è possible farlo attraverso il metodo *fft.dctn,* che inoltre ci ha permesso di impostare la *normalizzazione ortogonale*.

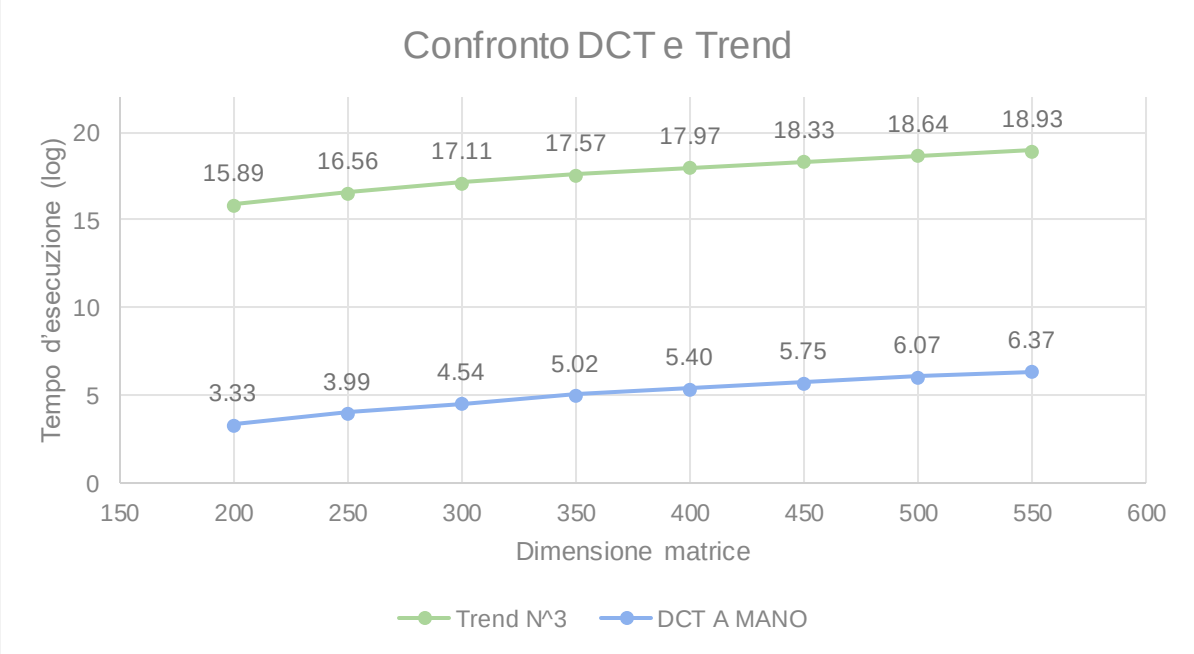
Implementazione

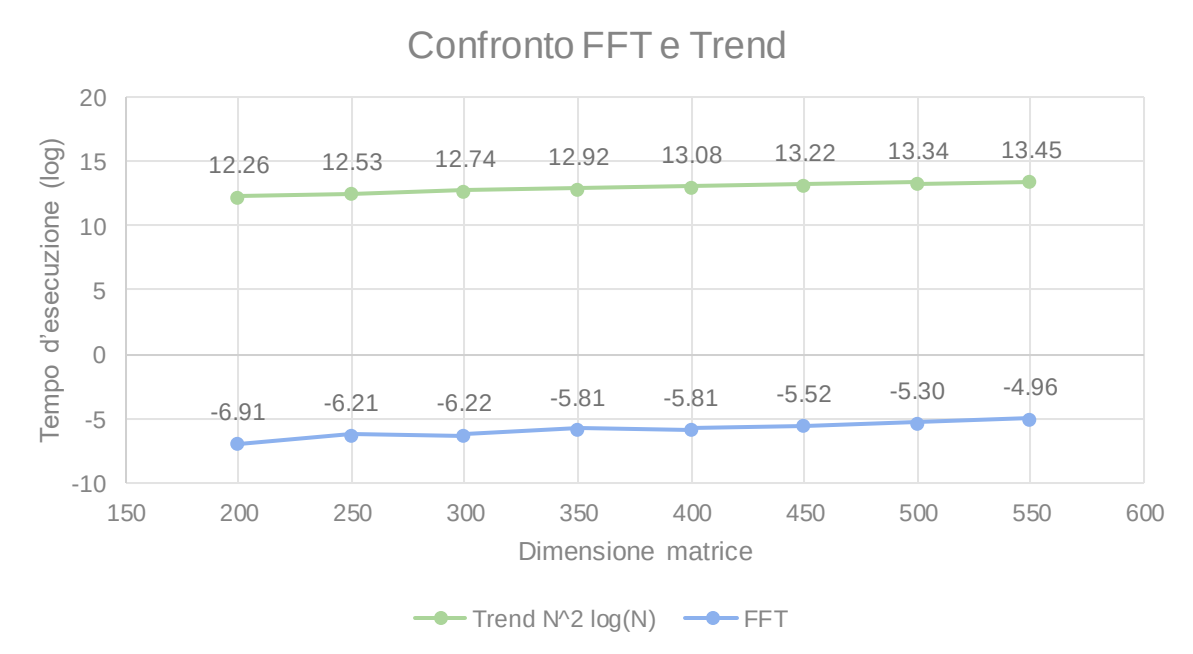
Abbiamo effettuato il confronto della DCT su 8 differenti matrici generate randomicamente, con dimensione compresa tra 200x200 e 550x550.

