Assignment 1 TTT4135 Multimedia Signal Processing

Table of Contents

Group members: Benjamin Strandli Fermann, Chenyan Zhang and Even Florenes

Problem 1

Task a

Find the probability pn for each state of the source X Xn (n =1, 2).

p(x1|x0) = 1/4 p(x1|x1) = 2/3 p(x0|x0) = 3/4 p(x0|x1) = 1/3

p(x1)/p(x0) = p(x1|x0)/p(x0/x1) = (1/4)/(1/3)

p(x1)/p(x0) = 3/4

p(x0)+p(x1) = 1

p(x0)+(3/4)p(x0) = 1

(7/4)p(x0) = 1

p(x0) = 4/7 = 0.571

p(x1) = 1-4/7 = 3/7 = 0.429

Simulation of Markov process gives the same result

```
x0 = 0;
x1 = 0;
state = 0;
for k = 1:1000000
    if state
        if rand < 2/3
            x1 = x1 + 1;
            state = 1;
        else
            x0 = x0 + 1;
            state = 0;
        end
    else
        if rand < 0.75
```

```
            x0 = x0 + 1;
            state = 0;
        else
            x1 = x1 + 1;
            state = 1;
        end
    end
end

fprintf('Probability of state 0: %0.2f \n',x0/k);
fprintf('Probability of state 1: %0.2f \n',x1/k);
```

```
Probability of state 0: 0.57
Probability of state 1: 0.43
```

Task b

Find entropy H(X)

H(X) = (4/7)*log2(7/4)+(3/7)*log2(7/3) = 0.985

Task c

Probability of extended states

P(x1 x1) = P(x1)*P(x1|x1) = 2/7 P(x1 x0) = P(x1)*P(x0|x1) = 1/7 P(x0 x1) = P(x0)*P(x1|x0) = 1/7 P(x0 x0) = P(x0)*P(x0|x0) = 3/7

```
Find Huffman code and entropy rate B:
              /
            /   \
      2/7 /   \   3/7
        /   \   2/7
     1/7     1/7

Bit encoding
P(x0 x0) | 1
P(x1 x1) | 01
P(x1 x0) | 001
P(x0 x1) | 000

Entropy: B = (2/7)*log2(3.5)+2*(1/7)*log2(7)+(3/7)*log2(7/3) = 1.842
Average bit: 0.25*(2+3+3+1) = 2.25
```

Using extended source compared to two states does not given any gain

Problem 2

Task a

error: e = sum(y_i-yhat_i)^2 = sum(y_i-a*x_i)^2

Find a which gives lowest error:

de/da = 2*sum((y_i-a*x_i)*x_i) = 2*sum(y_i*x_i)-2*a*sum(x_i^2)

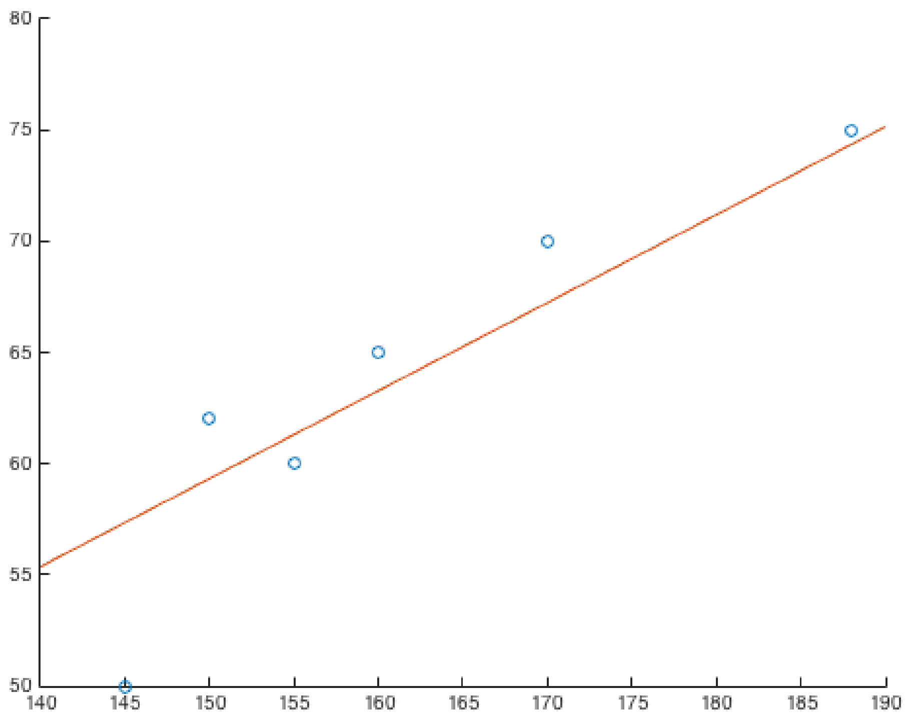
Find best a => de/da = 0

a = sum(y_i*x_i)/sum(x_i^2) Doing MATLAB-simulation gives: a = 0.3955

```
xh = [160, 188, 155, 170, 145, 150];
yv = [65, 75, 60, 70, 50, 62];

a = sum(xh.*yv)/sum(xh.*xh);
xLine = [140,190];
yEst = xLine.*a;
figure;
scatter(xh,yv);
hold on;
plot(xLine, yEst);
hold off;
fprintf('a = %g', a);
```

```
a = 0.395504
```

