MT12 - TP4 : Transformation de Fourier discrète

Dans ce TP l'objectif principal est de manipuler la transformée de Fourier discrète.

---

1 Implémentation de la matrice de transformée de Fourier discrète (TFD ou DFT)

Pour un vecteur $\boldsymbol{y} = (y_0, ..., y_{N-1})^\top \in \mathbb{R}^N$, on rappelle la définition de la transformée de Fourier discrète (TFD ou DFT) d'ordre N ($N \geq 2$) :

$$Y_n = \frac{1}{N} \sum_{k=0}^{N-1} y_k \, \omega_N^{-nk}, \qquad n = 0, ..., N-1,$$

où ω_N désigne le nombre complexe suivant :

$$\omega_N = e^{2i\frac{\pi}{N}}.$$

On note $\boldsymbol{Y} = (Y_0, ..., Y_{N-1})^\top$ le vecteur de $\mathbb{R}^N$ et, $\mathcal{F}_N$ l'application linéaire de $\mathbb{R}^N$ dans $\mathbb{R}^N$ telle que $Y_n = (\mathcal{F}_N \boldsymbol{y})_n$ pour $n \in \{0, \ldots, N-1\}$.

1. (a) Justifier l'écriture matricielle suivante de la TFD

$$\boldsymbol{Y} = S_N \, \boldsymbol{y}, \quad \text{où} \quad S_N = \frac{1}{N} \, \overline{\Omega}_N,$$

avec Ω_N matrice carrée de taille N de terme général $(\Omega_N)_{nk} = e^{2i\pi\frac{nk}{N}} = \omega_N^{nk}, \qquad 0 \leq n, k \leq N-1.$

(b) Justifier également que l'inverse de la matrice S_N vérifie

$$(S_N)^{-1} = \Omega_N.$$

Autrement dit, la transformée de Fourier discrète inverse (TFDI) a pour représentation matricielle la matrice Ω_N.

Attention. Pour les questions suivantes on rappelle que dans `Scilab`, les indices de tableaux (et de matrices) commencent par l'indice 1, et non par 0. Pour émuler des tableaux d'indices commençant par 0 (et pour des raisons de lisibilité de code), on pourra définir la fonction suivante :

```
function j=I(i)
    j = i+1;
endfunction
```

2. Implémentation naïve de la TFD.

   (a) En `Scilab`, pour un N fixé et en utilisant le code ci-dessous, écrire une fonction `TFD1` qui permet d'implémenter la matrice S_N.

```
function y=TFD1(N)
    omegaN = exp(...);
    OMEGAN = zeros(N,N);
    for n=...
            for k=...
            OMEGAN(I(n),I(k)) = ...
            end
    end
    y=conj(OMEGAN)/N;
endfunction
```

   La fonction `conj` permet de calculer le conjugué d'un nombre complexe, cette fonction s'applique aussi aux matrices.

   (b) Indiquer pourquoi le code de la fonction `TFD1` n'est pas très performant.

3. Implémentation plus performante de la TFD que la méthode naïve.
   Pour cette question, on considère le vecteur $\boldsymbol{w} \in \mathbb{R}^N$ dont le n-ième composante est définie par

$$w_n = \sqrt{\frac{2\pi}{N}}\, n\, e^{i\pi/4}, \quad n = 0, ..., N-1,$$

   (a) Que vaut le produit

$$\boldsymbol{w}\boldsymbol{w}^\top = \begin{pmatrix} 0 \\ \vdots \\ \sqrt{\frac{2\pi}{N}}\,(N-1)\, e^{i\pi/4} \end{pmatrix} \begin{pmatrix} 0 & \dots & \sqrt{\frac{2\pi}{N}}\,(N-1)\, e^{i\pi/4} \end{pmatrix} \quad ?$$

   On pourra simplement calculer les éléments $(\boldsymbol{w}\boldsymbol{w}^T)_{nk}$ de cette matrice.

   (b) En déduire que $\Omega_N = \exp\big(\boldsymbol{w}\boldsymbol{w}^T\big)$ (en notation vectorielle `Scilab`).

   (c) Définir une fonction `TFD2` qui prend en entrée $N \geq 1$, qui utilise $\Omega_N = \exp\big(\boldsymbol{w}\boldsymbol{w}^T\big)$ et qui renvoie la matrice S_N.

   Attention : en `Scilab`, si $\boldsymbol{w}$ est un vecteur contenant des nombres complexes l'opération `w'` renvoie le vecteur transposé et conjugué de $\boldsymbol{w}$. En revanche, l'opération `w.'` renvoie bien le vecteur transposé de $\boldsymbol{w}$.

4. Comparer les temps d'exécution des fonctions `TFD1` et `TFD2` pour $N = 100$, 200, 300 et 400. Pour ce faire on utilisera les fonctions `tic` et `toc` (utiliser l'aide de Scilab).

5. Que se passe-t-il lorsqu'on exécute la fonction `TFD2` pour $N = 50000$?

2 FFT et « débruitage » d'un signal

Dans la pratique pour calculer efficacement la TFD d'un signal on utilise l'algorithme de FFT. Dans la suite on va considérer cette manière très performante de calculer la TFD. L'algorithme de FFT est implémenté sur `Scilab` sous le nom `fft`.

Signal simple

Débutons par un exemple de signal simple, défini par la fonction T-périodique (signal) f avec $T = 1$,

$$f(x) = \sin(2\pi f_0 x) + \sin(2\pi f_1 x), \quad \forall x \in [0, T],$$

et pour lequel les fréquences $f_0 = 50$ et $f_1 = 120$ sont connues.

6. En utilisant le vecteur t donné ci-dessous représenter graphiquement le signal f pour $N = 1000$.

```
t=linspace(0,T,N)
```

7. Calcul de la TFD par la commande `fft`.
   (a) En utilisant la fonction `fft` calculer `fhat` la TFD de `f`.
   (b) En `Scilab` la TFD d'un signal est le vecteur $\boldsymbol{Y}$ constitué des nombres complexes Y_n, $n = 0 : N-1$. Pour représenter l'importance des fréquences du spectre du signal on trace soit le module au carré des composantes de `fhat` (i.e. $|Y_n|^2$) soit le module des composantes de `fhat` (i.e. $|Y_n|$) en fonction de la fréquence (i.e. n/T). La première méthode donne la densité spectrale de puissance (ou spectre de puissance) et l'autre le spectre d'amplitude. Par ailleurs, par symétrie, voir la Section 3.2 du chapitre 3 du cours, on représente généralement le spectre de puissance ou d'amplitude uniquement pour la première moitié des fréquences. En complétant le code ci-dessous tracer le spectre d'amplitude de `fhat`.

```
freq=(0:N) ;//abscisse des frequences
L=1:floor(N/2) ;//on conserve seulement la moitie des frequences

scf()
plot(freq(L),abs(...)) //abs permet de calculer le module d'un nombre complexe
```

Signal bruité

8. On va à présent bruiter le signal f. Pour ce faire utiliser la commande suivante et tracer la représentation graphique de `funclean`.

```
funclean = f + 2.5*rand(t,"normal")
```

9. Dans la suite on veut débruiter `funclean` afin de retrouver le signal initial `f`. Ici nous connaissons la définition explicite de la fonction f mais supposons que f ne soit pas connue explicitement. Calculer dans un premier temps `funcleanhat` la DFT de `funclean`. En utilisant la fonction `subplot` donner la représentation graphique du spectre d'amplitude de `fhat` et celui de `funcleanhat`.

10. Afin de débruiter le signal `funclean` on va brutalement mettre à 0 les fréquences dont le module est inférieur à $r > 0$ un paramètre. En vous aidant du spectre d'amplitude de `funcleanhat` proposer une valeur de r.

11. On note $\hat{f}^{\text{unclean}} = (\hat{f}_n^{\text{unclean}})$ les composantes de la TFD de `funclean`, i.e., les composantes du vecteur `funcleanhat` (calculer via la FFT à la question 9). L'opération décrite dans la question précédente revient à appliquer un filtre au vecteur $\hat{f}^{\text{unclean}}$. Notons $H = (H_n)$ les composantes de ce filtre. Dans le domaine fréquentiel la définition de $H = (H_n)$ est la suivante

$$H_n = \begin{cases} 1 & \text{si } |\hat{f}_n^{\text{unclean}}| \geq r, \\ 0 & \text{sinon.} \end{cases}$$

où r désigne le paramètre donné dans la question précédente. On considère alors le vecteur $\hat{f}^{\text{clean}} = (\hat{f}_n^{\text{clean}})$ avec

$$\hat{f}^{\text{clean}} = \hat{f}^{\text{unclean}} \otimes H,$$

où $\otimes$ désigne le produit de Hadamard (l'opération `.*` en `Scilab`). Implémentez en `Scilab` le calcul permettant d'obtenir le vecteur $\hat{f}^{\text{clean}}$ (noté `fcleanhat` dans votre code).

Indication. Pour le calcul de `fcleanhat` et en particulier la définition du filtre vous pouvez vous aider de la commande suivante :

```
H=abs(...)>r;
```

12. Enfin en utilisant l'inverse de la TFD revenez dans le domaine temporel pour calculer `fclean`, on utilisera le script suivant :

```
fclean = ifft(fcleanhat);
```

Dans un même graphique représentez `f`, `fclean` et `funclean`. Est-ce que le filtrage précédent a permis de débruiter `funclean` ? Si ce n'est pas le cas, vous pouvez considérer une valeur de r différente.

Remarque. Il est important de comprendre que si la définition du filtre H est naturelle (ou en tout cas intuitive) dans le domaine fréquentiel, sa définition dans le domaine temporel est beaucoup moins aisée. Par exemple, rappelez-vous qu'un filtre passe-bas, passe-haut ou passe bande est souvent (toujours) défini dans le domaine fréquentiel.