



ANALYSE D'IMAGE

Rapport de TD3

Guénon Marie et Favreau Jean-Dominique
VIM / Master SSTIM

Table des matières

Algorithme de Metropolis.....	2
Calcul des énergies.....	2
Mise à jour.....	3
Tests	4
4-connexité.....	4
8-connexité.....	6
Cerveau	7
Amélioration de la dynamique de l'histogramme.....	8
Redéfinition des classes	9
Annexes	11
recruit.m.....	11
param_cerveau	15

Algorithme de Metropolis

Le but de ce TP est de classer chaque pixel d'une image pour identifier des zones caractéristiques que l'on espère homogènes. Pour chaque pixel, on donne une classe à priori. Ensuite, pour chaque pixel on tire de manière aléatoire une nouvelle classe et on calcule l'énergie associée à cette nouvelle classe et l'énergie associée à l'ancienne classe. Le but est de minimiser l'énergie associée au pixel.

Si l'énergie nouvelle est plus faible, on garde la nouvelle classe du pixel, sinon on la garde de manière aléatoire en étant de plus en plus restrictif au fur et à mesure du temps.

Nous allons donc commencer par nous intéresser au calcul de l'énergie, puis nous verrons le critère que nous utilisons pour décider de conserver la classe ou non et nous finirons par quelques tests.

Calcul des énergies

L'énergie correspondant à fidélité aux données (vraisemblance) s'écrit :

$$P(X|Y) = \prod_s p(x_s|y_s)$$

L'énergie du modèle à priori s'écrit :

$$Energie_a_priori = \beta \delta(y_s \neq y_{s'})$$

D'où l'énergie totale (en utilisant la 4-connexité) s'écrit :

$$U = \beta \left(\delta(y_s \neq y_{(i-1,j)}) + \delta(y_s \neq y_{(i+1,j)}) + \delta(y_s \neq y_{(i,j-1)}) + \delta(y_s \neq y_{(i,j+1)}) + \frac{(x_s - \mu_l)^2}{2\sigma_l^2} + \log(\sigma_l) \right)$$

$$U = 4_connexité + vraisemblance$$

On a une formule analogue pour la 8-connexité.

A partir de ces formules, nous obtenons le code suivant :

```
#####4-connexe
energie_courante = energie_courante + beta * sum(valeur_courante ~= value_around([2,4,6,8]));
energie_nouvelle = energie_nouvelle + beta * sum(valeur_nouvelle ~= value_around([2,4,6,8]));

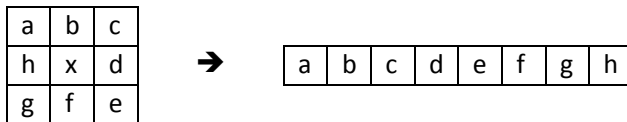
#####8-connexe
tab_dif=(valeur_courante~=value_around');
tmp_tri = tab_dif([2,4,6,8])|tab_dif([4,6,8,2]);
energie_courante = energie_courante + beta*(sum(tab_dif + (tab_dif | circshift(tab_dif,1)))
+ sum(tmp_tri + (tmp_tri | tab_dif([3,5,7,1]))));

tab_dif=(valeur_nouvelle~=value_around');
tmp_tri = tab_dif([2,4,6,8])|tab_dif([4,6,8,2]);
energie_nouvelle = energie_nouvelle + beta*(sum(tab_dif + (tab_dif | circshift(tab_dif,1)))
+ sum(tmp_tri + (tmp_tri | tab_dif([3,5,7,1]))));
```

```
% - Energie due a la vraisemblance
energie_courante = V1(valeur_courante+1,dep(j,k)+1);
energie_nouvelle = V1(valeur_nouvelle+1,dep(j,k)+1);
```

Les conditions aux limites sont gérées au préalable, avec le tableau value_around :

Ce tableau contient la valeur de tous les voisins de x, s'ils existent, sinon il contient -1. Et on déplie de la manière suivante :



Mise à jour

Comme nous l'avons déjà vu, si l'énergie nouvelle est plus faible, on garde la nouvelle classe du pixel, sinon on la garde de manière aléatoire en étant de plus en plus restrictif au fur et à mesure du temps.

