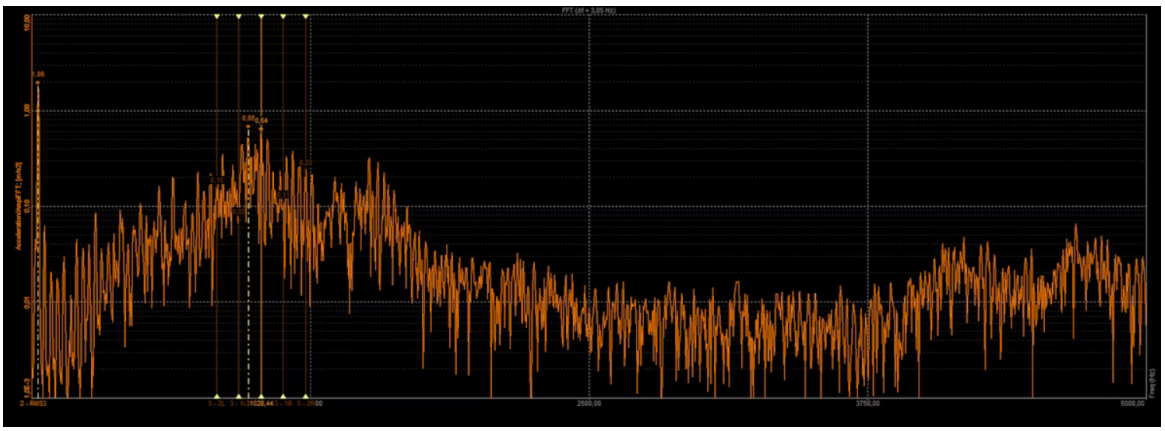
Éléments d'analyse harmonique de l'ingénieur

BE : une courte introduction à la transformée de Fourier discrète



LEFOL Romain, MBOCK Steeve, PERRET-BARDOU Guilhem

LIGNAC Simon, ZABALA Nahia

23 Octobre 2024

Table des matières

I Introduction

Dans le cadre des séances de cours et de travaux dirigés du module Ma312, nous avons abordé en profondeur la notion de transformée de Fourier discrète, un outil fondamental dans le traitement des signaux numériques. Contrairement à l'analyse conventionnelle de Fourier, qui s'applique aux fonctions continues, l'analyse de signaux numériques requiert une étude des valeurs discrètes, puisque la majorité des données traitées par les systèmes informatiques sont échantillonnées à intervalles réguliers. Ainsi, la simple transposition des résultats de l'analyse de Fourier continue ne suffit pas pour comprendre pleinement les signaux discrets. La transition vers un cadre discret est cruciale, non seulement pour modéliser correctement les phénomènes réels, en particulier lorsqu'ils sont numérisés, mais aussi pour rendre les outils mathématiques applicables aux nombreux domaines où les signaux sont numérisés, comme l'audio, l'imagerie et les télécommunications.

L'objectif majeur est de faire le lien entre la théorie mathématique et des applications pratiques, en s'appuyant sur des exemples issus du domaine de l'audio. En nous basant sur ces exemples pratiques, nous démontrerons comment les concepts étudiés en cours et en travaux dirigés peuvent être appliqués pour analyser, transformer et manipuler des signaux numériques avec une grande fiabilité. Cela inclut l'utilisation d'outils mathématiques tels que la transformée de Fourier discrète, qui permet de décomposer un signal en ses composantes fréquentielles, facilitant ainsi son traitement, sa compression ou son amélioration. Par cette approche, nous mettrons en évidence la pertinence des notions théoriques et montrerons qu'elles ne se limitent pas à des concepts abstraits, en illustrant leur implication directe et puissante dans des domaines technologiques comme la synthèse et la reconnaissance audio, la réduction de bruit ou encore le traitement de la parole. Ainsi, cette étude vise à illustrer le lien entre la rigueur mathématique et les besoins pratiques de l'ingénierie du signal.

II La Transformée de Fourier Discrète : Fondements Théoriques

Dans les chapitres précédents, notre attention s'est portée sur des **transformations** appliquées à des fonctions définies sur les ensembles des **nombres réels** ($\mathbb{R}$) ou **complexes** ($\mathbb{C}$). Toutefois, ces objets théoriques ne correspondent pas toujours à la réalité pratique. En effet, dans la plupart des cas, nous travaillons avec des **ensembles finis** de valeurs numériques. Ces ensembles peuvent être vus comme des **vecteurs** dans $\mathbb{R}^n$ ou $\mathbb{C}^n$, où « n » désigne la dimension de ces vecteurs.

Cela soulève une question importante : **est-il possible de développer une théorie discrète qui soit tout aussi riche et pertinente que celle qui concerne les fonctions continues, mais adaptée à des vecteurs de valeurs numériques finies ?** La réponse est affirmative. Non seulement une telle théorie existe, mais elle est également soutenue par des algorithmes très efficaces.

Ces algorithmes, comme la **Transformée de Fourier Discrète (TFD)**, sont cruciaux pour le traitement des signaux numériques dans divers domaines tels que l'audio, l'imagerie et les communications. Ils permettent de convertir des **séries temporelles** de données en une **représentation fréquentielle**, facilitant ainsi **l'analyse et la manipulation des signaux**. Grâce à ces techniques, il est possible de décomposer, filtrer ou compresser des données de manière efficace, tout en conservant les caractéristiques essentielles du signal d'origine.

II.1 Des indices modulo N : Découvertes des groupes cycliques

Nous pouvons considérer le vecteur f comme représentant les valeurs d'une fonction périodique échantillonnée à des intervalles réguliers. Pour ce faire, nous allons définir un ensemble de temps $t = (t_0, t_1, \ldots, t_{N-1})^T$ tel que chaque t_j soit donné par :

$$t_j = jh \quad \text{avec } h = \frac{2\pi}{N} \quad \text{et } j \in \{0, 1, \ldots, N-1\}$$

i.e. :

$$f_j := f(t_j)$$

Cela signifie que t_j correspond à des valeurs allant de 0 à 2π, avec un pas de $\frac{2\pi}{N}$.

Nous définissons maintenant la fonction périodique $f(t)$ comme suit :

$$f(t) := \sum_{j=0}^{N-1} f_j \chi_{[t_j, t_{j+1})}(t)$$

où $\chi_{[t_j,t_{j+1})}(t)$ est la fonction indicatrice, qui vaut 1 lorsque t est dans l'intervalle $[t_j, t_{j+1})$ et 0 sinon. Cela signifie que $f(t)$ prend la valeur f_j pour t situé dans l'intervalle $[t_j, t_{j+1})$.

II.2 Le cadre géométrique discret

La **géométrie discrète** est un domaine des mathématiques qui étudie des objets géométriques à partir d'une approche discrète, en se concentrant sur des structures constituées de points, de segments et de polygones, souvent dans des espaces euclidiens ou non euclidiens. Contrairement à la **géométrie classique**, qui examine des formes continues, la géométrie discrète analyse des configurations finies, comme les réseaux de points, les graphes et les arrangements de solides. Elle joue un rôle essentiel dans des domaines variés, tels que la modélisation en informatique, l'optimisation de réseaux, la conception assistée par ordinateur et l'analyse combinatoire. Elle permet en effet de traiter des problèmes géométriques de manière **algorithmique** et **technique**.

Nous allons montrer que $\mathbb{C}^N$ est un $\mathbb{C}$-espace vectoriel et il admet un **produit scalaire hermitien** défini pour f et g, deux vecteurs de $\mathbb{C}^N$, par :

$$\langle f, g \rangle = \frac{1}{N} \sum_{k=0}^{N-1} f_k \bar{g_k}$$

où $\bar{\bullet}$ note le **conjugué complexe**. Nous noterons la norme associée :

$$\|f\|_2 := \sqrt{\langle f, f \rangle} = \frac{1}{\sqrt{N}} \sqrt{\sum_{k=0}^{N-1} |f_k|^2}$$

Prouvons que $\langle \bullet, \bullet \rangle$ est un **produit scalaire** sur $\mathbb{C}^N$.

Démonstration de l'associativité :

Nous savons que :

$$\langle f, g \rangle = \frac{1}{N} \sum_{k=0}^{N-1} f_k \bar{g_k}$$

Par conséquent, si nous faisons le **produit scalaire** de λf et g, nous obtenons le résultat suivant :

$$\langle \lambda f, g \rangle = \frac{1}{N} \sum_{k=0}^{N-1} \lambda f_k \overline{g_k}$$

$$\Leftrightarrow \langle \lambda f, g \rangle = \lambda \frac{1}{N} \sum_{k=0}^{N-1} f_k \overline{g_k}$$

$$\Leftrightarrow \langle \lambda f, g \rangle = \lambda \langle f, g \rangle$$

Démonstration de la linéarité :

$$\langle f+g,h\rangle = \frac{1}{N}\sum_{k=0}^{N-1}\left(f_k(t)+g_k(t)\right)\overline{h(t)}$$

$$\Leftrightarrow \langle f+g,h\rangle = \frac{1}{N}\sum_{k=0}^{N-1} f_k(t)\overline{h(t)} + \frac{1}{N}\sum_{k=0}^{N-1} g_k(t)\overline{h(t)}$$

$$\Leftrightarrow \langle f+g,h\rangle = \langle f,h\rangle + \langle g,h\rangle$$

Démonstration de la distributivité :

$$\langle f+g,h\rangle = \frac{1}{N}\sum_{k=0}^{N-1}(f_k+g_k)\overline{h_k}$$

$$\Leftrightarrow \langle f+g,h\rangle = \frac{1}{N}\left(\sum_{k=0}^{N-1} f_k\overline{h_k} + \sum_{k=0}^{N-1} g_k\overline{h_k}\right) \tag{1}$$

$$\Leftrightarrow \langle f+g,h\rangle = \frac{1}{N}\sum_{k=0}^{N-1} f_k\overline{h_k} + \frac{1}{N}\sum_{k=0}^{N-1} g_k\overline{h_k}$$

$$\Leftrightarrow \langle f+g,h\rangle = \langle f,h\rangle + \langle g,h\rangle$$

Démonstration défini positif :

$$\langle f,f\rangle = \frac{1}{N}\sum_{k=0}^{N-1} f_k\overline{f_k} \tag{2}$$

Or, nous savons que pour $Z \in \mathbb{C}$:

$$Z\overline{Z} = |Z|^2 \tag{3}$$

Ainsi :

$$\langle f,f\rangle = \frac{1}{N}\sum_{k=0}^{N-1} f_k\overline{f_k} = \frac{1}{N}\sum_{k=0}^{N-1} |f_k|^2 \tag{4}$$

Nous supposons que :

$$\langle f,f\rangle = 0 \Leftrightarrow \frac{1}{N}\sum_{k=0}^{N-1} |f_k|^2 = 0$$
$$\Rightarrow f_k = 0 \tag{5}$$

Par conséquent, $\langle\cdot,\cdot\rangle$ est **défini positif**, donc c'est un **produit redoute hermitien**.

II.3 Premier pas vers la transformée de Fourier discrète

Rappelons quelques notions sur les fonctions de **l'espace de Dirichlet**. Cet espace est constitué de fonctions qui sont absolument intégrables sur un intervalle donné et qui satisfont certaines conditions de régularité. Dans ce qui suit, nous considérons f comme une fonction périodique. Une fonction périodique est une fonction qui se répète à intervalles réguliers, ce qui est essentiel pour **l'analyse par Fourier**.

L'opérateur est défini par :

$$\hat{\bullet} : \mathcal{D}_{2\pi}(\mathbb{R}, \mathbb{C}) \to \ell^2(\mathbb{Z})$$

Ici, $D_{2\pi}(\mathbb{R}, \mathbb{C})$ représente l'espace des fonctions 2π**-périodiques** à valeurs complexes, et $\ell^2(\mathbb{Z})$ désigne l'espace des suites à valeurs complexes qui sont sommables au carré. Pour chaque $k \in \mathbb{Z}$, $c_k(f)$ représente le **coefficient de Fourier** de la fonction f. Ce coefficient est obtenu en prenant l'intégrale de la fonction $f(t)$ multipliée par la fonction exponentielle complexe e^{-ikt}, ce qui permet de décomposer f en ses composants de fréquence.

