

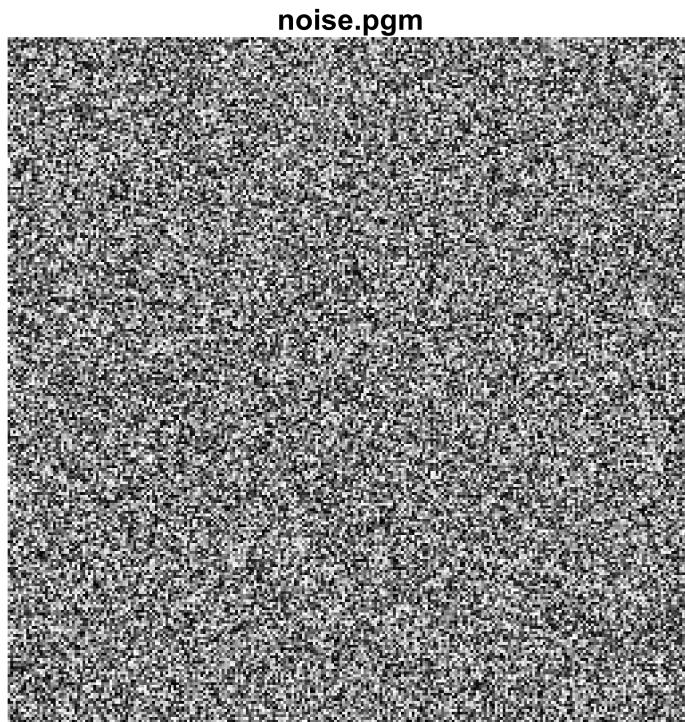
Homework 1

Francesco Roncolato - 2032442

Parte 1

1. Caricare un'immagine a livelli di grigio oppure a colori, ma in tal caso bisogna estrarne la componente di luminanza, che può essere approssimata come media delle componenti RGB.

```
close all; clear variables;
% Load the image
imageName = "noise";
fileName = sprintf('%s.pgm', imageName);
f = imread(fileName);
[row, col] = size(f);
nPixel = numel(f); % = row*col
% Show the image
figure; imagesc(f); colormap(gray); axis image; axis off; title(fileName);
```



- 2.1 Sia $f(n, m)$ l'immagine a livelli di grigio: stimarne l'entropia esprimendola in bit per pixel

```
tmp = transpose(f);
rasterScan = tmp(:); % Questo permette di leggere i pixel dell'immagine riga per
                     % riga
HX = hentropy(rasterScan); % Funzione definita in fondo al file
```

```
fprintf('L''entropia dell''immagine %s è HX = %5.4f bpp\n', imageName, HX);
```

L'entropia dell'immagine noise è HX = 7.9972 bpp

```
fprintf('Il rapporto di compressione ottenibile è di %5.4f\n', 8/HX);
```

Il rapporto di compressione ottenibile è di 1.0003

3. Utilizzare un'applicazione come zip in Windows oppure gzip in Linux e calcolare il bitrate risultante (dimensione del file in bit diviso numero di pixel)

```
cmd = sprintf('7z.exe a %s.zip %s.pgm > nul', imageName, imageName); % crea la  
stringa di comando  
% cmd = sprintf('tar.exe -a -cf %s.zip %s.pgm', imageName, imageName);  
% cmd = sprintf('zip %s %s.pgm', imageName, imageName);  
system(cmd); % la invia al SO per esecuzione  
info=dir(sprintf('%s.zip', imageName));  
nBytes = info.bytes; % recupera la dimensione del file ZIP  
zip_bpp = nBytes*8/nPixel; % dimensione espressa in bpp  
fprintf('Il file %s (7z.exe) ha un tasso di codifica %5.4f bpp\n', info.name,  
zip_bpp);
```

Il file noise.zip (7z.exe) ha un tasso di codifica 8.0204 bpp

4. Confrontare l'entropia ottenuta al punto 1 e il tasso ottenuto al punto 3. Discutere il risultato

La compressione eseguita dai software *zip cerca pattern di simboli comuni per riempire delle tabelle. Vengono quindi codificati insieme dei gruppi di pixel. Così facendo si può riformulare il teorema di Shannon dividendo tutti i membri per la lunghezza dei gruppi di simboli

$$H(X) \leq \mathcal{L}^* < H(x) + 1$$

$$\frac{H(X^K)}{K} \leq \frac{\mathcal{L}^*}{K} = \mathcal{L}_S^* < \frac{H(X^K)}{K} + \frac{1}{K}$$

Questo significa che all'aumentare della lunghezza K dei simboli, la lunghezza media ottima in bit per simbolo \mathcal{L}_S^* si avvicina sempre di più al tasso entropico H , secondo il teorema dei due carabinieri.

