Mathematics that Changed the World from Slow to Fast

Ehssan Khanmohammadi

Department of Mathematics
Franklin & Marshall College

Millersville University

Dec 1, 2016
Suppose $f : \mathbb{R} \to \mathbb{R}$ is \textbf{any} periodic function of period 1. In 1807 Fourier claimed that, for suitable choices of $a_n$ and $b_n$,

$$f(x) = \sum_{n=0}^{\infty} (a_n \cos(2\pi nx) + b_n \sin(2\pi nx)).$$
Or equivalently, $f$ can be written in the form

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{2\pi i nx}.$$
Or equivalently, \( f \) can be written in the form

\[
f(x) = \sum_{n=-\infty}^{\infty} c_n e^{2\pi i nx}.
\]

It was later proved (with the efforts of Dirichlet, Fejér, Carleson, and others) that Fourier was *almost* right!
Or equivalently, \( f \) can be written in the form

\[
f(x) = \sum_{n=-\infty}^{\infty} c_n e^{2\pi i n x}.
\]

It was later proved (with the efforts of Dirichlet, Fejér, Carleson, and others) that Fourier was *almost* right!

Observe that

\[
\int_0^1 e^{2\pi i m x} e^{-2\pi i n x} \, dx = \begin{cases} 
0 & \text{if } m \neq n \\
1 & \text{if } m = n
\end{cases}.
\]

By wishful thinking (i.e., a term-by-term integration) we get

\[
c_k = \int_0^1 f(x) e^{-2\pi i k x} \, dx.
\]
Let \( m \geq 1 \) be an integer. The Riemann sum

\[
C_k = \frac{1}{m} \sum_{j=0}^{m-1} f \left( \frac{j}{m} \right) e^{-2\pi ikj/m}, \quad 0 \leq k \leq m - 1
\]

is an approximation of \( c_k = \int_0^1 f(x)e^{-2\pi ikx} \, dx \).
Discretization of Fourier Series

Let $m \geq 1$ be an integer. The Riemann sum

$$C_k = \frac{1}{m} \sum_{j=0}^{m-1} f \left( \frac{j}{m} \right) e^{-2\pi i k j / m}, \quad 0 \leq k \leq m - 1$$

is an approximation of $c_k = \int_0^1 f(x) e^{-2\pi i k x} \, dx$.

The matrix multiplication $F_{2^n} B = C$ is equivalent to the above equations with $m = 2^n$:

$$
\begin{bmatrix}
  e^{-2\pi i 0 \frac{0}{2^n}} & e^{-2\pi i 0 \frac{1}{2^n}} & \cdots & e^{-2\pi i 0 \frac{2^n-1}{2^n}} \\
  e^{-2\pi i 1 \frac{0}{2^n}} & e^{-2\pi i 1 \frac{1}{2^n}} & \cdots & e^{-2\pi i 1 \frac{2^n-1}{2^n}} \\
  \vdots & \vdots & \ddots & \vdots \\
  e^{-2\pi i (2^n-1) \frac{0}{2^n}} & \cdots & \cdots & e^{-2\pi i (2^n-1) \frac{2^n-1}{2^n}}
\end{bmatrix}
\begin{bmatrix}
  f \left( \frac{0}{2^n} \right) \\
  f \left( \frac{1}{2^n} \right) \\
  \vdots \\
  f \left( \frac{2^n-1}{2^n} \right)
\end{bmatrix}
= 
\begin{bmatrix}
  2^n C_0 \\
  2^n C_1 \\
  \vdots \\
  2^n C_{2^n-1}
\end{bmatrix}
$$

where $F_{2^n}$ is the Fourier matrix and $B$ is the matrix of values of $f(x)$ at the points $\frac{j}{2^n}$ for $j = 0, 1, \ldots, 2^n-1$. 

$C$ is the matrix of approximations of the Fourier coefficients $c_k$. 

The matrix multiplication represents the transformation of the values of $f(x)$ into the Fourier coefficients as approximated by the Riemann sum.
For $x \in \mathbb{C}^m$, define the **discrete Fourier transform** (DFT) of $x$ to be the vector $\mathcal{F}(x) \in \mathbb{C}^m$ whose $k$th entry is

$$
\sum_{j=0}^{m-1} x_j e^{-2\pi ikj/m}, \quad 0 \leq k \leq m - 1.
$$

$\mathcal{F} : \mathbb{C}^m \to \mathbb{C}^m$ is an isomorphism of vector spaces. 

