Cours PO7 : Application du principe de "**Diviser pour régner**" : La transformée de Fourier rapide

PC* - Lycée Thiers

Principe de Diviser pour régner

- Principe de **Diviser pour régner** :

Pour résoudre un problème sur une donnée de taille N le réduire récursivement à $K \geqslant 2$ résolutions du problème sur des données de taille N/K.
(souvent $K = 2$).

Exemples :

1. Recherche d'un élément `e` dans un tableau ordonné (sens croissant) :
   - Comparer l'élément `e` avec l'élément médian `m`,
   - Si `e=m` retourné `True`
   - Si `e`$\leqslant$ `m` recommencer avec la première moitié du tableau,
   - Si `e`$\geqslant$ `m` recommencer avec la deuxième moitié du tableau,
   - Si tableau vide retourner `False`.

   De complexité $O(\log(N))$ dans un tableau ordonné de N éléments.
2. Tri rapide. Complexité $O(N\log(N))$ en moyenne. Application au calcul de la médiane.
3. Tri fusion. Complexité $O(N\log(N))$ dans le pire des cas.

La transformée de Fourier discrète

Soit $N \in \mathbb{N}^*$. On rappelle les racines N-ièmes de l'unité :

$$\omega_0 = 1,\ \omega_1 = e^{\frac{2i\pi}{N}},\ \omega_2 = e^{\frac{4i\pi}{N}},\ \ldots,\ \omega_k = e^{\frac{2in\pi}{N}},\ \ldots,\ \omega_{N-1} = e^{\frac{2i(N-1)\pi}{N}}$$

(pour $n \in [[0, N-1]]$). Puissances et conjugués des racines N-ièmes de l'unité sont encore des racines N-ièmes de l'unité.

Soit $x_0, x_1, \ldots, x_{N-1}$ une suite de N nombres (réels ou complexes).

Soit le polynôme de coefficients $x_0, x_1, \ldots, x_{N-1}$:

$$f(t) = x_0 + x_1 t + \cdots + x_n t^n + \cdots x_{N-1} t^{N-1}$$

La **transformée de Fourier discrète** de $(x_0, x_1, \ldots, x_{N-1})$ est la suite des valeurs prises par ce polynôme aux conjugués des racine N-ièmes de l'unité :

$$f(\overline{\omega_0}),\ f(\overline{\omega_1}), f(\overline{\omega_2}), \ldots,\ f(\overline{\omega_n}), \ldots, f(\overline{\omega_{N-1}})$$

Pour tout $n \in [[0, N-1]]$:

$$\widehat{x_n} = f(\overline{\omega_n}) = \sum_{k=0}^{N-1} x_k \left(\overline{\omega_n}\right)^k = \sum_{k=0}^{N-1} x_k e^{\frac{-2ink\pi}{N}}$$

La transformée de Fourier discrète inverse

Soit $x_0, x_1, \ldots, x_{N-1}$ une suite de N nombres (réels ou complexes).

Soit le polynôme de coefficients $x_0, x_1, \ldots, x_{N-1}$:

$$f(t) = x_0 + x_1 t + \cdots + x_n t^n + \cdots x_{N-1} t^{N-1}$$

La transformée de Fourier discrète de $(x_0, x_1, \ldots, x_{N-1})$ est la suite des valeurs prises par ce polynôme aux conjugués des racine N-ièmes de l'unité :

$$f(\overline{\omega_0}),\ f(\overline{\omega_1}), f(\overline{\omega_2}), \ldots,\ f(\overline{\omega_n}), \ldots, f(\overline{\omega_{N-1}})$$

Pour tout $n \in [[0, N-1]]$:

$$\widehat{x_n} = f(\overline{\omega_n}) = \sum_{k=0}^{N-1} x_k \left(\overline{\omega_n}\right)^k = \sum_{k=0}^{N-1} x_k e^{\frac{-2ink\pi}{N}}$$

A partir de la suite $(\widehat{x_0}, \widehat{x_1}, \ldots, \widehat{x_{N-1}})$ (transformée de Fourier discrète) on retrouve la suite $(x_0, x_1, \ldots, x_{N-1})$ par **transformée de Fourier inverse** : pour tout $n \in [[0, N-1]]$:

$$\boxed{x_n = \frac{1}{N} \sum_{k=0}^{N-1} \widehat{x_k} \left(\omega_n\right)^k = \frac{1}{N} \sum_{k=0}^{N-1} \widehat{x_k} e^{\frac{2ink\pi}{N}}}$$

