Guillaume BELDILMI & Guillaume GAUGUET

Outils mathématiques pour l'informatique : Rapport de projet

Intitulé du sujet

- Programmer la transformée de Fourier directe discrète pour une image
- Programmer la transformée de Fourier rapide discrète 1D
- Programmer la transformée de Fourier rapide discrète 2D
- Faire un rapport (au maximum 12 pages) expliquant les algorithmes utilisés
- Même travail pour la transformée de Fourier inverse rapide 1D et 2D

I. Présentation de la transformation de Fourier

La transformation de Fourier veut montrer que tout signal périodique peut être traduit comme étant la somme d'un ensemble de signaux sinusoïdaux.
Pour ce projet, nous nous intéresserons uniquement à l'application au domaine discret de la transformation de Fourrier.
La transformée de Fourier doit utiliser une fonction périodique pour fonctionner, or dans les matrices à une ou deux dimensions, que nous donnerons, il n'y aura majoritairement aucune période de présente. Mais pour résoudre ce problème, nous disons que la matrice donnée est la boucle complète de la fonction à donner à la transformée de Fourier, ainsi, imaginairement, si on continuait en dehors de la matrice donnée, on retrouverait la même. C'est donc par ce fait que les algorithmes analysent la matrice donnée, comme représentante d'un cycle complet d'une fonction. Avec ce type de données, on peut utiliser la transformée de Fourier sur un ensemble discret.

II. Des mathématiques à l'algorithmique

Pour ce projet, nous avons utilisé le langage de programmation Python ainsi que la librairie `numpy` afin de manipuler aisément les expressions et calculs mathématiques.

1. Transformée de Fourier directe discrète

La transformée de Fourier directe discrète est donnée par la formule suivante :

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

La transformée de Fourier issue de cette formule peut être transcrite en cette fonction :

```
import numpy as np

def ft(X):
    """Fourier Transform (FT)"""
    N = len(X)
    if N == 1:
        return X
```

```
    else:
        x = np.zeros(N, dtype=complex)
        for k in range(N):
            x[k] = np.sum(X*np.exp(-2j*np.pi*k*np.arange(N)/N))
        return x
```

2. Inverse de la transformée de Fourier directe discrète

L'inverse de la transformée de Fourier directe discrète est donnée par la formule suivante :

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

Cette formule est quasiment identique à la formule précédente (à un facteur près). Les fonctions `ft` et `ift` se ressemblent alors tout naturellement.

```
import numpy as np

def ift(X):
    """Inverse Fourier Transform (IFT)"""
    N = len(X)
    if N == 1:
        return X
    else:
        x = np.zeros(N, dtype=complex)
        for k in range(N):
            x[k] = np.sum(X*np.exp(2j*np.pi*k*np.arange(N)/N))
        return x/N
```

3. Transformée de Fourier rapide discrète

L'algorithme de la transformée de Fourier directe discrète a une complexité algorithmique de l'ordre $O(n) = n^2$. Afin d'économiser en temps de calcul, on y préfèrera une variante moins gourmande.
En reprenant la formule précédente, on sépare la somme en deux (une partie pour les valeurs ayant un index pair et une partie pour les valeurs ayant un index impair), puis on simplifie chaque somme :

