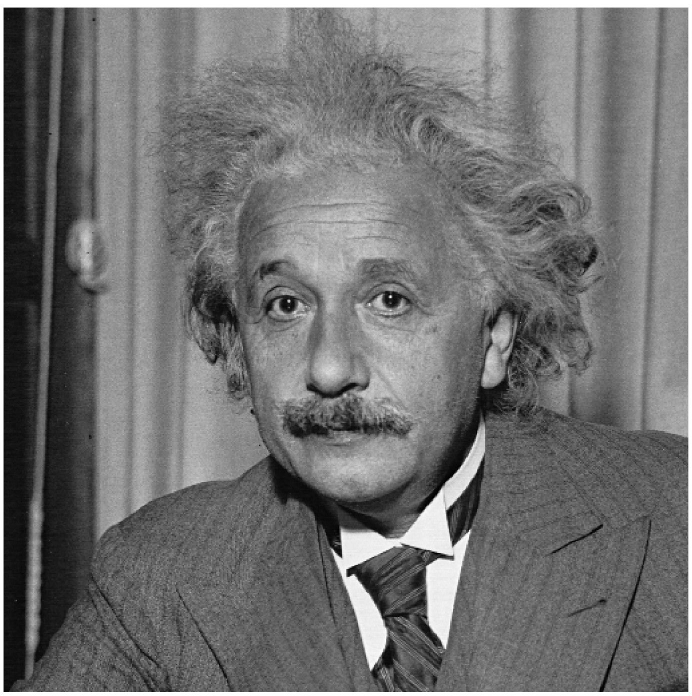
Homework 1 Roncolato

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 = "einst";
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;
```



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

```
L'entropia dell'immagine einst è di 6.7850 bpp
```

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

```
Il rapporto di compressione ottenibile è di 1.1791
```

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('tar.exe -a -cf %s.zip %s.pgm', imageName, imageName); % crea la
stringa di comando
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; % dmensione espressa in bpp
fprintf('Il file %s.zip ha un tasso di %5.4f bpp\n', imageName, zip_bpp);
```

```
Il file einst.zip ha un tasso di 6.3741 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, il teorema di Shannon $H(X) \leq \mathscr{L}^* < H(x) + 1$ si può riformulare nel seguente modo

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

Questo significa che all'aumentare della lunghezza K dei simboli, la lunghezza media ottima in bit per simbolo $\mathscr{L}_S^*$ si avvicina sempre di più al tasso entropico $\mathscr{H} = \frac{H(X^K)}{K}$, 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.

5. Effettuare la codifica predittiva "semplice"

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

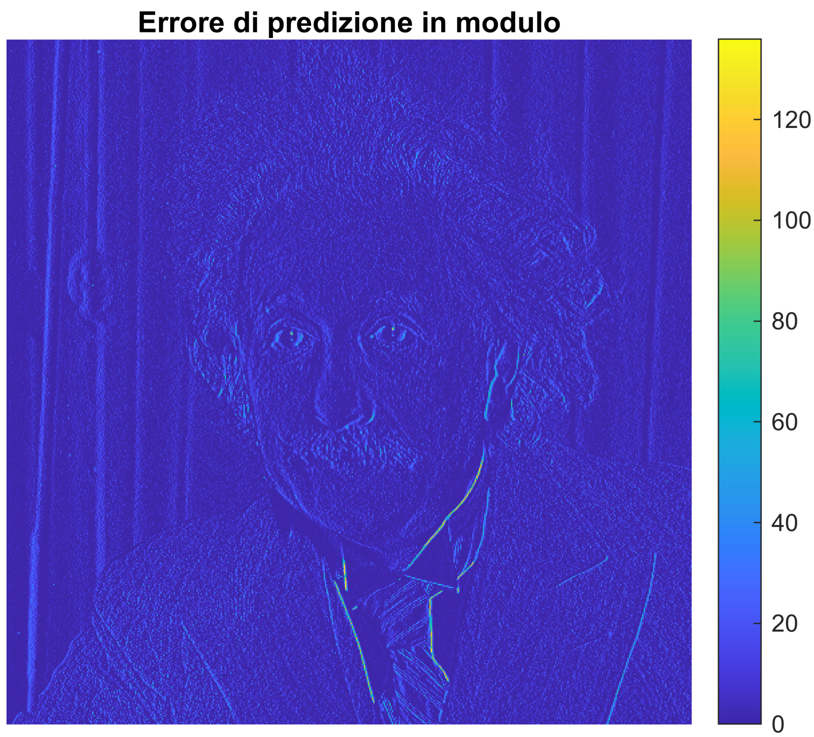
```
tmp = transpose(f);
rasterScan = tmp(:);
```

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; rasterScan(1:end-1)]; % viene ignorato l'ultimo pixel
```

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 = rasterScan-pred;
figure; imagesc(transpose(reshape(abs(predErr),row,col)));  axis image; axis off;
colorbar;
title('Errore di predizione in modulo')
```



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

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

