Yohann Tendero
yohann.tendero@telecom-paristech.fr
http://perso.telecom-paristech.fr/~ytendero/

Quelques rappels :

- Sténopé ; chaine de traitements numériques pour s'en approcher en pratique.
- Les photons, les photo-récepteurs, les variables aléatoire de Poisson, théorème fondamental de la photographie.
- Notion de convolution(s), réponse impulsionnelle, universalité de la convolution. Fourier : transformée, série et TFD.
- Théorème de Shannon-Whittaker. Fonctions "bande limitée" (observables).

Transformée de Fourier Discrète (TFD)

Soit u une suite finie définie sur $\{0, \ldots, N-1\}$. La TFD (ou DFT en anglais) de u est

$$\hat{u}(k) := \sum_{n=0}^{N-1} u(n) e^{-2i\pi \frac{k}{N} n} \text{ pour } k \in \{0, \ldots, N-1\}.$$

On a (formule d'inversion)

$$u(n) = \frac{1}{N} \sum_{k=0}^{N-1} \hat{u}(k) e^{2i\pi \frac{k}{N} n} \text{ pour } n \in \{0, \ldots, N-1\}. \quad (1)$$

Si on applique (1) pour $n = N$ on retrouve $u(0)$, etc.

Transformée de Fourier Discrète (TFD)

Soit u une suite finie définie sur $\{0, \ldots, N-1\}$. La TFD (ou DFT en anglais) de u est

$$\hat{u}(k) := \sum_{n=0}^{N-1} u(n) e^{-2i\pi \frac{k}{N} n} \text{ pour } k \in \{0, \ldots, N-1\}.$$

On a (formule d'inversion)

$$u(n) = \frac{1}{N} \sum_{k=0}^{N-1} \hat{u}(k) e^{2i\pi \frac{k}{N} n} \text{ pour } n \in \{0, \ldots, N-1\}. \quad (1)$$

Si on applique (1) pour $n = N$ on retrouve $u(0)$, etc. La suite u est implicitement vue comme un polynôme trigonométrique.

Transformée de Fourier Discrète (TFD)

Soit u une suite finie définie sur $\{0, \ldots, N-1\}$. La TFD (ou DFT en anglais) de u est

$$\hat{u}(k) := \sum_{n=0}^{N-1} u(n) e^{-2i\pi \frac{k}{N} n} \text{ pour } k \in \{0, \ldots, N-1\}.$$

On a (formule d'inversion)

$$u(n) = \frac{1}{N} \sum_{k=0}^{N-1} \hat{u}(k) e^{2i\pi \frac{k}{N} n} \text{ pour } n \in \{0, \ldots, N-1\}. \quad (1)$$

Si on applique (1) pour $n = N$ on retrouve $u(0)$, etc. La suite u est implicitement vue comme un polynôme trigonométrique. Elle est se prolonge naturellement

Transformée de Fourier Discrète (TFD)

Soit u une suite finie définie sur $\{0, \dots, N-1\}$. La TFD (ou DFT en anglais) de u est

$$\hat{u}(k) := \sum_{n=0}^{N-1} u(n) e^{-2i\pi \frac{k}{N} n} \text{ pour } k \in \{0, \dots, N-1\}.$$

On a (formule d'inversion)

$$u(n) = \frac{1}{N} \sum_{k=0}^{N-1} \hat{u}(k) e^{2i\pi \frac{k}{N} n} \text{ pour } n \in \{0, \dots, N-1\}. \quad (1)$$

Si on applique (1) pour $n = N$ on retrouve $u(0)$, etc. La suite u est implicitement vue comme un polynôme trigonométrique. Elle est se prolonge naturellement à tout $\mathbb{Z}$ par périodicité. On a en plus (Formule convolution/TFD)

$$\widehat{u \underset{per}{*} g} = \hat{u}\hat{g}.$$

Autrement dit $\forall k \in \{0, \dots, N-1\}$ $\widehat{u \underset{per}{*} g}(k) = \hat{u}(k)\hat{g}(k)$.

Les transformée de Fourier discrètes (TFD) se calculent rapidement (en Nlog(N)), par les algorithmes de "Fast Fourier Transform" (FFT).
Commandes matlab :

- fft (fft2 pour la 2D) pour la TFD (2D-DFT)
- ifft (ifft2 pour la 2D) pour la TFD inverse (2D-DFT inverse)
- Les commandes fftshift/ifftshift pour ré-ordonner (i pour la transformation inverse) les termes, si besoin.)

