Licence Sciences et Technologies – 3ème année
Module Outils mathématique pour l'informatique

Rapport

Projet outils mathématique pour l'informatique

Table des matières

INTRODUCTION

Dans ce projet ne sera traité que la transformée de Fourier discrète, qui constitue un équivalent discret à la transformée de Fourier, ce qui signifie sur un nombre fini de points (dans un tableau à 1 ou 2 dimensions).

Nous nous concentrerons également sur l'implémentation des versions dites « rapides » des transformées de Fourier. Celles-ci ont été créées afin de diminuer la complexité algorithmique, ce qui rend les calculs plus rapides pour un ordinateur.

IMPLÉMENTATION DES TRANSFORMÉS DE FOURIER DISCRÈTES 1D ET 2D

-Transformée de Fourier discrète 1D directe et inverse

La transformée de Fourier discrète 1D traite un vecteur d'1 dimension via la formule suivante :

$$\hat{g}(u) = F\big(g(x)\big) = \sum_{x=0}^{N-1} g(x)\, exp\left(-\frac{2i\pi ux}{N}\right) \qquad avec\ u = 0..N-1$$

Ici u est l'indice du vecteur où placer la valeur calculée (ĝ), x lui est l'indice du vecteur d'origine (g). L'algorithme Scilab est donc le suivant :

```
function res = maTF1D(tab)
    res =tab
    N=size(tab)(2);                                  //récupération de la taille du vecteur D
    for u=1:N                                        //parcour du vecteur d'arrivé
        res(u)=0;
        for x = 1:N                                  //parcour du vecteur source
            res(u)= res(u)+ tab(x)*exp((-2*%i*%pi*(u-1)*(x-1))/N);
        end
    end
endfunction
```

On constate le parcours de deux boucles (parcours des deux tableaux), la complexité est donc $O(N^2)$. Les cases du tableau ĝ sont tout d'abord mises à 0, puis on ajoute à chacune des cases de ĝ l'addition de toutes les cases de g, multipliées par $exp\left(-\frac{2i\pi ux}{N}\right)$. Il est nécessaire de préciser que les indices des vecteurs commencent à 1 et non 0.

La transformée de Fourier discrète 1D inverse est très semblable à sa version directe, voici sa formule :

$$F'(\hat{g}(u)) = g(x) = \sum_{u=0}^{N-1} \hat{g}(u)\, exp\left(\frac{2i\pi ux}{N}\right) \qquad avec\ x = 0..N-1$$

Voici l'algorithme scilab :

```
function res = maTFI(tab)
    res =tab
    N=size(tab)(2);
    for x=1:N
        res(x)=0;
        for u = 1:N
            res(x)= res(x)+ tab(u)*exp((2*%i*%pi*(x-1)*(u-1))/N)/N;        //division par N pour le coefficient
        end
    end
endfunction
```

Il y a deux différences entre cette version et la directe, le signe à l'intérieur de l'exponentielle, et la division par N à chaque nouvelle addition dans la case du vecteur d'arrivé. Cette division sert afin de retrouver le vecteur d'origine, après le passage de la transformée de Fourier directe.

La complexité de cet algorithme est de O(N^2), comme pour la version directe.

Cependant il semblerait que notre version de ces deux algorithmes ne soient pas assez précises. En effet on peut constater que l'une des valeurs du vecteur n'est pas bonne :

```
-->A=[1 2 5 4 6 8 9 1]
 A  =

    1.    2.    5.    4.    6.    8.    9.    1.
```

Création du vecteur

```
-->fft(A)
 ans  =

         column 1 to 5

    36. + 0.i   -11.363961 + 6.1213203i   -7. - 5.i    1.363961 - 1.8786797i    6. + 0.i

         column 6 to 8

    1.363961 + 1.8786797i   -7. + 5.i   -11.363961 - 6.1213203i
```

Transformée de Fourier de Scilab

Une valeur ne correspond pas (dû au format des complexes de scilab)

```
-->maTF1D(A)
 ans  =

         column 1 to 5

    36. + 0.i   -11.363961 + 6.1213203i   -7. - 5.i    1.363961 - 1.8786797i    6. + 3.307D-15i

         column 6 to 8

    1.363961 + 1.8786797i   -7. + 5.i   -11.363961 - 6.1213203i
```

Notre transformée de Fourier

```
-->ifft(A)
 ans  =

         column 1 to 5

    4.5 + 0.i   -1.4204951 - 0.765165i   -0.875 + 0.625i    0.1704951 + 0.234835i    0.75 + 0.i

         column 6 to 8

    0.1704951 - 0.234835i   -0.875 - 0.625i   -1.4204951 + 0.765165i
```

