LEFEVRE Henry

SEGURET Aymeric

***R - Introduction***

***Exercice 1 :Poor man image proccessing***

> dim(matrice)

[1] 512 512

La matrice image est de 512x512 pixels.

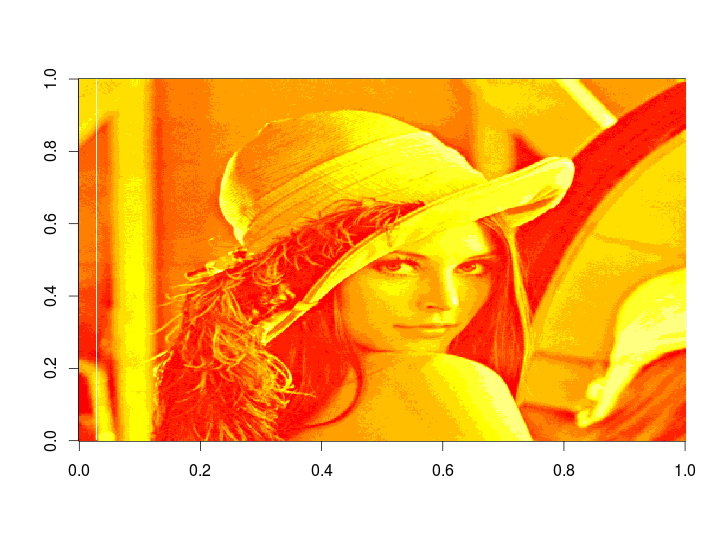
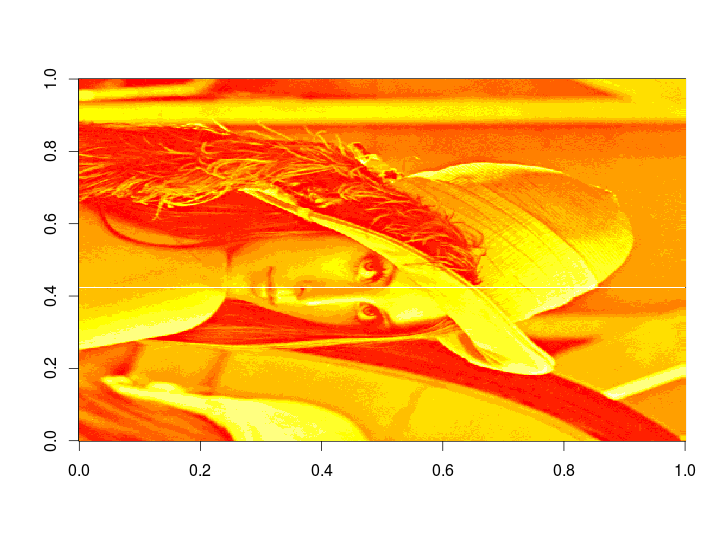
Rotation d’une matrice

La rotation de 90° s'effectue par permutation des colonnes de la matrice transposée de A.

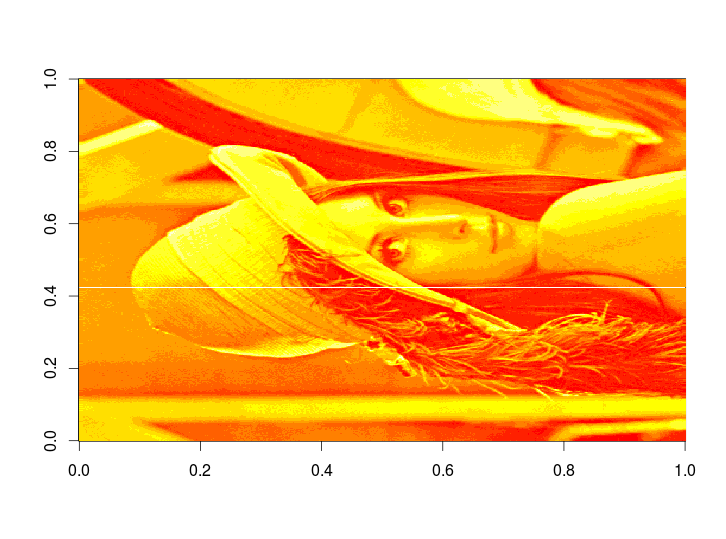
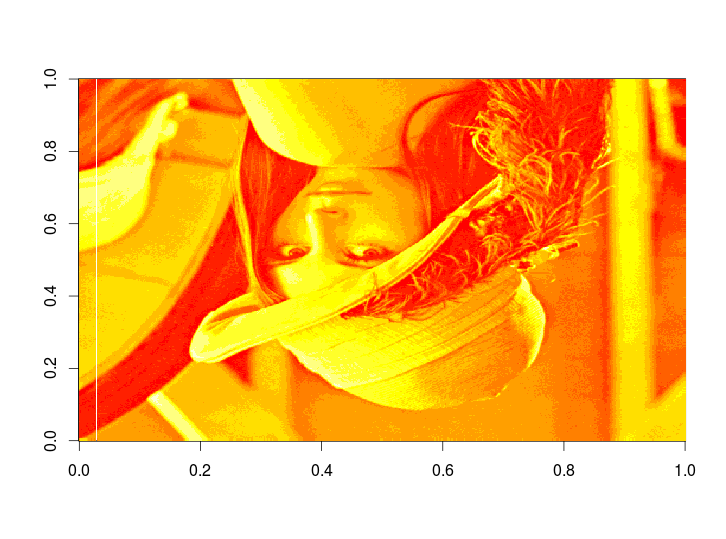
La rotation de 90° s'obtient par double rotation de 90°. Et celle de 270° par triple rotation.

