X2011 – Corrigé Maple V9 par Laure Gonnord, Mai 2012

```
> restart;with(linalg):
```

```
Warning, the protected names norm and trace have been redefined and
unprotected
```

Fonctions utilitaires

```
> maximatrice:=proc(h,l,M) #renvoie le max des éléments d'une matrice
  positive ou nulle
    local i,j,maxi;
    maxi:=0;
    for i from 1 to h do
       for j from 1 to l do
          if M[i,j]>maxi then maxi:=M[i,j] end if;
       end do;
    od;

    RETURN(maxi);

  end;
```

maximatrice := **proc**(*h*, *l*, *M*)
local *i*, *j*, *maxi*;
maxi := 0;
for *i* **to** *h* **do**
for *j* **to** *l* **do** **if** *maxi* < *M*[*i*, *j*] **then** *maxi* := *M*[*i*, *j*] **end if** **end do**
end do
RETURN(*maxi*)
end proc

```
> mat:=matrix(2,2,[5,7,12,0]);
```

$$mat := \begin{bmatrix} 5 & 7 \\ 12 & 0 \end{bmatrix}$$

```
> maximatrice(2,2,mat);
```

12

Une image sera considérée comme un record de H,L,P et la matrice des pixels M

```
> allouer:=proc(h,l,p)
  print(h,l,p);
  return(Record('H'=h,'L'=l,'P'=p,'M'=matrix(h,l)));
  end;
```

allouer := **proc**(*h*, *l*, *p*)
print(*h*, *l*, *p*);
return *Record*('H' = *h*, 'L' = *l*, 'P' = *p*, 'M' = '*matrix*'(*h*, *l*))
end proc

```
> nouvelle:=allouer(1,16,15);
```

1, 16, 15

nouvelle := *module* () *export H, L, P, M; option record; end module*

```
> nouvelle:-M := matrix(1,16,[15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0]);
```

M := [[15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]]

```
> imprimeimage:=proc(im,b) #b = true ssi on veut imprimer les pixels
     print("taille :");
     print(im:-H,im:-L);
     print("profondeur :");
     print(im:-P);
     if b then
     print("pixels:");
     print(evalm(im:-M));
     end if;
  end;
```

imprimeimage := **proc**(*im*, *b*)
print("taille :");
print(*im*:-*H*, *im*:-*L*);
print("profondeur :");
print(*im*:-*P*);
if *b* **then** *print*("pixels:"); *print*(*evalm*(*im*:-*M*)) **end if**
end proc

```
> imprimeimage(nouvelle,true);
```

"taille :"
1, 16
"profondeur :"
15
"pixels:"
[[15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]]

```
>
```

Ecrivons maintenant la fonction de chargement d'une images à partir d'un fichier donné par son nom
source :
http://www.math.sciences.univ-nantes.fr/~guilleme/enseignement/m4/intromaple.pdf un peu modifié

```
> loadpgm:=proc(f) # f = nom_de_fichier
  local lg,haut,larg,dims,l,m,p;
  lg:=readline(f); # marque P5
  lg:=readline(f);
  while (substring(lg,1..1) = "#") do # les commentaires
     lg:=readline(f);
  od;

  dims:=sscanf(lg,"%d %d"); # les dimensions
  haut:=dims[2]; larg:=dims[1];
```

```
  lg:=readline(f); # le nombre de nuances
  l:=readbytes(f,infinity); # les pixels

  m:=matrix(haut,larg,l);
  p:=maximatrice(haut,larg,m);

  return(Record('H'=haut,'L'=larg,'P'=p,'M'=m));
  end:
```

```
> image:=loadpgm("mandrill.pgm"):
```

```
> imprimeimage(image,false);
```

"taille :"

64, 64

"profondeur :"

233

```
> sauvepgm:=proc(im,f) #fonction image record --> image pgm sur le dd
  local canal,m,i,haut,mat2;
  m:=im:-M;
  haut:=im:-H;
  canal:=fopen(f,WRITE);
  fprintf(canal,"P5\n%d %d\n255\n",im:-L,haut);

  #conversion de la matrice vers une matrice entre 0 et 255

  mat2:=map(v->v*floor(255/im:-P),m);


  for i from 1 to haut do
     writebytes(canal,convert(row(eval(mat2),i),list));
  od;

  fclose(canal);
  end:
```

```
> sauvepgm(image,"copieMandrill.pgm"):
```