In altre parole, la lunghezza media delle parole di codice sarà sempre (anche nel file .zip) maggiore o uguale dell'entropia dell'immagine ma aumentando la lunghezza delle parole di codice si può raggiungere una codifica sempre migliore in termini di bit per pixel. Per questo $zip_bpp < HX$.

5. Effettuare la codifica predittiva “semplice”

5.1. L'immagine è rappresentata, dopo una scansione riga per riga (raster scan), su un vettore x

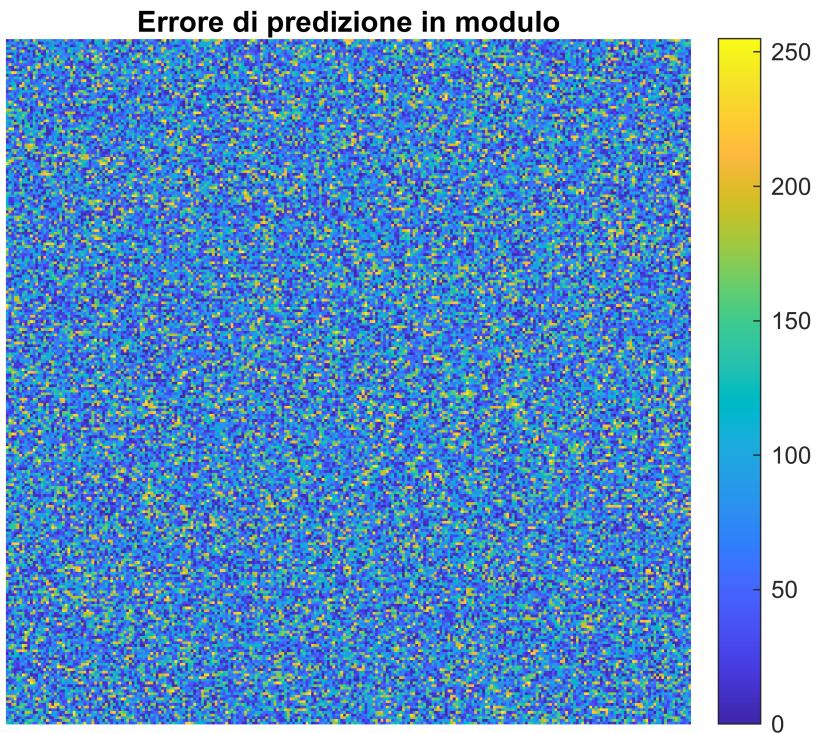
```
tmp = transpose(f);  
rasterScan = tmp(:);
```

5.2. La predizione di $x(n)$ è $x(n - 1)$, tranne per il primo pixel per il quale la predizione è 128

```
pred = [128; double(rasterScan(1:end-1))]; % nel creare la predizione si ignora  
l'ultimo pixel dell'immagine
```

5.3. L'errore di predizione è $y(n) = x(n) - x(n - 1)$, tranne per il primo pixel per il quale $y(0) = x(0) - 128$ (cioè $y(n)=x(n)-pred(n)$)

```
predErr = double(rasterScan)-pred;  
figure; imagesc(transpose(reshape(abs(predErr),row,col))); axis image; axis off;  
colorbar; title('Errore di predizione in modulo');  
saveas(gcf, sprintf('%sPredErr.png', imageName));
```



6. Stimare l'entropia dell'errore di predizione y

```
HY = hentropy(predErr);  
fprintf('L''entropia dell''errore di predizione per %s è HY = %5.4f bpp\n',  
imageName, HY);
```

L'entropia dell'errore di predizione per noise è HY = 8.7115 bpp

```
fprintf('Il rapporto di compressione ottenibile è di %5.4f\n', 8/HY);
```

