Progetto nr.4

Alyssa Pezzutti, Nicole Santero

Luglio, 2021

1 Esercizio 1

1.1 La funzione segment image

Input e output. La funzione `segment_image` prende in ingresso una matrice *img*, un intero k e altri parametri opzionali. k rappresenta il numero di cluster in cui vogliamo suddividere i dati. Per questo esercizio aggiungiamo in input i parametri *maxiter* e *stype*, dove *maxiter* fissa il numero massimo di iterazioni nei cicli `while` (su cui torneremo più avanti), mentre *stype* indica il tipo di "Start", ovvero il modo in cui vengono scelti i centri dei cluster di partenza. L'output della funzione è la matrice che rappresenta l'immagine quantizzata.

Corpo della funzione. `segment_image` si appoggia alla funzione `kmeans` di Matlab, che di fatto svolge il grosso del lavoro. Proveremo ora a entrare nel dettaglio di queste due funzioni: Una volta ricevuta l'immagine in input sotto forma di matrice di dimensioni $h \times w \times c$ (rispettivamente altezza e larghezza dell'immagine in pixel, e $c = 3$ corrispondente ai tre colori di base rosso, blu e verde), questa viene normalizzata e trasformata in una matrice bidimensionale $hw \times c$. La si passa ora a `kmeans`, insieme agli altri parametri $k, maxiter, stype$.

La funzione kmeans. Questa funzione esegue l'algoritmo di *k-means clustering* per suddividere le entrate della matrice in k cluster. Come punto di partenza, vengono scelti k punti in modo più o meno casuale che faranno da centroidi per il primo batch di cluster. Il metodo esatto per scegliere i punti è specificato dalla variabile *stype*, ed è anche possibile per l'utente passare delle coordinate scelte da lui. L'obiettivo dell'algoritmo è trovare una configurazione tale che la quantità

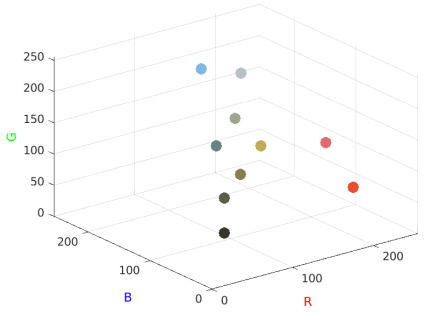
$$\sum_{i=1}^{k} \sum_{j=1}^{n(i)} ||x_j^i - c_i|| \tag{1}$$

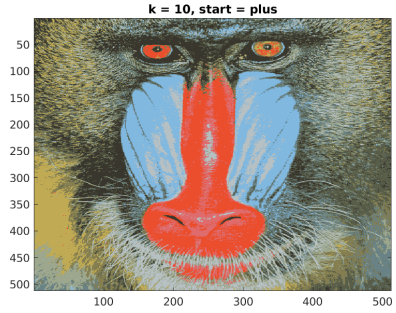
sia minima, dove c_i denota il centroide dell'i-esimo cluster, e x_j^i corrispondono agli elementi assegnati a quest'ultimo.

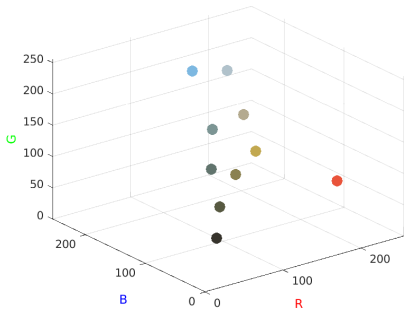
La funzione `kmeans` si sviluppa su due cicli `while`. Nel primo vengono eseguite due operazioni: fissati k centroidi si assegnano i punti al cluster del centroide più vicino, e poi si calcola il nuovo centroide di ciascun cluster. Non è però sempre possibile trovare una distribuzione ottimale dei cluster (in cui cioè esiste un'unica configurazione che minimizza (1)). Per questo si inserisce un secondo ciclo, in cui i punti vengono scorsi uno per volta, fino a raggiungere il numero massimo di iterazioni *maxiter*. Ciascun punto viene spostato in un altro cluster solo se questo porta a una riduzione di (1), per poi ricalcolare i due centroidi interessati (quelli "di partenza" e "di arrivo" del punto). Nella maggior parte dei casi, al termine dei due cicli si trova una configurazione buona ma non ottimale, dunque per migliorare il risultato si consiglia di rieseguire l'algoritmo varie volte usando diversi centroidi di partenza.
Ciascuna immagine viene rappresentata con 3 colori, in cui ciascuno può assumere 255 valori diversi. Dunque la dimensionalità del problema è 255^3.