$\mathcal{F}^{-1}$, called the **inverse discrete Fourier transform**, is a function that associates to a vector $y \in \mathbb{C}^m$ the vector $x \in \mathbb{C}^m$ defined by

$$
x_k = \frac{1}{m} \sum_{j=0}^{m-1} y_j e^{2\pi ikj/m}, \quad 0 \leq k \leq m - 1.
$$
Multiplication of Numbers

Consider the two-digit numbers $a_1a_0$ and $b_1b_0$. Then

$$a_1a_0 \times b_1b_0 = (a_110^1 + a_010^0)(b_110^1 + b_010^0)$$

$$= a_1b_110^2 + (a_1b_0 + a_0b_1)10^1 + a_0b_010^0.$$
Consider the two-digit numbers $\overline{a_1a_0}$ and $\overline{b_1b_0}$. Then

$$\overline{a_1a_0} \times \overline{b_1b_0} = (a_110^1 + a_010^0)(b_110^1 + b_010^0) = a_1b_110^2 + (a_1b_0 + a_0b_1)10^1 + a_0b_010^0.$$ 

This motivates the **discrete convolution** of two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{C}^m$:

$$(\mathbf{a} \ast \mathbf{b})_n = \sum_{k+l \equiv n \text{ mod } m} a_kb_l$$
**Lemma**

*DFT turns convolution into multiplication:*

\[
(\mathcal{F}_m(a \ast b))_n = (\mathcal{F}_m(a))_n (\mathcal{F}_m(b))_n, \quad n = 0, \ldots, m - 1.
\]
Lemma

DFT turns convolution into multiplication:

\[
(\mathcal{F}_m(a \ast b))_n = (\mathcal{F}_m(a))_n (\mathcal{F}_m(b))_n, \quad n = 0, \ldots, m - 1.
\]

Proof.

Write \( \omega = e^{-\frac{2\pi i}{m}} \). Then by changing the order of summation,

\[
(\mathcal{F}_m(a \ast b))_n = \sum_k \omega^{nk} \sum_j a_j b_{k-j}
\]

\[
= \sum_j \sum_k \omega^{nj} \omega^{n(k-j)} a_j b_{k-j}
\]

\[
= \sum_j \sum_l \omega^{nj} \omega^{nl} a_j b_l
\]

\[
= \sum_j \omega^{nj} a_j \sum_l \omega^{nl} b_l = (\mathcal{F}_m(a))_n (\mathcal{F}_m(b))_n
\]
Fast Fourier Transform

...discovered by Gauss in 1805!

Figure: Carl Friedrich Gauss (1777–1855)

This algorithm notably reduces the most painful aspects of the numerical computation.  Gauss
...rediscovered by Cooley and Tukey in 1965 as part of von Neumann’s project in 1940s to build an early digital computer at IAS.

- The Cooley–Tukey paper\(^1\) revolutionized the theory of mathematical algorithms. A secondary application at that time was for the acoustic detection of nuclear submarines.
- FFT is used routinely in optics, acoustics, quantum physics, signal processing, and image processing.

\(^1\)“An algorithm for the machine computation of complex Fourier series”, *Math. Comp.* is cited more times than any other math paper!
The key to the FFT is a very clever factorization of matrices:

\[
[F_{2^n}] = \begin{bmatrix}
I_{2^{n-1}} & D_{2^{n-1}} \\
I_{2^{n-1}} & -D_{2^{n-1}}
\end{bmatrix}
\begin{bmatrix}
F_{2^{n-1}} & 0 \\
0 & F_{2^{n-1}}
\end{bmatrix}
[S_{2^n}].
\]

For instance, 

\[
S_{4} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix},
\]

\[
D_{4} = \begin{bmatrix}
\omega_0 & 0 & 0 & 0 \\
0 & \omega_1 & 1 & 0 \\
0 & 0 & \omega_0 & 0 \\
0 & 0 & 0 & \omega_3
\end{bmatrix}
\]

where \(\omega = e^{-\frac{2\pi i}{4}}\).
FFT via the world’s trickiest matrix factorization