Il faut se souvenir de (Formule convolution/TFD)

$$\widehat{u \underset{per}{*} g} = \hat{u}\hat{g}.$$

Transformée de Fourier Discrète (TFD)

(formule d'inversion)

$$u(n) = \frac{1}{N} \sum_{k=0}^{N-1} \hat{u}(k) e^{2i\pi \frac{k}{N} n} \text{ pour } n \in \{0, \dots, N-1\}.$$

Comme tout polynôme trigonométrique, sa définition s'étend à tout $\mathbb{R}$: On calcule :

$$\hat{u}(k) := \sum_{n=0}^{N-1} u(n) e^{-2i\pi \frac{k}{N} n} \text{ pour } k \in \{0, \dots, N-1\},$$

et on a la formule d'interpolation

$$\mathcal{I}u(x) := \frac{1}{N} \sum_{k=0}^{N-1} \hat{u}(k) e^{2i\pi \frac{k}{N} x}, \text{pour } x \in \mathbb{R}.$$

On a évidemment $\mathcal{I}u(x) = u(n)$ pour tout $x \in \{0, \dots, N-1\}$.
On peut donc translater u arbitrairement, par exemple d'un demi-pixel.
Ce modèle continu pour u coïncide avec le théorème de Shannon : si nous observons un signal périodique et bande-limitée il serait exact.
Nous comparerons cette méthode avec un modèle affine par morceaux pour une rotation d'image.

Algorithme pour translater, formule de translation, cas 1D

Translater un signal permet p. ex. de recaler des images ou de synchroniser deux sons.

$$\begin{aligned}\mathcal{I}u(x-\frac{1}{2}) :&= \frac{1}{N}\sum_{k=0}^{N-1}\hat{u}(k)e^{2i\pi\frac{k}{N}(x-\frac{1}{2})}, \forall x\in\mathbb{R}.\\ &= \frac{1}{N}\sum_{k=0}^{N-1}\left(\hat{u}(k)e^{-2i\pi\frac{k}{N}\frac{1}{2}}\right)e^{2i\pi\frac{k}{N}x}\end{aligned}$$

Donc pour translater un signal u de τ pixel(s) :
1) On calcule la FTD de u
2) On multiplie $\hat{u}(0)$ par $e^{-2i\pi\frac{0}{N}\tau}$, $\hat{u}(1)$ par $e^{-2i\pi\frac{1}{N}\tau}$, ..., $\hat{u}(N-1)$ par $e^{-2i\pi\frac{N-1}{N}\tau}$.
3) On calcule la FTD inverse.
4) (Optionnel ne garder que la partie réelle)
Cette transformation est clairement inversible (car nous n'avons jamais multiplié une fréquence par 0).

Pour les mêmes raisons que le cas 1D on a une formule d'interpolation

$$\mathcal{I}u(x,y) := \frac{1}{N}\frac{1}{M}\sum_{k=0}^{M-1}\sum_{l=0}^{N-1}\hat{u}(k,l)e^{2i\pi\frac{l}{N}x}e^{2i\pi\frac{k}{M}y}.$$

qui permet d'étendre le signal à tout $\mathbb{R}^2$.

Algorithme pour filtrer une image ou un signal :

On se souvient que $DFT(u \underset{per}{*} g)(k) = DFT(u)(k)DFT(g)(k)$, pour tout k.

Entrée : Signal u, réponse impulsionnelle g.
1) Calculer la TFD de u et de g. Si u et g n'ont pas la même taille (souvent g est "trop court") prolonger g par des zéros avant de calculer sa TFD [1].
2) Multiplier point à point : $\hat{u}(0)\hat{g}(0), \ldots, \hat{u}(N-1)\hat{g}(N-1)$
(En matlab c'est .* ; si u est v ont même longueur u.*v calcule le produit point à point. * seul est utilisé pour les produits matriciels.)
3) Calculer la TFD-inverse du résultat.
4) (Optionnel : ne garder que la partie réelle.)