Partie I

```
> inverser:=proc(im)
     local new,l,c;
     new:=allouer(im:-H,im:-L,im:-P);

     for l from 1 to im:-H do
       for c from 1 to im:-L do
           new:-M[l,c]:=(im:-P - im:-M[l,c]);
        end do;
     end do;
     return new;
  end:
```

Pour inverser une image on en cree une nouvelle (mêmes H, L, P) et on initialise les pixels new[l,c] = P - im[l,c]

```
> imB:=inverser(image);
```

64, 64, 233

imB := *module () export H, L, P, M; option record; end module*

```
> imprimeimage(imB,false);
```

"taille :"

64, 64

"profondeur :"

233

```
> sauvepgm(imB,"imB.pgm"):
```

Pour le flip, c'est presque pareil, sauf que new[l,c]= im[l,-c+1]

```
> flipH:=proc(im)
  local new,l,c;
   new:=allouer(im:-H,im:-L,im:-P);

     for l from 1 to im:-H do
       for c from 1 to im:-L do
           new:-M[l,c]:=(im:-M[l,im:-L - c+1]);
        end do;
     end do;
     return new;
  end:
```

```
> imC:=flipH(image);
```

64, 64, 233

imC := *module () export H, L, P, M; option record; end module*

```
> sauvepgm(imC,"imC.pgm"):
```

POur poserV, l'image nouvelle est de dimensions H=2*H(im), L, et la profondeur le max des profondeurs des deux images.
ensuite on copie la premiere image directement : new[l,c]=im1[l,c] et la deuxième on décale : new[H1+l,c]=im2[l,c]

```
> poserV:=proc(im1,im2)
  local new,l,c,prof;
   prof := max(im1:-P,im2:-P);
   new:=allouer(2*im1:-H,im1:-L,prof);

     for l from 1 to im1:-H do #copie im1
       for c from 1 to im1:-L do
           new:-M[l,c]:=(im1:-M[l,c])
        end do;
     end do;

    for l from 1 to im2:-H do #copie im2
       for c from 1 to im2:-L do
           new:-M[im1:-H+l,c]:=(im2:-M[l,c])
```

```
        end do;
    end do;

    return new;
end:
```

```
> imD:=poserV(imB,imC);
```

128, 64, 233

imD := *module () export H, L, P, M; option record; end module*

```
> sauvepgm(imD,"imD.pgm"):
```

de même ... dans l'autre sens

```
> poserH:=proc(im1,im2)
  local new,l,c,prof;
   prof := max(im1:-P,im2:-P);
   new:=allouer(im1:-H,2*im1:-L,prof);

     for l from 1 to im1:-H do #copie im1 telle qu'elle
       for c from 1 to im1:-L do
           new:-M[l,c]:=(im1:-M[l,c])
        end do;
     end do;

    for l from 1 to im2:-H do #copie im2 + décalage
       for c from 1 to im2:-L do
           new:-M[l,c+im1:-L]:=(im2:-M[l,c])
        end do;
     end do;

     return new;
  end:
```

```
> imE:=poserH(imB,imC);
```

64, 128, 233

imE := *module () export H, L, P, M; option record; end module*

```
> sauvepgm(imE,"imE.pgm"):
>
```

Partie II Transferts

Q5 la fonction est construite de la même façon que précédemment, le pixel l,c de la nouvelle image étant maintenant t[im.M[l,c]]

```
> transferer:=proc(im,pp,t)

  local new,l,c;
     new:=allouer(im:-H,im:-L,pp);

     for l from 1 to im:-H do
       for c from 1 to im:-L do
           new:-M[l,c]:=t[im:-M[l,c]];
        end do;
     end do;
     return new;

  end;
```

transferer := **proc**(*im*, *pp*, *t*)
local *new*, *l*, *c*;
  new := *allouer*(*im*:-*H*, *im*:-*L*, *pp*);
  for *l* **to** *im*:-*H* **do**
    for *c* **to** *im*:-*L* **do** (*new*:-*M*)[*l*, *c*] := *t*[(*im*:-*M*)[*l*, *c*]] **end do**
  end do
  return *new*
end proc

```
>
```