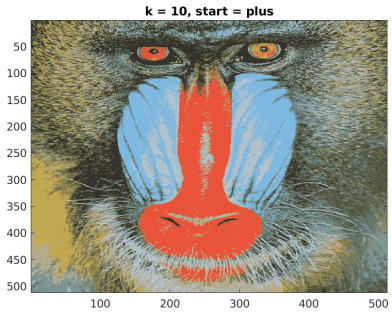
1.2

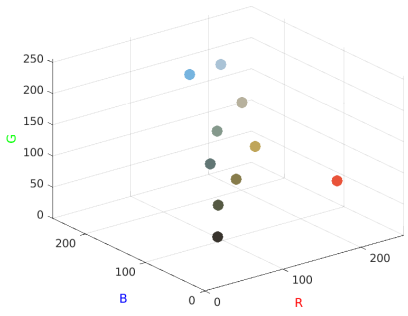
Definiamo una stringa *stype* per indicare il tipo di Start, e la passiamo a kmeans come input. Fissiamo *stype* = 'plus', cioè lo Start predefinito. In questa sezione proveremo ad eseguire più volte l'algoritmo per $k = 10$ e $k = 2$, per vedere se, a pari condizioni iniziali, i risultati dipendano dalla configurazione iniziale dei centroidi o meno.
Siano $k = 10$, $maxiter = 100$. Usando molti colori e permettendo un alto numero di iterazioni otteniamo risultati molto simili tra loro, dunque la configurazione iniziale influisce poco sull'output, come ci si poteva aspettare.
I k colori selezionati dal programma non sono mai identici, ma alla fine le immagini quantizzate sono pressoché indistinguibili. Alleghiamo le rappresentazioni dei colori usati e le corrispondenti immagini ottenute su 3 esecuzioni dell'algoritmo:

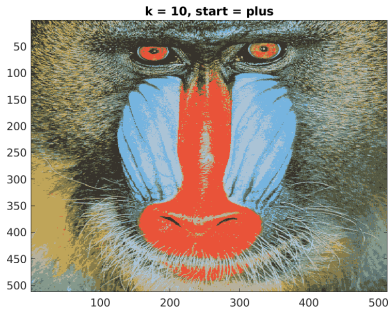




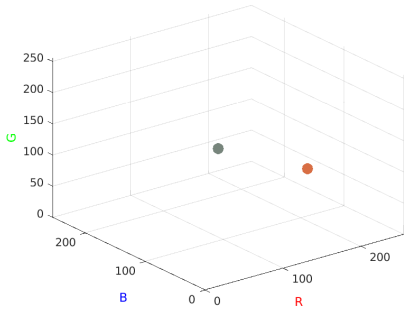


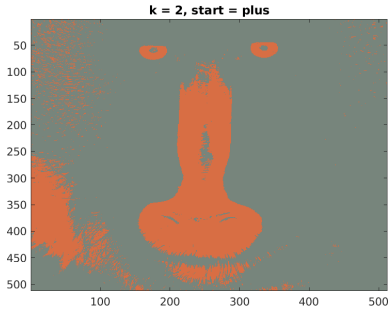


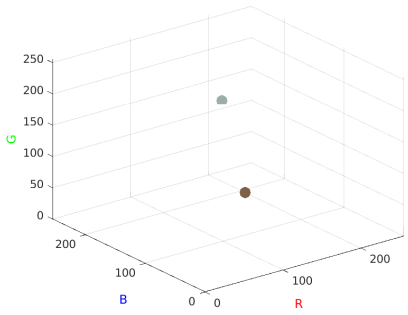


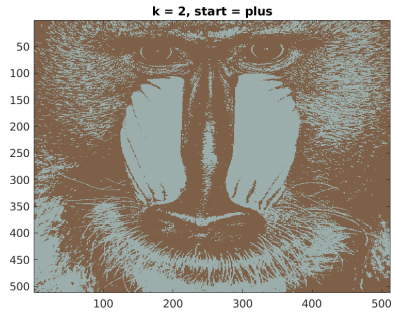


Prendendo un k molto basso invece, per esempio $k = 2$, l'output dipende fortemente dalla configurazione iniziale, che viene scelta in modo casuale. Otteniamo infatti risultati molto diversi tra loro, come si vede nelle figure seguenti:









Naturalmente l'avere così pochi colori non permette un rendering accurato dell'immagine, ma nella seconda figura abbiamo comunque ottenuto una silhouette fedele all'originale, che è un buon risultato.

