



**UNIVERSITÀ
DEGLI STUDI
DI UDINE**
hic sunt futura

**Dipartimento di Scienze
Matematiche, Informatiche e Fisiche**

**TESI DI LAUREA IN
INFORMATICA**

Un algoritmo per il calcolo randomizzato della SVD

CANDIDATO

Massimo Fedrigo

RELATORE

Prof. Enrico Bozzo

CORRELATRICE

Prof.ssa Rossana Vermiglio

CONTATTI DELL'ISTITUTO

Dipartimento di Scienze Matematiche, Informatiche e Fisiche

Università degli Studi di Udine

Via delle Scienze, 206

33100 Udine — Italia

+39 0432 558400

<https://www.dmif.uniud.it/>

Sommario

In questa tesi applicheremo un algoritmo randomizzato [1] al calcolo della Singular Value Decomposition (SVD).

Nel Capitolo 1 introduciamo la Randomized Numerical Linear Algebra (RandNLA), le sue applicazioni e la sua importanza nel panorama scientifico odierno.

Nel Capitolo 2 riportiamo la notazione matematica adottata e presentiamo le nozioni fondamentali di algebra lineare a cui faremo riferimento nel corso dell'intera tesi.

Nel Capitolo 3 introduciamo la SVD, spiegandone l'utilità e le applicazioni principali, per poi definirla formalmente. Descriveremo anche le sue varianti, utili per ridurre il tempo di calcolo o lo spazio di memoria richiesto. Di fondamentale importanza sarà la scelta del *threshold*, la quale può essere effettuata secondo specifici criteri. Analizzeremo, in particolare, il criterio proposto da M. Gavish e D. L. Donoho [6].

Nel Capitolo 4 introduciamo l'algoritmo randomizzato, denominato RSVD, che rappresenta il nucleo centrale di questa tesi. Analizzeremo la sua complessità teorica e lo applicheremo a un test preliminare, seguito da un test iterativo sul numero di colonne di una matrice in input. In quest'ultimo confronto, valuteremo i tempi di esecuzione e gli errori relativi rispetto a quelli ottenuti con la SVD classica troncata. Il test sarà eseguito in due modalità: prima utilizzando il *threshold* proposto da M. Gavish e D. L. Donoho [6], poi con un *threshold* inferiore, di tipo logaritmico, per osservare i miglioramenti nei tempi di esecuzione e l'incremento, inevitabile, degli errori.

Nel Capitolo 5 sintetizziamo i risultati ottenuti e valutiamo la convenienza, in termini di tempi di esecuzione, dell'uso di RSVD rispetto all'algoritmo deterministico.

Infine, nell'Appendice A riportiamo gli script Matlab che sono stati usati per effettuare i test e costruire i grafici.

Indice

1	Introduzione	1
2	Conoscenze preliminari di algebra lineare	5
2.1	Notazione	5
2.2	Definizioni e teoremi	7
3	Singular Value Decomposition (SVD)	11
3.1	Motivazioni e applicazioni della SVD	11
3.2	SVD generale	12
3.3	SVD economica	14
3.4	SVD compatta	15
3.5	SVD troncata e gerarchia di approssimazioni	15
3.6	Scelta del <i>threshold</i> per SVD troncata	17
4	Randomized SVD	19
4.1	Motivazioni della randomizzazione	19
4.2	Un algoritmo randomizzato per il calcolo di SVD	19
4.3	Analisi di complessità teorica	21
4.4	Esecuzione della Randomized SVD su un'immagine	21
4.5	Test delle prestazioni e dell'accuratezza di RSVD	23
4.6	Test con <i>threshold</i> logaritmico	28
5	Conclusioni	33
A	Codice Matlab	35
A.1	Calcolo del <i>threshold</i> ottimale	35
A.2	Randomized SVD	35
A.3	Test singolo su un'immagine	36
A.4	Test iterativo sulle colonne di un'immagine	39
A.5	Formattazione dei grafici	46
Bibliografia		47

1

Introduzione

La Randomized Numerical Linear Algebra (RandNLA) è un campo emergente dell'algebra lineare numerica che utilizza algoritmi randomizzati per risolvere problemi di grandi dimensioni ed elevata complessità computazionale [3, 7]. Questa disciplina è diventata sempre più rilevante nel contesto della crescente esigenza di elaborare e analizzare enormi quantità di dati in tempo reale, specialmente in aree come il machine learning, la statistica e le scienze computazionali.

Tradicionalmente, molte delle operazioni fondamentali in algebra lineare, come la decomposizione di matrici, la risoluzione di sistemi lineari e l'approssimazione di valori singolari, sono state affrontate attraverso algoritmi deterministici che, sebbene precisi, spesso incontrano difficoltà nel gestire le enormi dimensioni delle matrici moderne. La RandNLA introduce un'alternativa significativa: sfruttare il potere della randomizzazione per semplificare e accelerare questi calcoli. Le metodologie randomizzate sono in grado di fornire approssimazioni accurate utilizzando minori risorse computazionali rispetto agli approcci tradizionali. Ad esempio, il campionamento casuale e la proiezione a bassa dimensione permettono di approssimare in modo preciso le strutture e proprietà di grandi matrici, riducendo significativamente i tempi di calcolo necessari per le operazioni richieste. Di seguito elenchiamo le tecniche di randomizzazione più usate sono nel contesto delle operazioni matriciali.

- **Campionamento elemento-per-elemento:** Un sottoinsieme di elementi della matrice viene selezionato in modo casuale e ridimensionato, ottenendo uno *sketch* che è una stima imparziale della matrice originale. Risultati più accurati si ottengono assegnando probabilità più alte agli elementi con valori assoluti maggiori.
- **Campionamento per righe o colonne:** Piuttosto che singoli elementi, si campionano intere righe o colonne, mantenendo una struttura lineare che è spesso più efficiente. Questo approccio permette di ottenere stime migliori rispetto al campionamento elemento-per-elemento.
- **Proiezioni randomizzate:** La matrice viene preprocessata con una matrice di proiezione casuale per distribuire uniformemente l'informazione, facilitando un campionamento uniforme successivo senza perdita di accuratezza.

La versatilità degli algoritmi randomizzati ha permesso la loro applicazione a una vasta gamma di problemi, inclusi quelli di compressione dei dati, il recupero di informazioni e la risoluzione di sistemi

di equazioni lineari. Inoltre, la capacità di gestire efficacemente grandi dataset ha reso RandNLA un elemento chiave nello sviluppo di tecniche avanzate in machine learning e intelligenza artificiale, dove le dimensioni dei dati e la necessità di calcoli rapidi sono particolarmente pressanti. Di seguito elenchiamo alcuni esempi concreti di applicazioni della RandNLA.

- **Minimi quadrati e approssimazione a basso rango:** Il campionamento casuale e la proiezione randomizzata permette di risolvere i problemi dei minimi quadrati (LS) e delle approssimazioni a basso rango in modo più rapido e con errori relativi accettabili.
- **Precondizionatori per algoritmi iterativi:** La RandNLA viene utilizzata per costruire precondizionatori che migliorano le prestazioni di algoritmi numerici classici, come Blendenpik e LSRN, su problemi LS di grandi dimensioni. Questi algoritmi ora competono o superano le soluzioni tradizionali come LAPACK.
- **Matrix Completion e decomposizione di colonne:** La RandNLA supporta il completamento di matrici a basso rango, utile per sistemi di raccomandazione, identificando e campionando righe e colonne che conservano informazioni chiave. Questo approccio è stato usato anche per decomporre matrici kernel in apprendimento automatico.
- **Risoluzione di sistemi lineari basati su laplaciane:** In contesti come il machine learning non supervisionato, le tecniche randomizzate sono utilizzate per creare versioni *sparse* delle matrici di Laplaciane, riducendo il numero di bordi e velocizzando la risoluzione dei sistemi lineari.
- **Apprendimento automatico e statistica:** RandNLA migliora l'efficienza e l'interpretabilità in analisi di grandi dati, includendo decomposizioni CX/CUR per identificare relazioni significative in vari campi scientifici, come la genetica e astronomia, oltre all'analisi di matrici kernel, fondamentale nel machine learning basato su kernel.

Concretamente, RandNLA agisce sintetizzando casualmente i dati in input per ridurre la dimensione del problema, preservando al contempo tutte le informazioni essenziali. Gli algoritmi per calcolare approssimazioni a basso rango delle matrici sono stati storicamente importanti nel calcolo scientifico, poiché permettono di semplificare e velocizzare operazioni complesse mantenendo un alto grado di accuratezza. Oggi questi algoritmi trovano un nuovo orizzonte di sviluppo in campi come l'apprendimento automatico e l'analisi dei dati, dove vengono utilizzati per ridurre la dimensionalità e migliorare l'efficienza dei modelli computazionali [4].

La storia dell'algebra lineare computazionale inizia nel 1947 con Von Neumann e Goldstine, che propongono di utilizzare i calcolatori per i calcoli matriciali. Negli anni seguenti, il progresso si concentra su linguaggi e librerie dedicate come Fortran (1957) e BLAS (1979), che standardizza operazioni base su vettori e matrici, seguita da LINPACK per la risoluzione di sistemi lineari complessi. Negli anni '80 e '90, BLAS si evolve per includere operazioni avanzate e LAPACK estende le sue funzionalità per il calcolo ad alto livello, supportando sistemi con matrici dense. ScaLAPACK nel 1996 espande LAPACK per l'elaborazione distribuita, mentre altre librerie come ATLAS (ottimizzazione automatica su hardware specifici), cuBLAS (per GPU NVIDIA) e GPTune (2022, per l'autotuning avanzato) migliorano ulteriormente le prestazioni e l'adattabilità. RandBLAS e RandLAPACK sono concetti proposti come possibili standard

futuri per le librerie di algebra lineare randomizzata. L’idea è di creare un’infrastruttura software simile a BLAS e LAPACK, ma ottimizzata per algoritmi che utilizzano tecniche di randomizzazione. Questi progetti mirerebbero a formalizzare e rendere più accessibile l’uso di tali algoritmi. Sebbene al momento non siano librerie ufficiali, la proposta di RandBLAS e RandLAPACK riflette l’interesse crescente verso soluzioni standard per l’implementazione efficiente di algoritmi randomizzati.

In questa tesi, applicheremo la Randomized Numerical Linear Algebra alla Singular Value Decomposition (SVD) ed eseguiremo una valutazione comparativa dei tempi d’esecuzione dell’algoritmo classico e del suo equivalente randomizzato (RSVD), al variare del numero di colonne della matrice in input. Ci concentreremo sui miglioramenti computazionali ottenuti tramite RSVD e su come l’efficienza di quest’ultimo possa essere modulata attraverso la scelta del troncamento sulle colonne. In particolare, analizzeremo come l’adozione di *threshold* più piccoli comporti un aumento dell’efficienza, evidenziando però l’effetto collaterale di una maggiore approssimazione dell’errore, già influenzato dalla natura randomizzata dell’algoritmo. I risultati dimostrano che RSVD risulta significativamente più rapido rispetto alla SVD classica, evidenziando le condizioni ottimali di applicazione e bilanciamento tra precisione e prestazioni per trarre il massimo vantaggio dall’approccio randomizzato in contesti pratici. L’algoritmo randomizzato che analizzeremo è stato proposto da Steven L. Brunton e J. Nathan Kutz [1] ed è basato sulla riduzione dimensionale attraverso una matrice di proiezione randomizzata. Dimostreremo che la sua complessità in termini di tempo d’esecuzione è $\mathcal{O}(mnt)$, dove t è il *threshold* di troncamento, migliore rispetto alle classiche implementazioni di SVD che hanno complessità $\mathcal{O}(\max(m, n)(\min(m, n))^2)$.

2

Conoscenze preliminari di algebra lineare

In questo capitolo riportiamo la notazione, le definizioni e i teoremi di algebra lineare di cui faremo uso nel corso della tesi. Si assume nota la conoscenza delle definizioni fondamentali di algebra lineare, come quella di spazio vettoriale, di vettori linearmente indipendenti o dipendenti, di spazio generato da un insieme di vettori, di base di uno spazio vettoriale, e così via. Si danno inoltre per acquisiti i principali teoremi associati, come il teorema di esistenza di una base.

2.1 Notazione

Scalari

$\tilde{e}_t \in \mathbb{R}$	Errore relativo di $\tilde{\mathbf{X}}_t$ rispetto a \mathbf{X} .
$\hat{e}_t \in \mathbb{R}$	Errore relativo di $\hat{\mathbf{X}}_t$ rispetto a \mathbf{X} .
$k \in \mathbb{N}$	Minimo tra il numero di righe e colonne di una matrice.
$q_{ij} \in \mathbb{R}$	ij -esimo elemento di \mathbf{Q} , con $1 \leq i \leq m$ e $1 \leq j \leq m$.
$r \in \mathbb{N}$	Rango di una matrice.
$t \in \mathbb{N}$	Rango di troncamento (<i>threshold</i>) per SVD troncata o RSVD.
$u_i \in \mathbb{R}$	i -esimo elemento di \mathbf{u} , con $1 \leq i \leq n$.
$v_i \in \mathbb{R}$	i -esimo elemento di \mathbf{v} , con $1 \leq i \leq n$.
$x_{ij} \in \mathbb{R}$	ij -esimo elemento di \mathbf{X} , con $1 \leq i \leq m$ e $1 \leq j \leq n$.
$\beta \in \mathbb{R}$	Rapporto tra righe e colonne (o colonne e righe) di \mathbf{X} .
$\gamma \in \mathbb{R}$	Magnitudine nota del rumore bianco.
$\lambda \in \mathbb{C}$	Generico autovalore di una matrice quadrata.
$\lambda_i \in \mathbb{R}$	i -esimo autovalore di \mathbf{S}_u , con $1 \leq i \leq m$.
$\lambda(\beta)$	Coefficiente moltiplicativo del <i>threshold</i> ottimale.
$\mu_j \in \mathbb{R}$	j -esimo autovalore di \mathbf{S}_v , con $1 \leq j \leq n$.
$\mu_\beta \in \mathbb{R}$	Valore mediano della distribuzione di Marčenko-Pastur.
$\pi \in \mathbb{R}$	Percentuale prestabilita della varianza (o energia) nei dati originali.
$\sigma_i \in \mathbb{R}$	i -esimo valore singolare di \mathbf{X} , con $1 \leq i \leq \min(m, n)$.
$\sigma_{\text{median}} \in \mathbb{R}$	Valore singolare mediano di \mathbf{X} .
$\tau \in \mathbb{R}$	Legge continua del <i>threshold</i> ottimale.

Vettori

$\mathbf{u} \in \mathbb{R}^n$	Vettore generico.
$\mathbf{u}_i \in \mathbb{R}^m$	i -esimo vettore singolare sinistro di \mathbf{X} , con $1 \leq i \leq m$.
$\mathbf{v} \in \mathbb{R}^n$	Vettore generico.
$\mathbf{v}_j \in \mathbb{R}^n$	j -esimo vettore singolare destro di \mathbf{X} , con $1 \leq j \leq n$.
$\mathbf{x}_i \in \mathbb{R}^n$	i -esimo vettore riga di \mathbf{X} , con $1 \leq i \leq m$.
$\mathbf{y}_j \in \mathbb{R}^m$	i -esimo vettore colonna di \mathbf{X} , con $1 \leq j \leq n$.