1. rotation <- function (image, rot = 90) {
2. res <- apply(t(image),2,rev)
3. return ( res );
4. }

Résultats obtenus :

rotation 90° rotation 180°

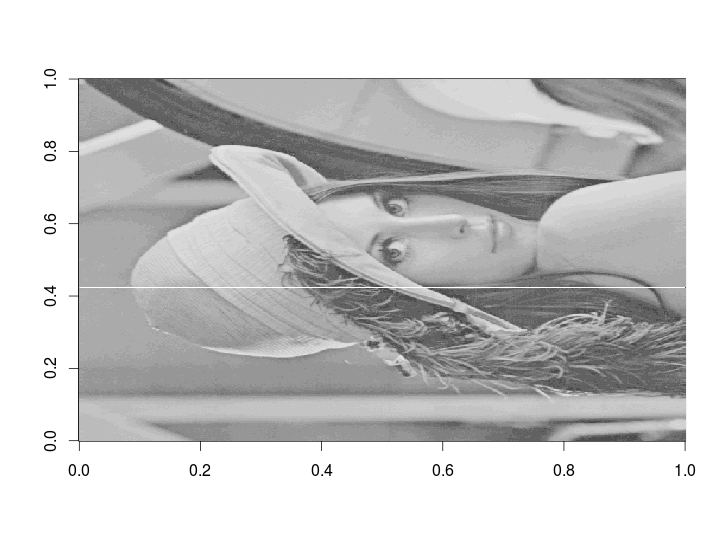
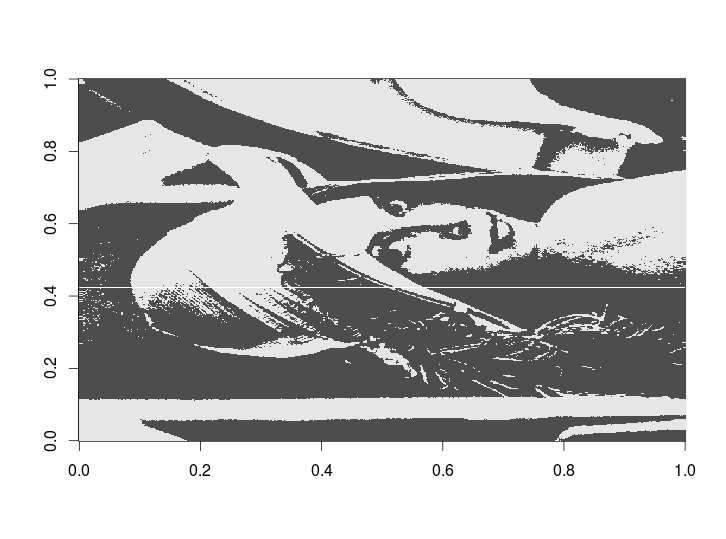


rotation 270° image originale

Quantification de paramètre k

Algorithme :

1. quantification <- function (image, k) {
2. image(image, col=gray.colors(2^k))
3. }



