Ce TP est en très grande partie issue du livre "Informatique pour tous en classes préparatoires aux grandes écoles" de Wack et al. aux éditions Eyrolles

---

Exercice 1 :

Sur la page web du cours télécharger dans un répertoire personnel les fichier **image.py** et **chromosome.pgm**.

(1) Analyser le fichier **image.py**.

Il apparaît que pour beaucoup de traitement de l'image auront la même structure que la fonction **negatif**. On utilisera donc cette fonction comme modèle pour la suite.

(2) La fonction **lumiere** doit jouer sur la luminosité de l'image.
Pour cela, il suffit d'augmenter les valeurs des pixels de pour éclaircir l'image et de les diminuer pour l'assombrir
Écrire cette fonction qui prend en argument une image et un nombre et qui éclaire ou assombrit chaque pixel en fonction de x.

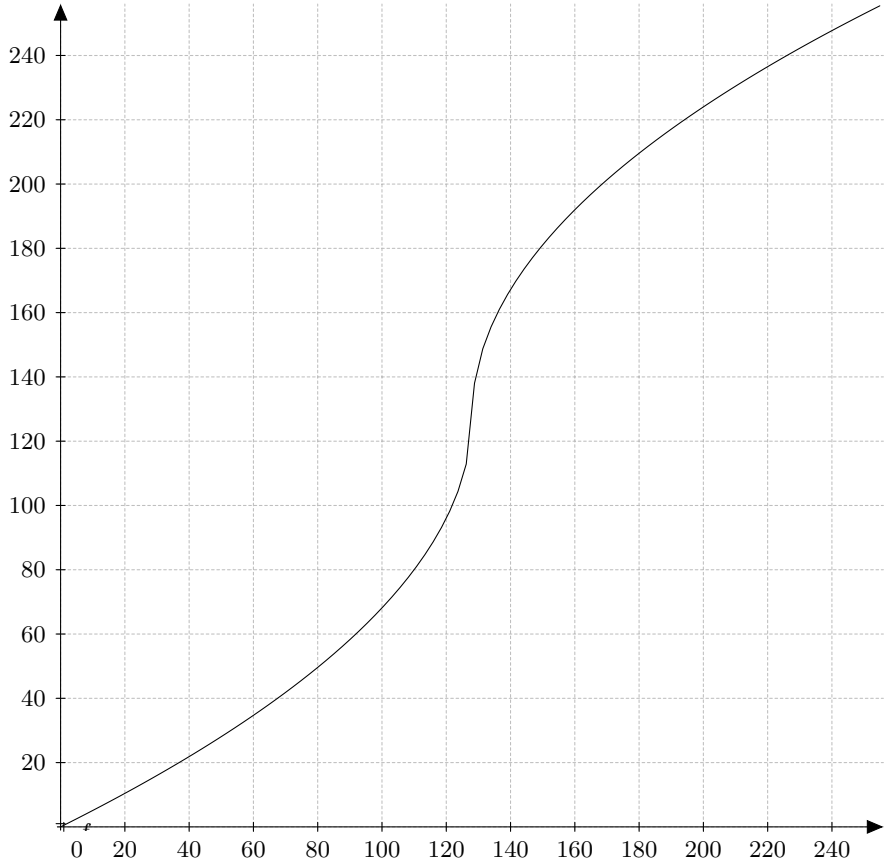
Avec $x = 50$, on s'aperçoit vite que l'effet obtenu n'est pas celui escompté : tous les pixels dont la valeur était supérieure à 205 reçoivent une valeur entre 255 et 305. Comme les valeurs des pixels sont comptées modulo 255, ces pixels deviennent en fait presque noirs.

(3) Modifier la fonction **lumiere** en s'assurant par un test que la valeur des pixels est plafonnée comprise entre 0 et 255.

(4) La solution précédente n'est pas satisfaisante car elle crée de grands aplats blancs dans les zones les plus claires (on pourra augmenter la valeur des pixels de 100 ou 150 pour s'en convaincre).
Au lieu d'utiliser une fonction affine, pour éclairer l'image, on va utiliser une fonction concave qui fait une bijection de $[0; 255]$ dans $[0; 255]$.
À l'aide de la fonction racine carrée, construire une bijection de $[0; 255]$ dans $[0; 255]$. Programmer cette fonction pour qu'elle manipule et renvoie des entiers (ce n'est alors évidemment plus une bijection).
En déduire la fonction **eclairer** qui prend une image et éclaire celle-ci.

(5) Construire de même la fonction **assombrir** pour assombrir une image.

(6) Voici le graphe de la fonction $f : x \mapsto 128 + \text{signe}(x - 128) \times \sqrt{128|x - 128|}$ sur $[0; 255]$.