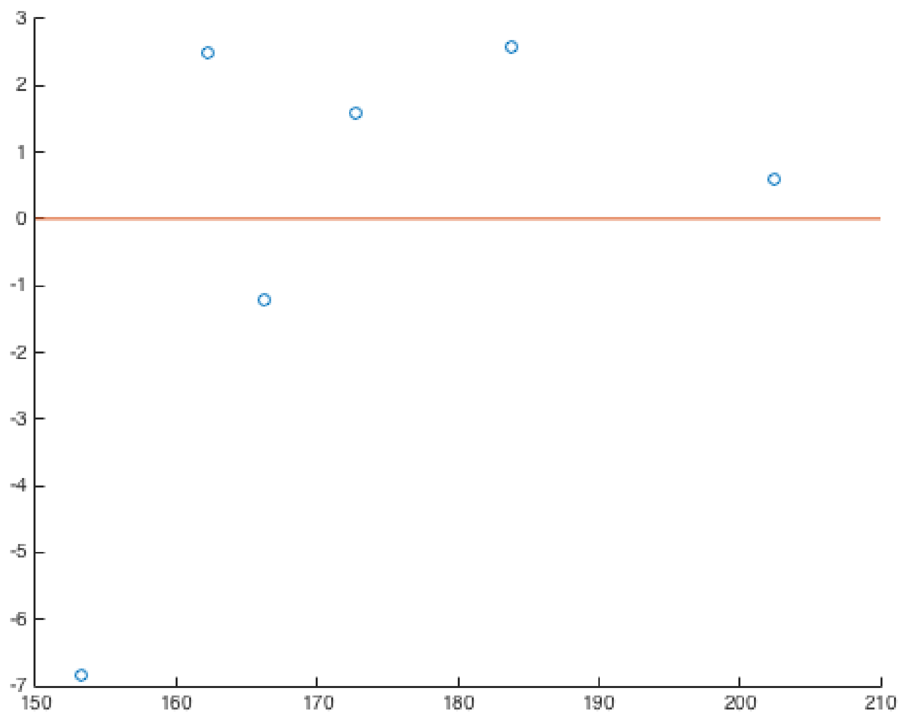


Task b

We see that the line is now flipped onto the x-axis, so it's easy to see the distance from the estimated line. We notice that the shortest person does not fit the linear regression well and has a large impact on the position of the line due to the nature of MSE.

```
phi = atan(a);
xMat = [xh;yv];
```

```
T = [cos(phi), sin(phi); -sin(phi), cos(phi)];
f = T*xMat;
figure;
scatter(f(1,:), f(2,:));
hold on;
plot([150,210], [0,0]); %line at zero
hold off;
```



Task c

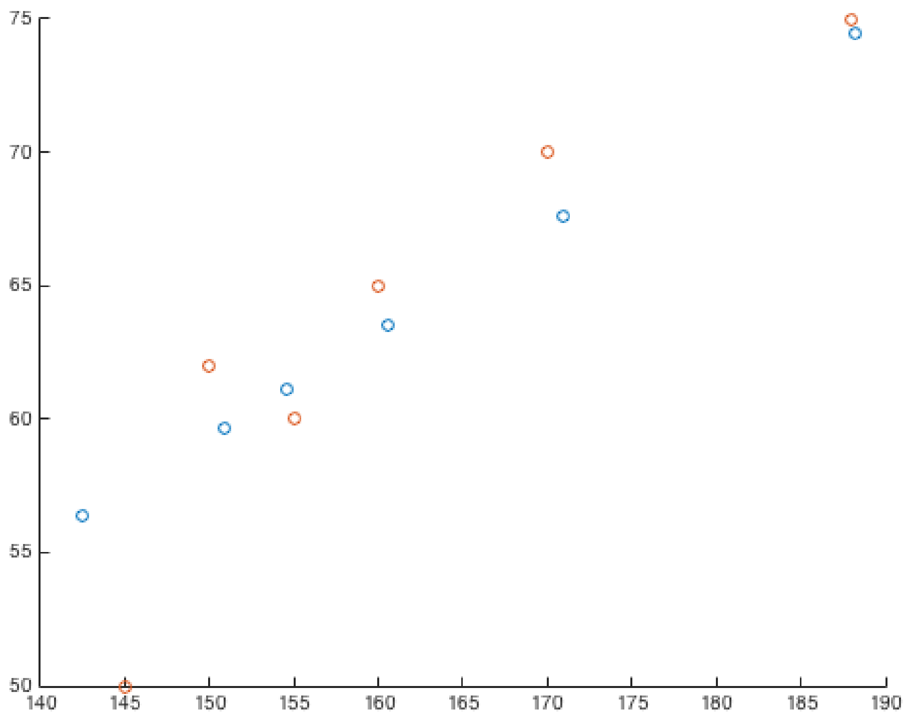
We make some assumptions about the statistical properties, namely the linear relation between height and weight and also that the error is normally distributed with a constant variance. As long as this is true, and the variance is sufficiently small, this can give good compression.

```
fCompress = f;
fCompress(2,:) = 0;

xCompress = T'*fCompress;
figure;
scatter(xCompress(1,:), xCompress(2,:));
hold on;
scatter(xh,yv);

avgErrorYv = mean(abs(xCompress(2,:)-yv));
fprintf('Average weight error: %g\n', avgErrorYv);
```

```
Average weight error: 2.37145
```



