Guénon Marie et Favreau Jean-Dominique

VIM / Master SSTIM

Analyse d’image

Rapport de TD3

Table des matières

[Algorithme de Metropolis 2](#_Toc373826920)

[Calcul des énergies 2](#_Toc373826921)

[Mise à jour 3](#_Toc373826922)

[Tests 4](#_Toc373826923)

[4-connexité 4](#_Toc373826924)

[8-connexité 6](#_Toc373826925)

[Annexes 9](#_Toc373826926)

[recruit.m 9](#_Toc373826927)

# Algorithme de Metropolis

Le but de ce TP est de classifier 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

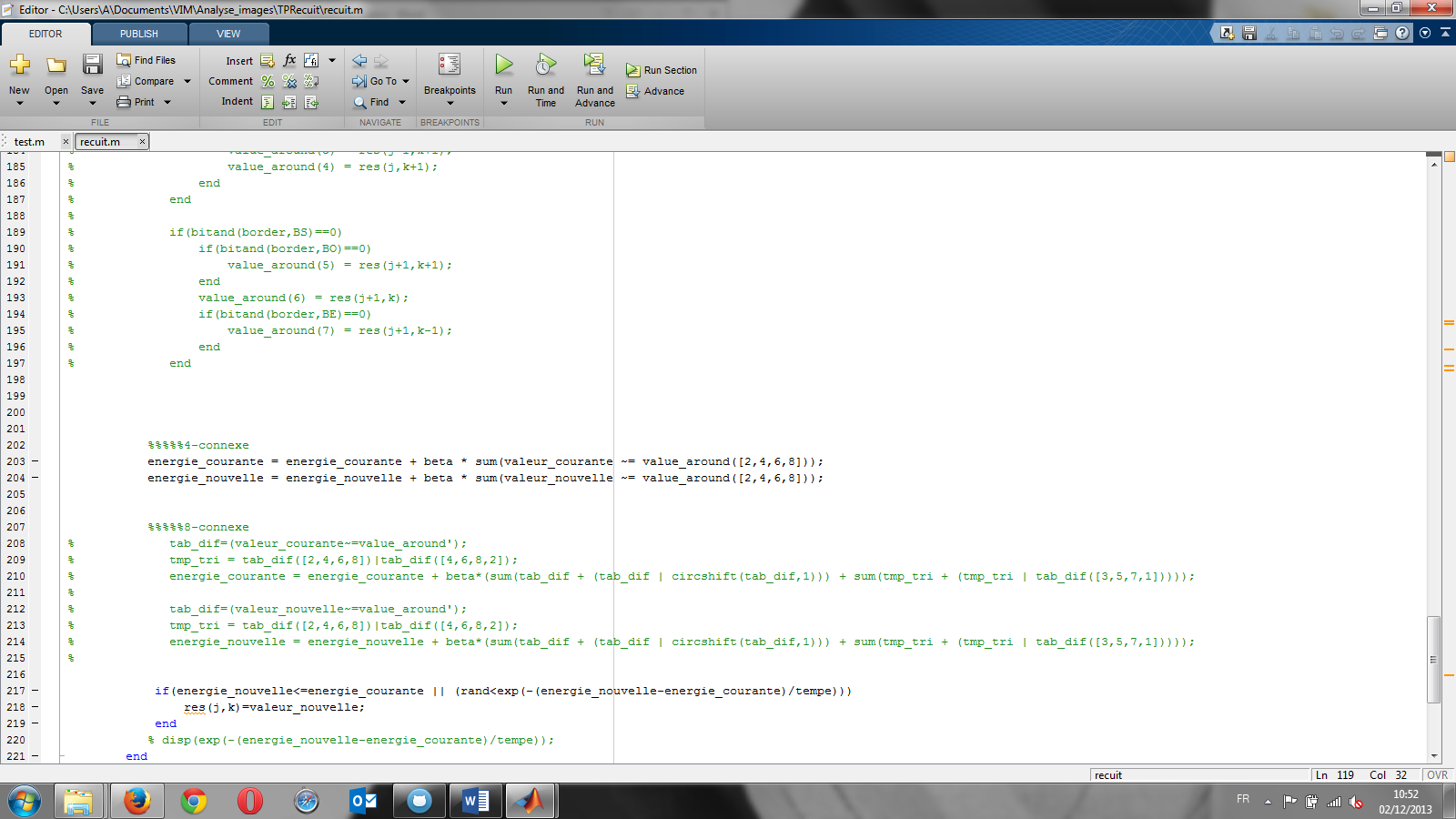
L’énergie correspondant à fidélité aux donné (vraisemblance) s’écrit :