Voici la formule correspondante :

Coefficient complexe de Fourier

$$c_k(f) = \frac{1}{2\pi} \int_0^{2\pi} f(t) e^{-ikt} \, dt \quad \text{pour tout } k \in \mathbb{Z}. \tag{6}$$

Il est important de noter que cet opérateur est **injectif**. Cela signifie que si tous les coefficients de Fourier $c_k(f)$ sont nuls, alors la fonction f elle-même doit être nulle.

Nous notons également la **série de Fourier** associée à f :

Série de Fourier complexe

$$S(f)(t) = \sum_{k=-\infty}^{+\infty} c_k(f) e^{ikt} \tag{7}$$

Cette série, $S(f)(t)$, représente la reconstruction de la fonction périodique f à partir de ses coefficients de Fourier. La somme s'étend sur tous les entiers k, ce qui signifie que toutes les fréquences contribuent à la forme finale de la fonction.

Dans l'analyse de Fourier, nous travaillons souvent avec des fonctions périodiques. Cependant, lorsqu'on passe à un **cadre discret**, il est crucial de prendre en compte les limitations propres à ce domaine. En particulier, les **sommes infinies** deviennent impraticables, ce qui nous oblige à adopter des **sommes partielles**.

Dans le contexte discret, les données sont souvent représentées sous forme de **séries finies**. Par exemple, nous ne pouvons pas traiter une **série infinie** en raison de contraintes pratiques, comme la mémoire et le temps de calcul. Dans le cas des séries de Fourier, cela signifie que nous ne pouvons pas considérer toutes les fréquences qui composent une fonction.

En effet, lorsque nous cherchons à **approximer une fonction périodique** $f(t)$ dans un cadre discret, il nous faut tronquer la série de Fourier. Cela nous amène à définir une **somme partielle des coefficients de Fourier**, limitant ainsi le nombre de termes utilisés. Si nous avions une somme infinie, cela entraînerait des difficultés dans la mise en œuvre algorithmique et pourrait rendre impossible le traitement numérique des données.

Ainsi, pour un entier $N > 0$ choisi pair, nous définissons la **série de Fourier tronquée à l'ordre** N comme suit :

Série de Fourier tronquée à l'ordre N

$$S_N(f)(t) = \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} c_k(f) e^{ikt} \tag{8}$$

où $c_k(f)$ sont les coefficients de Fourier donnés par :

$$c_k(f) := \frac{1}{2\pi} \int_0^{2\pi} f(t) e^{-ikt}\, dt$$

Cette approximation nous permet de capturer les principales caractéristiques de la fonction périodique tout en respectant les **limitations du cadre discret**.

En limitant notre série à N termes, nous pouvons obtenir une approximation de $f(t)$ qui conserve les composantes essentielles sans nécessiter l'intégration de toutes les **fréquences**. La **série tronquée**, bien que simplifiée, est souvent suffisante pour une bonne représentation de la fonction.

II.4 La transformée de Fourier discrète

Pour rendre cette somme plus compatible avec les langages de programmation comme Python, où les tableaux commencent à l'indice 0, nous allons **ré-indexer** cette somme. Cela signifie que nous allons transformer les indices de telle sorte qu'ils commencent à zéro, sans changer la nature de la somme elle-même.

II.4.1 Étape 1 : Changement de variable

Pour cela, nous posons un **changement de variable**, en définissant un nouvel indice j tel que :

$$j = k + \frac{N}{2}$$

Ainsi, l'indice k, qui variait de $-\frac{N}{2}$ à $\frac{N}{2} - 1$, devient un indice j qui varie de 0 à $N - 1$.

II.4.2 Étape 2 : Substitution dans la somme

Série de Fourier tronquée après changement de variable :

En remplaçant k par $j - \frac{N}{2}$ dans l'expression de la **série tronquée**, nous obtenons :

$$S_N(f)(t) = \sum_{j=0}^{N-1} c_{j-\frac{N}{2}}(f) e^{i(j-\frac{N}{2})t} \tag{9}$$

Cette nouvelle expression a l'avantage que la somme commence à l'indice 0, ce qui est plus instinctif dans les langages de programmation comme **Python**.

Réflexes :

— **Simplification des calculs numériques :** En partant de l'indice zéro, il est plus simple de manipuler la série tronquée dans un tableau numérique.

— **Alignement avec les conventions :** Dans la majorité des langages de programmation (y compris Python), les indices des tableaux commencent à 0. Cette ré-indexation permet, par conséquent, d'écrire la série de Fourier tronquée de manière compatible avec ces conventions.

D'un point de vue pratique, les valeurs de t pour lesquelles nous désirons évaluer notre série de Fourier doivent être **discrètes**. En d'autres termes, nous devons choisir un ensemble d'instants spécifiques où nous effectuons cette évaluation.

Supposition :

Supposons que nous disposions d'un vecteur $\mathbf{t}$, défini comme suit :

$$\mathbf{t} := (t_0, t_1, \ldots, t_{N-1})^T$$

Où chaque t_j est donné par l'expression $t_j = jh$, avec $h = \frac{2\pi}{N}$ et $j \in \{0, 1, \ldots, N-1\}$.

Ce vecteur $\mathbf{t}$ représente donc une **discrétisation** des points à travers lesquels nous allons **échantillonner** nos fonctions périodiques. Ces points sont uniformément répartis sur un intervalle de longueur 2π, ce qui est cohérent avec l'hypothèse de **périodicité** que nous imposons souvent en analyse de Fourier.

Étant donné ce **vecteur de points d'échantillonnage**, les seules valeurs de la fonction f que nous pouvons réellement observer ou connaître sont celles correspondant à ces instants spécifiques. En d'autres termes, pour chaque t_j, la valeur de la fonction f est notée $f(t_j)$, que nous désignerons ici par f_j. Ces valeurs f_j sont donc les valeurs discrètes de la fonction f, obtenues à partir du vecteur de temps $\mathbf{t}$.

Ainsi, nous avons :

$$f(t_j) := f_j$$

Ces valeurs seront ensuite utilisées dans le cadre de **l'analyse de Fourier** pour **approximer** la fonction f à travers sa série de Fourier discrète, qui repose sur ces **points d'échantillonnage**.

II.4.3 Équivalent discret des coefficients c_k

Dans le cadre de **l'analyse des signaux**, il est essentiel d'introduire **l'équivalent discret des coefficients** c_k, définis par l'intégrale suivante :

Équivalent discret des coefficients c_k :

$$c_k = \frac{1}{2\pi} \int_0^{2\pi} f(t) e^{-ikt}\, dt. \tag{10}$$

Cette formule permet de passer d'une **représentation continue** à une **représentation discrète**, facilitant ainsi les manipulations numériques.

II.4.4 Remplacement de la Fonction Continue

Nous allons remplacer la fonction continue $f(t)$ par sa version discrète $f_d(t)$. La définition de cette version discrète est donnée par :

Équivalent discret des coefficients c_k :

$$f_d(t) = \sum_{j=0}^{N-1} f(t_j)\chi_{[t_j,t_{j+1}[}(t), \tag{11}$$

Où $\chi_{[t_j,t_{j+1}[}(t)$ est la **fonction indicatrice**, qui vaut 1 si t appartient à l'intervalle $[t_j, t_{j+1}[$ et 0 sinon. Cette définition nous permet de segmenter la fonction continue en intervalles et d'attribuer à chaque intervalle une valeur correspondant à $f(t_j)$.

II.4.5 Propriété de la Fonction Discrète

Il est important de noter que, pour une telle fonction, si t appartient à l'intervalle $[t_j, t_{j+1}[$, nous avons :

$$f_d(t) = f(t_j) = f_j. \tag{12}$$

Cela signifie que, dans chaque intervalle, la valeur de la fonction discrète est constante et égale à la valeur de la fonction continue à l'extrémité gauche de l'intervalle.

II.4.6 Notation générale pour une fonction G

De manière plus générale, pour toute fonction g, nous utiliserons g_d ou $(g)_d$ pour désigner sa **version discrète**. Cette notation permet de simplifier l'expression et l'identification de la version discrète des fonctions, facilitant ainsi leur traitement dans le cadre de **l'analyse numérique**.

II.4.7 Formulation discrète des coefficients $c_k(f)$

Nous allons dériver ici **l'expression discrète des coefficients** $c_k(f)$ pour $k \in \{0, N-1\}$. Ces coefficients sont définis comme suit :

Expression discrète des coefficients c_k :

$$c_{k,D}(f) := \frac{1}{2\pi}\int_0^{2\pi} \left(t \mapsto f(t)e^{-i\left(k-\frac{N}{2}\right)t}\right)_D dt \tag{13}$$

La version discrétisée de cette formule peut être réarrangée de la manière suivante :

Démonstration de la formule discrétisée des coefficients $c_k(f)$:

$$
\begin{aligned}
c_{k,D}(f) &= \frac{1}{2\pi}\int_0^{2\pi}\left(t \mapsto f(t)e^{-i\left(k-\frac{N}{2}\right)t}\right)_D\, dt \\
&= \frac{1}{2\pi}\int_0^{2\pi}\sum_{j=0}^{N-1} f(t_j)e^{-i\left(k-\frac{N}{2}\right)t_j}\chi_{[t_j,t_{j+1}[}(t)\, dt \\
&= \frac{1}{2\pi}\sum_{j=0}^{N-1} f(t_j)e^{-i\left(k-\frac{N}{2}\right)t_j}\int_{t_j}^{t_{j+1}} dt \\
&= \frac{1}{2\pi}\sum_{j=0}^{N-1} f(t_j)e^{-i\left(k-\frac{N}{2}\right)t_j}h \\
&= \frac{1}{N}\sum_{j=0}^{N-1} f(t_j)e^{-i\left(k-\frac{N}{2}\right)t_j} \qquad (14)
\end{aligned}
$$

Ensuite, nous introduisons la **racine primitive d'ordre** N, notée W_N, qui est définie par :

$$W_N := e^{-ih} = e^{-i\frac{2\pi}{N}} \qquad (15)$$

Cette racine primitive représente une unité complexe sur le cercle unité, et permet de reformuler la version discrète des coefficients $c_{k,D}(f)$. En réécrivant l'expression de $c_{k,D}(f)$ à l'aide de W_N, on obtient :

$$c_{k,D}(f) = \frac{1}{N}\sum_{j=0}^{N-1} f(t_j)W_N^{(k-\frac{N}{2})j}$$

Explication approfondie :

- La **racine primitive** W_N est un concept clé en analyse de Fourier discrète du fait qu'elle exprime les rotations sur le cercle complexe en divisant ce dernier en N parties égales. Chaque terme $W_N^{(k-\frac{N}{2})j}$ dans la somme introduit une dépendance à k et j, ce qui correspond à une transformation complexe des points $f(t_j)$.
- L'exposant $(k-\frac{N}{2})j$ ajuste le **décalage fréquentiel** de manière à centrer les fréquences autour de $\frac{N}{2}$, afin de mieux capturer les propriétés spectrales du signal discrétisé $f(t_j)$.
- Cette reformulation est particulièrement utile dans les applications numériques, notamment dans les algorithmes de **transformée de Fourier rapide (FFT)**. L'utilisation de W_N y simplifie les calculs et optimise les multiplications par des exponentielles complexes, accélérant ainsi la transformation.

II.4.8 Réécriture de la transformée de Fourier discrète

Pour établir un lien entre la **version discrète du théorème de Dirichlet** et sa **version plus répandue**, nous allons définir le vecteur $e_{k-\frac{N}{2}N}$, dont les composantes sont exprimées comme suit :

$$e_{k-\frac{N}{2}N} = W(k-\frac{N}{2})^j N = W^N(k-\frac{N}{2})^j = e^{i(k-\frac{N}{2})t_j} \qquad (16)$$

Dans cette formulation, chaque composante du vecteur est générée par une **fonction exponentielle complexe**, reliant ainsi les différentes versions du **théorème de Dirichlet** à travers des transformations spécifiques de la variable k et des termes associés. Ce changement de perspective nous permet de mieux comprendre les implications et les applications de ce théorème dans le cadre de la théorie des nombres.

Réécriture de la transformée de Fourier discrète

La **transformée discrète de Fourier** peut être formulée de la manière suivante :

$$S_N^D(f)(t) := \sum_{k=0}^{N-1} c_{k,D}(f) e^{i\left(k-\frac{N}{2}\right)t} \tag{17}$$

Dans cette expression, nous considérons une somme qui se s'étend sur N termes, chacun étant le produit d'un coefficient $c_{k,D}(f)$ et d'une fonction exponentielle complexe. Pour l'instant, avant de procéder à la démonstration d'un **résultat similaire au théorème de Dirichlet**, nous allons désigner cette série sous le nom de série de Fourier discrète. Notre objectif actuel est de **démontrer une version « discrète »** du théorème de Dirichlet.