Problem 3

Task a

```
image = [124 125 122 120 122 119 117 118;
121 121 120 119 119 120 120 118;
126 124 123 122 121 121 120 120;
124 124 125 125 126 125 124 124;
127 127 128 129 130 128 127 125;
143 142 143 142 140 139 139 139;
150 148 152 152 152 152 150 151;
156 159 158 155 158 158 157 156 ];

imageDCTTransformed = dct2(image);
```

Task b

```
quantizationTable = [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];
```

```
transformTableImage = zeros(size(image,1),size(image,2));
transformTableImageDCT = zeros(size(image,1),size(image,2));

for i = 1:size(image,1)
    for j = 1:size(image,2)
        transformTableImage(i,j) = floor(image(i,j)/
quantizationTable(i,j)+0.5);
        transformTableImageDCT(i,j) = floor(imageDCTTransformed(i,j)/
quantizationTable(i,j)+0.5);
    end
end
% The coefficents k_ij for image:
fprintf('The coefficents k_ij for image: \n');
transformTableImage
fprintf('The coefficents k_ij for transform: \n');
transformTableImageDCT
```

```
The coefficents k_ij for image:

transformTableImage =

     8    11    12     8     5     3     2     2
    10    10     9     6     5     2     2     2
     9    10     8     5     3     2     2     2
     9     7     6     4     2     1     2     2
     7     6     3     2     2     1     1     2
     6     4     3     2     2     1     1     2
     3     2     2     2     1     1     1     1
     2     2     2     2     1     2     2     2

The coefficents k_ij for transform:

transformTableImageDCT =

    66     1     0     0     0     0     0     0
    -9     0     0     0     0     0     0     0
     3     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
```

Using the coefficients for the DCT-transformed will give us less quantization error. In the original image the energy is somewhat evenly distributed over the whole image, but for the DCT-transform all the energy is in the first part of the table. These values are a large an easy to quantize with high precision in the reconstruction. Using the DCT-transform also gives us the possibility of using zigzag- walktrough the table and stop when there is only zeros left.

Task c

```
imageDCTQuantized = zeros(size(image,1),size(image,2));
imageQuantized =  zeros(size(image,1),size(image,2));
for i = 1:size(image,1)
```

```
    for j = 1:size(image,2)
    imageDCTQuantized(i,j) =
 (transformTableImageDCT(i,j)*quantizationTable(i,j)-0.5);
    imageQuantized(i,j) =
 (transformTableImage(i,j)*quantizationTable(i,j)-0.5);

    end
end

imageInverseDCTQuantized = idct2(imageDCTQuantized);

psnrImageQuantizedDCT = psnr(image,imageInverseDCTQuantized,255)
psnrImageQuantized = psnr(image,imageQuantized,255)
```

```
psnrImageQuantizedDCT =

   40.2907


psnrImageQuantized =

   21.2311
```

Published with MATLAB® R2015a

Task 4

Second order predictor

$$\hat{X}(n) = \sum_{i=1} b_i x(n-i)$$

$R_x(0) = \sigma_x$
$R_x(1) = c^2$
$R_x(k) = 0, |k| > 1$

$$= b_1 x(n-1) + b_2 x(n-2)$$

Error between predictor and signal

$$e(n) = x(n) - \hat{x}(n)$$

$$= x(n) - b_1 x(n-1) \cdot b_2 x(n-2)$$

Error power

$$\sigma_e^2 = E(e^2(n)) = E\left((x(n) - b_1 x(n-1) - b_2 x(n-2))^2\right)$$

$$= E\big(x^2(n) - b_1 x(n) x(n-1) - b_2 x(n) x(n-2)$$
$$- b_1 x(n) x(n-1) + b_1^2 x^2(n-1) + b_1 b_2 x(n-1) x(n-2)$$
$$\ldots\big)$$