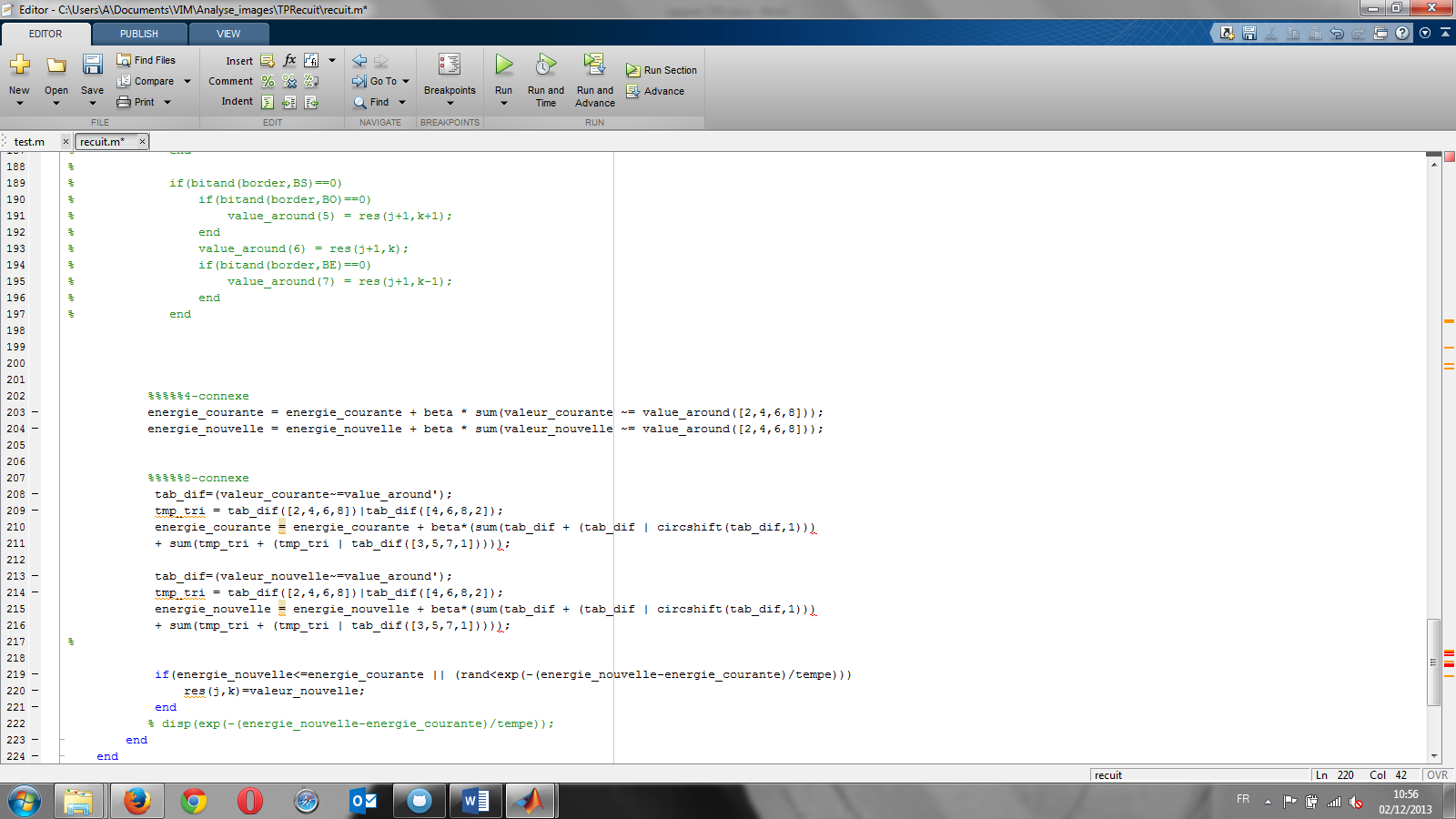
L’énergie du model à priori s’écrit :

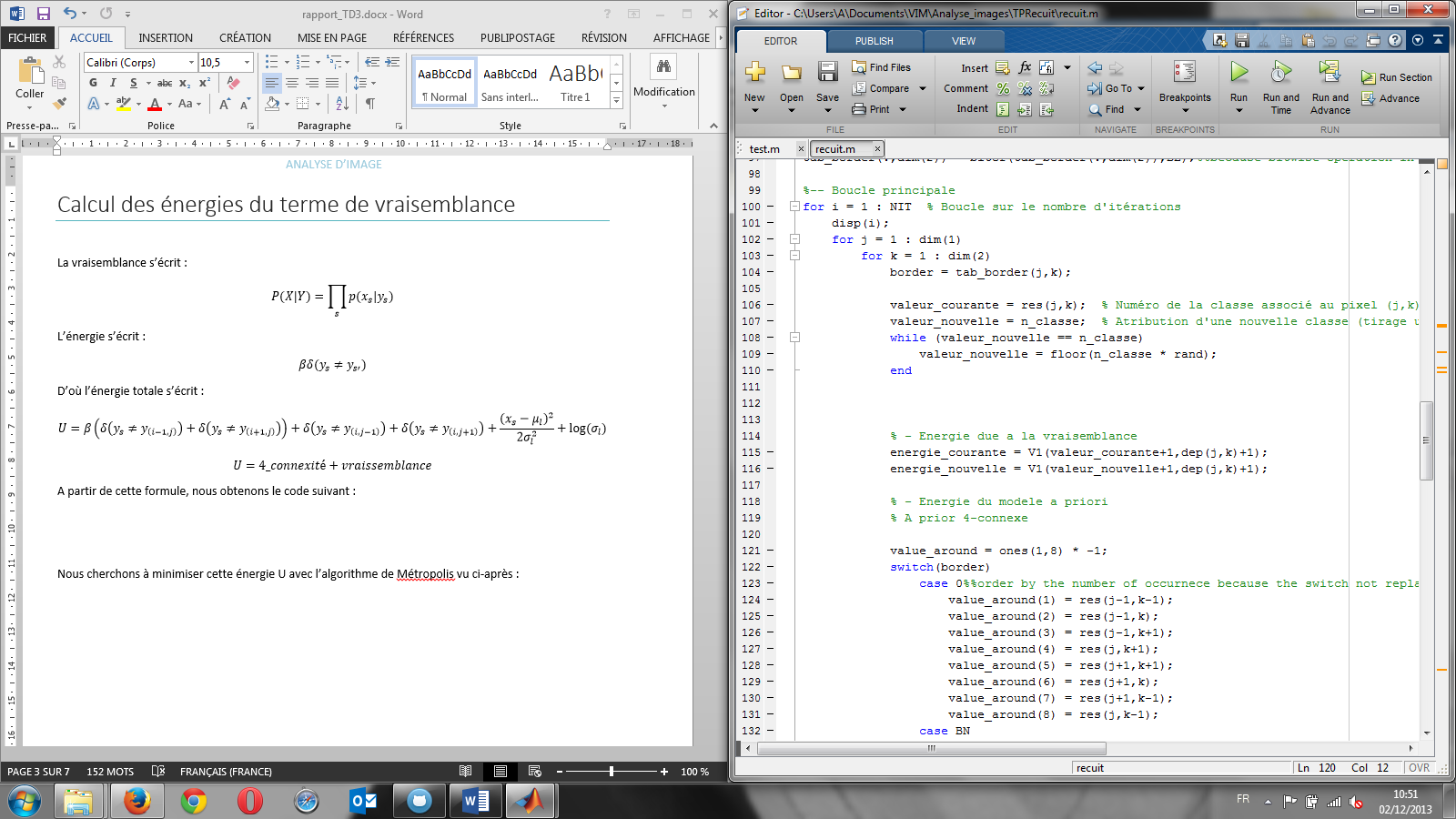
D’où l’énergie totale (en utilisant la 4-connexité) s’écrit :

On a une formule analogue pour la 8-connexité.