Il rapporto di compressione ottenibile è di 0.9183

7. Valutare il numero di bit necessari per codificare l'errore di predizione y con la codifica *Exp Golomb con segno*, dedurne il bitrate di codifica e confrontare tale valore con quello ottenuto ai punti 1, 3 e 5

```
bitCount = 0;
```

```

for index = 1:numel(predErr)
    symbol = predErr(index);
    codeword = expGolombSigned(double(symbol));
    bitCount = bitCount + numel(codeword);
end
EG_bpp = bitCount/nPixel;
fprintf('Tasso di codifica S-EG su errore di predizione per %s: %5.4f bpp\n',
imageName, EG_bpp);

```

Tasso di codifica S-EG su errore di predizione per noise: 13.7085 bpp

La codifica lossless predittiva consiste nel formulare una predizione del simbolo i-esimo sulla base dei simboli precedenti*. Se questa predizione è "buona" allora l'errore di predizione avrà entropia piccola e in generale minore di quella della sorgente: $HY < HX$. Si può quindi pensare di codificare l'errore di predizione al posto dei simboli originali.

Se la predizione è buona l'errore è formato da valori piccoli diventa vantaggioso utilizzare una codifica come Exp Golomb, che assegna meno bit agli interi con modulo minore. Si può arrivare all'assurdo (si vedano i dati per vStripes al punto 8) dove l'entropia dell'errore di predizione è bassa (1,003 bpp) in quanto l'errore è costante, ma il modulo dell'errore è grande (sempre 255 tranne nel primo pixel dove è 128) e quindi la codifica Exp Golomb assegna 17 bit per pixel.

La predizione che si è fatta al punto 5 consiste nell'ipotizzare che ogni pixel sia uguale al suo precedente**. Questo può portare ad un "effetto di bordo" che consiste in un errore diverso, in ogni riga, tra il primo e gli altri pixel. Questo è dovuto al fatto che il pixel "precedente" al primo di una riga è l'ultimo della riga precedente, che si trova al lato opposto dell'immagine e che quindi potrebbe facilmente essere diverso, rendendo imprecisa la predizione.

Quando muovendosi da sinistra a destra i colori dei pixel cambiano poco (o non cambiano affatto), la predizione è buona e di conseguenza l'errore di predizione piccolo. La predizione così fatta premia quindi le immagini composte da linee orizzontali.

* È necessario che la predizione si basi solamente sui simboli precedenti perché altrimenti sarebbe impossibile decodificare l'immagine. Il decodificatore può partire dal primo pixel, supporre che il suo valore sia 128 e sommare l'errore per ottenere il valore corretto. Una volta noto il valore corretto del primo pixel si somma a questo l'errore del secondo pixel, e via così.

** Si noti che per trovare il pixel "precedente" si è trasposta e linearizzata l'immagine nell'array rasterScan in modo da avere una lista che itera i pixel dal primo in alto a sinistra, procedendo verso destra e andando a capo alla fine di una riga. Questa operazione risente fortemente dell'orientamento dell'immagine. Se l'immagine è composta perlopiù da strisce verticali si avranno continui cambi di colore e quindi l'errore di predizione sarà grande mentre se le stesse strisce fossero orizzontali si avrebbe un errore più piccolo.

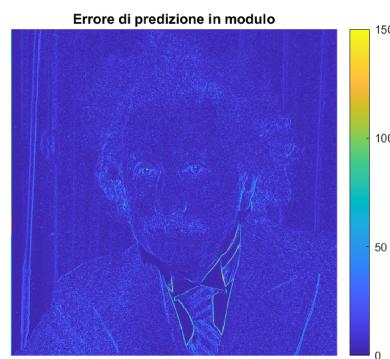
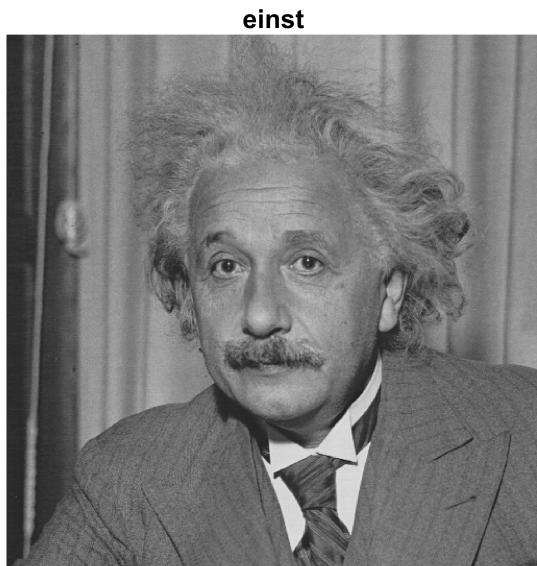
8. Ripetere gli esperimenti per più immagini e riportare i risultati. Commentare quanto trovato