$$\hat{I}(u) = \sum_{x=0}^{N-1} I(x) \exp\left(\frac{-2i\pi ux}{N}\right)$$
$$\hat{I}(u) = \sum_{x=0}^{N-2} I(x) \exp\left(\frac{-2i\pi ux}{N}\right) + \sum_{x=0}^{N-1} I(x) \exp\left(\frac{-2i\pi ux}{N}\right)$$
$$\hat{I}(u) = \sum_{x=0}^{\frac{N}{2}-1} I(2x) \exp\left(\frac{-2i\pi u2x}{N}\right) + \sum_{x=0}^{\frac{N}{2}-1} I(2x+1) \exp\left(\frac{-2i\pi(2x+1)u}{N}\right)$$
$$\hat{I}(u) = \sum_{x=0}^{\frac{N}{2}-1} I(2x) \exp\left(\frac{-2i\pi ux}{\frac{N}{2}}\right) + \sum_{x=0}^{\frac{N}{2}-1} I(2x+1) \exp\left(\frac{-2i\pi 2xu}{N}\right) \exp\left(\frac{-2i\pi u}{N}\right)$$
$$\hat{I}(u) = \sum_{x=0}^{\frac{N}{2}-1} I(2x) \exp\left(\frac{-2i\pi ux}{\frac{N}{2}}\right) + \exp\left(\frac{-2i\pi u}{N}\right) \sum_{x=0}^{\frac{N}{2}-1} I(2x+1) \exp\left(\frac{-2i\pi xu}{\frac{N}{2}}\right)$$

Ainsi, nous avons pour les paires $p = \sum_{x=0}^{\frac{N}{2}-1} I(2x) \exp\left(\frac{-2i\pi ux}{\frac{N}{2}}\right)$ et pour les impaires $q = \exp\left(\frac{-2i\pi u}{N}\right) \sum_{x=1}^{\frac{N}{2}-1} I(2x+1) \exp\left(\frac{-2i\pi xu}{\frac{N}{2}}\right)$.
Cela nous permet d'avoir les valeurs de $\hat{I}(u)$ pour u allant de 0 à $\frac{N}{2}$ par $\hat{I}(u) = p + q$. Cherchons à présent $\hat{I}(u + \frac{N}{2})$.

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

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

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

$$\hat{I}(u+\frac{N}{2}) = \exp(-2i\pi x)(p+q(\exp(-i\pi)))$$

$$\hat{I}(u+\frac{N}{2}) = (\cos(-2\pi x)+i\sin(-2\pi x))(p+q(\cos(-\pi)+i\sin(-\pi)))$$

La variable x est un entier naturel, et 2π représente un tour complet du cercle trigonométrique :
$\cos(-2\pi x) = \cos(0) = 1$ et $\sin(-2\pi x) = \sin(0) = 0$.
Donc, nous avons : $\hat{I}(u+\frac{N}{2}) = (1+i0)(p+q(-1+i0)) = p-q$
Ainsi, nous venons de diviser le précédent calcul de la transformée (qui reposait sur le calcul de l'ensemble du tableau de N valeurs) en une somme de deux tableau de taille $\frac{N}{2}$. Cela nous donne une complexité algorithmique de l'ordre $O(n) = N\log N$, ainsi que le code suivant :

```
import numpy as np

def fft(X):
    """Fast Fourier Transform (FFT)"""
    N = len(X)
    if N == 1:
        return X
    else:
        x = np.zeros(N, dtype=complex)
        xe = fft(X[::2])
        xo = fft(X[1::2])
        for k in range(N//2):
            p = xe[k]
            q = np.exp(-2j*np.pi*k/N) * xo[k]
            x[k] = p+q
            x[k+(N//2)] = p-q
        return x
```

4. Inverse de la transformée de Fourier rapide discrète

Puisque la différence entre la transformation de Fourier et son inverse est juste un facteur, on reprend la fonction `fft` et on change le facteur en question. Ce qui nous donne :

```
import numpy as np

def ifft(X):
    """Inverse Fast Fourier Transform (IFFT)"""
    N = len(X)
    if N == 1:
        return X
    else:
        x = np.zeros(N, dtype=complex)
        xe = ifft(X[::2])
        xo = ifft(X[1::2])
        for k in range(N//2):
            p = xe[k]
```

```
            q = np.exp(2j*np.pi*k/N) * xo[k]
            x[k] = p+q
            x[k+(N//2)] = p-q
        return x/N
```