Parallèlement, nous souhaitons établir l'existence d'une **formule d'inversion** qui nous permettra de récupérer la fonction originale à partir de ses **coefficients de Fourier**.

II.5 Théorème de Dirichlet discret

II.5.1 Un pas vers le théorème de Dirichlet discret

En outre, nous souhaitons, en utilisant les notations que nous avons établies précédemment, démontrer la version discrète du théorème de Dirichlet. Ce dernier peut être exprimé sous la forme suivante :

Expression de la version discrète du théorème de Dirichlet :

$$f = \sum_{k=0}^{N-1} D\left(f, e_{k-\frac{N}{2}}\right) e_{k-\frac{N}{2}}. \tag{18}$$

Dans cette expression, f représente un vecteur dans $\mathbb{C}^N$, et le vecteur $e_{k-\frac{N}{2}}$ est défini par les composantes suivantes :

$$\left(e_{k-\frac{N}{2}}\right)_j = W\left(k-\frac{N}{2}\right)\frac{j}{N} = W_N\left(k-\frac{N}{2}\right)\frac{j}{N}, \tag{19}$$

Où $W(z)$ est une fonction spécifique, et $W_N(z)$ représente le **facteur d'échelle normalisé** associé à N. Cette formulation nous permettra de mieux comprendre les implications du **théorème de Dirichlet dans un cadre discret** et d'explorer ses applications dans divers domaines de **l'analyse numérique** et du **traitement du signal**.

II.5.2 Un petit lemme : Dernier pas vers le théorème de Dirichlet discret

Après avoir démontré que la **famille des exponentielles complexes** constitue une **base hilbertienne**, nous allons établir un résultat similaire pour la famille des vecteurs de la forme $e_{k-\frac{N}{2}N}$.

Ainsi, démontrons le lemme suivant :

Un petit lemme :

Considérons deux entiers l et j tels que $0 \leq l, j \leq N-1$. Avec les notations établies précédemment, nous avons :

$$\left\langle e_{j-\frac{N}{2}}, e_{l-\frac{N}{2}} \right\rangle = \delta_{jl} \tag{20}$$

Où δ_{jl} représente le **symbole de Kronecker**.

Démonstration du petit lemme :

$$\begin{aligned}\left\langle e_N^{k-\frac{N}{2}}, e_N^{j-\frac{N}{2}} \right\rangle &= \frac{1}{N}\sum_{l=0}^{N-1} e_N^{k-\frac{N}{2}}\overline{e_N^{j-\frac{N}{2}}} \\ &= \frac{1}{N}\sum_{l=0}^{N-1} e^{ih\left(k-\frac{N}{2}\right)l}\cdot e^{-ih\left(j-\frac{N}{2}\right)l} \\ &= \frac{1}{N}\sum_{l=0}^{N-1} e^{i\frac{2\pi}{N}\left(k-\frac{N}{2}\right)l}\cdot e^{-i\frac{2\pi}{N}\left(j-\frac{N}{2}\right)l} \\ &= \frac{1}{N}\sum_{l=0}^{N-1} e^{\frac{i\pi}{N}l(2k-N-2j+N)} \\ &= \frac{1}{N}\sum_{l=0}^{N-1} e^{i\frac{2\pi}{N}l(k-j)}\end{aligned} \tag{21}$$

— Si $k = j$:

$$\left\langle e_N^{k-\frac{N}{2}}, e_N^{j-\frac{N}{2}} \right\rangle = \frac{1}{N}\sum_{l=0}^{N-1} e^0 = \frac{1}{N}\cdot N = 1$$

— Si $k \neq j$:

$$\begin{aligned}\left\langle e_N^{k-\frac{N}{2}}, e_N^{j-\frac{N}{2}} \right\rangle &= \frac{1}{N}\sum_{l=0}^{N-1}\left(e^{i\frac{2\pi}{N}(k-j)}\right)^l \\ &= \frac{1}{N}\cdot\frac{1-e^{i2\pi(k-j)}}{1-e^{i\frac{2\pi}{N}(k-j)}}\end{aligned} \tag{22}$$

Soit :

$$1 - e^{i2\pi(k-j)} = \cos(2\pi(k-j)) + i\sin(2\pi(k-j)) \tag{23}$$

Or, comme $k - j \in \mathbb{Z}$, on a $\cos(2\pi(k-j)) = 1$ et $\sin(2\pi(k-j)) = 0$. Donc :

$$\left\langle e_N^{k-\frac{N}{2}}, e_N^{j-\frac{N}{2}} \right\rangle = \frac{1}{N}\cdot\frac{1-1}{1-e^{\frac{2i\pi}{N}(k-j)}} = 0. \tag{24}$$

II.5.3 Démonstration du théorème de Dirichlet discret

Pour donner suite aux divers calculs, démontrons ainsi le théorème de Dirichlet discret :

$$
\begin{aligned}
S_N^D(f)(t) &= \sum_{k=0}^{N-1} c_{M,D}(f) e^{i\left(k-\frac{N}{2}\right)t} f \\
&= \sum_{k=0}^{N-1} \frac{1}{N} \sum_{j=0}^{N-1} f(t_j) W_N^{\left(k-\frac{N}{2}\right)j} e^{i\left(k-\frac{N}{2}\right)t_j} \\
&= \sum_{k=0}^{N-1} \frac{1}{N} \sum_{j=0}^{N-1} f(t_j) e^{-i\frac{2\pi}{N}\left(k-\frac{N}{2}\right)j} e^{i\left(k-\frac{N}{2}\right)\frac{2\pi}{N}j} \\
&= \sum_{k=0}^{N-1} \frac{1}{N} \sum_{j=0}^{N-1} f(t_j) \overline{e_N^{k-\frac{N}{2}}} e_N^{k-\frac{N}{2}} \\
&= \sum_{k=0}^{N-1} \left\langle f, e_N^{k-\frac{N}{2}} \right\rangle e_N^{k-\frac{N}{2}} \\
&= f(t_j)
\end{aligned}
\tag{25}
$$

Il est désormais possible de formuler la définition suivante de manière rigoureuse et adéquate, en s'appuyant sur les bases établies précédemment.

Définition : Transformée de Fourier discrète et inverse

La ***transformée de Fourier discrète*** (DFT, pour *Discrete Fourier Transform*) est une fonction allant de $\mathbb{C}^N$ dans $\mathbb{C}^N$:

$$\hat{\bullet}_D : (f_j := f(t_j))_{j\in\{0,\dots,N-1\}} \longmapsto (c_{k,\mathcal{D}}(f))_{k\in\{0,\dots,N-1\}}$$

De même, la ***transformée de Fourier inverse*** (IDFT, pour *Inverse Discrete Fourier Transform*) est une application de $\mathbb{C}^N$ dans $\mathbb{C}^N$:

$$(\hat{\bullet}_D)^{-1} : (c_{k,\mathcal{D}}(f))_{k\in\{0,\dots,N-1\}} \longmapsto (f_j = f(t_j))_{j\in\{0,\dots,N-1\}}$$

III Un algorithme initial de calcul

La **transformée de Fourier discrète** peut être représentée sous **forme matricielle**. En désignant :

$$\mathbf{f} := (f_0, f_1, \dots, f_{N-1})^T$$

la séquence de données, on a, pour un $k \in \{0, \dots, N-1\}$, l'expression de $c_{k,\mathcal{D}}(f)$ donnée par :

$$c_{k,\mathcal{D}}(f) = \frac{1}{N} \sum_{j=0}^{N-1} f(t_j) W_N^{(k-\frac{N}{2})j}$$

ce qui s'écrit également sous forme vectorielle :

$$c_{k,\mathcal{D}}(f) = \frac{1}{N} \begin{pmatrix} w_N^{(k-\frac{N}{2})0} \\ w_N^{(k-\frac{N}{2})1} \\ \vdots \\ w_N^{(k-\frac{N}{2})(N-1)} \end{pmatrix}^T \mathbf{f}$$

En notant $\mathbf{c} := (c_{0,\mathcal{D}}, c_{1,\mathcal{D}}, \dots, c_{N-1,\mathcal{D}})^T$, et en définissant pour tout $(k, j) \in \{0, \dots, N-1\}^2$:

$$A_{kj} := w_N^{(k-\frac{N}{2})j} \tag{26}$$

la **transformée de Fourier discrète** peut être exprimée **matriciellement** sous la forme suivante :

$$\frac{1}{N}A\mathbf{f} = \mathbf{c} \tag{27}$$

De manière similaire, en s'appuyant sur le théorème discret de Dirichlet, on sait que la somme de Fourier discrète est égale à la fonction elle-même évaluée en un point donné. Plus précisément, on a :

$$S_N^{\mathcal{D}}(f)(t_j) = f(t_j)$$

ce qui équivaut à l'expression suivante :

$$\sum_{k=0}^{N-1} c_{k,\mathcal{D}}(f) e^{i(k-\frac{N}{2})t_j} = f(t_j)$$

Cette équation montre que la série de Fourier discrète reconstruit exactement les valeurs de la fonction à partir de ses coefficients $c_{k,\mathcal{D}}(f)$. En réécrivant cette expression sous une forme vectorielle, nous obtenons :

$$\left(W_N^{\frac{N}{2}j} \quad W_N^{-(1-\frac{N}{2})j} \quad \dots \quad W_N^{-(N-1-\frac{N}{2})j}\right)^T \mathbf{c} = f(t_j)$$

Ainsi, en regroupant ces équations, on peut déduire que la transformée de Fourier inverse est donnée par une relation matricielle. Si on définit A_{jk}^{-1} comme suit :

$$A_{jk}^{-1} = W_N^{-(k-\frac{N}{2})j} \tag{28}$$

Alors, la transformée inverse s'écrit de manière compacte sous la forme matricielle :

$$A^{-1}\mathbf{c} = \mathbf{f} \tag{29}$$

Autrement dit, en inversant la matrice A, nous retrouvons la fonction originale $\mathbf{f}$ à partir de ses coefficients de Fourier $\mathbf{c}$.

Propriété : Linéarité de la transformée de Fourier discrète

La **transformée de Fourier discrète (DFT)** et sa transformée inverse (IDFT) sont des opérations linéaires. En d'autres termes, la DFT d'une combinaison linéaire de signaux est équivalente à la combinaison linéaire des DFT de ces signaux. Cette propriété est fondamentale, puisqu'elle permet de simplifier de nombreux calculs et analyses dans le domaine du traitement du signal.

La **formule matricielle** précédente permet de définir un premier algorithme simple de calcul de la DFT. L'idée principale est de construire une matrice $\mathbf{A}$, dont les éléments sont définis en fonction des racines N-ièmes de l'unité. Cette matrice est ensuite utilisée pour transformer le signal en effectuant une **multiplication matricielle**.

Les étapes de l'algorithme sont les suivantes :

1. Construire la matrice $\mathbf{A}$, où chaque coefficient est défini comme suit :

$$\mathbf{A}_{kj} = w_N^{(k-\frac{N}{2})j}, \quad \text{où} \quad w_N = e^{-2\pi i/N}$$

   Ici, w_N est la racine N-ième de l'unité.

2. Effectuer la multiplication matricielle entre $\mathbf{A}$ et le signal f, puis normaliser le résultat en multipliant par $\frac{1}{N}$:

$$\text{DFT}(f) = \frac{1}{N}\mathbf{A}f$$

Cet algorithme, bien que conceptuellement simple, nécessite un nombre d'opérations de l'ordre de $O(N^2)$, ce qui peut devenir inefficace pour des **signaux de grande taille**. Cependant, il reste un bon point de départ pour comprendre la **DFT**.