Sotto è possibile inoltre osservare l'implementazione del metodo DCT, con indicazioni sulle varie misure calcolate. Questo algoritmo è stato applicato prima sulle righe e in seguito sulle colonne della matrice, per ottenere il calcolo della DCT2.

```
def DCT_riga(matrice_iniziale):
    dct_per_riga = np.zeros(np.shape(matrice_iniziale))
    for i in range(0, DIM[0]):
        for k in range(0, DIM[1]):
            alpha = math.sqrt(1 / DIM[1]) if k == 0 else math.sqrt(2 / DIM[1])
            value = ['None'] * DIM[1]
            for j in range(0, DIM[1]):
                value[j] = (math.cos(
                    math.pi * k * (2 * j + 1) / (2 * DIM[1])) * matrice_iniziale[i][j])
            dct_per_riga[i][k] = (alpha * np.sum(value))
    return DCT_colonna(dct_per_riga)
```

Analisi grafica





Per verificare l'effettiva proporzionalità di DCT A MANO a N^3 e di FFT a N^2 log(N), sono state calcolate le correlazioni tra le coppie di valori, riscontrando che effettivamente si avvicinano di molto a 1 (DCT A MANO: 0,999971216; FFT: 0,975457271).

Conclusioni

Attraverso il confronto effettuato tra la DCT2 sviluppata da noi e quella fornita dalla libreria *scipy*, abbiamo constatato che i risultati ottenuti sono quelli attesi, poiché:

- La DCT2 sviluppata da noi ha una complessità di $O(N^3)$;
- La DCT2 fornita dalla libreria *scipy* ha complessità $O(N^2 \log(N))$.

Parte 2: Software per la compressione d’immagini

Introduzione

È stata sviluppata un’applicazione che permette di effettuare la compressione di un’immagine bitmap.

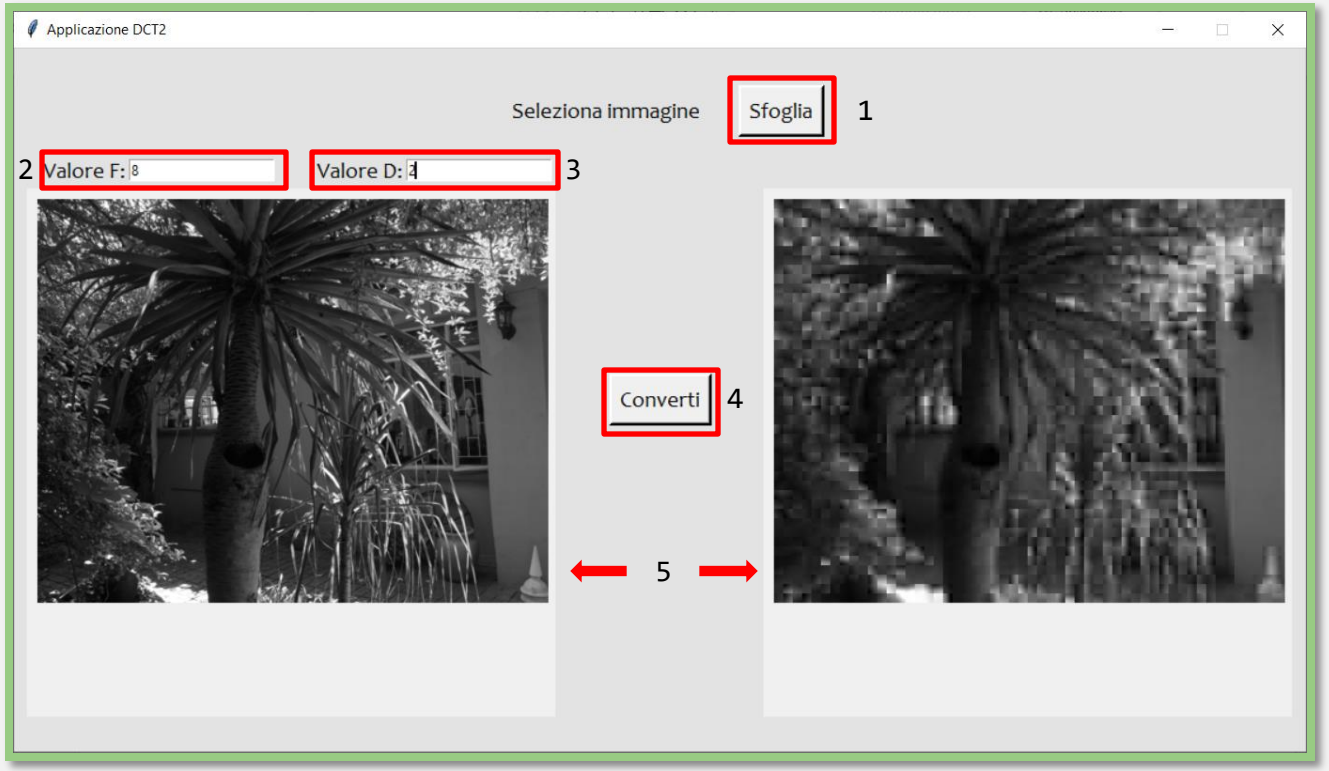
L'applicazione sviluppata presenta le seguenti caratteristiche:

- Interfaccia grafica per la selezione di immagini e parametri;
- Visualizzazione dell'immagine di partenza e dell'immagine prodotta;
- Esecuzione dell’algoritmo di compressione sull’immagine.

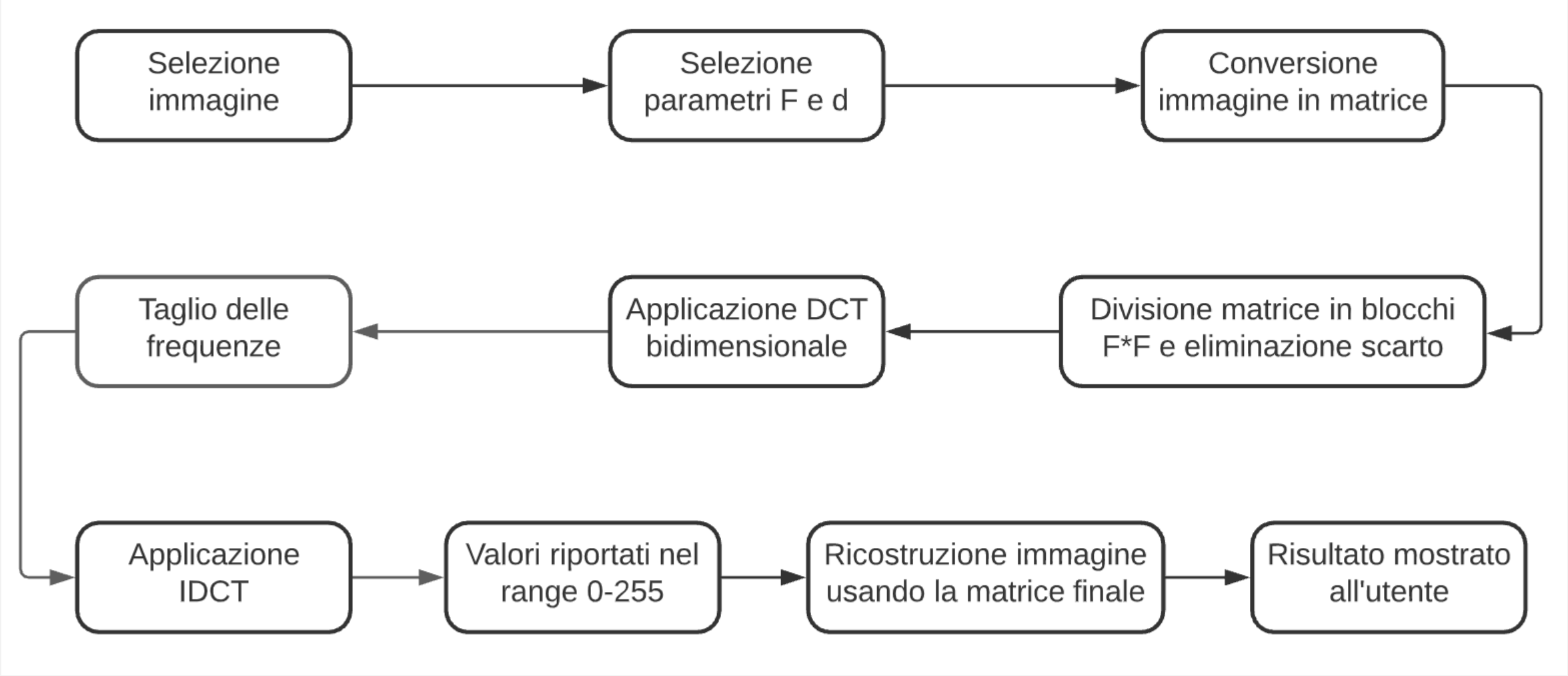
Interfaccia utente

L’interfaccia è stata sviluppata utilizzando la libreria Pygubu (wrapper di Tkinter), e permette di:

1. Selezionare un immagine dal dispositivo
2. Impostare la grandezza della finestra F, utilizzata durante la compressione
3. Impostare il valore per effettuare il taglio delle frequenze
4. Effettuare la compressione dell’immagine
5. Visualizzare l’immagine originale e il risultato ottenuto



Funzionamento



Implementazione

Attraverso le righe di codice riportate affianco, vengono:

- calcolati il numero di blocchi per riga e per colonna della matrice rappresentante l'immagine.
- eliminate le righe e le colonne in eccesso della matrice, ovvero quelle per le quali non si riesce a costruire il blocco di dimensione FxF.

```
numero_blocchi_per_riga = int(matrice.shape[0] / F)
numero_blocchi_per_colonna = int(matrice.shape[1] / F)

righeDaTogliere = matrice.shape[0] % F
colonneDaTogliere = matrice.shape[1] % F

if righeDaTogliere != 0:
    matrice = matrice[:-righeDaTogliere, :]
if colonneDaTogliere != 0:
    matrice = matrice[:, :-colonneDaTogliere]
```

Implementazione

Vengono presi i blocchi della matrice (partendo dal primo in alto a sinistra).

Su ogni blocco, viene eseguita la DCT2 e viene effettuato il taglio delle frequenze.

In seguito, viene eseguita la IDCT2 e i valori vengono arrotondati all'intero più vicino, ponendoli nel range da 0 a 255.

Infine il blocco modificato viene sovrascritto nella sua posizione.

```
for i in range(numero_blocchi_per_riga):
    for j in range(numero_blocchi_per_colonna):
