



# Progetto di Metodi del Calcolo Scientifico

Tonelli Lidia Lucrezia (m. 813114)

Grassi Marco (m. 830694)

Giudice Gianluca (m. 829664)

University of Milano Bicocca

Giugno 2021



## Metodi diretti per matrici sparse

Approccio al problema

Analisi delle librerie

Campagna sperimentale

## JPEG

Custom DCT2

Custom JPEG



## Metodi diretti per matrici sparse

Approccio al problema

Analisi delle librerie

Campagna sperimentale

## JPEG

Custom DCT2

Custom JPEG



# Sistema operativo e hardware

## Sistema operativo (installazione da zero)

- Windows 10 Pro
- Ubuntu 20.04 LTS

## Hardware

- CPU: Intel Core i7-8550U 4 x 1.8 - 4 GHz
- RAM: 32 GB, DDR4-2400



## Ambienti di programmazione e librerie utilizzate

Abbiamo utilizzato 3 ambienti di programmazione; per Python abbiamo usato 3 librerie, di cui una necessaria per ottimizzare il calcolo su matrici molto grandi.

- Matlab R2021a
- GNU Octave 6.1.0
- Python 3.8.7
  - numpy 1.20.3
  - scipy 1.6.3
  - scikit-sparse 0.4.4

N.B.: Per entrambi i sistemi operativi é stata utilizzata la stessa versione di libreria in modo da avere risultati conmparabili.



## Metriche di performance

Per misurare le performance sono state utilizzate le seguenti metriche:

- Tempo di calcolo della soluzione
- Piccola memoria RAM utilizzata per risolvere il sistema
- Errore della soluzione:

$$err\_rel = \frac{\|x - x_e\|_2}{\|x_e\|_2}$$

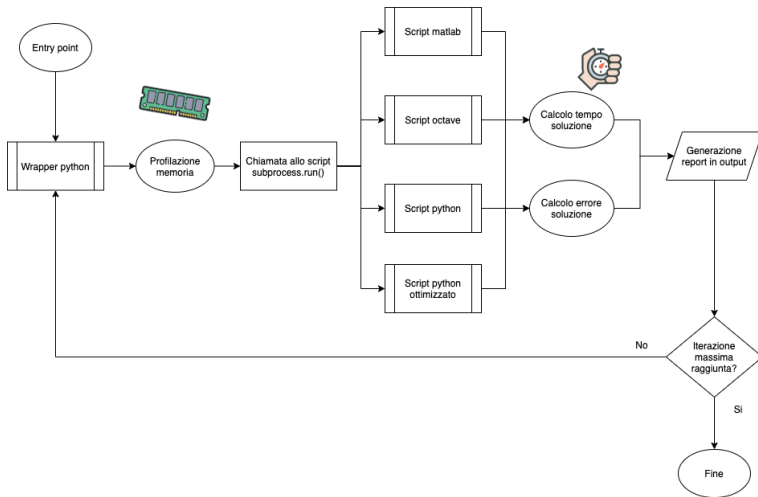
con  $\|\cdot\|$  la norma euclidea dei vettori



## Tecniche di misurazione

1. Creazione dei 4 script per il calcolo della soluzione di una matrice
  - INPUT: file matrice
  - OUTPUT: File di report
2. 10 iterazioni per il calcolo delle misure (in modo tale da avere risultati statisticamente validi)
  - Script python con funzione di wrapper per profilare la memoria utilizzata.
    - 2.1 Run di ogni script per ogni matrice su ogni sistema operativo.
    - 2.2 Misurazione del tempo (si considera solo il tempo per il calcolo della soluzione)
    - 2.3 Misurazione dell'errore
3. Analisi dei report generati

## Tecniche di misurazione







## Metodi diretti per matrici sparse

Approccio al problema

**Analisi delle librerie**

Campagna sperimentale

## JPEG

Custom DCT2

Custom JPEG



# Matlab

Matlab é ben documentato alla pagina  
[it.mathworks.com/help/matlab](http://it.mathworks.com/help/matlab), con motore di ricerca ed  
esempi.

**Memorizzazione di matrici sparse:** data una matrice sparsa,  
Matlab salva solo gli elementi diversi da 0 in una lista, insieme al  
numero di colonna e riga.



# Matlab

**Risoluzione di  $Ax = b$ :** Matlab ha a disposizione tanti risolutori di sistemi, il cui uso dipende dalla forma della matrice  $A$ ; pertanto, quando viene eseguito il comando  $A \setminus b$  si fanno una serie di controlli su  $A$ , i casi presi in considerazione sono: matrice quadrata, triangolare, triangolare permutata, Hermitiana, Hessenberg superiore, la matrice ha la diagonale tutta positiva o tutta negativa, la matrice é simmetrica e definita positiva. In tutti questi casi viene usato un solutore diverso, ad esempio se la matrice é simmetrica e definita positiva si applica l'algoritmo di Cholesky.



# Octave

Octave é documentato in modo scomodo rispetto a Matlab alla pagina <https://octave.org/doc/v6.2.0>, che é un semplice manuale.

**Memorizzazione di matrici sparse:** usa il *compressed column format*, ovvero salva solo gli elementi diversi da 0 con il proprio numero di riga, e per ogni colonna salva il numero di elementi diversi da 0 per quella colonna; pertanto al posto di una tripletta per ogni elemento diverso da 0, si salva una coppia e in piú un numero per ogni colonna, ottimizzando la memorizzazione.



# Octave

**Risoluzione di  $Ax = b$ :** si comporta allo stesso modo di Matlab, ovvero considera la forma della matrice  $A$  per scegliere il risolutore adatto. Anche Octave, come Matlab, prova ad applicare Cholesky e se non riesce significa che la matrice non é definita positiva e/o simmetrica.



## Python (*numpy* + *scipy* + *scikit-sparse*)

Python è un linguaggio di programmazione general purpose. Abbiamo utilizzato 3 librerie apposite per il calcolo scientifico, ognuna con un compito diverso.

1. *numpy*: libreria utilizzata per leggere e gestire le matrici in memoria
2. *scipy*: libreria specifica per il calcolo scientifico. Utilizza metodi diretti per il calcolo della soluzione di un sistema. Il metodo standard per la risoluzione diretta di sistemi lineari non è ottimizzato come in Matlab o Octave, ovvero non divide i risolutori in base alla matrice. Supporta la fattorizzazione LU tuttavia non è abilitata di default; non supporta Cholesky su matrici sparse.
3. *scikit-sparse*: libreria che permette di applicare Cholesky su matrici sparse; è implementata in c.



## Python (*numpy* + *scipy* + *scikit-sparse*)

*numpy* e *scipy* é documentato peggio di Matlab ma meglio di Octave: ci sono molti esempi ed é disponibile il codice essendo open source, anche non essendo presente una buona spiegazione del workflow come in Matlab, si può capire leggendo il codice; la documentazione si trova alla pagina

<https://www.scipy.org/docs.html>.

*sciki-sparse* non ha una documentazione chiara, ma anche in questo caso, essendo open source, é possibile leggere direttamente il codice; la documentazione si trova alla pagina <https://scikit-sparse.readthedocs.io/en/latest/cholmod.html>.



## Python (*numpy* + *scipy* + *scikit-sparse*)

**Memorizzazione di matrici sparse:** *numpy* salva le matrici sparse come Matlab.

**Risoluzione di  $Ax = b$ :** il risolutore assume che la soluzione  $x$  sia sparsa, perché può capitare spesso; se invece  $x$  è densa, la costruzione risulta molto più costosa. Utilizza sempre la fattorizzazione LU per risolvere il sistema, quindi il suo comportamento non è differente nel caso di matrici simmetriche e definite positive, perché non utilizza mai Cholesky; per questo motivo abbiamo introdotto la libreria *scikit-sparse*, che permette di applicare Cholesky a grandi matrici sparse, perciò abbiamo potuto adattare l'algoritmo scritto in Python ai casi di matrici simmetriche e definite positive.





## Nuovo workflow con *scikit-sparse*



## Metodi diretti per matrici sparse

Approccio al problema

Analisi delle librerie

Campagna sperimentale

## JPEG

Custom DCT2

Custom JPEG



## Proprietá delle matrici

Matrice	Simmetrica	Def. pos.	Cand. Cholesky
Hook_1498	Y	Y	Y
G3_circuit	Y	Y	Y
nd24k	Y	Y	Y
boundle_adj	Y	Y	Y
ifiss_mat	N	N	N
TSC_OPF_1047	Y	N	N
ns3Da	N	N	N
GT01R	N	N	N

Durante l'esecuzione di ogni script si suppone di non essere a conoscenza di queste informazioni, così da rimanere il più generali possibili.



