

INFO335
Computación de Alto Rendimiento
Magíster en Informática
Universidad Austral de Chile

Cristóbal A. Navarro


11 de agosto de 2020

ÍNDICE GENERAL

| | | | |
|---|----------|---|----------|
| 1. Fast Fourier Transform | 1 | Discrete Fourier Transform | 2 |
| Introducción | 1 | Definición de la DFT | 2 |
| Breve Historia | 1 | Algoritmo Básico DFT | 3 |
| Estudio de Fourier (1822) | 1 | Algoritmo FFT | 3 |
| Contribución de Gauss (1805) | 1 | Cooley-Tuckey: versión Radix-2 | 3 |
| FFT: Cooley and Tuckey (1965) | 1 | Entendiendo Periodicidad y Simetría | 4 |
| Otros Trabajos y Actualidad (1965-2020) | 1 | FFT: Algoritmo Radix-2 Recursivo | 5 |
| Fourier Transform | 2 | FFT: Algoritmo Radix-2 Iterativo | 5 |
| Definición de la FT | 2 | 2. Colors | 6 |
| | | Themed Colors | 6 |

CAPÍTULO 1: FAST FOURIER TRANSFORM

INTRODUCCIÓN

 A transformada de Fourier (Fourier Transform o FT) es una transformación matemática que descompone una función, típicamente en el tiempo (como una señal), en las frecuencias que la constituyen. Existen diversos algoritmos para realizar esta transformación, y el *Fast Fourier Transform* es actualmente el más eficiente para procesar señales discretas, con un tiempo computacional de $\mathcal{O}(n \log n)$ con n la cantidad de muestras de la señal, o simplemente el largo de la señal. Desde que FFT se implementó en los computadores, la FT se ha vuelto una herramienta de primera importancia, siendo utilizada en diversas áreas de la ciencia y la ingeniería, tales como procesamiento de audio, astroinformática, ingeniería, física, geología, videojuegos, procesamiento de imágenes, compresión de información, entre muchas otras. En general, la FT puede ser de gran utilidad en cualquier ámbito donde exista una secuencia de datos en el tiempo.

BREVE HISTORIA

Debido al nombre de la transformada, *Fourier Transform*, se suele dar el crédito completo al matemático Joseph Fourier por la relación encontrada. Sin embargo, a lo largo de los años, se ha descubierto que Gauss realizó contribuciones importantes en la formulación de un algoritmo eficiente para calcular la FT. Es más, los primeros indicios de representar funciones con expresiones trigonométricas data de los años 1700s con la participación de matemáticos como Euler, Clairaut, Lagrange y Bernoulli.

ESTUDIO DE FOURIER (1822)

Joseph Fourier, un matemático Francés (1768-1830), mostró en 1807 que algunas funciones podían ser representadas como una suma infinita de armónicos. Su trabajo no fue publicado hasta 1822, ya que no fue bien recibido y rechazaron su publicación en la *Memoirs of the Academy*. Hay registros de que uno de sus primeros manuscritos en el tema data de los años 1804-1805 e incluye investigación que pudo comenzar en 1802 [10].

CONTRIBUCIÓN DE GAUSS (1805)

De forma paralela a Fourier, Carl Friedrich Gauss (1777-1855) trabajó propuso alrededor del año 1805 en técnicas para lograr una interpolación trigonométrica de funciones periódicas. La expresión propuesta por Gauss es la primera evidencia de la idea de un algoritmo eficiente para calcular la *Discrete Fourier Transform* (DFT) [2].

FFT: COOLEY AND TUCKEY (1965)

En 1965, Cooley y Tuckey publicaron una técnica eficiente [6], de costo $\mathcal{O}(n \log n)$, para calcular la DFT, que hoy en día es conocida como la *Fast Fourier Transform* [13].

OTROS TRABAJOS Y ACTUALIDAD (1965-2020)

Un trabajo importante en la historia de FT y sus algoritmos es el Danielson-Lanczos [7] en 1942, que establece las bases para el algoritmo de Cooley-Tuckey. El trabajo de Danielson-Lanczos no es el único caso, existen muchos otros que han contribuido de alguna u otra forma a FT y los algoritmos involucrados.

