Lab 1

Entropy estimation

The entropy of the signal is calculated through the following formula:

$$H(X) = \sum_{i=1}^{L} p_i \cdot i(a_i) = -\sum_{i=1}^{L} p_i \cdot \log \ p_i$$

And implemented in matlab as following:

```
a = hist (x, unique(x));
p = a/length(x);
entropy = -sum(p.*log2(p));
```

For doing the pair entropy:

$$H(X,Y) = -\sum_{i,j} P_{XY}(a_i, b_j) \cdot \log P_{XY}(a_i, b_j)$$

A probability matrix is first constructed by counting occurrence and diving by the total length and then the equation above is implemented after the loop:

```
pair_counting_matrix = zeros(2*128);

    for i = 1:length(x)-1
        %Use + 128 in order to get an index for neg values
        num1 = x(i)+128;
        num2 = x(i + 1)+128;
        pair_counting_matrix(num1, num2) = pair_counting_matrix(num1,
num2)+1;
    end
    prob_pair = pair_counting_matrix./length(x);
    %heatmap(prob_pair);
    pairEntropy = -sum(prob_pair(:) .* log2(prob_pair(:)), 'omitnan');
```

Finally the conditional entropy is calculated using:

$$H(X,Y) = H(X) + H(Y|X) = H(Y) + H(X|Y)$$

And moving the terms in the equation above it is implemented as this:

```
%% Conditional Entropy
conditional_entropy_y1 = pair_entropy_y1 - entropy_y1; %4.2725
conditional_entropy_y2 = pair_entropy_y2 - entropy_y2; %2.7221
conditional_entropy_y3 = pair_entropy_y3 - entropy_y3; %3.2891
```

Results:

	Entropy	Pair Entropy	Conditional Entropy
'hey04_8bit.wav'	7.1638	11.4363	4.2725
'nuit04_8bit.wav'	6.8060	9.5281	2.7221
'speech.wav'	5.4340	8.7231	3.2891

Memoryless Huffman coding

The provided function for doing huffman coding was used:

```
%Jonas Svanberg                    (C) 1995 Image Coding Group. LiU,
SWEDEN
```

First the relative frequencies of the symbols of the source is calculated and provided as an argument for the huffman function. The function returns the expected codeword length.

$$R = \frac{\bar{l}}{n}$$

The rate is therefore calculated as the expected rate divided by the number of symbols.

```
a1 = hist (y1, unique(y1));
p1 = a1/length(y1);
```

```
a2 = hist (y2, unique(y2));
p2 = a2/length(y2);

a3 = hist (y3, unique(y3));
p3 = a3/length(y3);

expected_codeword_length_1 = huffman(p1); %7.1968
expected_codeword_length_2 = huffman(p2); %6.8273
expected_codeword_length_3 = huffman(p3); %5.4512

R1 = expected_codeword_length_1/length(a1); %0.0300 [bits/symbol]
R2 = expected_codeword_length_2/length(a2); %0.0274
R3 = expected_codeword_length_3/length(a3); %0.0235
```

Simple predictive coding

For the predictive coding a simple loop was made and the predictor was calculated according to the formula:

```
% pi = xi-1
    prediction = zeros(size(y, 1),1);
    for i = 2:size(prediction, 1)
        prediction(i) = y(i-1);
    end
```

```
%2*xi-1 - xi-2
    prediction = zeros(size(y, 1),1);
    for i = 3:size(prediction, 1)
        prediction(i) = 2*y(i-1)-y(i-2);
    end
```

Still images

For calculating the entropy in 2D we start of by counting the relative frequency of the source:

```
x = size(img,1);
y = size(img,2);
count = zeros(256,1); %0-255
```

```
%Looping through all pixels in the given image
for i = 1:x
    for j = 1:y
        count(img(i,j)+1) = count(img(i,j)+1) + 1; %increment
    end
end
```

And then use the same equation for calculating entropy as before:

```
p = count./ (x*y);
entropy = -sum(p .* log2(p), 'omitnan');
```

For the vertical and horizontal pair entropy, I first start by counting the horizontal/vertical pairs:

```
for i = 1:x
        for j = 1:y-1
            %Use + 128 in order to get an index for neg values
            num1 = img(i,j)+1;
            num2 = img(i,j+1)+1;
            pair_counting_matrix(num1, num2) = pair_counting_matrix(num1,
num2)+1;
        end
    end
    prob_pair = pair_counting_matrix./(x*y);
```

Then same equation as before:

```
    pairEntropy = -sum(prob_pair(:) .* log2(prob_pair(:)), 'omitnan');
```