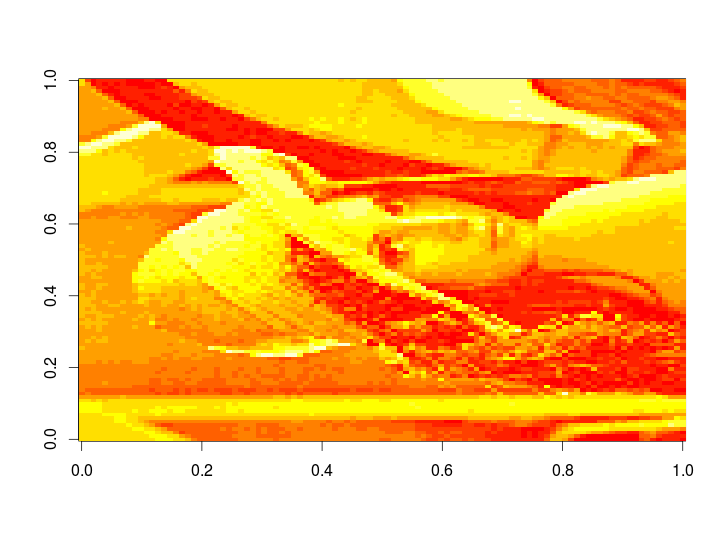
K = 1 K = 4

***Réduction de paramètre k***

Les pixels de l’image réduite correspondent aux pixels de l’image de départ qui intersectent les lignes et les colonnes retenues. Sachant que seule 1 ligne et 1 colonne sur k sont prises.

Algorithme :

1. reduction <- function (image, k) {
2. dimension <- dim(image)
3. ligne <- seq(1,dimension[1],k)
4. colonne <- seq(1,dimension[2],k)
5. res <- image[ligne, colonne]
6. return ( res);
7. }

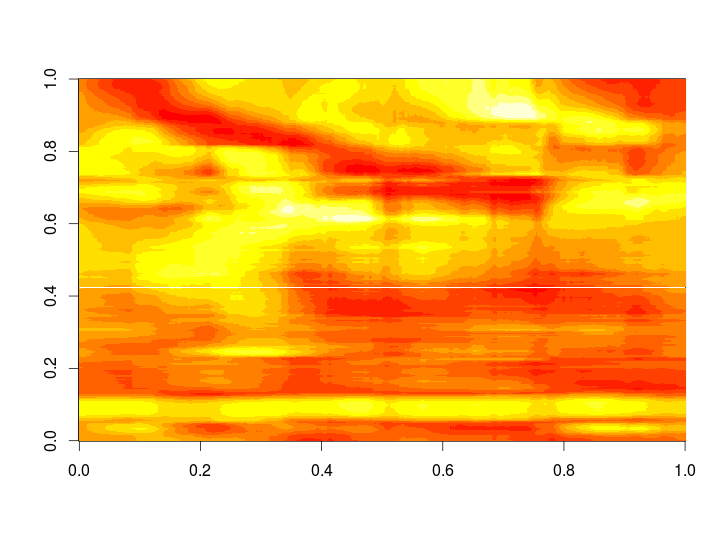


Réduction avec k = 5

***Compression svd de paramètre k***

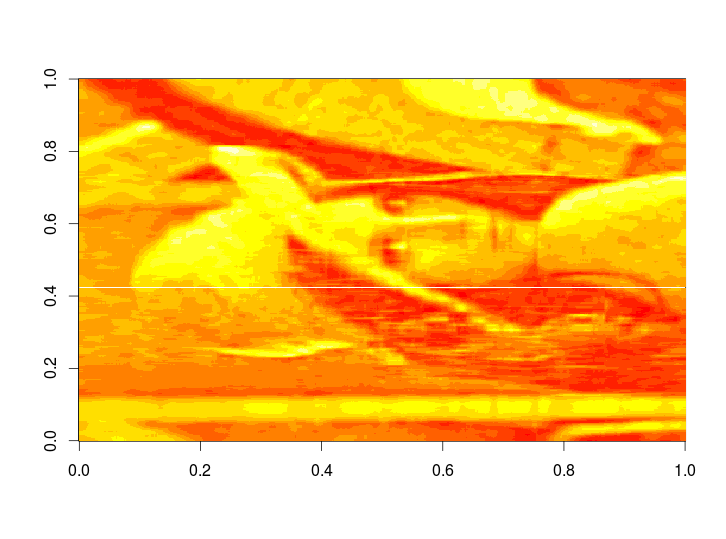
Algorithme :

1. svdp <- function (image, i) {
2. image.svd <- svd(image,i)
3. d <- image.svd$d
4. u <- image.svd$u
5. v <- image.svd$v
6. image.compressed <- u[,1:i] %\*% diag(d[1:i]) %\*% t(v[,1:i])
7. return( image.compressed )
8. }



