Séries de Fourier et FFT

Étienne Dupuis
Université d'Ottawa

Mars 1999

1 Introduction

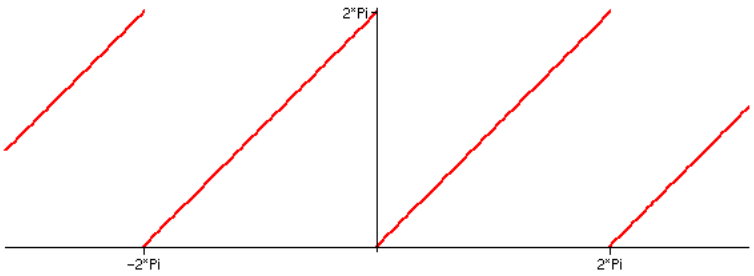
Soit $g : \mathbb{R} \to \mathbb{R}$ une fonction périodique de période 2π. Nous avons probablement tous vu, lors d'un cours de calcul intégral, que si certaines conditions sont satisfaites la fonction $g(t)$ peut être approchée d'aussi près que l'on veut par sa série de Fourier $F_g(x)$, donnée par

$$F_g(x) = \frac{a_0}{2} + \sum_{j=1}^{\infty} \left(a_j \cos(jx) + b_j \sin(jx) \right), \tag{1}$$

où

$$\begin{aligned} a_j &= \frac{1}{\pi} \int_{-\pi}^{\pi} g(t) \cos(jt) dt \\ b_j &= \frac{1}{\pi} \int_{-\pi}^{\pi} g(t) \sin(jt) dt. \end{aligned}$$

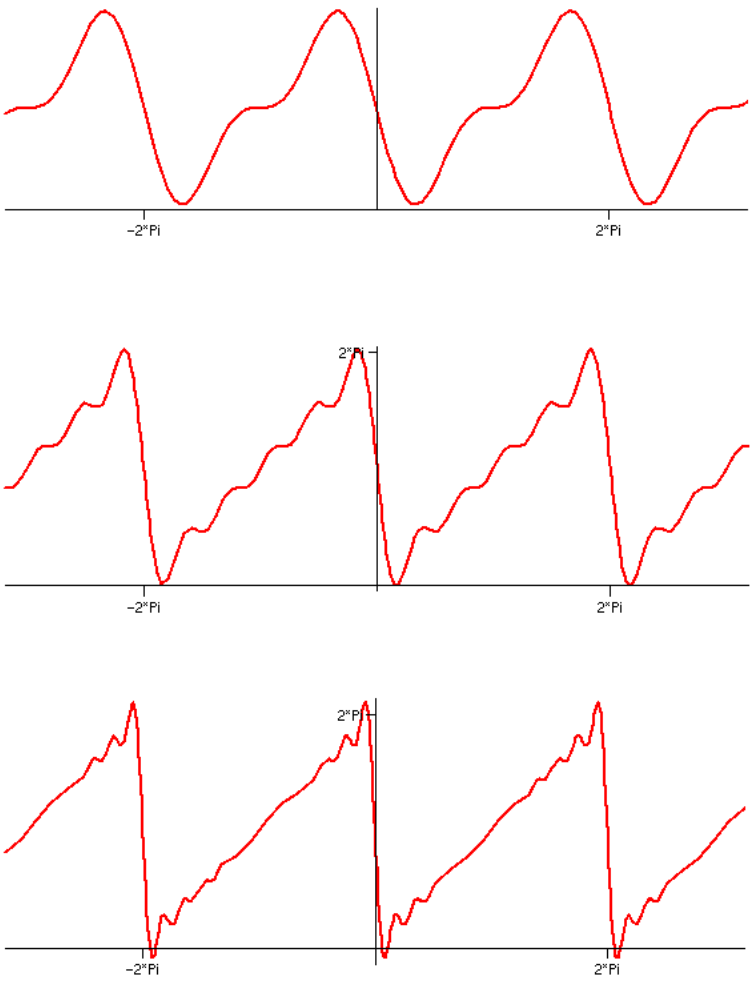
Considérons par exemple la fonction périodique définie par $g(t) = t$ pour $0 \leq t < 2\pi$:



Sa série de Fourier est donnée par (cf Spiegel [8])

$$F_g(x) = \pi - 2\sum_{j=1}^{\infty} \frac{\sin jx}{j},$$

dont voici trois approximations, la première étant la somme des trois premiers termes (π, $-2\sin(x)$ et $-\sin(2x)$), la deuxième étant la somme des six premiers termes et la dernière étant la somme des 12 premiers termes de la série :



Dans ce document nous nous intéresserons plus particulièrement à l'évaluation numérique des coefficients a_j et b_j. Cependant, avant de poursuivre la discussion,

il serait bon de définir rigoureusement le concept de série de Fourier.

Lemme 1.1. *Soit $\mathbb{T}$ le cercle unité dans le plan complexe, et $\tilde{f} : \mathbb{T} \to \mathbb{C}$. Alors $f : \mathbb{R} \to \mathbb{C}$ définie par $f(t) = \tilde{f}(e^{it})$ est une fonction périodique de période 2π. De plus, si $f : \mathbb{R} \to \mathbb{C}$ est une fonction périodique de période 2π, il existe une fonction unique $\tilde{f} : \mathbb{T} \to \mathbb{C}$ telle que $f(t) = \tilde{f}(e^{it})$.*

En d'autres mots, on peut associer les fonctions définie sur $\mathbb{T}$ aux fonctions périodiques de période 2π définie sur $\mathbb{R}$. En munissant $\mathbb{T}$ d'une mesure μ, soit celle de Lebesgue divisée par 2π, on peut définir l'espace de Banach $L^p(\mathbb{T})$, $1 \leq p < \infty$, qui est l'ensemble de toutes les fonctions mesurables pour lesquelles la norme

$$\|f\|_p = \left(\int_{\mathbb{T}} |f(t)|^p d\mu \right)^{1/p} = \left(\frac{1}{2\pi} \int_0^{2\pi} |f(t)|^p dt \right)^{1/p}$$

est finie. En fait, $L^p(\mathbb{T})$ est un ensemble de classes d'équivalences, deux fonctions étant équivalentes si elles sont égales presque partout, mais nous ne nous arrêterons pas à ces détails d'analyse fonctionnelle. Nous sommes maintenant prêts à définir le concept de série de Fourier :

Définition 1.1. *Soit $g \in L^1(\mathbb{T})$. La série de Fourier de g est donnée par*

$$F_g(x) = \lim_{N \to \infty} \sum_{j=-N}^{N} \hat{g}_j e^{ijx}, \tag{2}$$

où les coefficients de Fourier $\hat{g}_j$ sont donnés par

$$\hat{g}_j = \frac{1}{2\pi} \int_0^{2\pi} g(t) e^{-ijt} dt. \tag{3}$$

Le théorème qui suit justifie cette définition. Mais avant de l'énoncer, rappelons-nous que puisque $\mu(\mathbb{T}) = 1 < \infty$, $L^2(\mathbb{T}) \subseteq L^1(\mathbb{T})$. Ainsi on peut définir la série de Fourier de g pour tout $g \in L^2(\mathbb{T})$. La raison pour laquelle nous nous intéressons à $L^2(\mathbb{T})$ plutôt qu'à $L^1(\mathbb{T})$ est que le premier est un espace de Hilbert avec le produit scalaire

$$\langle f_1 | f_2 \rangle = \int_{\mathbb{T}} f \bar{g} d\mu.$$

Ainsi, le théorème de Riesz-Fischer pour les espaces complets et l'égalité de Parseval impliquent le théorème suivant :

Théorème 1.1. *L'ensemble $\{e^{ijx}\}_{j \in \mathbb{Z}}$ est orthonormal et maximal (par conséquent dense) dans $L^2(\mathbb{T})$, et pour pour tout $g \in L^2(\mathbb{T})$, g est égal (presque partout) à sa série de Fourier F_g.*

En d'autres mots, l'application $g \mapsto \hat{g} = \{\hat{g}_j\}_{j \in \mathbb{Z}}$ est un isomorphisme entre les espaces de Hilbert $L^2(\mathbb{T})$ et $l^2(\mathbb{Z})$. Nous omettons la preuve de ce résultat, que le lecteur intéressé pourra trouver dans RUDIN [7]. Avant de conclure cette introduction, mentionnons que

Lemme 1.2. *Les définitions* (1) *et* (2) *données pour la série de Fourier F_g de g sont équivalentes.*

Démonstration:

Définissons, afin d'alléger la notation,

$$\tilde{g}_j = \frac{1}{2\pi}\int_0^{2\pi} g(t)e^{ijt}dt.$$

On déduit aisément, en se servant des identités d'Euler et du fait que $\cos(jt)$ et $\sin(jt)$ sont des fonctions périodiques de période 2π pour $j = 1, 2, \ldots$, les relations suivantes :

$$\begin{aligned}
a_j &= \frac{1}{\pi}\int_{-\pi}^{\pi} g(t)\cos(jt)dt \\
&= \frac{1}{2\pi}\int_0^{2\pi} g(t)(e^{ijt} + e^{-ijt})dt \\
&= \frac{1}{2\pi}\int_0^{2\pi} g(t)e^{ijt}dt + \frac{1}{2\pi}\int_0^{2\pi} g(t)e^{-ijt}dt \\
&= \tilde{g}_j + \hat{g}_j \\
b_j &= \frac{\tilde{g}_j - \hat{g}_j}{i} \\
&= i\hat{g}_j - i\tilde{g}_j.
\end{aligned}$$

Maintenant,

$$\begin{aligned}
a_j\cos(jx) + b_j\sin(jx) &= (\tilde{g}_j + \hat{g}_j)\frac{e^{ijx} + e^{-ijx}}{2} + (i\hat{g}_j - i\tilde{g}_j)\frac{e^{ijx} - e^{-ijx}}{2i} \\
&= \tilde{g}_j e^{-ijx} + \hat{g}_j e^{ijx},
\end{aligned}$$

d'où on tire, grâce à la relation $\tilde{g}_{-j} = \hat{g}_j$, que

$$\begin{aligned}
F_g(x) &= \frac{a_0}{2} + \sum_{j=1}^{\infty}(\tilde{g}_j e^{-ijx} + \hat{g}_j e^{ijx}) \\
&= \frac{\tilde{g}_0 + \hat{g}_0}{2} + \sum_{j=-1}^{-\infty}\tilde{g}_{-j}e^{ijx} + \sum_{j=1}^{\infty}\hat{g}_j e^{ijx} \\
&= \hat{g}_0 + \sum_{j=-1}^{-\infty}\hat{g}_j e^{ijx} + \sum_{j=1}^{\infty}\hat{g}_j e^{ijx} \\
&= \sum_{j=-\infty}^{\infty}\hat{g}_j e^{ijx}
\end{aligned}$$

□

Nous allons utiliser, tout au long de ce document, la définition (2), qui est plus concise et par conséquent plus facile à étudier. Remarquez que cette définition se généralise aisément aux fonctions périodiques de période autre que 2π. Nous allons maintenant aborder le calcul numérique des coefficients de Fourier $\hat{g}_j$.

2 DFT : Définition et propriétés

Soit $n > 0$ un entier. Posons

$$\Pi_n = \{\{x_k\}_{k\in\mathbb{Z}} \mid x_{k+n} = x_k,\ x_k \in \mathbb{C}\ \ \forall k \in \mathbb{Z}\}\,.$$

C'est l'ensemble des suites bi-infinies périodiques de période n. Évidemment, toute suite de Π_n est entièrement et uniquement déterminée par x_0, x_1, …, x_{n-1}. Π_n est donc un espace vectoriel de dimension n isomorphe à $\mathbb{C}^n$. Nous sommes maintenant prêt à définir une DFT :

Définition 2.1. *Soit $\omega_n = e^{\frac{2\pi i}{n}}$ une n-ième racine de l'unité. La transformée de Fourier discrète (DFT) d'ordre n est l'application $\mathcal{F}_n : \Pi_n \to \Pi_n$ définie par $y = \mathcal{F}_n(x)$, où*

$$y_j = \frac{1}{n}\sum_{k=0}^{n-1} \omega_n^{-jk} x_k. \tag{4}$$

À première vue, il n'est pas évident que y est bien un élément de Π_n. Cependant, puisque $\omega_n^n = 1$, on a

$$\begin{aligned} y_{j+n} &= \frac{1}{n}\sum_{k=0}^{n-1} \omega_n^{-(j+n)k} x_k \\ &= \frac{1}{n}\sum_{k=0}^{n-1} \omega_n^{-jk} (\omega_n^n)^{-k} x_k \\ &= \frac{1}{n}\sum_{k=0}^{n-1} \omega_n^{-jk} x_k \\ &= y_j, \end{aligned}$$

ce qui prouve que $y \in \Pi_n$.

Lemme 2.1. *$\mathcal{F}_n$ est en fait un automorphisme de Π_n dont l'inverse $x = \mathcal{F}_n^{-1}(y)$ est donné par*

$$x_k = \sum_{j=0}^{n-1} \omega_n^{jk} y_j.$$

Un automorphisme est simplement un isomorphisme d'un espace vectoriel dans lui-même. Iserles [4] considère une application linéaire injective comme étant un

isomorphisme alors que pour nous un isomorphisme est une application linéaire bijective, soit injective et surjective.