Conditional entropy vertically and horizontally are calculated similarly as before:

```
conditional_entropy_vert_img1 = pair_entropy_vert_img1 - entropy_img1;
%6.5219
conditional_entropy_vert_img2 = pair_entropy_vert_img2 - entropy_img2;
%4.6686
conditional_entropy_vert_img3 = pair_entropy_vert_img3 - entropy_img3;
%4.1854

conditional_entropy_horiz_img1 = pair_entropy_horiz_img1 - entropy_img1;
%6.2580
conditional_entropy_horiz_img2 = pair_entropy_horiz_img2 - entropy_img2;
%4.9047
conditional_entropy_horiz_img3 = pair_entropy_horiz_img3 - entropy_img3;
```

```
%5.7886
```

Results:

	Entropy	Pair Entropy Vertical	Pair Entropy Horizontal	Conditional Entropy Vertical	Conditional Entropy Horizontal
'baboon.png'	7.4745	13.9964	13.7325	6.5219	6.2580
'boat.png'	7.1238	11.7924	12.0285	4.6686	4.9047
'woodgrain.png'	6.3562	10.5416	12.1448	4.1854	5.7886

Memoryless Huffman coding

The huffman coding is done the same way as before, but counting the relative frequency in a loop instead:

```
for i = 1:x
    for j = 1:y
       count1(img1(i,j)+1) = count1(img1(i,j)+1) + 1; %increment
       count2(img2(i,j)+1) = count2(img2(i,j)+1) + 1; %increment
       count3(img3(i,j)+1) = count3(img3(i,j)+1) + 1; %increment
    end
end

p1 = count1./ (x*y);
p2 = count2./ (x*y);
p3 = count3./ (x*y);
```

```
expected_codeword_length_1 = huffman(p1); %7.5171
expected_codeword_length_2 = huffman(p2); %7.1468
expected_codeword_length_3 = huffman(p3); %6.3914

R1 = expected_codeword_length_1/length(count1); %0.0294
R2 = expected_codeword_length_2/length(count2); %0.0279
R3 = expected_codeword_length_3/length(count3); %0.0250
```

Simple predictive coding

For the first predictor each row and column is looped through and for each row the prediction is made based on the previous column, but the row value is maintained.

```
% pi,j = xi,j-1
    prediction = zeros(size(y, 1), size(y, 2));
    for i = 2:size(prediction, 1)
        prediction(i,1) = 128;
        for j = 2:size(prediction, 2)
           prediction(i,j) = y(i,j-1);
        end
    end
```

This is probably wrong:

The first predictor is implemented as this: (where acf is the auto correlation function)

```
acf = zeros(5,1);
    for lag = 0:4
        acf(lag+1) = corr(y(1:end - lag), y(lag + 1:end));
    end
    a = acf(2)/acf(1);
```

The variance of the first predictor can be expressed as:

$$\sigma_d^2 = E\{(X_n - p_n)^2\} \approx E\{(X_n - a_1 X_{n-1})^2\} = (1 + a_1^2)R_{XX}(0) - 2a_1 R_{XX}(1)$$

After differentiating and setting equal to zero we get that the parameter a is equal to

$$a_1 = \frac{R_{XX}(1)}{R_{XX}(0)}$$ which due to indexing in matlab is `acf(2)/acf(1);`

The second predictor is implemented like this:

```
acf = zeros(5,1);
    for lag = 0:4
        acf(lag+1) = corr(y(1:end - lag), y(lag + 1:end));
```

```
    end
    a1 = (acf(2)*acf(1)-acf(3)*acf(2)) / (acf(1)*acf(1)-acf(2)*acf(2));
    a2 = (acf(3)-a1*acf(2))/acf(1);
    %a2 = (acf(1)*acf(3)-acf(2)*acf(2)) / (acf(1)*acf(1)-acf(2)*acf(2));
```

With the same principle as before we differentiate the variance with regards to a1 and a2 and set to zero:

$$\frac{\partial}{\partial a_1}\sigma_d^2 = 2a_1R_{XX}(0) - 2R_{XX}(1) + 2a_2R_{XX}(1) = 0$$

$$\frac{\partial}{\partial a_2}\sigma_d^2 = 2a_2R_{XX}(0) - 2R_{XX}(2) + 2a_1R_{XX}(1) = 0$$

Solving out a1 and a2 gets us the equations written in the code above.

Final answer is:

```
a1 = predictor1(y1); %0.9820
a2 = predictor1(y2); %0.9981
a3 = predictor1(y3); %0.9507


[a1, a2] = predictor2(y1); %a1: 1.2969, a2: -0.3206
[a1, a2] = predictor2(y2); %a1: 1.8431, a2: -0.8466
[a1, a2] = predictor2(y3); %a1: 1.7716, a2: -0.8636
```