La DFT se puede extender a múltiples dimensiones sin mayor dificultad y en la actualidad se siguen realizando contribuciones a FFT. Es más, existe una pregunta abierta en ciencias de la computación, y es si acaso existe o no una cota $\Omega(n \log n)$ para DFT. De existir, entonces el problema ya ha sido resuelto computacionalmente. De lo contrario, existiría la posibilidad de inventar un algoritmo mejor que el FFT. Por el momento no se ha podido encontrar respuesta a la pregunta.

Por un lado, FFT tiene versiones mejoradas para casos particulares, como lo es el Hexagonal FFT [11], el algoritmo de Rader [14], el algoritmo de Bruun [3]. En las últimas décadas, se ha propuesto formas eficientes para calcular la FFT en paralelo [9, 5] para aprovechar el rendimiento acelerado de las CPUs [8] modernas como también de GPUs modernas [12, 4]. Recientes investigaciones ya preparan las adaptaciones necesarias de FFT para lograr rendimiento en el orden *Exascale* [1].

A pesar de que los fundamentos de la FT, la DFT y los algoritmos fueron establecidos hace muchos años, aun existe interés científico en encontrar mejores algoritmos para el calculo de la DFT, que puedan reducir el numero de operaciones a un mínimo, o paralelizar las operaciones de forma eficiente en consideración de la jerarquía de memoria de las CPUs, GPUs y otros procesadores de la actualidad. Además, aún existe la pregunta abierta de si acaso la complejidad de la DFT necesita al menos $\Omega(n \log n)$ operaciones o no.

FOURIER TRANSFORM

La transformada de Fourier (FT) es capaz de descomponer una onda complicada en una secuencia de ondas mas simples y elementales. La idea es analoga a como un acorde de musica de descompone en las distintas notas que lo componen.

DEFINICIÓN DE LA FT

Sea $f : \mathbb{R} \mapsto \mathbb{C}$ una función integrable, entonces la transformada de Fourier \hat{f} se define como:

$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i \xi x} dx \quad (1.1)$$

Típicamente $f(x)$ representa una cantidad en el dominio del tiempo x (segundos), mientras que $f(\xi)$ se convierte en una cantidad para el dominio de las frecuencias ξ (hertz). Si consideramos la formula de Euler:

$$e^{i\varphi} = \cos(\varphi) + i \sin(\varphi) \quad (1.2)$$

Entonces podemos escribir $\hat{f}(\xi)$ como:

$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(x) [\cos(-2\pi \xi x) + i \sin(-2\pi \xi x)] dx \quad (1.3)$$

Si las condiciones analíticas lo permiten, es posible recuperar $f(x)$ de $\hat{f}(\xi)$ mediante la transformada inversa:

$$f(x) = \int_{-\infty}^{\infty} \hat{f}(\xi) e^{2\pi i \xi x} d\xi \quad (1.4)$$

Hasta ahora las ecuaciones introducidas han asumido un espacio continuo en el dominio y recorrido de las funciones. En la práctica es bastante común trabajar en un espacio discreto ya sea por la naturaleza del dominio (ejemplo, un dominio virtual puede preferir enteros) o bien porque en la práctica muestreamos datos del entorno para ingresarlos al computador como una serie discreta en el tiempo. Para estos casos podemos usar *Discrete Fourier Transform* (DFT).

DISCRETE FOURIER TRANSFORM

Cuando una señal es de naturaleza discreta, o proviene de un muestreo finito de una fuente continua, se usa la transformada de fourier discreta o *discrete Fourier transform* (DFT) en Inglés. Para el ámbito computacional y experimental, la DFT es una herramienta de mucha utilidad pues permite la realizacion de algoritmos que dependen de utilizar el espectro de frecuencias de la señal.

DEFINICIÓN DE LA DFT

Discrete Fourier Transform (DFT) transforma una secuencia de n números complejos x_0, x_1, \dots, x_{n-1} que representa cantidades en el tiempo, en otra secuencia de números complejos X_0, X_1, \dots, X_{n-1} que representa cantidades en ciertas frecuencias. La transformación, $X_k = \mathcal{F}_k$, se define como

$$X_k = \mathcal{F}_k = \sum_{r=0}^{n-1} x_r e^{-\frac{i2\pi}{n} kr} \quad (1.5)$$

Utilizando la formula de Euler, tenemos que

$$X_k = \sum_{r=0}^{n-1} x_r \left[\cos\left(\frac{2\pi}{n} kr\right) - i \sin\left(\frac{2\pi}{n} kr\right) \right] \quad (1.6)$$

El valor de k funciona en el rango $[0..n-1]$ y estos valores no son directamente valores de frecuencia, sino que índices a los eventuales valores de frecuencia. Se asume que los valores x_0, x_1, \dots, x_{n-1} vienen uniformemente espaciados en intervalos de tiempo Δt , o equivalentemente con una frecuencia de muestreo de $f_s = 1/\Delta t$. Entonces, dado un X_k , su frecuencia asociada f_k viene dada por

$$f_k = \frac{k}{n} f_s \quad (1.7)$$

Los valores f_k son importantes para graficar la intensidad de la señal en relación al espectro de frecuencias en el cual se desenvuelve.

ALGORITMO BÁSICO DFT

Dada una señal x_0, x_1, \dots, x_{n-1} , un algoritmo básico de DFT puede calcular la expresión de la Eq. 1.5 o 1.6 para cada X_0, X_1, \dots, X_{n-1} tal como se muestra en el Algoritmo 1. Si bien el algoritmo básico es

Algorithm 1: Algoritmo básico DFT

```

Input:  $x[n] : \{x_0, x_1, \dots, x_{n-1}\}$ 
Result:  $X[n] : \{X_0, X_1, \dots, X_{n-1}\}$ 
 $X[] \leftarrow \text{new complex}[n];$ 
for  $k \in 0..n-1$  do
     $s \leftarrow \{0, 0\};$ 
     $c \leftarrow 2\pi k/n;$ 
    for  $r \in 0..n-1$  do
         $s.\text{real} \leftarrow s.\text{real} + x[r] \cdot \cos(c \cdot r);$ 
         $s.\text{imag} \leftarrow s.\text{imag} - x[r] \cdot \sin(c \cdot r);$ 
    end
     $X[k] \leftarrow s;$ 
end
return  $X;$ 

```

sencillo y puede calcular correctamente la DFT para cualquier tamaño n de señal, este tiene un costo computacional de $\Theta(n^2)$. Para valores pequeños de n el costo computacional es abordable, sin embargo a medida que crece n , el comportamiento cuadrático se hace notar rápidamente.

Si se analiza mejor la Eq. 1.5, es posible ver que parte del trabajo realizado para calcular un termino X_k de la DFT puede servir para el cálculo de otro X_j . Al usar esta simetría en su completitud, se puede llegar a la transformada rápida de Fourier, o mas conocida como FFT por fast Fourier transform.

ALGORITMO FFT

Fast Fourier Transform, o FFT, se refiere a el conjunto de algoritmos que logran calcular la DFT en tiempo $\mathcal{O}(n \log n)$. En esta sección, revisaremos el algoritmo de Cooley-Tuckey propuesto en 1965. En particular, nos referiremos a la versión *Radix-2*, un caso particular de Cooley-Tuckey que particiona recursivamente el problema en dos partes iguales.

COOLEY-TUCKEY: VERSIÓN RADIX-2

La principal idea de la variante Radix-2 es dividir el cómputo suma de la DFT de las muestras pares $\{x_0, x_2, \dots, x_{n-2}\}$ con la DFT de las muestras impares $\{x_1, x_3, \dots, x_{n-1}\}$. La ecuación Eq. 1.8 muestra la idea, donde el índice r ahora es reemplazado por una sucesión par o impar según corresponda.

$$X_k = \sum_{m=0}^{n/2-1} x_{2m} e^{-\frac{2\pi i}{n}(2m)k} + \sum_{m=0}^{n/2-1} x_{2m+1} e^{-\frac{2\pi i}{n}(2m+1)k} \quad (1.8)$$