Pour utiliser la fonction précédente il faut créer un tableau adéquat. Ici c'est t=[P,P-1.....0] (tiens, pourquoi ?)

```
> inverser2:=proc(im)
  local t,pp,i;
  pp:=im:-P;
  t:=array(0..pp);

  for i from 0 to pp do
    t[i] := pp-i;
  end do;

  return(transferer(im,pp,t));

  end;
```

inverser2 := **proc**(*im*)
local *t*, *pp*, *i*;
  pp := *im*:-*P*;
  t := *array*(0 .. *pp*);
  for *i* **from** 0 **to** *pp* **do** *t*[*i*] := *pp* – *i* **end do**
  return *transferer*(*im*, *pp*, *t*)
end proc

```
> imBbis:=inverser2(image);
```

64, 64, 233

imBbis := *module () export H, L, P, M; option record; end module*

```
> imprimeimage(imBbis,false);
```

"taille :"

64, 64

"profondeur :"

233

```
> sauvepgm(imBbis,"imBbis.pgm"):
>
```

Question 7 L'histogramme se contruit en parcourant la matrice de l'image. En maple, on est obligé de faire une copie du tableau argument pour pouvoir le retourner :-(

```
> histogramme := proc(im,h)
  local tab,pp,i,l,c,mat;

  mat:=im:-M;

  pp:=im:-P; #pour pouvoir parcourir
  #on commence par mettre à 0 chacune des cases de tab

  for i from 0 to pp do
    h[i] := 0;
  end do;

   for l from 1 to im:-H do #parcourt de la matrice
       for c from 1 to im:-L do
           h[mat[l,c]]:=h[mat[l,c]]+1;
        end do;
     end do;

  return;

  end:
```

```
> image:-P;
```

$$233$$

```
> h:=array(0..image:-P);
```

$$h := array(0 .. 233, [])$$

```
> histogramme(image,h);#modifie h
```

```
> h[233];
```