The key to the FFT is a very clever factorization of matrices:

\[
[F_{2^n}] = \begin{bmatrix}
I_{2^n-1} & D_{2^n-1} \\
I_{2^n-1} & -D_{2^n-1}
\end{bmatrix}
\begin{bmatrix}
F_{2^{n-1}} & 0 \\
0 & F_{2^{n-1}}
\end{bmatrix} [S_{2^n}]_{\text{shuffle}}.
\]

For instance,

\[
S_4 = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix},
D_4 = \begin{bmatrix}
\omega^0 & 0 & 0 & 0 \\
0 & \omega^1 & 1 & 0 \\
0 & 0 & \omega^2 & 0 \\
0 & 0 & 0 & \omega^3
\end{bmatrix},
\]

where \( \omega = e^{-\frac{2\pi i}{4}} \).
A recursive equation for the number of operations:

\[ Nop(2^n) = 2 \text{Nop}(2^{n-1}) + 2 \cdot 2^n \text{ operations to compute } \begin{bmatrix} I_{2^{n-1}} & D_{2^{n-1}} \\ I_{2^{n-1}} & -D_{2^{n-1}} \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \end{bmatrix} \]

and \( Nop(2^1) = 2 \).
How Fast is the Fast Fourier Transform?

A recursive equation for the number of operations:

\[ \text{Nop}(2^n) = 2 \text{Nop}(2^{n-1}) + 2 \cdot 2^n \]

and \( \text{Nop}(2^1) = 2 \).

It is easy to see inductively that \( \text{Nop}(2^n) = (2n - 1)2^n \).
How Fast is the Fast Fourier Transform?

A recursive equation for the number of operations:

\[
N_{\text{op}}(2^n) = 2 N_{\text{op}}(2^{n-1}) + \begin{cases} 
\text{work done by subcontractor} & \text{operations to compute} \\
2 \cdot 2^n & \begin{bmatrix}
I_{2^{n-1}} & D_{2^{n-1}} \\
I_{2^{n-1}} & -D_{2^{n-1}}
\end{bmatrix}
\begin{bmatrix}
u_0 \\
u_1
\end{bmatrix}
\end{cases}
\]

and \(N_{\text{op}}(2^1) = 2\).

It is easy to see inductively that \(N_{\text{op}}(2^n) = (2n - 1)2^n\).

With an ordinary computer, we could compute \(\pi\) to a billion decimal places, probably in less than a minute. Without the FFT, the same computation would take 10000 years!
Proof by Poem: The RSA Encryption Algorithm

Figure: Rivest, Shamir, and Adleman in 1978

Take two large prime numbers, q and p.
Find the product n, and the totient \( \phi \).
If \( e \) and \( \phi \) have GCD one
and \( d \) is \( e \)’s inverse, then you’re done!
For sending \( m \) raised to the \( e \)
reduced mod \( n \) gives secre-c.  

Daniel Treat, National Security Agency
A good secret code is a function that everyone knows how to compute but only the person who receives the message knows the inverse function.
A good secret code is a function that everyone knows how to compute but only the person who receives the message knows the inverse function.

- Your bank chooses two \textbf{large} prime numbers \( p \) and \( q \), also two numbers \( d \) and \( e \) such that \( de \equiv 1 \mod \phi(n) \).
A good secret code is a function that everyone knows how to compute but only the person who receives the message knows the inverse function.

- Your bank chooses two large prime numbers $p$ and $q$, also two numbers $d$ and $e$ such that $de \equiv 1 \mod \phi(n)$.
- The bank sends you the numbers $n$ and $e$ where $n = pq$ is computed by the FFT.

Exercise: Show that $(m^e)^d \equiv m \mod n$. 

A good secret code is a function that everyone knows how to compute but only the person who receives the message knows the inverse function.