Matrici

$\mathbf{0} \in \mathbb{R}^{m \times n}$	Matrice nulla generica.
$\mathbf{0}_Y \in \mathbb{R}^{t \times (n-t)}$	Sottomatrice nulla di $\boldsymbol{\Sigma}_Y$.
$\mathbf{I} \in \mathbb{R}^{m \times m}$	Matrice identità.
$\mathbf{P} \in \mathbb{R}^{n \times t}$	Matrice di proiezioni casuali usata nella RSVD.
$\mathbf{Q} \in \mathbb{R}^{m \times m}$	Matrice quadrata generica.
$\mathbf{Q} \in \mathbb{R}^{m \times t}$	Matrice con colonne ortonormali nella decomposizione QR.
$\mathbf{R} \in \mathbb{R}^{t \times t}$	Matrice triangolare superiore nella decomposizione QR.
$\mathbf{S} \in \mathbb{R}^{m \times m}$	Matrice simmetrica generica.
$\mathbf{S}_u \in \mathbb{R}^{m \times m}$	Matrice simmetrica sinistra associata a \mathbf{X} .
$\mathbf{S}_v \in \mathbb{R}^{m \times m}$	Matrice simmetrica destra associata a \mathbf{X} .
$\mathbf{U} \in \mathbb{R}^{m \times m}$	Matrice dei vettori singolari sinistri di \mathbf{X} .
$\mathbf{U}_\perp \in \mathbb{R}^{m \times (m-n)}$	Sottomatrice di \mathbf{U} contenente gli ultimi $m - n$ vettori singolari sinistri di \mathbf{X} .
$\mathbf{U}_k \in \mathbb{R}^{m \times k}$	Sottomatrice di \mathbf{U} contenente i primi k vettori singolari sinistri di \mathbf{X} .
$\mathbf{U}_n \in \mathbb{R}^{m \times n}$	Sottomatrice di \mathbf{U} contenente i primi n vettori singolari sinistri di \mathbf{X} .
$\mathbf{U}_r \in \mathbb{R}^{m \times r}$	Sottomatrice di \mathbf{U} contenente i primi r vettori singolari sinistri di \mathbf{X} .
$\mathbf{U}_t \in \mathbb{R}^{m \times t}$	Sottomatrice di \mathbf{U} contenente i primi t vettori singolari sinistri di \mathbf{X} .
$\mathbf{U}_Y \in \mathbb{R}^{t \times t}$	Matrice dei vettori singolari sinistri di \mathbf{Y} .
$\hat{\mathbf{U}}_t \in \mathbb{R}^{m \times t}$	Matrice risultante dalla proiezione $\mathbf{Q}\mathbf{U}_Y$.
$\mathbf{V} \in \mathbb{R}^{n \times n}$	Matrice dei vettori singolari destri di \mathbf{X} .
$\mathbf{V}_\perp \in \mathbb{R}^{n \times (n-m)}$	Sottomatrice di \mathbf{V} contenente gli ultimi $n - m$ vettori singolari destri di \mathbf{X} .
$\mathbf{V}_k \in \mathbb{R}^{n \times k}$	Sottomatrice di \mathbf{V} contenente i primi k vettori singolari destri di \mathbf{X} .
$\mathbf{V}_m \in \mathbb{R}^{n \times m}$	Sottomatrice di \mathbf{V} contenente i primi m vettori singolari destri di \mathbf{X} .
$\mathbf{V}_r \in \mathbb{R}^{n \times r}$	Sottomatrice di \mathbf{V} contenente i primi r vettori singolari destri di \mathbf{X} .
$\mathbf{V}_t \in \mathbb{R}^{n \times t}$	Sottomatrice di \mathbf{V} contenente i primi t vettori singolari destri di \mathbf{X} .
$\mathbf{X} \in \mathbb{Q}^{m \times n}$	Matrice razionale usata come input nei test di questa tesi.
$\mathbf{X} \in \mathbb{R}^{m \times n}$	Matrice generica.
$\mathbf{X}_{\text{noise}} \in \mathbb{R}^{m \times n}$	Rumore bianco di \mathbf{X} .
$\mathbf{X}_t \in \mathbb{R}^{m \times n}$	Generica approssimazione di \mathbf{X} al rango t .
$\mathbf{X}_{\text{true}} \in \mathbb{R}^{m \times n}$	Dati puri di \mathbf{X} .
$\hat{\mathbf{X}}_t \in \mathbb{R}^{m \times n}$	Approssimazione di \mathbf{X} al rango t calcolata con RSVD.
$\tilde{\mathbf{X}}_t \in \mathbb{R}^{m \times n}$	Migliore approssimazione di \mathbf{X} al rango t .

$\mathbf{Y} \in \mathbb{R}^{n \times m}$	Matrice generica.
$\mathbf{Y} \in \mathbb{R}^{t \times n}$	Proiezione di \mathbf{X} tramite \mathbf{Q} della decomposizione QR.
$\mathbf{Z} \in \mathbb{R}^{m \times t}$	Matrice risultante dalla proiezione di \mathbf{X} tramite \mathbf{P} .
$\Sigma \in \mathbb{R}^{m \times n}$	Matrice dei valori singolari di \mathbf{X} .
$\Sigma_k \in \mathbb{R}^{k \times k}$	Sottomatrice quadrata di Σ contenente i valori singolari di \mathbf{X} .
$\Sigma_m \in \mathbb{R}^{m \times m}$	Sottomatrice quadrata di Σ contenente i valori singolari di \mathbf{X} .
$\Sigma_n \in \mathbb{R}^{n \times n}$	Sottomatrice quadrata di Σ contenente i valori singolari di \mathbf{X} .
$\Sigma_r \in \mathbb{R}^{r \times r}$	Sottomatrice quadrata di Σ contenente i valori singolari non nulli di \mathbf{X} .
$\Sigma_t \in \mathbb{R}^{t \times t}$	Sottomatrice quadrata di Σ contenente i primi t valori singolari non nulli di \mathbf{X} .
$\Sigma_Y \in \mathbb{R}^{t \times n}$	Matrice dei valori singolari di \mathbf{Y} .
$\Sigma_{Y,t} \in \mathbb{R}^{t \times t}$	Sottomatrice quadrata di Σ_Y contenente i valori singolari di \mathbf{Y} .

Operatori

\cdot	Prodotto scalare tra due vettori.
\cdot^{-1}	Inversione di una matrice.
\cdot^T	Trasposizione di una matrice.
$\lceil \cdot \rceil$	Approssimazione di uno scalare per eccesso.
$\lfloor \cdot \rfloor$	Approssimazione di uno scalare per difetto.
$ \cdot $	Numero di elementi di un vettore o di una matrice.
$ \cdot $	Valore assoluto di uno scalare.
$\ \cdot\ _2$	Norma euclidea di un vettore.
$\ \cdot\ _F$	Norma di Frobenius di una matrice.
$\langle \cdot \rangle$	Spazio generato da un insieme di vettori.
$\underset{\star}{\operatorname{argmin}}(\cdot)$	Elemento di \star che minimizza l'espressione \cdot .
$\log(\cdot)$	Logaritmo naturale di uno scalare.
$\min(\cdot, \cdot)$	Minimo tra due scalari.
$\max(\cdot, \cdot)$	Massimo tra due scalari.
$\mathcal{O}(\cdot)$	O-grande di una funzione.
$\operatorname{rank}(\cdot)$	Operatore per ottenere il rango di una matrice.
$\operatorname{tr}(\cdot)$	Traccia di una matrice quadrata.

2.2 Definizioni e teoremi

Le seguenti nozioni riguardano vettori e matrici nel campo dei numeri reali \mathbb{R} . Esse sono valide anche nel campo dei numeri razionali \mathbb{Q} , che verrà utilizzato nella parte sperimentale di questa tesi. Le definizioni e le proprietà relative alla Singular Value Decomposition (SVD) verranno trattate nel Capitolo 3.

Definizione 2.1 (Prodotto scalare) Il prodotto scalare tra due vettori $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ è uno scalare in \mathbb{R} dato dalla somma dei prodotti delle coordinate di \mathbf{u} e \mathbf{v} :

$$\mathbf{u} \cdot \mathbf{v} = \sum_{i=1}^n u_i v_i = u_1 v_1 + u_2 v_2 + \cdots + u_n v_n$$

Definizione 2.2 (Vettori ortogonali) Due vettori $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ sono ortogonali se il loro prodotto scalare è nullo: $\mathbf{u} \cdot \mathbf{v} = 0$.

Definizione 2.3 (Norma euclidea o norma L^2) La norma euclidea (o norma L^2) di un vettore $\mathbf{v} \in \mathbb{R}^n$ è data da:

$$\|\mathbf{v}\|_2 = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}$$

Definizione 2.4 (Vettore normale) Un vettore $\mathbf{v} \in \mathbb{R}^n$ è detto normale se la sua norma, secondo una determinata definizione di norma, è 1.

Definizione 2.5 (Vettori ortonormali) Due vettori $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ sono ortonormali se sono ortogonali e se sono entrambi normali.

Definizione 2.6 (Spazio generato dalle righe) Siano $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m \in \mathbb{R}^n$ i vettori associati alle righe della matrice $\mathbf{X} \in \mathbb{R}^{m \times n}$. Definiamo lo spazio generato dalle righe di \mathbf{X} come $\langle \mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m \rangle$.

Definizione 2.7 (Spazio generato dalle colonne) Siano $\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_n \in \mathbb{R}^m$ i vettori associati alle colonne della matrice $\mathbf{X} \in \mathbb{R}^{m \times n}$. Definiamo lo spazio generato dalle colonne di \mathbf{X} come $\langle \mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_n \rangle$.

Teorema 2.1 Siano $\langle \mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m \rangle$ e $\langle \mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_n \rangle$ gli spazi generati dalle righe e dalle colonne di $\mathbf{X} \in \mathbb{R}^{m,n}$. Si ha

$$\dim(\langle \mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m \rangle) = \dim(\langle \mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_n \rangle)$$

Definizione 2.8 (Matrice triangolare superiore e inferiore) Una matrice quadrata $\mathbf{Q} \in \mathbb{R}^{m \times m}$ è triangolare superiore se $q_{ij} = 0$ per $m \geq i > j \geq 1$; è triangolare inferiore se $q_{ij} = 0$ per $m \geq j > i \geq 1$.

Definizione 2.9 (Rango di una matrice) Definiamo il rango $r \leq \min(m, n)$ di una matrice $\mathbf{X} \in \mathbb{R}^{m \times n}$, indicato anche con $\text{rank}(\mathbf{X})$, come il massimo numero di colonne (o righe) linearmente indipendenti. Equivalentemente, r è la dimensione dello spazio generato dalle righe o dalle colonne (per il precedente teorema, esse sono uguali).

Definizione 2.10 (Matrice a rango completo) Una matrice $\mathbf{X} \in \mathbb{R}^{m \times n}$ è di rango completo se $r = \min(m, n)$.

Se non specificato, le matrici descritte nel corso di questa tesi saranno ritenute, senza perdita di generalità, a rango completo. Infatti, una matrice di grandi dimensioni ha una probabilità quasi nulla di avere colonne (o righe) linearmente dipendenti. Tale probabilità è esattamente nulla se gli elementi della matrice sono generati da una distribuzione di probabilità continua. Inoltre, in quest'ultimo caso, le colonne (o le righe) della matrice sono vettori pressoché ortogonali.

Definizione 2.11 (Matrice trasposta) La trasposta $\mathbf{X}^T \in \mathbb{R}^{n \times m}$ di una matrice $\mathbf{X} \in \mathbb{R}^{m \times n}$ è la stessa matrice con righe e colonne scambiate.

Definizione 2.12 (Matrice diagonale) Una matrice quadrata $\mathbf{Q} \in \mathbb{R}^{m \times m}$ è diagonale se $q_{ij} = 0$ per $1 \leq i \neq j \leq m$. Tale definizione si può estendere anche a matrici rettangolari $\mathbf{X} \in \mathbb{R}^{m \times n}$, ponendo $x_{ij} = 0$ per $i \neq j$, con $1 \leq i \leq m$ e $1 \leq j \leq n$.

Definizione 2.13 (Traccia) Data una matrice quadrata $\mathbf{Q} \in \mathbb{R}^{m \times m}$, definiamo la sua traccia come la somma degli elementi sulla diagonale:

$$\text{tr}(\mathbf{Q}) = \sum_{i=1}^m x_{ii}$$

Proprietà 2.1 (Proprietà della traccia) Date le matrici $\mathbf{X} \in \mathbb{R}^{m \times n}$ e $\mathbf{Y} \in \mathbb{R}^{n \times m}$, si possono facilmente dimostrare le seguenti proprietà sulla traccia:

- $\text{tr}(\mathbf{X}\mathbf{X}^T) = \sum_{i=1}^m x_{ii}^2$;
- $\text{tr}(\mathbf{XY}) = \text{tr}(\mathbf{YX})$.

Definizione 2.14 (Matrice identità) La matrice identità $\mathbf{I} \in \mathbb{R}^{m \times m}$ è una matrice che ha tutti gli elementi sulla diagonale principale uguali a 1 e tutti gli altri uguali a 0.

Definizione 2.15 (Autowettore e autovalore di una matrice quadrata) Un autovettore di una matrice quadrata $\mathbf{Q} \in \mathbb{R}^{m \times m}$ è un vettore $\mathbf{v} \in \mathbb{R}^m$ che, quando subisce la trasformazione lineare \mathbf{Q} , produce come risultato un multiplo scalare del vettore originale, ossia $\mathbf{Qv} = \lambda\mathbf{v}$. Lo scalare $\lambda \in \mathbb{C}$ è detto autovalore.

Gli autovettori $\mathbf{v} \in \mathbb{R}^m$ di una matrice quadrata $\mathbf{Q} \in \mathbb{R}^{m \times m}$ si possono trovare risolvendo l'equazione caratteristica $(\mathbf{Q} - \lambda\mathbf{I})\mathbf{v} = \mathbf{0}$, dove gli autovalori si trovano ponendo $\det(\mathbf{Q} - \lambda\mathbf{I}) = 0$.

Teorema 2.2 Una matrice simmetrica $\mathbf{S} \in \mathbb{R}^{m \times m}$, ha autovalori reali e autovettori ortogonali, che possono essere normalizzati.

Definizione 2.16 (Matrice semidefinita positiva) Una matrice simmetrica $\mathbf{S} \in \mathbb{R}^{m \times m}$ è semidefinita positiva se, per ogni vettore $\mathbf{v} \in \mathbb{R}^m$, il prodotto quadratico è non negativo: $\mathbf{v}^T \mathbf{S} \mathbf{v} \geq 0$. Ciò è equivalente a dire che i suoi autovalori sono non negativi.

Definizione 2.17 (Matrice ortogonale) Una matrice quadrata $\mathbf{Q} \in \mathbb{R}^{m \times m}$ è ortogonale se le sue colonne o, equivalentemente, le sue righe sono vettori ortonormali. Ciò implica che

- $\mathbf{QQ}^T = \mathbf{Q}^T \mathbf{Q} = \mathbf{I}$.
- \mathbf{Q} è sempre invertibile, con inversa uguale alla sua trasposta: $\mathbf{Q}^{-1} = \mathbf{Q}^T$;

Definizione 2.18 (Decomposizione QR) Una matrice $\mathbf{X} \in \mathbb{R}^{m \times n}$, con $m \geq n$, può essere fattorizzata nel seguente modo:

$$\mathbf{X} = \mathbf{QR}$$

dove $\mathbf{Q} \in \mathbb{R}^{m \times n}$ ha colonne ortonormali ($\mathbf{Q}^T \mathbf{Q} = \mathbf{I}$) e $\mathbf{R} \in \mathbb{R}^{n \times n}$ è triangolare superiore.

Di seguito riportiamo la definizione di norma di Frobenius, di cui faremo ampio uso per confrontare una matrice \mathbf{X} con sue approssimazioni. Essa può essere vista come un'estensione della norma euclidea dei vettori, applicata alle matrici.

Definizione 2.19 (Norma di Frobenius) *La norma di Frobenius di una matrice $\mathbf{X} \in \mathbb{R}^{m \times n}$ è definita come:*

$$\|\mathbf{X}\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n |x_{ij}|^2},$$

ossia la radice quadrata della somma dei quadrati dei valori assoluti degli elementi di \mathbf{X} .

Proprietà 2.2 (Relazione tra traccia e norma di Frobenius) *Si può facilmente dimostrare che il quadrato della norma di Frobenius di una matrice $\mathbf{X} \in \mathbb{R}^{m \times n}$ è uguale alla traccia di $\mathbf{X}\mathbf{X}^T \in \mathbb{R}^{m \times m}$:*

$$\|\mathbf{X}\|_F^2 = \text{tr}(\mathbf{X}\mathbf{X}^T)$$

3

Singular Value Decomposition (SVD)

In questo capitolo trattiamo la SVD, una delle più importanti decomposizioni di matrici. Dopo averne illustrato le applicazioni e l'utilità, la definiremo in modo formale. Successivamente, esamineremo le varianti ridotte della SVD, utili quando la matrice originale soddisfa determinati criteri. Infine, per la SVD troncata, analizzeremo la scelta del *threshold*, con un focus specifico sul criterio proposto da M. Gavish e D. L. Donoho [6].

3.1 Motivazioni e applicazioni della SVD

La decomposizione ai valori singolari (SVD) è una delle tecniche fondamentali e più potenti dell'algebra lineare, con un impatto trasversale in numerosi campi scientifici e tecnologici. Si tratta di una metodologia che consente di scomporre una matrice generica in tre componenti distinte: una matrice ortogonale delle direzioni di variabilità nello spazio delle righe, una matrice diagonale che ne cattura l'importanza relativa attraverso i valori singolari e una matrice ortogonale che rappresenta le direzioni di variabilità nello spazio delle colonne. Questa decomposizione fornisce una visione approfondita delle proprietà strutturali della matrice e facilita l'analisi e la manipolazione di dati complessi.

Uno degli usi più comuni della SVD è nella riduzione della dimensionalità, un processo cruciale quando si lavora con grandi quantità di dati che contengono ridondanze o rumore. Nel campo dell'elaborazione delle immagini, ad esempio, la SVD è utilizzata per comprimere immagini riducendo la quantità di informazioni mantenendo solo le componenti principali: in pratica, si rimuovono i dettagli meno rilevanti, preservando però le caratteristiche visive più importanti. Questo processo non solo facilita l'archiviazione e la trasmissione delle immagini, ma consente anche di ridurre significativamente i tempi di calcolo senza compromettere troppo la qualità visiva.

Oltre alla riduzione dimensionale, la SVD trova un'applicazione fondamentale nel filtraggio collaborativo, un metodo utilizzato in piattaforme di raccomandazione come Netflix o Spotify. Qui, la SVD permette di analizzare le preferenze degli utenti e correlare gusti simili tra di loro, suggerendo film, serie TV o brani musicali che potrebbero risultare di interesse. In questi contesti, la decomposizione consente di ridurre la complessità del problema, estrapolando solo le informazioni rilevanti, che descrivono al meglio le relazioni latenti tra utenti e prodotti.

Le motivazioni per l'uso della SVD risiedono nella sua capacità di semplificare strutture complesse e di evidenziare le componenti principali di una matrice. Questa tecnica riesce a trasformare problemi di elevata dimensionalità e complessità in rappresentazioni più accessibili e interpretabili, che permettono di ridurre il rumore nei dati e di mettere in evidenza tendenze e pattern nascosti. Ad esempio, nell'analisi dei dati, la SVD può essere utilizzata per estrarre le principali direzioni di variazione in un dataset, facilitando l'interpretazione delle correlazioni tra variabili o l'identificazione di cluster di dati simili.