Oltre alle immagini proposte, ho generato e testato la codifica sulle immagini (256x256) vStripes.pgm, hStripes.pgm, vShade.pgm, hShade.pgm e noise.pgm (rispettivamente composte da linee verticali, linee orizzontali, sfumatura verticale, sfumatura orizzontale e pixel generati randomicamente).

```

fileList = {'einst', 'house', 'lake', 'lena', 'peppers', 'plane', 'spring',
'veStripes', 'hStripes', 'vShade', 'hShade', 'noise'};
for i=1:numel(fileList)
    figure; tiledlayout(1,2,"TileSpacing", 'none', "Padding", 'none');
    nexttile; imshow(imread(sprintf("%s.pgm", fileList{i}))); title(fileList{i});
    nexttile; imshow(imread(sprintf("%sPredErr.png", fileList{i})));
end

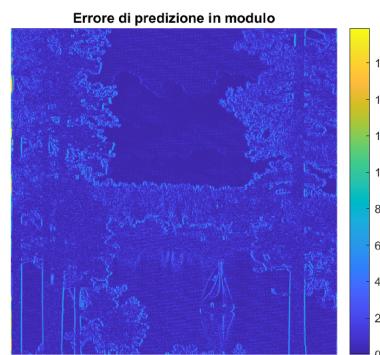
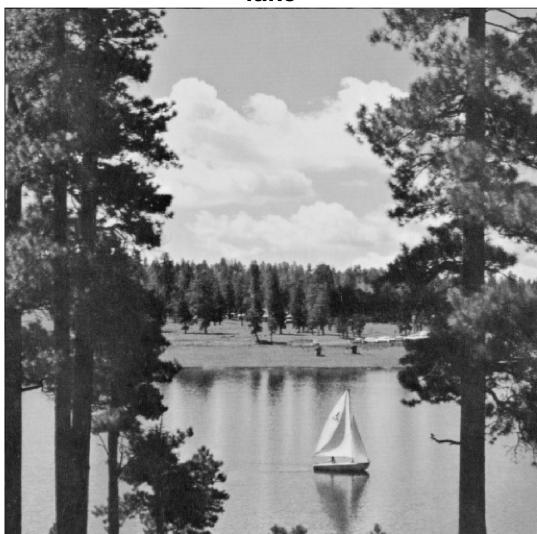
```