III.1 Exemple en Python

III.1.1 Programme Python sur la transformée de Fourier discrète

En guise d'exemple, voici un code Python qui implémente cet algorithme pour calculer la Transformée de Fourier Discrète d'un signal donné.

```python
import numpy as np

def DFT(x):

    # Définir la fonction DFT qui prend en entrée un vecteur x (les échantillons dans le domaine temporel).

    N = len(x)  # Calculer la longueur du vecteur x, représentant le nombre de points dans la transformée.

    # Calculer la racine N-ième de l'unité. WN est utilisé pour construire la matrice de transformation.

    WN = np.exp(-1j * 2 * np.pi / N)  # WN = e^(-2πi/N)

    # Initialiser un vecteur complexe de zéros de taille (N, 1).

    vecgen = np.zeros((N, 1)).astype(complex)

    # Remplir vecgen avec les racines N-ièmes de l'unité pour les puissances de WN.

    vecgen = (WN  np.arange(-N/2, N/2)).reshape((N, 1))  # Créer un vecteur de racines.

    # Initialiser une matrice de coefficients W avec des uns de taille (N, 1).

    W = np.ones((N, 1))

    # Construction de la matrice W

    for i in range(1, N):

        # Ajouter une colonne à W pour chaque puissance de vecgen.

        W = np.block([W, vecgen ** i])  # Construction de la matrice W.

    # Normalisation de la matrice

    W = (1 / N) * W  # Normaliser la matrice en divisant chaque élément par N.

    # Retourner le produit matriciel entre W et le vecteur x.

    return W @ x  # Calcul de la DFT, qui donne les coefficients de la transformée de Fourier discrète.
```

Listing 1 – Implémentation de l'iDFT en Python

Ce code Python définit une fonction `DFT(x)` qui prend en entrée un signal discret `x` de longueur N. La **matrice des coefficients de la DFT** est construite et la **multiplication matricielle** est effectuée pour obtenir la DFT du signal. La matrice est ensuite **normalisée** par un facteur $\frac{1}{N}$.

III.1.2 Programme Python de la transformée de Fourier inverse

L'**algorithme matriciel** présenté ici offre une approche simple et directe pour comprendre comment la DFT est calculée. Bien qu'il ne soit pas optimal en termes de complexité, il permet de saisir les principes fondamentaux avant d'aborder des algorithmes plus efficaces, comme la **transformée de Fourier rapide** (FFT).

Ainsi, pour la transformée de Fourier discrète inverse :

```
1
2  def iDFT(x):
3      # Définir la fonction iDFT qui prend un vecteur x (les coefficients de la DFT)
       comme argument.
4
5      N = len(x)  # Calculer la longueur du vecteur x, representant le nombre de points
        dans la transformée.
6
7      # Calculer la racine N-ième de l'unite. WNbar est utilise pour construire la
       matrice de transformation.
8
9      WNbar = np.exp(1j * 2 * np.pi / N)  # WNbar = e^(2πi/N)
10
11     # Initialiser un vecteur complexe de zéros de taille (N, 1).
12
13     vecgen = np.zeros((N, 1)).astype(complex)
14
15     # Remplir vecgen avec les racines N-ièmes de l'unite pour les puissances de WNbar
       .
16
17     vecgen = (WNbar ** np.arange(-N / 2, N / 2)).reshape((N, 1))  # Creer un vecteur
       de racines.
18
19     # Initialiser un vecteur colonne W avec des uns de taille (N, 1).
20
21     W = np.ones((N, 1))
22
23     # Remplir la matrice W avec les puissances de vecgen.
24
25     for i in range(1, N):
26
27         # Ajouter une colonne a W pour chaque puissance de vecgen.
28
29         W = np.block([W, vecgen ** i])  # Construction de la matrice W.
30
31     # Retourner le produit matriciel entre la transposée de W et le vecteur x.
32
33     return W.T @ x  # Calcul de l'iDFT, qui est la somme ponderée des entrées de x.
```

Listing 2 – Implémentation de l'iDFT en Python

III.1.3 Échantillonnage de la Fonction Sinus

Dans cette section, nous allons examiner l'échantillonnage de la fonction sinus sur l'intervalle $[0, 2\pi]$. Ce processus est essentiel pour comprendre comment les signaux continus peuvent être représentés et analysés en utilisant des échantillons discrets.

```
1  import numpy as np  # Importer la bibliotheque NumPy pour les opérations numériques.
2
3  import time  # Importer le module time pour mesurer le temps d'exécution.
```

```
4
5  def f(t):
6
7      \color{red}{ Definir la fonction f qui prend un argument t.}
8
9      # Cette fonction retourne le sinus de t, redimensionne en un tableau (1, 1).
10
11     return np.sin(t).reshape((-1, 1))  # Reshape pour obtenir un tableau de forme (1,
       1).
12
13 # Définir N comme 2 puissance 4 (soit 16).
14
15 N = 2**4
16
17 # Générer un tableau t avec N points uniformément espacés entre 0 et 2π.
18
19 t = np.linspace(0, 2 * np.pi, N)
20
21 # Démarrer le chronomètre pour mesurer le temps d'exécution de la DFT.
22
23 start = time.time()
24
25 # Calculer la Transformée de Fourier Discrète (DFT) de la fonction f évaluée en t.
26
27 TF = DFT(f(t))
28
29 # Arrêter le chronomètre et stocker le temps écoulé.
30
31 end = time.time()
32
33 # Afficher le temps d'execution.
34
35 print("Version matricielle", end - start)  # Afficher la version matricielle et le
       temps ecoule.
```

```
1
2  import matplotlib.pyplot as plt
3
4  fig, ax = plt.subplots(3, 1, figsize=(18, 10))
5
6  ax[0].plot(t, f(t), 'b.')
7
8  ax[1].bar(np.arange(-int(N / 2), int(N / 2)), (TF.real).reshape(N), 1)
9
10 ax[1].set_title('Amplitude algébrique des cosinus')
11
12 ax[1].set_ylim(-1, 1)
13
14 ax[2].bar(np.arange(-int(N / 2), int(N / 2)), (TF.imag).reshape(N), 1)
15
16 ax[2].set_title('Amplitude algébrique des sinus')
17
18 ax[2].set_ylim(-1, 1)
```

— Premier exemple : Prenons $n = 4$:

Nous retrouvons bien la décomposition du sinus sous la forme suivante :

$$\sin(t) = \frac{e^{it} - e^{-it}}{2i} = i\left(\frac{e^{-it}}{2} - \frac{e^{it}}{2}\right).$$

Puisque le **vecteur est réel**, le spectre issu de cette décomposition est donc **symétrique**.

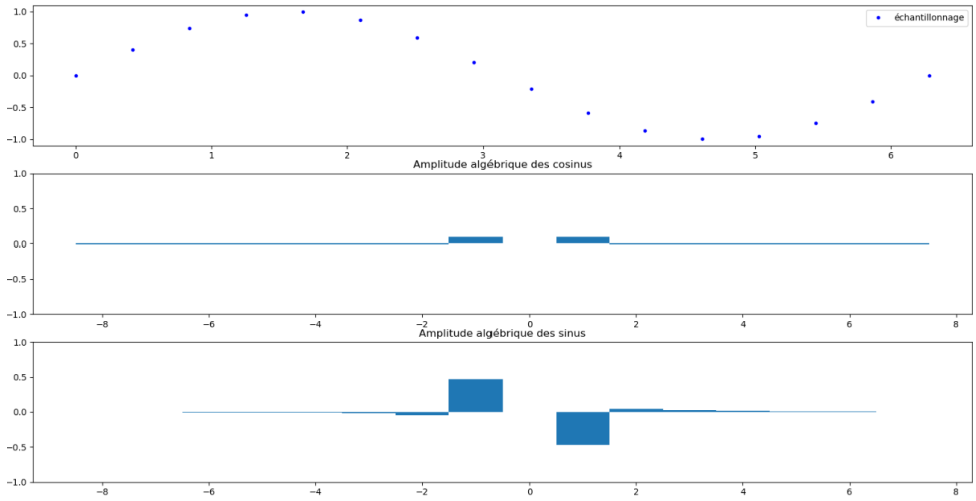


FIGURE 1 – Description de la première image (facultatif)

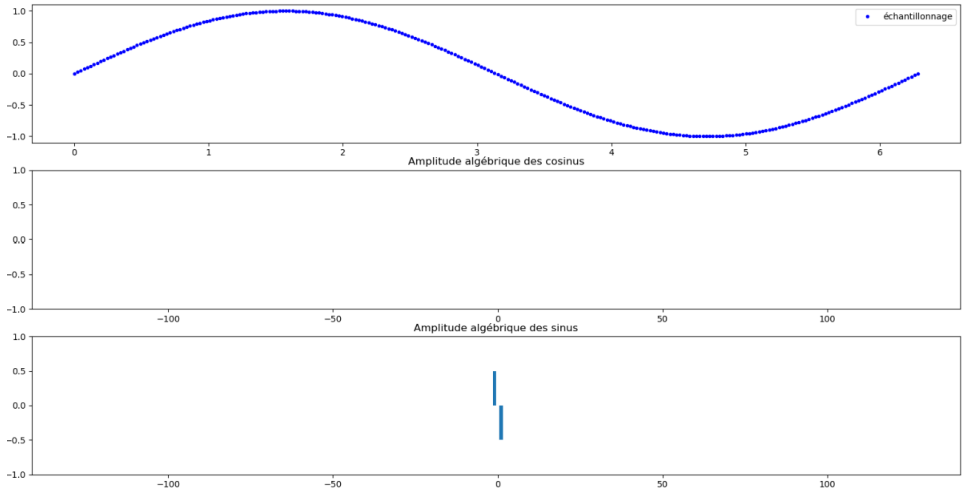


FIGURE 2 – Description de la première image (facultatif)

De plus, la taille de la matrice A nécessaire pour effectuer ces calculs augmente rapidement avec N. Cela pose des problèmes de performance, car pour des valeurs de N élevées, le calcul peut devenir extrêmement lent.

Le spectre détectable est également limité par la taille du vecteur d'entrée. Pour améliorer artificiellement la résolution du spectre, on peut utiliser une méthode appelée ***zero-padding***, qui consiste à ajouter des zéros à la fin du vecteur d'entrée. Cette technique permet d'accroître la résolution apparente du spectre sans modifier le **contenu fréquentiel réel du signal**.

III.2 Petit aparté sur les réseaux de neurones et la méthode du zero-padding

III.2.1 Réseaux de neurones

Un **réseau de neurones** est un modèle informatique inspiré par le fonctionnement du cerveau humain. Utilisé principalement dans les domaines de l'intelligence artificielle (IA) et de l'apprentissage automatique, ce modèle est constitué de couches de ***neurones artificiels*** (ou *unités*) qui se connectent entre eux pour traiter des données complexes. Les réseaux de neurones sont particulièrement utiles pour des tâches telles que la **reconnaissance d'image**, la **classification de texte**, et la **prédiction de séries temporelles**.

III.2.2 Structure d'un réseau de neurones

La structure d'un réseau de neurones se décompose en plusieurs éléments essentiels :

1. **Neurones artificiels** : Chaque neurone est une unité de calcul qui reçoit une ou plusieurs entrées, les pondère, applique une fonction d'activation, puis produit une sortie. Ce processus imite le comportement des neurones biologiques.
2. **Couches** :

   — **Couche d'entrée** : Il s'agit de la première couche du réseau, qui reçoit les données brutes, telles qu'une image, un texte ou des données tabulaires.

   — **Couches cachées** : Ces couches intermédiaires, non directement accessibles en entrée ou en sortie, traitent les données de manière hiérarchique et progressive. Elles permettent d'extraire des caractéristiques de plus en plus abstraites.

   — **Couche de sortie** : Cette couche finale produit le résultat ou la prédiction du réseau, par exemple, la catégorie d'une image ou la probabilité d'un événement.

III.3 Zero-padding dans les réseaux de neurones

La technique du **zero-padding** consiste à ajouter des zéros autour des données d'entrée (souvent les bords d'une image) avant de les traiter dans un réseau de neurones. Ce procédé présente plusieurs avantages :

— **Conservation de la dimension des données** : En ajoutant des zéros, la taille de la sortie après un traitement par convolution reste identique à celle de l'entrée, ce qui simplifie l'architecture du réseau.

— **Amélioration de la résolution du spectre** : En traitement du signal, le zero-padding est utilisé pour accroître artificiellement la résolution spectrale. Cela permet de détecter des caractéristiques plus subtiles dans les données.

— **Réduction des effets de bord** : Les pixels situés aux bords des images sont mieux traités, car le zero-padding leur permet de conserver un voisinage symétrique pour les opérations de convolution.

Ainsi, le zero-padding est une méthode essentielle dans les réseaux de neurones pour améliorer l'analyse et la précision des données traitées, tout en facilitant la construction d'architectures plus robustes.