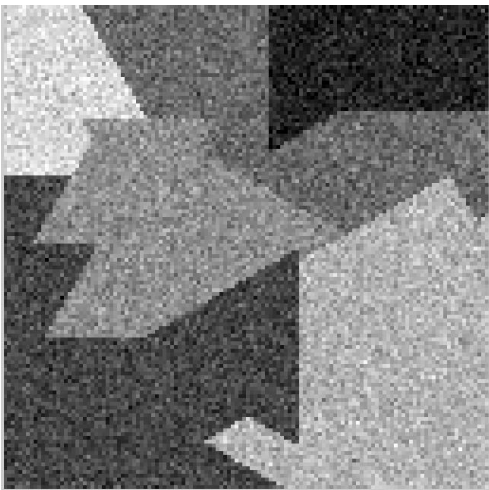
```
if(energie_nouvelle<=energie_courante || (rand<exp(-(energie_nouvelle-energie_courante)/tempe)))
    res(j,k)=valeur_nouvelle;
end
```

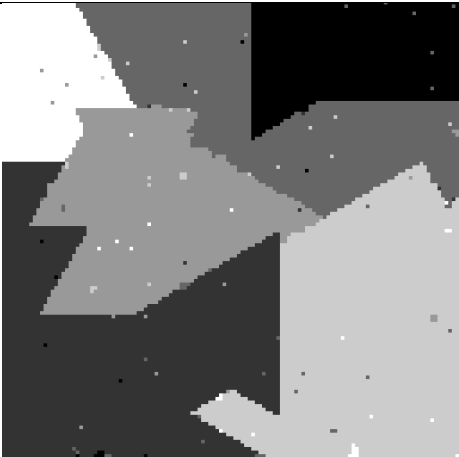

La première partie de la condition ci-dessus correspond à ce qu'on aurait dans le cas d'un algorithme variationnel (type descente de gradient...). La deuxième partie correspond à la partie stochastique de l'algorithme et est là pour permettre de visiter des "endroits" que l'on ne visiterait jamais avec un algorithme variationnel classique, ce qui nous permet d'atteindre le minimum global et pas seulement un minimum local.




Tests

4-connexité

Image initiale :



β	<i>tempe</i>	<i>decrease</i>	<i>NIT</i>	résultat	commentaire
1.0	1.0	0.99	150		On voit que les zones sont plutôt bien séparées, mais qu'il reste du bruit à l'intérieur de celles-ci.
5.0	2.0	0.99	250		Il y a moins de bruits ici mais les erreurs sont produites sur des zones plus larges et généralement sur les contours. Cela est dû au fait qu'on donne moins de poids à la vraisemblance.

2.0	0.1	0.99	150		Nous pouvons voir ici qu'il y a beaucoup plus de bruit. Ceci est due au fait que la température initiale est faible est nous acceptons moins de variations et nous convergeons donc plus vers des minimaux locaux.
2.0	1.0	0.9	150		Le résultat ici est similaire au cas précédent mais pour des raisons différentes. Le decrease est plus faible, et nous faisons donc diminuer la température plus rapidement, ce qui nous limite les sauts aléatoires plus vite.
2.0	1.0	0.99	150		Nous avons ici un résultat plutôt correct, même si nous pouvons constater quelques irrégularités sur les contours.



