Convolution / déconvolution.

1 Convolution discrète.

La convolution discrète est la base de nombreux algorithmes de modification d'images, et particulièrement lorsqu'il s'agit de mimer, dans l'espace discret, des modifications localement linéaires dans l'espace continu. Cette séance a pour but de vous familiariser avec la notion de convolution.

1.1 Principe

Les trois acteurs d'une convolution sont :

1- Une image discrète représentant l'information que l'on souhaite modifier (image source),
2- Une image discrète ne contenant que des zéros (image cible),
3- Un noyau de convolution (image de dimension réduite).

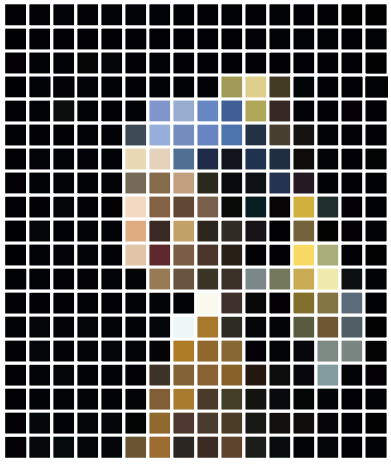


Image source : Is

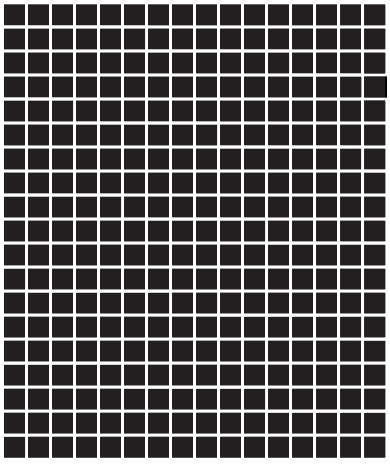
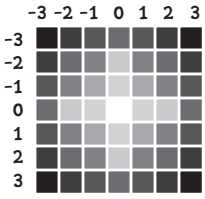


Image cible : Ic



Noyau : π

En supposant que les valeurs du noyau sont indexées en partant du centre du noyau tandis que les images sont généralement indexées en ligne, colonne comme une matrice.

1.2 Algorithme

L'algorithme de la convolution est très simple. En supposant que le noyau ait une dimension 7x7 comme représenté ci-dessus, la valeur du pixel (l,c) de l'image cible est obtenue par :

$$Ic(l,c) = \sum_{u=-3}^{3} \sum_{v=-3}^{3} Is(l+u,c+v).\pi(u,v)$$

Quand vous programmez, faites attention aux effets de bord. Il existe plusieurs méthodes pour calculer les valeurs sur les bords. La plus simple est de considérer que toutes les valeurs en dehors du cadre de l'image sont nulles.
Vous pouvez aussi supposer un « effet miroir ». En supposant que l va de 0 à L, si (l+u < 0) ou (l+u > N) on remplace (l+u) par (l-u) – et idem pour les colonnes.

1.3 Normalisation du noyau.

Dans la plupart des applications, il est important que la somme des éléments du noyau soient unitaire. Quand ce n'est pas le cas, on remplace le noyau π par sa version normalisée π/ß avec :

$$ß = \sum_{u=-3}^{3} \sum_{u=-3}^{3} \pi(u,v)$$

1.4 Quel noyau ?

On a vu en cours que la plupart des noyaux discrets sont obtenus par discrétisation de noyaux continus. Un noyau continu est défini par une fonction de forme f(x,y) et un paramètre d'étendue δ. On va s'intéresser ici aux noyaux moyenneurs, c'est-à-dire ceux provoquant un flou sur l'image. Dans ce cas les fonctions de forme f sont à support sur [-1,1]x[-1,1] – c'est-à-dire que f(x,y) = 0 si (x,y)∉[-1,1]x[-1,1]. Enfin la majorité de ces fonction sont séparables, c'est-à-dire qu'elles s'écrivent comme le produit de deux fonction : $f(x,y) = f_x(x).f_y(y)$.