house



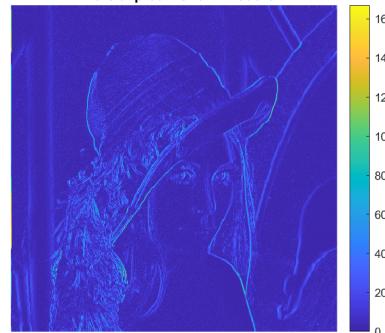
lake



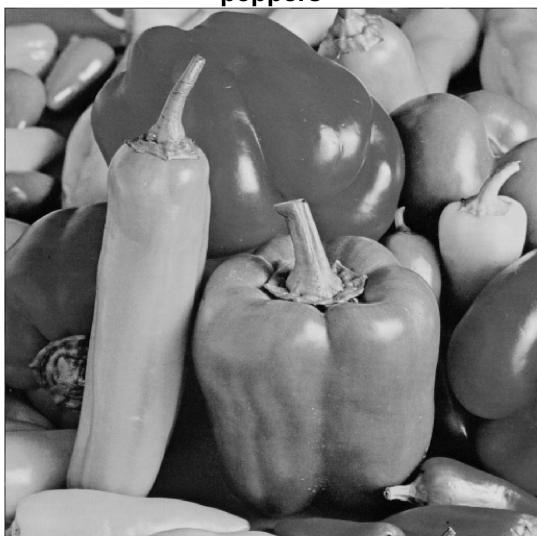
lena



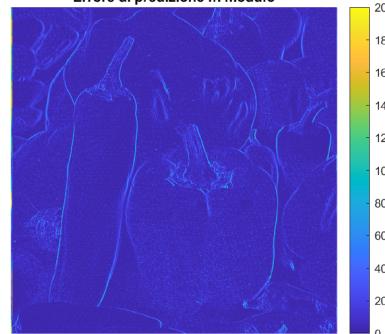
Errore di predizione in modulo



peppers



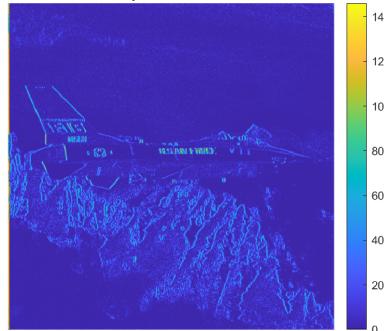
Errore di predizione in modulo



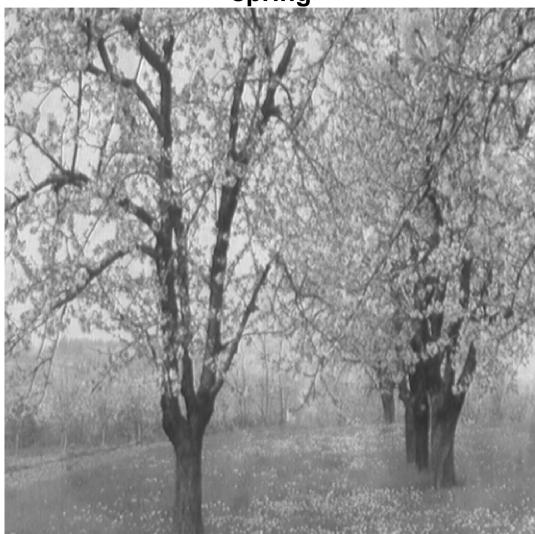
plane



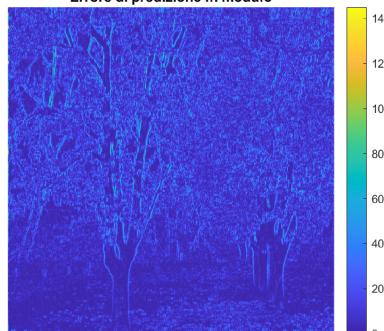
Errore di predizione in modulo



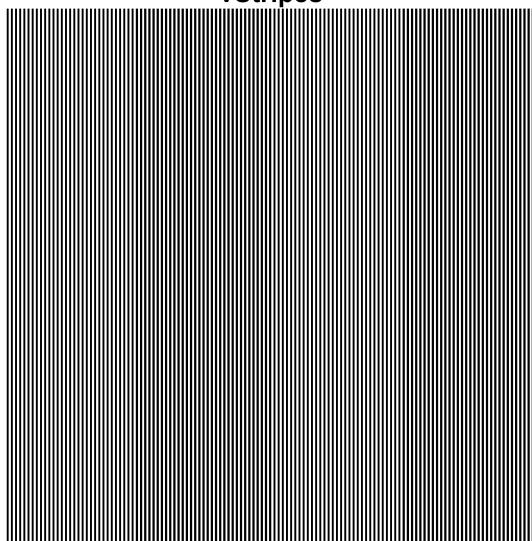
spring



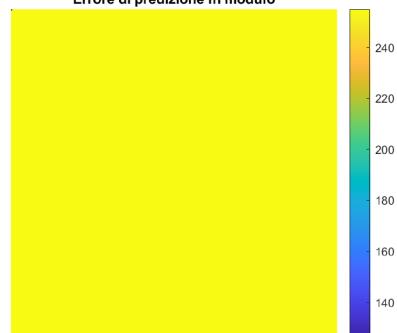
Errore di predizione in modulo



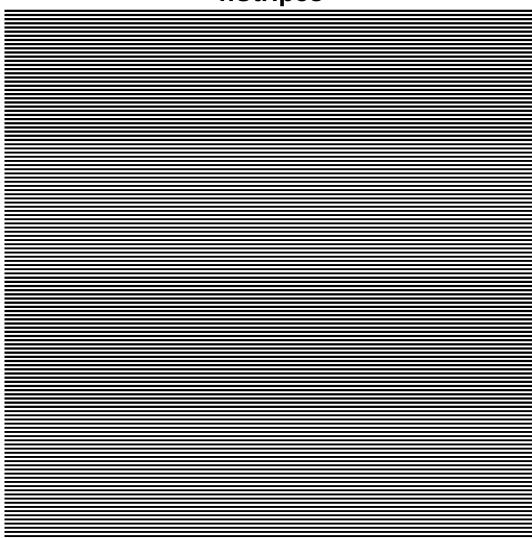
vStripes



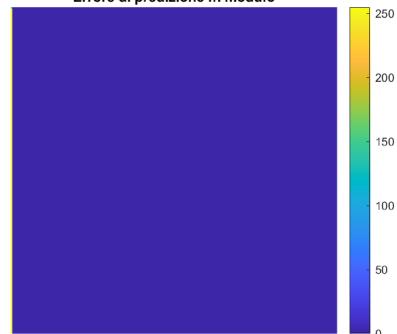
Errore di predizione in modulo



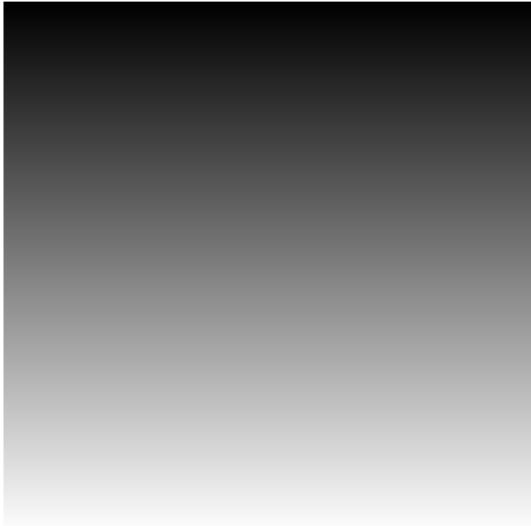
hStripes



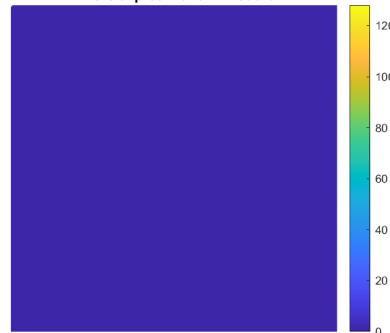
