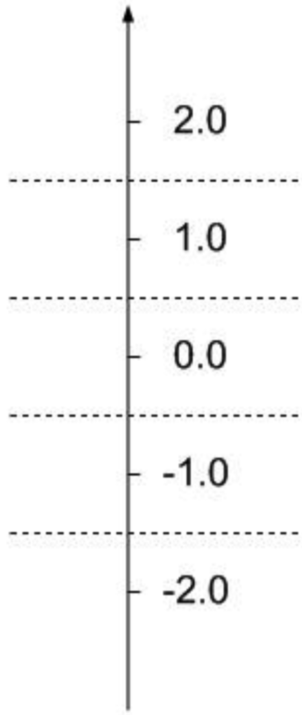
8.1 Quantization and Huffman coding



8.1.1 Use a 5-level uniform scalar quantizer as shown to quantize the sample sequence {0.25, -1.10, -0.15, 2.35, -1.40, 0.10, 0.90, -0.05}. Provide the output sequence.

[I] $\{-\infty, -1.5\} = -2.0$
[II] $\{-1.5, -0.5\} = -1.0$
[III] $\{-0.5, 0.5\} = 0.0$
[IV] $\{0.5, 1.5\} = 1.0$
[V] $\{1.5, \infty\} = 2.0$

For {0.25, -1.10, -0.15, 2.35, -1.40, 0.10, 0.90, -0.05}

Output:
{0.0, -1.0, 0.0, 2.0, -1.0, 0.0, 1.0, 0.0}

8.1.2 Design the best fixed-length code for the outputs of this quantizer, i.e. for an alphabet A={2.0, 1.0, 0.0, -1.0, -2.0}. Then encode the quantization output sequence from 8.1.1 using this code.

For Alphabet A (Assign Values):
$2.0 \rightarrow 001$ $1.0 \rightarrow 010$ $0.0 \rightarrow 011$ $-1.0 \rightarrow 100$ $-2.0 \rightarrow 101$

Output:
{011, 100, 011, 001, 100, 011, 010, 011}

8.1.3 Design a Huffman code for the same alphabet A = {2.0, 1.0, 0.0, -1.0, -2.0} assuming the probabilities P(2.0)=0.15, P(1.0)=0.20, P(0.0)=0.40, P(-1.0)=0.15, P(-2.0)=0.10. Then encode the quantization output sequence from 8.1.1 using this code.

***0.0**: 0.40*
***1.0**: 0.20*
***2.0**: 0.15* ***0.0**: 0.40*
***-1.0**: 0.15* —0— ***N1**: 0.25* ***0.0**: 0.40*
***-2.0**: 0.10* —1— ***1.0**: 0.20* —0— ***N2**: 0.35* —0— ***N3**: 0.60* —0— ***N4**: 1.00*
***2.0**: 0.15* —1— ***N1**: 0.25* —1— ***0.0**: 0.40* —1—

2.0	001
1.0	000
0.0	1
-1.0	010
-2.0	011

Output:
{1, 010, 1, 001, 010, 1, 000, 1}

8.2 Differential Coding (assuming there is no quantization or coding error, i.e. $\hat{x}[n] = x[n]$)

8.2.1 Use differential coding with the predictor $\tilde{x}[n] = \hat{x}[n-1]$ to encode the sequence
10 11 12 11 12 13 12 11

$x[0] = 10$
$x[1] = 11 - 10 = 1$ $\quad$ $x[2] = 12 - 11 = 1$ $\quad$ $x[3] = 11 - 12 = -1$
$x[4] = 12 - 11 = 1$ $\quad$ $x[5] = 13 - 12 = 1$ $\quad$ $x[6] = 12 - 13 = -1$
$x[7] = 11 - 12 = -1$

Output:
{10, 1, 1, -1, 1, 1, -1, -1}

8.2.2 Use the same predictor to encode another sequence
10 -10 8 -7 8 -8 7 -7

$x[0] = 10$
$x[1] = -10 - 10 = -20$ $\quad$ $x[2] = 8 - (-10) = 18$ $\quad$ $x[3] = -7 - 8 = -15$
$x[4] = 8 - (-7) = 15$ $\quad$ $x[5] = -8 - 8 = -16$ $\quad$ $x[6] = 7 - (-8) = 15$
$x[7] = -7 - 7 = -14$

Output:
{10, -20, 18, -15, 15, -16, 15, -14}

8.2.3 Find a better linear predictor for this second sequence in 8.2.2.and perform the differential coding again. (Hint: your objective is to make sure the coded sequence has generally low amplitudes.)