Démonstration:

La linéarité de l'application $\mathcal{F}_n$ est aisément vérifiée à partir de l'équation (4) la définissant. Vérifions maintenant la formule donnée pour l'inverse en calculant $\mathcal{F}_n^{-1}(\mathcal{F}_n(x))$:

$$\begin{aligned}
x_s &= \sum_{j=0}^{n-1} \omega_n^{js} \frac{1}{n} \sum_{k=0}^{n-1} \omega_n^{-jk} x_k \\
&= \frac{1}{n} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} \omega_n^{js-jk} x_k \\
&= \frac{1}{n} \sum_{k=0}^{n-1} \left(\sum_{j=0}^{n-1} \omega_n^{js-jk} \right) x_k \\
&= \frac{1}{n} \sum_{j=0}^{n-1} \omega_n^{js-js} x_s + \frac{1}{n} \sum_{\substack{k=0 \\ k \neq s}}^{n-1} \left(\sum_{j=0}^{n-1} \omega_n^{(s-k)j} \right) x_k \\
&= \frac{1}{n} \sum_{j=0}^{n-1} x_s + \frac{1}{n} \sum_{\substack{k=0 \\ k \neq s}}^{n-1} \left(\frac{1-(\omega_n^{s-k})^n}{1-(\omega_n^{s-k})} \right) x_k \\
&= x_s + \underbrace{\frac{1}{n} \sum_{\substack{k=0 \\ k \neq s}}^{n-1} \underbrace{\left(\frac{1-1^{s-k}}{1-\omega_n^{s-k}} \right)}_{0} x_k}_{0} \\
&= x_s.
\end{aligned}$$

Puisque qu'à chaque élément $y = \mathcal{F}_n(x)$ correspond un inverse x, l'application est automatiquement injective. On en conclut donc qu'elle est également surjective puisque Π_n est un espace vectoriel de dimension $n < \infty$. □

Afin d'évaluer numériquement les coefficients de Fourier $\hat{g}_j$, nous devons remplacer l'intégrale (3) par une somme finie. C'est ici que la DFT entre en jeu. En effet, soit $n > 0$ un entier et posons $x_k = \frac{2\pi k}{n}$. L'ensemble $\{x_k \mid k = 0, 1, \ldots, n\}$ est une partition de l'intervalle $[0, 2\pi]$ qui nous permet d'évaluer approximativement les coefficients $\hat{g}_j$:

$$\begin{aligned}
\hat{g}_j &= \frac{1}{2\pi} \int_0^{2\pi} g(t) e^{-ijt} dt \\
&\approx \frac{1}{2\pi} \sum_{k=0}^{n-1} (x_{k+1} - x_k) g(x_k) e^{-ijx_k}
\end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{2\pi}\sum_{k=0}^{n-1}\frac{2\pi}{n}g(x_k)e^{\frac{-2ijk\pi}{n}} \\
&= \frac{1}{n}\sum_{k=0}^{n-1}\omega_n^{-jk}g_k,
\end{aligned}
$$

où $g_k = g(x_k)$. Cette approximation, que nous appellerons maintenant $\hat{h}_j$, ressemble étrangement à une DFT. Cependant, puisque j varie de $-\infty$ à $+\infty$ dans l'équation (2), nous sommes intéressés non pas par les coefficients $\hat{g}_j$ pour $j = 0, 1, \ldots, n-1$ mais plutôt, en supposant n pair, pour $j = -n/2+1, -n/2+2, \ldots, n/2-1, n/2$. En remplaçant j par $l - n/2 + 1$ dans l'approximation ci-haut et en utilisant la relation $\omega_n^{\frac{n}{2}} = -1$, on obtient

$$
\begin{aligned}
\hat{h}_j &= \frac{1}{n}\sum_{k=0}^{n-1}\omega_n^{-(l-\frac{n}{2}+1)k}g_k \\
&= \frac{1}{n}\sum_{k=0}^{n-1}\omega_n^{-lk}(-\omega_n^{-1})^k g_k \\
&= (\mathcal{F}_n(\tilde{\mathbf{g}}))_l, \qquad (5)
\end{aligned}
$$

où $\tilde{\mathbf{g}} = \{-\omega_n^{-k}g_k\}_{k\in\mathbb{Z}}$ et $(\mathcal{F}_n(\tilde{\mathbf{g}}))_l$ est la l-ième composante de $\mathcal{F}_n(\tilde{\mathbf{g}})$. Ainsi, en effectuant une seule DFT, nous calculons d'un seul coup les approximations des n plus importants coefficients $\hat{g}_j$ de la série de Fourier de g, soit les coefficients $\hat{g}_{-\frac{n}{2}+1}$ à $\hat{g}_{\frac{n}{2}}$. Avant de poursuivre, attardons-nous un peu sur la validité des approximations $\hat{h}_j$ ainsi que sur le choix de la partition x_k.

Lemme 2.2. *Soit S_n le sous-espace de $L^2(\mathbb{T})$ engendré par l'ensemble orthonormal $\{e^{ilt} \mid |l| < n\}$ de cardinalité $2n-1$. Soit $f \in S_n$ et soit $x_k = \frac{2k\pi}{n}$. Alors la formule de quadrature*

$$
\frac{1}{2\pi}\int_0^{2\pi} f(t)dt \approx \frac{1}{n}\sum_{k=0}^{n-1} f(x_k) \qquad (6)
$$

est exacte.

Démonstration:

Écrivons

$$
f(t) = \sum_{l=-n+1}^{n-1} c_j e^{ilt},
$$

où les c_j sont des nombres complexes. On calcule la valeur du membre de gauche de (6) :

$$
\frac{1}{2\pi}\int_0^{2\pi} f(t)dt = \frac{1}{2\pi}\sum_{l=-n+1}^{n-1}\int_0^{2\pi} c_l e^{ilt}dt
$$

$$
\begin{aligned}
&= \frac{1}{2\pi}\left(\int_0^{2\pi} c_0 dt + \sum_{\substack{l=-n+1\\ l\neq 0}}^{n-1} c_l \int_0^{2\pi} e^{ilt} dt\right)\\
&= c_0 + \frac{1}{2\pi}\underbrace{\sum_{\substack{l=-n+1\\ l\neq 0}}^{n-1} c_l \underbrace{\frac{e^{ilt}}{il}\Big|_{t=0}^{2\pi}}_{0}}_{0}\\
&= c_0,
\end{aligned}
$$

ainsi que celle du membre de droite :

$$
\begin{aligned}
\frac{1}{n}\sum_{k=0}^{n-1} f(x_k) &= \frac{1}{n}\sum_{k=0}^{n-1}\sum_{l=-n+1}^{n-1} c_l e^{\frac{2ilk\pi}{n}}\\
&= \frac{1}{n}\sum_{l=-n+1}^{n-1} c_l \sum_{k=0}^{n-1} \omega_n^{lk}\\
&= \frac{1}{n}\left(c_0 \sum_{k=0}^{n-1} 1 + \sum_{\substack{l=-n+1\\ l\neq 0}}^{n-1} c_l \sum_{k=0}^{n-1} \omega_n^{lk}\right)\\
&= c_0 + \frac{1}{n}\underbrace{\sum_{\substack{l=-n+1\\ l\neq 0}}^{n-1} c_l \underbrace{\frac{1-(\omega_n^l)^n}{1-(\omega_n^l)}}_{0}}_{0}\\
&= c_0,
\end{aligned}
$$

ce qui prouve le résultat. □

On en déduit immédiatement le corollaire suivant :