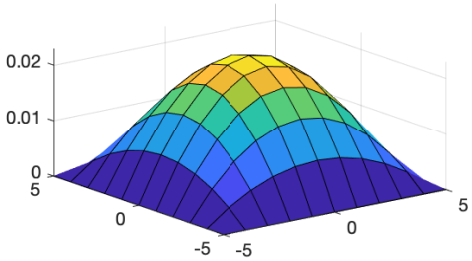
Voici un ensemble de fonction de forme pour (x,y)∈[-1,1]x[-1,1] :

- la fonction rectangle : f(x,y) = 1,
- la fonction triangle : : f(x,y) = (1-|x|).(1-|y|),
- la fonction d'Epanechnikov : $f(x,y) = (1-x^2).(1-y^2)$,
- la fonction circulaire : $\cos(\frac{\pi}{2}x).\cos(\frac{\pi}{2}y)$,
- ...,

vous pouvez en trouver beaucoup d'autres. Pour plus de simplicité, j'ai retiré les normalisation dans l'espace continu. Toutes les fonctions ci-dessus sont positives et à support borné. Cependant elles ne le sont pas toutes, comme par exemple la fonction sinus cardinal.

Pour passer d'une fonction f à support [-1,1]x[-1,1] à une fonction f_δ à support [-δ, δ]x[-δ, δ], il suffit d'écrire :

$$f_\delta(x,y) = f\left(\frac{x}{\delta},\frac{y}{\delta}\right)$$



Noyau d'Epanechnikov à support δ=4.9

Pensez à toujours normaliser votre noyaux échantillonné (la somme des coefficient = 1).

1.5 Création de noyaux.

Créez des noyaux échantillonnés de support quelconque (entre 1 et 5). Vous n'êtes pas obligés de choisir un nombre entier pour δ (vous pouvez choisir par exemple δ=1.2) par contre votre noyau aura forcément une dimension entière de (2.m+1)x(2.m+1) avec m = [δ], où [.] est l'arrondi (e.g. [1.2] = 1). Vous pouvez aussi créer des noyaux anisotropes (c'est-à-dire que l'étalement en x n'est pas le même qu'en y) par exemple : $f_\delta(x,y) = f\left(\frac{x}{3.2},\frac{y}{5.1}\right)$.

Visualisez ces noyaux comme image et comme fonction bidimensionnelles. Pour l'image n'oubliez pas de normaliser entre 0 et 255 et de passer en entier 8 bits.

Faites une fonction qui calcule directement le noyau (ici par exemple Epanechnikov). Dans un fichier que vous appelerez Noyau.m saisissez :

```
function MonNoyau = FaireUnNoyau(delta_x,delta_y)
m = ceil(max(delta_x,delta_y)) ;
MonNoyau = zeros(m,m) ;
for u=-m:m
    for v=-m:m
        x = u/delta_x ;
        y = v/delta_y ;
        MonNoyau(u+m+1,v+m+1) = max(0,(1-x*x))*max(0,(1-y*y)) ;
    end
end
MonNoyau = MonNoyau / sum(MonNoyau(:)) ;
```

Inspirez-vous de cette fonction pour réaliser d'autre noyaux de convolution. Vous pouvez par exemple créer une fonction dont le prototype serait :
`MonNoyau = FaireUnNoyau(delta_x,delta_y, type)`
La variable type permettrait de sélectionner le noyau (1 pour rectangle, 2 pour triangle, ...).

