Project Report

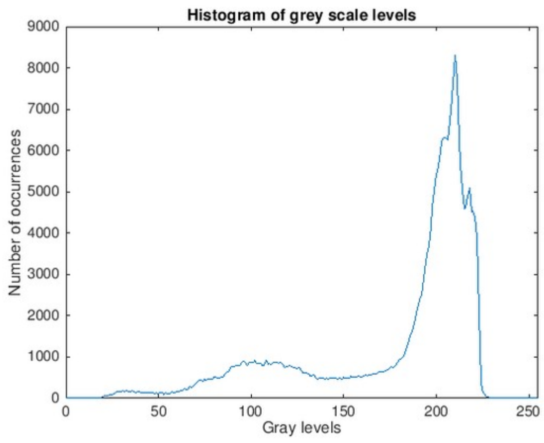
Cyril BERGER

<u>Provided files:</u>

- Assignment.mlx – *Implement all the required code of the questions*
- GRestimation.m – *Contain the function asked in subject*
- image_compress.m – *Contain the code only required to compress an image. The cropping of the image has been preserved and is part of the function. It's basically a copy of assignment, without any ploting or displaying. The cropping can be disabled by commenting the 6th line.*
- bonus1.mlx – *Code required for first bonus*
- bonus2.mlx – *Code required for second bonus*

<u>Answers :</u>

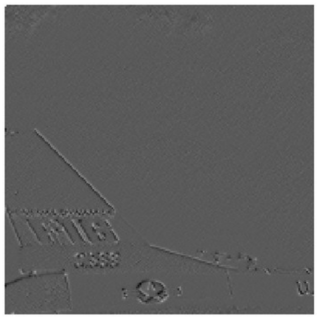
The provided image has the following distribution of grey scale :



Computed entropy = 6.7025

We only work on a subset of 200 by 200 pixels from the picture from now on, as asked in the subject.

In order to compress efficiently, we need to decorelate the data. For this, we use the Martucci predicator. On the first colum, we compute instead the difference with the pixel above, on the first row, the difference with the pixel on the left. For the top-left corner (the first element of e), we simply encode the value B(1,1) itself. This makes the transformation reversible This results is the following picture :

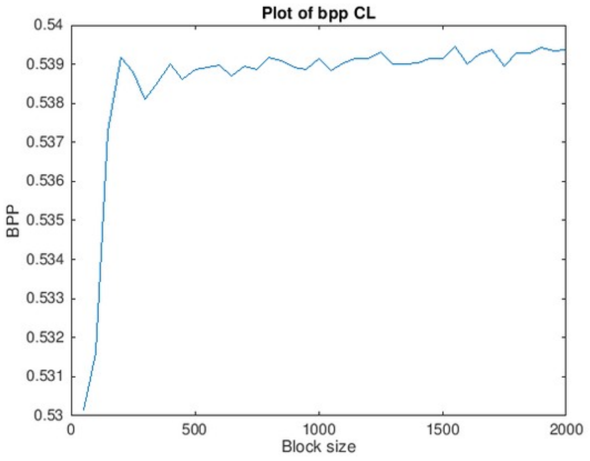


(lowest value is black, highest is white)

The image has far less variations than the original one, which means it likely has a lower entropy, and is therefore much more compressible. We have succefully decorelated the data.

We are going to compress the error matrix E using Golomb-Rice code. A first aproach is to find the optimal p to encode e. The optimal compression we found is 174748 bits (plus eventualy 8 bits for the value of p required to decode).

However, we can optimize the compression of the error matrix E by cutting it into blocks, with each block having its own p value. With our image, the optimal block size in the range [50:2000] is 50.



The compression of the error matrix with this size of block has a size of 169647 bits (is included the space required to encode the values of p for each block).

This is already an improvement. To find more effective compression, we can enlarge the possibilities of block sizes (regarding this image, it seems blocks smaller than 50 pixels can be interesting to try), or even try other encoding strategies than Golomb-Rice.

<u>Bonus 1 :</u>
The codelength obtained with Huffman encoding is 161186 bits, which is smaller than the Golomb-Rice encoding.

Using entropy as the average codelength per symbol, the estimation of the Huffman encoding is 160231 bits, slightly smaller than the real size obtained at the end.

<u>Bonus 2 :</u>
Since the image is purely black and white, we can only encode the run-length of pixels. I went for an implementation of Elias code to do this task. The implementation, as mentionned in the code, is designed to match the table seen at page 144 in the slides of the course.

With this technique, the image can be compressed at 139712 bits colomn-wise, and 94417 bits row-wise. The better performance of row-wise is due to characteristics in this image, namely there are large patches of color that are more wide than high (the shadow of the keyboard, the space bar, light parts on the keyboard…). Theses patches are better encoded row-wise than column-wise.

To fully decode the RLE encoding, one needs the width of the image and the color of the first pixel (since we only encode run-length). We could also encode a bit signaling wether the image has been encode row-wise or column-wise, allowing us to choose the best option with only 1 bit cost.

This RLE encoding is slightly better than gif compression (11.8 KB against 17 KB).