In un mondo sempre più orientato ai dati, la SVD si dimostra uno strumento essenziale per l'estrazione di informazioni rilevanti da insiemi di dati vasti e complessi. La sua capacità di trovare rappresentazioni compatte e significative rende possibile non solo una gestione più efficiente dei dati, ma anche una maggiore comprensione delle strutture sottostanti, che altrimenti potrebbero rimanere nascoste dietro la complessità e la vastità delle informazioni disponibili. Sia che si tratti di big data, machine learning, bioinformatica, fisica computazionale o elaborazione del linguaggio naturale, la SVD rappresenta un ponte tra complessità matematica e utilità pratica, permettendo di derivare risultati utili e pratici a partire da dati eterogenei e non strutturati.

3.2 SVD generale

Consideriamo una matrice $\mathbf{X} \in \mathbb{R}^{m \times n}$ e la sua trasposta $\mathbf{X}^T \in \mathbb{R}^{n \times m}$. Eseguiamo le seguenti moltiplicazioni matriciali:

- $\mathbf{S}_u = \mathbf{X}\mathbf{X}^T \in \mathbb{R}^{m \times m}$;
- $\mathbf{S}_v = \mathbf{X}^T\mathbf{X} \in \mathbb{R}^{n \times n}$.

Moltiplicando una matrice per la sua trasposta si ottiene come risultato una matrice quadrata simmetrica. Una tale matrice possiede autovalori reali e autovettori ortonormali. In questo caso, i prodotti \mathbf{S}_u e \mathbf{S}_v sono matrici quadrate simmetriche: \mathbf{S}_u viene detta matrice simmetrica sinistra e \mathbf{S}_v matrice simmetrica destra (in base alla posizione di \mathbf{X} nel prodotto). Gli autovettori $\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_m$ di \mathbf{S}_u sono chiamati vettori singolari sinistri di \mathbf{X} , mentre gli autovettori $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$ di \mathbf{S}_v sono detti vettori singolari destri di \mathbf{X} . Gli autovalori $\lambda_1, \lambda_2, \dots, \lambda_m$ di \mathbf{S}_u e $\mu_1, \mu_2, \dots, \mu_m$ di \mathbf{S}_v godono delle seguenti proprietà:

- $(\lambda_1 = \mu_1) \geq (\lambda_2 = \mu_2) \geq \dots \geq (\lambda_{\min(m,n)} = \mu_{\min(m,n)}) \geq 0$;
- Se $m \geq n$, allora $\lambda_i = 0$ per $n < i \leq m$; altrimenti, se $n > m$, $\mu_i = 0$ per $m < i \leq n$.

Quindi, le matrici \mathbf{S}_u e \mathbf{S}_v sono simmetriche e semidefinite positive. Le radici quadrate degli autovalori sono i valori singolari della matrice \mathbf{X} e li indicheremo con $\sigma_1 = \sqrt{\lambda_1} = \sqrt{\mu_1}$, $\sigma_2 = \sqrt{\lambda_2} = \sqrt{\mu_2}$, ..., $\sigma_r = \sqrt{\lambda_{\min(m,n)}} = \sqrt{\mu_{\min(m,n)}}$. Essi forniscono un'importante misura del grado di dipendenza lineare delle colonne di una matrice. Questo rende i valori singolari uno strumento cruciale nell'analisi dei dati e nella riduzione della dimensionalità. Si può dimostrare che \mathbf{X} ha rango $r \leq \min(m, n)$ se e solo se Σ ha r valori non nulli sulla diagonale:

- $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r \geq 0$;
- $\sigma_{r+1} = \sigma_{r+2} = \dots = \sigma_{\min(m,n)} = 0$.

Definizione 3.1 (Singular Value Decomposition) La Singular Value Decomposition (SVD) fattorizza la matrice $\mathbf{X} \in \mathbb{R}^{m \times n}$ nel seguente modo:

$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$$

dove

- $\mathbf{U} \in \mathbb{R}^{m \times m}$ è una matrice quadrata ortogonale le cui colonne sono i vettori singolari sinistri $\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_m$ di \mathbf{X} .
- $\Sigma \in \mathbb{R}^{m \times n}$ è una matrice diagonale rettangolare, i cui elementi sulla diagonale sono i valori singolari $\sigma_1, \sigma_2, \dots, \sigma_{\min(m,n)}$ di \mathbf{X} , disposti in ordine decrescente;
- $\mathbf{V} \in \mathbb{R}^{n \times n}$ è una matrice quadrata ortogonale le cui colonne sono i vettori singolari destri $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$ di \mathbf{X} . Nella decomposizione si considera la sua trasposta \mathbf{V}^T .

Possiamo dunque esprimere la matrice \mathbf{X} come somma di matrici a rango 1:

$$\mathbf{X} = \sum_{i=1}^r \sigma_i \mathbf{u}_i \mathbf{v}_i^T$$

Osserviamo che è più conveniente memorizzare i singoli vettori colonna (o riga) $\mathbf{v}_i, \mathbf{u}_i$ anziché l'intera matrice \mathbf{X} . Infatti, ogni addendo $\mathbf{v}_i \mathbf{u}_i$ è formato da $m + n$ elementi (m per \mathbf{u}_i , n per \mathbf{v}_i), da cui otteniamo che il numero di elementi necessari è $r(m + n)$.

Con le proprietà 2.1 e 2.2 otteniamo

$$\|\Sigma\|_F^2 = \text{tr}(\Sigma \Sigma^T) = \sum_{i=1}^r \sigma_i^2$$

ossia, il prodotto $\Sigma \Sigma^T \in \mathbb{R}^{m \times m}$ è una matrice diagonale con i quadrati dei valori singolari sulla diagonale. Calcoliamo ora il quadrato della norma di Frobenius di \mathbf{X} , sfruttando le proprietà 2.1 e 2.2:

$$\begin{aligned} \|\mathbf{X}\|_F^2 &= \text{tr}(\mathbf{X} \mathbf{X}^T) \\ &= \text{tr}((\mathbf{U} \Sigma \mathbf{V}^T)(\mathbf{U} \Sigma \mathbf{V}^T)^T) \\ &= \text{tr}(\mathbf{U} \Sigma \mathbf{V}^T \mathbf{V} \Sigma^T \mathbf{U}^T) \\ &= \text{tr}(\mathbf{U} \Sigma \Sigma^T \mathbf{U}^T) \\ &= \text{tr}(\Sigma^T \mathbf{U}^T \mathbf{U} \Sigma) \\ &= \text{tr}(\Sigma \Sigma^T) \\ &= \|\Sigma\|_F^2 \end{aligned}$$

Abbiamo così dimostrato la seguente proprietà che lega la norma di Frobenius ai valori singolari

Proprietà 3.1

$$\|\mathbf{X}\|_F^2 = \sum_{i=1}^r \sigma_i^2$$

3.3 SVD economica

Quando le dimensioni della matrice $\mathbf{X} \in \mathbb{R}^{m \times n}$ sono molto grandi, è utile considerare una versione ridotta o *economica* della decomposizione, che preserva le informazioni essenziali riducendo il numero di operazioni e lo spazio di memoria necessari.

Se $m \geq n$, ossia il numero di righe supera quello delle colonne, allora si può suddividere la matrice dei valori singolari $\Sigma \in \mathbb{R}^{m \times n}$ nella seguente forma:

$$\begin{bmatrix} \Sigma_n \\ \mathbf{0} \end{bmatrix}$$

dove

- $\Sigma_n \in \mathbb{R}^{n \times n}$ contiene i valori singolari di \mathbf{X} sulla diagonale;
- $\mathbf{0} \in \mathbb{R}^{(m-n) \times n}$ è la sottomatrice nulla.

Si può inoltre suddividere la matrice $\mathbf{U} \in \mathbb{R}^{m \times m}$ in due sottomatrici che identificano due spazi tra loro ortogonali e complementari:

$$\begin{bmatrix} \mathbf{U}_n & \mathbf{U}_\perp \end{bmatrix}$$

dove

- $\mathbf{U}_n \in \mathbb{R}^{m \times n}$ contiene i primi n vettori singolari sinistri;
- $\mathbf{U}_\perp \in \mathbb{R}^{m \times (m-n)}$ contiene i restanti $m - n$ vettori singolari sinistri.

È facile verificare la seguente uguaglianza:

$$\mathbf{X} = \mathbf{U} \Sigma \mathbf{V}^T = \begin{bmatrix} \mathbf{U}_n & \mathbf{U}_\perp \end{bmatrix} \begin{bmatrix} \Sigma_n \\ \mathbf{0} \end{bmatrix} \mathbf{V}^T = \mathbf{U}_n \Sigma_n \mathbf{V}^T$$

da cui è evidente che \mathbf{U}_\perp non viene considerata nel calcolo, siccome si annulla con $\mathbf{0}$.

Equivalentemente, è possibile riformulare SVD nella sua versione economica se il numero di colonne supera quello di righe, $n \geq m$:

$$\mathbf{X} = \mathbf{U} \Sigma \mathbf{V}^T = \mathbf{U} \begin{bmatrix} \Sigma_m & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{V}_m^T \\ \mathbf{V}_\perp^T \end{bmatrix} = \mathbf{U} \Sigma_m \mathbf{V}_m^T$$

dove

- $\Sigma_m \in \mathbb{R}^{m \times m}$ contiene i valori singolari di \mathbf{X} sulla diagonale;
- $\mathbf{0} \in \mathbb{R}^{m \times (n-m)}$ è la sottomatrice nulla;
- $\mathbf{V}_m^T \in \mathbb{R}^{m \times n}$ contiene i primi m vettori singolari destri;
- $\mathbf{V}_\perp^T \in \mathbb{R}^{(n-m) \times n}$ contiene i restanti $n - m$ vettori singolari sinistri.

dove, questa volta, la parte inutile è \mathbf{V}_\perp^T , che viene eliminata dal calcolo.

Entrambi i casi possono essere unificati in una formulazione generale della SVD economica:

$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T = \mathbf{U}_k\Sigma_k\mathbf{V}_k^T$$

dove

- $k = \min(m, n)$;
- $\mathbf{U}_k \in \mathbb{R}^{m \times k}$ contiene le prime k colonne di \mathbf{U} , ossia i primi k vettori singolari sinistri;
- $\Sigma_k \in \mathbb{R}^{k \times k}$ contiene i primi k valori singolari di \mathbf{X} ;
- $\mathbf{V}_k^T \in \mathbb{R}^{k \times n}$ contiene le prime k righe di \mathbf{V}^T , ossia i primi k vettori singolari destri.

La SVD *economica* è particolarmente utile quando $k \ll \max(m, n)$. In questa tesi, ogni calcolo della SVD sarà eseguito con tale versione ridotta.

3.4 SVD compatta

La versione economica di SVD può essere ulteriormente migliorata se la matrice Σ contiene dei valori singolari nulli. Possiamo infatti riscrivere la SVD nel seguente modo

$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T = \mathbf{U}_r\Sigma_r\mathbf{V}_r^T$$

dove

- r è il numero di valori singolari non nulli presenti in Σ ;
- $\mathbf{U}_r \in \mathbb{R}^{m \times r}$ contiene r vettori (colonna) singolari sinistri di \mathbf{X} ;
- $\Sigma_r \in \mathbb{R}^{r \times r}$ contiene, sulla diagonale, gli r valori singolari non nulli di \mathbf{X} ;
- $\mathbf{V}_r^T \in \mathbb{R}^{r \times n}$ contiene r vettori (riga) singolari destri di \mathbf{X} .

3.5 SVD troncata e gerarchia di approssimazioni

In molti casi si cerca addirittura di eliminare dal calcolo di SVD alcuni valori singolari non nulli di \mathbf{X} . In tal caso, SVD non rappresenta più una fattorizzazione esatta della matrice \mathbf{X} originale, ma fornisce una sua approssimazione $\tilde{\mathbf{X}}_t$ di rango inferiore $t \leq r$, detto *threshold*, uguale al numero di valori singolari mantenuti. Ma non è tutto qui: la SVD troncata a un determinato rango t fornisce la migliore approssimazione $\tilde{\mathbf{X}}_t$ della matrice originale \mathbf{X} al rango t in termini di distanza quadratica¹. Ciò viene formalizzato nel seguente teorema:

¹In altre parole, non è possibile trovare un'altra matrice di rango t che si discosti da \mathbf{X} meno di quanto faccia $\tilde{\mathbf{X}}_t$ calcolata con la SVD troncata al rango t . La misura del *discostamento* è data dalla norma di Frobenius.

Teorema 3.1 (Eckard-Young) La miglior approssimazione $\tilde{\mathbf{X}}_t$ al rango t di una matrice $\mathbf{X} \in \mathbb{R}^{m \times n}$ si ottiene con la SVD troncata (TSVD) al rango t :

$$\tilde{\mathbf{X}}_t = \underset{\mathbf{X}_t \text{ t.c. } \text{rank}(\mathbf{X}_t)=t}{\operatorname{argmin}} (\|\mathbf{X} - \mathbf{X}_t\|_F) = \mathbf{U}_t \boldsymbol{\Sigma}_t \mathbf{V}_t^T$$

dove

- $\mathbf{U}_t \in \mathbb{R}^{m \times t}$ contiene le prime t colonne di \mathbf{U} , ossia i primi t vettori singolari sinistri di \mathbf{X} ;
- $\boldsymbol{\Sigma}_t \in \mathbb{R}^{t \times t}$ contiene, sulla diagonale, i primi t valori singolari non nulli di \mathbf{X} ;
- $\mathbf{V}_t^T \in \mathbb{R}^{t \times n}$ contiene le prime t righe di \mathbf{V}^T , ossia i primi t vettori singolari destri di \mathbf{X} ;
- $\|\mathbf{X} - \mathbf{X}_t\|_F$ è la norma di Frobenius del *discostamento* di \mathbf{X}_t da \mathbf{X} .

La TSVD, dunque, al variare del suo troncamento al rango t , fornisce una gerarchia di approssimazioni *ottimali* della matrice originale \mathbf{X} . Poiché la matrice $\boldsymbol{\Sigma}_t$ è diagonale, otteniamo che l'approssimazione $\tilde{\mathbf{X}}_t$ al rango t di \mathbf{X} fornita da SVD è data da

$$\tilde{\mathbf{X}}_t = \sum_{i=1}^t \sigma_i \mathbf{u}_i \mathbf{v}_i^T = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^T + \sigma_2 \mathbf{u}_2 \mathbf{v}_2^T + \cdots + \sigma_t \mathbf{u}_t \mathbf{v}_t^T \approx \mathbf{X}$$

dove l'uguaglianza $\tilde{\mathbf{X}}_t = \mathbf{X}$ vale solo se $t \geq r$. Come dimostrato nella Sezione 3.2 otteniamo

$$\|\tilde{\mathbf{X}}_t\|_F^2 = \sum_{i=1}^t \sigma_i^2$$

Si può facilmente dimostrare che il quadrato della norma di Frobenius del *discostamento* è uguale alla somma dei quadrati dei valori singolari *persi* dopo il troncamento:

$$\|\mathbf{X} - \tilde{\mathbf{X}}_t\|_F^2 = \sum_{i=t+1}^r \sigma_i^2$$

Il rapporto tra il numero di elementi della fattorizzazione troncata $\mathbf{U}_t \boldsymbol{\Sigma}_t \mathbf{V}_t^T$ e quello di \mathbf{X} è dato da

$$\frac{|\mathbf{U}_t| + |\boldsymbol{\Sigma}_t| + |\mathbf{V}_t^T|}{|\mathbf{X}|} = \frac{mt + t^2 + tn}{mn}$$

Se tale rapporto è ≤ 1 allora, grazie alla TSVD, verrà risparmiato $mn - (mt + t^2 + tn)$ spazio in memoria. Si può facilmente dimostrare che, memorizzando i fattori \mathbf{U}_t , $\boldsymbol{\Sigma}_t$ e \mathbf{V}_t^T anziché dell'intera matrice \mathbf{X} , si può risparmiare spazio se e solo se

$$t \leq \left\lfloor \frac{\sqrt{(m+n)^2 + 4mn} - (m+n)}{2} \right\rfloor$$

Supponiamo, ad esempio, che la matrice \mathbf{X} abbia 2000 righe e 1500 colonne. Otteniamo che il *threshold* massimo che permette di risparmiare spazio è $t = 712$. Di seguito sono elencati alcuni valori del rapporto tra gli elementi totali della fattorizzazione e quelli della matrice originaria, per alcuni valori della soglia $t \leq 712$:

- Se $t = 150$, allora il rapporto è circa 18%;
- Se $t = 100$, allora il rapporto è 12%;
- Se $t = \lceil \sqrt{1500} \rceil = 39$, allora il rapporto è circa 5%;
- Se $t = \lceil \log 1500 \rceil = 8$, allora il rapporto è circa 0%;

Osserviamo che nel caso di SVD economica, $t = r$. Se \mathbf{X} è a rango completo, allora $t = r = n = 1500$ e il rapporto precedente diventa

$$\frac{mr + r^2 + nr}{mn} = \frac{2n^2 + mn}{mn} = \frac{2n + m}{m}$$

Sostituendo $m = 2000$ e $n = 1500$ otteniamo che il rapporto è $\frac{5}{2} = 250\%$.

In questa tesi, tale versione *troncata* della decomposizione verrà indicata con l'acronimo TSVD.

