OUTILS MATHEMATIQUES POUR L'INFORMATIQUE

PROJET – TRANSFORMEE DE FOURIER

ALEXIS GUYOT – ALEXANDRE LABROSSE
LICENCE 3 INFORMATIQUE - UB
2018 -2019

Table des matières

I. Introduction

Dans le cadre de la matière Outils Mathématiques pour l'Informatique, nous avons mené lors de ce premier semestre de notre troisième année de licence un projet gravitant autour de la Transformée de Fourier. Il s'agit d'une opération qui permet d'obtenir le spectre fréquentiel d'un signal de manière générale, mais ici dans notre cas surtout le spectre d'une image (d'une matrice de pixels). En image, ces spectres permettent d'apprécier l'agitation (s'il y a beaucoup de basses fréquences / de zones homogènes, ou au contraire beaucoup de hautes fréquences / de contours, …) mais aussi d'appliquer beaucoup plus facilement et rapidement des filtres.

L'objectif de ce projet était donc de comprendre le fonctionnement de la transformée de Fourier puis de l'implémenter dans le langage de notre choix pour pouvoir traiter des images 1D et 2D, dans les sens direct et indirect. Deux implémentations devaient être proposées : l'implémentation traditionnelle, de complexité $O(N^2)$ pour les images 1D et $O(N^4)$ pour les images 2D (donc très vite inefficace pour les images de grandes dimensions), et l'implémentation rapide, de complexité beaucoup plus intéressante en $O(N\log_2 N)$.

Nous avons choisi d'utiliser le logiciel Matlab et son langage intégré pour mener à bien ce projet. Ce choix s'explique simplement par le fait que Matlab est un logiciel très optimisé pour manipuler des images et des matrices et qu'il possède une implémentation fiable de la transformée de Fourier avec laquelle on pouvait facilement comparer nos résultats. Nous avons également fait le choix de travailler uniquement sur des matrices de taille 2^n pour être cohérent avec ce qui a été fait pendant les cours.

II. Transformée de Fourier directe 2D

Le premier algorithme à réaliser est celui de la transformée de Fourier directe discrète pour une image. On utilisera la formule suivante pour calculer cette transformée :

$$F(g(x,y)) = \hat{g}(u,v) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} g(x,y)exp(-2i\pi(\frac{ux}{M} + \frac{vy}{N}))$$

1. Algorithme utilisé

```
function [tf] = TFdirecte2D(img)
    %On cherche le nombre de ligne et de colonne de l'image passée en
    %paramètre.
    [nbLigne,nbColonne]=size(img);
    %On crée la matrice qui sera renvoyée à la fin du programme (de la
    %même taille que celle passée en paramètre), initialisée à 0.
    tf=zeros(nbLigne,nbColonne);
    %Pour chaque pixel de l'image à renvoyée
    for u=1:nbLigne
        for v=1:nbColonne
            %Pour chaque pixel  de l'image passée en paramètre
            for x=1:nbLigne
                for y=1:nbColonne
                    %On applique la formule de la transformée de Fourier
                    %sur le pixel de la nouvelle image
                    tf(u,v) = tf(u,v) + img(x,y)*exp((-2*1i*pi)*(((u-1)*...
                    (x-1)/nbLigne) + ((v-1)*(y-1)/nbColonne)));
                end
            end
        end
    end
end
```

2. Explications

Code réalisé	Explications
`function [tf] = TFdirecte2D(img)`	La fonction créée a pour nom TFdirecte2D. Elle prend en paramètre une matrice à deux dimensions représentant l'image sur laquelle nous voulons appliquer la transformée de Fourier. Elle renvoie la transformée de Fourier de l'image passée en paramètre.
`[nbLigne,nbColonne]=size(img);`	Nous calculons la taille de l'image passée en paramètre grâce à la fonction ***size***.
`tf=zeros(nbLigne,nbColonne);`	Nous créons une matrice nulle de la même taille que l'image passée en paramètre de la fonction. Cette matrice permettra de stocker la transformée de Fourier de l'image. C'est donc elle qui sera renvoyée à la fin du programme.
`for u=1:nbLigne` `  for v=1:nbColonne`	Nous parcourons tous les pixels de tf, image qui sera renvoyée à la fin du programme. Pour chacun de ces pixels, il faut calculer la transformée de Fourier du pixel qui leur correspond dans l'image passée en paramètre.

`for x=1:nbLigne`
`  for y=1:nbColonne`
`  tf(u,v) = tf(u,v) + img(x,y) *`
`  exp((-2*1i*pi)*`
`  (((u-1) *(x-1)/nbLigne)`
`  + ((v-1)*(y-1) /`
`  nbColonne)));`
	Nous appliquons la formule évoquée précédemment pour chaque pixel de l'image à renvoyer. Une différence est à noter par rapport à la formule d'origine : les indices des valeurs dans une matrice commencent à 1 dans MatLab contrairement à ceux utilisés dans la formule théorique de la transformée de Fourier où les indices commencent à 0. Il faut donc appliquer cette spécification lors du calcul de la transformée de Fourier dans notre algorithme (voir le code ci-contre).

III. Transformée de Fourier rapide 1D

3. Directe

La version rapide de la transformée de Fourier est une implémentation de l'algorithme qui permet de passer d'une complexité élevée en $O(N^2)$ à une complexité beaucoup plus intéressante en $O(n\log_2 n)$. Elle se base sur un type d'algorithme appelé « Divide and Conquer » (Diviser pour régner en français) qui part du principe qu'il vaut mieux traiter récursivement deux tableaux de taille n/2 plutôt qu'un gros tableau de taille n.

Si on applique ce principe à notre problème, il est alors plus intéressant d'appliquer récursivement la transformée de Fourier sur les pixels pairs de notre image 1D puis sur les pixels impairs et enfin d'assembler le tout. C'est le principe de l'algorithme de Cooley-Tukey.

Plutôt qu'avoir $\hat{I}(u) = \sum_{x=0}^{N-1} I(x) * \exp(\frac{-2i\pi ux}{N})$ où N est la taille de la matrice et Î sa transformée.

On a alors $\hat{I}(u) = \sum_{x=0}^{(\frac{N}{2})-1} I(2x) * \exp\left(\frac{-2i\pi ux}{\frac{N}{2}}\right) + \sum_{x=0}^{(\frac{N}{2})-1} I(2x+1) * \exp\left(\frac{-2i\pi u(2x+1)}{N}\right)$

Soit en français : Transformée de Fourier d'un pixel = transformée de Fourier des pixels pairs + Transformée de Fourier des pixels impairs

Si on développe le membre de droite, on obtient

$$\hat{I}(u) = \text{TFPairs} + \sum_{x=0}^{(\frac{N}{2})-1} I(2x+1) * \exp\left(\frac{-2i\pi u 2x}{N} + \frac{2i\pi u}{N}\right)$$

$$\hat{I}(u) = \text{TFPairs} + \sum_{x=0}^{(\frac{N}{2})-1} I(2x+1) * \exp\left(\frac{-2i\pi u 2x}{N}\right) * \exp(\frac{-2i\pi u}{N})$$

$$\hat{I}(u) = \text{TFPairs} + \exp\left(\frac{-2i\pi u}{N}\right) * \sum_{x=0}^{(\frac{N}{2})-1} I(2x+1) * \exp\left(\frac{-2i\pi u x}{\frac{N}{2}}\right)$$

$$\hat{I}(u) = \text{TFPairs} + \exp\left(\frac{-2i\pi u}{N}\right) * \text{TFImpairs (1)}$$

La formule pour trouver la transformée de Fourier d'un pixel compris entre 0 et (N/2)-1 est donc la formule (1). Cependant, on voit ici que seule la première moitié des pixels du tableau initial (de taille N) sont traités. On cherche alors à trouver une formule qui nous permet de trouver la valeur en u+(N/2) pour chaque valeur de u.

$$\hat{I}(u) = \text{TFPairs} + \exp\left(\frac{-2i\pi u}{N}\right) * \text{TFImpairs (1)}$$

$$\hat{I}\left(u + (\frac{N}{2})\right) = \text{TFPairs} + \exp\left(\frac{-2i\pi(u + \frac{N}{2})}{N}\right) * \text{TFImpairs}$$

$$\hat{I}\left(u + (\frac{N}{2})\right) = \text{TFPairs} + \exp\left(\frac{-2i\pi * u}{N} + \frac{-2i\pi * \frac{N}{2}}{N}\right) * \text{TFImpairs}$$

$$\hat{I}\left(u + (\frac{N}{2})\right) = \text{TFPairs} + \exp\left(\frac{-2i\pi u}{N} - i\pi\right) * \text{TFImpairs}$$

$$\hat{I}\left(u + (\frac{N}{2})\right) = \text{TFPairs} + \exp(-i\pi) * \exp\left(\frac{-2i\pi u}{N}\right) * \text{TFImpairs}$$

$$\hat{I}\left(u + (\frac{N}{2})\right) = \text{TFPairs} + (\cos(-\pi) + isin(-\pi)) * \exp\left(\frac{-2i\pi u}{N}\right) * \text{TFImpairs}$$

$$\hat{I}\left(u + (\frac{N}{2})\right) = \text{TFPairs} + (-1 + i * 0) * \exp\left(\frac{-2i\pi u}{N}\right) * \text{TFImpairs}$$

$$\hat{I}\left(u + (\frac{N}{2})\right) = \text{TFPairs} - \exp\left(\frac{-2i\pi u}{N}\right) * \text{TFImpairs (2)}$$

Grâce aux formules (1) et (2), on peut maintenant obtenir les valeurs des transformées de Fourier de tous les pixels d'une image 1D en utilisant la méthode du « Divide and Conquer ». Nous avons alors implémenté l'algorithme suivant, qui fonctionne sur une image 1D horizontale (nombre de lignes = 1) :

Code réalisé	Explications
`function [tf] = TFRapide1D(Img)`	La fonction créée a pour nom TFRapide1D. Elle prend en paramètre une matrice horizontale à une dimension représentant l'image sur laquelle nous voulons appliquer la transformée de Fourier. Elle renvoie la transformée de Fourier de l'image passée en paramètre.
`N = size(Img,2);`	On récupère le nombre de pixels (nombre de colonnes de la matrice 1D) à traiter.
`TF = Img ;`	La transformée de Fourier d'une image 1D est une autre image de même taille. Grâce à cette ligne de code, on initialise la taille de la variable de retour.
`if (n ~= 1)`	Si la taille de la matrice est supérieure à 1, alors il faut appliquer l'algorithme Divide and Conquer. Sinon, la transformée de Fourier d'une matrice de taille 1 est elle-même. Puisque la variable de retour a été initialisée pour être égale à la matrice en entrée, il n'y a rien d'autre à faire.
`pairs = Img(1:2:(n-1));` `impairs = Img(2:2:n);`	On sépare les pixels pairs et impairs de l'image en entrée. On utilise pour cela les techniques de sélection propres à Matlab, de la forme suivante : première_valeur:pas:dernière_valeur. A noter qu'on voit à partir d'ici apparaître un problème dû à Matlab : Les formules trouvées précédemment utilisent des indices qui vont de 0 à (N/2)-1 alors que les indices du langage commencent à 1. Le pixel d'indice 1 (impair donc) sur Matlab équivaut alors au pixel 0 (pair) de notre raisonnement. C'est pour cela que les pixels impairs de Matlab finissent par être les pixels pairs et vice versa.
`TFPairs = TFRapide1D(pairs);` `TFImpairs = TFRapide1D(impairs)`	On calcule récursivement les transformées de Fourier des pixels pairs et impairs.
`for u=1:n/2`	On remplit ensuite l'image de retour.