$$= E\big(x^2(n) - 2b_1 x(n) x(n-1) - 2b_2 \cancel{x(n) x(n-2)}$$
$$+ 2 \cdot b_1 b_2 x(n-1) x(n-2) + b_1^2 x^2(n-1) + b_2^2 x^2(n-2)\big)$$

$$= R_x(0) - 2b_1 R_x(1) + 2b_1 b_2 R_x(1) + b_1^2 R_x(0) + b_2^2 R_x(0)$$

$$= R_x(0)(1 + b_1^2 + b_2^2) + 2R_x(1)(b_1 b_2 - b_1)$$

$$\frac{\partial \sigma_e^2}{\partial b_1} = 2b_1 R_x(0) + 2R_x(1) b_2 - 2R_x(1) = 0$$

$$\cancel{2} b_1 R_x(0) + \cancel{2} R_x(1)(b_2 - 1) = 0$$

$$b_1 = \frac{R_x(1)(1 - b_2)}{R_x(0)}$$

$$\frac{\partial \sigma_e^2}{\partial b_2} = \cancel{2} b_2 R_x(0) + \cancel{2} R_x(1) b_1 = 0$$

$$b_2 R_x(0) = - b_1 R_x(1)$$

$$b_2 = -\frac{R_x(1)}{R_x(0)} \cdot b_1$$

$$= - \frac{R_x(1)}{R_x(0)} \frac{R_x(1)}{R_x(0)} (1 - b_2)$$

$$b_2 \left(1 - \frac{R_x^2(1)}{R_x^2(0)}\right) = - \frac{R_x^2(1)}{R_x^2(0)}$$

$$b_2 = - \frac{a^2}{\sigma_x^2} \frac{1}{1 - \frac{a^2}{\sigma_x^2}}$$

$$= - \underline{\underline{\frac{a^2}{\sigma_x^2 - a^2}}}$$

$$b_1 = \frac{R_x(1)}{R_x(0)} (1 - b_2)$$

$$= \frac{a}{\sigma_x} \left(1 + \frac{a^2}{\sigma_x^2 - a^2}\right)$$

$$= \frac{a}{\sigma_x} \left(\frac{\sigma_x^2 - \cancel{a^2} + \cancel{a^2}}{\sigma_x^2 - a^2} \right)$$

$$= \frac{a}{\cancel{\sigma_x}} \frac{\sigma_x^{\cancel{2}}}{\sigma_x^2 - a^2} = \underline{\frac{a \sigma_x}{\sigma_x^2 - a^2}}$$

Task 5

a)

Both JPEG and JPEG2000 do lossy compression, but JPEG2000 is also capable of lossless compression. JPEG2000 also offers scalability, the wavelet transform is done several times and gives several low-resolution approximations of the picture. This way lower-resolution versions can be displayed before the entire bitstream has been received. JPEG uses DCT on 8x8 blocks, while JPEG2000 uses wavelets on the entire image. JPEG's partitioning of the image is in blocks and macro-blocks of sections of the image, but JPEG2000 divides the image into sub-bands of high and low frequency.

JPEG2000 uses EBCOT and a binary arithmetic coder (MQ-coder) for entropy coding and encodes each sub-band separately in order to achieve good error resilience. JPEG supports both Huffman and arithmetic coding for entropy coding, although Huffman is most commonly used.

c)

In general JPEG2000 looks better than JPEG at lower bitrates, but the type of distortion is different between the two. JPEG has a lot of contouring because it codes the image on a block by block basis. JPEG2000 has no contouring artefacts, but does have some ringing along sharp edges because JPEG2000 compresses in the frequency domain. As a result of this, images of similar quality look better on smooth even surfaces with JPEG2000, but the very detailed areas with sharp edges look better with JPEG.

d)

Café

Rates [quality]	Bits per pixel	PSNR for JPEG [dB]	PSNR for JPEG2000 [dB]
5	0,296	21,69	23,98
20	0,729	26,50	29,35
40	1,097	29,21	32,84
60	1,419	31,20	35,30
80	2,073	34,67	39,53

Bike

Rates [quality]	Bits per pixel	PSNR for JPEG [dB]	PSNR for JPEG2000 [dB]
5	0,198	24,67	28,45
20	0,454	29,64	32,97
40	0,707	32,18	35,62
60	0,942	33,95	37,71
80	1,436	36,93	40,95

Woman

Rates [quality]	Bits per pixel	PSNR for JPEG [dB]	PSNR for JPEG2000 [dB]
5	0,167	24,91	28,34
20	0,412	29,63	32,48
40	0,651	32,09	35,26
60	0,891	33,90	37,64
80	1,401	37,01	41,07