3.6 Scelta del *threshold* per SVD troncata

La scelta del *threshold* $t \in \mathbb{N}$ ottimale che catturi la maggior parte dei valori singolari significativi di $\mathbf{X} \in \mathbb{R}^{m \times n}$ è di fondamentale importanza quando si vuole lavorare con approssimazioni della matrice originale.

Una tecnica comune consiste nel mantenere solo i valori singolari che conservano una percentuale $\pi \leq 1$ prestabilita della varianza (o energia) dei dati originali, ossia

$$\|\tilde{\mathbf{X}}_t\|_F^2 = \sum_{i=1}^t \sigma_i^2 \geq \pi \sum_{i=1}^r \sigma_i^2 = \pi \|\mathbf{X}\|_F^2$$

dove comunemente $0,9 \leq \pi \leq 1$, in modo che si possa catturare almeno il 90% della varianza. Sebbene questo approccio sia relativamente semplice, è ampiamente utilizzato.

Altre tecniche consistono nell'identificare il punto di *svolta* dei valori singolari della matrice, ossia la transizione da valori singolari significativi a valori singolari relativamente *piccoli*. M. Gavish e D. L. Donoho [6] hanno tentato di calcolare analiticamente il *threshold* ottimale. Supponiamo, per ipotesi, che la matrice originale \mathbf{X} sia data dalla somma di dati *puri* $\mathbf{X}_{\text{true}} \in \mathbb{R}^{m \times n}$ e rumore bianco² $\mathbf{X}_{\text{noise}} \in \mathbb{R}^{m \times n}$, secondo l'equazione

$$\mathbf{X} = \mathbf{X}_{\text{true}} + \gamma \mathbf{X}_{\text{noise}}$$

dove $\gamma \in \mathbb{R}$ è la magnitudine del rumore. Se γ è conosciuto, allora il *threshold* t è dato dalla discretizzazione della seguente legge continua

$$\tau = \lambda(\beta) \sqrt{n} \gamma$$

dove

$$\lambda(\beta) = \sqrt{\left(2(\beta + 1) + \frac{8\beta}{\beta + 1 + \sqrt{\beta^2 + 14\beta + 1}} \right)}$$

²In tale contesto il rumore bianco si riferisce a un modello statistico in cui il rumore è costituito da componenti casuali indipendenti e identicamente distribuite (i.i.d.), con media zero e varianza costante.

è una funzione del rapporto tra righe e colonne (o colonne e righe)

$$\beta = \begin{cases} \frac{m}{n} & \text{se } n \geq m, \\ \frac{n}{m} & \text{altrimenti.} \end{cases}$$

Osserviamo che, nel caso di matrice quadrate, $m = n$, si ha $\beta = 1$ e $\lambda(\beta) = \frac{4}{\sqrt{3}}$, da cui

$$\tau = \frac{4}{\sqrt{3}} \sqrt{n} \gamma$$

Nella Sezione A.1 riportiamo lo script Matlab che utilizzeremo per calcolare il *threshold* nel caso di γ noto.

Nella maggior parte dei casi concreti, la magnitudine γ è sconosciuta ed è necessario stimare il rumore bianco. L'idea di Gavish e Donoho è utilizzare il valore singolare mediano σ_{median} . In tal caso, si può dimostrare che il *threshold* ottimale segue la seguente legge

$$\tau = \frac{\lambda(\beta)}{\mu_\beta} \sigma_{\text{median}}$$

dove μ_β è la soluzione della seguente equazione integrale³

$$\int_{(1-\beta)^2}^{\mu_\beta} \frac{\sqrt{((1+\sqrt{\beta})^2 - \tau)(\tau - (1-\sqrt{\beta})^2)}}{2\pi\tau} d\tau = \frac{1}{2}$$

che può essere approssimata con uno script Matlab fornito dagli stessi autori [5].

In fase di computazione, poiché sia $\lambda(\beta)$ che μ_β sono valori reali, è necessario effettuare una discretizzazione di $\tau \in \mathbb{R}$ per ottenere un valore intero di $t \in \mathbb{N}$. Nel nostro test, abbiamo scelto di approssimare il risultato per eccesso, utilizzando l'operatore $\lceil \cdot \rceil$. Inoltre, in alcune configurazioni, specialmente per valori m e n di *confine*, potrebbe accadere che $\tau > n$, una situazione che non ha senso pratico. Tenendo conto di queste considerazioni, il calcolo di t , sia in presenza di rumore noto sia quando il rumore è stimato, diventa

Definizione 3.2 (Threshold ottimale secondo M. Gavish e D. L. Donoho)

$$t = \min(\lceil \tau \rceil, n), \quad \text{dove}$$

$$\tau = \begin{cases} \lambda(\beta) \sqrt{n} \gamma & \text{se } \gamma \text{ è noto,} \\ \frac{\lambda(\beta)}{\mu_\beta} \sigma_{\text{median}} & \text{altrimenti.} \end{cases}$$

Questa formulazione garantisce che il valore di t sia sempre coerente con le dimensioni della matrice X , rispettando i vincoli pratici e teorici del problema. Si può dimostrare, in entrambi i casi, che $t \in \mathcal{O}(\sqrt{n})$.

³Tale integrale è noto come distribuzione di Marčenko-Pastur. Nell'equazione, μ_β è il valore mediano della distribuzione.

4

Randomized SVD

In questo capitolo introduciamo l'uso della randomizzazione nel calcolo della SVD. L'algoritmo che utilizziamo è quello proposto da Steven L. Brunton e J. Nathan Kutz [1]. Inizialmente eseguiremo un semplice test preliminare per osservare, in prima analisi, i miglioramenti prestazionali apportati dalla RSVD e gli errori introdotti. Successivamente, eseguiremo un test iterativo che ci permetterà di analizzare i tempi di esecuzione e gli errori di RSVD al variare del numero di colonne della matrice in input, confrontando i risultati con quelli ottenuti con la SVD deterministica. Infine, ripeteremo il test utilizzando un *threshold* logaritmico e confronderemo questi risultati con quelli ottenuti utilizzando il criterio di *threshold* proposto da M. Gavish e D. L. Donoho [6].

4.1 Motivazioni della randomizzazione

Se una matrice ha una struttura a basso rango, algoritmi di decomposizione basati su campionamento casuale possono ottenere una buona approssimazione del rango basso, ma a una frazione del costo computazionale rispetto ai metodi tradizionali (deterministici). Questo approccio è strettamente legato alla teoria della sparsità e alla geometria degli spazi ad alta dimensione. Con l'aumento esponenziale delle dimensioni dei dati (es. video in 4K e 8K, Internet of Things), il volume di misurazioni cresce enormemente, ma il rango intrinseco dei dati non aumenta in modo proporzionale. Questo significa che, anche se le dimensioni dei dataset crescono, la parte rilevante delle informazioni contenute rimane contenuta in poche componenti principali. Di conseguenza, le tecniche randomizzate per la decomposizione di matrici, come la Randomized SVD, stanno diventando sempre più importanti poiché permettono di gestire questo *diluvio* di dati con minori costi computazionali.

4.2 Un algoritmo randomizzato per il calcolo di SVD

Consideriamo una matrice $\mathbf{X} \in \mathbb{R}^{m \times n}$ stretta e alta, ossia $m \geq n$ (possibilmente $m \gg n$). Studiamo come algoritmo per il calcolo randomizzato di SVD quello proposto da Steven L. Brunton e Nathan Kutz [1].

1. Proiezione \mathbf{Z} .

Utilizziamo $\mathbf{P} \in \mathbb{R}^{n \times t}$, con t ottenuto secondo la Definizione 3.2, per proiettare \mathbf{X} su uno spazio di dimensioni inferiori: $\mathbf{Z} = \mathbf{X}\mathbf{P} \in \mathbb{R}^{m \times t}$. La matrice ottenuta \mathbf{Z} è molto più piccola e maneggevole di \mathbf{X} . La casualità della matrice \mathbf{P} rende altamente improbabile che le componenti rilevanti della matrice \mathbf{X} vengano eliminate durante la proiezione.

2. Decomposizione QR di \mathbf{Z} .

Fattorizziamo \mathbf{Z} usando la decomposizione QR, ottenendo $\mathbf{Z} = \mathbf{Q}\mathbf{R}$, con $\mathbf{Q} \in \mathbb{R}^{m \times t}$ matrice con colonne ortonormali e $\mathbf{R} \in \mathbb{R}^{t \times t}$ matrice triangolare superiore. In questo modo troviamo una base ortonormale per la matrice \mathbf{Z} che approssimi la base ortonormale per la matrice originaria \mathbf{X} . Ciò è molto utile perché semplifica i calcoli successivi: la moltiplicazione di una matrice per una matrice con colonne ortogonali non introdurrà distorsioni numeriche e preserva le proprietà geometriche del sottospazio. Questo è dovuto al fatto che \mathbf{Q} copre l'intero sottospazio generato dalle colonne di \mathbf{Z} .

3. Proiezione ortogonale \mathbf{Y} .

Proiettiamo \mathbf{X} sul sottospazio identificato da \mathbf{Q} , ottenendo $\mathbf{Y} = \mathbf{Q}^T \mathbf{X} \in \mathbb{R}^{t \times n}$. Ciò riduce le dimensioni del problema, consentendo di eseguire la decomposizione SVD su una matrice molto più piccola, mantenendo comunque le componenti più importanti di \mathbf{X} . Questo accade perché la decomposizione QR fornisce una base ortonormalizzata per lo spazio generato dalle colonne di \mathbf{Z} , preservando le direzioni principali del sottospazio proiettato. In altri termini, \mathbf{Q} approssima uno spazio di rango t che contiene quasi tutte le informazioni essenziali di \mathbf{X} . Dunque, vale $\mathbf{X} \approx \mathbf{Q}\mathbf{Y}$.

4. SVD di \mathbf{Y} .

Decomponiamo \mathbf{Y} con SVD, utilizzando preferibilmente la sua versione economica, ottenendo così

$$\mathbf{Y} = \mathbf{U}_Y \boldsymbol{\Sigma}_Y \mathbf{V}^T = \mathbf{U}_Y \begin{bmatrix} \boldsymbol{\Sigma}_{Y,t} & \mathbf{0}_Y \end{bmatrix} \begin{bmatrix} \mathbf{V}_t^T \\ \mathbf{V}_\perp^T \end{bmatrix} = \mathbf{U}_Y \boldsymbol{\Sigma}_{Y,t} \mathbf{V}_t^T$$

con $\mathbf{U}_Y \in \mathbb{R}^{t \times t}$, $\boldsymbol{\Sigma}_Y \in \mathbb{R}^{t \times n}$, $\boldsymbol{\Sigma}_{Y,t} \in \mathbb{R}^{t \times t}$, $\mathbf{0}_Y \in \mathbb{R}^{t \times (n-t)}$, $\mathbf{V} \in \mathbb{R}^{n \times n}$, $\mathbf{V}_t^T \in \mathbb{R}^{t \times n}$ e $\mathbf{V}_\perp^T \in \mathbb{R}^{(n-t) \times n}$. Ciò sarà computazionalmente meno costoso rispetto a decomporre direttamente \mathbf{X} , perché \mathbf{Y} ha dimensioni più piccole. Per paragone, consideriamo la SVD economica sull'intera matrice originale:

$$\mathbf{X} = \mathbf{U} \boldsymbol{\Sigma} \mathbf{V}^T = \begin{bmatrix} \mathbf{U}_n & \mathbf{U}_\perp \end{bmatrix} \begin{bmatrix} \boldsymbol{\Sigma}_n \\ \mathbf{0} \end{bmatrix} \mathbf{V}^T$$

con $\mathbf{U} \in \mathbb{R}^{m \times m}$, $\mathbf{U}_n \in \mathbb{R}^{m \times n}$, $\mathbf{U}_\perp \in \mathbb{R}^{m \times (m-n)}$, $\boldsymbol{\Sigma} \in \mathbb{R}^{m \times n}$, $\boldsymbol{\Sigma}_n \in \mathbb{R}^{n \times n}$, $\mathbf{0} \in \mathbb{R}^{(m-n) \times n}$, $\mathbf{V} \in \mathbb{R}^{n \times n}$.

5. Costruzione della matrice approssimata $\hat{\mathbf{X}}_t$.

Per ottenere l'approssimazione $\hat{\mathbf{X}}_t$ di \mathbf{X} è sufficiente costruire la matrice $\hat{\mathbf{U}}_t \in \mathbb{R}^{m \times t}$ tale che

$$\hat{\mathbf{X}}_t = \hat{\mathbf{U}}_t \boldsymbol{\Sigma}_{Y,t} \mathbf{V}_t^T$$

Ciò può essere fatto moltiplicando $\mathbf{Q} \in \mathbb{R}^{m \times t}$ per la matrice \mathbf{U}_Y , ottenuta dalla SVD di \mathbf{Y} :

$$\hat{\mathbf{U}}_t = \mathbf{Q}\mathbf{U}_Y$$

In altre parole, l'approssimazione dei primi t vettori singolari sinistri di \mathbf{X} può essere effettuata proiettando all'indietro, nello spazio generato dalle colonne di \mathbf{Q} , i vettori singolari sinistri di \mathbf{Y} .

Nella Sezione A.2 forniamo il codice Matlab dell'algoritmo appena descritto. L'algoritmo può essere facilmente adattato per matrici basse e larghe ($m < n$). Tuttavia, in questa tesi non analizzeremo tale caso né dal punto di vista teorico, né sperimentale, in quanto i risultati ottenuti sarebbero analoghi.

4.3 Analisi di complessità teorica

Studiamo ora i benefici, in termini di tempo computazionale, apportati dalla randomizzazione nel calcolo della fattorizzazione SVD di una matrice $\mathbf{X} \in \mathbb{R}^{m \times n}$. Come termine paragone utilizziamo la complessità del classico algoritmo utilizzato per calcolare SVD, la cui complessità è $\mathcal{O}(mn^2)$. [2] Per semplicità supponiamo che i prodotti matriciali \mathbf{AB} , con $\mathbf{A} \in \mathbb{R}^{i \times j}$ e $\mathbf{B} \in \mathbb{R}^{j \times k}$, vengano calcolati utilizzando i classici algoritmi di complessità $\mathcal{O}(ijk)$. Analizziamo la complessità di ogni passo dell'algoritmo randomizzato:

1. La proiezione $\mathbf{Z} = \mathbf{XP}$ ha complessità $\mathcal{O}(mnt)$, dove t è il numero di colonne della matrice \mathbf{P} . La matrice $\mathbf{P} \in \mathbb{R}^{n \times t}$ è generata casualmente, ma questo ha costo trascurabile rispetto alla moltiplicazione matriciale.
2. La decomposizione QR su $\mathbf{Z} \in \mathbb{R}^{m \times t}$ ha complessità $\mathcal{O}(mt^2)$, perché \mathbf{Z} ha solo t colonne.
3. La proiezione $\mathbf{Y} = \mathbf{Q}^T \mathbf{Z}$ ha complessità $\mathcal{O}(mnt)$.
4. La SVD della matrice $\mathbf{Y} \in \mathbb{R}^{t \times n}$ ha complessità $\mathcal{O}(t^2n)$. Notare che \mathbf{Y} ha dimensioni molto più piccole rispetto a \mathbf{X} se $t \ll m$.
5. La ricostruzione di \mathbf{U} dalla moltiplicazione $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}_Y$ ha complessità $\mathcal{O}(mt^2)$.

Sommendo tutti i contributi, otteniamo la complessità totale:

$$\mathcal{O}(mnt) + \mathcal{O}(mt^2) + \mathcal{O}(t^2n) = \mathcal{O}(\max(mnt, mt^2, t^2n))$$

Poiché l'algoritmo randomizzato ha come scopo la riduzione della dimensionalità del problema, possiamo supporre $t \ll n$ senza perdita di generalità. Otteniamo così $\mathcal{O}(mnt)$. Supponendo, plausibilmente, che il *threshold* sia logaritmico rispetto alla dimensione delle colonne, $t \in \mathcal{O}(\log n)$, la complessità diventa $\mathcal{O}(mn \log n)$. Se, invece, il *threshold* è quello ottimale [6], $t \in \mathcal{O}(\sqrt{n})$, allora $\mathcal{O}(mnt) = \mathcal{O}(mn\sqrt{n})$.

4.4 Esecuzione della Randomized SVD su un'immagine

L'algoritmo proposto fornisce un'approssimazione $\mathbf{U}_t = \mathbf{Q}\hat{\mathbf{U}}_Y \in \mathbb{R}^{m \times t}$ della matrice ortogonale $\mathbf{U} \in \mathbb{R}^{m \times m}$ contenente i vettori singolari sinistri della matrice originale $\mathbf{X} \in \mathbb{R}^{m \times n}$. Calcoliamo l'errore

d'approssimazione dovuto all'utilizzo della SVD randomizzata, confrontandolo con quello derivante dalla SVD troncata (senza randomizzazione). Sia

- $\tilde{e}_t = \frac{\|\mathbf{X} - \tilde{\mathbf{X}}_t\|_F}{\|\mathbf{X}\|_F}$ errore relativo di $\tilde{\mathbf{X}}_t$, ottenuta dalla SVD di \mathbf{X} troncata al rango t . Dal Teorema 3.1 sappiamo che $\tilde{\mathbf{X}}_t$ è la matrice che meglio approssima \mathbf{X} al rango t .

- $\hat{e}_t = \frac{\|\mathbf{X} - \hat{\mathbf{X}}_t\|_F}{\|\mathbf{X}\|_F}$ errore relativo di $\hat{\mathbf{X}}_t$, ottenuta dalla SVD randomizzata di \mathbf{X} troncata al rango t . Esso sarà maggiore rispetto al precedente e vogliamo valutarne l'entità.

