Eryk Urbanski

Practice 5 (Session 2)

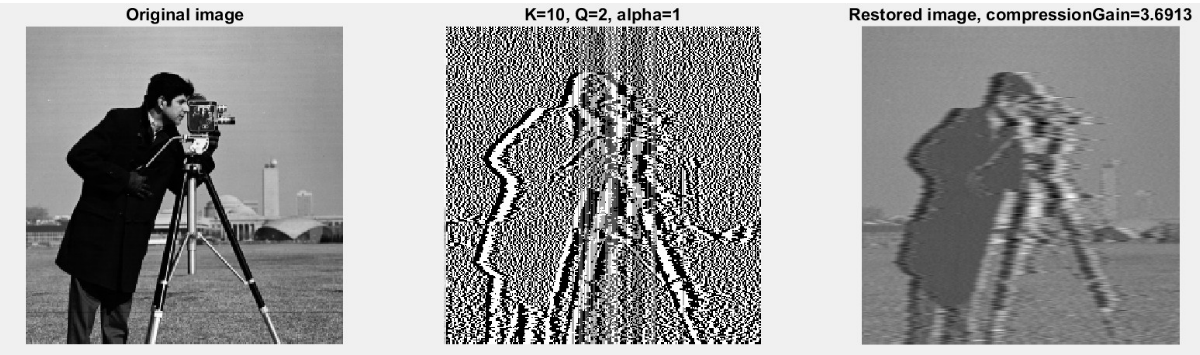
Exercise 5.3

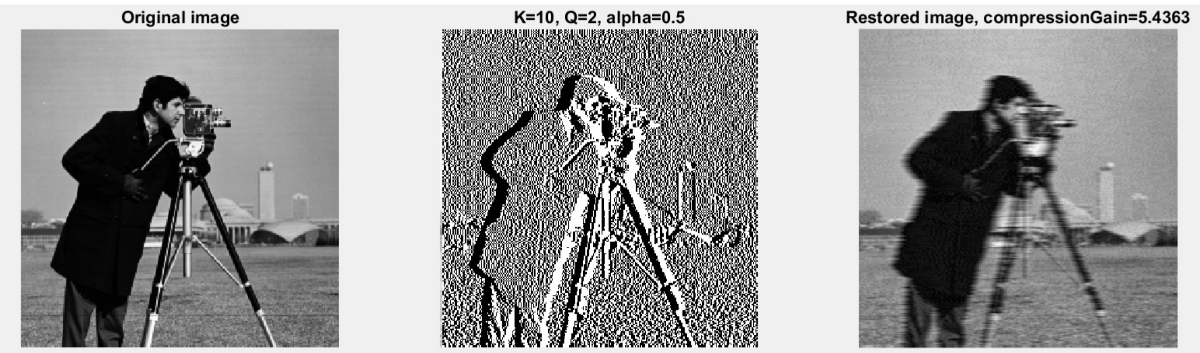
Design a lossy predictor, and use the entropy function to evaluate the compression ratio. Delta modulation with quantization step K will be used. The parámeter of the design will be adjusted from the initial values of **K** = 10 and $\alpha = 1$ for the a) section. And $\alpha = 0.5$ for the b) section. The function must have the form:

imagen comprimida = **predictorDelta** (imagen, **K**, α)

The predictor equation will be: $f_{n+1} = f_n + \alpha(f_n - f_{n-1})$

a) Compute the compression gain and compare the result obtained by using the two lossy predictors.





Code:

```
function e_ = lossyPredictor2ndOrder(im, K, alpha)
   Q = 2;
   im = double(im);
   [m, n] = size(im);
   e_ = double(zeros(m, n));
   e_(1:end, 1:2) = im(1:end, 1:2);
   im_ = double(zeros(m, n));
   im_(1:end, 1:2) = im(1:end, 1:2);

   for i = 1:m
      for j = 3:n
         prediction = round( im_(i, j-1) + alpha * ( im_(i, j-1) - im_(i, j-2) ) );
         e = im(i, j) - prediction;
         e_(i, j) = quant(e, K, Q);
         im_(i, j) = e_(i, j) + prediction;
      end
   end
end
```

```
function y = decompressor2ndOrder(error, alpha)
   [m, n] = size(error);
   y = zeros(m, n);
   y(1:end, 1:2) = error(1:end, 1:2);

   for i = 1:m
      for j = 3:n
         prediction = round( y(i, j-1) + alpha * ( y(i, j-1) - y(i, j-2) ) );
         y(i, j) = error(i, j) + prediction;
      end
   end
end
```

```
function e_ = quant(e, K, Q)
   delta = 511 / Q;
   level = ceil( abs(e) / delta);
   e_ = level * K * sign(e);
end
```