La transformée de Fourier discrète inverse

$$\boxed{x_n = \frac{1}{N}\sum_{k=0}^{N-1} \widehat{x_k}\left(\omega_n\right)^k = \frac{1}{N}\sum_{k=0}^{N-1} \widehat{x_k} e^{\frac{2ink\pi}{N}}}$$

Preuve : Si ω est une racine N-ième de l'unité, alors $\omega^N = 1$.

$$\Longrightarrow\ 0 = \omega^N - 1 = (\omega - 1)\cdot\sum_{k=0}^{N-1}\omega^k$$

$\Longrightarrow$ si $\omega \neq 1$ est une racine N-ième de l'unité alors $\displaystyle\sum_{k=0}^{N-1}\omega^k = 0$.

$$\boxed{\sum_{k=0}^{N-1}\widehat{x_k}\exp\left(\frac{2ink\pi}{N}\right)} = \sum_{k=0}^{N-1}\underbrace{\left(\sum_{j=0}^{N-1} x_j \exp\left(\frac{-2i\pi}{N}kj\right)\right)}_{=\widehat{x_k}}\exp\left(\frac{2i\pi}{N}nk\right)$$

$$= \sum_{j=0}^{N-1} x_j \underbrace{\sum_{k=0}^{N-1}\exp\left(\frac{2i\pi(n-j)}{N}\right)^k}_{=0 \text{ si } j\neq n} = x_n \times \sum_{k=0}^{N-1} 1 = \boxed{x_n \times N} \qquad \blacksquare$$

La transformée de Fourier Discrète

Définition. Soit $X = (x_n)_{0 \leqslant n \leqslant N-1}$ une suite finie de N nombres. La Transformée de Fourier discrète de (x_n) est la suite de N nombres $(TF(X)(n))_{0 \leqslant n \leqslant N-1}$ définie par :

$$TFD(X)(n) = \sum_{k=0}^{N-1} x_k e^{-2i\pi \frac{kn}{N}} \qquad \text{pour tout } n \in [[0, N-1]]$$

• Dans la base canonique de $\mathbb{C}^N$, la transformée de Fourier est l'isomorphisme de matrice associée dans la base canonique, en posant $\omega = \exp(-2i\pi/N)$ (c'est une racine N-ième de l'unité) :

$$M_N(\omega) = \begin{pmatrix} 1 & 1 & \cdots & 1 & 1 \\ 1 & \omega & \cdots & \omega^{N-2} & \omega^{N-1} \\ \vdots & & & & \vdots \\ 1 & \omega^{N-2} & \cdots & \omega^{(N-2)^2} & \omega^{(N-2)(N-1)} \\ 1 & \omega^{N-1} & \cdots & \omega^{(N-1)(N-2)} & \omega^{(N-1)^2} \end{pmatrix}$$

$$\hat{X} = M_n(\omega) \cdot X$$

Sa matrice inverse est : $M_N(\omega)^{-1} = \frac{1}{N} \cdot M_N(\bar{\omega})$

$$X = M_n(\omega)^{-1} \cdot \hat{X}$$

La transformée de Fourier Discrète inverse

Définition. Soit $\hat{X} = (\hat{x}_k)_{0 \leqslant k \leqslant N-1}$ une suite finie de N nombres. La Transformée de Fourier discrète inverse de $(\hat{x}_k)$ est la suite de N nombres $X = (x_k)_{0 \leqslant k \leqslant N-1}$ définie par :

$$x_n = TFI(\hat{X})(n) = \frac{1}{N} \sum_{k=0}^{N-1} \hat{x}_k e^{2i\pi \frac{kn}{N}} \qquad \text{pour tout } n \in [[0, N-1]]$$

- C'est l'opération inverse de la transformée de Fourier discrète :

$$\hat{X} = TFD(X) = M_N(\omega) \times X \quad \Longleftrightarrow \quad X = TFI(\hat{X}) = \frac{1}{N} \cdot M_N(\bar{\omega}) \times \hat{X}$$