Consideriamo come matrice d'esempio un'immagine $\mathbf{X} \in \mathbb{Q}^{2397 \times 1795}$ i cui elementi sono pixel (Figura 4.1). Sia $\gamma = 1$. Settiamo la magnitudine del rumore bianco. Il *threshold* t , calcolato come indicato nella Definizione 3.2, sarà $t = \min(\lceil \lambda(\beta) \sqrt{n} \gamma \rceil, n) = 92$. Lo spazio colonne ridotto è il $\frac{t+p}{n} = \frac{92}{1795} \approx 5\%$ di quello originale. La matrice originale, di dimensione $2397 \cdot 1795 = 4302615$, può essere fattorizzata in tre matrici la cui dimensione complessiva è $2397 \cdot 92 + 92^2 + 92 \cdot 1795 = 394128$, circa il 9%.



Figura 4.1: Immagine originale.



Figura 4.2: Immagine dopo la SVD troncata.

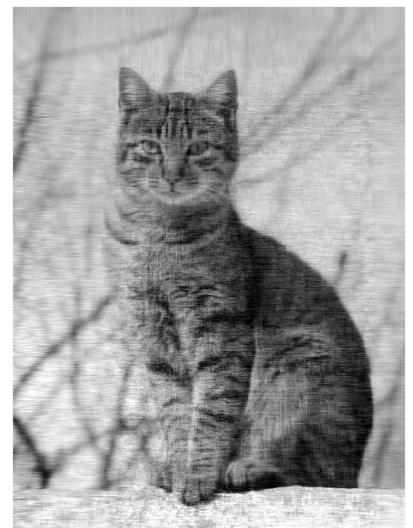


Figura 4.3: Immagine dopo la SVD randomizzata.

Ottieniamo, in output, gli errori relativi e i tempi di esecuzione descritti nella Tabella 4.1. L'errore

	Errore relativo	Tempo di esecuzione [s]
TSVD	0,007076	2,5259
RSVD	0,020356	0,04703

Tabella 4.1: Errori e tempi di esecuzione per le due varianti di SVD.

$e_{\text{RSVD}} = 0,020356$ della SVD randomizzata è circa 3 volte l'errore $e_{\text{TSVD}} = 0,007076$ della SVD troncata. Tale perdita d'accuracy è giustificata da un tempo di esecuzione 54 volte inferiore rispetto a quello della SVD troncata. In base al contesto e all'applicazione concreta bisogna scegliere quale dei due aspetti, prestazioni o accuracy, valorizzare. Per matrici di dimensioni elevate¹ spesso è utile ridurre il tempo di esecuzione anche se ciò potrebbe indurre un errore sul risultato finale.

¹Solitamente, nel contesto della RandNLA, le matrici considerate *grandi* hanno milioni di righe e migliaia, o milioni, di colonne.

Nella Figura 4.4, che raffigura i valori singolari in ordine decrescente in percentuale rispetto alla loro somma complessiva, possiamo notare che solo alcuni di essi sono realmente significativi. Il valore singolare più elevato, da solo, ha una magnitudine pari al 21% della somma di tutti i valori singolari. Il *threshold* usato per troncare SVD cattura i primi 92 valori singolari, la cui somma è circa il 56% di quella complessiva.

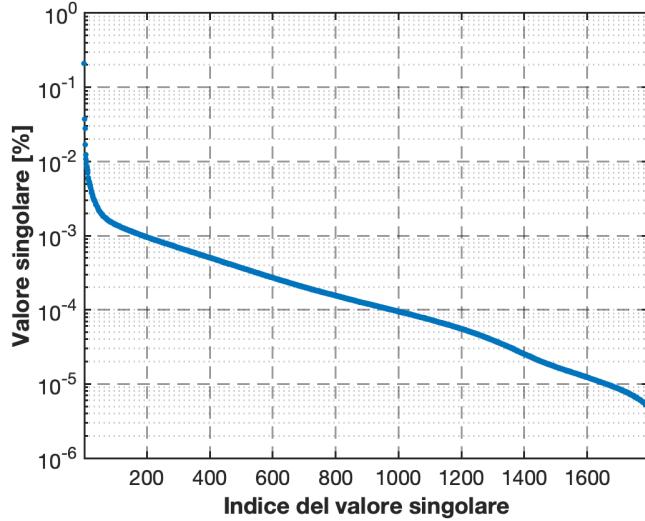


Figura 4.4: I valori singolari di \mathbf{X} in percentuale rispetto alla somma complessiva degli stessi (scala logaritmica sulle ordinate).

In sintesi, i risultati ottenuti indicano che il troncamento *ottimale*, corrispondente in tal caso al 5% dello spazio colonne, ha consentito di catturare il 56% delle componenti significative della matrice originale, mentre la randomizzazione ha ridotto il tempo di esecuzione di un fattore di 54 rispetto al metodo deterministico. Nella Sezione A.3 riportiamo lo script Matlab che ha permesso di eseguire questo test.

4.5 Test delle prestazioni e dell'accuratezza di RSVD

In questa sezione analizziamo, al variare delle colonne della matrice in input, le prestazioni e l'errore relativo della SVD randomizzata e troncata. La matrice di input sarà un'immagine quadrata $\mathbf{X} \in \mathbb{Q}^{12000 \times 12000}$, riportata nella Figura 4.5 assieme alle sue approssimazioni tramite le due varianti di SVD (Figura 4.6 e 4.7).

L'esperimento consiste nel mantenere fisso il numero di righe (12000) e incrementare il numero di colonne da 120 a 12000; a ogni passo calcoliamo le due versioni di SVD, memorizzandone i tempi di esecuzione, gli errori relativi e il *threshold*. Impostiamo la granularità dell'incremento a 120, così da ridurre accettabilmente di 120 volte le computazioni. Il *threshold* t sarà calcolato a ogni computazione, nel modo indicato nella Definizione 3.2, con $\gamma = 1$. I dati ottenuti dall'esperimento risultano simili mantenendo fisso il numero di colonne e variando quello delle righe; di conseguenza, non approfondiremo ulteriormente questo caso.



Figura 4.5: Immagine originale.



Figura 4.6: Immagine dopo la SVD troncata.



Figura 4.7: Immagine dopo la SVD randomizzata.

I risultati mostrano una correlazione² significativa tra il tempo di esecuzione e l'errore relativo, così come tra il tempo e il *threshold*. In particolare, la correlazione tra il tempo e l'errore relativo è pari a $-0,860530$, indicando una forte relazione inversa: all'aumentare del tempo di esecuzione, l'errore tende a diminuire. Al contrario, la correlazione tra il tempo e il *threshold* è positiva ($0,926408$), suggerendo che matrici con un numero maggiore di colonne richiedono tempi di calcolo più lunghi, e ciò è prevedibilmente legato all'aumento del rango target. Inoltre, l'errore diminuisce all'aumentare del *threshold*, come evidenziato dalla correlazione negativa tra errore e *threshold* ($-0,919566$).

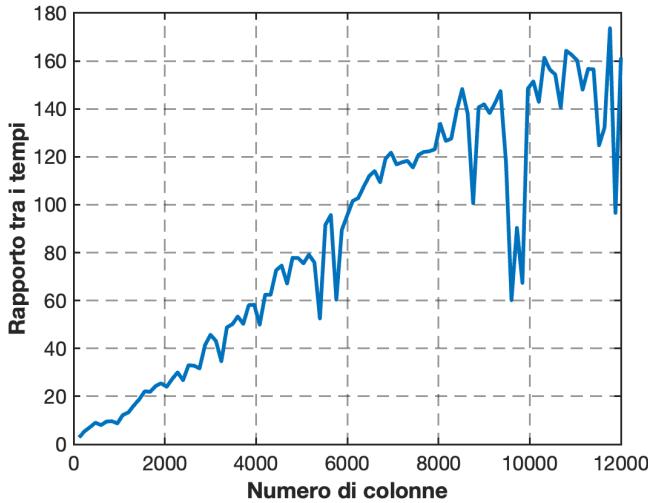


Figura 4.8: Rapporto, al variare del numero di colonne, tra il tempo di esecuzione di TSVD e quello di RSVD.

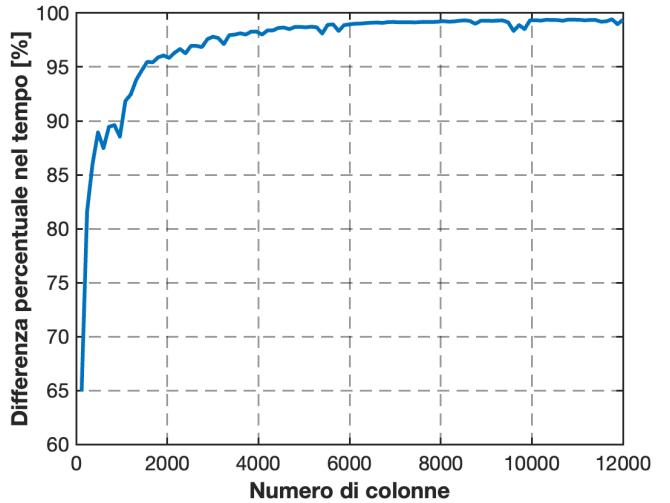


Figura 4.9: Differenze tra i tempi di esecuzione tra TSVD e RSVD in percentuale sui tempi di TSVD, al variare delle colonne.

²La correlazione di Pearson tra due variabili X e Y è definita dalla seguente formula

$$\frac{\sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^n (X_i - \bar{X})^2 \sum_{i=1}^n (Y_i - \bar{Y})^2}}$$

dove

- X_i e Y_i sono i valori delle due variabili;
- \bar{X} e \bar{Y} sono le medie delle variabili X e Y , rispettivamente;
- n è il numero di coppie di dati.

Come discusso nella Sezione 4.3, la complessità dell'algoritmo deterministico è $\mathcal{O}(mn^2)$, mentre quella dell'algoritmo randomizzato è $\mathcal{O}(mnt)$. Quest'ultima complessità diventa $\mathcal{O}(mn\sqrt{n})$ quando il valore ottimale del parametro t , come indicato nella Definizione 3.2, è $t = \min(\lceil \lambda(\beta)\sqrt{n}\gamma \rceil, n) \in \mathcal{O}(\sqrt{n})$. Di conseguenza, il rapporto tra i tempi di esecuzione dovrebbe seguire un andamento pari a $\mathcal{O}\left(\frac{mn^2}{mn\sqrt{n}}\right) = \mathcal{O}(\sqrt{n})$, andamento confermato dal grafico ottenuto sperimentalmente (Figura 4.8).

Il tempo di esecuzione medio è di 88,870873 secondi per l'algoritmo deterministico e di 0,765446 per quello randomizzato, quest'ultimo uguale allo 0,86% di quello di TSVD. Il tempo massimo misurato per eseguire TSVD, come intuibile, è quello per decomporre la matrice completa (12000×12000): 299,058596 secondi. RSVD ha tempo massimo di 2,921871 secondi per 11880 colonne. La massima durata misurata per RSVD è 102 volte inferiore rispetto a quella di TSVD. Il tempo minimo di esecuzione di TSVD si registra banalmente per la matrice più piccola (120×120): 0,086681 secondi. Il tempo minimo di RSVD è di 0,017116 secondi, corrispondente a 240 colonne.

Per quanto riguarda le differenze nei tempi di esecuzione, la differenza media tra TSVD e RSVD è di 88,105 secondi, con una differenza massima di 297,206 secondi per 12000 colonne, mentre la differenza minima è di soli 0,056 secondi per 120 colonne. Tali differenze sono meglio visualizzabili in percentuale sui tempi di esecuzione della TSVD, come riportato nella Figura 4.9. Osserviamo che tale differenza inizia a superare il 90% dopo le 1000 colonne. Ossia, è sufficiente che il rapporto tra colonne e righe sia $1/12$, per matrici alte e strette, affinché RSVD inizi a essere il 90% più efficiente di TSVD.

L'errore relativo medio, al variare delle colonne, è 0,058845 per TSVD e 0,094308 per RSVD, 1,6 volte più impreciso. Come possiamo notare nella Figura 4.11 l'errore relativo ha un andamento simile tra TSVD e RSVD. Infatti, l'errore massimo è 0,084452 per TSVD e 0,137948 per RSVD entrambi corrispondenti a 1680 colonne. I minimi al variare delle colonne sono 0,043995, corrispondente a 10080 colonne per TSVD, e 0,069688, corrispondente a 9960 colonne, per RSVD. L'andamento *irregolare* dell'errore relativo al variare delle colonne, con un massimo intorno a $7/14$ delle colonne totali e un minimo verso i $5/6$, e con flessi attorno a 4000 e 7000 colonne, può essere interpretato analizzando la distribuzione dei valori singolari nelle diverse sottomatrici. In particolare, osserveremo l'andamento del rapporto percentuale tra la somma dei primi t valori singolari e la loro somma complessiva:

$$\frac{\sum_{i=1}^t \sigma_i}{\sum_{i=1}^r \sigma_i} \cdot 100$$

Come mostrato nella Figura 4.10, la somma dei primi t valori singolari varia significativamente: passando dal 75% al 58% del totale quando il numero di colonne aumenta da 120 a 1680. Ciò spiega l'aumento iniziale dell'errore, fino al massimo. Successivamente, tra 1680 e 8760 colonne, la somma percentuale dei primi t valori singolari torna a crescere leggermente, passando dal 58% al 61%. Questa variazione si riflette in una corrispondente diminuzione dell'errore relativo. Oltre le 8760 colonne, la somma percentuale ricomincia a decrescere, sebbene più lentamente rispetto alla fase iniziale. Questo comportamento giustifica l'aumento finale dell'errore relativo.

Le differenze negli errori relativi di RSVD e TSVD sono le seguenti: la differenza media è 0,0355, con una differenza massima di 0,0535 per 1680 colonne, e una differenza minima di 0,0257 per 9960 colonne. Le differenze in percentuale sugli errori relativi di RSVD sono riportate nella Figura 4.12. Le differenze degli errori in percentuale sugli errori di RSVD si stabilizzano sul 37% dopo le 4000 colonne

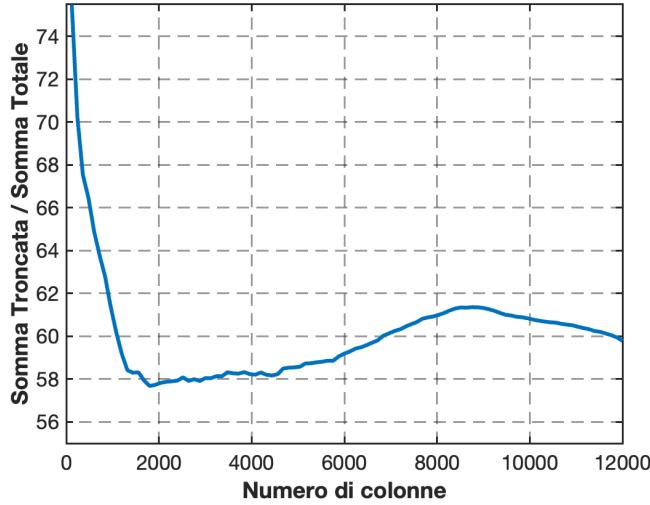


Figura 4.10: Somma dei valori singolari troncati in percentuale sulla somma di tutti valori singolari.

(corrispondente, rispettivamente, a una matrice con rapporto righe-colonne di 3 : 1). Ciò significa che, per matrici con tale rapporto, RSVD avrà un errore relativo del 20% più alto rispetto a quello di TSVD superate le 4000 colonne.

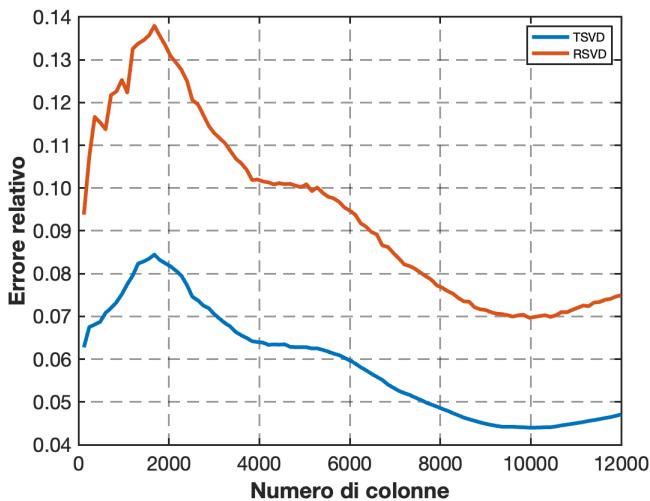


Figura 4.11: Errore relativo al variare del numero di colonne.

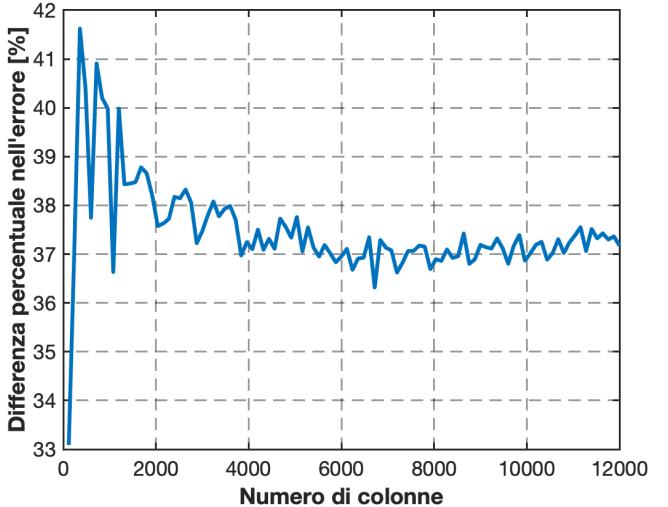


Figura 4.12: Differenze tra gli errori relativi tra RSVD e TSVD in percentuale sugli errori di RSVD, al variare delle colonne.

Il *threshold* segue l'andamento di \sqrt{n} . Non c'è da stupirsi, è sufficiente analizzare la funzione che lo definisce. Nel caso di righe fisse e colonne variabili si ha $\beta(m, n) = \min(m/n, n/m) = n/m$ siccome in questo caso le colonne, durante le iterazioni, sono minori o uguali delle righe. Inoltre, notiamo che $m = 12000$ è fisso, dunque $\beta(m, n) = \beta(n)$ è una funzione nel dominio delle colonne. Lo stesso varrà per $\lambda(\beta) = \lambda(n)$ e per τ , che sarà

$$\tau(n) = \lambda(n)\sqrt{n}\gamma = \gamma \sqrt{\left(2\left(\frac{n}{m} + 1 \right) + \frac{8\frac{n}{m}}{\frac{n}{m} + 1 + \sqrt{\frac{n^2}{m} + 14\frac{n}{m} + 1}} \right) n}$$

in cui i parametri noti sono $m = 12000$ e $\gamma = 1$. Otteniamo il grafico in Figura 4.13. Ovviamente, il

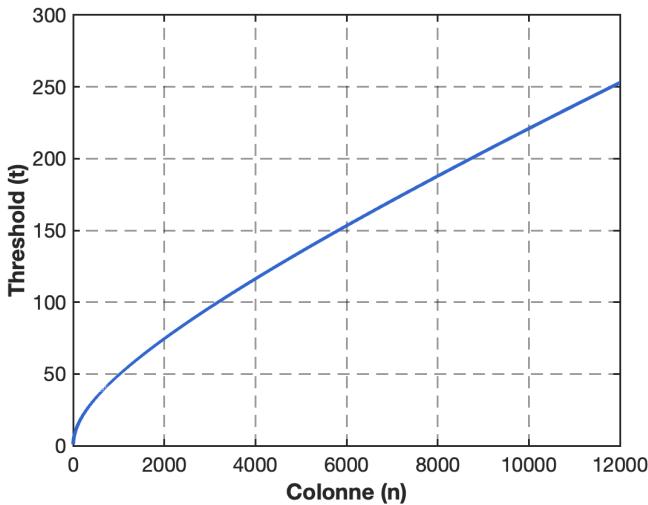


Figura 4.13: Grafico del *threshold* continuo τ al variare delle colonne.

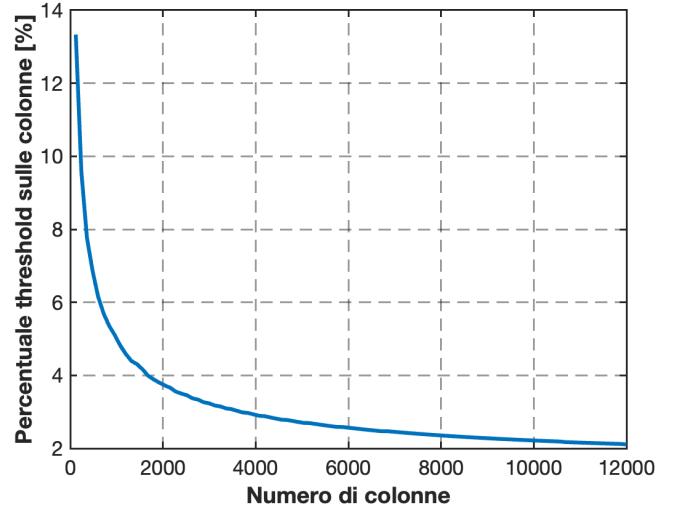


Figura 4.14: *Threshold* t in percentuale sul numero di colonne del test.

grafico del *threshold* t ottenuto durante l'esperimento ricalca³ il comportamento continuo τ , pertanto ci riferiremo indistintamente a entrambi. Al variare delle colonne, il massimo ottenuto è $t = 253$ in corrispondenza della matrice massima (12000×12000). Il minimo è $t(n = 120) = 16$. Il *threshold* medio è 150. Nella Figura 4.14 riportiamo il rapporto tra *threshold* e colonne in percentuale.

In conclusione, i risultati indicano che, sebbene la RSVD offra significativi vantaggi in termini di tempo di esecuzione rispetto alla TSVD, soprattutto per matrici di grandi dimensioni, l'accuratezza del metodo randomizzato tende ad essere leggermente inferiore, sebbene la differenza sia relativamente piccola. La scelta del *threshold* si conferma un parametro chiave sia per il tempo di esecuzione che per l'accuratezza della decomposizione. Nella Sezione A.4 riportiamo il codice Matlab che ha permesso di effettuare il test sull'immagine in input e ottenere tutti i grafici e i dati precedentemente descritti.

	Tempo vs Errore	Tempo vs Threshold	Errore vs Threshold
Correlazione	-0,860530	0,926408	-0,919566

Tabella 4.2: Correlazioni lineari tra tempo di esecuzione, errore relativo e threshold.

	Tempo medio [s]		Tempo massimo [s]		Tempo minimo [s]	
	Valore	Colonne	Valore	Colonne	Valore	Colonne
TSVD	88,870873		299,058596		12000	
RSVD	0,765446		2,921871		11880	

Tabella 4.3: Tempi di esecuzione di TSVD e RSVD, con il numero di colonne corrispondenti.

³Naturalmente, salvo approssimazioni o valori di contorno non accettabili, come nel caso in cui $t > n$, per il quale si impone $t = n$. Come indicato nella Definizione 3.2, poniamo $t = \min(\lceil \tau \rceil, n)$.

	Errore medio	Errore massimo		Errore minimo	
		Valore	Colonne	Valore	Colonne
TSVD	0,058845	0,084452	1680	0,043995	10080
RSVD	0,094308	0,137948	1680	0,069688	9960

Tabella 4.4: Errori relativi di TSVD e RSVD, con numero di colonne corrispondenti.

	Media	Massima		Minima	
		Valore	Colonne	Valore	Colonne
Differenza tempi [s]	88,105	297,206	12000	0,056	120
Differenza errori	0,0355	0,0535	1680	0,0257	9960

Tabella 4.5: Differenze nei tempi di esecuzione e negli errori di TSVD e RSVD, con numero di colonne corrispondenti.

	Medio	Massimo		Minimo	
		Valore	Colonne	Valore	Colonne
Threshold	150	253	12000	16	120

Tabella 4.6: *Threshold* di TSVD e RSVD, con numero di colonne corrispondenti.

4.6 Test con *threshold* logaritmico

Nella sezione precedente abbiamo confrontato le prestazioni e l'accuratezza di TSVD e RSVD utilizzando un *threshold* scelto come descritto nella Definizione 3.2. Ora esploriamo le prestazioni dei due algoritmi considerando, sperimentalmente, un *threshold* inferiore rispetto a quello ottimale proposto da Donoho e Gavish [6]. Anche in questo caso, varieremo il numero di colonne della stessa immagine $\mathbf{X} \in \mathbb{R}^{12000 \times 12000}$.

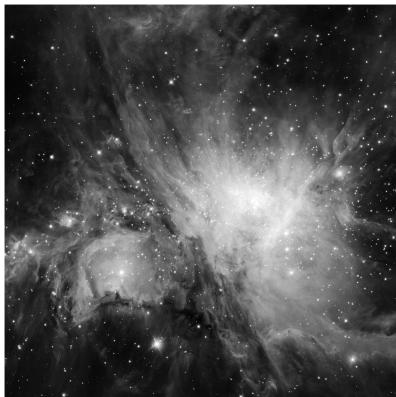


Figura 4.15: Immagine originale.

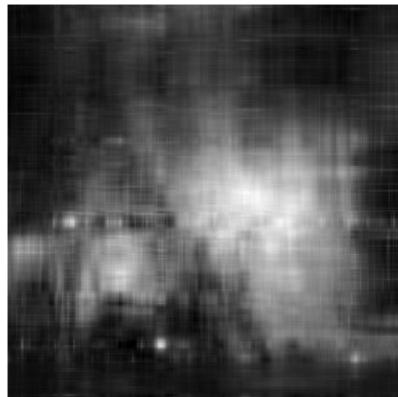


Figura 4.16: Immagine dopo la SVD troncata.

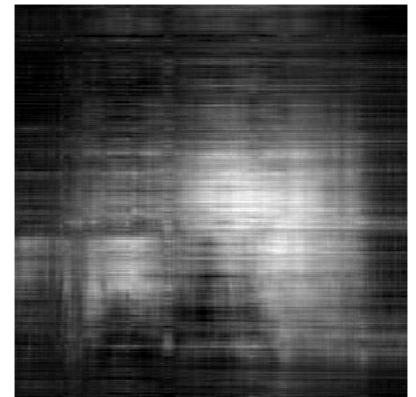


Figura 4.17: Immagine dopo la SVD randomizzata.

Impostiamo $t = \lceil \log n \rceil$. La complessità teorica dell'algoritmo randomizzato è $\mathcal{O}(mn \log n)$, da cui si ricava il rapporto tra la complessità di TSVD e quella di RSVD: $\mathcal{O}\left(\frac{mn^2}{mn \log n}\right) = \mathcal{O}\left(\frac{n}{\log n}\right)$. Il grafico in Figura 4.18, ottenuto sperimentalmente, conferma questo andamento teorico. Studiamo ora i tempi di esecuzione medi, massimi e minimi ottenuti con la nuova soglia, confrontandoli con

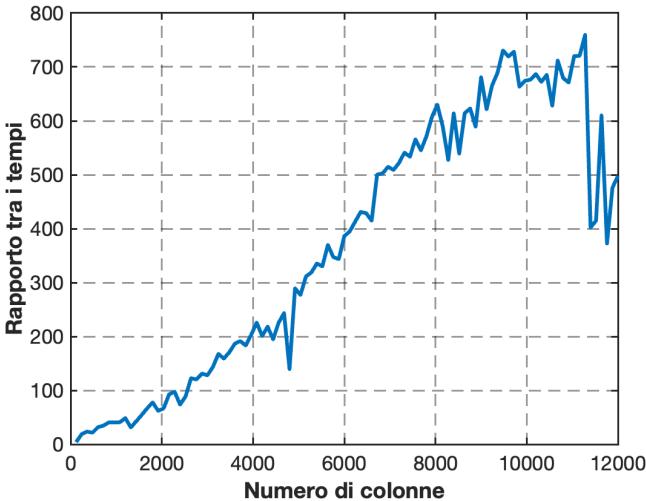


Figura 4.18: Rapporto tra il tempo di esecuzione di TSVD e quello di RSVD, con $t = \lceil \log n \rceil$.

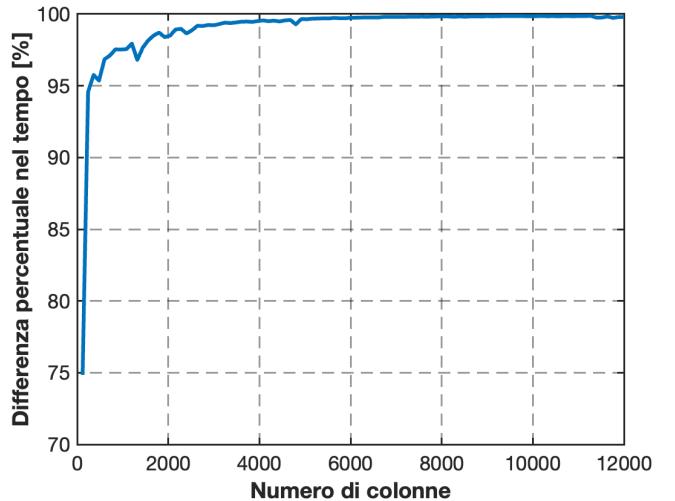


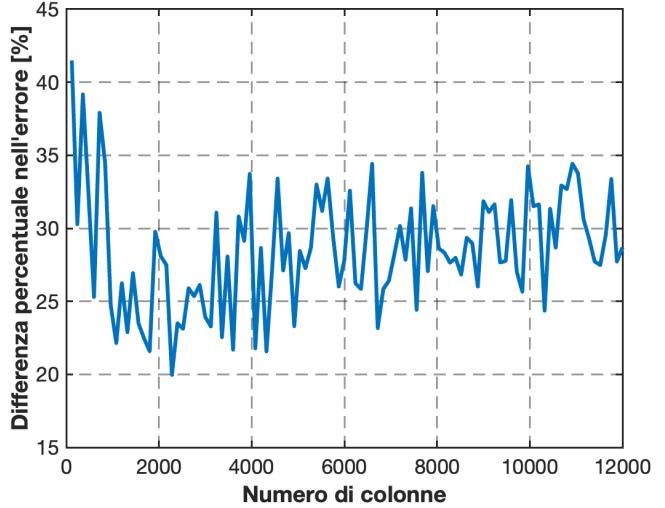
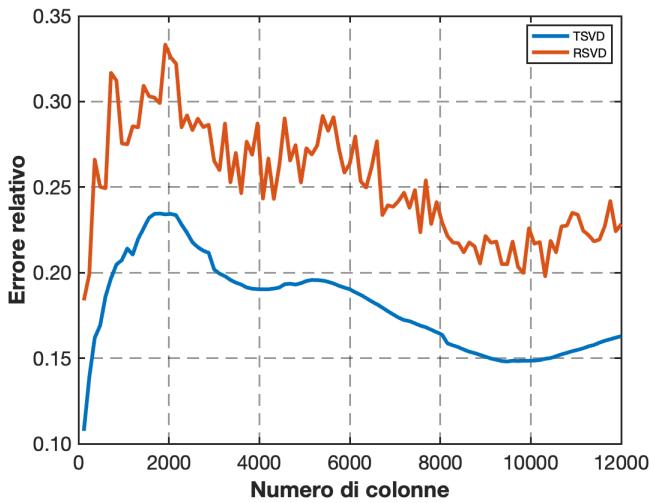
Figura 4.19: Differenze tra i tempi di esecuzione tra TSVD e RSVD in percentuale sui tempi di TSVD, con $t = \lceil \log n \rceil$.

quelli dell'esperimento precedente. Poiché il tempo di esecuzione di TSVD non dipende dal *threshold* utilizzato⁴, si ottengono risultati del tutto simili a quelli precedenti. Il tempo medio è di 0,1777 secondi, presentando una riduzione significativa del 76,8%. Per quanto riguarda il tempo massimo, osserviamo una diminuzione del 74%, corrispondente a 0,7556 secondi su 11760 colonne. Il tempo minimo di esecuzione per RSVD, misurato su 240 colonne (dimensione minima della matrice), è di 0,0048 secondi, circa il 72% inferiore rispetto a prima.

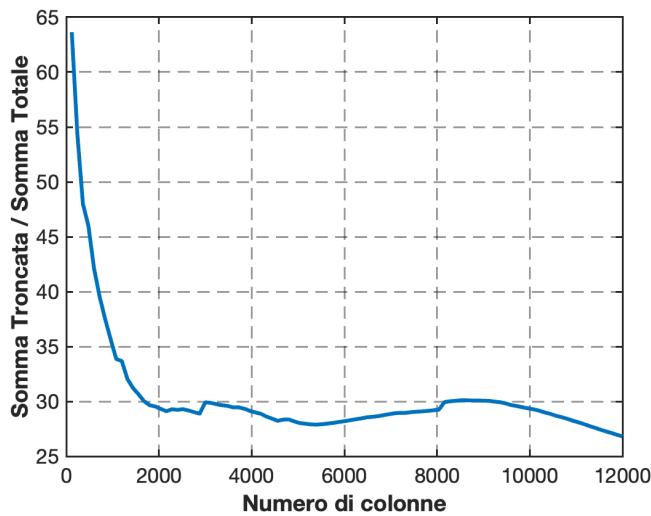
Nella Figura 4.19 è riportata la differenza percentuale nei tempi di esecuzione tra i due algoritmi, rispetto ai tempi di TSVD. Essa ha un andamento analogo a quello della sezione precedente (il 95% viene però raggiunto già a 240 colonne, anziché dopo le 1000). La differenza media è pari a 88,34 secondi, con una differenza massima di 283,70 secondi (per la matrice 12000×12000) e una differenza minima di 0,063 secondi (per la matrice 12000×120). Questi risultati sono abbastanza simili a quelli ottenuti nell'esperimento della sezione precedente, poiché i miglioramenti di efficienza, anche del 75%, per RSVD non risultano significativi quando le dimensioni della matrice sono ridotte.

Dunque, dal confronto dei tempi, si evince che impostare $t = \lceil \log n \rceil$ aumenta sensibilmente l'efficienza di RSVD. Tuttavia, occorre considerare che ridurre la soglia può comportare un incremento dell'errore; sarà quindi necessario valutare se tale incremento sia accettabile o se comprometta l'integrità dei dati. L'errore relativo medio è pari a 0,1803 per TSVD, con un incremento del 206%, e a 0,2520 per RSVD, con un aumento del 166%. L'errore massimo per TSVD raggiunge 0,2345 (incremento del 178%) su 1800 colonne, mentre per RSVD l'errore massimo è 0,3333 (incremento del 142%) su 1920 colonne. I valori minimi si registrano con 120 colonne per entrambi gli algoritmi, con 0,1075 per l'algoritmo deterministico (aumento del 144%) e 0,1838 per l'algoritmo randomizzato (aumento del 164%). Anche in questo caso si osservano punti di flesso attorno alle 4000 e 7000 colonne, evidenziando una similarità nell'andamento dell'errore relativo tra i due test. Questo comportamento può essere spiegato, analogamente al test precedente, analizzando il rapporto percentuale tra la somma troncata e quella totale dei valori singolari (Figura 4.22). Nella fascia compresa tra 120 e 1680 colonne, il rapporto dimi-

⁴Il troncamento avviene dopo aver eseguito la SVD classica sull'intera matrice in input.



nuisce dal 64% al 30%, evidenziando un calo decisamente più marcato rispetto a quello registrato nel test precedente. Successivamente, la percentuale si stabilizza, con variazioni trascurabili sia in aumento sia in diminuzione. A partire dalle 8760 colonne, si nota un nuovo decremento che culmina in un minimo di circa 27%.



La differenza media degli errori tra i due algoritmi è pari a 0,0717 (incremento del 102%), con una differenza massima di 0,1201 (incremento del 124%) su 720 colonne e una differenza minima di 0,0482 (incremento dell'87,5%) su 10320 colonne. La Figura 4.21 mostra la differenza degli errori come percentuale rispetto agli errori di RSVD.

In sintesi, l'adozione di un *threshold* logaritmico comporta un incremento delle prestazioni del 75% per RSVD, ma implica un raddoppio (almeno) dell'errore, che va considerato nella valutazione della qualità dei risultati. L'entità degli errori può essere visualizzata confronto le versioni dell'immagine in input (Figura 4.15, 4.16 e 4.17). Come si può notare, essa diventa abbastanza irriconoscibile dopo

RSVD.

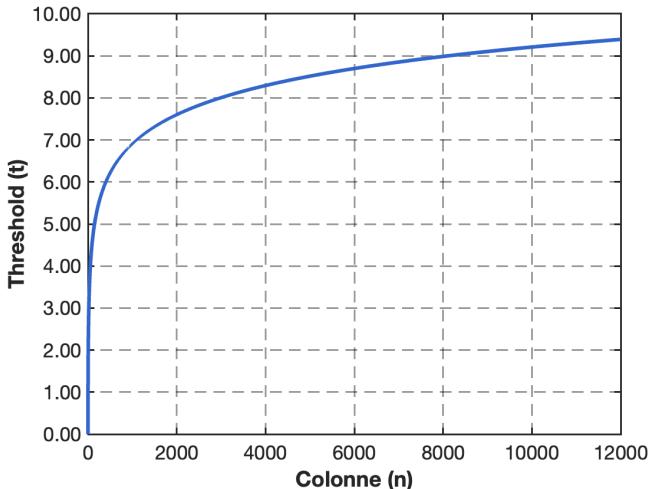


Figura 4.23: Grafico analitico del *threshold* $t = \log n$.

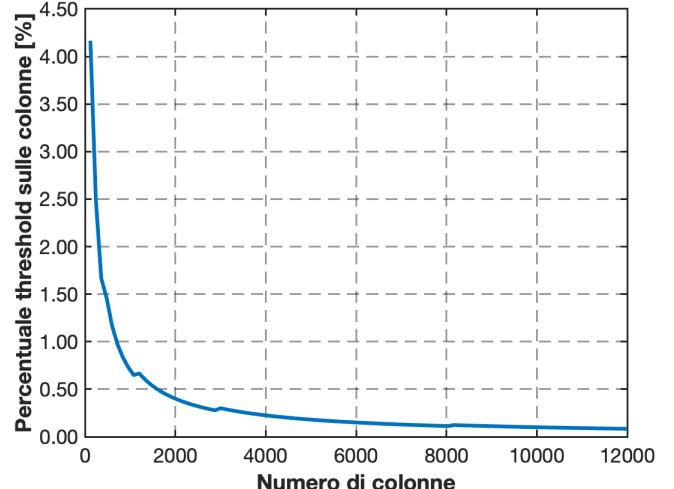


Figura 4.24: $Threshold t = \log n$ in percentuale sul numero di colonne del test.

	Tempo vs Errore	Tempo vs Threshold	Errore vs Threshold
Correlazione	-0,7256	0,7601	-0,4619

Tabella 4.7: Correlazioni lineari tra tempo di esecuzione, errore relativo e *threshold*.

	Tempo medio [s]		Tempo massimo [s]		Tempo minimo [s]	
	Valore	Colonne	Valore	Colonne	Valore	Colonne
TSVD	88,5171		284,2675		12000	0,0841
RSVD	0,1777		0,7556		11760	0,0048

Tabella 4.8: Tempi di esecuzione di TSVD e RSVD, con numero di colonne corrispondenti.

	Errore medio		Errore massimo		Errore minimo	
	Valore	Colonne	Valore	Colonne	Valore	Colonne
TSVD	0,1803		0,2345		1800	0,1075
RSVD	0,2520		0,3333		1920	0,1838

Tabella 4.9: Errori relativi di TSVD e RSVD, con numero di colonne corrispondenti.

	Media		Massima		Minima	
	Valore	Colonne	Valore	Colonne	Valore	Colonne
Differenza tempi [s]	88,105		297,206		12000	0,056
Differenza errori	0,0717		0,1201		720	0,0482

Tabella 4.10: Differenze nei tempi di esecuzione e negli errori di TSVD e RSVD, con numero di colonne corrispondenti.

Medio	Massimo		Minimo	
	Valore	Colonne	Valore	Colonne
<i>Threshold</i>	9	10	8160	5
				120

Tabella 4.11: *Threshold* di TSVD e RSVD, con numero di colonne corrispondenti.

5

Conclusioni

L'algoritmo randomizzato per la Singular Value Decomposition (RSVD) si è dimostrato una soluzione estremamente efficace nel ridurre i limiti computazionali della SVD classica, offrendo un metodo rapido e flessibile per analisi su larga scala. Grazie all'utilizzo di proiezioni casuali, l'algoritmo consente di preservare, con alta probabilità, le informazioni essenziali della matrice, permettendo una riduzione dimensionale significativa senza compromettere in modo critico l'accuratezza dei risultati. Questo lo rende particolarmente indicato in ambiti applicativi in cui è fondamentale il bilanciamento tra velocità e precisione.

Uno degli aspetti più importanti dell'approccio randomizzato è la sua scalabilità. In scenari in cui i dati crescono in modo esponenziale, come nel caso dei big data, dell'elaborazione di immagini o dei sistemi di raccomandazione, la RSVD si pone come uno strumento indispensabile. La riduzione della complessità computazionale da $\mathcal{O}(mn^2)$ a $\mathcal{O}(mnt)$, dove $t \ll n$, rappresenta una svolta per analizzare matrici di dimensioni, non gestibili con i metodi tradizionali. A ciò si aggiunge la possibilità di modulare il parametro t per personalizzare l'algoritmo in funzione delle necessità specifiche, garantendo un controllo preciso sul compromesso tra tempo di calcolo ed errore di approssimazione.

I risultati sperimentali, riassunti nella Tabella 5.1, hanno confermato l'efficacia del metodo, mostrando come soglie diverse influenzino le performance dell'algoritmo. L'adozione di una soglia di troncamento, come quella proposta da M. Gavish e D. L. Donoho [6], ha permesso di ottenere una riduzione del tempo di esecuzione di oltre due ordini di grandezza rispetto alla SVD deterministica, mantenendo un errore di approssimazione moderato. Parallelamente, l'uso di soglie logaritmiche, pur comportando un aumento degli errori relativi, ha permesso di accelerare ulteriormente i tempi di calcolo.

Un ulteriore vantaggio dell'RSVD è la sua semplicità di implementazione e la compatibilità con le architetture computazionali moderne. Grazie alla natura casuale delle proiezioni, il metodo si presta a essere combinato con tecniche di compressione e apprendimento automatico, aprendo prospettive interessanti per la ricerca e l'innovazione.

Le implicazioni applicative di questo approccio sono molteplici. Nel machine learning, l'RSVD può essere utilizzato per ridurre la dimensionalità dei dati di input, migliorando la velocità di algoritmi come PCA, regressione o *clustering*. Nel campo dell'elaborazione del linguaggio naturale, può accelerare la decomposizione di matrici di *embedding*, fondamentali per tecniche come la riduzione di dimensionalità in *word embedding*. In ambito scientifico e ingegneristico, l'RSVD consente di analizzare rapidamente

Algoritmo	Soglia t	Tempo [s]	Accelerazione	Errore	Discrepanza
TSVD	$\mathcal{O}(\sqrt{n})$	88,870873	1x	0,058845	1 x
TSVD	$\lceil \log n \rceil$	88,5171	1x	0,1803	3 x
RSVD	$\mathcal{O}(\sqrt{n})$	0,765446	116x	0,094308	1,6x
RSVD	$\lceil \log n \rceil$	0,1777	500x	0,2520	4,3x

Tabella 5.1: Confronto tra i tempi di esecuzione, gli errori relativi e l’accelerazione per diverse soglie nell’algoritmo randomizzato per la SVD. Accelerazione e discrepanza sono riferite a TSVD con *threshold* ottimale. [6]

sistemi complessi, come matrici di trasferimento, risolvendo problemi che richiederebbero altrimenti risorse computazionali proibitive.

Un altro tema rilevante è la robustezza statistica dell’RSVD. Le proiezioni casuali, pur essendo intrinsecamente approssimative, si basano su principi probabilistici che garantiscono con alta probabilità il mantenimento delle caratteristiche principali della matrice. Questo apre la strada a studi futuri per migliorare ulteriormente la qualità dell’approssimazione, ad esempio sviluppando tecniche ibride che combinino il *random sampling* con approcci deterministici in fasi critiche dell’elaborazione.

Infine, dal punto di vista teorico, i risultati di questa tesi confermano l’importanza della soglia t nella progettazione di algoritmi randomizzati e suggeriscono che ulteriori approfondimenti possano portare a una comprensione ancora più approfondita delle soglie ottimali per specifici contesti applicativi. In questo senso, la RSVD non solo rappresenta uno strumento pratico, ma costituisce anche una base per indagini teoriche sul comportamento di algoritmi probabilistici per la riduzione dimensionale.

In prospettiva, l’adozione crescente di metodi randomizzati nella scienza computazionale promette di rivoluzionare il modo in cui affrontiamo problemi complessi. La RSVD, con la sua combinazione di efficienza computazionale, flessibilità e robustezza, è destinato a giocare un ruolo centrale nello sviluppo di tecnologie future per l’analisi dei dati, l’apprendimento automatico e la risoluzione di problemi scientifici e ingegneristici su larga scala.

A

Codice Matlab

In questo capitolo riportiamo il codice Matlab utilizzato per eseguire i test, salvare i dati e generare i grafici.

A.1 Calcolo del *threshold* ottimale

In questa tesi, per calcolare il *threshold* ottimale, abbiamo approssimato per eccesso i valori restituiti dall'espressione $\lambda(\beta)\sqrt{n}\gamma$ utilizzando la funzione `ceil`. Nella riga 14, è possibile sostituire la funzione `ceil` con `floor` se si desidera approssimare per difetto, o con `round` se si preferisce arrotondare all'intero più vicino.

```
1 % Funzione per calcolare il rango target.  
2 % Input:  
3 %     numRows > 0,  
4 %     numCols > 0,  
5 %     gamma: magnitudine del rumore bianco, gamma > 0.  
6 % Output:  
7 %     t: threshold, t <= n.  
8 function t = optimalThreshold(numRows, numCols, gamma)  
9     % Il threshold non viene definito per matrici degeneri.  
10    assert(all(numRows>0));  
11    assert(all(numCols>0));  
12    beta = min(numRows/numCols, numCols/numRows);  
13    lambda = sqrt(2*(beta+1)+8*beta/(beta+1+sqrt(beta^2+14*beta+1)));  
14    t = min([ceil(lambda * sqrt(numCols) * gamma), numCols]);  
15 end
```

A.2 Randomized SVD

Il codice sottostante funziona solo per matrici alte e strette ($m \geq n$). L'algoritmo può essere facilmente modificato per adattarlo al caso duale ($n \geq m$), ma in questa tesi non ce ne occupiamo.

```

1 % Funzione per il calcolo randomizzato di svd.
2 % Input:
3 %     X: matrice m x n,
4 %     t: threshold, t <= n <= m.
5 % Output:
6 %     U_approx: matrice m x t,
7 %     S: matrice t x t,
8 %     U_approx: matrice t x n.
9 function [U_approx,S,V] = rsvd(X,t)
10 % O. Controllo input.
11 [m,n] = size(X); % Dimensioni di X.
12 assert((t <= n) && (n <= m));
13
14 % 1. Proiezione Z
15 P = randn(n,t); % Matrice di proiezione casuale.
16 Z = X*P; % Proiezione di X sullo spazio di P.
17
18 % 2. Decomposizione QR di Z.
19 [Q,R] = qr(Z,0);
20
21 % 3. Proiezione ortogonale Y.
22 Y = Q'*X;
23
24 % 4. SVD di Y.
25 [U,S,V] = svd(Y,'econ');
26
27 % 5. Ricostruzione approssimata dei modi di U.
28 U_approx = Q*UY;
29 end

```

A.3 Test singolo su un'immagine

Nel seguente codice è stato utilizzato il *threshold* ottimale suggerito da M. Gavish e D. L. Donoho [6], calcolato con la funzione precedente. Per utilizzare un altro *threshold* (come ad esempio quello logaritmico) si modifichi opportunamente la riga 19.

```

1 % Test per il calcolo di TSVD e RSVD di un'immagine.
2 % Mostra il grafico dei valori singolari.
3 % Mostra immagine originale, dopo TSVD e dopo RSVD.
4 % Mostra il threshold utilizzato e il suo rapporto rispetto alle
   colonne.
5 % Mostra tempo ed errore di TSVD e RSVD.

```

```

6 % Salva i grafici e i dati sopraelencati.
7 clear all; close all; clc;
8
9 % Parametri generali dell'esperimento.
10 name = 'cat'; % Nome dell'esperimento.
11 imgName = 'cat';
12 imgExtension = '.jpg';
13 A = imread(['Sample Images/', imgName, imgExtension]);
14 X = double(rgb2gray(A)); % Matrice associata all'immagine.
15 [m, n] = size(X); % Dimensioni della matrice.
16
17 %% Calcolo del threshold.
18 gamma = 1; % Magnitudine del rumore bianco.
19 t = optimalThreshold(m, n, gamma); % Rango target.
20
21 %% SVD troncata.
22 tic; % Inizia a misurare il tempo.
23 [U, S, V] = svd(X, 'econ');
24 timeTSVD = toc; % Salva il tempo impiegato.
25 % Ricostruzione da SVD troncata: i min servono per gestire matrici
26 % piccole.
26 XTSVD = U(:, 1:min(t, size(U, 2))) * S(1:min(t, size(S, 1)), 1:min(t,
27 % size(S, 2))) * V(:, 1:min(t, size(V, 2)))';
27 errTSVD = norm(X - XTSVD, 2) / norm(X, 2); % Errore relativo della SVD
28 troncata.
29
30 %% SVD randomizzata.
30 tic; % Inizia a misurare il tempo.
31 [rU, rS, rV] = rsvd(X, t); % SVD randomizzata.
32 timerSVD = toc; % Salva il tempo impiegato.
33 XRSVD = rU * rS * rV'; % Ricostruzione da SVD randomizzata.
34 errRSVD = norm(X - XRSVD, 2) / norm(X, 2); % Errore relativo della SVD
35 randomizzata.
36 %% Visualizzazione delle immagini.
37
38 % Immagine originale.
39 figure(1);
40 imshow(uint8(X)); % Mostra immagine originale.
41 exportgraphics(gcf, ['Images/', name, '_SVD.png']);
42

```

```
43 % Immagine dopo la SVD troncata.  
44 figure(2);  
45 imshow(uint8(XTSVD)); % Mostra l'immagine approssimata con SVD troncata  
  
46 exportgraphics(gcf, ['Images/', name, '_TSVD.png']);  
47  
48 % Immagine dopo la SVD randomizzata.  
49 figure(3);  
50 imshow(uint8(XRSVD)); % Mostra l'immagine approssimata con SVD  
randomizzata.  
51 exportgraphics(gcf, ['Images/', name, '_RSVD.png']);  
52  
53 %% Visualizzazione del grafico dei valori singolari (in percentuale).  
54  
55 % Somma dei valori singolari in percentuale.  
56 singularValues = diag(S); % Estrai i valori singolari dalla matrice S.  
57 totalSingularValueSum = sum(singularValues); % Somma totale dei valori  
singolari.  
58  
59 figure(4); clf; % Crea una nuova figura e pulisci il contenuto  
precedente.  
60  
61 x = linspace(1, min(m,n), min(m,n));  
62 y = singularValues / totalSingularValueSum;  
63 plot(x, y, 'o', 'MarkerFaceColor', 'b', 'LineWidth', 1, 'MarkerSize',  
10);  
64  
65 % Formatta il grafico.  
66 formatGraph('Indice del valore singolare', 'Valore singolare [%]);  
67  
68 % Regola gli assi.  
69 xlim([1, min(m,n)]);  
70 set(gca, 'YScale', 'log'); % Imposta l'asse y in scala logaritmica.  
71  
72 % Salva la figura.  
73 exportgraphics(gcf, ['Images/', name, '_singular_values.png']);  
74  
75 %% Output del threshold utilizzato, della somma dei primi t valori  
singolari, degli errori e dei tempi d'esecuzione.  
76  
77 % Mostra il threshold utilizzato.
```

```

78 sumFirstTSingularValues = sum(singularValues(1:t)); % Somma dei primi t
    valori singolari
79 percentageFirstTSingularValues = (sumFirstTSingularValues /
    totalSingularValueSum) * 100;
80 disp(['Threshold: ', num2str(t)]);
81
82 % Stampa la percentuale.
83 disp(['Somma dei primi t valori singolari in percentuale: ', num2str(
    percentageFirstTSingularValues), '%']);
84
85 % Mostra tempi d'esecuzione.
86 disp(['Tempo SVD troncata: ', num2str(timerSVD)]);
87 disp(['Tempo SVD randomizzata: ', num2str(timerRSVD)]);
88
89 % Mostra errori relativi.
90 disp(['Errore SVD troncata: ', num2str(errTSVD)]);
91 disp(['Errore SVD randomizzata: ', num2str(errRSVD)]);
92
93 %% Salva i dati.
94 save(['Data/', name, '_test.mat']);

```

A.4 Test iterativo sulle colonne di un'immagine

Anche nel seguente codice è stato utilizzato il *threshold* ottimale suggerito da M. Gavish e D. L. Donoho [6]. Per utilizzare un altro *threshold* (come ad esempio quello logaritmico) si modifichi opportunamente la riga 114.

```

1 % Test per il calcolo di TSVD e RSVD di un'immagine al variare delle
    colonne.
2 % Mostra i seguenti grafici:
3 %     tempi, errori, threshold, ratio dei tempi,
4 %     differenze tempi tra TSVD e RSVD,
5 %     differenze errori tra TSVD e RSVD,
6 %     percentuale somma troncata e somma totale dei valori singolari,
7 %     ratio tra threshold e colonne.
8 % Stampa i seguenti risultati:
9 %     correlazioni,
10 %     differenza tempi media, massima, minima tra TSVD e RSVD,
11 %     differenza errori media, massima, minima tra TSVD e RSVD,
12 %     tempi medi, massimi, minimi di TSVD e RSVD,
13 %     errori medi, massimi, minimi di TSVD e RSVD,
14 %     threshold medio, massimo e minimo.