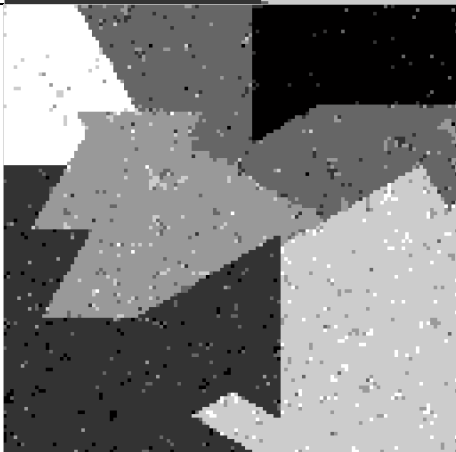
β = Paramètre du modèle de Potts, force des interactions entre pixels (fidélité)

tempe = température initiale

decrease = Coefficient de décroissance de la température (erreur que l'on se permet)

NIT = nombre d'itérations

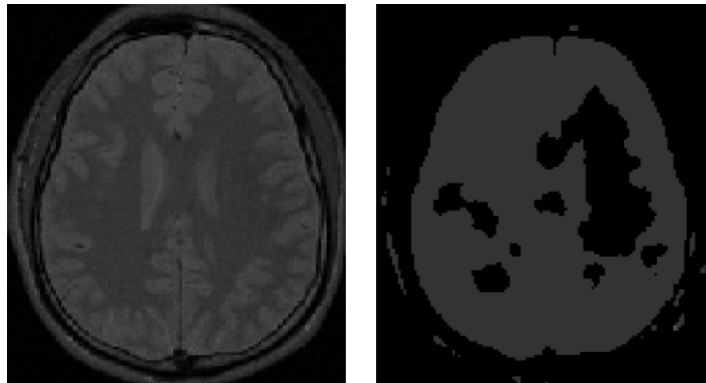
8-connexité

β	tempe	decrease	NIT	résultat	commentaire
2.0	1.0	0.99	150		Le β est démultiplié par tous les clics de la 8-convexité et donne donc trop peu d'importance à la valeur initiale du pixel. On remarque par ailleurs, que les problèmes sont surtout situés sur les angles et les contours.
0.5	1.0	0.99	150		Nous avons ici diminué β et donc donné plus d'importance à la fidélité des données. Et nous pouvons remarquer que nous avons de bons résultats malgré une petite erreur au milieu à gauche.
0.1	1.0	0.99	150		Nous avons continué à diminuer β . Et nous pouvons constater ici que nous n'accordons maintenant plus assez d'importance au model à priori.

Cerveau

Pour continuer nos test en 8-connexité, nous avons appliqué notre algorithme sur l'image du cerveau qui nous avait été fournie avec les valeurs qui avaient été les plus concluantes lors des tests précédents, c'est-à-dire $\beta=0.5$, $\text{tempe}=1.0$, $\text{decrease}=0.99$ et $\text{NIT}=150$.

Nous obtenons alors les résultats suivants, avec à gauche l'image initiale :



Nous pouvons constater que le résultat n'est pas concluant. Ceci est due au fait qu'il n'y a pas assez de contraste dans l'image d'origine pour pouvoir reconnaître les 6 classes de couleurs qui ont été définies.

Pour résoudre ce problème, il y a deux solutions envisageables :

- Améliorer la dynamique de l'histogramme de l'image à traiter pour correspondre au mieux aux classes de couleurs définies.
- Redéfinir les classes de couleurs et en choisir de plus adaptées aux niveaux de gris présents dans cette image

Nous avons testé ces deux solutions possibles et les résultats sont présentés ci-après.

Amélioration de la dynamique de l'histogramme

Dans ce test, nous avons cherché à améliorer nos résultats sur la segmentation de l'image du cerveau en changeant la dynamique de l'histogramme pour correspondre au mieux aux classes de couleurs définies. Pour cela, nous avons étalé l'histogramme original de l'image (à gauche sur les illustrations suivantes) pour obtenir une image avec une meilleure répartition des couleurs.

