Basic Compression Methods: Tutorial 1

1 Arithmetic coding

Recall: principles of arithmetic coding of a sequence of symbols:
After the counting of the frequencies of each symbols and sort them in the decreasing order, we calculate the corresponding intervals of probability with a High and a Low bound. The range of the intervals corresponds to the probability of the symbol.
Then, the sequence is iteratively coded with the following formulas:

NewHigh:=OldLow+Range*HighRange(X);
NewLow:=OldLow+Range*LowRange(X);
Range=OldHigh−OldLow

OldLow and ldHigh are initialized to 0 and 1.

1) Prepare a matlab script for the encoder.
   The inputs will be the list of symbols, the probabilities and the sequence to code. Symbols and probabilities list are sorted in the decreasing order of probabilities. The outpout is the binary stream of the sequence.
2) Propose the corresponding decoder

1.1 Test 1

Consider again the DNA alphabet {A,C,T,G} with respective symbols probabilities {0.5,0.3,0.15,0.05}.
Verify your code by evaluating the arithmetic code for the chain : ACTAGC and propose the decoding process

1.2 Test 2

Provide the arithmetic code for the sentence: BE_A_BEE

2 Huffman

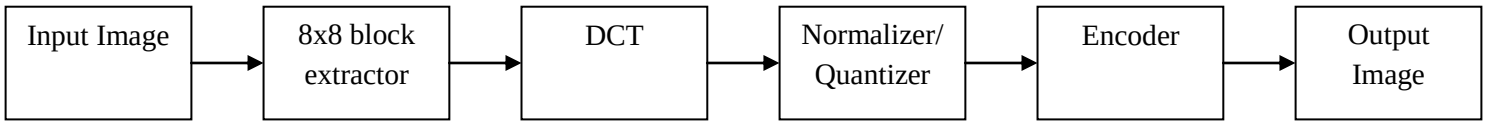
Useful reminders:

- *Global VAR (declared inside a function) ensures that the variable VAR will be retained through the various iterations of the function.*
- *The 'cell (m, n)' function creates an array of empty matrices that can be referenced by cell (using ()) or by content (using {})*

1. Write a function that gives the Huffman coding of a probability vector.
2. Write a function that encodes an image into a binary Huffman coding.

3 Simplified JPEG encoder

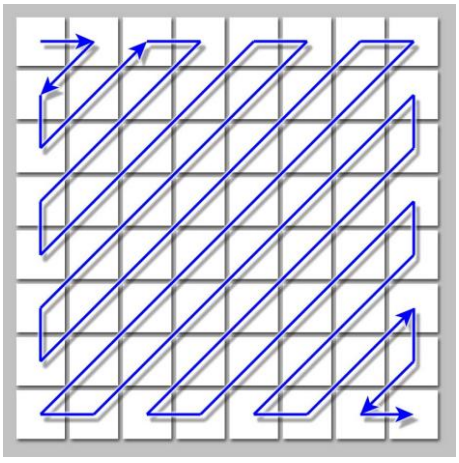
JPEG can be represented as the following block diagram:



The compression is performed in four sequential steps.
The input image is subdivided into non-overlapping pixel blocks of size 8x8. They are subsequently processed left to right, top to bottom. Its 64 pixels are level shifted by substracting 2^{m-1}, where 2^m is the number of gray levels in the image.
The function *B=blkproc(A,[M N], FUN, P1, P2, ...)* can be used to automate the process of dealing with images in blocks: *A* is the image input, *M N* are the sizes of the blocks that will be processed with function *FUN*. *P1, P2, …* are optional parameters of the function *FUN*. The result is reassembled into *B*.
The 2D discrete cosine transform is computed. The function *dctmtx(8)* returns a transformation matrix *H* of size 8. The resulting DCT *T* of *F* is obtained by: $T = HFH^T$, with *F* an 8x8 block of the image.
$\forall u, v \in \|0,7\|$, the resulting coefficients T(u,v) are then simultaneously normalized and quantized into $\hat{T}(u,v)$: $\hat{T}(u,v) = round\left[\frac{T(u,v)}{Z(u,v)}\right]$,
where Z(u,v) is a transform normalization array like:

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

By scaling Z(u,v), various image quality can be achieved.
After each block's DCT coefficients are quantized, the elements of $\hat{T}(u,v)$ are reordered in accordance with the zigzag pattern:



The resulting one-dimensional array has been rearranged to increase the spatial frequency, so the encoder is designed to take advantage of the long runs of zeros that normally results from reordering. For this tutorial, we will simply truncate the stream (obtained from the DCT coefficients reordering) after its last non-zero coefficient. All the streams are then concatenated together and separated using an "End Of Block" symbol.

Block 1	EO	Block 2	EO	Block 3	EO

The resulting stream is then encoded using Huffman encoder.
The output of function *B=im2col(A, [M N], 'distinct')* is a matrix *B* in which each column contains the elements of one distinct block of size *MxN* of the input image *A*. String 'distinct' stands for non-overlapping blocks.

4 Supplementary material: 1-D lossless predictive coding

The approach called lossless predictive coding eliminates the inter-pixel redundancies of closely spaced pixels by extracting and coding only the new information in each pixel. Each successive pixel f of the input image is introduced to the encoder and the predictor generates the anticipated value of that pixel based on some number of past inputs. The output of the predictor is then rounded to the nearest integer, denoted $\hat{f}$ and used to form the difference or prediction error *e*:

$$e(x,y) = f(x,y) - \hat{f}(x,y)$$

$$\hat{f}(x,y) = round\left[\sum_{i=1}^{m} \alpha_i f(x,y-i)\right]$$

We want to encode an image using the simple first-order linear predictor:

$$\hat{f}(x,y) = round\left[f(x,y-1)\right]$$

1. Implement this function. A common value for α is 1.
2. Compute the entropy of the original image and of the encoded image.
3. Compute the image containing only the prediction coefficients and its histogram.
4. Which other steps could be applied to this image to obtain a compression?