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







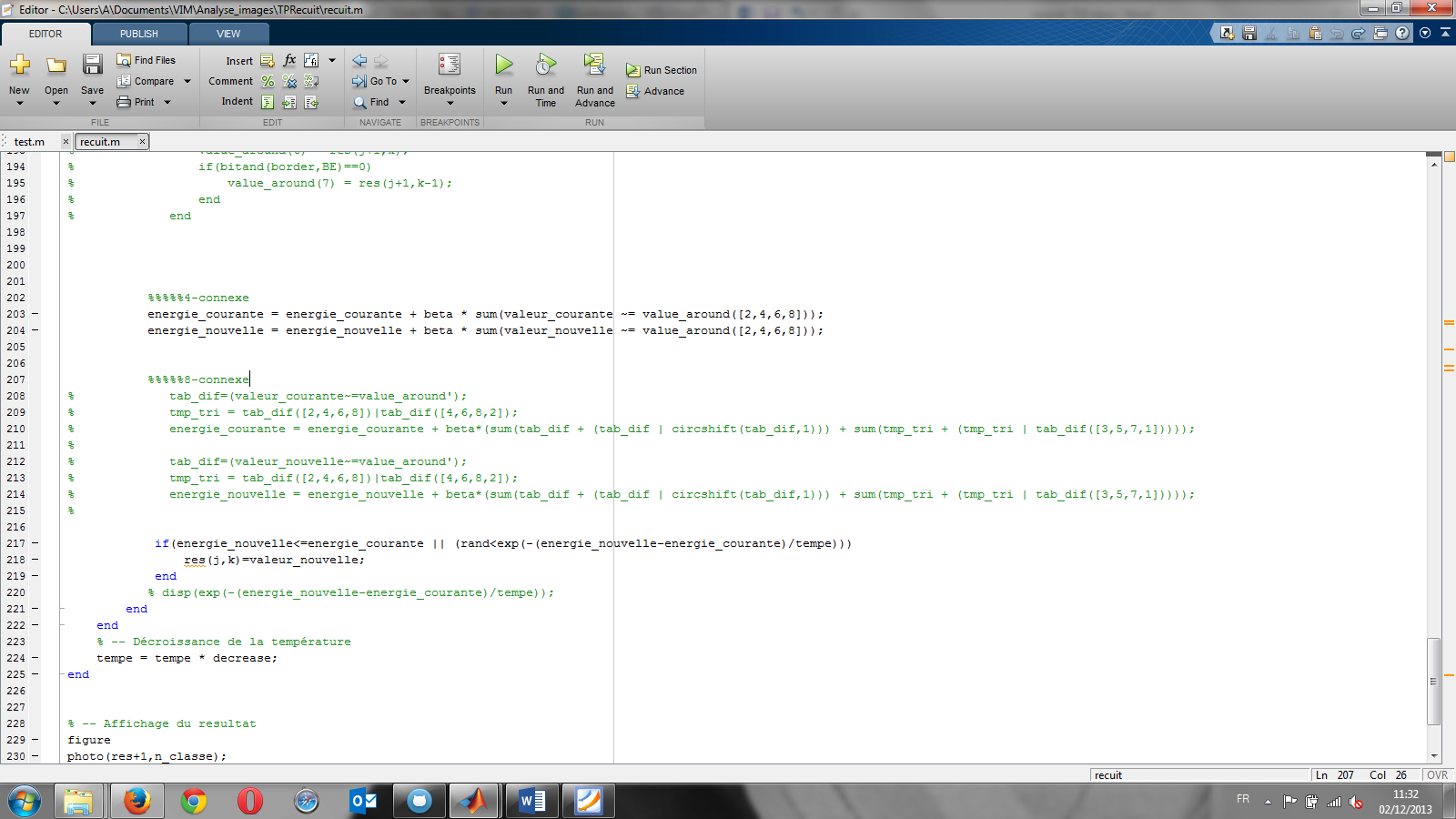
Les conditions aux limites sont gérées au préalable, avec le tableau value\_around :

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 :

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a | b | c |  |  |  |  |  |  |  |  |  |  |  |
| h | x | d |  | 🡺 |  | a | b | c | d | e | f | g | h |
| g | f | e |  |  |  |  |  |  |  |  |  |  |  |