Lemme 2.3. *Supposons toujours n pair afin de faciliter la notation. Soit g une fonction du sous-espace de $L^2(\mathbb{T})$ engendré par l'ensemble orthonormal $\{e^{ilt} \mid -\frac{n}{2} < l \leq \frac{n}{2}\}$ de cardinalité n. Alors les approximations $\hat{h}_j$ données par la DFT* (5) *sont exactes. De plus, par choix de g, les autres coefficients de Fourier (pour $j \leq -\frac{n}{2}$ et $j > \frac{n}{2}$) sont nuls.*

En d'autres mots, en effectuant une DFT d'ordre n, on peut calculer exactement la série de Fourier de n'importe quelle fonction g engendrée par l'ensemble des n *premières* fonctions trigonométriques. La preuve n'est qu'une application du lemme 2.2 :

Démonstration:

Posons $f_j(t) = g(t)e^{-ijt}$. Alors par choix de g, pour tout $j \in \{-n/2+1, \ldots, n/2\}$,

f_j est élément de S_n. Ainsi, puisque

$$\hat{g}_j = \frac{1}{2\pi}\int_0^{2\pi} f_j(t)dt,$$

le lemme précédent implique que $\hat{g}_j = \hat{h}_j$ pour tout $j \in \{-n/2+1, \ldots, n/2\}$. $\square$

Ce lemme peut également être obtenu comme conséquence directe du lemme suivant, qui caractérise l'erreur provenant de l'approximation $\hat{h}_j$:

Lemme 2.4. *Supposons que les sommes partielles de la série de Fourier de g convergent uniformément (vers g). Alors l'erreur provenant de l'approximation des coefficients de Fourier est donnée par*

$$\hat{h}_s - \hat{g}_s = \sum_{\nu \in \mathbb{Z}\setminus\{0\}} \hat{g}_{s+\nu n}. \tag{7}$$

Avant de passer à la démonstration, il y a plusieurs remarques intéressantes à faire. D'abord, si g satisfait aux conditions du lemme 2.3, alors sa série de Fourier converge uniformément vers (un représentant de la classe d'équivalence de) g, ainsi le lemme 2.4 s'applique. On en déduit immédiatement que le membre de droite de (7) est nul, car par choix de g si $\nu \neq 0$ alors $|s+\nu n| \geq \frac{n}{2}$ pour $-\frac{n}{2} < s \leq \frac{n}{2}$ et donc $\hat{g}_{s+\nu n} = 0$ pour tout $\nu \neq 0$, ce qui prouve le lemme 2.3. Deuxièmement, si les sommes partielles de la série de Fourier F_g de g (pour un certain $g \in L^2(\mathbb{T})$) convergent rapidement vers F_g, on peut s'attendre à ce que l'erreur $\hat{h}_s - \hat{g}_s$ diminue rapidement en fonction de n. Finalement, n'est-il pas étonnant que l'erreur totale commise en estimant n coefficients de Fourier soit égale à la somme de tous les autres coefficients de Fourier ? Les mathématiques sont parfois pour le moins étranges, même si elles sont la conséquence de preuves irréfutables :

Démonstration:

Puisque les sommes partielles de F_g convergent uniformément vers g, on calcule

$$\begin{aligned}
\hat{h}_s &= \frac{1}{n}\sum_{k=0}^{n-1} \omega_n^{-sk} g_k \\
&= \frac{1}{n}\sum_{k=0}^{n-1} F_g(x_k)\omega_n^{-sk} \\
&= \frac{1}{n}\sum_{k=0}^{n-1} \left(\sum_{j=-\infty}^{\infty} \hat{g}_j e^{\frac{2ijk\pi}{n}} \right) \omega_n^{-sk} \\
&= \frac{1}{n}\sum_{j=-\infty}^{\infty} \hat{g}_j \underbrace{\left(\sum_{k=0}^{n-1} \omega_n^{(j-s)k} \right)}_{=0 \text{ si } j \not\equiv s \bmod n}.
\end{aligned}$$

Nous avons déjà calculé au moins deux fois une expression comme celle entre parenthèse ci-haut. Il suffit, si $\omega_n^{(j-s)k} \neq 1$ de calculer la somme de la progression

géométrique pour obtenir une expression dont le numérateur est nul. Si $\omega_n^{(j-s)k} = 1$, ce qui est le cas lorsque $j \equiv s \bmod n$, alors l'expression entre parenthèse vaut simplement n, d'où

$$\begin{aligned} \hat{h}_s &= \sum_{\{j \mid j \equiv s \bmod n\}} \hat{g}_j \\ &= \sum_{\nu=-\infty}^{\infty} \hat{g}_{s+\nu n}, \end{aligned}$$

ce qui prouve le résultat. □

Avant de clore cette section consacrée à la DFT, mentionnons que l'application $y = \mathcal{F}_n(x)$ peut être écrite sous forme matricielle :

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{pmatrix} = \frac{1}{n} \begin{pmatrix} 1 & \omega_n^{-0} & \omega_n^{-0} & \cdots & \omega_n^{-0(n-1)} \\ 1 & \omega_n^{-1} & \omega_n^{-2} & \cdots & \omega_n^{-1(n-1)} \\ 1 & \omega_n^{-2} & \omega_n^{-4} & \cdots & \omega_n^{-2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{-1(n-1)} & \omega_n^{-2(n-1)} & \cdots & \omega_n^{-(n-1)^2} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{n-1} \end{pmatrix}. \quad (8)$$

Le calcul d'une DFT est donc à priori de complexité $\mathcal{O}\left(n^2\right)$. Rappelons qu'une complexité de ce type signifie que le nombre d'opérations à effectuer pour un problème de taille n est de l'ordre de n^2. Il est maintenant temps d'introduire la FFT, qui n'est autre qu'un algorithme efficace pour calculer une DFT.

3 FFT : Calcul efficace d'une DFT

L'algorithme que nous allons présenter maintenant est essentiellement celui de Cooley-Tukey, (re)découvert dans les années soixante (cf Cooley-Tukey [2]). Iserles [4] mentionne qu'un certain Lanczos en serait en fait l'auteur et Leighton [6] fait remonter l'algorithme aux années vingt. Finalement, certains croient que Gauss le connaissait. Qui dit vrai ?