Exercise 5.4

a) Program a first order lossy. The quantifier has to be a lineal function where the number of levels can be selected. The value Q (number of levels) and K are introduced as input arguments to the predictor. The program has to be structured in functions.

Code:

```
function e_ = lossyPredictorQ(im, K, Q, alpha)
  im = double(im);
  [m, n] = size(im);
  e_ = double(zeros(m, n));
  e_(1:end, 1) = im(1:end, 1);
  im_ = double(zeros(m, n));
  im_(1:end, 1) = im(1:end, 1);

  for i = 1:m
    for j = 2:n
      prediction = alpha * im_(i, j-1);
      e = im(i, j) - prediction;
      e_(i, j) = quant(e, K, Q);
      im_(i, j) = e_(i, j) + prediction;
    end
  end
end
```

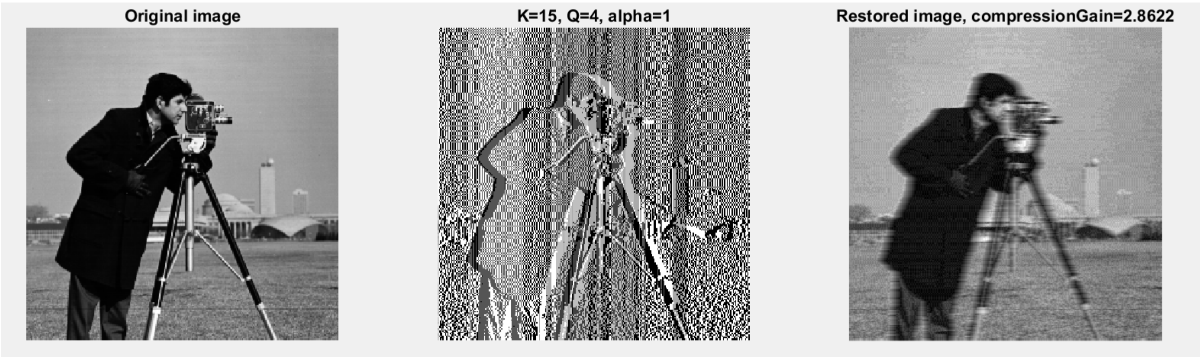
```
function y = decompressor(error, alpha)
  [m, n] = size(error);
  y = zeros(m, n);
  y(1:end, 1) = error(1:end, 1);

  for i = 1:m
    for j = 2:n
      y(i, j) = error(i, j) + round(alpha * y(i, j-1));
    end
  end
end
```

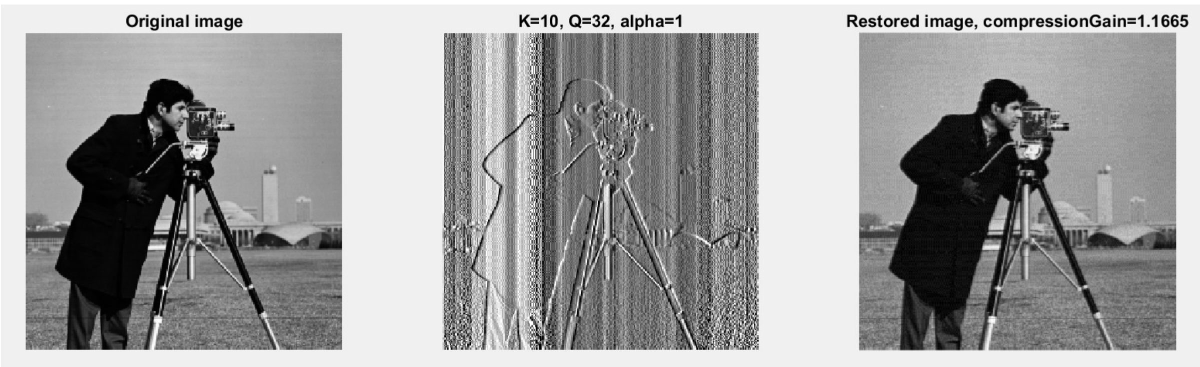
Helper functions:

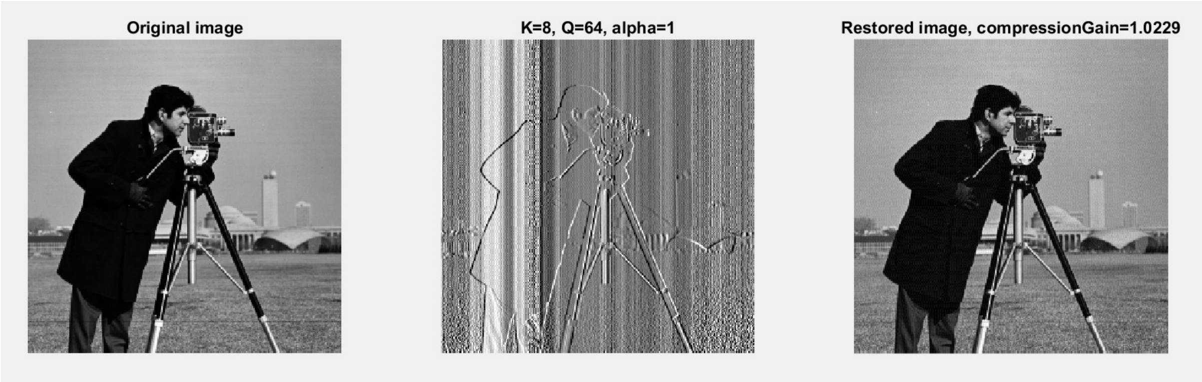
```
function y = normalize(x)
  y = x;
  mn = min(y);
  y = y + abs(mn);
  mx = max(y);
  y = uint8(y./mx.*255);
end
```

```
function y = computeCompressionGain(original, compressed)
  compressed = normalize(compressed);
  y = entropy(original)/entropy(compressed);
end
```

b) Compute the results for different values of Q (4, 8, 32, 64). Adjusting the K value to optimize the final quality of the resulting image.

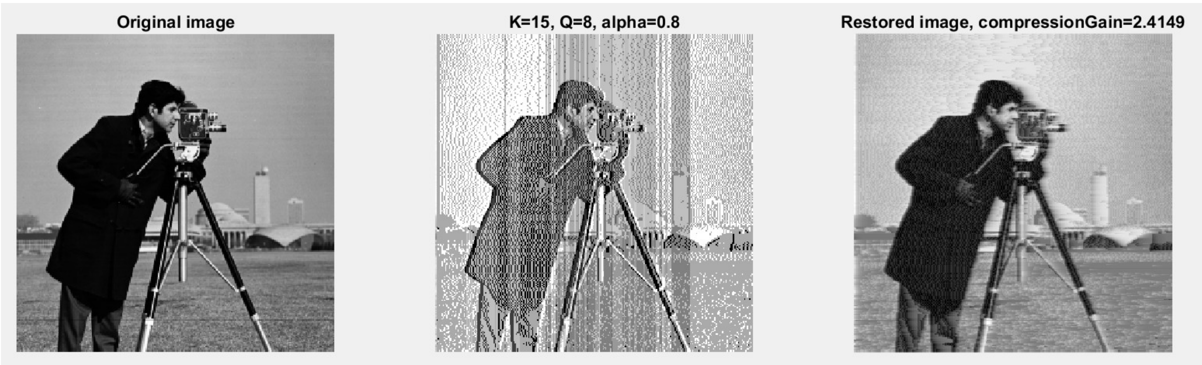






c) Name and comment an improvement in the algorithm to rise the quality of the compressed image without modifying Q or K.

We can also modify the alpha value. For example, setting the value of alpha at 0.8 in the case of Q = 8, gives a compression gain improvement of around 0.5 not worsening the quality of the restored image much.



Other methods to improve the algorithms could be: using higher order predictions, using a better quantization method (non-linear), or making the alpha parameter adaptive, changing its value based on other characteristics.