## 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.

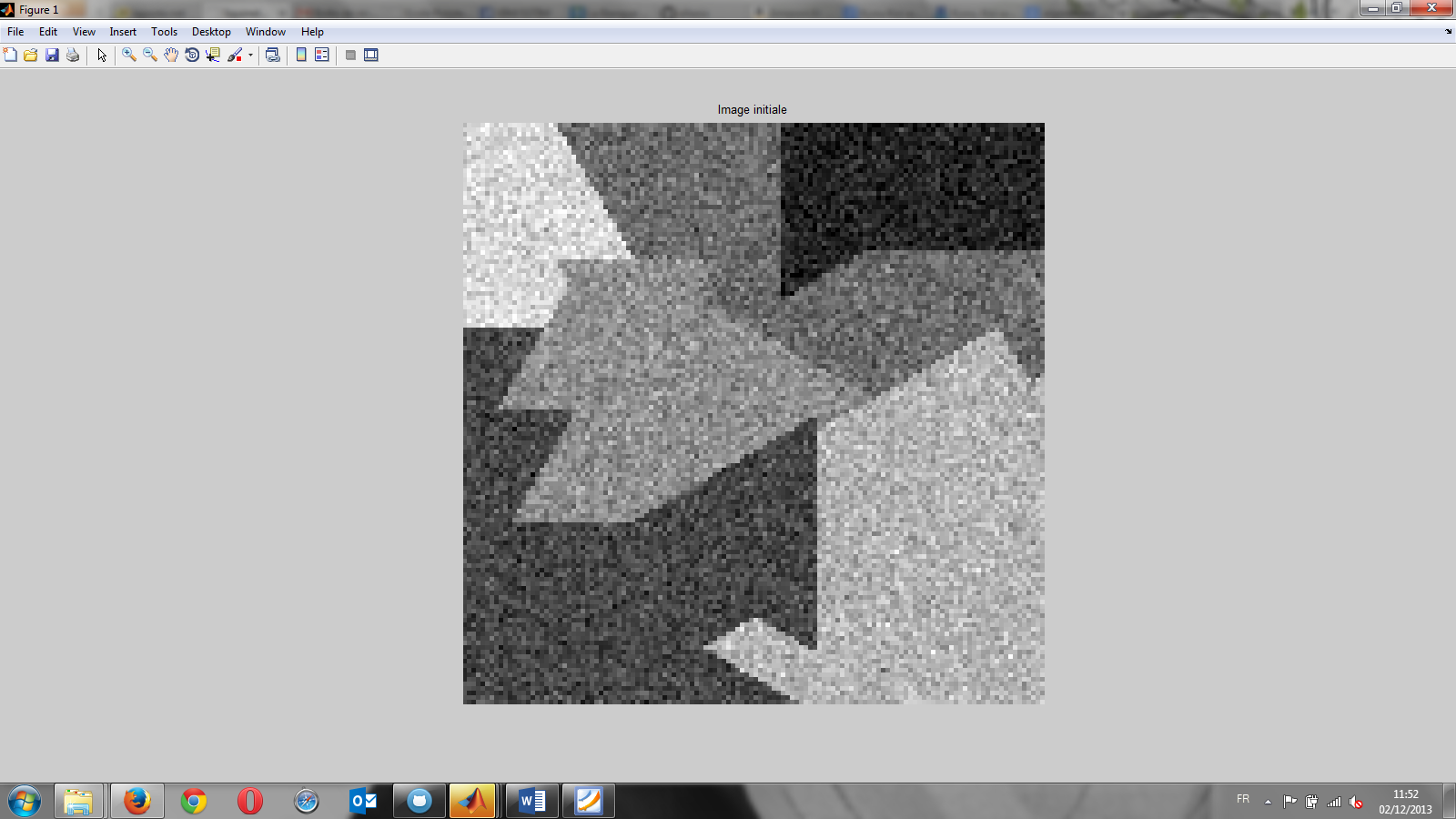


La première partie de la condition ci-dessus correspond à ce qu’on aurait dans le cas d’un algorithme variationnel (type descente de gradient…). La deuxième partie correspond à la partie stochastique de l’algorithme et est là pour permettre de visiter des "endroits" que l’on ne visiterait jamais avec 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* | *tempe* | *decrease* | *NIT* | 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 vraissemblance. |
| 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. |