$$1$$

```
> print(h);
```

$$ARRAY([0 .. 233],$$
$$[(0) = 0, (1) = 0, (2) = 0, (3) = 0, (4) = 0, (5) = 0, (6) = 0, (7) = 0, (8) = 0, (9) = 0,$$
$$(10) = 0, (11) = 0, (12) = 0, (13) = 0, (14) = 0, (15) = 0, (16) = 0, (17) = 0, (18) = 0,$$
$$(19) = 1, (20) = 0, (21) = 0, (22) = 0, (23) = 1, (24) = 1, (25) = 0, (26) = 1, (27) = 1,$$
$$(28) = 0, (29) = 0, (30) = 0, (31) = 0, (32) = 2, (33) = 0, (34) = 0, (35) = 0, (36) = 1,$$
$$(37) = 0, (38) = 0, (39) = 1, (40) = 1, (41) = 1, (42) = 1, (43) = 3, (44) = 1, (45) = 3,$$
$$(46) = 2, (47) = 2, (48) = 0, (49) = 3, (50) = 2, (51) = 1, (52) = 4, (53) = 1, (54) = 6,$$
$$(55) = 6, (56) = 7, (57) = 5, (58) = 5, (59) = 3, (60) = 5, (61) = 5, (62) = 7, (63) = 3,$$
$$(64) = 9, (65) = 7, (66) = 8, (67) = 8, (68) = 9, (69) = 8, (70) = 9, (71) = 10, (72) = 4,$$
$$(73) = 14, (74) = 12, (75) = 9, (76) = 10, (77) = 5, (78) = 10, (79) = 10, (80) = 12,$$
$$(81) = 10, (82) = 6, (83) = 11, (84) = 6, (85) = 16, (86) = 11, (87) = 9, (88) = 19,$$
$$(89) = 6, (90) = 15, (91) = 16, (92) = 17, (93) = 12, (94) = 18, (95) = 19, (96) = 10,$$
$$(97) = 15, (98) = 17, (99) = 14, (100) = 16, (101) = 19, (102) = 21, (103) = 30,$$
$$(104) = 24, (105) = 12, (106) = 22, (107) = 21, (108) = 21, (109) = 16, (110) = 25,$$
$$(111) = 22, (112) = 29, (113) = 21, (114) = 27, (115) = 35, (116) = 30, (117) = 27,$$
$$(118) = 43, (119) = 30, (120) = 26, (121) = 36, (122) = 34, (123) = 22, (124) = 44,$$
$$(125) = 42, (126) = 38, (127) = 51, (128) = 54, (129) = 48, (130) = 45, (131) = 41,$$
$$(132) = 46, (133) = 56, (134) = 40, (135) = 49, (136) = 53, (137) = 39, (138) = 39,$$
$$(139) = 38, (140) = 30, (141) = 48, (142) = 59, (143) = 29, (144) = 38, (145) = 50,$$
$$(146) = 37, (147) = 32, (148) = 40, (149) = 37, (150) = 46, (151) = 39, (152) = 44,$$
$$(153) = 37, (154) = 35, (155) = 31, (156) = 41, (157) = 30, (158) = 34, (159) = 43,$$
$$(160) = 27, (161) = 35, (162) = 33, (163) = 28, (164) = 27, (165) = 37, (166) = 34,$$
$$(167) = 35, (168) = 39, (169) = 34, (170) = 36, (171) = 28, (172) = 31, (173) = 32,$$
$$(174) = 20, (175) = 29, (176) = 34, (177) = 31, (178) = 29, (179) = 31, (180) = 25,$$
$$(181) = 31, (182) = 42, (183) = 23, (184) = 25, (185) = 31, (186) = 22, (187) = 22,$$
$$(188) = 19, (189) = 32, (190) = 25, (191) = 21, (192) = 23, (193) = 19, (194) = 22,$$
$$(195) = 22, (196) = 19, (197) = 17, (198) = 22, (199) = 26, (200) = 20, (201) = 28,$$
$$(202) = 24, (203) = 34, (204) = 29, (205) = 16, (206) = 19, (207) = 17, (208) = 22,$$
$$(209) = 17, (210) = 14, (211) = 22, (212) = 11, (213) = 9, (214) = 15, (215) = 9,$$
$$(216) = 6, (217) = 14, (218) = 13, (219) = 6, (220) = 6, (221) = 0, (222) = 7,$$
$$(223) = 6, (224) = 2, (225) = 5, (226) = 3, (227) = 3, (228) = 0, (229) = 0, (230) = 0,$$
$$(231) = 1, (232) = 0,$$
$$(233) = 1])$$

Question 8 : egaliser. 1-calcul de vmin 2- alloc d'un tableau pour la fonction de transfert, il est de taille im.P+1 , les cases 0 à vmin -1 sont à 0,
les cases de vmin à P sont à calculer avec la formule de l'énoncé 3-appeler transferer

```
> egaliser := proc(im)
  local vmin,histo,hvmin,coeffmul,sum,v,tab,i;

  histo:=array(0..im:-P);
  histogramme(im,histo);
  #vmin est le premier indice de histo avec histo[i]!=0
  i:=0;
  while(histo[i]=0) do
      i:=i+1;
  end do;
  vmin :=i;
  hvmin:=histo[vmin];
  print(vmin); #doit afficher 19

  tab:=array(0..im:-P); #tableau pour le transfert, j'initialise à 0
  pour les v undef
  for i from 0 to vmin-1 do
    tab[i] := 0;
  end do;

  #calcul de v' pour v entre vmin et image:-P

  coeffmul := im:-P / (im:-H * im:-L - hvmin);
  print(coeffmul);
  sum:=0; #la somme du haut peut se calculer de proche en proche à
  partir de vmin


  for v from vmin to im:-P do
     sum:=sum+histo[v];
     tab[v]:=round(coeffmul*(sum-hvmin));
  end do;
```

```
  transferer(im,im:-P,tab);
  end:
```

```
> imEq:=egaliser(image);
```

$$\frac{19}{\frac{233}{4095}}$$

64, 64, 233

imEq := *module () export H, L, P, M; option record; end module*

```
> sauvepgm(imEq,"imEq.pgm"):
```

