MT12 - TP4 : Transformation de Fourier discrète

Avril 2023

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

Dans ce TP je conseille de définir un fichier .sce pour chaque section (par exemple TP4_exo1.sce etc).

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

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

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

On note $\boldsymbol{Y} = (Y_0, ..., Y_{N-1})^T$ et $\mathcal{F}_N$ l'application linéaire de $\mathbb{R}^N$ dans $\mathbb{R}^N$ avec $Y_n = (\mathcal{F}_N \boldsymbol{y})_n$ pour $n \in \{0, \ldots, N-1\}$.

1. Justifier que la TFD s'écrit sous forme matricielle

   $$\boldsymbol{Y} = S_N \, \boldsymbol{y},$$

   où

   $$S_N = \frac{1}{N} \, \overline{\Omega}_N,$$

   et Ω_N est la matrice carrée $N \times N$ de terme général $(\Omega_N)_{nk} = e^{2i\pi\frac{nk}{N}} = \omega_N^{nk}$ pour $0 \leq n, k \leq N-1$. Justifier également que l'inverse de la matrice S_N est

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

   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) En `Scilab`, pour un N fixé et en utilisant le code ci-dessous écrire une fonction `DFT1` 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.

3. (Implémentation plus performante de la TFD) Le codage de Ω_N ci-dessus par double boucle n'est pas performant. On préfère utiliser un codage vectoriel utilisant des opérations sur des vecteurs pour gagner en performance. Pour ce faire on considère le vecteur $\boldsymbol{w} \in \mathbb{R}^N$ avec

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

Que vaut le produit

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

On peut simplement calculer les éléments $(\boldsymbol{w}\boldsymbol{w}^T)_{nk}$ de cette matrice. En déduire que $\Omega_N = \exp\big(\boldsymbol{w}\boldsymbol{w}^T\big)$ (en notation vectorielle `Scilab`). En conséquence, définir une fonction `TFD2` qui prend en entrée $N \geq 1$ et renvoie la matrice S_N en suivant cette méthode.

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

4. pour $N = 100$, 200, 300 et 400 comparez les temps d'exécution des fonctions `TFD1` et `TFD2`. Pour ce faire on utilisera les fonctions `tic` et `toc` (utiliser l'aide de Scilab). Lancez la fonction `TFD2` pour $N = 50000$, que se passe-t-il ?

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 déjà implémenté sur `Scilab` sous le nom `fft`...

Débutons par un exemple simple, dans la suite on considère la fonction 1-périodique (signal) f avec

$$f(x) = \sin(2\pi f_0 x) + \sin(2\pi f_1 x), \quad \forall x \in [0,1],$$

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

5. Représenter dans un graphique le signal f en vous aidant des commandes suivantes :

```
dt=.001;
t=0:dt:1;
f=...;
```

6. En utilisant la fonction `fft` calculer `fhat` la TFD de `f`. En `Scilab` La TFD d'un signal est un vecteur constitué de nombres complexes. Pour représenter l'importance des fréquences du spectre du signal on trace soit le module au carré des composantes de `fhat` soit le module des composantes de `fhat` en fonction de la fréquence. La première méthode donne la densité spectrale de puissance (ou spectre de puissance) et l'autre le spectre d'amplitude. Par ailleurs par symétrie on représente généralement (dans les deux cas) le spectre de puissance ou d'amplitude pour les fréquences positives. En complétant le code ci-dessous tracer et **commenter** le spectre d'amplitude de `fhat` obtenu.

```
N=length(t);

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

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

7. On va à présent bruiter le signal f. Pour ce faire utiliser la commande suivante

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

et tracer la représentation graphique de `funclean`.

8. 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 `funcleanhat`.
9. 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.
10. 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 8). 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 = H=abs(...)>r;
```

11. 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 prendre un r plus grand.

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 toujours défini dans le domaine fréquentiel.