1.3

La variabile k rappresenta il numero di colori da usare per rappresentare l'immagine quantizzata, dove ciascun colore è associato a un cluster. Quindi in generale aumentando k si ottiene una figura che approssima sempre meglio l'originale, e i risultati sono sicuramente migliori in termini di approssimazione. Per k bassi però ($k < 5$ per esempio), i risultati sono poco stabili nel senso che l'output dipende fortemente dalla configurazione iniziale scelta casualmente, quindi eseguendo l'algoritmo più volte si ottengono immagini con colori completamente diversi tra loro e dall'originale (come nelle figure sopra).
Il costo computazionale però aumenta rapidamente all'aumentare di k, perciò conviene cercare un valore di k che sia un buon compromesso.

1.4

Proviamo quindi a cercare un valore di k che sia un buon compromesso tra approssimazione e costo computazionale.
Sia `quants` la matrice $512^2 \times 3$ che rappresenta l'immagine quantizzata, e `imgs` la matrice delle stesse dimensioni che rappresenta l'immagine originale. Scorriamo gli elementi riga per riga, dove ciascuna riga contiene i 3 valori (normalizzati) che codificano ciascun colore. Supponiamo che la norma di ciascuna riga rappresenti "il colore della cella", cioè un valore di riferimento per poter confrontare quanto si discostano i colori approssimati da quelli originali. Per ogni valore di k quindi, calcoliamo la quantità:

$$\sum_{i=1}^{n} \frac{(||\texttt{quants(i,:)}|| - ||\texttt{imgs(i,:)}||)^2}{n} \tag{2}$$

Il grafico che rappresenta l'andamento dell'errore è il seguente:

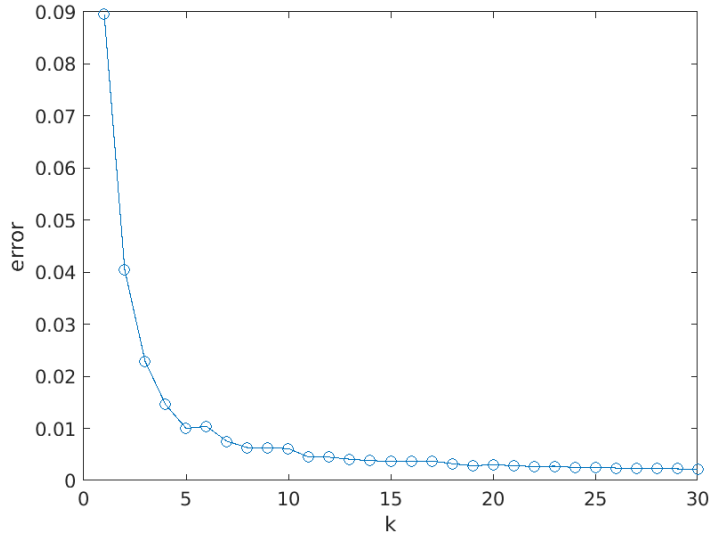
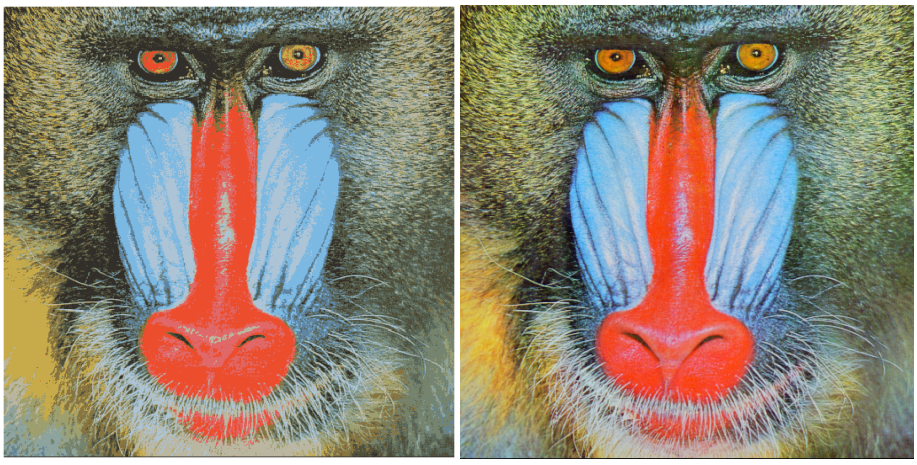


Figure 1: andamento dell'errore al variare di k $\in$ {1,..,30}

Questo grafico suggerisce di scegliere un k ottimale nell'insieme {13,..,16 }. Per questi valori infatti, l'errore è sufficientemente basso, e prendendo dei k maggiori non c'è una riduzione abbastanza significativa dell'errore che mitighi l'aumento del costo computazionale. Come si può vedere nella pagina seguente, le immagini quantizzate approssimano bene l'originale, anche se chiaramente usando solo circa 14 colori la quantizzazione non potrà riprodurre esattamente il livello di dettaglio dell'originale.