Nous allons limiter notre étude au cas simple où $n = 2^m$, $m > 0$. Soit $x = \{x_k\}_{k\in\mathbb{Z}} \in \Pi_n$ et définissons, afin de simplifier la discussion qui va suivre,

$$\mathcal{F}_n^*(x) = n\mathcal{F}_n(x).$$

Étant donné $\mathcal{F}_n^*(x)$, le calcul de $\mathcal{F}_n(x)$ ne consiste qu'à multiplier le vecteur colonne $\mathcal{F}_n^*(x)$ par le scalaire $1/n$, ce qui est une opération de complexité $\mathcal{O}(n)$. Ainsi nous sommes intéressés par le calcul efficace de $\mathcal{F}_n^*(x)$. Définissons

$$\begin{aligned} x^{[0]} &= \{x_{2k}\}_{k\in\mathbb{Z}} \\ x^{[1]} &= \{x_{2k+1}\}_{k\in\mathbb{Z}}, \end{aligned}$$

soit les composantes paires et impaires de la suite bi-infinie $\{x_k\}_{k\in\mathbb{Z}}$. Puisque $x^{[0]},\ x^{[1]} \in \Pi_{\frac{n}{2}}$, on peut calculer $y^{[0]} = \mathcal{F}^*_{\frac{n}{2}}(x^{[0]})$ et $y^{[1]} = \mathcal{F}^*_{\frac{n}{2}}(x^{[1]})$. Puis, à partir de ces deux valeurs, on calcule $y = \mathcal{F}^*_n(x)$ de la façon suivante :

$$\begin{aligned}
y_j &= \sum_{k=0}^{n-1} \omega_n^{-jk} x_k \\
&= \sum_{\substack{k=0 \\ k \text{ pair}}}^{n-1} \omega_n^{-jk} x_k + \sum_{\substack{k=0 \\ k \text{ impair}}}^{n-1} \omega_n^{-jk} x_k \\
&= \sum_{k=0}^{\frac{n}{2}-1} \omega_n^{-2jk} x_{2k} + \sum_{k=0}^{\frac{n}{2}-1} \omega_n^{-j(2k+1)} x_{2k+1} \\
&= \sum_{k=0}^{\frac{n}{2}-1} \omega_{\frac{n}{2}}^{-jk} x_{2k} + \omega_n^{-j} \sum_{k=0}^{\frac{n}{2}-1} \omega_{\frac{n}{2}}^{-jk} x_{2k+1} \\
&= y_j^{[0]} + \omega_n^{-j} y_j^{[1]}.
\end{aligned}$$

En d'autres mots, étant donnés $y^{[0]}$ et $y^{[1]}$, il est possible de les combiner en $\mathcal{O}(n)$ opérations pour obtenir y. Les calculs ci-haut peuvent cependant être simplifiés davantage. En effet, $y^{[\alpha_1]}$ (où $\alpha_\nu \in \{0,1\}$ pour éventuellement $\nu = 1,2,\ldots,m$) étant périodique de période $n/2$, $y_j^{[\alpha]} = y_{j+\frac{n}{2}}^{[\alpha]}$. De plus, puisque $\omega_n^{\frac{n}{2}} = -1$, on peut calculer simultanément deux composantes de y :

$$\left\{ \begin{array}{rcl} y_j &=& y_j^{[0]} + \omega_n^{-j} y_j^{[1]} \\ y_{j+\frac{n}{2}} &=& y_j^{[0]} - \omega_n^{-j} y_j^{[1]}. \end{array} \right. \tag{9}$$

Il ne reste plus qu'à régler un petit détail, soit celui du calcul des deux $y^{[\alpha]}$. On pourrait bien sûr les calculer en effectuant la multiplication matricielle (8), ce qui nous donnerait une complexité totale de $\mathcal{O}(n) + 2\mathcal{O}\left(n^2/4\right)$, qui est en fait égale à $\mathcal{O}\left(n^2\right)$, soit la même complexité que la formule (8). Cependant, il est possible de faire beaucoup mieux en appliquant exactement le même raisonnement que nous venons d'effectuer. En effet, il suffit de définir

$$\begin{aligned}
x^{[00]} &= \{x_{2(2k)}\}_{k\in\mathbb{Z}} \\
x^{[01]} &= \{x_{2(2k+1)}\}_{k\in\mathbb{Z}} \\
x^{[10]} &= \{x_{2(2k)+1}\}_{k\in\mathbb{Z}} \\
x^{[11]} &= \{x_{2(2k+1)+1}\}_{k\in\mathbb{Z}},
\end{aligned}$$

de calculer les 4 transformées d'ordre $n/4$

$$y^{[\alpha_1\alpha_2]} = \mathcal{F}^*_{\frac{n}{4}}(x^{[\alpha_1\alpha_2]})$$

et finalement de calculer

$$y_j^{[0]} = y_j^{[00]} + \omega_{\frac{n}{2}}^{-j} y_j^{[01]}$$

$$y_j^{[1]} = y_j^{[10]} + \omega_{\frac{n}{2}}^{-j} y_j^{[11]}$$

pour $j = 0, 1, \ldots, \frac{n}{2} - 1$ (ou $j = 0, 1, \ldots, \frac{n}{4} - 1$ si on calcule de façon analogue à la formule (9) deux composantes simultanément). Les lecteurs familiers avec le principe de récurrence verront immédiatement que les 4 valeurs $y^{[\alpha_1\alpha_2]}$ peuvent à leur tour être obtenue de huit transformée d'ordre $n/8$, qui peuvent à leur tour être obtenue de seize transformées d'ordre $n/16$, etc. On répète le processus jusqu'à l'étape où

$$\begin{aligned} y_j^{[\alpha_1\alpha_2\cdots\alpha_m]} &= \mathcal{F}_1^*(x_j^{[\alpha_1\alpha_2\cdots\alpha_m]}) \\ &= x_j^{[\alpha_1\alpha_2\cdots\alpha_m]}, \end{aligned} \tag{10}$$