## Dimensione delle matrici

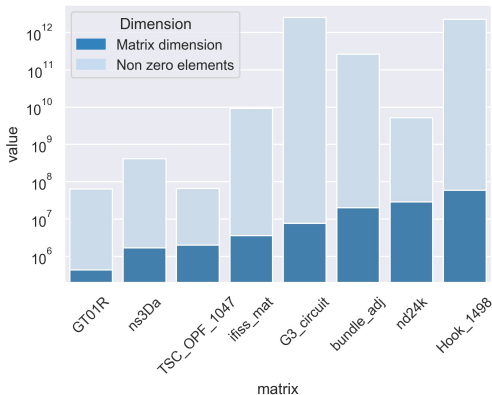


Figure: Confronto delle dimensioni e del numero di nnz delle matrici.



## Grafici performance

Confronto tra Matlab, Octave, Python su Windows o Linux per i parametri velocità, precisione e occupazione di memoria.

# Tempo di esecuzione

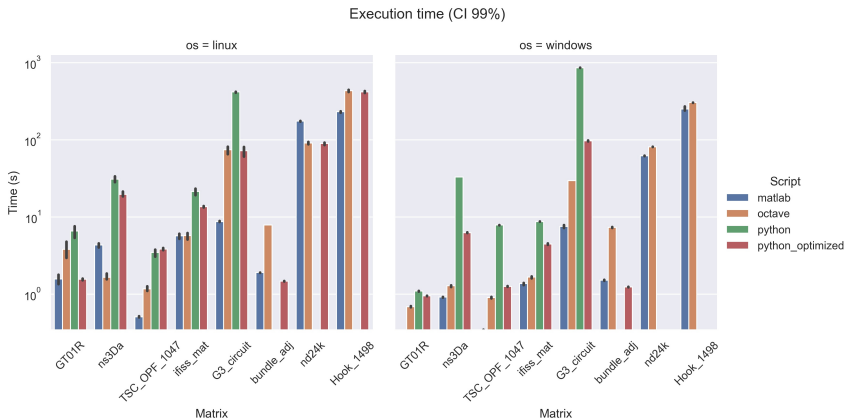


Figure: Confronto tempo di esecuzione.



# Memoria utilizzata

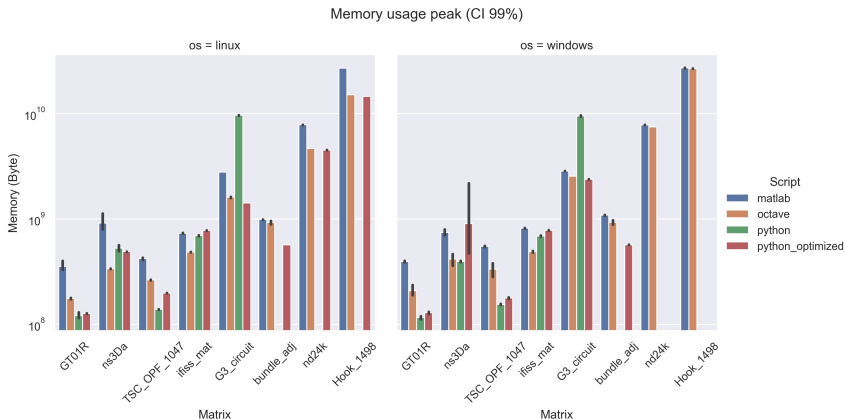


Figure: Picco di memoria utilizzata (in byte).



# Tempo di esecuzione

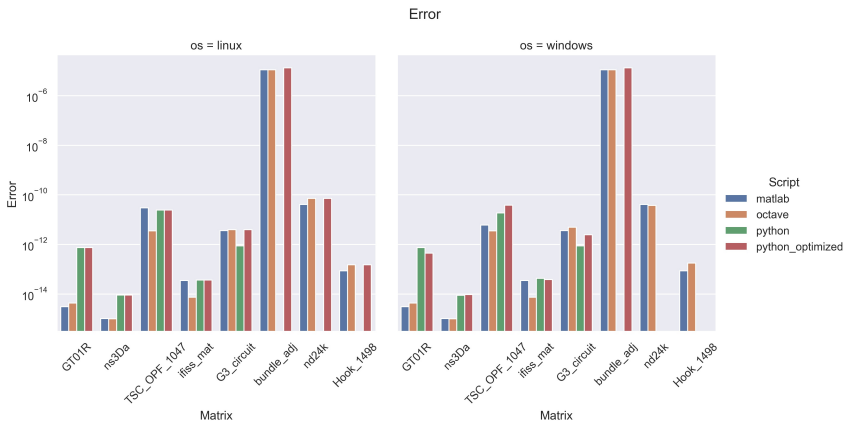


Figure: Confronto errore relativo soluzione.





## Codice

Script per il calcolo delle soluzione utilizzando le diverse librerie.