`coef = exp(-2*pi*1i* (u-1) / n);`
`TF(u) = TFPairs(u) + coef * TFImpairs(u);`
`TF(u+(n/2)) = TFPairs(u) - coef * TFImpairs(u);`
	On calcule d'abord le coefficient en commun dans les formules (1) et (2). U devient (u-1) puisque l'indice commence à 1.
	Enfin on applique les formules trouvées précédemment.

Grâce à cet algorithme, on peut obtenir la transformée de Fourier rapide d'une image 1D passée en paramètre.

4. Inverse

Pour trouver la transformée de Fourier rapide inverse d'une image 1D, il suffit d'inverser le signe devant le 2 du coefficient. Cette manipulation vient tout simplement du fait que la formule pour trouver la transformée de Fourier inverse est la suivante.

$$F'(\hat{I}(u)) = \sum_{x=0}^{N-1} \hat{I}(u) * \exp(\frac{2i\pi ux}{N})$$

L'inverse d'une transformée de Fourier d'une image permet de retomber sur cette dernière multipliée par un scalaire, la taille de la matrice. Il faut alors penser à rediviser la variable résultat par n.

IV. Transformée de Fourier rapide 2D

1. Directe

Pendant le cours, nous avons vu qu'une transformée de Fourier discrète 2D était presque équivalente au résultat de l'application d'une transformée de Fourier 1D sur chaque ligne d'une image, puis d'une nouvelle application sur chaque colonne du résultat précédent (autrement dit il s'agit d'une composition de fonctions). Pour l'implémentation de notre transformée de Fourier 2D, nous avons donc choisi d'utiliser ce principe.

Code réalisé	Explications
`function [tf] = TFRapide2D(Img)`	La fonction créée a pour nom TFRapide2D. Elle prend en paramètre une matrice à deux dimensions représentant l'image sur laquelle nous voulons appliquer la transformée de Fourier. Elle renvoie la transformée de Fourier de l'image passée en paramètre.
`[nbLignes, nbColonnes] = size(Img);`	On récupère les dimensions de l'image.

`for ligne=1:nbLignes`
`  TF(ligne,:) = FRapide1D(TF(ligne,:));`
`end`
	Pour chaque ligne de l'image, on applique la transformée de Fourier 1D.
	TF(ligne,:) permet de retourner les valeurs de toutes les colonnes de la ligne « ligne » dans une matrice 1D horizontale.

`for colonne=1:nbColonnes`
`  TF(:,colonne) =`
`TFRapide1D(TF(:,colonne).');`
`end`
	Pour chaque colonne de la matrice calculée précédemment, on applique la transformée de Fourier 1D.
	L'opérateur « .' » permet d'obtenir la transposée de la matrice TF(:,colonne). La transposée d'une matrice A est une autre matrice A' où les colonnes de la première sont devenues les lignes de la seconde, et vice versa. TF(:,colonne) correspondant à un vecteur vertical contenant les valeurs de toutes les lignes de la colonne précisée, sa transposée sera juste l'équivalent mais sous la forme d'une matrice horizontale.

On reviendra dans la partie V sur les limites de cette méthode.

2. Inverse

Le principe pour obtenir l'inverse d'une transformée de Fourier est exactement le même, mais évidemment on utilise l'algorithme de calcul de la transformée de Fourier inverse. Comme pour la 1D, il faut bien penser à la fin à diviser le résultat par la taille de la matrice pour retomber exactement sur l'image de base.

V. En plus

1. Tatouage

Parmi les avantages apportés par la transformée, on peut parler de la possibilité de tatouer une image. Pour cela, on place un message ou un symbole distinctif en bas à droite de la transformée de Fourier de l'image, puis lors de l'inversion celui-ci va se retrouver disperser parmi les pixels. A l'œil nu il sera alors impossible de faire la distinction entre l'image tatouée et l'image originale mais une simple nouvelle transformée de Fourier permettra de révéler le message caché.