- C'est à dire :

$$\hat{x}_n = TFD(X)(n) = \sum_{k=0}^{N-1} x_k e^{-2i\pi \frac{kn}{N}} \qquad \text{pour tout } n \in [[0, N-1]]$$

$$\Longrightarrow x_n = TFI(\hat{X})(n) = \frac{1}{N} \sum_{k=0}^{N-1} \hat{x}_k e^{2i\pi \frac{kn}{N}} \qquad \text{pour tout } n \in [[0, N-1]]$$

TFD et TFI : code "naïf"

- Code pour la TFD :

```
from numpy import exp, pi

def TFD(x):
    N= len(x)
    omega = exp(-2j*pi/N)
    M = []
    for k in range(N):
        M.append([(omega ** k) **i for i in range(N)])
    Mat = np.array(M)
    return np.dot(Mat,x)
```

- Code pour la TFI :

```
def TFI(x):
    N= len(x)
    omega_c = exp(2j*pi/N)
    M = []
    for k in range(N):
        M.append([(omega_c ** k) ** i for i in range(N)])
    Mat = np.array(M)
    return np.dot(Mat,x) / N
```

- Complexité temporelle quadratique $\Theta(N^2)$: $\Theta(N^2)$ opérations pour constituer M puis $\Theta(N^2)$ opérations pour le produit matriciel retourné.

TFD et TFI

```
>>> X = np.arange(1,10)    # Suite finie X
[1 2 3 4 5 6 7 8 9]

>>> TX = TFD(X)    # TFD de X
[ 45.0 +0.j -4.5+12.36364839j -4.5 +5.36289117j, ...
...  -4.5 -2.59807621j -4.5 -5.36289117j -4.5-12.36364839j]

>>> TI = TFI(TX)    # TFI de TFD(X)
[ 1. -9.67127613e-15j 2. -9.62193288e-15j 3. -5.32907052e-15j,
..., 7. +1.13489465e-15j 8. +1.97372982e-15j 9. +6.90805438e-15j]

>>> abs(TI)    # C'est mieux en prenant les modules
[ 1. 2. 3. 4. 5. 6. 7. 8. 9.]

>>> TI.real    # ou mieux encore, la partie réelle :
array([ 1., 2., 3., 4., 5., 6., 7., 8., 9.])
```

Applications de la transformée discrète

- La transformée de Fourier discrète admet un large champ d'application :
  1. en traitement numérique :
     - 1.1 Déterminer la bande passante fréquentielle d'un signal, Analyse spectrale,
     - 1.2 Filtrage (atténuation ou accentuation de certaines fréquences du signal), traitement sonore, traitement d'image.
     - 1.3 Mesures Doppler, radar, transmission numérique (DVB),
     - 1.4 Compression de données (son), etc...
  2. En informatique et Mathématiques :
     - 2.1 Résolution d'EDP,
     - 2.2 Multiplication de grands nombres, etc...

Ce n'est qu'une version discrète de la transformée (continue) de Fourier qui permet de passer de la représentation temporelle d'un signal à sa représentation fréquentielle, obtenue par méthode des rectangles :

$$\begin{array}{ccc} L_1(\mathbb{R}) & \longrightarrow & L_1(\mathbb{R}) \\ f & \longrightarrow & TF(f) : \omega \mapsto \displaystyle\int_{-\infty}^{+\infty} f(x) \cdot \exp(-i2\pi\omega x)\, dx \end{array}$$

Si elle est très employée c'est parce qu'il existe un algorithme rapide de calcul.

Transformée de Fourier rapide

• On peut calculer la transformée de Fourier discrète en temps $O(N\log(N))$ plutôt que $O(N^2)$ en appliquant la méthode de diviser pour régner.

• Nous voyons l'algorithme le plus célèbre dû à Cooley et Tuckey. Appliquons-le pour N une puissance de 2 : $N = 2^p$.

• Pour décomposer le problème en deux sous-problèmes on scinde le calcul en deux parties : termes de rangs pairs et termes de rangs impairs :

Pour tout $n \in [[0, N-1]]$:

$$\widehat{x_n} = \sum_{k=0}^{N-1} x_k \exp\left(\frac{-2in\pi}{N}k\right) \qquad k = 2m \text{ ou } k = 2m+1$$

$$= \underbrace{\sum_{m=0}^{N/2-1} x_{2m} \exp\left(\frac{-2in\pi}{N} \times 2m\right)}_{\text{termes pairs}} + \underbrace{\sum_{m=0}^{N/2-1} x_{2m+1} \exp\left(\frac{-2in\pi}{N} \times (2m+1)\right)}_{\text{termes impairs}}$$

$$= \sum_{m=0}^{N/2-1} x_{2m} \exp\left(\frac{-2in\pi}{N/2} \times m\right) + e^{\frac{-in\pi}{N/2}} \times \sum_{m=0}^{N/2-1} x_{2m+1} \exp\left(\frac{-2in\pi}{N/2} \times m\right)$$