1. La formule de $\underset{per}{*}$ nous dit qu'il faut les ajouter après les valeurs connues de g, sinon on va translater le résultat. Ex g=[1 -1] => g=[1 -1 0 .... 0]

Algorithme pour calculer une dé-convolution (p. ex. enlever du flou) :

On se souvient toujours que $DFT(u \underset{per}{*} g)(k) = DFT(u)(k)DFT(g)(k)$, pour tout k.
Cette formule nous donne aussi la procédure pour dé-convoler, et les conditions pour que ce soit possible : Si $DFT(g)(k)$ ne s'annule pas alors le filtre est inversible.
Si v est connu ou a été estimé on applique : Entrée : Signal convolé u, réponse impulsionnelle g.
1) Calculer la TFD de u et de g. Si u et g n'ont pas la même taille (souvent g est "trop court") prolonger g par des zéros avant de calculer sa TFD [2].
2) Diviser point à point : $\hat{u}(0)/\hat{g}(0), \ldots, \hat{u}(N-1)/\hat{g}(N-1)$
(En matlab c'est ./.)
3) Calculer la TFD-inverse du résultat.
4) (Optionnel : ne garder que la partie réelle.)

2. La formule de $\underset{per}{*}$ nous dit qu'il faut les ajouter après les valeurs connues de g, sinon on va translater le résultat. Ex g=[1 -1] => g=[1 -1 0 .... 0]

En pratique, si pour certains k la quantité $|DFT(g)(k)|$ est très petite, le résultat ne va pas être bon : on va multiplier le bruit par $\frac{1}{|DFT(g)(k)|}$.
Pour les filtres non-inversibles on va préférer un filtre de type Wiener. Ce filtre est défini directement par sa réponse fréquentielle :

$$\frac{conj(DFT(g)(k))}{\lambda + abs(DFT(g)(k))^2}$$

λ est un paramètre qui contrôle la "quantité de déconvolution".
$\lambda = 0$ correspond à une division et ne fonctionne que pour les filtres inversibles.
Quand λ croit on déconvole de plus en plus partiellement.
(Voir plus bas pour des exemples sur des images.)

Algorithme pour zoomer "in" un signal : zero padding.

Idée : le signal est un polynôme.
La TFD a comme sortie autant de point que le nombre de coefficients (degré) du polynôme.

Algorithme pour zoomer "in" un signal : zero padding.

Idée : le signal est un polynôme.
La TFD a comme sortie autant de point que le nombre de coefficients (degré) du polynôme.
On a ajouté des fréquences plus hautes, avec comme valeur zero.
Equivalent à remplacer $X + X^2 - X^4 + X^5$ par
$X + X^2 - X^4 + X^5 + 0X^6 + 0X^7 + 0X^8 + \cdots$.
Cela ne change donc pas le polynôme, juste le nombre de points d'évaluation.
(Voir plus loin un exemple sur une image.)

Transformée de Fourier Discrète (2D-TFD)

Soit u une image finie définie sur $\{0,\dots,M-1\}\times\{0,\dots,N-1\}$. LA TFD de u est

$$\hat{u}(k,l) := \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} u(n,m)e^{-2i\pi\frac{k}{N}n}e^{-2i\pi\frac{l}{M}m} \ \forall (k,l)\in\{0,\dots,M-1\}\times\{0,\dots,N-1\}.$$

Cette transformation est séparable. Elle consiste à calculer la TFD des lignes, puis la TFD des colonnes du résultat ou le contraire : d'abord les colonnes puis les lignes (l'ordre de sommation n'a pas d'importance).
On a la formule d'inversion : $\forall (m,n)\in\{0,\dots,M-1\}\times\{0,\dots,N-1\}$:

$$u(m,n) := \frac{1}{N}\frac{1}{M}\sum_{k=0}^{M-1}\sum_{l=0}^{N-1}\hat{u}(k,l)e^{2i\pi\frac{l}{N}n}e^{2i\pi\frac{k}{M}m}.$$

Algorithme pour translater, cas 2D

$$\mathcal{I}u(x+\tau_x, y+\tau_y) := \frac{1}{N}\frac{1}{M}\sum_{k=0}^{M-1}\sum_{l=0}^{M-1}\hat{u}(k,l)e^{2i\pi\frac{l}{N}(x+\tau_x)}e^{2i\pi\frac{k}{M}(y+\tau_y)}.$$

Donc pour translater un signal u de τ_x, τ_y pixel(s) :
Entrée u définie sur $\{0,\dots,M-1\}\times\{0,\dots,N-1\}$, tau_x, τ_y :
1) On calcule la 2D-FTD de u
2) On multiplie chaque $\hat{u}(k,l)$ par $e^{2i\pi\frac{k}{M}\tau_y}e^{2i\pi\frac{l}{N}\tau_x}$,
3) On calcule la FTD inverse.
4) (optionnel ne garder que la partie réelle)
Cette transformation est clairement inversible (car nous n'avons jamais multiplié par 0).