(a) Immagine quantizzata con k = 14 (b) Immagine originale

Figure 2: Confronto tra immagine quantizzata e originale

2 Esercizio 2

2.1

In questa seconda parte del progetto ci occupiamo del condizionamento di un problema. Consideriamo la matrice

$$K = \begin{pmatrix} 1 & 0 \\ 0 & 1/2^{10} \end{pmatrix}$$

e il vettore y=$(1,2^{-10})^t$. Il vettore x è la soluzione esatta del sistema Kx=y. Ora, consideriamo due diversi problemi perturbati:

- Primo problema perturbato:
  il termine noto è dato da $y_{p_1} = y + p_1$,con p_1 =$(0,2^{-10})^t$ e la soluzione relativa a questo problema è x_{p_1}.
- Secondo problema perturbato:
  il termine noto è dato da $y_{p_2} = y + p_2$,con p_2 =$(2^{-10},0)^t$ e la soluzione relativa a questo problema è x_{p_2}.

In entrambi i casi calcoliamo pure r=$\|x_{p_i} - x\|_2/\|p_i\|_2$, $i = 1, 2$.
Implementiamo in Matlab il seguente problema e otteniamo i seguenti risultati. La soluzione esatta del sistema Kx=y è x=$(1,1)^t$.
Consideriamo ora i due problemi perturbati singolarmente:

- Nel primo problema perturbato la soluzione è x_{p_1}=$(1,2)^t$ e r risulta essere uguale a 1024. Possiamo notare che la soluzione perturbata è molto diversa da quella esatta infatti, il secondo termine di x_{p_1} differisce di 1 dal secondo termine del vettore soluzione x. Anche r risulta essere un numero molto elevato. Queste due osservazioni sono dovute al fatto che il condizionamento della matrice è 1024, valore molto maggiore di 1, dunque il problema è mal condizionato. Un problema è mal condizionato quando le soluzioni sono molto sensibili a piccole perturbazioni dei dati iniziali. Il condizionamento viene calcolato come $\frac{|\lambda_{max}|}{|\lambda_{min}|}$, dove λ_{max} e λ_{min} sono rispettivamente il valore singolare più grande e quello più piccolo della matrice K. Essendo la matrice K simmetrica i valori singolari di K coincidono con gli autovalori della matrice K.
- Nel secondo caso, la soluzione perturbata è x_{p_2}=$(1.0009765625,1)^t$ che è molto simile alla soluzione esatta, infatti il primo termine del vettore della soluzione perturbata differisce dal primo termine della soluzione esatta di 0.0009765625 e i secondi termini sono entrambi uguali a 1. A differenza del caso precedente r è uguale a 1.

2.2

Consideriamo ora il problema "regolarizzato" con matrice K_α

$$K_\alpha = \begin{pmatrix} 1 & 0 \\ 0 & \frac{1}{(1-\alpha)^{10}}\frac{1}{2^{10}} \end{pmatrix}$$

con $0 < \alpha < 1$.
Prendiamo in esame i due diversi sistemi perturbati:

- Nel primo caso abbiamo il sistema $K_\alpha x_{p_1} = y_{p_1}$. La quantità r risulta essere minima quando α vale 0.067 e r in questo caso vale 0.36202. Per questo valore di α abbiamo che la soluzione perturbata vale $(1,0.99964646)^t$ che è molto simile alla soluzione esatta del problema.

- Nel secondo caso abbiamo il sistema $K_\alpha x_{p_2} = y_{p_2}$. Ora abbiamo che il valore minimo di r è 10.242973 e questo si verifica per $\alpha = 10^{-3}$. La soluzione del problema perturbato relativa a questo valore di α è $(1.0009766,0.990044880)^t$. Dal punto 1 di questo esercizio possiamo però notare che il sistema $K_\alpha x_{p_2} = y_{p_2}$ avente $\alpha = 0$ conduce ad una soluzione perturbata x_{p_2} molto più simile alla soluzione esatta. Quindi più α tende a zero e più r diminuisce.

Prendiamo in considerazione il numero di condizionamento della matrice K_α al variare di α. Possiamo notare che il numero di condizionamento è minimo per $\alpha = 0.5$, infatti per questo valore di α la matrice K_α è la matrice identità. Per gli α compresi tra 0 e 0.5 abbiamo che il numero di condizionamento rispetto ad α è una funzione monotona decresce, invece, per gli α appartenenti all'intervallo (0.5,1) la funzione è monotona crescente.
Nel caso del prima problema perturbato abbiamo che per α che rende minimo r il condizionamento della matrice K_α è 511.81.
Invece, nel secondo caso prendendo sempre α che minimizza r abbiamo che il numero di condizionamento della matrice K_α è 1013.805.
Possiamo notare che in entrambi i casi il condizionamento della matrice K_α è minore rispetto al condizionamento della matrice K.