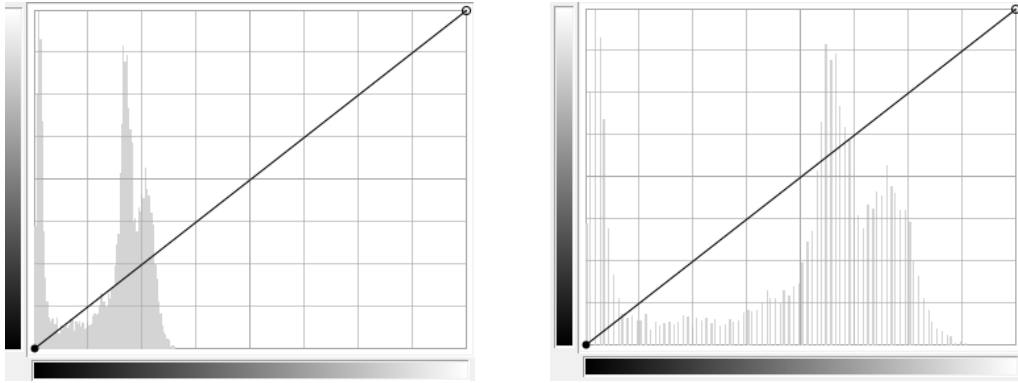
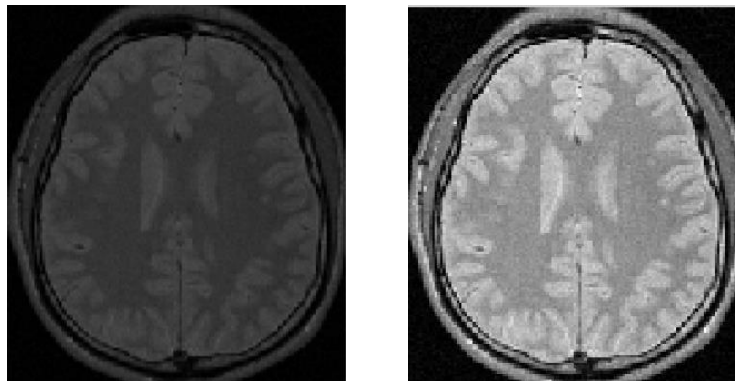
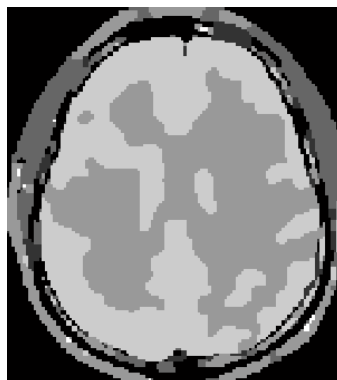


Image obtenue après amélioration de la dynamique de l'histogramme (originale à gauche) :



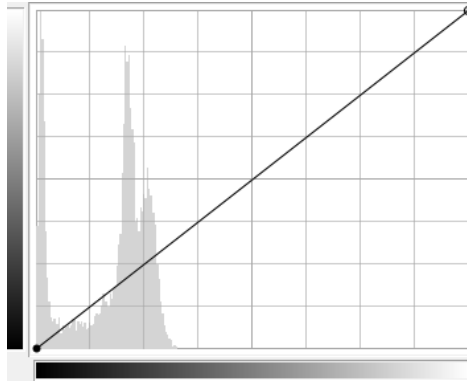
Résultat obtenu après segmentation sur l'image obtenue après amélioration de l'histogramme :



Nous pouvons constater que nous avons un meilleur résultat que dans le cas précédent : plus de classes sont identifiables sur le résultat. Cependant la définition des bords des classes à l'intérieur du cerveau reste assez grossière et mériterait un raffinement.