Errore di predizione in modulo



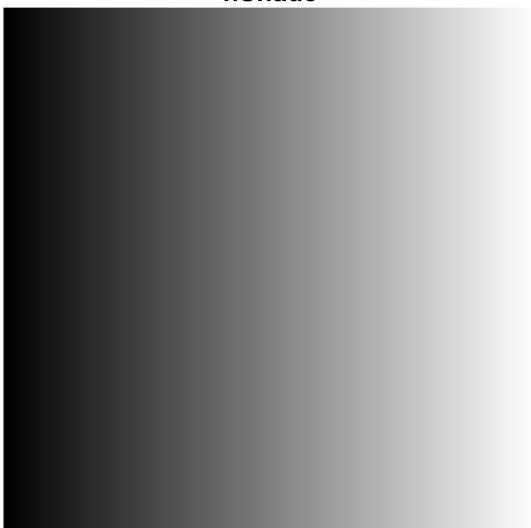
vShade



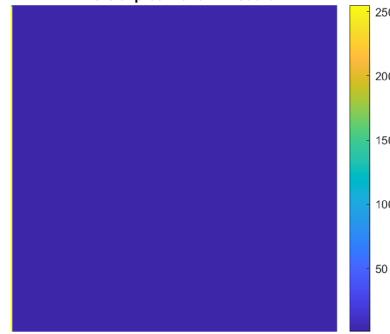
Errore di predizione in modulo

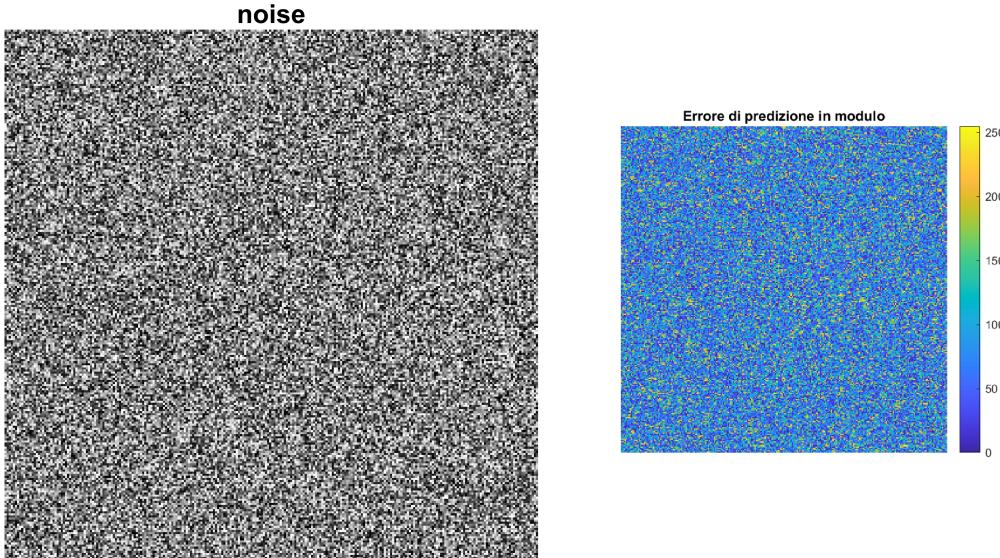


hShade



Errore di predizione in modulo





Dati raccolti

Immagine	dim	HX	7zip	HY	S-EG
einst	512x512	6,7850 bpp	6,1744 bpp	5,3945 bpp	6,9466 bpp
house	512x512	7,0564 bpp	7,7683 bpp	3,3123 bpp	3,4280 bpp
lake	512x512	7,4845 bpp	6,6909 bpp	5,6092 bpp	7,0357 bpp
lena	512x512	7,4451 bpp	6,4579 bpp	5,0645 bpp	6,1036 bpp
peppers	512x512	7,5937 bpp	6,7127 bpp	5,0970 bpp	6,3497 bpp
plane	512x512	6,7039 bpp	5,4900 bpp	4,6268 bpp	5,2084 bpp
spring	512x512	7,1571 bpp	6,8140 bpp	5,9502 bpp	7,7694 bpp