Algorithme pour calculer une convolution périodique :

Entrée : Signal u, réponse impulsionnelle g.
1) Calculer la TFD de u et de g.
2) Multiplier point à point : fft2(u).*fft2(g)
3) Calculer la TFD-inverse du résultat.
4) (Optionnel : ne garder que la partie réelle.)

Déconvolution Gaussienne (sans bruit) : image observée

Déconvolution Gaussienne (sans bruit) : image déconvolée, noyau connu

Déconvolution Gaussienne (avec bruit, $\mathcal{N}(0, 0.01)$) : image observée. Image a ses valeurs dans $[0, 1]$

Déconvolution Gaussienne avec bruit, $\mathcal{N}(0, 0.01)$) : déconvolée par division

Déconvolution Gaussienne avec bruit, $\mathcal{N}(0, 0.01)$) : déconvolée par Wiener

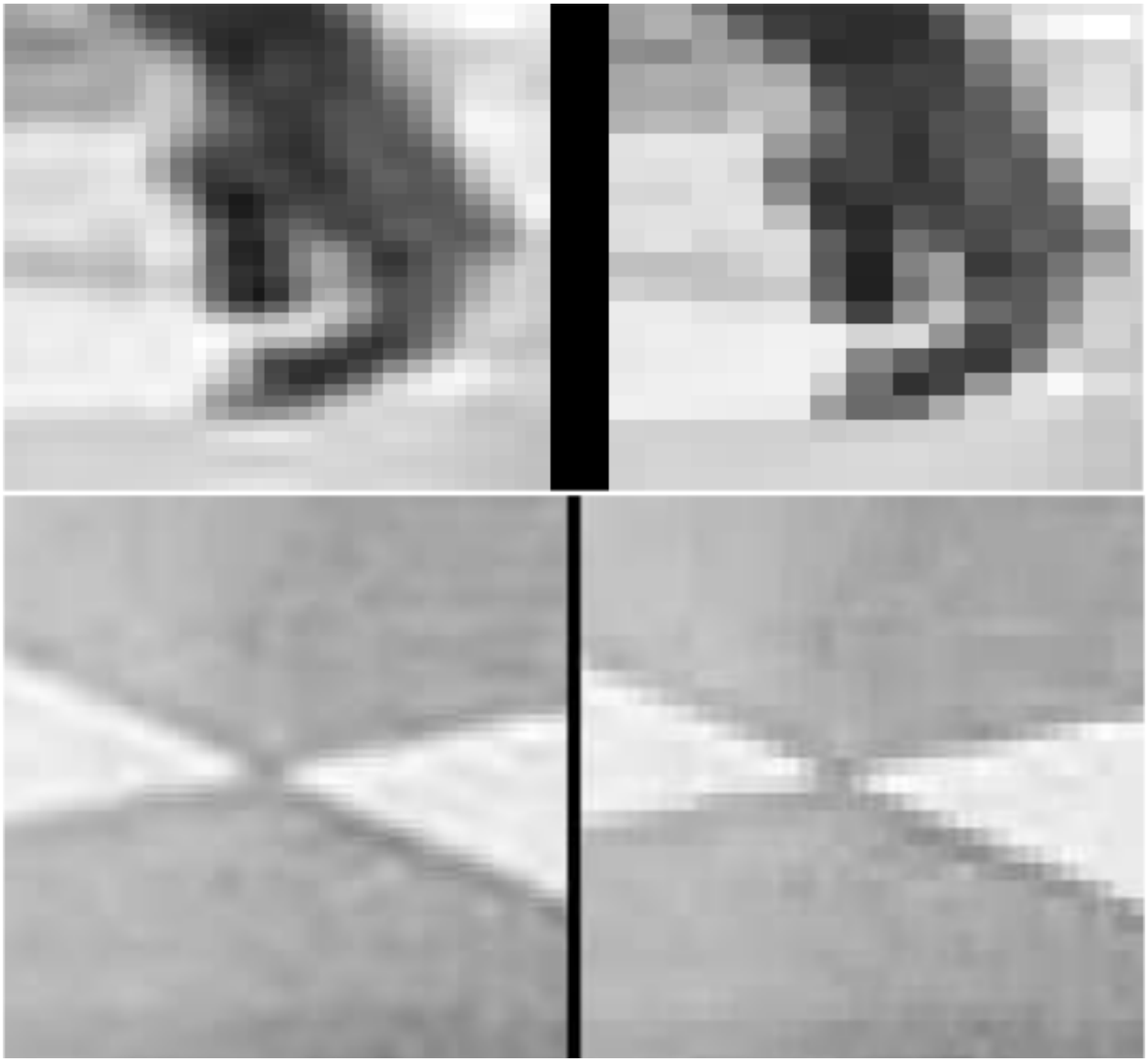
Zero padding

1) Créer une grande image contenant des zéros appelée p. ex I_hat_zoom.
2) Calculer le 2D-TFD de l'image, enregistrer dans p. ex I_hat.
3) Placer les coefficients de I_hat dans I_hat_zoom à la positions des coefficients basse fréquence (BF). Si vous faites un fftshift I_hat sera au centre de I_hat_zoom (calculez bien la position du centre), sinon vous pouvez procéder p. ex. cadrant par cadrant (il y en a 4) avec les BF positives dans les deux directions, etc.
3) TFD inverse, partie réelle, afficher.

Original

Zoom X4 : padding

Zoom par réplication des pixels



Rotation

Nous voudrions ré-échantillonner u aux points

$$\begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} m \\ n \end{pmatrix},$$

pour $(m, n) \in \{0, \ldots, M-1\} \times \{0, \ldots, N-1\}$. La formule

$$\mathcal{I}u(x, y) := \frac{1}{N}\frac{1}{M} \sum_{k=0}^{M-1} \sum_{l=0}^{M-1} \hat{u}(k, l) e^{2i\pi \frac{x}{N} n} e^{2i\pi \frac{y}{M} m}.$$

conduit à effectuer MN sommes pour évaluer un point de l'image tournée $=>$ $(MN)^2$ sommes au total.
Cela rend la méthode directe inopérante en pratique.

Algorithme pour calculer une rotation d'image

Cette méthode est due L. Yaroslavsky. Elle repose sur la remarque suivante