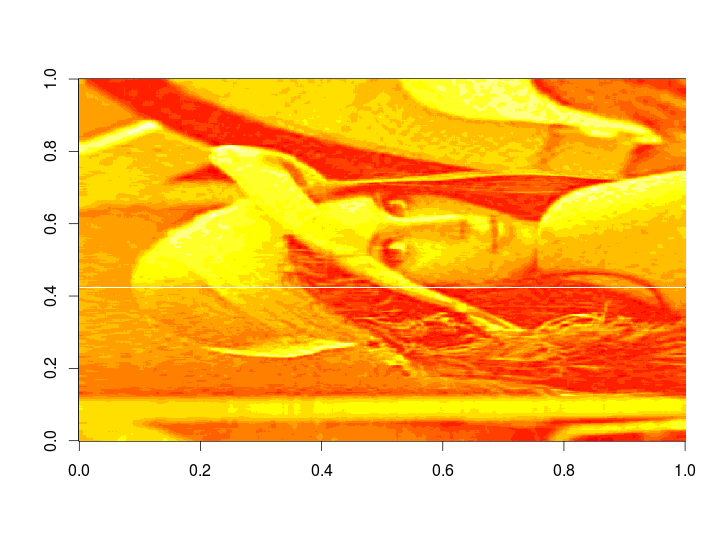
K = 5

Distance : 13696.26



K = 20

Distance : 7085.225



K = 40

Distance : 4737.341

Algorithme :

La distance euclidienne est utilisée pour calculer la distance entre 2 matrices :

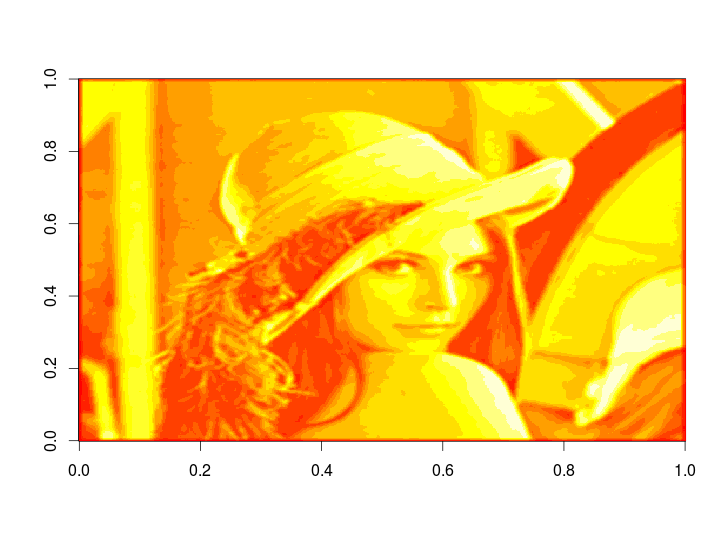
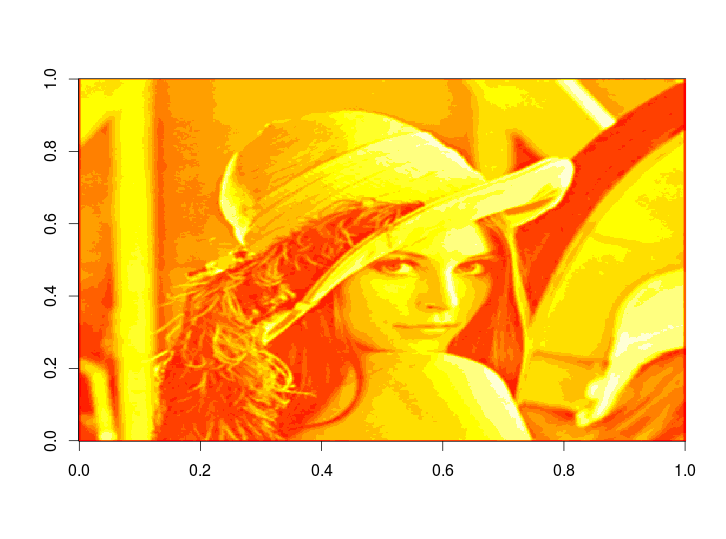
1. distance <- function(matrix1, matrix2) {
2. dim1 <- dim(matrix1)
3. dim2 <- dim(matrix2)
4. if (dim1[1] == dim2[1] && dim1[2] == dim2[2]) {
5. return ( sqrt( sum( (matrix1-matrix2)^2 )) )
6. } else {
7. return ( -1 )
8. }
9. }

Plus K augmente plus l’image compressée est proche (au sens de la distance) de celle d’origine.

***Image denoising***

Algorithme :

1. moyenneur <- function (image, taille) {
2. noyaux <- matrix( 1/(taille\*taille), 1, taille)
3. res <- convolve2D(image, noyaux,type="open")
4. res <- rotation (res, 90)
5. res <- convolve2D(res, noyaux,type="open")
6. return( res );
7. }



Noyau 3x3 Noyau 5x5