III.3.1 Structure d'un réseau de neurones

— **Couches cachées** : Ces couches intermédiaires, non visibles dans les données d'entrée ou de sortie, effectuent des calculs sur les données pour détecter des motifs ou des relations complexes. Elles sont responsables de la majeure partie de l'apprentissage du réseau.

— **Couche de sortie** : C'est la couche finale qui produit le résultat, comme une classification ou une prédiction numérique.

— **Connexions et poids** : Chaque connexion entre deux neurones est associée à un poids qui contrôle l'importance de cette connexion. Pendant l'apprentissage, les poids sont ajustés pour minimiser l'erreur entre les prédictions du réseau et les valeurs réelles.

— **Fonctions d'activation** : Ces fonctions introduisent la non-linéarité dans le réseau, permettant au modèle de représenter des relations complexes. Les fonctions d'activation courantes incluent *ReLU*, *Sigmoid* et *Tanh*.

III.3.2 Types de réseaux de neurones

— **Réseaux de neurones artificiels (ANN)** : Ce type de réseau de neurones de base est composé de couches entièrement connectées, où chaque neurone d'une couche est connecté à chaque neurone de la couche suivante.

— **Réseaux de neurones convolutifs (CNN)** : Principalement utilisés pour le traitement d'images, les CNN détectent des motifs, comme des contours ou des textures, grâce à des opérations de convolution.

— **Réseaux de neurones récurrents (RNN)** : Ces réseaux sont conçus pour traiter des séquences de données, telles que des séries temporelles ou du texte. Les connexions récurrentes permettent aux RNN de se souvenir d'informations antérieures dans la séquence, ce qui est utile pour la modélisation de dépendances temporelles.

III.3.3 Applications des réseaux de neurones

Les réseaux de neurones sont utilisés dans de nombreux domaines, notamment :

— **Reconnaissance d'images et d'objets** : Par exemple, pour la reconnaissance faciale.

— **Traitement du langage naturel** : Comme dans la traduction automatique.

— **Prévisions financières** : Pour l'analyse des tendances du marché.

— **Systèmes de recommandation** : Utilisés pour suggérer des produits ou des contenus.

— **Conduite autonome** : Pour permettre aux véhicules de détecter et interpréter leur environnement.

III.3.4 Résumé

En résumé, un réseau de neurones est un système d'apprentissage automatique capable d'apprendre des modèles complexes à partir de grandes quantités de données. Il peut effectuer des tâches variées, telles que la reconnaissance, la classification, ou la prédiction, grâce à une architecture de couches et de connexions ajustées lors de l'entraînement.

III.4 Structure d'un réseau de neurones

— **Couches cachées** : Ces couches intermédiaires, non visibles dans les données d'entrée ou de sortie, effectuent des calculs sur les données pour détecter des motifs ou des relations complexes. Elles sont responsables de la majeure partie de l'apprentissage du réseau.

— **Couche de sortie** : C'est la couche finale qui produit le résultat, comme une classification ou une prédiction numérique.

— **Connexions et poids** : Chaque connexion entre deux neurones est associée à un poids qui contrôle l'importance de cette connexion. Pendant l'apprentissage, les poids sont ajustés pour minimiser l'erreur entre les prédictions du réseau et les valeurs réelles.

— **Fonctions d'activation** : Ces fonctions introduisent la non-linéarité dans le réseau, permettant au modèle de représenter des relations complexes. Les fonctions d'activation courantes incluent $ReLU$, *Sigmoid* et *Tanh*.

III.5 Types de réseaux de neurones

— **Réseaux de neurones artificiels (ANN)** : Ce type de réseau de neurones de base est composé de couches entièrement connectées, où chaque neurone d'une couche est connecté à chaque neurone de la couche suivante.

— **Réseaux de neurones convolutifs (CNN)** : Principalement utilisés pour le traitement d'images, les CNN détectent des motifs, comme des contours ou des textures, grâce à des opérations de convolution.

— **Réseaux de neurones récurrents (RNN)** : Ces réseaux sont conçus pour traiter des séquences de données, telles que des séries temporelles ou du texte. Les connexions récurrentes permettent aux RNN de se souvenir d'informations antérieures dans la séquence, ce qui est utile pour la modélisation de dépendances temporelles.

III.5.1 Applications des réseaux de neurones

Les réseaux de neurones sont utilisés dans de nombreux domaines, notamment :

— **Reconnaissance d'images et d'objets** : Par exemple, pour la reconnaissance faciale.

— **Traitement du langage naturel** : Comme dans la traduction automatique.

— **Prévisions financières** : Pour l'analyse des tendances du marché.

— **Systèmes de recommandation** : Utilisés pour suggérer des produits ou des contenus.

— **Conduite autonome** : Pour permettre aux véhicules de détecter et interpréter leur environnement.

III.5.2 Résumé

En résumé, un réseau de neurones est un système d'apprentissage automatique capable d'apprendre des modèles complexes à partir de grandes quantités de données. Il peut effectuer des tâches variées, telles que la reconnaissance, la classification, ou la prédiction, grâce à une architecture de couches et de connexions ajustées lors de l'entraînement.

III.5.3 Remarque et limite

Le code suivant génère un **signal sinusoïdal de fréquence 392 Hz (note *La*)** et le joue pendant une seconde à l'aide de **la bibliothèque** `sounddevice`.

```

import numpy as np

import sounddevice as sd  # Importation de la bibliothèque sounddevice pour jouer du son

fe = 44100  # Fréquence d'échantillonnage (en Hz)

freq = 392  # Fréquence du signal (en Hz), ici 392 Hz (note La)

duree = 1   # Durée du signal en secondes

t = np.linspace(0, duree, fe * duree)  # Génération du vecteur temps de 0 à 1 seconde avec fe * duree points

amplitude = 1  # Amplitude du signal sinusoïdal

la = amplitude * np.sin(2 * np.pi * t * freq)  # Création du signal sinusoïdal de fréquence 392 Hz

sd.play(la, fe)  # Lecture du signal 'la' avec la fréquence d'échantillonnage 'fe'
```

Listing 3 – Génération et lecture d'un signal sinusoïdal en Python de fréquence 392 Hz (note *La*)

III.5.4 Aparté sur la reconstruction

La matrice A devrait avoir une taille de 44100×44100 pour représenter une seconde de signal. Nous pouvons également tester la « **reconstruction** » de ce signal.

```

signalrecons = iDFT(TF).real

# Reconstruction du signal original à partir de la transformée de Fourier inverse (iDFT), en ne gardant que la partie réelle
```

```
5
6  print(\'\"Ecart au vecteur originel :\", np.linalg.norm(f(t) - iDFT(TF)))
7
8  # Calcul et affichage de la norme (différence) entre le signal original
9
10 # f(t) et le signal reconstruit à partir de la transformée inverse
```

Listing 4 – Reconstruction du signal à partir de la transformée de Fourier inverse

Pour les entiers N précédents, on obtient des erreurs extrêmement faibles, de l'ordre de 10^{-16}.

IV La transformée de Fourier rapide (FFT)

IV.1 Vers une optimisation de l'algorithme pour un gain de rapidité

L'algorithme de **transformation de Fourier rapide**, communément appelé ***Fast Fourier Transform* (FFT)**, est aujourd'hui considéré comme l'un des algorithmes les plus influents dans le domaine du **calcul numérique** et de **l'analyse des signaux**. Il permet de convertir une série de données temporelles en une représentation fréquentielle, rendant possible l'analyse de fréquences présentes dans les signaux de façon bien plus rapide que les méthodes traditionnelles de **transformation de Fourier discrète (DFT)**.

Les premières traces de cet algorithme remontent aux travaux de **Carl Friedrich Gauss** en 1805, qui utilisa une méthode similaire dans le cadre de ses recherches en astronomie pour analyser des séries périodiques. Cependant, **l'algorithme FFT** tel que nous le connaissons aujourd'hui doit sa popularité au mathématicien **James W. Cooley** et au statisticien **John W. Tukey**. En 1965, ils publient un article fondamental intitulé ***"An Algorithm for the Machine Calculation of Complex Fourier Series"***, qui formalise la version moderne de la FFT et en expose la mise en œuvre pratique sur ordinateurs.

Cet article marque un tournant en démontrant comment le calcul de la **transformation de Fourier discrète** peut être optimisé de façon **exponentielle**. Alors que la **méthode brute de DFT** nécessite $\mathcal{O}(N^2)$ opérations pour N échantillons, la **FFT** réduit cette complexité à $\mathcal{O}(N \log N)$, permettant ainsi une application efficace dans des domaines tels que le **traitement du signal**, la **compression d'image**, et la **simulation numérique**. Dans ce document, nous explorerons une **version simplifiée de l'algorithme de transformation de Fourier rapide (FFT)**. Cette version pédagogique vise à illustrer les **principes fondamentaux de l'algorithme** sans entrer dans la complexité des optimisations avancées souvent intégrées dans les implémentations modernes.

Le but est de mieux comprendre le **fonctionnement interne de l'algorithme FFT**, en examinant ses étapes essentielles : la décomposition en sous-problèmes, l'application de la symétrie complexe pour réduire le nombre de calculs et l'assemblage final des résultats partiels pour obtenir la transformation complète. En suivant cette approche, nous pourrons saisir les concepts fondamentaux de la réduction de la complexité de $\mathcal{O}(N^2)$ à $\mathcal{O}(N \log N)$.

Bien que l'**approche simplifiée** permette de comprendre les **bases de la FFT**, les applications pratiques de cette transformation reposent souvent sur des **implémentations performantes et optimisées**, disponibles dans des bibliothèques numériques standard. Ces bibliothèques, telles que `NumPy` en Python ou `FFTW` (Fastest Fourier Transform in the West), fournissent des versions de la FFT qui exploitent au maximum les architectures matérielles modernes pour des calculs rapides et précis.

En pratique, l'utilisation de ces **bibliothèques pré-implémentées** est préférable pour traiter des données volumineuses ou pour répondre aux contraintes de performances dans des applications en temps réel, telles que l'analyse de signaux audio, la compression d'images ou le traitement de données scientifiques. Ces **outils** permettent d'accéder à la **puissance de la FFT** sans avoir besoin de réimplémenter l'algorithme, rendant l'utilisation de cette **méthode accessible** et **efficace** dans un large éventail de domaines.

Lorsque N est une puissance de 2, il est possible de développer un algorithme de calcul extrêmement efficace

pour la Transformée de Fourier Discrète (DFT), appelé Transformée de Fourier Rapide (FFT). Cet algorithme repose sur certaines propriétés spécifiques des racines de l'unité et permet de réduire considérablement le nombre de calculs nécessaires. Détaillons cette méthode en utilisant les propriétés de la racine principale d'ordre N.

IV.1.1 Définition de la racine principale

Nous définissons la racine principale d'ordre N par :

Définition : racine principale d'ordre N

$$W_N := e^{-i\frac{2\pi}{N}} \tag{30}$$

Cette notation est essentielle pour simplifier les calculs des coefficients de la **DFT** dans l'algorithme **FFT**.

IV.1.2 Propriétés pour un calcul rapide

La **méthode FFT** repose sur deux observations mathématiques fondamentales. Considérons la propriété suivante, qui découle directement de l'utilisation de W_N :

Propriété de la méthode FFT :

Pour k et N fixés, un calcul simple montre que :

$$\left(W_N^{\left(k-\frac{N}{2}\right)}\right)^2 = W_N^{2\left(k-\frac{N}{2}\right)} = e^{-i\frac{2\pi}{N}2\left(k-\frac{N}{2}\right)} = e^{-i\frac{2\pi}{N}2k} \cdot e^{i2\pi} \tag{31}$$

En simplifiant, on obtient :

$$\left(W_N^{\left(k-\frac{N}{2}\right)}\right)^2 = W_{\frac{N}{2}}^{k} \tag{32}$$

Cette **simplification** est cruciale dans **l'algorithme FFT**, car elle réduit de manière exponentielle le nombre d'opérations nécessaires en tirant parti de la **symétrie des racines de l'unité**.

En conclusion, en exploitant ces propriétés de W_N, **l'algorithme FFT** est capable de **transformer un signal discret en son spectre fréquentiel** de façon beaucoup plus rapide que la **DFT classique**, ouvrant la voie à des applications en temps réel dans divers domaines de l'ingénierie et du traitement du signal.