        blocco = matrice[i * F:(i + 1) * F, j * F:(j + 1) * F]

        x = fft.dctn(blocco, norm="ortho", type=2)
        for k in range(x.shape[0]):
            for l in range(x.shape[1]):
                if k + l >= d:
                    x[k][l] = 0

        y = fft.idctn(x, norm="ortho", type=2)
        for k in range(y.shape[0]):
            for l in range(y.shape[1]):
                y[k][l] = round(y[k][l])
                if y[k][l] < 0:
                    y[k][l] = 0
                if y[k][l] > 255:
                    y[k][l] = 255
        matrice[i * F:(i + 1) * F, j * F:(j + 1) * F] = y
```

Risultati

Esempio di risultati ottenuti eseguendo la compressione fissando i parametri F=170, d=50.

Risultati

Esempio di risultati ottenuti eseguendo la compressione fissando i parametri F=227, d=150

Risultati

Esempio di risultati ottenuti eseguendo la compressione fissando i parametri F=632, d=150

Risultati

Esempio di risultati ottenuti eseguendo la compressione fissando i parametri F=1500, d=150

Risultati

Esempio di risultati ottenuti eseguendo la compressione fissando i parametri F=1000, d=250

Risultati

Esempio di risultati ottenuti eseguendo la compressione fissando i parametri F=1000, d=20

Conclusioni

Come si può vedere dai risultati, all’aumentare del valore di d, la qualità delle immagini migliora e viceversa.

All’aumentare del valore di F invece, cambia il numero di blocchi creati e la porzione di immagine da scartare (perchè non divisibile per F).

Il valore di F influisce indirettamente anche sulla qualità dell’immagine, in quanto a parità di d, viene eseguito un taglio delle frequenze più drastico se i blocchi sono pochi e grandi.

Per una migliore visione dei risultati ottenuti è possible osservarli all’interno della cartella Drive.