Nous avons pris pour notre projet l'exemple du tatouage des initiales « AL » au sein d'une image. Voici l'algorithme créé :

```
function [iout] = tatouageInitial (img)
    %On applique la transformée de Fourier à l'image avant de dessiner les
    %initials
    out = fft2(img);
    %On calcule la taille de l'image
    [nbLigne,nbColonne]=size(img);
    %On dessine les lettres A et L en gris pixel par pixel en bas à droite
    %de l'image
    for k=0:7
            %On dessine la barre de droite, de gauche et centrale du A.
            out(nbLigne-k,nbColonne-k-15)=0.5;
            out(nbLigne-k,nbColonne-29+k)=0.5;
            out(nbLigne-3,nbColonne-k-18)=0.5;
            %On dessine les barres verticales et horizontales du L.
            out(nbLigne-k,nbColonne-13)=0.5;
            out(nbLigne,nbColonne-k-5)=0.5;
    end
    %On applique la transformée de Fourier inverse
    iout = ifft2(out);
    %Affichage
    imshow(iout)
end
```

Dans un premier temps, nous appliquons la Transformée de Fourier à l'image passée en paramètre afin de pouvoir cacher notre message au sein de cette dernière. Nous décidons d'écrire notre message en bas à droite de l'image. Pour cela, nous créons une boucle au sein de laquelle nous modifions la couleur de quelques pixels afin d'obtenir les lettres A et L en gris (valeur=0.5). Enfin, nous appliquons la transformée de Fourier inverse pour retrouver l'image d'origine avec les modifications faites précédemment, invisibles à l'œil nu.

En tant que tel, le tatouage fonctionne. Cependant, il n'utilise pas nos fonctions de transformées. Il y a une raison très simple à cela, et elle est mentionnée dans la partie sur la transformée de Fourier rapide 2D. En effet, nous avons dit à ce moment-là qu'utiliser l'enchainement de deux transformées de Fourier successives permettait d'obtenir un résultat quasi équivalent à celui d'une transformée 2D directe. C'est le « quasi » qui est important dans la phrase, puisque le résultat est égal à la fonction fft à quelques rares signes près. De fait, lorsqu'on utilise la transformée pour de l'analyse fréquentielle en analysant et comparant les spectres, cette différence est complètement négligeable. Cependant, dans notre cas pour le tatouage, elle empêche la conservation des initiales après inversion.

Malgré le fait que le tatouage en utilisant nos fonctions ne fonctionne pas, nous tenions à mentionner cette partie dans ce rapport pour deux raisons. Tout d'abord parce que le tatouage est une fonctionnalité très utile et très simple possible grâce à la transformée de Fourier, ce qui en faisait un concept tout à fait en rapport avec notre sujet. Et ensuite parce qu'en parler nous permettait ainsi d'exposer les limites de la méthode que nous avons décidé d'utiliser pour l'implémentation de nos transformées 2D.

VI. Conclusion

Pour conclure, ce projet nous aura permis de travailler plus en profondeur le concept de la transformée de Fourier et surtout son fonctionnement. La principale difficulté rencontrée aura surtout été d'appréhender le fait que les indices de Matlab commencent à 1 alors que les indices dans nos formules commencent à 0. Afin de détecter ce problème, qui nous aura quand même bien occupé pendant plusieurs heures, il a fallu reprendre plusieurs fois étape par étape les explications et les résultats obtenus, ce qui aura eu finalement pour conséquence de nous obliger à tout comprendre.

VII. Bibliographie

Cours d'Outils Mathématiques pour l'Informatique

http://jl.baril.u-bourgogne.fr/Licence3.html

Pages Wikipédia

https://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm

https://fr.wikipedia.org/wiki/Transformation_de_Fourier_rapide