## Codice - Matlab

---

```

1 function [] = matlab_solver(matrix_file , result_file)
2
3     load(matrix_file);           % Carica matrice dato il path
4     A = Problem.A;              % Accesso effettivo alla matrice
5
6     xe = ones(length(A(:,1)), 1); % Ground truth soluzione
7     b = A * xe;                 % Calcolo termine noto data soluzione
8
9     tic;                         % Inizio timer
10    x = A \ b;                   % Calcolo della soluzione
11    time = toc;                  % Tempo impiegato
12
13    % Statistiche
14    er = norm(xe - x) / norm(xe); % Errore relativo
15    m_size = numel(A);           % Dimensione matrice
16    m_nnz = nnz(A);              % Elementi non zero matrice
17
18    fid = fopen(result_file , "a+"); % Apri file in append
19    fprintf(fid , "%d;%d;%d;%d", m_size, m_nnz, er, time);
20    fclose(fid);                 % Chiudi il file
21
22 end

```

---



## Codice - Octave

---

```

1  args = argv();           # Argomenti da linea di comando
2  matrix_file = args{1};   # Path matrice input
3  result_file = args{2};   # Path matrice output
4
5  load(matrix_file);       # Carica la matrice dato il path
6
7  A = Problem.A;           # Accesso effettivo alla matrice
8  xe = ones(rows(A), 1);   # Ground truth soluzione
9  b = A * xe;              # Calcolo termine noto data soluzione
10
11 tic;                      # Inizio timer
12 x = A \ b;               # Calcolo della soluzione
13 time = toc;              # Tempo impiegato
14
15 er = norm(xe - x) / norm(xe); # Errore relativo
16 m_size = numel(A);        # Dimensione della matrice
17 m_nnz = nnz(A);          # Elementi non zero
18
19 fid = fopen(result_file, "a+"); # Apri file
20 fprintf(fid, "%d;%d;%d;%d", m_size, m_nnz, er, time);
21 fclose(fid);             # Chiudi file

```

---



## Codice - Python

---

```

1 matrix_file = sys.argv[1]           # Path matrice input
2 result_file = sys.argv[2]          # Path matrice output
3
4 struct = loadmat(matrix_file)
5
6 A = struct['Problem']['A'][0, 0]     # Accesso matrice
7 xe = np.ones(A.shape[0])           # Ground truth soluzione
8 b = A.dot(xe)                       # Termine noto dato xe
9
10 start_time = time.time()            # Start timer
11 x = spsolve(A, b)                   # Risoluzione sistema
12 elapsed = time.time() - start_time  # Tempo trascorso
13
14 er = norm(xe - x) / norm(xe)        # Errore relativo
15 m_size = A.shape[0] * A.shape[1]    # Dimensione matrice
16 nnz = A.nnz                         # Elementi non zero della matrice
17
18 # Scrittura risultati su file
19 with open(result_file, 'a+') as f:
20     f.write(f'{m_size};{nnz};{er};{elapsed}')
```

---



## Codice - Python ottimizzato

---

```

1 matrix_file = sys.argv[1]          # Path matrice input
2 result_file = sys.argv[2]          # Path matrice output
3
4 struct = loadmat(matrix_file)
5
6 A = struct['Problem']['A'][0, 0]    # Accesso matrice
7 xe = np.ones(A.shape[0])           # Ground truth soluzione
8 b = A.dot(xe)                       # Termine noto dato xe
9
10 start_time = time.time()           # Start timer
11 try:
12     # Calcolo soluzione con Cholesky
13     factor = cholesky(A)             # Fattorizzazione con Cholesky
14     x = factor(b)                    # Calcolo soluzione con fattorizzazione
15 except CholmodNotPositiveDefiniteError:
16     # Eccezione: La matrice non e' definita positiva
17     # Calcolo soluzione utilizzando fattorizzazione LU
18     x = spsolve(A, b, use_umfpack=False)
19 elapsed = time.time() - start_time  # Tempo trascorso
20
21 er = norm(xe - x) / norm(xe)        # Errore relativo
22 m_size = A.shape[0] * A.shape[1]    # Dimensione matrice
23 nnz = A.nnz                         # Elementi non zero della matrice
24
25 # Scrittura risultati su file
26 with open(result_file, 'a+') as f:
27     f.write(f'{m_size};{nnz};{er};{elapsed}')
```

---



## Metodi diretti per matrici sparse

Approccio al problema

Analisi delle librerie

Campagna sperimentale

## JPEG

Custom DCT2

Custom JPEG



## Metodi diretti per matrici sparse

Approccio al problema

Analisi delle librerie

Campagna sperimentale

## JPEG

Custom DCT2

Custom JPEG



## Algoritmo usato

DCT2 fatta con numpy e FFT per confrontare presa da scipy.fft  
array crescenti di dimensione  $2^i$