$$\begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} = \begin{pmatrix} 1 & -\tan(\frac{\theta}{2}) \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ \sin(\theta) & 1 \end{pmatrix} \begin{pmatrix} 1 & -\tan(\frac{\theta}{2}) \\ 0 & 1 \end{pmatrix},$$

qui est valide pour $\theta \neq \pm\pi$ modulo 2π. Si $\theta = \pm\pi$ modulo 2π, il suffit de retourner l'image (ou de la laisser telle quelle).

Regardons ce qu'il se passe pour la première matrice pour $(m, n) \in \{0, \ldots, M-1\} \times \{0, \ldots, N-1\}$:

$$\begin{pmatrix} 1 & -\tan(\frac{\theta}{2}) \\ 0 & 1 \end{pmatrix} \begin{pmatrix} m \\ n \end{pmatrix} = \begin{pmatrix} m - n\tan(\frac{\theta}{2}) \\ n \end{pmatrix}$$

$=>$ C'est une translation suivant le premier indice. Les colonnes ne bougent pas. La première ligne est translatée de 0, la seconde de $-\tan(\frac{\theta}{2})$, la troisième de $-2\tan(\frac{\theta}{2})$, la suivante de $-3\tan(\frac{\theta}{2})$, etc.
$=>$ La seconde matrice correspond à translater les lignes de manière similaire.
$=>$ La troisième translate les colonnes à nouveau.

Algorithme pour calculer une rotation d'image

Placer l'image au centre d'une plus grande image en complétant par des zéros ou une autre constante
(Sinon par périodisation un bord qui sort ré-entre de l'autre coté.)
Utiliser l'algorithme pour translater un signal (les lignes/colonnes) avec les valeurs de translations données par

$$\begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} = \begin{pmatrix} 1 & -\tan(\frac{\theta}{2}) \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ \sin(\theta) & 1 \end{pmatrix} \begin{pmatrix} 1 & -\tan(\frac{\theta}{2}) \\ 0 & 1 \end{pmatrix}.$$

Rotation avec la méthode de Yaroslavsky (16 fois $\frac{\pi}{4}$)

Rotation avec interpolation bilineaire (16 fois $\frac{\pi}{4}$)

Image originale

Rotation avec la méthode de Yaroslavsky

Zoom out : original

Zoom out : direct decimation : Aliasing

Zoom out (2) : original

Zoom out :(2) direct decimation : Aliasing