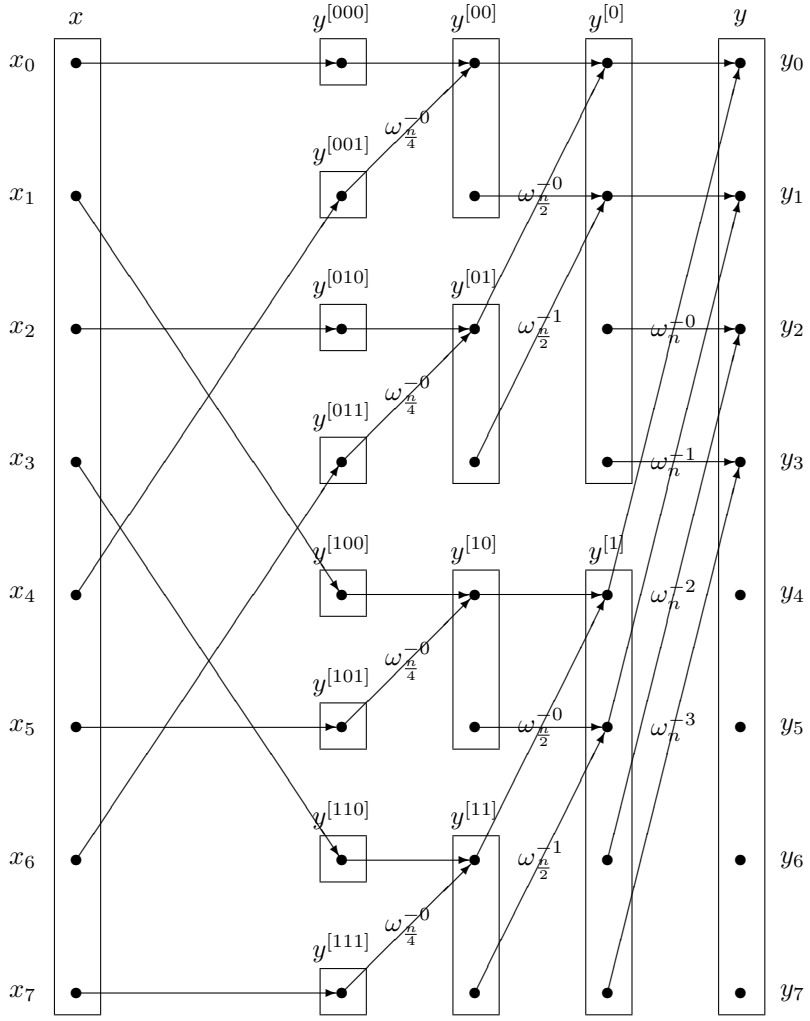
pour $j = 0$, où

$$\begin{aligned} x^{[\alpha_1\alpha_2\cdots\alpha_m]} &= \{x_{2(2(\cdots(2k+\alpha_m)\cdots)+\alpha_2)+\alpha_1}\}_{k\in\mathbb{Z}} \\ &= \{x_{2^m k+2^{m-1}\alpha_m+2^{m-2}\alpha_{m-1}+\cdots+\alpha_1}\}_{k\in\mathbb{Z}} \\ &= x_\alpha, \end{aligned} \tag{11}$$

où

$$\alpha = \sum_{\nu=1}^{m} 2^{\nu-1}\alpha_\nu. \tag{12}$$

Le schéma suivant représente le déroulement de l'algorithme dans le cas où $n = 2^m = 8 = 2^3$:


x
$y^{[000]}$
$y^{[00]}$
$y^{[0]}$
y
x_0
x_1
x_2
x_3
x_4
x_5
x_6
x_7
$y^{[001]}$
$y^{[010]}$
$y^{[011]}$
$y^{[100]}$
$y^{[101]}$
$y^{[110]}$
$y^{[111]}$
$y^{[01]}$
$y^{[10]}$
$y^{[11]}$
$y^{[1]}$
y_0
y_1
y_2
y_3
y_4
y_5
y_6
y_7
$\omega_{\frac{n}{4}}^{-0}$
$\omega_{\frac{n}{2}}^{-0}$
$\omega_{\frac{n}{2}}^{-1}$
ω_n^{-0}
ω_n^{-1}
ω_n^{-2}
ω_n^{-3}

Tout d'abord, à partir du vecteur x, on calcule les huit transformées d'ordre 1, ce qui est équivalent à changer l'ordre des éléments du vecteur. En effet, en écrivant l'indice d'une composante de x en binaire, les relations (11) et (12) sont équivalentes à l'égalité

$$x_{\alpha_m \alpha_{m-1} \cdots \alpha_1} = x^{[\alpha_1 \alpha_2 \cdots \alpha_m]}$$

et selon (10) une transformée d'ordre 1 n'est que l'identité. Il suffit donc d'inverser les bits de l'indice de chaque composante de x pour connaître leur nouvelle position. À partir de ces huit vecteurs de longueur 1 on en calcule 4 autres (de longueur 2) à l'aide de la formule (9), et ensuite deux autres (de longueur 4) et finalement un dernier, qui est le vecteur y cherché. La complexité totale de cet algorithme se calcule aisément. Puisque la complexité de la combinaison de deux transformées d'ordre $n/2$ est $\mathcal{O}(n)$, on obtient pour l'algorithme une complexité de

$$\begin{aligned} \mathcal{O}(n) + 2\mathcal{O}(n/2) + 4\mathcal{O}(n/4) + \cdots + n\mathcal{O}(1) &= \sum_{\nu=0}^{m} 2^{\nu} \mathcal{O}\left(2^{-\nu} n\right) \\ &= \sum_{\nu=0}^{m} \mathcal{O}(n) \\ &= \mathcal{O}(nm) \\ &= \mathcal{O}(n \log_2 n), \end{aligned}$$

toute une amélioration par rapport à la multiplication matricielle de complexité $\mathcal{O}\left(n^2\right)$. Afin de donner une idée du gain, le temps requis pour effectuer une DFT d'ordre $2^{10} = 1024$ sera comparable à celui nécessaire pour effectuer une FFT d'ordre $2^{16} = 65536$. Les lecteurs intéressés par la programmation de cet algorithme et le temps d'exécution sont invités à consulter l'annexe A.

4 Une application

Les applications de la transformée de Fourier rapide, la FFT, sont innombrables. Elle est omniprésente dans le traitement, l'interpolation et l'analyse de signaux de toutes sortes et indispensable pour la résolution de certains types d'équations différentielles. Cependant, l'application que j'ai choisi de présenter n'appartient pas à ni l'une ni l'autre de ces deux immenses catégories. En effet, nous allons utiliser la FFT pour accélérer le calcul du produit de deux entiers.

Soit $h > 1$ un entier et soient a, b deux entiers positifs de n chiffres exprimés en base h :

$$\begin{aligned} a &= \sum_{\nu=0}^{n-1} a_\nu h^\nu \\ b &= \sum_{\nu=0}^{n-1} b_\nu h^\nu, \end{aligned}$$

où a_ν, $b_\nu \in \{0, 1, \ldots, h-1\}$ pour $\nu = 0, 1, \ldots, n-1$. Le produit $c = ab$ est donné par le polynôme

$$c = \sum_{k=0}^{2(n-1)} c_k h^k, \tag{13}$$

où

$$c_k = \sum_{i+j=k} a_i b_j.$$

La conversion de ce polynôme en un nombre exprimé en base h est aisée, il suffit d'abord de distribuer les retenues en ajoutant $\lfloor c_{k-1}/h \rfloor$ à c_k et ensuite de poser

$$\tilde{c}_k = c_k \bmod h.$$

Les $2n$ entiers $\tilde{c}_k$ ainsi formés sont inférieurs à h ; ils sont donc des chiffres en base h. La complexité de cet algorithme, qui n'est autre chose que celui que nous appliquons pour multiplier *à la main* des nombres en base 10 est de complexité $\mathcal{O}\left(n^2\right)$, à la différence près qu'à la main nous distribuons les retenues au fur et à mesure qu'elles se présentent. Grâce à un ingénieux artifice de calcul, nous allons utiliser une FFT pour réduire cette complexité de façon significative. Nous allons calculer le produit $d \equiv ab \bmod (h^n + 1)$. Pour avoir la relation $c = d$, il suffit de prendre n deux fois trop grand, c'est à dire que si a et b sont des nombres de N chiffres, on pose $n = 2N$, ainsi $a < h^N$ et $b < h^N \Longrightarrow c = ab < h^{2N} = h^n$, d'où le calcul $d \equiv ab$ est en fait exact. Puisque $h^n \equiv -1 \bmod h^n + 1$, l'équation (13) devient

$$d = \sum_{k=0}^{n-1} c_k h^k + \sum_{k=n}^{2n-1} c_k h^n h^{k-n} \equiv \sum_{k=0}^{n-1} c_k h^k - \sum_{k=n}^{2n-1} c_k h^{k-n} = \sum_{k=0}^{n-1} d_k h^k,$$

où

$$\begin{aligned} d_k &= c_k - c_{k+n} \\ &= \sum_{i+j=k} a_i b_j - \sum_{i+j=k+n} a_i b_j \\ &= \sum_{\nu=0}^{k} a_\nu b_{k-\nu} - \sum_{\nu=k+1}^{n-1} a_\nu b_{k+n-\nu}. \end{aligned}$$