Calcul du coefficient discret

Le calcul d'un **coefficient discret** peut s'exprimer comme suit :

$$C_{k,D}(f) = \frac{1}{N} \sum_{j=0}^{N-1} f(t_j) W_N^{\left(k-\frac{N}{2}\right)j} \tag{33}$$

Nous introduisons la **notation** suivante :

$$X := W_N^{\left(k-\frac{N}{2}\right)j} \tag{34}$$

Puis, nous **définissons le polynôme** :

$$p(X) := \frac{1}{N} \sum_{j=0}^{N-1} f_j X^j \tag{35}$$

Pour **simplifier le calcul**, nous séparons les **contributions des puissances paires et impaires**. Nous posons ainsi :

$$p_{\text{pair}}(X) := \frac{1}{N} \left(f_0 + f_2 X + \ldots + f_{N-2} X^{\frac{N}{2}-1} \right) \tag{36}$$

et

$$p_{\text{impair}}(X) := \frac{1}{N} \left(f_1 + f_3 X + \ldots + f_{N-1} X^{\frac{N}{2}-1} \right) \tag{37}$$

Alors, il est immédiat que :

$$p\left(X\right) = p_{\text{pair}}\left(X^2\right) + X \cdot p_{\text{impair}}\left(X^2\right) \tag{38}$$

Ainsi, en regroupant ces deux observations, nous pouvons tirer des conclusions importantes sur le **calcul des transformées de Fourier**. En effet, nous remarquons que :

— Le calcul d'une **transformée de Fourier d'ordre** N peut être efficacement réduit à deux calculs de **transformée de Fourier d'ordre** $\frac{N}{2}$. Cela signifie que chaque étape de la transformation permet de diminuer la taille du problème, facilitant ainsi le **traitement de données volumineuses**. Cette réduction est particulièrement bénéfique dans le cas de **signaux complexes** ou de **grandes images**, où le temps de calcul peut devenir prohibitif sans une approche optimisée.

— De plus, si N est une puissance de 2, cette **méthode de réduction** peut être appliquée de manière **récursive**. Cela entraîne une série de calculs de **transformées de Fourier** de plus en plus petites, jusqu'à atteindre une taille suffisamment réduite pour être traitée efficacement. En conséquence, cette approche conduit à la conception d'un algorithme extrêmement rapide, souvent appelé **l'algorithme FFT (*Fast Fourier Transform*)**, qui est largement utilisé dans divers domaines tels que le traitement du signal, l'analyse d'images et les communications numériques. L'**algorithme FFT** permet de réaliser une **transformée de Fourier en** $O(N \log N)$, contrairement à l'algorithme naïf qui nécessite $O(N^2)$, ce qui constitue un **gain de performance** considérable.

Cette capacité à réduire la complexité du calcul de la **transformée de Fourier** est essentielle du fait qu'elle permet d'obtenir des résultats en un temps significativement réduit, rendant ainsi les applications pratiques de cette transformation plus réalisables dans des contextes de grande envergure. Par ailleurs, **l'efficacité de l'algorithme FFT** a conduit à son intégration dans de nombreux logiciels et bibliothèques de calcul numérique, rendant la **transformée de Fourier** accessible à un large éventail d'applications, allant de l'ingénierie audio à la compression d'images, en passant par l'analyse spectrale. Ainsi, comprendre et appliquer cet algorithme devient une compétence cruciale pour les chercheurs et les ingénieurs travaillant dans des domaines techniques avancés.

IV.1.3 Introduction d'un exemple d'une fonction FFT récursive avec la librairie *Numpy*

```
def FFTrec(f, NN):

    "f en vecteur colonne de taille NN une puissance de 2"

    N = np.shape(f)[0]  # obtention de la taille de f

    WN = np.exp(-1j * 2 * np.pi / N)  # calcul de la racine de l'unité pour la FFT

    FFT = np.zeros((NN, 1)).astype(complex)  # initialisation du vecteur FFT avec des
     zéros

    if N == 2:  # cas de base où N est égal à 2

        FFT = f[0, 0] * np.ones((NN, 1)) + WN ** np.arange(-NN / 2, NN - NN / 2).
    reshape((NN, 1)) * f[1, 0]

        # calcul de FFT pour le cas de base

    else:  # sinon, récursion
        pair = f[::2]  # extrait les termes pairs de f
        impair = f[1::2]  # extrait les termes impairs de f
        p1 = FFTrec(pair, NN)  # appel récursif sur les termes pairs
        p2 = FFTrec(impair, NN)  # appel récursif sur les termes impairs

        for k in range(-int(NN / 2), NN - int(NN / 2)):  # boucle pour combiner les r
    ésultats des sous-appels
            FFT[k + int(NN / 2), 0] = p1[k + int(NN / 2), 0] + (WN ** k) * p2[k + int
    (NN / 2), 0]

    return FFT  # renvoie le vecteur FFT
```

Nous commencerons par vérifier la validité numérique de ce théorème en utilisant des **fichiers audio réels au format `.wav` avec Python**. Pour cela, nous utiliserons la transformée de Fourier rapide fournie par Numpy (ou Scipy). Il est important de noter qu'avec Numpy, la convention choisie pour la **transformée de Fourier inverse** inclut un facteur $\frac{1}{N}$, alors que la transformée directe ne l'inclut pas. Nous adapterons le code pour respecter cette convention. Revenons à l'exemple du La (392 Hz).

```
import numpy as np

import matplotlib.pyplot as plt

# Obtenir la taille du vecteur temporel t

N = np.shape(t)[0]

# Calculer la transformée de Fourier de la note 'la' normalisée par N
TF = np.fft.fft(la) / N

# Calculer les fréquences associées à la transformée de Fourier en Hertz
frequences = np.fft.fftfreq(t.shape[-1]) * fe

# Tracer la partie réelle de la transformée de Fourier en bleu et la partie
    imaginaire en vert

plt.plot(frequences, TF.real, "b-", frequences, TF.imag, "g-")

# Tracer des lignes verticales rouges à la fréquence spécifiée
```

```
plt.vlines(freq, -1.5, 1.5, 'r', alpha=0.5, label=f"{freq}")

# Tracer des lignes verticales magenta à la fréquence négative spécifiée
plt.vlines(-freq, -1.5, 1.5, 'm', alpha=0.5, label=f"{-freq}")

# Afficher la légende
plt.legend()
```

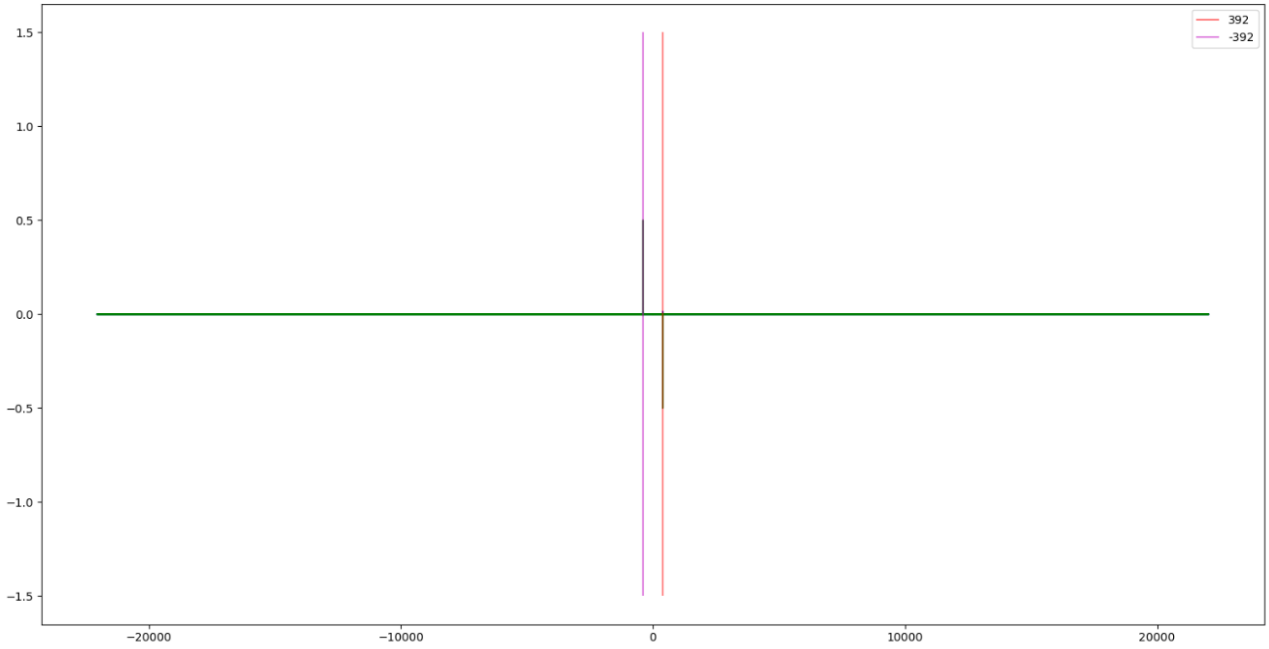


FIGURE 3 – Exemple du *La* (392 Hz)

V Théorème de Parseval pour les signaux discrets : principes et propriétés

V.1 Théoème de Parseval continue : Essentiel à la compréhension de sa version discrète

L'analyse des signaux, essentielle dans des domaines variés tels que les télécommunications, la bio-ingénierie, le traitement d'images et l'acoustique, requiert des outils permettant d'examiner les signaux aussi bien dans le domaine temporel que dans le domaine fréquentiel. Ces deux représentations offrent une vue complémentaire sur les caractéristiques des signaux : **l'analyse temporelle** met en évidence **l'évolution d'un signal au fil du temps**, tandis que **l'analyse fréquentielle** permet **d'examiner la répartition de son contenu en fréquences**, facilitant ainsi la détection de motifs ou d'anomalies.

Le **théorème de Parseval**continue joue un rôle central dans cette analyse. En établissant **l'équivalence énergétique** entre les **domaines temporel** et **fréquentiel**, il permet d'assurer que la totalité de l'énergie d'un signal est préservée, quel que soit le domaine dans lequel on l'étudie. Cette propriété est particulièrement utile pour l'analyse de signaux dans des applications où la conservation d'énergie et l'intégrité des informations sont primordiales, comme dans la transmission de données et les systèmes de compression.

Considérons une fonction $f \in D_{2\pi}(\mathbb{R}, \mathbb{C})$, **périodique** de période 2π et à **valeurs complexes**. Le **théorème de Parseval continue** nous indique que la norme de f dans le **domaine temporel** est identique à celle dans le **domaine fréquentiel**.

Le théorème de Parseval continue

$$\lim_{n\to+\infty} \|S_m(f) - f\|_2 = \lim_{n\to+\infty} \sqrt{\frac{1}{2\pi}\int_0^{2\pi} |S_m(f)(u) - f(u)|^2 \, du} = 0 \tag{39}$$

La **convergence quadratique de la série de Fourier de** f implique que la **série de Fourier** converge vers f en moyenne quadratique, garantissant ainsi que l'énergie du signal reste identique dans les deux domaines.

V.1.1 Égalité de Parseval

De plus, **l'égalité de Parseval** dans l'espace $D_{2\pi}(\mathbb{R}, \mathbb{C})$ est donnée par :

Propriété : L'égalité de Parseval

$$\|f\|_2^2 = \frac{1}{2\pi} \int_0^{2\pi} |f(u)|^2 \, du = \sum_{k \in \mathbb{Z}} |c_k(f)|^2 \tag{40}$$

Cette égalité indique que **l'énergie d'un signal dans le domaine temporel** est égale à la **somme des carrés des amplitudes de ses coefficients de Fourier dans le domaine fréquentiel**. Elle est particulièrement utile dans des applications où il est important de **quantifier l'énergie ou la puissance des composantes fréquentielles,** comme en traitement audio et en analyse des vibrations.

V.2 Démonstration de la version discrète du théorème de Parseval

Nous allons maintenant démontrer la **version discrète du théorème de Parseval**. Soit $A \in \mathbb{M}_N(\mathbb{C})$ une matrice carrée à coefficients complexes. Pour tous les vecteurs f et g appartenant à $\mathbb{C}^N$, nous voulons montrer que :

$$\langle Af, g \rangle = \langle f, A^* g \rangle$$

Où $A^* = A^T$ est la **matrice transconjuguée** de A. Cela signifie que A^* est obtenue en prenant la **matrice conjuguée** de A (où chaque coefficient est remplacé par son conjugué complexe) et en la transposant.