5. Application à un tableau à deux dimensions

L'application de la transformation de Fourier discrète dans un tableau à deux dimensions est assez simple puisqu'il suffit d'appliquer la transformation de Fourier à chaque ligne du tableau puis de faire la transformée de Fourier de chaque colonne du précedent résultat. L'inverse de la transformée de Fourier d'un tableau à deux dimensions s'obtient de la même façon en appliquant la transformation inverse à chaque colonne/ligne.

```
import numpy as np

def fft2d(X):
    """2D Fast Fourier Transform (FFT)"""
    N, M = X.shape
    x = np.array([fft(X[i]) for i in range(N)])
    x = np.array([fft(x[:,i]) for i in range(M)]).T
    return x

def ifft2d(X):
    """2D Inverse Fast Fourier Transform (FFT)"""
    N, M = X.shape
    x = np.array([ifft(X[i]) for i in range(N)])
    x = np.array([ifft(x[:,i]) for i in range(M)]).T
    return x
```

On notera l'utilisation de la transposée afin de simplifier le code.

III. Illustrations

Les fonctions définies précédemment ont été regroupées et placées dans le fichier `ft.py` afin de pouvoir les importer à partir de différents fichiers.
Nous avons utilisé `matplotlib` afin d'afficher et de pouvoir interprêter plus facilement les résultats.
Dans chacun des exemples, nous effectuons, premièrement, les traitements les uns après les autres sans altérer les valeurs des étapes intermédiaires, ensuite nous effectuons les mêmes traitements en altérant les spectres obtenus par les fonctions `ft`, `fft` et `fft2d`. Pour cela, nous mettons à `0` toutes les valeurs du tableau dont la somme de l'index en `i` et de l'index en `j` dépassent une valeur `N` prédéfinie.

1. `cmp.py`

Cet exemple sert à montrer les résultats obtenus par les fonctions `ft` , `ift`, `fft` et `ifft` sur une suite de valeurs aléatoires ainsi que leur temps de traitement respectif.
Nous pouvons observer que le résultat obtenu par la fonction `fft` diffère de celui obtenu par `ft`, cette différence se répercute naturellement sur le résultat obtenu par `ifft`. On prendra en compte cette légère altération dans le traitement des deux exemples suivants.
De plus, dans le second traitement (déformé), nous remarquons que le signal résultant est aussi altéré

dans les deux cas (`ift` et `ifft`). De plus, on remarque que plus `N` est élevé, moins le signal est altéré. Cela s'explique par le fait que, plus on réduit la valeur de `N`, plus on supprime d'harmoniques du signal. Or, plus une harmonique est élevée, moins elle est significative dans le signal. Car en réduisant la valeur de `N`, on retire des harmoniques de plus en plus significatives dans le signal.

2. `dd.py`

Cet exemple sert à illustrer l'application des fonctions `fft2d` et `ifft2d` sur un tableau à deux dimensions avec des valeurs aléatoires.
On remarquera que le signal d'origine et le résultat de l'inverse de la transformée sans altétation (valeurs mises à `0`) semblent identiques malgré la légère altération faite par `fft`.
N.B.: Dans la représentation de de la transformée de Fourier, nous avons supprimé la valeur du premier pixel afin de rendre plus visible les nuances des valeurs de celle-ci. On appliquera la même modification aux exemples suivants.

3. `gr.py`

Cet exemple sert à illustrer l'application des fonctions `fft2d` et `ifft2d` sur un tableau à deux dimensions avec des valeurs issues d'une image convertie en niveaux de gris.

4. `rgb.py`

Cet exemple sert à illustrer l'application de la transformation de Fourier séparément à chaque composantes (RGB) d'une image pour ensuite reconstruire l'image d'origine.
Contrairement aux exemples précédents, nous avons utilisé les fonctions `numpy.fft.fft2` et `numpy.fft.ifft2` issues de `numpy`. Ces dernières sont plus rapides et les résultats obtenues sont plus faciles à normaliser afin de reconstruire l'image d'origine.