Compte rendu TP TNI

# Exercice 1

**1 )**

Programmer une fonction entropie(I), I étant la matrice d’une image lue avant l’appel à la fonction. NB : ne pas oublier de transformer I au format ‘double’ dès le début de la fonction.

**2 )** | A l’aide de la fonction précédente, calculer l’entropie de l’image *lena512.bmp*.

7.445506719708220

**3 )** Ajouter une constante à l’image lena512.bmp (attention à ne pas sortir de la dynamique de l’image de départ) et calculer l’entropie. Commenter

L’entropie ne change pas

Plus y’a un nombre d’éléments distinct plus entropie plus élevée

Ici le nombre est égal, les valeurs ont juste changé.

**4 )** Calculer l’entropie de l’image *souris.bmp*.

5.564301180819218

**5|** Commenter les différences entre les entropies des images *lena512.bmp* et *souris.bmp* et les performances de compression auxquelles on peut s’attendre. Confirmer ces différences en affichant les histogrammes des deux images.

Entropie plus faible car il y a moins d’éléments distinct, car l’image est plus uniforme

# Exercice 2

d = sum((A(:)|B(:)).^2) / prod(size(A));

psnr = 10\*log10(255\*255/d);

**2 ) la même image : -** inf

PSNR de la souris à elle-même bruité environ égal 20.5

# Exercice 3

Les images JPEG sont meilleures à taux de compression identiques ou presque. la supériorité visuelle de l’image codée JPEG est lié au QSU qui dégrade l’image

# Exercice 4

# Question 1 & 2

ASCII |Probability | Length | Code

33 | 0.000000 | 14 | 00001111001000

48 | 0.054318 | 4 | 1100

49 | 0.217688 | 2 | 10

50 | 0.067777 | 4 | 0011

51 | 0.053166 | 4 | 1101

52 | 0.068947 | 4 | 0010

53 | 0.070226 | 4 | 0001

54 | 0.052002 | 4 | 1110

55 | 0.048240 | 5 | 00000

56 | 0.044482 | 6 | 000010

57 | 0.051818 | 4 | 1111

File length = 968008

Entropy = 3.1004

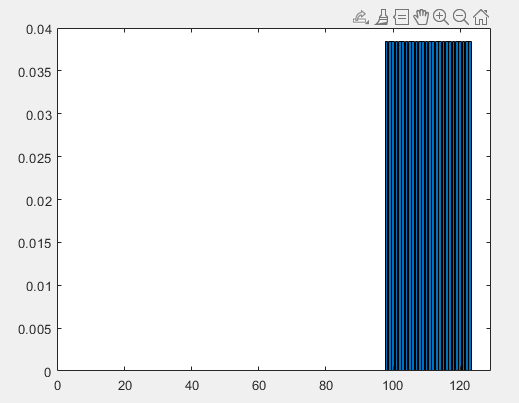
Average code length = 3.1618

Compression ratio = 0.4517

Entropy ratio = 1.0198

Aucun code attribué n'est préfixe d'un autre grâce à la fonction.

# Question 3



ASCII |Probability | Length | Code

a | 0.038462 | 5 | 10100

b | 0.038462 | 5 | 10011

c | 0.038462 | 5 | 10010

d | 0.038462 | 5 | 10001

e | 0.038462 | 5 | 10000

f | 0.038462 | 5 | 01111

g | 0.038462 | 5 | 01110

h | 0.038462 | 5 | 01101

i | 0.038462 | 5 | 01100

j | 0.038462 | 5 | 01011

k | 0.038462 | 5 | 01010

l | 0.038462 | 5 | 01001

m | 0.038462 | 5 | 01000

n | 0.038462 | 5 | 00111

o | 0.038462 | 5 | 00110

p | 0.038462 | 5 | 00101

q | 0.038462 | 5 | 00100

r | 0.038462 | 5 | 00011

s | 0.038462 | 5 | 00010

t | 0.038462 | 5 | 00001

u | 0.038462 | 5 | 00000

v | 0.038462 | 4 | 1111

w | 0.038462 | 4 | 1110

x | 0.038462 | 4 | 1101

y | 0.038462 | 4 | 1100

z | 0.038462 | 4 | 1011

File length = 26

Entropy = 4.7004

Average code length = 4.8077

Compression ratio = 0.6868

Entropy ratio = 1.0228

Même remarque que la question d’avant

# Question 4

97 a 0.500000 1 1

98 b 0.020000 6 010010

99 c 0.020000 6 010001

100 d 0.020000 6 010000

101 e 0.020000 6 001111

102 f 0.020000 6 001110

103 g 0.020000 6 001101

104 h 0.020000 6 001100

105 i 0.020000 6 001011

106 j 0.020000 6 001010

107 k 0.020000 6 001001

108 l 0.020000 6 001000

109 m 0.020000 6 000111

110 n 0.020000 6 000110

111 o 0.020000 6 000101

112 p 0.020000 6 000100

113 q 0.020000 6 000011

114 r 0.020000 6 000010

115 s 0.020000 6 000001

116 t 0.020000 6 000000

117 u 0.020000 5 01111

118 v 0.020000 5 01110

119 w 0.020000 5 01101

120 x 0.020000 5 01100

121 y 0.020000 5 01011

122 z 0.020000 5 01010

123 { 0.000000 12 010011101010

124 | 0.000000 12 010011101001

125 } 0.000000 12 010011101000

126 ~ 0.000000 12 010011100111

127  0.000000 12 010011100110

Cette fois ci, comme le a est présent a 50%, l’algorithme lui a fourni un code plus cours que l’exemple d’avant .