À l'aide de ce graphe, prévoir l'effet qu'aura cette fonction sur l'image, puis tester votre hypothèse.

(7) Expérimenter avec des fonctions linéaires, non monotones, etc.

Exercice 2 :

On souhaite maintenant écrire des fonctions de transformation géométrique de l'image.

(1) Écrire la fonction **horizontale** qui effectue la symétrie de l'image par rapport à une droite horizontale (le haut se retrouve en bas)

(2) Écrire la fonction **rotation** qui effectue la rotation de 90 degré de l'image.

Attention pour cette fonction, les dimensions de l'image peuvent changer. En effet si l'image fait 200×50 *pixels, avec la rotation elle fera* 50×200 *pixels. On inversera donc bien la hauteur et l'image dans la déclaration du nouveau tableau.*

Exercice 3 :

Pour finir on va redimensionner l'image.

Encore une fois, les dimensions des images vont changer.

(1) Écrire la fonction **agrandissement** qui double la taille d'une image (quadruple le nombre de pixels). Cette fonction créera une image contenant deux fois plus de ligne et deux fois plus de colonnes. Elle remplacera chaque pixel par un bloc de 2×2 pixels.

(2) Écrire la fonction **reduction** qui divise par deux la taille d'une image. Chaque bloc de 2×2 pixels sera remplacé par un unique pixel dont la valeur est la moyenne des 4 pixels originaux.

Exercice 4 :

On s'intéresse maintenant à des traitements qui font intervenir plusieurs pixels voisins.

(1) Écrire une fonction qui produit une image floutée. Pour cela, il suffit de remplacer chaque pixel par la moyenne des pixels qui l'entourent.

On obtient différents degrés de floutage selon qu'on considère les 9 pixels immédiatement adjacents, ou bien un carré de 5 voire 7 pixels de côté centré sur le pixel à calculer.

(2) À l'inverse, on peut chercher des contours dans une image ; il s'agit de détecter les endroits où les pixels changent brutalement de couleur. Pour cela, on calcule pour chaque pixel une moyenne pondérée des pixels qui l'entourent, par exemple avec les coefficients :

-1	-1	-1
-1	8	-1
-1	-1	-1

Quel est le résultat de ce calcul pour un pixel qui est entouré d'autres pixels d'une couleur proche ? Et si au contraire il est d'une couleur nettement différente de celle d'une partie de ses voisins ?

Programmer cette méthode de détection des contours.

(3) Expérimenter avec d'autres coefficients. On retrouve ainsi encore d'autres fonctionnalités des logiciels de traitement d'image.

Exercice 5 :

On rappelle que pour les images en couleurs, chaque pixel est représenté par un triplet (r, v, b) de nombres compris entre 0 et 255 représentant successivement l'intensité du rouge, du vert et du bleu.

Écrire les fonctions suivantes permettants de transformer une image couleur en une image en niveau de gris.

(1) **Clarté** , où le niveau de gris de chaque pixel est la moyenne entre le minimum et le maximum des trois composantes RVB. Si par exemple $(R, V, B) = (122, 200, 147)$, cette moyenne vaut $(122+200)/2 = 161$, et le résultat est $(R, V, B) = (161, 161, 161)$.

(2) **Luminosité** , où chaque pixel devient $0,21 \times R + 0,71 \times V + 0,07 \times B$.

(3) **Moyenne**, où chaque pixel devient $(R + V + B)/3$.

(4) **Noir et blanc**, trouver une méthode pour que l'image devienne noir et blanc.

Exercice 6 :

On souhaite maintenant adapter l'exercice 1 aux images en couleurs.

(1) Adapter les fonctions de l'exercice 1 aux images en couleurs.

(2) Tester la fonction négative et accentuation des contours.

(3) Accentuer le niveau de rouge et atténuer le niveau de bleu et vert. Qu'obtenez-vous ?

Exercice 7 :

On va finir par créer des coloriages pour enfant.

(1) En utilisant les algorithmes précédents, proposer une méthode qui permet de partir d'une image couleur et d'obtenir une image ne contenant que les contours.

(2) Cette méthode est-elle satisfaisante ?

On souhaite donc maintenant à fermer les contours des images.

(3) Écrire la fonction **dilater** qui rend noir un pixel si au moins un de ses voisins est noir et le rend blanc sinon.

(4) Écrire la fonction **erosion** qui rend noir un pixel si tous ses voisins sont noirs et le rend blanc sinon.

(5) Appliquer **dilatation** puis **erosion** sur l'image. On crée alors une fermeture de l'image.

(6) Plutôt que de ne considérer que les 8 voisins, on peut se donner un paramètre r et faire le test sur tous les pixels de coordonnées d'abscisses comprises entre $[\![i-r;i+r]\!]$ et d'ordonnées comprises entre $[\![j-r;j+r]\!]$