1.6 Essais de convolution.

Créez une fonction réalisant la convolution d'une image et un noyau (dans un fichier `Convolution.m`). On vous donne ici le début du prototype ... à vous de compléter.

```
function ImageCible = Convolution(ImageSource, Noyau)

[Nlin, Ncol] = size(ImageSource) ; % dimension image source
ImageCible = zeros(size(ImageSource)) ; % creation image cible

[nu, nv] = size(Noyau) ; % dimension du noyau


if( mod(nu,2) && mod(nv,2) )
    % les dimensions du noyau doivent etre impaires
    nu = floor(nu/2) ;
    nv = floor(nv/2) ;
    for lin=1:Nlin
        for col=1:Ncol
            for u=-nu:nu
                l = lin+u ;
                if (l>=1) && (l<=Nlin)
                    for v=-nv:nv
                        c = col+v ;
                        if (c>=1) && (c<=Ncol)
                            ImageCible(lin, col) = % votre code ici
                        end
                    end
                end
            end
        end
    end
end
```

Chargez des images de test disponibles ou bien utilisez vos propres images et testez le filtrage avec différentes fonctions et différents supports.
Si vos images sont en couleur, le filtrage est réalisé en filtrant chaque plan couleur.
Vous pouvez simuler une distorsion chromatique en n'appliquant pas le même filtre pour chaque couleur ($\delta_{bleu} \leq \delta_{vert} \leq \delta_{rouge}$).

1.7 Convolution séparable.

Comme vous l'avez remarqué, tous les noyaux qui vous sont proposés sont séparable. Il est alors possible de réduire considérablement la charge de calcul en transformant une convolution bidimensionnelle en deux convolutions monodimensionnelles.
Voici par exemple comment réaliser une convolution monodimensionnelle avec un noyau monodimensionnel :

```
function SignalCible = Convolution(SignalSource, Noyau)

[N] = length(SignalSource) ; % dimension signal source
SignalCible = zeros(size(SignalSource)) ; % creation signal cible

[n] = length(Noyau) ; % dimension du noyau

if( mod(n,2) )
    % la dimension du noyau doit etre impaires
    n = floor(n/2) ;
    for k=1:N
        for u=-n:n
            l = k+u ;
            if (l>=1) && (l<=N)
               SignalCible(k) = SignalCible(k) + % votre code ici
            end
        end
    end
end
```

Vous pouvez alors créer une fonction ayant pour argument le noyau permettant de filtrer les lignes qui aurait pour prototype :
`function ImageCible = Convolution1D(ImageSource, Noyau)`
Pour filtrer les colonnes, il suffit de transposer la matrice `ImageCible` obtenue en filtrant les lignes, rappeler la fonction avec le noyau permettant de filtrer les colonnes et retransposer le résultat :

```
ImageCible = Convolution1D(ImageSource, Noyau_ligne) ;
ImageCible = ImageCible' ;
ImageCible = Convolution1D(ImageSource, Noyau_colonne) ;
ImageCible = ImageCible' ;
```

Comparez ce que vous obtenez avec différents noyaux 1D et 2D.

2 Deconvolution

2.1 Qu'est-ce que la déconvolution ?

Dans de nombreux cas les images acquises en biologie – en microscopie optique par exemple ou en imagerie par fluorescence, souffrent d'une mauvaise résolution optique. On peut généralement modéliser cette dégradation par une convolution dans le domaine continu.
Sur le plan de l'image numérique, cette dégradation peut être vue comme une convolution avec un noyau échantillonné qui est l'échantillonnage du noyau de convolution dans l'espace continu.

2.2 Algorithmes de déconvolution itératifs.

Ces algorithmes ont été vus en cours. On les rappelle ici.
Si M est l'image mesurée (de taille nxm), I l'image originale qu'on souhaite reconstruire qui a été convoluée avec le noyau Π, on écrit $M = I \otimes \Pi$.
L'inversion de cette équation, et donc l'estimation de I, supposerait qu'on possède un noyau inverse (c'est-à-dire une noyau Π^{-1} tel que $\Pi \otimes \Pi^{-1} = \Delta$ (Δ étant un noyau tel que $\Delta(0,0)=1$, et $\Delta(u,v)=0$ si $u.v \neq 0$). Ce noyau n'existe généralement pas (dans le cas du noyau rectangulaire, le noyau inverse est le noyau sinus cardinal qui est à support infini par exemple).
Une écriture matricielle de cette estimation induirait l'inversion d'une matrice trop volumineuse pour être inversée avec sécurité, c'est pourquoi on se tourne vers des algorithmes itératifs minimisant une distance entre l'image reconstruite convoluée avec le noyau et l'image mesurée. Ces algorithmes consistent en une suite itérative I_k (k allant de 0 à une valeur K prédéfinie) convergeant vers I.
Comme nous utilisons des noyaux symétriques ($\Pi(u,v) = \Pi(-u,-v)$) ces algorithmes prennent une forme assez simple.