$Let\ a_1 = \frac{1}{10}$

$\hat{x}[n] = a_1(x[n-1]) = \frac{x[n-1]}{10}$

$x[0] = 10/10 = 1$
$x[1] = -10 - 10 = -20/10 = -2$ $\quad$ $x[2] = 8 - (-10) = round(18/10) = 2$
$x[3] = -7 - 8 = round(-15/10) = 2$ $\quad$ $x[4] = 8 - (-7) = round(15/10) = 2$
$x[5] = -8 - 8 = round(-16/10) = -2$ $\quad$ $x[6] = 7 - (-8) = round(15/10) = 2$
$x[7] = -7 - 7 = round(-14/10) = -1$

Output:
{1, -2, 2, -2, 2, -2, 2, -1}

8.3 In a JPEG image coder, after the DCT, quantization and zig-zag scanning, all the AC coefficients are coded through a run-length coding. This run-length coding is defined as pairs of (*zero-run*, *amplitude*), where the *amplitude* is a non-zero coefficient and the *zero-run* is the number of zeros prior to this non-zero coefficient. At a certain point when there is no more non-zero coefficient in the block, a symbol EOB (end-of-block) is coded.

1. Now open image "**lenna.256**" in Matlab and process the first 8×8 block and name it x1:

   ```
   fid=fopen('lenna.256','r');
   x=fread(fid,[256,256],'uchar');
   fclose(fid);
   x1=x(1:8,1:8);
   ```

2. apply 2D DCT on this block use "dct2" function (in Matlab);
3. apply the quantization table $\boldsymbol{Q}$ on page 8 of Lecture 10 (in Matlab);
4. perform zig-zag scan and generate the run-length pairs (by hand);
5. Repeat 3 and 4 with a scaled quantization table $\mathbf{0.1}\boldsymbol{Q}$.

Code:

```
fid=fopen("lenna.256","r");
x=fread(fid,[256,256],"uchar");

fclose(fid);
x1=x(1:8,1:8);

q_mtx =     [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];

x2=dct2(x1);

x2a=round(x2/q_mtx);

x2b=round(x2a/(0.1*q_mtx));

diary off
```

X1:

	1	2	3	4	5	6	7	8
1	137	137	138	133	129	131	131	131
2	136	136	133	133	133	133	130	132
3	133	133	134	133	130	130	130	130
4	136	136	134	130	130	122	130	130
5	138	138	136	134	133	132	132	131
6	134	134	132	133	131	131	131	131
7	134	134	130	128	132	130	128	130
8	132	132	130	125	128	130	130	128

X2:

	1	2	3	4	5	6	7	8
1	1.0557e...	14.8539	5.6554	−1.0214	−1.2500	−1.5480	−0.7189	1.1394
2	7.1983	3.1651	−1.0094	−3.7645	0.3573	1.9691	0.8567	−0.9480
3	−3.1311	−2.3714	0.5732	2.6375	−2.6692	−1.1503	2.8839	−1.1851
4	7.2608	1.2428	−0.7337	0.0421	0.0515	0.3755	0.6720	−1.8109
5	0.2500	3.6567	2.9630	−1.9652	−2.2500	1.3049	−0.6861	2.5863
6	−4.2844	2.0431	1.6565	−2.6852	1.0094	−0.1028	−1.0352	2.1821
7	−1.6796	−1.6362	−1.1161	−1.7208	−1.8710	3.1041	0.9268	−0.9581
8	3.0593	−0.0087	−1.8256	1.0916	−1.2653	0.0033	1.8997	−1.1044