Démonstration de la version discrète du théorème de Parseval

$$\begin{aligned}
\langle f, A^* g \rangle &= \frac{1}{N} \sum_{k=0}^{N-1} f_k (A^* g)_k \\
&= \frac{1}{N} \sum_{k=0}^{N-1} f_k^T A^T g_k \\
&= \frac{1}{N} \sum_{k=0}^{N-1} (A f_k)^T g_k
\end{aligned}$$

Ainsi, nous obtenons l'égalité suivante :

$$\langle f, A^* g \rangle = \langle Af, g \rangle \tag{41}$$

Nous souhaitons démontrer que pour tous vecteurs f et g dans $\mathbb{C}^N$, on a :

$$\langle \widehat{f}, \widehat{g} \rangle = \frac{1}{N} \langle f, g \rangle$$

avec, en particulier,

$$\frac{1}{N} \|f\|_2^2 = \|\widehat{f}\|_2^2$$

Où $\|f\|_2$ représente la **norme euclidienne** de f et $\widehat{f}$ est la **transformée discrète de Fourier de** f.

Démonstration de la version discrète de la transformée de Fourier

Soit A **la matrice de la transformée discrète de Fourier**, de taille $N \times N$, telle que $\widehat{f} = \frac{1}{\sqrt{N}} Af$ et $\widehat{g} = \frac{1}{\sqrt{N}} Ag$.
Le **produit scalaire** entre $\widehat{f}$ et $\widehat{g}$ s'écrit alors comme :

$$\langle \widehat{f}, \widehat{g} \rangle = \left(\frac{1}{\sqrt{N}} Af \right)^{\mathrm{T}} \overline{\left(\frac{1}{\sqrt{N}} Ag \right)} \tag{42}$$

En détaillant cette expression :

$$\langle \widehat{f}, \widehat{g} \rangle = \frac{1}{N} f^{\mathrm{T}} A^{\mathrm{T}} \overline{A} g \tag{43}$$

Sachant que $A^{\mathrm{T}} \overline{A} = N I_N$, où I_N est la matrice identité de taille N, on obtient :

$$\langle \widehat{f}, \widehat{g} \rangle = \frac{1}{N} f^{\mathrm{T}} (N I_N) g = \frac{1}{N} f^{\mathrm{T}} g = \frac{1}{N} \langle f, g \rangle \tag{44}$$

Ce qui prouve le résultat souhaité :

$$\langle \widehat{f}, \widehat{g} \rangle = \frac{1}{N} \langle f, g \rangle \tag{45}$$

En particulier, si $f = g$, alors :

$$\langle \widehat{f}, \widehat{f} \rangle = \frac{1}{N} \langle f, f \rangle \tag{46}$$

Ce qui implique :

$$\| \widehat{f} \|_2^2 = \frac{1}{N} \| f \|_2^2 \tag{47}$$

Démontrant ainsi le **théorème de Parseval** généralisé pour les **vecteurs de dimension N dans $\mathbb{C}^N$.**
En particulier, nous obtenons :

$$\begin{aligned} \| \hat{f} \|_2^2 &= \langle \widehat{f}, \hat{f} \rangle \\ &= \frac{1}{N} < f, f > \\ &= \frac{1}{N} \| f \|_2^2 \end{aligned} \tag{48}$$

VI Produit de Convolution et Application en Numérisation

Le **produit de convolution** est un outil fondamental dans le **traitement des signaux**, permettant de modéliser comment un signal est modifié par un **système linéaire invariant dans le temps (LTI)**. En pratique, nous allons **discrétiser** cette opération pour la rendre applicable dans des contextes numériques, tels que la **simulation en Python**.

VI.1 Forme continue et périodique

Dans le cas d'une **convolution continue**, nous partons de sa définition pour des fonctions 2π**-périodiques**.

Définition : Produit de convolution continue

Soient f et g deux fonctions de l'ensemble $D_{2\pi}(\mathbb{R}, \mathbb{C})$, alors le **produit de convolution continue** est défini par :

$$(f \star g)(t) := \frac{1}{2\pi} \int_0^{2\pi} f(t-u)\, g(u)\, \mathrm{d}u \tag{49}$$

pour tout $t \in \mathbb{R}$.

VI.2 Convolution circulaire discrète

En numérique, notamment en traitement de signal et en calcul matriciel, nous utilisons une **version discrète de la convolution**, appelée **convolution circulaire**. Soient f et g deux vecteurs de $\mathbb{C}^N$

Définition : Produit de convolution circulaire

$$\forall j \in [\![0, N-1]\!], \quad (f \star g)_j := \frac{1}{N} \sum_{i=0}^{N-1} f_{j-i}\, g_i \tag{50}$$

Où l'indice $j - i$ est pris modulo N, assurant ainsi la périodicité de la convolution.

Cette **discrétisation** rend le calcul applicable en contexte numérique et est particulièrement utile pour la **manipulation de vecteurs dans $\mathbb{C}^N$.**

VI.3 Compatibilité avec la transformation de Fourier discrète

Une propriété fondamentale de la **convolution circulaire** est sa compatibilité avec la transformation de Fourier discrète (TFD). En effet, soient f et g deux vecteurs de $\mathbb{C}^N$ et soit $k \in [\![0, N-1]\!]$, alors la **convolution circulaire dans l'espace temporel** se traduit par un produit simple dans **l'espace fréquentiel**.

Définition : Compatibilité avec la transformation de Fourier

$$c_{k,d}(f \star g) = c_{k,d}(f)\, c_{k,d}(g) \tag{51}$$

Où $c_{k,d}$ désigne les **coefficients de Fourier discrets** des vecteurs f et g. Cette propriété est d'une grande importance en **traitement numérique des signaux**, puisqu'elle permet d'optimiser le calcul de convolutions via la **TFD.**

VI.4 Conséquences Pratiques

VI.4.1 Outil essentiel dans le traitement du signal : Le filtrage numérique

Le résultat précédent ouvre la voie au **filtrage numérique**, un outil essentiel dans le traitement du signal :

- Filtrer un signal, tel qu'un enregistrement sonore f, revient souvent à **calculer la convolution** $f \star h$, où h représente le **filtre** désiré, conçu pour modifier certaines caractéristiques du signal.
- Si un tel calcul peut être **complexe à manipuler** dans le **domaine temporel**, il devient beaucoup plus **simple** et **efficace** dans le **domaine fréquentiel** grâce à la **transformation de Fourier.**

VI.4.2 Exemple spécifique d'application avec un filtre coupe-bande

Voyons un exemple concret d'application avec un **filtre coupe-bande**, utilisé pour **supprimer certaines fréquences d'un signal.** Les étapes à suivre sont les suivantes :

- Importer un **fichier sonore au format** `.wav`. Si le fichier est **en stéréo** (présentant plusieurs canaux), il est nécessaire de **séparer ces canaux** pour appliquer le filtre sur chacun d'eux individuellement.
- Calculer la **transformée de Fourier discrète** $\widehat{f_D}$ du signal f à l'aide de la fonction `np.fft.rfft` de Python, qui réalise une transformée de Fourier rapide (FFT) optimisée pour les **signaux réels**.
- Multiplier chaque composante de $\widehat{f_D}$ par un vecteur $\widetilde{h}$ contenant des 0 aux indices des fréquences à supprimer et des 1 pour celles à conserver.

Ensuite, appliquer la **transformée de Fourier inverse** pour revenir dans le **domaine temporel** :

Application de la transformée de Fourier inverse

$$\mathcal{F}_{\boldsymbol{D}}^{-1}\left(\widehat{f_D} \times \widetilde{h}\right) \tag{52}$$

Le résultat est un signal filtré.

Remarque importante

Attention : il est crucial de convertir le signal final dans le format de données d'origine (par exemple, `int16`, `int32` ou `float32`) pour assurer une restitution sonore correcte.

Supposons ici que $f \in \mathbb{R}^N$ (comme dans le cas des signaux sonores).

Symétrie dans les coefficients de Fourier

En conséquence, pour tout k $\in [\![0, N]\!]$, nous obtenons la relation suivante en raison de la **symétrie dans les coefficients de Fourier** :

$$c_{k,d}(f) = \overline{c_{-k+N,d}(f)} \tag{53}$$

Pour comprendre la correspondance entre les **indices** k des **coefficients de Fourier** $c_{k,d}(f)$ et les **fréquences en Hertz**, supposons que f est un **signal temporel échantillonné** à une fréquence f_e. Étant donné que le signal est réel, il suffit d'étudier les valeurs de k pour k $\in [\![0, \frac{N}{2}]\!]$ grâce à la **symétrie des fréquences** (la fréquence $k = \frac{N}{2}$ étant la plus haute représentable).

Appliquons ainsi le **théorème de Dirichlet discret**.

Symétrie dans les coefficients de Fourier

En appliquant le **théorème de Dirichlet discret**, nous obtenons :

$$S_N^D(f)(t_j) := \sum_{k=0}^{N-1} c_{k,d}(f) e^{i\left(k-\frac{N}{2}\right)t_j} = \sum_{a=-\frac{N}{2}}^{\frac{N}{2}-1} c_{a+\frac{N}{2},d}(f) e^{iat_j} \tag{54}$$

Dans cette formule, **l'indice** a représente les **fréquences** : en effet, pour chaque **unité temporelle** t, a doit être dans **l'unité inverse** (i.e., des unités de fréquence). La **symétrie des composantes** e^{ikt} et e^{-ikt} montre que pour un vecteur de longueur N, seules les fréquences de k $\in [\![0, \frac{N}{2}]\!]$ sont accessibles, avec une fréquence pour chaque indice k donnée par :

$$\text{freq}_k := k\frac{f_e}{N} \text{ Hertz}$$

Par conséquent, la **fréquence maximale** accessible via la **transformée de Fourier discrète** d'un signal échantillonné à N points et à une fréquence d'échantillonnage f_e est obtenue pour $k = \frac{N}{2}$, soit :

$$\text{freq}_{\max} = \frac{f_e}{2}$$

Le code python précédents implémente les principes d'un égalisateur, il sépare le spectre sonore en différentes bandes fréquences correspondantes à une plage de fréquence précise. Il sert donc à atténuer ou augmenter certaines plages de fréquences pour rendre le fichier audio adapté au besoin. La transformation de fourrier

discrètes est donc très pertinente dans le fonctionnement d'un égalisateur puisqu'elle nous permet de décomposer un signal complexe (comme un son) en une somme de sinusoïdes de fréquences différentes et nous donne la possibilité d'identifier les plages de fréquences que nous allons modifier.

Par exemple, lors du mixage d'une chanson, réduire les fréquences basses de la voix la rendra plus claire. De plus, réduire les fréquences plus élevées dans une plage spécifique peut ajouter de la chaleur et de la profondeur à la voix.

Nous pouvons donc jouer sur les plages de fréquences et modifier la perception du son.

VII Analyse et Application de la Transformée de Fourier Discrète (TFD) et Effets Sonores en Python

VII.1 Introduction

Les analyses portent sur l'importation, la création, et la manipulation des signaux audio en Python via les, en explorant des applications comme la détection de fréquence fondamentale, le filtrage fréquentiel, et la synthèse de musique. Les effets musicaux tels que le délai (delay) et la réverbération (reverb) sont également analysés pour illustrer l'impact de la transformée de Fourier discrète (TFD) dans l'enrichissement et la modification sonore.

VII.2 Importation et Lecture de Fichiers Audio en Python

Pour lire des fichiers '.wav', la bibliothèque 'scipy.io' fournit la commande 'wavfile.read'. Cette commande retourne deux valeurs : la fréquence d'échantillonnage du signal « fe » et les données audios sous forme de tableau numérique numpy « data ».

```
from scipy.io import wavfile
fe, data = wavfile.read('tempo.wav')
sd.play(data, fe)
```

« fe » est un entier représentant la fréquence d'échantillonnage (en Hertz), qui indique le nombre de valeurs de données audio captées par seconde. 'data' est un tableau numpy contenant les échantillons audio, permettant d'accéder aux données pour le traitement et l'analyse.