Transformée de Fourier inverse de Scilab

De même ici

```
-->maTFI(A)
 ans  =

         column 1 to 4

    4.5 + 0.i   -1.4204951 - 0.765165i   -0.875 + 0.625i    0.1704951 + 0.234835i

         column 5 to 8

    0.75 - 4.133D-16i    0.1704951 - 0.234835i   -0.875 - 0.625i   -1.4204951 + 0.765165i
```

Notre transformée de Fourier inverse

```
-->real(maTFI(maTF1D(A)))
 ans  =

    1.    2.    5.    4.    6.    8.    9.    1.
```

Pourtant nous arrivons à reconstituer le vecteur d'origine grâce aux parties réelles des valeurs

-Transformée de Fourier discrète 2D directe et Inverse

Maintenant nous allons traiter la transformée de Fourier 2D directe. Ici 2D signifie que la transformée sera réalisée sur une matrice et non plus un tableau 1D (ou vecteur). Voici l'algorithme Scilab :

```
function res=maTF2D(tab)
    [rows,n]=size(tab)
        res = tab
        //LIGNES
        for i = 1:rows
            res(i,:) = maTF1D(tab(i,:))
        end
        
        //Tourne la matrice
        for i = 1:n
            for j = 1:rows
                tabInv(i,j) = res (j,i)
            end
        end
        
        //LIGNE (Anciennes colonnes)
        for i = 1:n
            ret2(i,:) = maTF1D(tabInv(i,:))
        end
        
        //Tourne la matrice
       for i = 1:rows
            for j = 1:n
                res(i,j) = ret2 (j,i)
            end
        end
endfunction
```

Celui-ci réalise une transformée de Fourier 1D naïve sur chacune des lignes de la matrice source, résultat que l'on place dans une nouvelle matrice. Puis on effectue une rotation de la matrice source, afin de transformer ses colonnes en lignes comme suit :

$$\begin{array}{|c|c|}\hline 1 & 2 \\ \hline 3 & 4 \\ \hline \end{array} \longrightarrow \begin{array}{|c|c|}\hline 1 & 3 \\ \hline 2 & 4 \\ \hline \end{array}$$

Puis on refait une transformée de Fourier sur les lignes, on placera les résultats dans une nouvelle matrice. Enfin, on place dans la matrice résultat la rotation de la matrice intermédiaire.

En ce qui concerne la transformée de Fourier 2D inverse :

```
function res=maTFI2D(tab)
    [rows,n]=size(tab)
        res = tab
        //LIGNES
        for i = 1:rows
            res(i,:) = maTFI(tab(i,:))
        end

        //Tourne la matrice
        for i = 1:n
            for j = 1:rows
                tabInv(i,j) = res (j,i)
            end
        end

        //LIGNE (Anciennes colonnes)
        for i = 1:n
            ret2(i,:) = maTFI(tabInv(i,:))
        end

        //Tourne la matrice
       for i = 1:rows
            for j = 1:n
                res(i,j) = ret2 (j,i)
            end
        end
endfunction
```

Le fonctionnement est le même que l'algorithme précédent, sauf que l'on appelle la transformée de Fourier inverse.

TRANSFORMÉE DE FOURIER DISCRÈTE RAPIDE 1D

Jusqu'à présent, nous parcourions les tableaux de gauche à droite à calculant au fur et à mesure les valeurs. Cette méthode peut être optimisée car elle occasionne beaucoup de calculs inutiles.

Ici l'astuce est de séparer le tableau en deux tableaux stockant les valeurs des indices pairs, et l'autre les valeurs des indices impairs, et ceci de manière récursive.

-Explications mathématique

Premièrement, afin que cet algorithme fonctionne, il faut que la taille du tableau soit $N=2^n$.
On pose $\hat{g}(u)=F(g(x))$ la transformée de Fourier sur la fonction g et à l'indice u du tableau résultat, par la fonction F à l'indice x du tableau d'origine (on part du principe que le premier indice est 0) :

$$\hat{g}(u) = \sum_{x=0}^{N-1} g(x)\, exp\left(-\frac{2i\pi ux}{N}\right) \qquad avec\ u = 0..N-1$$

Séparation en deux sommes, une avec les indices pairs, une autre avec les indices impairs :

$$= \sum_{\substack{x=0 \\ x\ pair}}^{N-1} g(x)\, exp\left(-\frac{2i\pi ux}{N}\right) + \sum_{\substack{x=0 \\ x\ impair}}^{N-1} g(x)\, exp\left(-\frac{2i\pi ux}{N}\right)$$