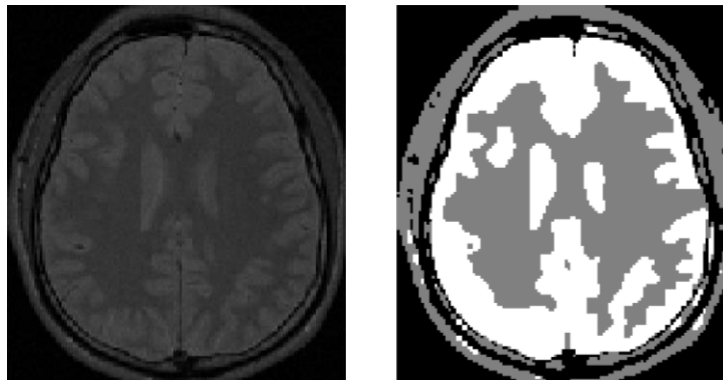
Redéfinition des classes

Dans ce test, pour améliorer les résultats obtenus précédemment, nous avons choisi de redéfinir les classes de couleurs et en choisir de plus adaptées aux niveaux de gris présents dans cette image. Nous avons pu remarquer trois pics principaux dans l'histogramme dans l'image d'origine du cerveau, ce qui dénote une concentration particulière de pixels de ces couleurs dans l'image.



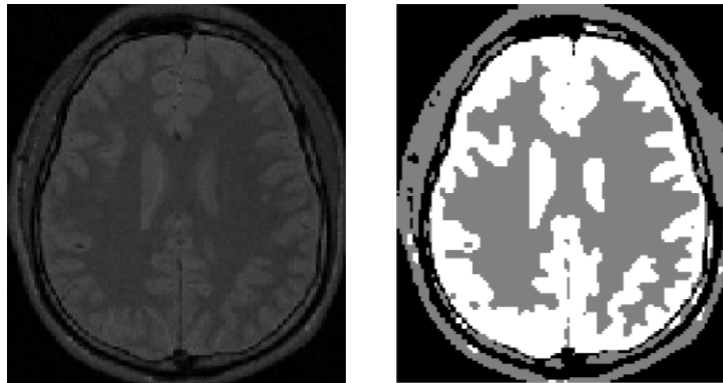
Ces pics se situent en 5, en 145 et en 177 (valeurs données en niveau de gris). Il s'agit donc de créer trois classes centrées en ces valeurs et avec une certaine variance autour de celles-ci.

Pour des variances de 600, 200 et 200 (appliquées dans l'ordre aux pics 5, 145 et 177), nous avons obtenu le résultat suivant :



Nous pouvons voir que nous avons déjà de meilleurs résultats que dans le test précédent (avec l'amélioration d'histogramme). En effet, nous avons plus de détails sur la différence entre la substance blanche et la matière grise. Cependant, il reste encore des zones de la matière grise (en blanc ici, pic à 177) qui sont définies de manière trop large sur notre résultat. C'est pourquoi nous avons relancé un test en réduisant la valeur de la variance sur le dernier pic.

Pour des variances de 600, 200 et 100 (appliquées dans l'ordre aux pics 5, 145 et 177), nous avons obtenu le résultat suivant :



Nous pouvons constater ici que le résultat obtenu est bien meilleur et que les zones de substance blanche matières grises sont beaucoup plus proches de la réalité. Les différents replis du cerveau sont plutôt bien soulignés ici et même si l'on pourrait probablement encore améliorer nos résultats en faisant varier nos différents paramètres (β , *tempe*, *decrease*, NIT et variance), ceci demanderait beaucoup de temps et de test. En effet, nous devrions les faire varier un par un et constater ou non une amélioration souvent minime.

Nous pouvons donc dire que nous avons ici un résultat très satisfaisant qui permet déjà de faire une première analyse de l'état du cerveau.