```

```
15 % Salva i grafici e i dati sopraelencati.  
16 clear all, close all, clc  
17  
18 % Parametri generali dell'esperimento.  
19 name = 'nebula'; % Nome dell'esperimento.  
20 imgName = 'nebula';  
21 imgExtension = '.jpg';  
22 A = imread(['Sample Images/', imgName, imgExtension]); % Immagine demo.  
23 X = double(rgb2gray(A)); % Matrice associata all'immagine.  
24 [m,n] = size(X); % Dimensioni dell'immagine.  
25 gamma = 1; % Magnitudine del rumore bianco.  
26  
27 % Passi personalizzabili.  
28 step = 120; % Incremento per le colonne.  
29  
30 %% Funzione per calcolare la correlazione di Pearson tra due vettori.  
31 function r = pearsonCorr(x, y)  
32     assert(length(x) == length(y), 'I vettori devono avere la stessa  
         lunghezza.');//  
33  
34     % Calcola la media dei vettori.  
35     meanX = mean(x);  
36     meanY = mean(y);  
37  
38     % Calcola il numeratore e denominatore della correlazione di  
         Pearson.  
39     numerator = sum((x - meanX) .* (y - meanY));  
40     denominator = sqrt(sum((x - meanX).^2) * sum((y - meanY).^2));  
41  
42     % Correlazione.  
43     r = numerator / denominator;  
44 end  
45  
46 %% Funzione per eseguire la SVD troncata e calcolare il tempo ed errore  
.  
47 function [timeTSVD, errorTSVD] = execTSVD(XDynamic, t)  
48     tic;  
49     [U, S, V] = svd(XDynamic, 'econ');  
50     timeTSVD = toc;  
51     % Ricostruzione da SVD troncata: i min servono per gestire matrici  
         piccole.
```

```

52 XTSVD = U(:, 1:min(t, size(U, 2))) * S(1:min(t, size(S, 1)), 1:min(
53     t, size(S, 2))) * V(:, 1:min(t, size(V, 2)))';
54 errorTSVD = norm(XDynamic - XTSVD, 'fro') / norm(XDynamic, 'fro');
55 end
56 %% Funzione per eseguire la SVD randomizzata e calcolare il tempo ed
57 %% errore.
58 function [timeRSVD, errorRSVD] = execRSVD(XDynamic, t, p)
59 tic;
60 [rU, rS, rV] = rsvd(XDynamic, t);
61 timeRSVD = toc;
62 XRSVD = rU * rS * rV';
63 errorRSVD = norm(XDynamic - XRSVD, 'fro') / norm(XDynamic, 'fro');
64 end
65 %% Funzione per plottare i risultati e salvarli.
66 function plotRisultati(xValues, sumsRatio, timeTSVD, timeRSVD,
67 errorTSVD, errorRSVD, thresholds, name)
68 xlabelText = 'Numero di colonne';
69 % Plot tempo di esecuzione
70 figure;
71 plot(xValues, [timeTSVD, timeRSVD], 'LineWidth', 3);
72 legend('TSVD', 'RSVD', 'FontSize', 10);
73 formatGraph(xlabelText, "Tempo di esecuzione [s]");
74 exportgraphics(gcf, ['Images/', name, '_', '_times.png']);
75 % Plot errore relativo
76 figure;
77 plot(xValues, [errorTSVD, errorRSVD], 'LineWidth', 3);
78 legend('TSVD', 'RSVD', 'FontSize', 10);
79 formatGraph(xlabelText, "Errore relativo");
80 exportgraphics(gcf, ['Images/', name, '_', '_errors.png']);
81 % Plot threshold
82 figure;
83 plot(xValues, thresholds, 'LineWidth', 3);
84 formatGraph(xlabelText, "Threshold");
85 exportgraphics(gcf, ['Images/', name, '_', '_thresholds.png']);
86 % Plot somma troncata / somma totale
87

```

```

90 figure;
91 plot(xValues, sumsRatio * 100, 'LineWidth', 3);
92 formatGraph(xlabelText, 'Somma Troncata / Somma Totale [%]');
93 exportgraphics(gcf, ['Images/', name, '_',
94                 singular_values_sums_perc.png]);
95
96 % Rapporto dei tempi.
97 figure;
98 plot(xValues, timesTSVD./timesRSVD, 'LineWidth', 3);
99 formatGraph('Numero di colonne','Rapporto tra i tempi')
100 exportgraphics(gcf, ['Images/', name, '_time_ratio.png']);
101 end
102 %% Test: Tenere fisso il numero di righe e variare il numero di colonne
103
104 numTests = ceil(n / step);
105 timesTSVD = zeros(numTests, 1);
106 timesRSVD = zeros(numTests, 1);
107 errorsTSVD = zeros(numTests, 1);
108 errorsRSVD = zeros(numTests, 1);
109 thresholds = zeros(numTests, 1);
110 sumsRatio = zeros(numTests, 1);
111 singularValues = cell(numTests, 1); % Per salvare i valori singolari.
112
113 index = 1;
114 for numCols = step:step:n
115     t = optimalThreshold(m, numCols, gamma);
116     thresholds(index) = t;
117     XDynamic = X(:, 1:numCols);
118
119     % Calcolo SVD troncata e randomizzata.
120     [timesTSVD(index), errorsTSVD(index)] = execTSVD(XDynamic, t);
121     [timesRSVD(index), errorsRSVD(index)] = execRSVD(XDynamic, t, gamma
122 );
123
124     % Calcolo valori singolari.
125     s = svd(XDynamic, 'econ');
126     singularValues{index} = s;
127
128     % Calcolo rapporto somma valori singolari troncata / totale.
129     sumsRatio(index) = sum(s(1:t)) / sum(s);

```

```

128
129     index = index + 1;
130 end
131
132 xValues = step:step:n;
133
134 %% Plot dei risultati
135 plotRisultati(xValues, sumsRatio, timesTSVD, timesRSVD, errorsTSVD,
    errorsRSVD, thresholds, name);
136
137 %% Differenza percentuale nel tempo di esecuzione tra TSVD e RSVD.
138 deltaTimes = timesTSVD - timesRSVD;
139 deltaPercTimes = deltaTimes ./ timeTSVDs * 100;
140 figure;
141 plot(xValues, deltaPercTimes, 'LineWidth', 3);
142 formatGraph('Numero di colonne','Differenza percentuale nel tempo [%]')
143 exportgraphics(gcf, ['Images/', name, '_delta_times.png']);
144
145 %% Differenza percentuale nell'errore tra RSVD e TSVD.
146 deltaErrors = errorsRSVD - errorsTSVD;
147 deltaPercErrors = deltaErrors ./ errorsRSVD * 100;
148 figure;
149 plot(xValues, deltaPercErrors, 'LineWidth', 3);
150 formatGraph('Numero di colonne',"Differenza percentuale nell'errore [%"]
    ])
151 exportgraphics(gcf, ['Images/', name, '_delta_errors.png']);
152
153 %% Percentuale threshold sul numero di colonne.
154 percThresholds = thresholds ./ xValues' * 100;
155 figure;
156 plot(xValues, percThresholds, 'LineWidth', 3);
157 formatGraph('Numero di colonne','Percentuale threshold sulle colonne'
    [%])
158 exportgraphics(gcf, ['Images/', name, 'thresholds_perc.png']);
159
160 %% Calcolo delle correlazioni.
161 corrTimesErrors = pearsonCorr(timesTSVD, errorsTSVD);
162 corrTimesThresholds = pearsonCorr(timesTSVD, thresholds);
163 corrErrorsThresholds = pearsonCorr(errorsTSVD, thresholds);
164 fprintf('Correlazione tra tempo ed errore: %f.\n', corrTimesErrors);

```

```
165 fprintf('Correlazione tra tempo e threshold: %f.\n',
    corrTimesThresholds);
166 fprintf('Correlazione tra errore e threshold: %f.\n',
    corrErrorsThresholds);
167
168 %% Differenza media, minima e massima dei tempi tra TSVD e RSVD.
169 meanDeltaTime = mean(deltaTimes);
170 [maxDeltaTime, idxMaxTime] = max(deltaTimes);
171 [minDeltaTime, idxMinTime] = min(deltaTimes);
172 colMaxTime = xValues(idxMaxTime);
173 colMinTime = xValues(idxMinTime);
174 fprintf('Differenza media dei tempi: %f secondi\n', meanDeltaTime);
175 fprintf('Differenza massima dei tempi: %f secondi, corrispondente a %d
    colonne.\n', maxDeltaTime, colMaxTime);
176 fprintf('Differenza minima dei tempi: %f secondi, corrispondente a %d
    colonne.\n', minDeltaTime, colMinTime);
177
178 %% Differenza media, minima e massima degli errori tra TSVD e RSVD.
179 meanDeltaError = mean(deltaErrors);
180 [maxDeltaError, idxMaxError] = max(deltaErrors);
181 [minDeltaError, idxMinError] = min(deltaErrors);
182 colMaxError = xValues(idxMaxError);
183 colMinError = xValues(idxMinError);
184 fprintf('Differenza media degli errori: %f\n', meanDeltaError);
185 fprintf('Differenza massima degli errori: %f, corrispondente a %d
    colonne.\n', maxDeltaError, colMaxError);
186 fprintf('Differenza minima degli errori: %f, corrispondente a %d
    colonne.\n', minDeltaErrors, colMinError);
187
188 %% Tempi medi, minimi e massimi.
189 meanTimeTSVD = mean(timesTSVD);
190 meanTimeRSVD = mean(timesRSVD);
191 [maxTimeTSVD, idxMaxTimeTSVD] = max(timesTSVD);
192 [maxTimeRSVD, idxMaxTimeRSVD] = max(timesRSVD);
193 colMaxTimeTSVD = xValues(idxMaxTimeTSVD);
194 colMaxTimeRSVD = xValues(idxMaxTimeRSVD);
195 [minTimeTSVD, idxMinTimeTSVD] = min(timesTSVD);
196 [minTimeRSVD, idxMinTimeRSVD] = min(timesRSVD);
197 colMinTimeTSVD = xValues(idxMinTimeTSVD);
198 colMinTimeRSVD = xValues(idxMinTimeRSVD);
```

```

199 fprintf('Media dei tempi: TSVD: %f secondi; RSVD: %f secondi\n',
200   meanTimeTSVD, meanTimeRSVD);
201 fprintf('Tempo massimo: TSVD: %f secondi, corrispondente a %d colonne;
202   RSVD: %f secondi, corrispondente a %d colonne.\n', maxTimeTSVD,
203   colMaxTimeTSVD, maxTimeRSVD, colMaxTimeRSVD);
204 fprintf('Tempo minimo: TSVD: %f secondi, corrispondente a %d colonne;
205   RSVD: %f secondi, corrispondente a %d colonne.\n', minTimeTSVD,
206   colMinTimeTSVD, minTimeRSVD, colMinTimeRSVD);

207 %% Errori medi, minimi e massimi.
208 meanErrorTSVD = mean(errorsTSVD);
209 meanErrorRSVD = mean(errorsRSVD);
210 [maxErrorTSVD, idxMaxErrorTSVD] = max(errorsTSVD);
211 [maxErrorRSVD, idxMaxErrorRSVD] = max(errorsRSVD);
212 colMaxErrorTSVD = xValues(idxMaxErrorTSVD);
213 colMaxErrorRSVD = xValues(idxMaxErrorRSVD);
214 fprintf('Media degli errori: TSVD: %f; RSVD: %f\n', meanErrorTSVD,
215   meanErrorRSVD);
216 fprintf('Errore massimo: TSVD: %f, corrispondente a %d colonne; RSVD: %f,
217   corrispondente a %d colonne.\n', maxErrorTSVD, colMaxErrorTSVD,
218   maxErrorRSVD, colMaxErrorRSVD);
219 fprintf('Errore minimo: TSVD: %f, corrispondente a %d colonne; RSVD: %f,
220   corrispondente a %d colonne.\n', minErrorTSVD, colMinErrorTSVD,
221   minErrorRSVD, colMinErrorRSVD);

222 %% Threshold medio, minimo e massimo.
223 meanThreshold = mean(thresholds);
224 [maxThreshold, idxMaxThreshold] = max(thresholds);
225 colMaxThreshold = xValues(idxMaxThreshold);
226 [minThreshold, idxMinThreshold] = min(thresholds);
227 colMinThreshold = xValues(idxMinThreshold);
228 fprintf('Threshold medio: %f.\n', meanThreshold);
229 fprintf('Threshold massimo: %f, corrispondente a %d colonne.\n',
230   maxThreshold, colMaxThreshold);
231 fprintf('Threshold minimo: %f, corrispondente a %d colonne.\n',
232   minThreshold, colMinThreshold);

```

```

228 %% Salva i dati.
229 save(['Data/' , name , '.mat']);

```

A.5 Formattazione dei grafici

```

1 % Funzione per formattare in modo coerente i grafici.
2 % Input:
3 %     xlabel: titolo dell'asse x (string);
4 %     ylabel: titolo dell'asse y (string).
5 function formatGraph(xLabel, ylabel)
6     % Nomina gli assi.
7     xlabel(xLabel, 'FontSize', 18, 'FontWeight', 'bold');
8     ylabel(yLabel, 'FontSize', 18, 'FontWeight', 'bold');
9
10    % Imposta le caratteristiche degli assi.
11    set(gca, 'FontSize', 16, 'LineWidth', 1.5, 'Box', 'on', '
12        GridLineStyle', '--');
13
14    % Attiva la griglia.
15    grid on;
16    ax = gca;
17    ax.XGrid = 'on';
18    ax.YGrid = 'on';
19    ax.GridAlpha = 0.4;
20    ax.GridColor = [0, 0, 0];
21
22    % Ottieni i limiti correnti per l'asse y.
23    yLimits = get(gca, 'YLim');
24
25    % Controlla il limite massimo dell'asse y.
26    if yLimits(2) <= 10
27        % Imposta il formato delle etichette delle ordinate.
28        ytickformat('%.2f'); % Formato decimale con due posizioni.
29    end
29 end

```

Bibliografia

- [1] Steven L. Brunton e J. Nathan Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, USA, 1st edizione, 2019.
- [2] Alan Cline e Inderjit Dhillon. *Computation of the Singular Value Decomposition*, pp. 1027–1039. Chapman and Hall/CRC, 12 2013.
- [3] Petros Drineas e Michael W. Mahoney. Randnla: randomized numerical linear algebra. *Commun. ACM*, 59(6):80–90, maggio 2016.
- [4] Petros Drineas e Michael W. Mahoney. Lectures on randomized numerical linear algebra. *CoRR*, abs/1712.08880, 2017.
- [5] M. Gavish e D. L. Donoho. Code supplement to "the optimal hard threshold for singular values is $4/\sqrt{3}$ ", 2014. <https://purl.stanford.edu/vg705qn9070>.
- [6] Matan Gavish e David L. Donoho. The optimal hard threshold for singular values is $4/\sqrt{3}$. *IEEE Transactions on Information Theory*, 60(8):5040–5053, Aug 2014.
- [7] Mutual Information. Is the future of linear algebra.. random?, 2024. https://www.youtube.com/watch?v=6htbyY3rH1w&t=1134s&ab_channel=MutualInformation.