Es posible factorizar el término $e^{-\frac{2\pi i}{n}k}$ de la segunda sumatoria, debido al $+1$ del exponente. Si además ubicamos el factor 2 que acompaña a m en cada sumatoria como divisor de n , quedan dos DFTs de tamaño $n/2$, una para pares E_k y otra para impares O_k ,

$$X_k = \sum_{m=0}^{n/2-1} x_{2m} e^{-\frac{2\pi i}{n/2}mk} + e^{-\frac{2\pi i}{n}k} \sum_{m=0}^{n/2-1} x_{2m+1} e^{-\frac{2\pi i}{n/2}mk} \quad (1.9)$$

$$X_k = E_k + e^{-\frac{2\pi i}{n}k} O_k \quad (1.10)$$

Dado que el termino $e^{-\frac{2\pi i}{n}}$ aparece con frecuencia, usaremos la notación

$$W_n = e^{-\frac{2\pi i}{n}} \quad (1.11)$$

De esta forma podemos expresar X_k como

$$X_k = E_k + W_n^k O_k \quad (1.12)$$

con $W_n^k = (e^{-\frac{2\pi i}{n}})^k = e^{-\frac{2\pi i}{n}k}$. Es importante mencionar en este punto que Tanto E_k como O_k pueden volver a ser tratados como problemas separables, esta vez en sub-problemas de $n/4$ cada uno. Siguiendo esta lógica, se puede construir un algoritmo recursivo que subdivide en mitades (par e impar) el problema. Si bien la subdivisión recursiva es una componente clave de la FFT, por si sola no es suficiente para generar un algoritmo de costo $\mathcal{O}(n \log n)$, ya que habria que hacerle el mismo tratamiento a todos los X_k por igual. Sin embargo, al combinar la recursión con las propiedades de periodicidad y simetría, se abre la posibilidad de encontrar el algoritmo eficiente FFT Radix-2.

ENTENDIENDO PERIODICIDAD Y SIMETRÍA

Existen dos propiedades críticas que deben aprovecharse para generar un algoritmo FFT de costo $\mathcal{O}(n \log n)$, i) periodicidad y ii) simetría.

PERIODICIDAD

Para un input de tamaño $n/2$, se tiene que $X_{k+\frac{n}{2}}$ se puede calcular con los mismos términos E_k e O_k usados para calcular X_k . Para el caso par, tenemos

$$E_{k+\frac{n}{2}} = \sum_{m=0}^{n/2-1} x_{2m} W_{n/2}^{(k+n/2)m} \quad (1.13)$$

$$= \sum_{m=0}^{n/2-1} x_{2m} W_{n/2}^{km} W_{n/2}^{(n/2)m} \quad (1.14)$$

en el cual $W_{n/2}^{(n/2)m} = e^{-2\pi i m}$ y por formula de Euler $W_{n/2}^{(n/2)m} = e^{-2\pi i m} = \cos(-2\pi m) + i \sin(-2\pi m) = 1$. Así, **emerge la primera propiedad de periodicidad**, que establece

$$E_{k+\frac{n}{2}} = E_k \quad (1.15)$$

El proceso es el mismo para los impares, i.e., $O_{k+\frac{n}{2}} = O_k$.

SIMETRÍA DE MEDIA DISTANCIA

Para la expresión completa, tenemos entonces que

$$X_{k+\frac{n}{2}} = E_{k+\frac{n}{2}} + W_n^{k+\frac{n}{2}} O_{k+\frac{n}{2}} \quad (1.16)$$

$$= E_k + W_n^k e^{-\pi i} O_k \quad (1.17)$$

$$= E_k - W_n^k O_k \quad (1.18)$$

donde por formula de Euler, $e^{-\pi i} = \cos(-\pi) + i \sin(-\pi) = -1$. De la expresión, **emerge la segunda propiedad de simetría de media distancia**, la cual establece que

$$W_n^{k+n/2} = -W_n^k. \quad (1.19)$$

Finalmente, como resumen de las simetrías, se tiene que

$$X_k = E_k + W_{n/2}^k O_k \quad (1.20)$$

$$X_{k+\frac{n}{2}} = E_k - W_{n/2}^k O_k \quad (1.21)$$

para $k \in [0..(n-1)]$.