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

Tempe = température initiale

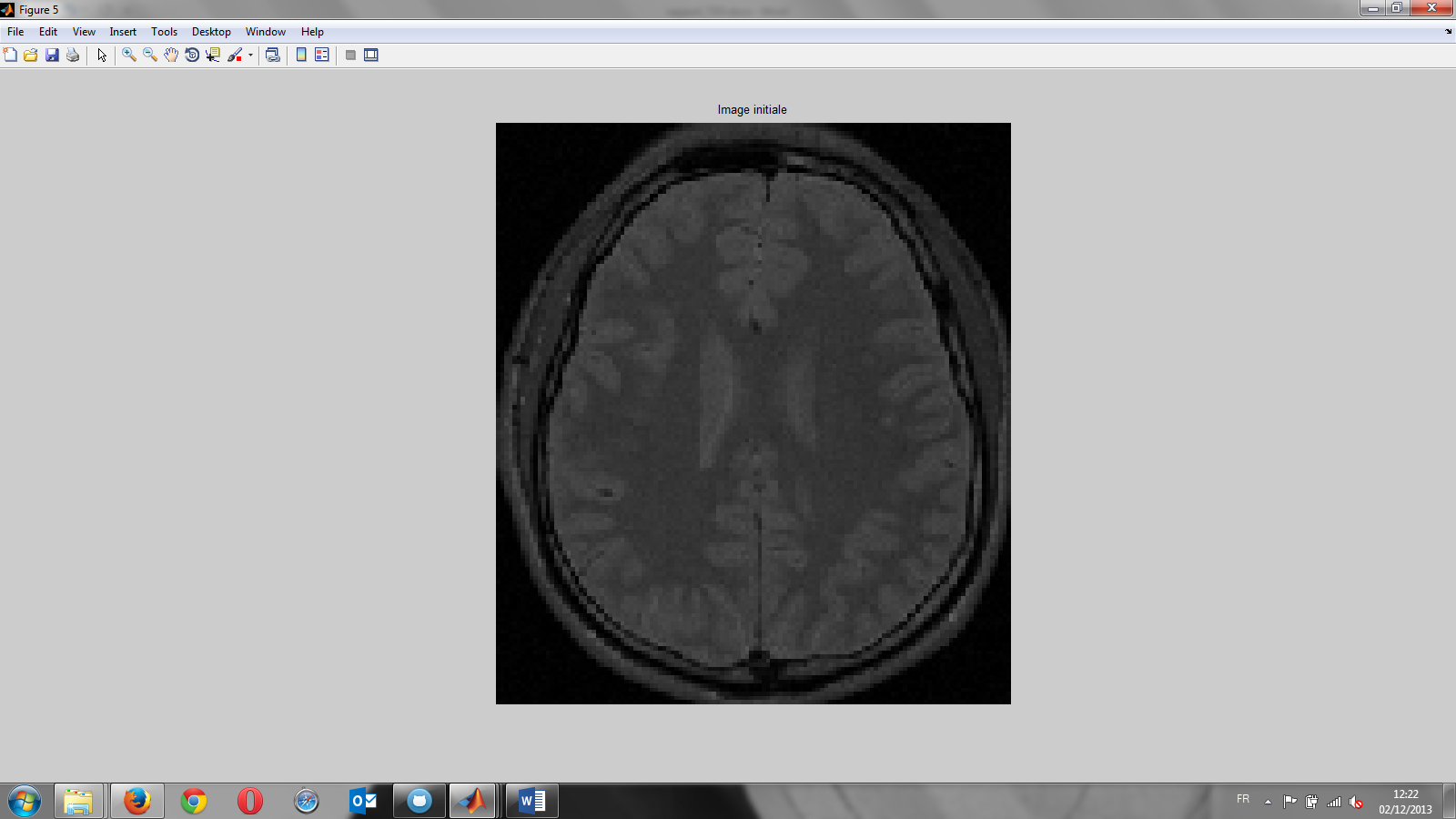
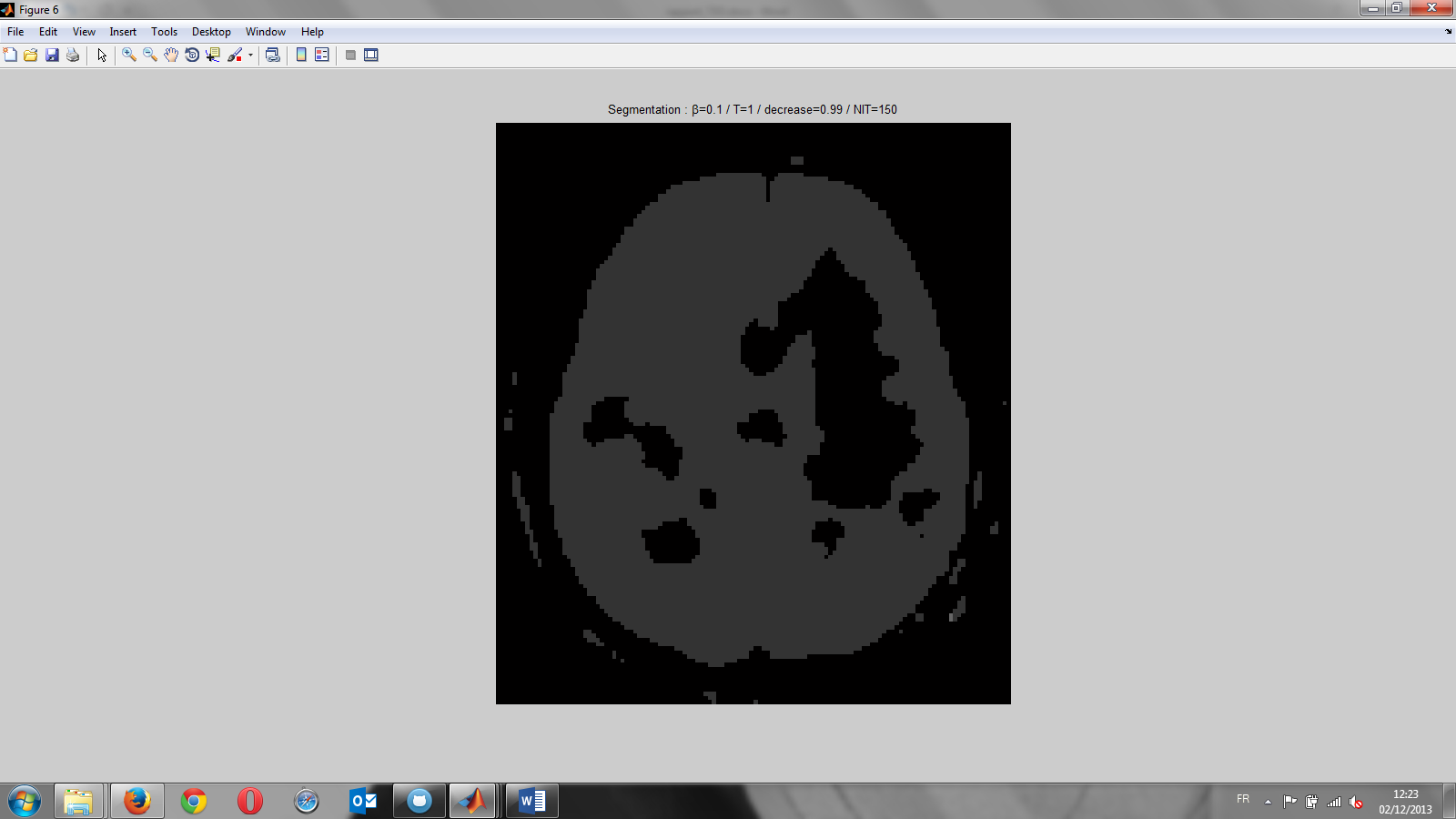
Decrease = Coefficient de décroissance de la température (erreur qu’on se permet)

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. |

Pour continuer nos test en 8-connexité, nous avons appliqué notre algorithme sur l’image du cerveau qui nous avait était fournie avec les valeurs qui avaient étaient les plus concluantes lors des tests précedents, c'est-à-dire β=0.5, tempe=1.0, decrease=0.99 et 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 reconnaitre les 6 classes de couleurs qui ont été définies.

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

* Redéfinir les classes de couleurs et en choisir de plus adaptées aux niveaux de gris présents dans cette image
* Changer l’histogramme de l’image à traiter pour correspondre au mieux aux classes de couleurs définies.

Nous avons choisies cette deuxième solution et avons obtenu les résultats suivants :

Histogrammes (original à gauche) :