vStripes	256x256	1,0000 bpp	0,0325 bpp	1,0003 bpp	17,0000 bpp
hStripes	256x256	1,0000 bpp	0,0620 bpp	0,0409 bpp	1,0625 bpp
vShade	256x256	8,0000 bpp	0,0878 bpp	0,0370 bpp	1,0080 bpp
hShade	256x256	8,0000 bpp	0,0955 bpp	0,0370 bpp	3,0547 bpp
noise	256x256	7,9972 bpp	8,0204 bpp	8,7115 bpp	13,7085 bpp

Si può notare come il programma di compressione riesca a fare un ottimo lavoro sulle immagini che hanno una certa regolarità

Parte 2

1. Effettuare la codifica predittiva “avanzata”: per prima cosa si costruisce il predittore p tramite una scansione dell’immagine f:

```

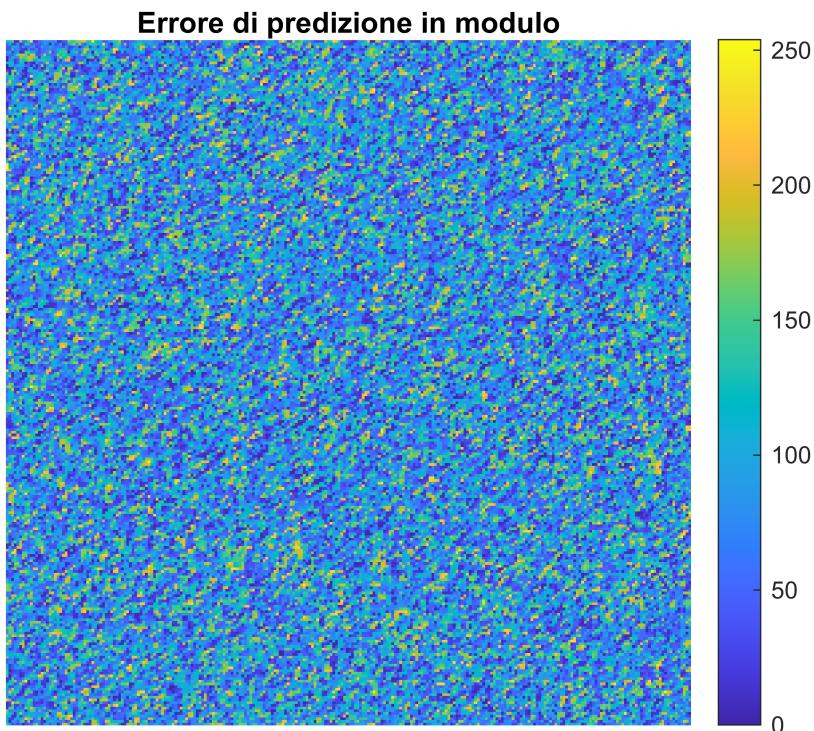
pred = zeros(row, col, 'uint8');
% 1.1. Per ogni pixel dell'immagine in posizione n, m
for n = 1:row
    for m = 1:col