Nous désignerons cette opération sous la forme concise suivante :

$$\vec{d} = \vec{a} \otimes \vec{b}.$$

Le lemme qui suit est en fait une méthode pour calculer de façon efficace l'expression $\vec{d} = \vec{a} \otimes \vec{b}$ en utilisant des transformées de Fourier discrètes :

Lemme 4.1. *Soit ω_n une n-ième racine de l'unité, et posons*

$$\vec{\Omega} = (1, \omega_{2n}, \omega_{2n}^2, \ldots, \omega_{2n}^{\frac{n}{2}-1}).$$

Alors

$$\left(\mathcal{F}_n^{-1}(\vec{\Omega}\cdot\vec{a})\right)\left(\mathcal{F}_n^{-1}(\vec{\Omega}\cdot\vec{b})\right)=\mathcal{F}_n^{-1}(\vec{\Omega}\cdot(\vec{a}\otimes\vec{b})),$$

où $\cdot$ *représente le produit composante par composante.*

Démonstration:

La j-ième composante du membre de gauche est

$$\begin{aligned}
&\left(\sum_{k_1=0}^{n-1}\omega_n^{jk_1}\Omega_{k_1}a_{k_1}\right)\left(\sum_{k_2=0}^{n-1}\omega_n^{jk_2}\Omega_{k_2}b_{k_2}\right)\\
&= \sum_{k_1=0}^{n-1}\sum_{k_2=0}^{n-1}\omega_n^{j(k_1+k_2)}\omega_{2n}^{k_1+k_2}a_{k_1}b_{k_2}\\
&= \sum_{k=0}^{n-1}\left(\sum_{k_1+k_2=k}\omega_n^{jk}\omega_{2n}^{k}a_{k_1}b_{k_2}+\sum_{k_1+k_2=k+n}\omega_n^{j(k+n)}\omega_{2n}^{k+n}a_{k_1}b_{k_2}\right)\\
&= \sum_{k=0}^{n-1}\left(\sum_{k_1+k_2=k}\omega_n^{jk}\Omega_k a_{k_1}b_{k_2}-\sum_{k_1+k_2=k+n}\omega_n^{jk}\Omega_k a_{k_1}b_{k_2}\right)\\
&= \sum_{k=0}^{n-1}\omega_n^{jk}\Omega_k d_k,
\end{aligned}$$

qui est la j-ième composante du membre de droite. □

Ainsi, l'opération $\vec{d}=\vec{a}\otimes\vec{b}$ peut être calculée à l'aide de quatre produits composante par composante et de trois FFT. En supposant que n est une puissance de 2, la complexité de cette opération est de $4\mathcal{O}(n)+3\mathcal{O}(n\log_2 n)=\mathcal{O}(n\log_2 n)$. Bien sûr, n doit être assez grand pour réaliser une économie de temps, chaque opération de la méthode classique étant très rapide à effectuer. De plus, le nombre réel de multiplications pour la méthode classique est approximativement N^2 alors que dans la méthode utilisant la FFT, ce nombre est environ de $14N+6N\log_2 N$. Ainsi, même si a et b sont des nombres de 16 chiffres en base 2^{32}, c'est-à-dire des nombres de l'ordre de 10^{150}, la méthode classique sera nettement plus rapide.

A Programmation d'une FFT

Nous allons maintenant nous intéresser à la programmation d'une routine permettant le calcul d'une FFT comme décrit à la section 3. Les fragments de programme seront donnés en pseudo-code afin qu'ils puissent être transposés dans n'importe quel langage. Les temps d'exécution ont été obtenus sur un Pentium MMX 200 Mhz avec 64 Mo de RAM, les programmes ayant été compilés à l'aide de Microsoft Visual C++ 5.0, en utilisant l'objet `complex<double>` de la Standard Template Library. Le vecteur $\vec{x}$ devant subir une transformée de Fourier est constitué de nombres complexes choisis au hasard.

Commençons d'abord par une version naïve du calcul d'une DFT à l'aide d'une multiplication matricielle :

- `De` $\vec{x}$ `et` n `on calcule`
  - $\omega \leftarrow e^{\frac{2\pi i}{n}}$
  - `Pour` $j = 0, 1, \ldots, n-1$ `faire`
    - $y_j \leftarrow 0$
    - `Pour` $k = 0, 1, \ldots, n-1$ `faire`
      - $y_j \leftarrow y_j + \omega^{-jk} x_k$
    - $y_j \leftarrow y_j / n$
- `Retourner` $\vec{y}$.

Temps d'exécution : $5,1$ secondes pour $n = 1024 = 2^{10}$. Cet algorithme est vraiment lent, entre autres parce qu'à chaque itération nous élevons ω à une certaine puissance, une opération très coûteuse. Remplaçons-le donc par la version suivante, qui calcule une fois pour toute les racines de l'unité et les sauvegarde dans un tableau :

- `De` $\vec{x}$ `et` n `on calcule`
  - $\theta = -2\pi/n$
  - `Pour` $s = 0, 1, \ldots, n-1$ `faire`
    - $\omega_s \leftarrow \cos(s\theta) + i \sin(s\theta)$
  - `Pour` $j = 0, 1, \ldots, n-1$ `faire`
    - $y_j \leftarrow 0$
    - `Pour` $k = 0, 1, \ldots, n-1$ `faire`
      - $y_j \leftarrow y_j + \omega_{jk \bmod n} x_k$
    - $y_j \leftarrow y_j / n$
- `Retourner` $\vec{y}$.

Cette deuxième version nécessite cependant n unités de mémoire pour sauvegarder le vecteur $\vec{\omega}$. Temps d'exécution : $0,7$ secondes. Par ailleurs, en supposant que n est toujours une puissance de 2, le calcul de $jk \bmod n$ peut être effectué très rapidement, ce qui fait baisser le temps d'exécution à $0,5$ secondes.

Passons maintenant à la FFT. Nous avons besoin d'une fonction σ qui inverse les bits d'un entier α de m bits, afin de réordonner les éléments du vecteur x :

- `De` α `et` m `on calcule`
  - $\beta \leftarrow 0$
  - `Faire` m `fois :`
    - $\beta \leftarrow 2\beta$

        - $\beta \leftarrow \beta + (\alpha \bmod 2)$
        - $\alpha \leftarrow \alpha/2$
- `Retourner` β.