VII.3 Place à la génération de signaux et harmoniques

Le code permet de créer un signal sinusoïdal de 440 Hz (un La standard), en spécifiant l'amplitude, la fréquence, et la durée du signal. Cette fréquence est ajustable pour créer différents sons, et en multiplient la fréquence fondamentale, le son devient plus aigu tout en conservant la note d'origine.

```
amplitude = 10
freq = 440
duree = 2
fe = 44100
t = np.linspace(0, duree, fe * duree)
signal = amplitude * np.sin(2 * np.pi * t * freq)
sd.play(signal, fe)
```

VII.4 On va créer des Signaux en Forme d'Ondes (Exemple d'un Signal Carré)

Un signal carré est généré en alternant des intervalles de haute et basse amplitude. Le code illustre comment créer une onde périodique carré.

```
amplitude = 10
freq = 10
duree = 3
fe = 44100
t = np.linspace(0, duree, fe * duree)
signal = amplitude * np.sign(np.sin(2 * np.pi * freq * t))  # Génération d'une onde carrée
sd.play(signal, fe)
```

Le résultat obtenu est illustré à la Fig. 4.

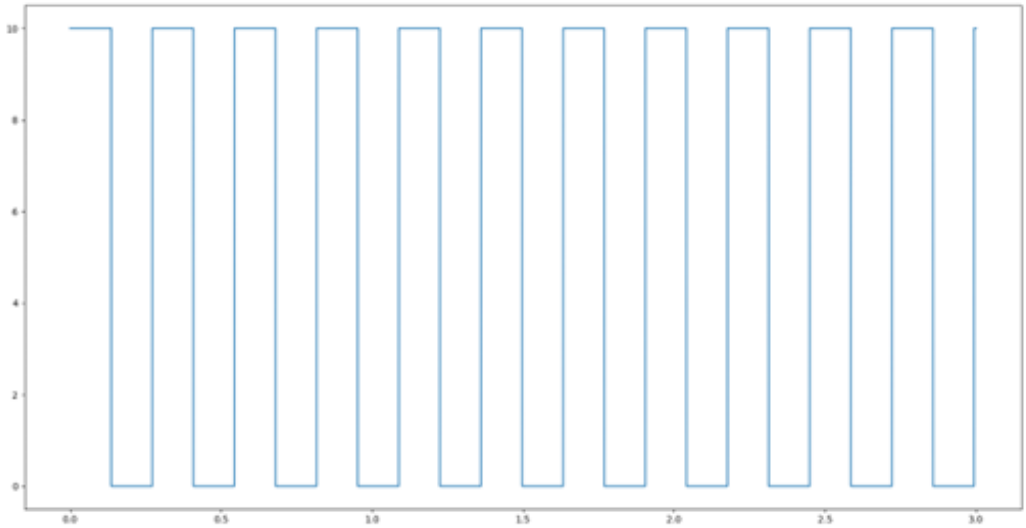


FIGURE 4 – Signal carré sur 3 secondes

Les signaux carrés et triangulaires, par leur structure harmonique unique, produisent des sons très différents d'une onde sinusoïdale pure. Dans la musique numérique, les signaux carrés sont utilisés pour imiter des instruments comme les synthétiseurs et produisent un son plus « métallique » ou « électronique ». Comme par exemple celle des armes feues.

VII.5 Création du morceau de musique : « Ah vous dirai-je Maman »

On a décidé de définir les fréquences des notes de la gamme musicale (Do, Ré, Mi, etc.), le code génère un morceau simple en concaténant des signaux sinusoïdaux correspondant à chaque note. Les « pause » sont nécessaire pour bien différencier chaque note.

```
tempo=1/1.5
amplitude=1
echantillonage=44100
temps = tempo/4

#note pause
pause = np.linspace(0,temps ,int(echantillonage*temps))

#note do
freq=262 #do
temps=tempo/2
```

```
t=np.linspace(0,temps ,int(echantillonage*temps))
notedo=amplitude*np.sin( 2 * np.pi * t * freq )

#note sol
freq=392 #sol
temps=tempo/2
t=np.linspace(0,temps ,int(echantillonage*temps))
notesol=amplitude * np.sin( 2 * np.pi * t * freq )

#on repetete cela pour chaque note
signal=np.block( [ notedo , pause , notedo , pause , notesol ,pause , notesol , pause
     , notela ,pause , notela , pause ,notesollong , pause , notefa , pause , notefa
    ,pause ,notemi , pause ,notemi ,pause, notere ,pause,notere , pause, notedolong
    ])
gros_signal = signal

for i in range(5):
  gros_signal = np.block( [gros_signal , signal] )

sd.play(gros_signal , echantillonage)
```

Cette approche démontre comment les signaux sonores numériques peuvent être utilisés pour recréer des mélodies. En modifiant les fréquences et en utilisant des pauses, le morceau prend forme, illustrant la puissance de la programmation en musique numérique pour générer des séquences de notes dynamiquement.

VII.6 Analyse Fréquentielle d'un Signal de Guitare

Le signal 'guitare1.wav' est un tableau avec deux colonnes (stéréo) et 383 407 lignes (échantillons). Le code applique une TFD rapide sur le signal de guitare pour en analyser les fréquences présentes.

```
spectre = np.fft.rfft(data[:,0])
freq = np.fft.rfftfreq(data[:,0].size, d=1/fe)
spectre_abs = np.abs(spectre)
plt.plot(freq, spectre_abs)
plt.xlabel("Fréquence (Hz)")
plt.ylabel("Amplitude")
```

La fréquence fondamentale (110 Hz) est détectée par la lecture graphique en mesurant l'écart entre deux pics, confirmant qu'il s'agit d'un « La ». Pour cela on a dérivé les valeurs absolues de la transformer de fourrier grâce à la fonction « np.diff ». Le résultat obtenu est présenté à la Fig. 5.

En supprimant les basses fréquences (1 à 1800 Hz), le son devient plus aigu, démontrant l'importance des basses fréquences pour la perception de profondeur et de chaleur dans un son. Nous allons le réaliser dans le code suivant :

```
def supression(spectre , fe , a=0 , b=0):
  spectre = np.fft.rfft(data[:,0] )
  freq = np.fft.rfftfreq(data[:,0].size, d=1/fe)

  spectre_abs = np.abs(spectre)
  spectre[ a:b ] = 0

  new_spectre = np.fft.irfft(spectre)
  new_spectre = new_spectre / np.max(np.abs(new_spectre))
  sd.play(new_spectre , fe)
  plt.plot(freq , spectre )
  plt.xlabel("frequence , Hz")
  plt.ylabel("Amplitude")
```

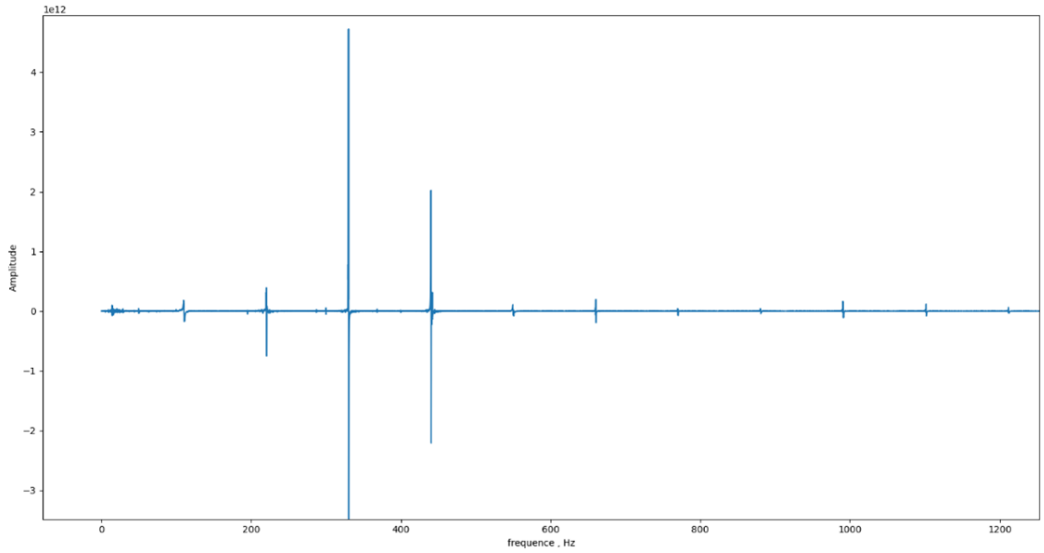


FIGURE 5 – Dérivée de la transformée de Fourier

La Fig. 6. montre des graphiques comparant le son original et le son décalé.

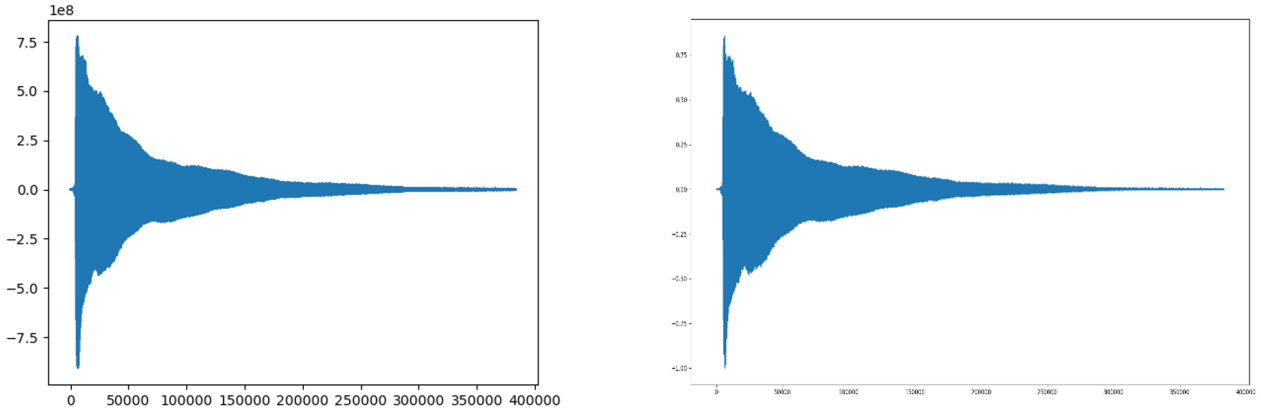


FIGURE 6 – Son original (à gauche) et son décalé (à droite)

En modifiant la fréquence fondamentale, le code permet de transposer le son original pour jouer une nouvelle note, comme un Do à 523 Hz.

Cette analyse montre comment la TFD permet de décomposer et de manipuler les composants fréquentiels du signal. En détectant la fréquence fondamentale, on identifie la note jouée, et en supprimant certaines fréquences, on ajuste la tonalité perçue. Le décalage fréquentiel permet également de transposer le son vers d'autres notes, un outil essentiel en traitement audio.

VII.7 Création d'Effets Musicaux

Effet de Délai

L'effet de délai est créé en ajoutant des échantillons retardés pour simuler un écho. Le signal est mélangé avec lui-même après un décalage temporel, ajusté par les paramètres de rétroaction (‘feedback‘) et de mixage

(‘mix‘).

```
def delay(signal, sampling_rate, delay_time=0.5, feedback=0.5, mix=0.5):
  delay_samples = int(delay_time * sampling_rate)
  output = np.zeros(len(signal) + delay_samples)
  for i in range(len(signal)):
    output[i] += signal[i]
    if i >= delay_samples:
      output[i] += feedback * output[i - delay_samples]
  output = (1 - mix) * signal + mix * output[:len(signal)]
  return output
```

Le délai donne un effet de profondeur, en ajoutant un sentiment de spatialité au son. Les spectres avant et après application du délai montrent une légère atténuation des pics fréquentiels, correspondant à la réverbération introduite.

VII.8 Effet de Réverbération (Convolution)

L’effet de réverbération est obtenu en appliquant une convolution avec une réponse impulsionnelle, modélisant l’acoustique d’un environnement.

```
def conv_fft(signal, impulse_response):
  N = len(signal) + len(impulse_response) - 1
  X = np.fft.rfft(signal, n=N)
  H = np.fft.rfft (impulse_response, n=N)
  Y = X * H
  y = np.fft.irfft(Y)
  return y[:len(signal)]
```

L’utilisation d’une convolution avec une réponse impulsionnelle introduit une réverbération, recréant la sensation d’un environnement sonore. En modifiant la réponse impulsionnelle, on simule différents types de pièces, ajoutant une profondeur et une richesse acoustiques au son.

VII.9 Conclusion

Ce rapport démontre les multiples applications de la transformée de Fourier discrète dans le traitement audio. L’importation et la génération de signaux, la détection de notes, et l’application d’effets sonores illustrent les possibilités de manipulation des signaux numériques en Python. Les exemples d’effets comme le délai et la réverbération montrent comment la TFD enrichit l’expérience sonore en ajustant les caractéristiques spectrales.