# Algoritmo

---

```

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

```

---



# Algoritmo

---

```

1 def dct2_personal(a):
2     n = len(a)  # rows
3     m = len(a[0])  # columns
4     c = np.zeros((n, m))  # result matrix
5     # dct by rows
6     for i in range(n):
7         c[i] = dct_personal(a[i])  # i-th row
8     # dct by columns
9     for j in range(m):
10        c[:, j] = dct_personal(c[:, j])

```

---



# Algoritmo

---

```

1 lib_times = []
2 my_times = []
3
4 for i in range(4, 12):
5     N = 2 ** i
6     a = np.random.rand(N, N)
7
8     lib_ti = time.time()
9     lib_dct = dct(dct(a, axis=1, norm="ortho"), axis=0,
10 norm="ortho")
11     lib_tf = time.time()
12     lib_times.append(lib_tf - lib_ti)
13
14     my_ti = time.time()
15     my_dct = dct2_personal(a)
16     my_tf = time.time()
17     my_times.append(my_tf - my_ti)

```

---



## Notizie su scipy.fft

`docs.scipy.org/doc/scipy/reference/generated/scipy.  
fftpack.dct.html`

For a single dimension array  $x$ , `dct(x, norm='ortho')` is equal to MATLAB `dct(x)`. Ma é fft? Mi sa di no, provare fft

`docs.scipy.org/doc/scipy/reference/tutorial/fft.html`



## Confronto tra Custom DCT2 e FFT

grafici - la dct é  $O(N^3)$ , la FFT é  $O(N^2)$



## Metodi diretti per matrici sparse

Approccio al problema

Analisi delle librerie

Campagna sperimentale

## JPEG

Custom DCT2

Custom JPEG



# Algoritmo

---

```

1 import math
2 import numpy as np
3 from PIL import Image
4 from scipy.fft import dct
5 from scipy.fft import idct

```

---



---

```

1 ### fissa un range di valori possibili per n
2 def clamp(self, n, smallest, largest):
3     return max(smallest, min(n, largest))
4
5 ### forza n ad un valore intero tra 0 e 255
6 def fix_number(self, n):
7     return self.clamp(round(n), 0, 255)

```

---



# Algoritmo

---

```

1 def pseudo_jpeg(self, img_path, f, d):
2     img = Image.open(img_path)
3     img_mat = np.array(img)
4     c_mat = np.zeros_like(img_mat)
5     rows, columns = img_mat.shape
6
7     max_i = math.floor(rows / f)
8     max_j = math.floor(columns / f)
  
```

---





# Algoritmo

---

```

1  for i in range(max_i):
2      for j in range(max_j):
3
4          ## Applicazione DCT2
5          ## Slice della matrice fxf
6          block = img_mat[i * f: (i + 1) * f, j * f: (j + 1) * f]
7          block = dct(dct(block, axis=1, norm="ortho"), axis=0, norm="ortho")

```

---



# Algoritmo

---

```

1      ### Eliminazione elementi sotto diagonale
2      block_rows, block_columns = block.shape
3      for k in range(block_rows):
4          for l in range(block_columns):
5              if (k + l) >= d:
6                  block[k, l] = 0
  
```

---



# Algoritmo

---

```

1      ## Applicazione IDCT2
2      block = idct(idct(block, axis=1, norm="ortho"), axis=0, norm="ortho")
3
4      ## fix dei numeri
5      for k in range(block.shape[0]):
6          for l in range(block.shape[1]):
7              block[k, l] = self.fix_number(block[k, l])
8
9      c_mat[i * f: (i + 1) * f, j * f: (j + 1) * f] = block
10
11     return c_mat

```

---



## Esempi con le immagini proposte

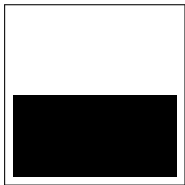


Figure: Originale 400x400

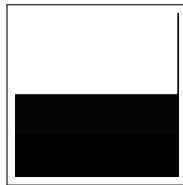


Figure:  $f = 99$ ,  $d = 1$



## Esempi con le immagini proposte



Figure:  $f = 99$ ,  $d = 20$

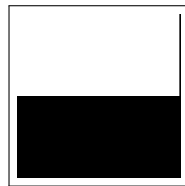


Figure:  $f = 99$ ,  $d = 196$



# Sperimentazione

Momento della presentazione in cui facciamo girare il nostro programma



# Grazie per l'attenzione