- Algorithme additif (ou ART ou méthode de Schultz ou méthode de Hotteling).

L'objet de cet algorithme est de faire converger la suite d'image I_k de façon à ce que $\Pi \otimes I_k$ se rapproche de M au sens de la norme Euclidienne : $\| \Pi \otimes I_{k+1} - M \| \leq \| \Pi \otimes I_k - M \|$.
Cet algorithme s'écrit :
$I_{k+1} = I_k + \lambda.(\Pi \otimes (M - \Pi \otimes I_k))$, où λ est une valeur assurant la convergence.
On prend généralement $\lambda = \| \Pi \|^{-2}$, où $\| . \|$ est la norme euclidienne (racine de la somme des carrés des valeurs).

- Algorithme multiplicatif (ou MLEM).

L'objet de cet algorithme est de faire converger la suite d'image I_k de façon à ce que $\Pi \otimes I_k$ se rapproche de M au sens de la différence des logarithmes. On essaye de rapprocher $\Pi \otimes I_k$ de M de façon à ce que la division terme à terme de $\Pi \otimes I_k$ par M tende vers 1 (une image ne contenant que des 1).
Cet algorithme s'écrit :
$I_{k+1} = I_k . (\Pi \otimes (M / (\Pi \otimes I_k)))$, où / est la division terme à terme.

2.3 Programmation des algorithmes.

Vous disposez d'une image acquise par un microscope électronique « ImageMicroscope.tif » et d'une coupe cytoplasmique d'ascidie « CoupeAscidie.tif ». Après avoir placé une mire sous le microscope, on a pu identifier que la dégradation subie par l'image issue du microscope électronique était équivalente à une convolution avec un noyau d'Epanechnikov avec $\delta = 5$, et, pour la coupe cytoplasmique, la dégradation est équivalente à une convolution avec un noyau circulaire d'étendu $\delta = 8$.
Essayez les deux algorithmes et comparer 1/ leur vitesse de convergence 2/ le résultat visuel.

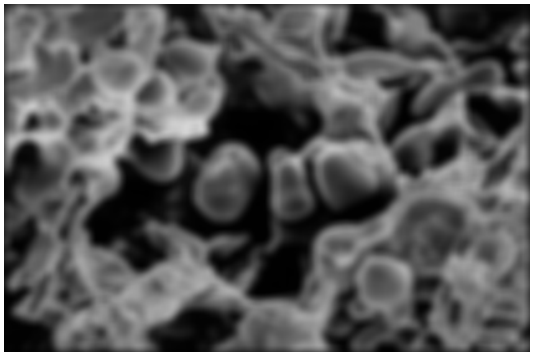
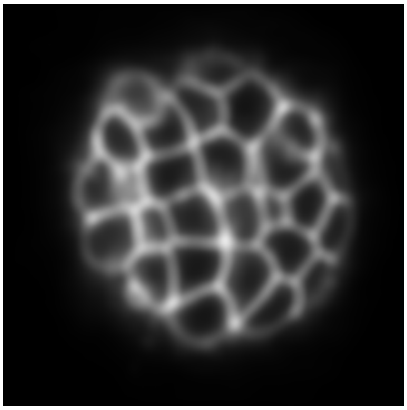


Image obtenue par un microscope électronique.


Coupe cytoplasmique d'une ascidie.

Pour simplifier le travail, utilisez les fonctions de convolution que vous avez déjà créés.
Vous pouvez visualiser la convergence de votre algorithme en affichant l'erreur résiduelle qui est la somme des valeurs absolues des écarts entre $\Pi \otimes I_k$ et M.