Combinando las dos propiedades junto con la estructura recursiva, es posible formular un algoritmo FFT eficiente de costo $\mathcal{O}(n \log n)$. Existen dos formas de implementar la idea, una es usando un algoritmo recursivo mientras que la segunda idea es utilizar un algoritmo iterativo. La diferencia entre cada forma es que la primera (recursivo) aplica un esquema *depth-first-search* (DFS) mientras que la segunda *breadth-first-search* (BFS) en el arbol de recursion.

| | |
|--|---|
| <pre> $X_0, \dots, X_{N-1} \leftarrow \text{ditfft2}(x, N, s):$ if $N = 1$ then $X_0 \leftarrow x_0$ else $X_0, \dots, X_{N/2-1} \leftarrow \text{ditfft2}(x, N/2, 2s)$ $X_{N/2}, \dots, X_{N-1} \leftarrow \text{ditfft2}(x+s, N/2, 2s)$ for $k = 0$ to $N/2-1$ do $t \leftarrow X_k$ $X_k \leftarrow t + \exp(-2\pi i k/N) X_{k+N/2}$ $X_{k+N/2} \leftarrow t - \exp(-2\pi i k/N) X_{k+N/2}$ end for end if </pre> | <p><i>DFT of $(x_0, x_s, x_{2s}, \dots, x_{(N-1)s})$:</i></p> <p><i>trivial size-1 DFT base case</i></p> <p><i>DFT of $(x_0, x_{2s}, x_{4s}, \dots)$</i></p> <p><i>DFT of $(x_s, x_{s+2s}, x_{s+4s}, \dots)$</i></p> <p><i>combine DFTs of two halves into full DFT:</i></p> |
|--|---|

Figura 1.1: Algoritmo recursivo para el Radix-2 FFT.

FFT: ALGORITMO RADIX-2 RECURSIVO

La idea principal del Radix-2 recursivo es ilustrada en la Figura 1.1.

FFT: ALGORITMO RADIX-2 ITERATIVO

La idea principal del Radix-2 iterativo es ilustrada en la Figura 1.1.

```

algorithm iterative-fft is
    input: Array  $a$  of  $n$  complex values where  $n$  is a power of 2.
    output: Array  $A$  the DFT of  $a$ .

    bit-reverse-copy( $a, A$ )
     $n \leftarrow a.\text{length}$ 
    for  $s = 1$  to  $\log(n)$  do
         $m \leftarrow 2^s$ 
         $\omega_m \leftarrow \exp(-2\pi i/m)$ 
        for  $k = 0$  to  $n-1$  by  $m$  do
             $\omega \leftarrow 1$ 
            for  $j = 0$  to  $m/2 - 1$  do
                 $t \leftarrow \omega A[k + j + m/2]$ 
                 $u \leftarrow A[k + j]$ 
                 $A[k + j] \leftarrow u + t$ 
                 $A[k + j + m/2] \leftarrow u - t$ 
             $\omega \leftarrow \omega \omega_m$ 
        end for
    end for

    return  $A$ 

```

Figura 1.2: Algoritmo iterativo para el Radix-2 FFT.

Notar que esta versión utiliza un proceso de bit-reverse para ubicar de forma correcta los resultados.

CAPÍTULO 2: COLORS

This package provides several global color variables to style `DndComment`, `DndReadAloud`, `DndSidebar`, and `DndTable` environments.

BOX COLORS

| Color | Description |
|-----------------------------|---|
| <code>commentcolor</code> | <code>DndComment</code> background |
| <code>readaloudcolor</code> | <code>DndReadAloud</code> background |
| <code>sidebarcolor</code> | <code>DndSidebar</code> background |
| <code>tablecolor</code> | background of even <code>DndTable</code> rows |

They also accept an optional color argument to set the color for a single instance. See Table 2.1 for a list of core book accent colors.

```
\begin{DndTable}[color=PhbLightCyan]{cX}
  \textbf{d8} & \textbf{Item} \\
  1 & Small wooden button \\
  2 & Red feather \\
  3 & Human tooth \\
  4 & Vial of green liquid \\
  6 & Tasty biscuit \\
  7 & Broken axe handle \\
  8 & Tarnished silver locket \\
\end{DndTable}
```

| d8 | Item |
|----|-------------------------|
| 1 | Small wooden button |
| 2 | Red feather |
| 3 | Human tooth |
| 4 | Vial of green liquid |
| 6 | Tasty biscuit |
| 7 | Broken axe handle |
| 8 | Tarnished silver locket |