Traitons la somme des indices pairs :

On pose x=2x' afin de ne prendre que les indices pairs Puis on modifie la borne supérieure du tableau :

$$= \sum_{x'=0}^{\frac{N}{2}-1} g(2x')\, exp\left(-\frac{2i\pi u 2x'}{N}\right) + \cdots$$

$$= \sum_{x=0}^{\frac{N}{2}-1} g(2x)\, exp\left(-\frac{2i\pi u x}{\frac{N}{2}}\right) + \cdots$$

Traitons la somme des indices impairs :

$$\ldots + \sum_{x'=0}^{\frac{N}{2}-1} g(2x'+1)\, exp\left(-\frac{2i\pi u(2x'+1)}{N}\right)$$

$$\ldots + \sum_{x=0}^{\frac{N}{2}-1} g(2x+1)\, exp\left(-\frac{2i\pi u(2x+1)}{N}\right)$$

On développe :

$$\ldots + \sum_{x=0}^{\frac{N}{2}-1} g(2x+1)\, exp\left(-\frac{2i\pi u 2x}{N} + \frac{-2i\pi u}{N}\right)$$

$$\ldots + \sum_{x=0}^{\frac{N}{2}-1} g(2x+1)\, exp\left(-\frac{2i\pi u 2x}{N}\right) exp\left(-\frac{2i\pi u}{N}\right)$$

On peut sortir la deuxième exponentielle de la somme car elle ne dépend pas de x :

$$\ldots + exp\left(-\frac{2i\pi u}{N}\right) \sum_{x=0}^{\frac{N}{2}-1} g(2x+1)\, exp\left(-\frac{2i\pi u 2x}{N}\right)$$

$$\ldots + exp\left(-\frac{2i\pi u}{N}\right) \sum_{x=0}^{\frac{N}{2}-1} g(2x+1)\, exp\left(-\frac{2i\pi u x}{\frac{N}{2}}\right)$$ Finalement :

Finalement :

$$= \sum_{x=0}^{\frac{N}{2}-1} g(2x)\, exp\left(-\frac{2i\pi u x}{\frac{N}{2}}\right) + exp\left(-\frac{2i\pi u}{N}\right) \sum_{x=0}^{\frac{N}{2}-1} g(2x+1)\, exp\left(-\frac{2i\pi u x}{\frac{N}{2}}\right)$$

$$= \hat{g}^{pair}(u) + exp\left(-\frac{2i\pi u}{N}\right) \hat{g}^{impair}(u)$$

-Implémentation de l’algorithme et complexité

Voici l’algorithme Scilab :

```
function res=maTFR1D(tab)
    [rows,N]=size(tab)                          //n doit être une puissance de 2 (vectueur colonne avec une seule ligne car on est en 1D)
    if N<=1 then
        res = tab                               //si taille tableau=1, retourne tab
    else
        Wn=exp((-2*%pi*%i)/N)
        W=1

        impair=tab(2:2:N)                       //commence à l'indice 2, pas de 2, jusqu'à n
        pair=tab(1:2:(N-1))                     //commence à l'indice 1, pas de 2, jusqu'à n-1

        ypair=maTFR1D(pair)                     //récursion afin de séparer les tableaux
        yimpair=maTFR1D(impair)

        for u=1:(N/2)
            res(1,u)=ypair(u)+yimpair(u)*W
            res(1,u+(N/2))=ypair(u)-yimpair(u)*W
            W=W*Wn
        end
    end
endfunction
```

Ce programme récursif sépare le vecteur de départ en 2, l’un avec les valeurs dans les indices pairs, l’autre avec les valeurs des indices impairs. Puis grâce à la formule de l’exponentielle de la transformée de Fourier discrète 1D, et en fonction de la taille du tableau actuel, on calcul la valeur à placer dans le tableau résultat.

En ce qui concerne la complexité :

Soit C_N le nombre d’opérations élémentaires, $C_{N/2}$ le nombre d’opérations élémentaires à réaliser sur un tableau de taille N/2, et N pour le nombre d’opérations afin de rassembler les deux tableaux de taille N :

$$C_N = C_{N/2} + C_{N/2} + N$$

$$= \cancel{2C_{N/2}} + N$$

On résout la récurrence, à chaque étape on redivise le tableau en 2, on remultiplie donc le tout par 2 pour « compenser » :

$$\cancel{2C_{N/2}} = \cancel{4C_{N/4}} + \frac{N}{\cancel{2}}$$

$$\cancel{4C_{N/4}} = \cancel{8C_{N/8}} + N$$

$$\cancel{2^{n-1}}C_{N/(\frac{N}{2})} = 2^n C_{N/N} + N$$

$$C_N = N + nN$$

$$= N(log_2 N + 1)$$

$$\approx Nlog_2 N \leq N^2$$

2^n=N

n=log_2N

On trouve donc une complexité plus faible sur la version rapide. Pour N=1024, le temps de calcul peut être 100 fois plus court pour la version rapide.