Q9 : dans ce cas la formule donne une erreur(div par 0). Normalement, elle devrait rendre une image blanche... pixels = P

Q10 : ici encore il faut trouver le bon tableau tab[i]=arrondi(i*P'/P)

```
> reduire:=proc(im,pprime)
  local tab,i;
  tab:=array(0..im:-P);
  for i from 0 to im:-P do
    tab[i] := round(i*pprime/im:-P);
   # print(tab[i]);
  end do;

  transferer(im,pprime,tab);

  end:
```

```
> imG:=reduire(image,1):
```

64, 64, 1

```
> sauvepgm(imG,"imG.pgm"):
```

```
> imH:=reduire(image,4):sauvepgm(imH,"imH.pgm"):
```

64, 64, 4

```
> imI:=reduire(image,16):sauvepgm(imI,"imI.pgm"):
```

64, 64, 16

```
>
```

Partie III

```
> imJ:=allouer(4,1,3);
```

4, 1, 3

imJ := *module () export H, L, P, M; option record; end module*

```
> imJ:-M:=matrix(4,1,[0,1,2,3]);
```

$$M := \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \end{bmatrix}$$

```
> imprimeimage(imJ,true);
```

"taille :"

4, 1

"profondeur :"

3

"pixels:"

$$\begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \end{bmatrix}$$

```
> deuxQuarts:=allouer(4,4,15);
```

4, 4, 15

deuxQuarts := *module () export H, L, P, M; option record; end module*

```
> deuxQuarts:-M:=matrix(4,4,[1,5,10,14,3,7,8,12,13,9,6,2,15,11,4,0]);
```

$$M := \begin{bmatrix} 1 & 5 & 10 & 14 \\ 3 & 7 & 8 & 12 \\ 13 & 9 & 6 & 2 \\ 15 & 11 & 4 & 0 \end{bmatrix}$$

```
> imprimeimage(deuxQuarts,true);
```

"taille :"

4, 4

"profondeur :"

15

"pixels:"

$$\begin{bmatrix} 1 & 5 & 10 & 14 \\ 3 & 7 & 8 & 12 \\ 13 & 9 & 6 & 2 \\ 15 & 11 & 4 & 0 \end{bmatrix}$$

```
>
```

Pour tramer on met à 0 ou à 1 le pixel suivant le pixel correspondant dans l'image de trame (utilisation de modulo)

```
> tramer:=proc(im,imM)
    local new,l,c;
     new:=allouer(im:-H,im:-L,1);

     for l from 1 to im:-H do
       for c from 1 to im:-L do
           if (im:-M[l,c]>imM:-M[1+(l mod imM:-H),1+(c mod imM:-L)])
  then
              new:-M[l,c]:=1
           else
              new:-M[l,c]:=0
   end if;
        end do;
     end do;
     return new;

  end:
```

```
> imL:=tramer(imH,imJ);#attention, on teste sur l'image H !
```

64, 64, 1

imL := *module () export H, L, P, M; option record; end module*

```
> sauvepgm(imL,"imL.pgm"):
```

Q12, on utilise la fonction précédente avec une image que l'on construit ç l'intérieur de la fonction

```
> tramerTV:=proc(im)
  local imM,p,i;

  p:=im:-P;

  print(p);

  imM:=allouer(p,1,p-1); #hauteur P largeur 1 profondeur P-1
  for i from 1 to p do
     imM:-M[i,1]:=i-1;
  end do;

  tramer(im,imM);

  end:
```

```
> toto:=tramerTV(imH);
```

4

4, 1, 3

64, 64, 1

toto := *module () export H, L, P, M; option record; end module*

```
> sauvepgm(toto,"toto.pgm"):
>
```

flemme du correcteur