Transformée de Fourier rapide

• Lorsque $n \in [[0, N/2-1]]$, la transformée de Fourier s'obtient en prenant la bonne combinaison linéaire de 2 transformée de Fourier sur des données de tailles moitié moindres :

$$\widehat{x_n} = \underbrace{\sum_{m=0}^{N/2-1} x_{2m} \exp\left(\frac{-2in\pi}{N/2} \times m\right)}_{\Longrightarrow \text{ TF de taille } N/2} + \exp\left(\frac{-in\pi}{N/2}\right) \times \underbrace{\sum_{m=0}^{N/2-1} x_{2m+1} \exp\left(\frac{-2in\pi}{N/2} \times m\right)}_{\Longrightarrow \text{ TF de taille } N/2}$$

• Lorsque $n \in [[N/2, N-1]]$ on applique la 2π-périodicité de l'exponentielle complexe $\theta \mapsto \exp(i\theta)$ pour s'y ramener :

• $\exp\left(\frac{-2in\pi}{N/2} \times m\right) = \exp\left(\frac{-2i(n'+N/2)\pi}{N/2} \times m\right) = \exp\left(\frac{-2in'\pi}{N/2} \times m\right)$

• $\exp\left(\frac{-in\pi}{N/2}\right) = \exp\left(\frac{-i(n'+N/2)\pi}{N/2}\right) = -\exp\left(\frac{-in'\pi}{N/2}\right)$ avec $n' \in [[0, N/2-1]]$

$$\widehat{x_{n'+N/2}} = \underbrace{\sum_{m=0}^{N/2-1} x_{2m} \exp\left(\frac{-2in'\pi}{N/2} \times m\right)}_{\Longrightarrow \text{ TF de taille } N/2} - \exp\left(\frac{-2in'\pi}{N}\right) \times \underbrace{\sum_{m=0}^{N/2-1} x_{2m+1} \exp\left(\frac{-2in'\pi}{N/2} \times m\right)}_{\Longrightarrow \text{ TF de taille } N/2}$$

Transformée de Fourier rapide

```
FFT(X):
    Si longueur(X) = 1:
        Retourner X
    # Diviser pour régner :
    Xpairs = termes de rangs pairs de X
    Ximpairs = Termes de rangs impairs de X
    # Appels récursifs sur les deux sous-suites
    Ypairs = FFT(Xpairs)
    Yimpairs = FFT(Ximpairs)
    N1 = N/2
    # Reconstruction de FFT(X) :
    # Y tableau vide de N éléments
    w = exp(-2iπ/N)
    W = 1
    Pour n variant de 0 à N1-1:
        Y[n]    ← Ypair[n] + W × Yimpair[n]
        Y[n+N1] ← Ypair[n] - W × Yimpair[n]
        W ← W × w
    Renvoyer Y
```

Transformée de Fourier rapide

```
from numpy import exp, pi
def fft(X):
    N = len(X)
    if N == 1:
        return X
    elif N%2 == 1:
        return False # Un nombre pair de termes est attendu
    N1 = N//2
    Pairs, Impairs = X[0::2], X[1::2] # rangs pairs et impairs
    Yp, Yi = fft(Pairs), fft(Impairs) # Appels récursifs
    Y = [0] *N
    w = exp(-2j*pi/N)
    W = 1
    for n in range(N1):
        Y[n] = Yp[n] + Yi[n] * W
        Y[n+N1] = Yp[n] - Yi[n] * W
        W *= w
    return Y
```

Transformée de Fourier rapide

Complexité :

```
def fft(X):
    N = len(X)
    if N == 1:
        return X
    elif N%2 == 1:
        return False # Un nombre pair de termes est attendu
    N1 = N//2
    Pairs, Impairs = X[0::2], X[1::2] # rangs pairs et impairs
    Yp, Yi = fft(Pairs), fft(Impairs) # Appels récursifs
    Y = [0] *N
    w = exp(-2j*pi/N)
    W = 1
    for n in range(N1):
        Y[n] = Yp[n] + Yi[n] * W
        Y[n+N1] = Yp[n] - Yi[n] * W
        W *= w
    return Y
```

$$C(N) = 2C(N/2) + \Theta(N)$$

$$\Longrightarrow C(N) = \Theta\left(\sum_{k=0}^{\log_2(N)} 2^k \times \frac{N}{2^k}\right) \Longrightarrow \boxed{\text{complexité } \Theta(N\log(N))}.$$