Annexes

recruit.m

```
function res=recuit(name1,name2,name3,beta,tempe0,decrease,NIT)
%-----
% function res=recuit(name1,name2,name3,beta,tempe,decrease,NIT)
%
% Inputs : name1 -> Nom de l'image (.ima) à segmenter (si l'image à traiter
%              s'appelle test.ima l'appel se fait avec name1='test')
%              name2 -> Nom du fichier texte contenant la définition des
%              classes utilisée pour la segmentation
%              name3 -> Nom pour enregistrer l'image en sortie
%              beta  -> Paramètre du modèle de Potts
%              tempe0 -> Température initiale
%              decrease -> Coefficient de décroissance de la température
%              NIT -> Nombre d'itérations
%
% Output : res -> Résultat de segmentation
%-----

    %if(j>1&&(res(j-1,k) ~= val))
%       e = e + beta;
%    end
%    if(k>1&&(res(j,k-1) ~= val))
%       e = e + beta;
%    end
%    if(j<dim(1)&&(res(j+1,k) ~= val))
%       e = e + beta;
%    end
%    if(k<dim(2)&&(res(j,k+1) ~= val))
%       e = e + beta;
%    end

%tab_dif = zeros(1000,1000);
% tic;      bitor(tab_dif,tab_dif);toc
% tic;      tab_dif|tab_dif;toc

BN = 1;
BS = 2;
BE = 4;
BO = 8;

BNE = 5;
BNO = 9;
BSE = 6;
BSO = 10;

%-- Lecture de l'image à segmenter
dep = ima2mat(name1);
dim = size(dep);

%-- Affichage de l'image
```

```
figure;
photo(dep,256);
title('Image initiale');

%-- Lecture des parametres des classes
fid = fopen(name2, 'r'); % ouverture du fichier

% -- Initialisation température
tempe=tempe0;

% initialisation du nombre de classes
n_classe=0;
while (~feof(fid)) % tant qu'on est pas à la fin du fichier
    n_classe = n_classe + 1;
    moy(n_classe) = fscanf(fid,'%f',1);
    var(n_classe) = fscanf(fid,'%f',1);
end;
fclose(fid); % fermeture du fichier


%-- Pre-calcul des potentiels de classes
for i = 1 : n_classe
    for j = 0 : 255
        V1(i,j+1) = (j-moy(i))*(j-moy(i))/(2.*var(i))+log(var(i));
    end
end

%-- Initialisation aleatoire
for i = 1 : dim(1)
    for j = 1 : dim(2)
        res(i,j) = floor(n_classe * rand); % peut prendre les valeurs
0/1/.../n_classe
        if (res(i,j) == n_classe) % si valeur n_classe on retire
            j = j-1;
        end
    end
end

tab_border = zeros(dim(1),dim(2));
tab_border(1,:) = bitor(tab_border(1,:),BN);
tab_border(dim(1),:) = bitor(tab_border(dim(1),:),BS);
tab_border(:,1) = bitor(tab_border(:,1),BO);
tab_border(:,dim(2)) = bitor(tab_border(:,dim(2)),BE);%%because bitwise
operation in one number is VERY slow in MATLAB

%-- Boucle principale
for i = 1 : NIT % Boucle sur le nombre d'itérations
    disp(i);
    for j = 1 : dim(1)
        for k = 1 : dim(2)
            border = tab_border(j,k);

            valeur_courante = res(j,k); % Numéro de la classe associé au
pixel (j,k)
```

```
    valeur_nouvelle = n_classe; % Attribution d'une nouvelle classe
    (tirage uniforme, on répète le tirage si égal à n_classe)
    while (valeur_nouvelle == n_classe)
        valeur_nouvelle = floor(n_classe * rand);
    end

    % - Energie due a la vraisemblance
    energie_courante = V1(valeur_courante+1,dep(j,k)+1);
    energie_nouvelle = V1(valeur_nouvelle+1,dep(j,k)+1);

    % - Energie du modele a priori
    % A prior 4-connexe

    value_around = ones(1,8) * -1;
    switch(border)
        case 0%order by the number of occurnece because the switch
not replace by array of ptr of function
            value_around(1) = res(j-1,k-1);
            value_around(2) = res(j-1,k);
            value_around(3) = res(j-1,k+1);
            value_around(4) = res(j,k+1);
            value_around(5) = res(j+1,k+1);
            value_around(6) = res(j+1,k);
            value_around(7) = res(j+1,k-1);
            value_around(8) = res(j,k-1);
        case BN