```

```

% 1.2. Se è il primo pixel, il predittore è p(0,0) = f(0,0) - 128 [In
Matlab, p(1,1) = f(1,1) - 128
if(n==1 && m==1) pred(n,m) = f(n,m) - 128;
% 1.3. Altrimenti, se siamo sulla prima riga, il predittore è il pixel a
sinistra di quello corrente
elseif(n==1) pred(n,m) = f(n,m) - f(n, m-1);
% 1.4. Altrimenti, se siamo sulla prima colonna, il predittore è quello in
alto
elseif(m==1) pred(n,m) = f(n,m) - f(n-1, m);
% 1.5. Altrimenti, se siamo sull'ultima colonna, il predittore è il valore
mediano tra f(n - 1, m), f(n, m - 1), e f(n - 1, m - 1).
elseif(m==col) pred(n,m) = f(n,m) - median([f(n-1, m), f(n, m-1), f(n-1,
m-1)]);
% 1.6. Altrimenti il predittore è il valore mediano tra f(n - 1, m), f(n, m
- 1), e f(n - 1, m + 1).
else pred(n,m) = f(n,m) - median([f(n-1, m), f(n, m-1), f(n-1, m+1)]);
end
end
% 1.7. Una volta costruito il predittore, si calcola l'errore di predizione y = f -
p (è un immagine)
predErr = double(f)-double(pred); % differenza tra matrici
figure; imagesc(reshape(abs(predErr),row,col)); axis image; axis off; colorbar;
title('Errore di predizione in modulo')

```



2. Valutare l'entropia di y

```

tmp = transpose(predErr);
rasterScan = tmp(:);
HY = hentropy(rasterScan)

```

HY = 7.6707

3. Valutare il numero di bit necessari per codificare l'errore di predizione con la codifica Exp Golomb con segno, dedurne il tasso di codifica

```

bitCount = 0;
for n = 1:row
    for m = 1:col
        symbol = predErr(n,m);
        codeword = expGolombSigned(double(symbol));
        bitCount = bitCount + numel(codeword);
    end
end
EG_bpp = bitCount/nPixel;
fprintf('Tasso di codifica S-EG su errore di pred.: %5.4f\n',EG_bpp );

```

Tasso di codifica S-EG su errore di pred.: 14.1730

4. Confrontare l'entropia e il tasso di codifica predittiva avanzata con quelle del caso "semplice", con l'entropia dell'immagine e con il tasso di codifica dell'applicazione zip

5. Ripetere gli esperimenti per più immagini e riportare i risultati. Commentare quanto trovato.

Utilities

```

function result = hentropy(values)
    occorrenze = hist(values,-255:255); % contiamo le occorrenze dei valori
    freqRel = occorrenze/sum(occorrenze); % Trasformiamo le occorrenze in frequenze
    relative
    p = freqRel(freqRel>0); %Rimuoviamo eventuali valori nulli di probabilità
    result = p*log2(1./p'); % formula dell'entropia. In Matlab "*" effettua il
    prodotto scalare
end

%Exp Golomb code for non-negative numbers
function bits= expGolombUnsigned(N)
    if N
        trailBits = dec2bin(N+1,floor(log2(N+1)));
        headBits = dec2bin(0,numel(trailBits)-1);
        bits = [headBits trailBits];
    else
        bits = '1';
    end

```

```
end  
end  
  
%Exp Golomb code for non-negative numbers  
function bits= expGolombSigned(N)  
    if N>0  
        bits = expGolombUnsigned(2*N-1);  
    else  
        bits = expGolombUnsigned(-2*N);  
    end  
end
```