Transformée de Fourier rapide inverse

- Pour la transformée de Fourier inverse :

$$x_n = \frac{1}{N}\sum_{k=0}^{N-1} \widehat{x_k}\left(\omega_n\right)^k = \frac{1}{N}\sum_{k=0}^{N-1} \widehat{x_k} e^{\frac{2ink\pi}{N}}$$

Il suffit de :

1. Prendre le conjugué complexe (terme à terme) de l'opérande `X`.
2. Lui appliquer FFT.
3. Retourner le conjugué complexe (terme à terme) du résultat divisé par `N=len(X)`.

```
def ifft(X):
    X1 = [x.conjugate() for x in X]
    Y1 = fft(X1)
    N = len(X)
    return [y.conjugate()/N for y in Y1]
```

La "conversion" se faisant en temps linéaire, l'algorithme est encore de complexité $O(N\log(N))$.

Application : Produit de polynômes

- Pour effectuer le produit de deux polynômes :

$$P(X) = a_0 + a_1 X + \cdots + a_n X^n \qquad Q(X) = b_0 + b_1 X + \cdots + b_m X^m$$

Le polynôme $R = PQ$ a pour degré $n + m$:

$$R(X) = c_0 + c_1 X + \cdots c_{n+m} X^{n+m} \qquad \text{avec } c_r = \sum_{p+q=r} a_p b_q$$

En supposant les polynômes donnés par les tableaux `P,Q` de leurs coefficients :

```
def ProduitPolynomes(P,Q):
    R = [0] * (len(P)+len(Q))
    for p in range(len(P)):
        for q in range(len(Q)):
            R[p+q] += P[p]*Q[q]
    return R
```

- De complexité $\Theta(\deg(P) \times \deg(Q))$.

En notant $N = \max(\deg(P), \deg(Q))$: complexité quadratique $O(N^2)$.

Application : Produit de polynômes

- Peut-on faire mieux ? Réponse oui !
- Idée : en notant $p = \deg(P)$, $q = \deg(Q)$.

Le polynome $R = PQ$ a degré $n = p + q$,
il est uniquement déterminé par ses valeurs prises en $n + 1$ points distincts :

En choisissant arbitrairement : $w_0, w_1, \ldots, w_n$: on calcule

$$P(w_0), P(w_1), \ldots, P(w_n)$$
$$Q(w_0), Q(w_1), \ldots Q(w_n)$$

En multipliant terme à terme on en déduit :

$$R(w_0) = P(w_0)Q(w_0), \quad R(w_1) = P(w_1)Q(w_1), \quad \ldots, \quad R(w_n) = P(w_n)Q(w_n)$$

Le polynôme R de degré (au plus) n est uniquement déterminé par ces valeurs.

Application : Produit de polynômes

Posons $N = n + 1$.

1. Prendre pour points les racines N-ièmes de l'unité.
2. Les suites des valeurs prises par P et Q s'obtiennent par TFD des suites de leurs coefficients (complétés avec des 0) : (a_k) de P et (b_k) de Q.
3. Le produit terme à terme donne la suite de longueur N des valeurs prises par $R = PQ$ en ces N points.
4. Sa transformée de Fourier inverse retourne les coefficients de R.

En prenant pour N la plus petite puissance de 2 supérieure à $\deg(P) + \deg(Q) + 1$ (remarquer que $N < 2 \times (\deg(P) + \deg(Q) + 1)$) et en appliquant l'algorithme de transformée de fourier rapide, on obtient une complexité en :

$$O(N \log(N))$$

au lieu de $O(N^2)$.

• En fait on peut obtenir une FFT en $O(N \log(N))$ pour N quelconque (admis). il suffit donc de prendre $N = \deg(P) + \deg(Q) + 1$.

• Utilisé pour multiplier de très grands nombres (vus comme les valeurs prises en 10 par des polynômes avec pour coefficients leurs chiffres).