            value_around(4) = res(j,k+1);
            value_around(5) = res(j+1,k+1);
            value_around(6) = res(j+1,k);
            value_around(7) = res(j+1,k-1);
            value_around(8) = res(j,k-1);
        case BE
            value_around(1) = res(j-1,k-1);
            value_around(2) = res(j-1,k);
            value_around(6) = res(j+1,k);
            value_around(7) = res(j+1,k-1);
            value_around(8) = res(j,k-1);
        case BS
            value_around(1) = res(j-1,k-1);
            value_around(2) = res(j-1,k);
            value_around(3) = res(j-1,k+1);
            value_around(4) = res(j,k+1);
            value_around(8) = res(j,k-1);
        case BO
            value_around(2) = res(j-1,k);
            value_around(3) = res(j-1,k+1);
            value_around(4) = res(j,k+1);
            value_around(5) = res(j+1,k+1);
            value_around(6) = res(j+1,k);
        case BSO
            value_around(2) = res(j-1,k);
            value_around(3) = res(j-1,k+1);
            value_around(4) = res(j,k+1);
        case BSE
            value_around(1) = res(j-1,k-1);
            value_around(2) = res(j-1,k);
            value_around(8) = res(j,k-1);
        case BNE
```

```

        value_around(6) = res(j+1,k);
        value_around(7) = res(j+1,k-1);
        value_around(8) = res(j,k-1);
    case BNO
        value_around(4) = res(j,k+1);
        value_around(5) = res(j+1,k+1);
        value_around(6) = res(j+1,k);
    end

    %matlab ne sait pas faire d'operation bit a bit avec une
    %vitesse convenable
    %
    value_around = -1 * ones(1,8);
    %
    if(bitand(border,BN)==0)
    %
        if(bitand(border,B0)==0)
    %
            value_around(1) = res(j-1,k-1);
    %
            value_around(8) = res(j,k-1);
    %
        end
    %
        value_around(2) = res(j-1,k);
    %
        if(bitand(border,BE)==0)
    %
            value_around(3) = res(j-1,k+1);
    %
            value_around(4) = res(j,k+1);
    %
        end
    %
    end
    %
    %
    if(bitand(border,BS)==0)
    %
        if(bitand(border,B0)==0)
    %
            value_around(5) = res(j+1,k+1);
    %
        end
    %
        value_around(6) = res(j+1,k);
    %
        if(bitand(border,BE)==0)
    %
            value_around(7) = res(j+1,k-1);
    %
        end
    %
    end
    %

    %%%%4-connexe
    %
    energie_courante = energie_courante + beta *
    sum(valeur_courante ~= value_around([2,4,6,8]));
    %
    energie_nouvelle = energie_nouvelle + beta *
    sum(valeur_nouvelle ~= value_around([2,4,6,8]));

    %%%%8-connexe
    tab_dif=(valeur_courante~=value_around');
    tmp_tri = tab_dif([2,4,6,8])|tab_dif([4,6,8,2]);
    energie_courante = energie_courante + beta*(sum(tab_dif +
    (tab_dif | circshift(tab_dif,1))) + sum(tmp_tri + (tmp_tri |
    tab_dif([3,5,7,1]))));

    tab_dif=(valeur_nouvelle~=value_around');
    tmp_tri = tab_dif([2,4,6,8])|tab_dif([4,6,8,2]);
    energie_nouvelle = energie_nouvelle + beta*(sum(tab_dif +
    (tab_dif | circshift(tab_dif,1))) + sum(tmp_tri + (tmp_tri |
    tab_dif([3,5,7,1]))));
    %

```

```
        if(energie_nouvelle<=energie_courante || (rand<exp(-(
(energie_nouvelle-energie_courante)/tempe))))
            res(j,k)=valeur_nouvelle;
        end
        % disp(exp(-(energie_nouvelle-energie_courante)/tempe));
    end
end
% -- Décroissance de la température
tempe = tempe * decrease;
end

% -- Affichage du resultat
figure
photo(res+1,n_classe);
title(['Segmentation : \beta=',num2str(beta),' / T=',num2str(tempe0),' /
decrease=',num2str(decrease),' / NIT=',num2str(NIT)]);

% -- Ecriture du resultat
mat2ima(res,name3);

%%tester avec le cerveau

end
```

param_cerveau

```
5.0 600.0
145.0 200.0
177.0 100.0
```