```
L'entropia dell'errore di predizione per einst è di 6.7850 bpp
```

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

```
Il rapporto di compressione ottenibile è di 1.1791
```

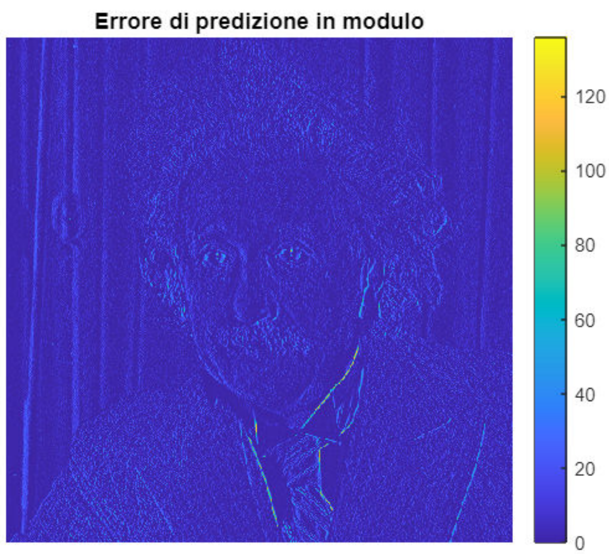
7. Valutare il *numero di bit necessari* per codificare l'errore di predizione y con la codifica *Exp Golomb con segno*, dedurne il bitrate di codifica e confrontare tale valore con quello ottenuto ai punti 1, 3 e 5

```
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 pred.: %5.4f bpp\n', EG_bpp);
```

```
Tasso di codifica S-EG su errore di pred.: 4.0203 bpp
```

[da riscrivere meglio ma il senso è che] In particolar modo per le immagini dove i colori cambiano in modo piuttosto graduale (da sinistra a destra), la predizione è buona e di conseguenza l'errore di predizione piccolo. La codifica Exp Golomb permette di codificare in modo efficiente i numeri piccoli quindi più è piccolo l'errore di predizione migliore sarà la codifica.

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

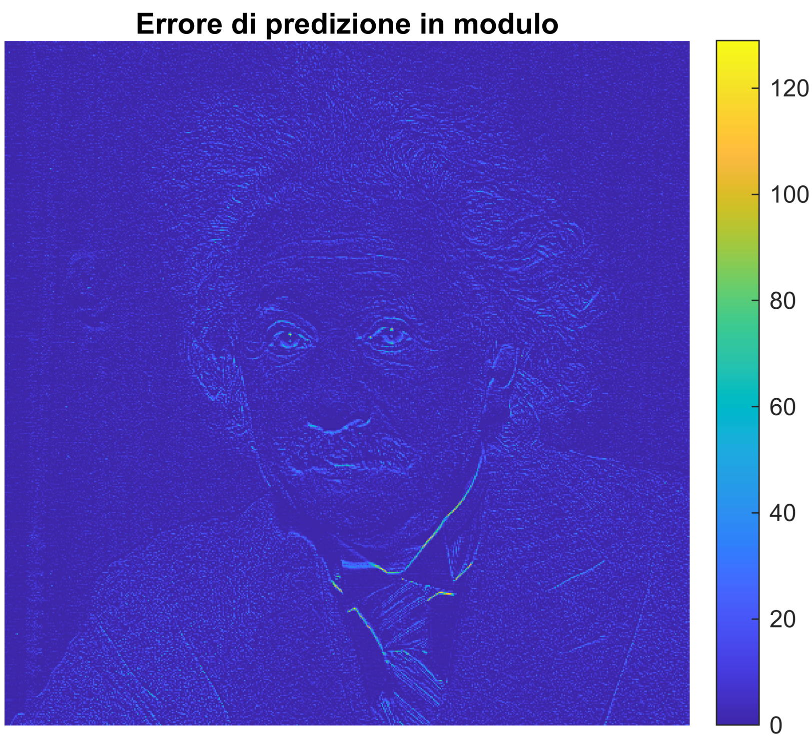


Parte 2

1. Effettuare la codifica predittiva "avanzata": per prima cosa si costruisce il predittore p tramite una scansione dell'immagine *f*:

```
p = 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) p(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) p(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) p(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) p(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 p(n,m) = f(n,m) - median([f(n-1, m), f(n, m-1), f(n-1, m+1)]);
        end
    end
end
% 1.7. Una volta costruito il predittore, si calcola l'errore di predizione y = f −
p (è un immagine)
y = f-p; % differenza tra matrici
figure; imagesc(reshape(abs(p),row,col));  axis image; axis off; colorbar;
title('Errore di predizione in modulo')
```



2. Valutare l'entropia di y

```
tmp = transpose(y);
rasterScan = tmp(:);
HY = hentropy(rasterScan)
```

```
HY = 6.7498
```

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 = y(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.: 15.4593
```

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

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

Utilities

```
function result = hentropy(values)
    occorrenze = hist(values,0:255); % 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


%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
```