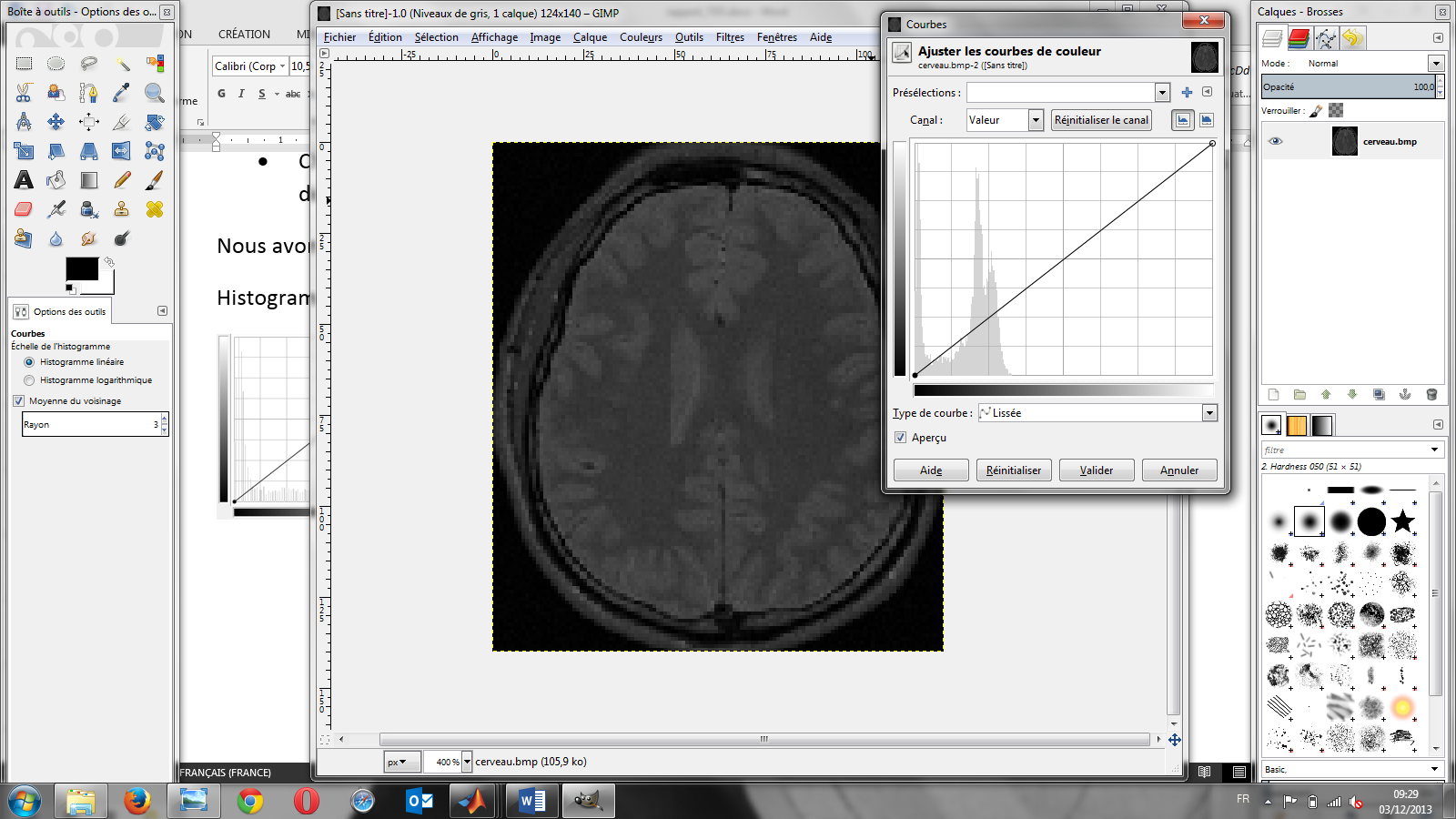
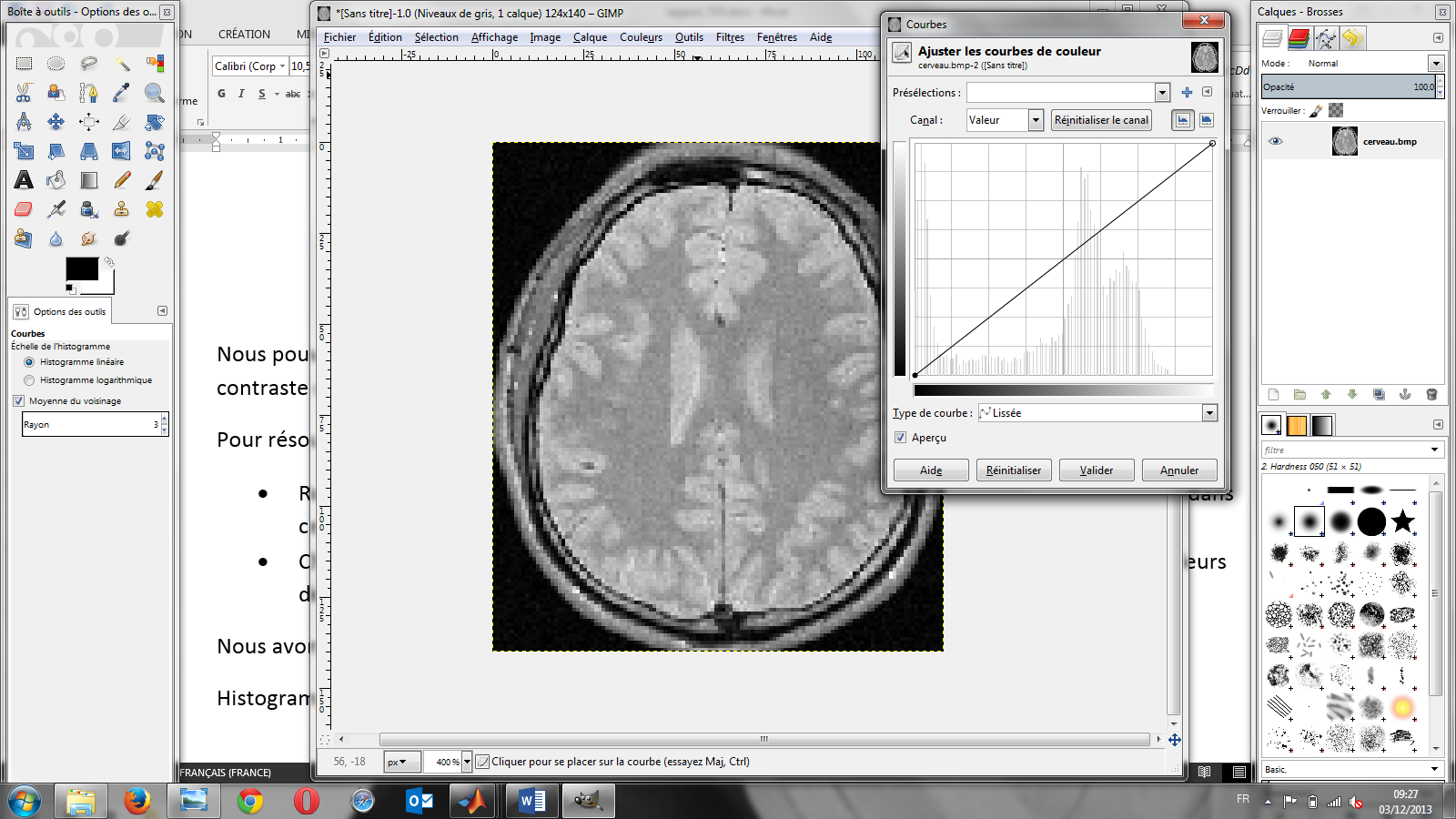
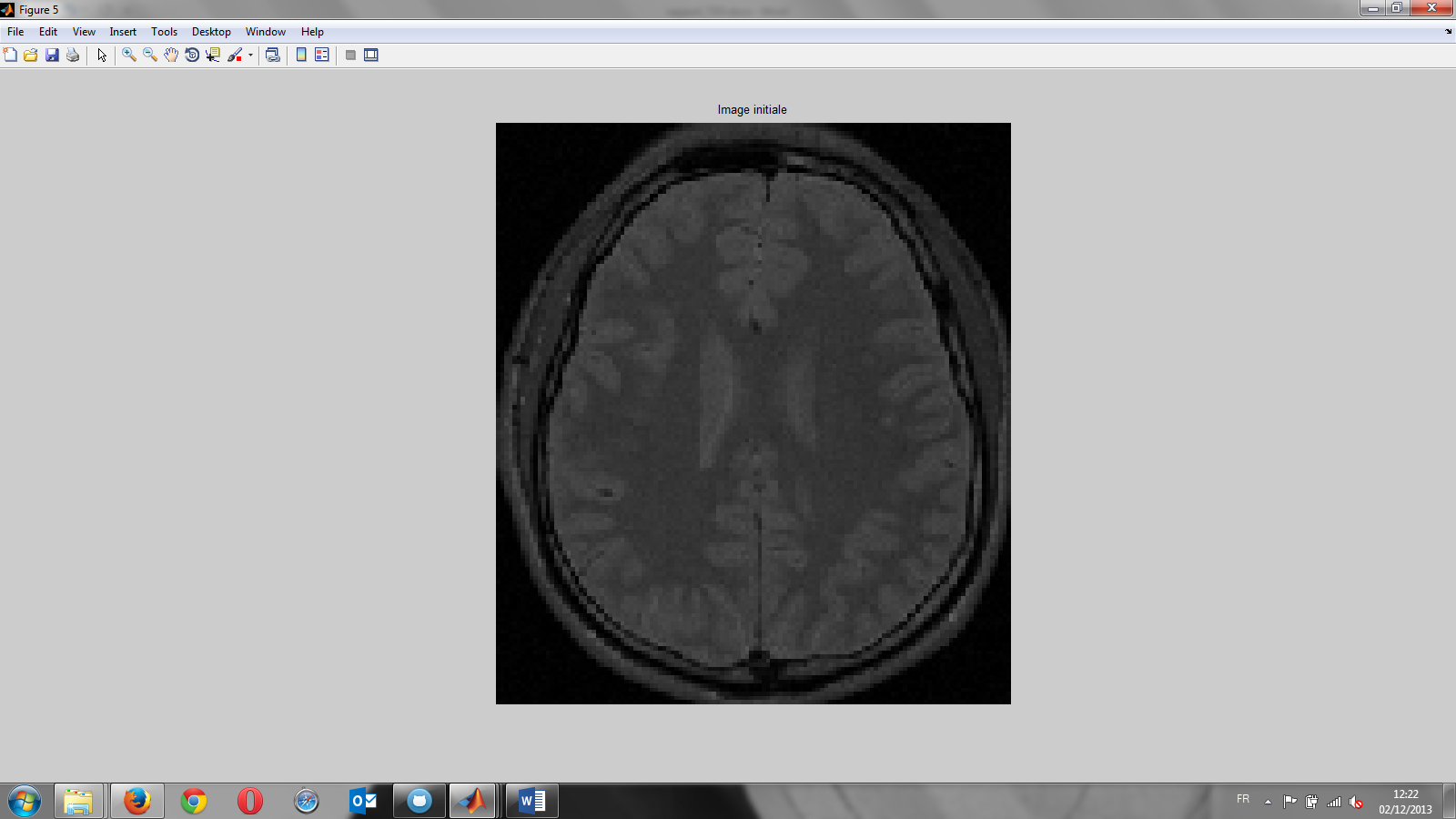
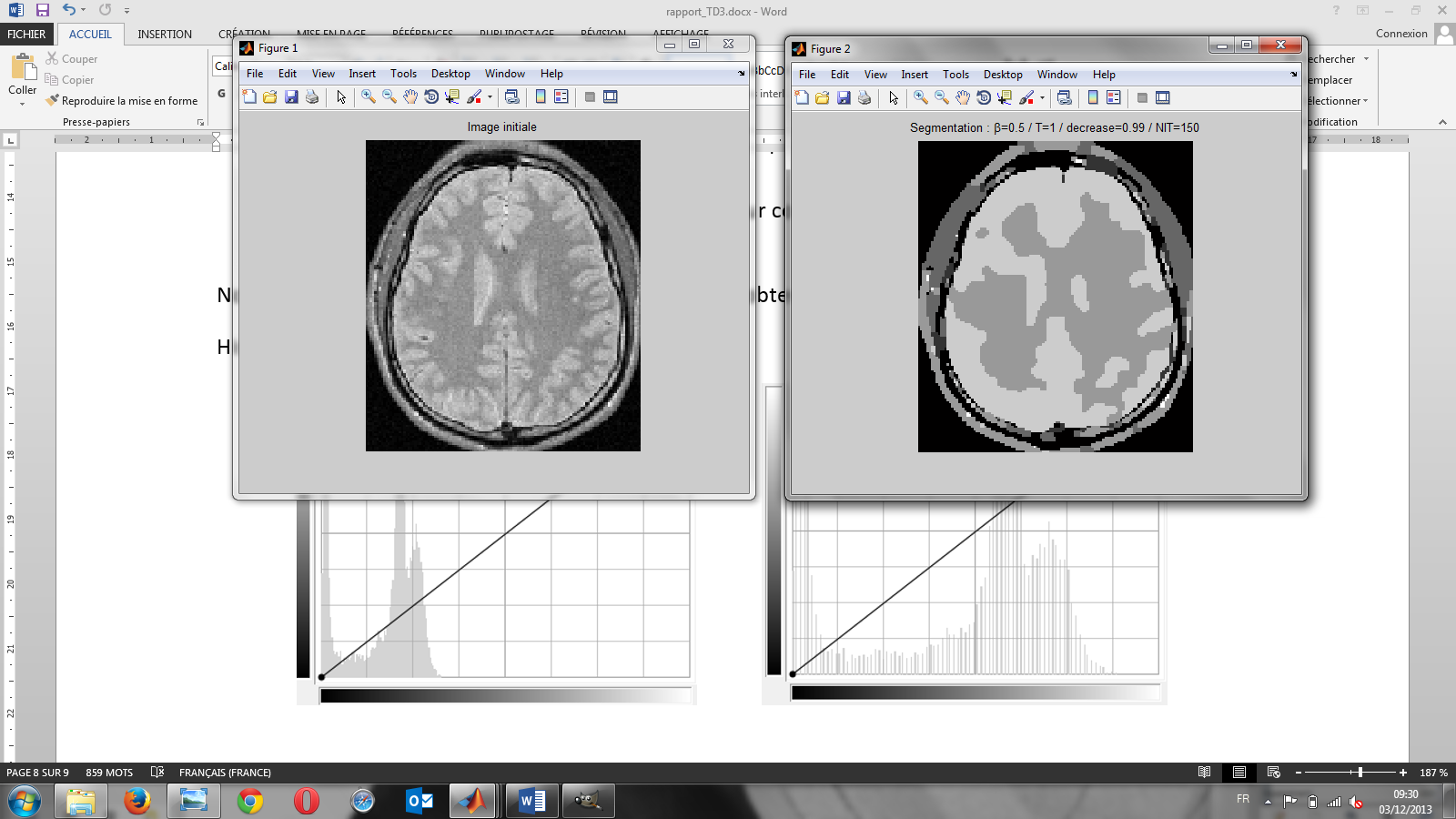
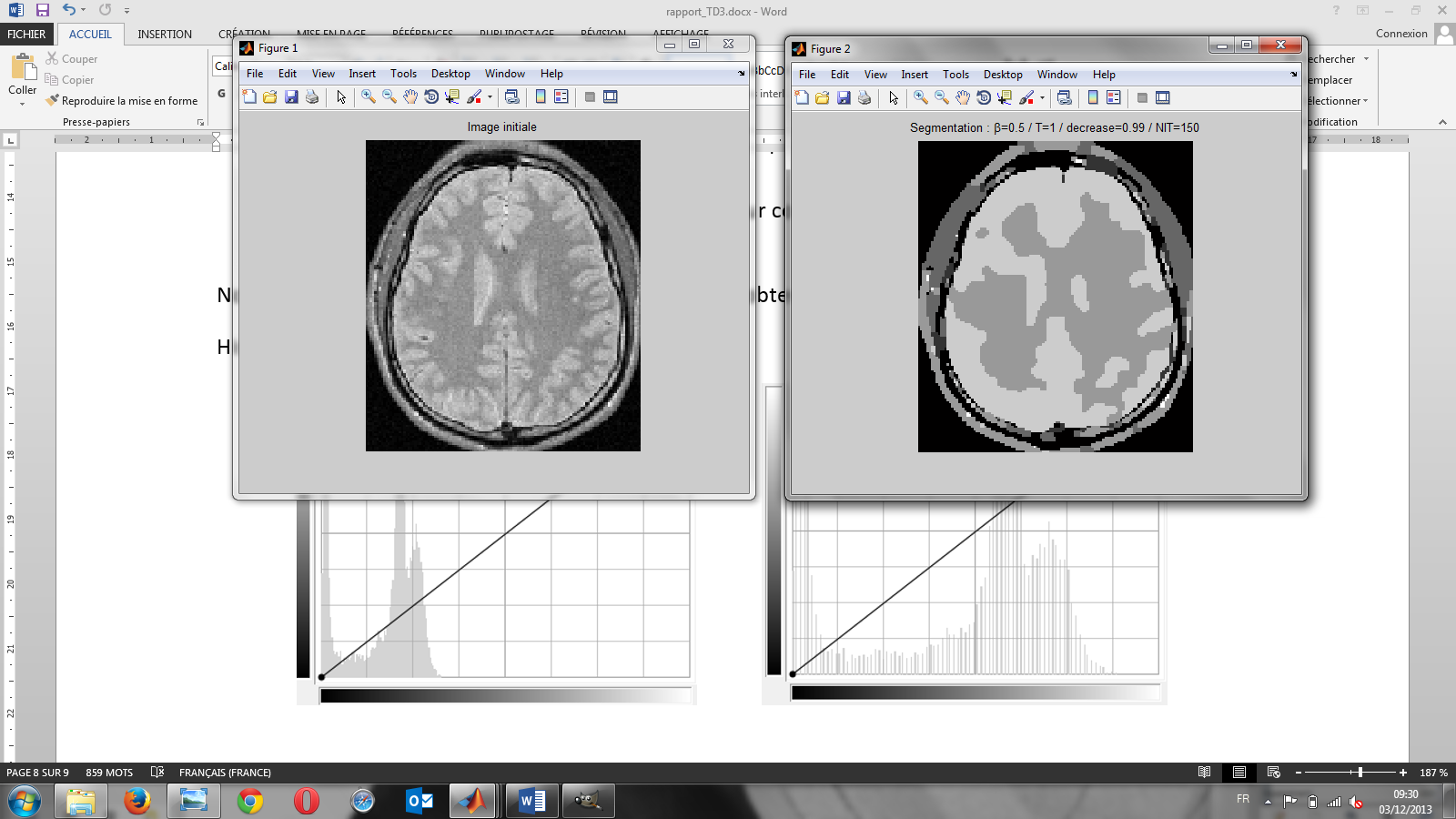
 

Image obtenue (originale à gauche) :

Résultat obtenue après segmentation :



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

# 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 a traiter

% s'apelle 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; % Atribution 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,BO)==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,BO)==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