Cette procédure pourrait bien sûr être accélérée en utilisant des tables pour inverser plus d'un bits à la fois, mais ce n'est pas ce genre d'optimisation sur lequel porte ce document. La première étape de la transformée consiste justement à réordonner les éléments du vecteur $\vec{x}$ afin de former les n vecteurs $\vec{y}^{[\alpha_1\alpha_2\cdots\alpha_m]}$ de taille 1. Ces vecteurs seront tous localisés les uns à la suite des autres dans un tableau de taille n, que nous désignons par $\vec{w}$. Tout en réordonnant les éléments, nous en profiterons pour les diviser par n :

- `De` $\vec{x}$ `et` m `on calcule`
    - $n \leftarrow 2^m$
    - `Pour` $\alpha = 0, 1, \ldots, n-1$ `faire`
        - $\beta \leftarrow \sigma(\alpha, m)$
        - `Si` $\alpha \leq \beta$ `alors`
            - $u \leftarrow x_\alpha/n$
            - $v \leftarrow x_\beta/n$
            - $w_\alpha \leftarrow v$
            - $w_\beta \leftarrow u$
- `Retourner` $\vec{w}$.

Remarquez que la façon dont cette fonction, que nous désignerons par Φ, est construite nous permet de placer les vecteurs $\vec{x}$ et $\vec{w}$ au même endroit en mémoire. L'algorithme principal est constitué de m étapes, chacune d'elle consistant à calculer les n/s transformées de Fourier d'ordre s à partir des $2n/s$ vecteurs de longueur $s/2$, pour $s = 2^1, 2^2, 2^3, \ldots, 2^m = n$. Donnons le programme, nous l'expliquerons après :

- `De` $\vec{x}$ `et` m `on calcule`
    - $n \leftarrow 2^m$
    - $\vec{w} \leftarrow \Phi(\vec{x}, m)$
    - `Pour` $s = 2, 4, 8, \ldots, n$ `faire`
        - $\theta \leftarrow -2\pi/s$
        - `Pour` $j = 0, 1, \ldots, s/2 - 1$ `faire`
            - $\omega \leftarrow \cos(j\theta) + i\sin(j\theta)$
            - `Pour` $\nu = 0, 1, \ldots, n/s - 1$ `faire`
                - $\vec{y} = \vec{w}_{\nu s}$
                - $u \leftarrow y_j$
                - $v \leftarrow \omega y_{j+s/2}$
                - $y_j \leftarrow u + v$

- $y_{j+s/2} \leftarrow u - v$

  - $\vec{y} \leftarrow \vec{w}$

- `Retourner` $\vec{y}$.

Fixons une étape s. Nous devons calculer n/s vecteurs de longueur s. Nous calculons la j-ième et la $(j + s/2)$-ième composante de ces n/s vecteurs à l'aide des équations (9). Remarquez que les boucles pour j et ν peuvent être interchangées : on peut soit calculer toutes les composantes d'un vecteur avant de passer au prochain ou soit, comme dans le programme ci-haut, calculer deux composantes pour tous les vecteurs avant de passer à la prochaine paire de composantes. La ligne $\vec{y} = \vec{w}_{\nu s}$ signifie seulement que le ν-ième vecteur $\vec{y}^{[\nu]}$ de longueur s commence dans le tableau $\vec{w}$ à la position νs. En pratique, cette ligne correspond à l'ajustement d'un pointeur. Les quatre lignes de codes à l'intérieur des trois boucles représentent les formules (9). Remarquez que grâce à ces deux formules, il est possible de faire toutes les opérations en un seul tableau de taille n dans lequel seront sauvegardés successivement les vecteurs $\vec{x}$, $\vec{w}$ et $\vec{y}$. Bien qu'à première vue complexe, cet algorithme est la fidèle reproduction du schéma de la section 3 pour $n = 8$. Le lecteur est invité à mettre en relation ce schéma et le programme ci-haut afin de bien comprendre ce dernier. Mais qu'en est-il du temps d'exécution ? Formidable : 4,0 secondes pour $n = 262\,144 = 2^{18}$. Les chiffres parlent d'eux-mêmes, il est plus rapide d'effectuer une FFT d'ordre 2^{18} qu'une DFT d'ordre 2^{12}, et nous n'avons pas encore optimisé notre FFT ! Notre version améliorée de la DFT aurait pris plus de 9 heures pour $n = 2^{18}$.

References

[1] E. Oran Brigham. *The Fast Fourier Transform and its Applications*, chapter 8, The Fast Fourier Transform (FFT), pages 131–166. Signal Processing Series. Prentice Hall, 1988.

[2] J. W. Cooley and J.W. Tukey. An algorithm for machine calculation of complex fourier series. *Math. Computation*, 19:297–301, April 1965.

[3] Robert Dautray and Jacques-Louis Lions. *Mathematical Analysis and Numerical Methods for Science and Technology*, volume 2, chapter 3.B, Discrete Fourier Transforms and Fast Fourier Transforms, pages 59–91. Springer-Verlag, 1984 (Translated from French, 1988).

[4] Arieh Iserles. *A First Course in the Numerical Analysis of Differential Equations*, chapter 12.2, The Fast Fourier Transform, pages 249–256. Cambridge Texts in Applied Mathematics. Cambridge, 1996 (Second Printing 1998).

[5] Abdul J. Jerri. *Integral and Discrete Transforms with Applications and Error Analysis*, chapter 4.1, Discrete Fourier Transform, pages 510–645. Number 162 in Pure and Applied Mathematics. Marcel Dekker Inc., 1992.

[6] F. Thomson Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, chapter 3.7, The Fast Fourier Transform, pages 711–729. Morgan Kaufmann Publishers, 1992.

[7] Walter Rudin. *Real & Complex Analysis*, chapter 4, Elementary Hilbert Space Theory, pages 76–94. Series in Higher Mathematics. McGraw-Hill, 1966 (Third Edition 1987).

[8] Murray R. Spiegel. *Mathematical Handbook of Formulas and Tables*, chapter 23, Fourier Series, pages 131–135. Schaum's Outline Series. McGraw Hill, 1968.

La notation utilisée dans ce document s'inspire de ISERLES [4], qui fut, en raison de sa clarté, ma principale référence pour les sections 2 et 3. Mentionnons cependant que l'auteur commet une bourde lors de la dérivation de la formule (5). Les lemmes 2.2 et 2.3 sont tirés de DAUTRAY & LIONS [3]. Ces deux auteurs présentent beaucoup de résultats (avancés) de façon extrêmement concise. Les deux lemmes mentionnés ci-haut n'occupent qu'un maigre espace de cinq lignes dans le texte! Le schéma décrivant le calcul d'une FFT est une adaptation de ceux que l'on trouve dans BRIGHAM [1]. Ce livre n'est pas idéal mais il présente de nombreux exemples qui aident à comprendre des explications difficiles à suivre. Pour ce qui est de l'introduction, l'exemple est tiré de SPIEGEL [8], tandis que les notions plus théoriques d'analayse sont tirées de l'excellent livre de RUDIN [7]. L'application de la FFT à la multiplication de deux entiers est une simplification de celui de LEIGHTON [6]. Finalement, le programme de la section A est une contribution personnelle.

L'article de COOLEY-TUCKEY [2] n'est donné qu'à titre de référence. Le livre de JERRI [5] contient beaucoup de résultats intéressants à propos des séries de Fourier et de la transformée de Fourier mais malheureusement l'auteur ne cesse d'entrer dans des discussions interminables et le fil conducteur est très difficile à suivre. De plus, le livre est truffé de coquilles, ce qui est loin d'en faciliter la lecture.