# Homework 4

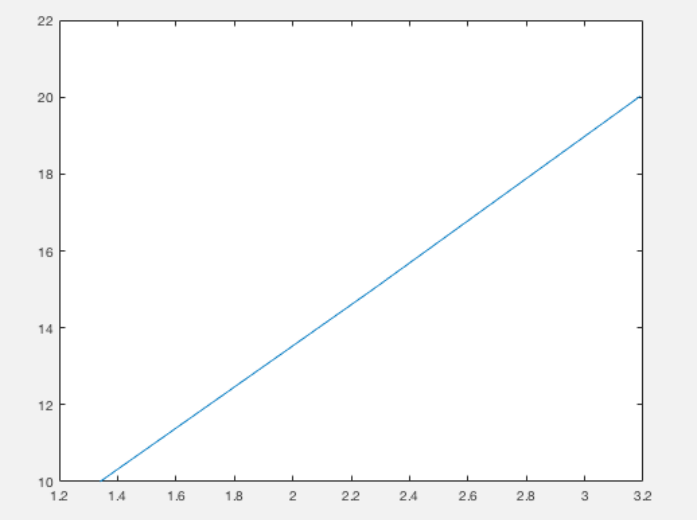
## a, b

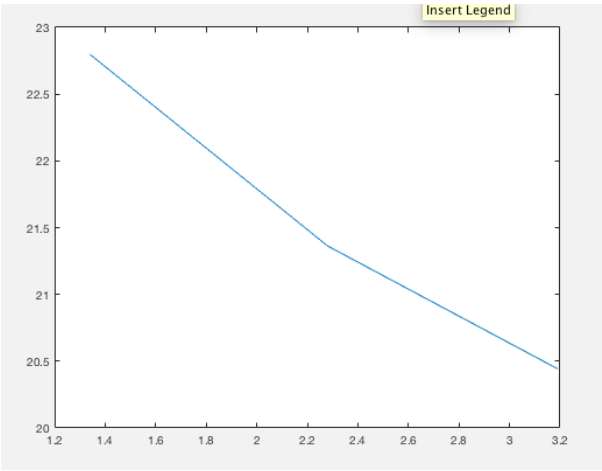
scalar = 1.34 => compression ratio = 10.0002, SNR = 22.7977

scalar = 2.28 => compression ratio = 15.0286, SNR = 21.3633

scalar = 3.19 => compression ratio = 20.0237, SNR = 20.4430

## c



# Instruction on bitstream structure

## Structure of DC data bit stream

hsm hsm hsm ... hsm

## The algorithm to decode the DC bit stream:

1. Read from the beginning of the stream, decode the bit stream using DC Huffman table(derived later) to get an "h", which is the corresponding codeword of k(a category of r in the DC residual)
2. Read the next one bit s which represents the sign
3. Read the next k-1 bits, which is m, the binary representation of t,
4. Until now we've completely read an "hsm" and decode to k, sign, t

   r = sign * 2^(k-1) + t
5. Repeat from procedure 1 until the bit stream is read out

## Structure of AC data bit stream:

hsm hsm hsm ... hsm EOB hsm hsm ... EOB ... EOB

## The algorithm to decode the AC bit stream:

1. Read from the beginning of the stream, decode the bit stream using AC Huffman table(derived later) until the decoded codeword is not 11110000, assume we've got p 11110000s and a last decoded word

   (If the first decoded codeword is not 11110000 => p = 0 and this first decoded word is exactly the last decoded word)

   The last decoded word have two possibility:

   *If it's not an EOB*, so it represents a pair of unique r and k, at this point, we've completely read an "h", which is decoded to p, r, k. Then read the next one bit which is the sign bit, we get s. Then read the next k-1 bits, which is the binary representation of t. Until now, we've read completely an single "hsm" and decode it to (p, r, k, s, t), the corresponding (d, x) is

   d = 15 * p + r

   x = (s==0 ? -1 : 1) * 2^(k-1) + t

*If it's an EOB*, which indicates this part is not an "hsm" but an EOB, so it's time to organize all previous (d, x) and rebuild them to an 8 * 8 block.

2. repeat procedure 1 until all bit stream is decoded

## Structure of header's DC Huffman:

L gc gc gc... gc

The Huffman table has 12 words, which is fixed, so we will transfer 12 gc and there is no need to transfer the category number.
For each "gc", c is the codeword. g represents the following c's length, and each g is of fixed length, indicated by L. L is 8 bits long.
The value of L depends on the longest codeword in DC Huffman table, we want to use several bits to represent the longest length so all lengths could be represented using "those several bits", so

L's value = 1 + floor(log2(longest codeword length))

So each g's length is L's decimal value

To decode the Huffman table from the bit stream:
1. Read first 8 bits and turn it to a decimal value LV
2. Read next LV bits and turn it to a decimal value CLV(codeword length value)
3. Read next CLV bits and record the bitstream
4. Repeat from procedure 2, we will repeat 12 times. Each time will get an bitstream, it is the codeword corresponding to "times"

## Structure of header's AC Huffman:

Based on the result, I found it's a waste to transfer all AC Huffman table since most codewords of the table is useless. So we could only transfer those codewords who shows up more than 0 times in the data part, So the structure is like this:

L N qgc qgc qgc qgc ... qgc

Each "c" is a codeword corresponding to a range "q", (a "q" represents a unique pair of r and k), we could make q = 10 * r + k since r is from 0 to 14 and k is from 1 to 10
e.g. q = 19 => r = 10, k = 9 ; q = 90 => r = 8, k = 10

So it seems q is from 1 to 150, but we still need to represent 11110000 and EOB, let's use 151 to represent 11110000, 152 to represent EOB. So q is from 1 to 152, using 8 bits.

(Note: though we could represent each pair of (r, k), plus 11110000 and EOB by using continuous numbers from 1 to 152, where we don't have to transfer q, but we don't want to transfer all table, so the price is to add 8 bits to represent each q)

Each g indicates the following c's length, the length of g is indicated by L

And add an "N"(8 bits) after L to indicate how many codewords in the Huffman table we are going to transfer.

To decode the Huffman table from the bitstreams
1. Read first 8 bits(L), turn it to decimal value LV
2. Read next 8 bits(N), turn it to decimal value NUM
3. Read next 8 bits(q), to get the current category RK
4. Read next LV bits(g), turn it to decimal value CLV
5. Read next CLV bits(c), record the bitstream corresponding to RK
6. Repeat procedure 3~5 NUM times to finish building the AC table(part of the table, enough to decode all ac terms)

## Running the code

The main program is in "jpeg.m". Before running, make sure "river.gif" and quantizing matrix "Q.mat" are in the same folder.

The important data of results are
% ACcoded(AC data part)
% ACdict(the huffman of ACs)
% dch, dcs, dcm(DC data part)
% DChuffmanTable(the huffman of DC)

Only header's length is calculated and while the real header bit stream is not generated.