TRANSFORMÉE DE FOURIER DISCRÈTE RAPIDE 1D INVERSE

-Explications mathématique

Le raisonnement mathématique est exactement le même que précédemment, la formule est la suivante :

$$F'\big(\hat{g}(u)\big) = g(x) = \sum_{x=0}^{N-1} g(x)\, exp\left(\frac{2i\pi ux}{N}\right) \qquad avec\ x = 0..N-1$$

Puis finalement :

$$\begin{aligned} g(x) \ &= \sum_{u=0}^{\frac{N}{2}-1} \hat{g}(2u)\, exp\left(\frac{2i\pi ux}{\frac{N}{2}}\right) + exp\left(\frac{2i\pi x}{N}\right) \sum_{u=0}^{\frac{N}{2}-1} \hat{g}(2u+1)\, exp\left(\frac{2i\pi xu}{\frac{N}{2}}\right) \\ &= g^{pair}(x) + exp\left(\frac{2i\pi x}{N}\right) g^{impair}(x) \end{aligned}$$

Tout le développement est strictement le même que pour la transformée de Fourier rapide directe 1D, la seule différence étant le signe à l'intérieur des exponentielles.

-Implémentation de l'algorithme et complexité

L'algorithme employé est le suivant :

```
function res=maTFRI1D(tab)
    res=conj(maTFR1D(conj(tab)))/size(tab)(2)
endfunction
```

Ici nous avons opté pour une conjugaison complèxe. En effet, une méthode existe afin de programmer une transformée de Fourier rapide inverse 1D en passant par la version directe. Il faut utiliser la formule suivante :

$$IFFT(X) = \frac{1}{N} conj(FFT(conj(X)))$$

La complexité reste la même avec la transformée de Fourier rapide 1D directe.

TRANSFORMÉE DE FOURIER DISCRÈTE RAPIDE 2D DIRECTE ET INVERSE

Maintenant nous abordons les transformées de Fourier rapide 2D directe et inverse.

-Explications mathématique et complexité

Voici la formule de la version directe :

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

Voici la formule de la version inverse :

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

Ici on constate qu'il sera nécessaire de parcourir les lignes et les colonnes de 2 matrices, donnant une complexité de O(N^4). On utilisera alors une transformée de Fourier rapide sur les lignes de la matrice source, comme utilisé précédemment, afin de diminuer les calculs de la machine.

-Implémentation des algorithmes

Version directe :

```
function res=maTFR2D(tab)
        [rows,n]=size(tab)

        //LIGNES
        for i = 1:rows
            res(i,:) = maTFR1D(tab(i,:))
        end

        //Tourne la matrice
        for i = 1:n
            for j = 1:rows
                tabInv(i,j) = res (j,i)
            end
        end

        //LIGNE (Anciennes colonnes)
        for i = 1:n
            ret2(i,:) = maTFR1D(tabInv(i,:))
        end

        //Tourne la matrice
       for i = 1:rows
            for j = 1:n
                res(i,j) = ret2 (j,i)
            end
        end
endfunction
```

Version inverse

```
function res=maTFRI2D(tab)
        [rows,n]=size(tab)

        //LIGNES
        for i = 1:rows
            //ret(i,:) = maTFRI1D(tab(i,:))
            res(i,:) = conj(maTFR1D(conj(tab(i,:))))/size(tab)(2)
        end

        //Tourne la matrice
        for i = 1:n
            for j = 1:rows
                tabInv(i,j) = res (j,i)
            end
        end

        //LIGNE (Anciennes colonnes)
        for i = 1:n
            //ret2(i,:) = maTFRI1D(tabInv(i,:))
            ret2(i,:) = conj(maTFR1D(conj(tabInv(i,:))))/size(tabInv)(2)
        end

        //Tourne la matrice
       for i = 1:rows
            for j = 1:n
                res(i,j) = ret2 (j,i)
            end
        end

        //res = real(res);
endfunction
```

Ces programmes sont très semblables à la transformée de Fourier 2D, la seule différence étant l'appelle des transformées de Fourier rapides afin de calculer les lignes des matrices.