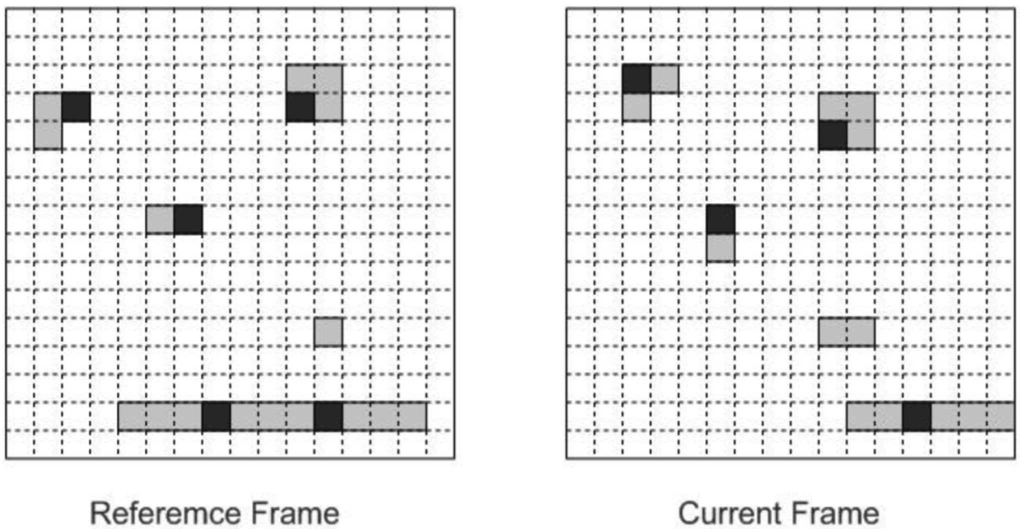
X2a:

	1	2	3	4	5	6	7	8
1	106	−124	142	−77	−166	−281	669	−321
2	0	0	1	0	−1	−1	3	−1
3	0	0	0	−1	0	0	0	0
4	0	−1	1	−1	−1	−2	4	−2
5	0	1	0	0	0	0	−1	0
6	0	0	−1	1	0	1	−3	1
7	0	0	0	0	0	0	0	0
8	0	0	1	0	0	−1	1	0

X2b:

	1	2	3	4	5	6	7	8
1	−343	185	1414	−905	−1041	−1008	3038	−1510
2	−3	3	5	−4	−3	−2	7	−4
3	0	0	0	1	−2	0	2	−1
4	−2	1	8	−5	−7	−5	18	−9
5	−1	1	−2	1	2	2	−8	4
6	2	−1	−5	2	5	2	−9	5
7	0	0	0	0	0	0	0	0
8	0	0	2	−1	−2	−1	5	−3

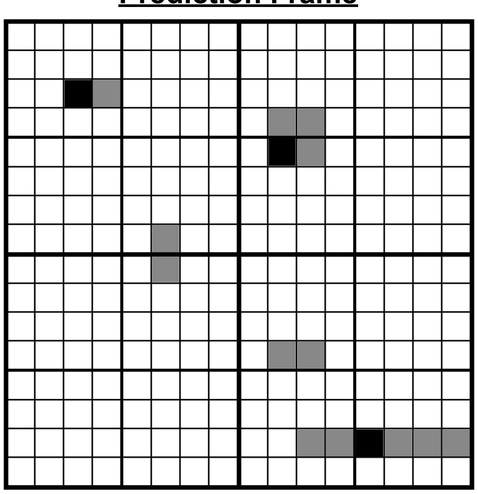
8.4 Based on the motion compensated estimation used in MPEG, find the motion vectors, the prediction frame and the difference frame for the current frame as shown. Assume each box represents a pixel, each macro-block is of 2×2 pixels, the white boxes have value of zero (0), the gray boxes have value of one (1), and the black boxes have value of two (2).



Motion Vectors

0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
0,1	**-8,-1**	0,0	0,0	**-1,1**	**-1,-1**	0,0	0,0
0,-1	0,0	0,0	0,0	**-1,1**	**-1,1**	0,0	0,0
0,0	0,0	0,0	**-1,0**	0,0	0,0	0,0	0,0
0,0	0,0	**0,1**	0,0	0,0	0,0	0,0	0,0
0,0	0,0	0,0	0,0	**-2,0**	**-1,0**	0,0	0,0
0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
0,0	0,0	**0,2**	**0,2**	**0,2**	**1,0**	**1,0**	**1,0**

Prediction Frame



Difference Frame