THEMED COLORS

Use `\DndSetThemeColor[<color>]` to set `commentcolor`, `readaloudcolor`, `sidebarcolor`, and `tablecolor` to a specific color. Calling `\DndSetThemeColor` without an argument sets those colors to the current `themecolor`. In the following example the group limits the change to just a few boxes; after the group finishes, the colors are reverted to what they were before the group started.

```
\begin{group}
\DndSetThemeColor[PhbMauve]
```

COLORS SUPPORTED BY THIS PACKAGE

| Color | Description |
|---|--|
| <code>PhbLightGreen</code> | Light green used in PHB Part 1 (Default) |
| <code>PhbLightCyan</code> | Light cyan used in PHB Part 2 |
| <code>PhbMauve</code> | Pale purple used in PHB Part 3 |
| <code>PhbTan</code> | Light brown used in PHB appendix |
| <code>DmgLavender</code> | Pale purple used in DMG Part 1 |
| <code>DmgCoral</code> | Orange-pink used in DMG Part 2 |
| <code>DmgSlateGray</code> (<code>DmgSlateGrey</code>) | Blue-gray used in PHB Part 3 |
| <code>DmgLilac</code> | Purple-gray used in DMG appendix |


```
\begin{DndComment}{This Comment Is in Mauve}  
  This comment is in the the new color.  
\end{DndComment}  
  
\begin{DndSidebar}{This Sidebar Is Also Mauve}  
  The sidebar is also using the new theme color.  
\end{DndSidebar}  
\endgroup
```

THIS COMMENT IS IN MAUVE

This comment is in the the new color.

THIS SIDEBAR IS ALSO MAUVE

The sidebar is also using the new theme color.

BIBLIOGRAFÍA

- [1] Alan Ayala, Stanimire Tomov, Azzam Haidar, and Jack Dongarra. heffte: Highly efficient fft for exascale. In *International Conference on Computational Science*, pages 262–275. Springer, 2020.
- [2] E Oran Brigham. 6 the discrete fourier transform. *The fast Fourier transform and its applications*, 1988.
- [3] Georg Bruun. z-transform dft filters and fft's. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):56–63, 1978.
- [4] Yifeng Chen, Xiang Cui, and Hong Mei. Large-scale fft on gpu clusters. In *Proceedings of the 24th ACM International Conference on Supercomputing*, pages 315–324, 2010.
- [5] Eleanor Chu and Alan George. *Inside the FFT black box: serial and parallel fast Fourier transform algorithms*. CRC press, 1999.
- [6] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.
- [7] Gordon Charles Danielson and Cornelius Lanczos. Some improvements in practical fourier analysis and their application to x-ray scattering from liquids. *Journal of the Franklin Institute*, 233(5):435–452, 1942.
- [8] Franz Franchetti, Markus Puschel, Yevgen Voronenko, Srinivas Chellappa, and José MF Moura. Discrete fourier transform on multicore. *IEEE Signal Processing Magazine*, 26(6):90–102, 2009.
- [9] Anshul Gupta and Vipin Kumar. The scalability of fft on parallel computers. *IEEE Transactions on Parallel and Distributed Systems*, 4(8):922–932, 1993.
- [10] Michael Heideman, Don Johnson, and Charles Burrus. Gauss and the history of the fast fourier transform. *IEEE ASSP Magazine*, 1(4):14–21, 1984.
- [11] Russell M Mersereau. The processing of hexagonally sampled two-dimensional signals. *Proceedings of the IEEE*, 67(6):930–949, 1979.
- [12] Kenneth Moreland and Edward Angel. The fft on a gpu. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 112–119, 2003.
- [13] Henri J Nussbaumer. The fast fourier transform. In *Fast Fourier Transform and Convolution Algorithms*, pages 80–111. Springer, 1981.
- [14] Charles M Rader. Discrete fourier transforms when the number of data samples is prime. *Proceedings of the IEEE*, 56(6):1107–1108, 1968.