- Your bank chooses two large prime numbers \( p \) and \( q \), also two numbers \( d \) and \( e \) such that \( de \equiv 1 \mod \phi(n) \).
- The bank sends you the numbers \( n \) and \( e \) where \( n = pq \) is computed by the FFT.
- You decompose your message into a sequence of numbers \( m_1, m_2, \ldots \) (all less than \( n \)) and encode your message as numbers \( \tilde{m}_1, \tilde{m}_2, \ldots \) where \( \tilde{m}_i \equiv m_i^e \mod n \).
A good secret code is a function that everyone knows how to compute but only the person who receives the message knows the inverse function.

- Your bank chooses two large prime numbers $p$ and $q$, also two numbers $d$ and $e$ such that $de \equiv 1 \pmod{\phi(n)}$.
- The bank sends you the numbers $n$ and $e$ where $n = pq$ is computed by the **FFT**.
- You decompose your message into a sequence of numbers $m_1, m_2, \ldots$ (all less than $n$) and encode your message as numbers $\tilde{m}_1, \tilde{m}_2, \ldots$ where $\tilde{m}_i \equiv m_i^e \pmod{n}$.
- When it receives the message, the bank computes $\tilde{m}_i^d$. 

**Exercise:** Show that $(m_i^e)^d \equiv m_i \pmod{n}$. 

A good secret code is a function that everyone knows how to compute but only the person who receives the message knows the inverse function.

- Your bank chooses two large prime numbers $p$ and $q$, also two numbers $d$ and $e$ such that $de \equiv 1 \mod \phi(n)$.
- The bank sends you the numbers $n$ and $e$ where $n = pq$ is computed by the FFT.
- You decompose your message into a sequence of numbers $m_1, m_2, \ldots$ (all less than $n$) and encode your message as numbers $\tilde{m}_1, \tilde{m}_2, \ldots$ where $\tilde{m}_i \equiv m_i^e \mod n$.
- When it receives the message, the bank computes $\tilde{m}_i^d$.

**Exercise:** Show that $(m^e)^d \equiv m$ in $\mathbb{Z}/n\mathbb{Z}$. 
Finding $p$ and $q$

Choose an odd number $r$ made up of 200 random digits, and take the first prime number from the list $r, r + 2, r + 4, \ldots$ as $p$.

- How do we find an odd number $r$ with 200 random digits?
Finding $p$ and $q$

Choose an odd number $r$ made up of 200 random digits, and take the first prime number from the list $r, r + 2, r + 4, \ldots$ as $p$.

- How do we find an odd number $r$ with 200 random digits?
- How many elements of the list will we need to consider before we find a prime number?
Finding $p$ and $q$

Choose an odd number $r$ made up of 200 random digits, and take the first prime number from the list $r, r + 2, r + 4, \ldots$ as $p$.

- How do we find an odd number $r$ with 200 random digits?
- How many elements of the list will we need to consider before we find a prime number?
- How do we know whether a number on the list $r, r + 2, \ldots$ is prime?
The Exercise showed that “knowing d” is equivalent to “knowing how to decode.” We would like to know that decoding is equivalent to factoring n. *This will assure us that decoding is as hard as factoring n.*
Why is knowing $d$ and $e$ equivalent to knowing $p$ and $q$?

The Exercise showed that “knowing $d$” is equivalent to “knowing how to decode.” We would like to know that decoding is equivalent to factoring $n$. This will assure us that decoding is as hard as factoring $n$.

Divide the equation $de = k(p - 1)(q - 1) + 1$ by $n$ to find

$$k = \frac{de}{n} + \frac{k(p + q - 1) - 1}{n}$$

This implies that $k = \lfloor de/n \rfloor + 1$. Knowing $k$ we compute $p + q$. Then use $p + q$ and $pq$ to find $p$ and $q$ by solving a quadratic equation.
Further Directions

What about higher dimensions? Many of the results true for one dimension are wrong or unknown in multiple dimensions.
Further Directions

- What about higher dimensions? Many of the results true for one dimension are wrong or unknown in multiple dimensions.
- Don’t expect miracles from Fourier! What are other useful (orthogonal) bases?
Further Directions

- What about higher dimensions? Many of the results true for one dimension are wrong or unknown in multiple dimensions.
- Don’t expect miracles from Fourier! What are other useful (orthogonal) bases? Wavelets are better for images, and Fourier is the right choice for music. Images have sharp edges; music is sinusoidal. FBI uses wavelets to compress fingerprints.
Thank You!

References and suggested reading: