Fourier Analysis

**Fourier theorem.** [Fourier, Dirichlet, Riemann] Any periodic function can be expressed as the sum of a series of sinusoids. 

\[ y(t) = \sum_{k} \frac{1}{N} \sin(kt) \quad N = 100 \]

Euler’s Identity

**Sinusoids.** Sum of sines and cosines.

\[ e^{ix} = \cos x + i \sin x \]

**Euler’s identity**

**Sinusoids.** Sum of complex exponentials.
Time Domain vs. Frequency Domain

**Signal.**  [touch tone button 1]  \[ y(t) = \frac{1}{2} \sin(2\pi \cdot 697 \ t) + \frac{1}{2} \sin(2\pi \cdot 1209 \ t) \]

Time domain.

![Time domain graph](image)

Frequency domain.

![Frequency domain graph](image)

Reference: Cleve Moler, *Numerical Computing with MATLAB*

---

**Fast Fourier Transform**

**FFT.**  Fast way to convert between time-domain and frequency-domain.

**Alternate viewpoint.**  Fast way to multiply and evaluate polynomials.

"If you speed up any nontrivial algorithm by a factor of a million or so the world will beat a path towards finding useful applications for it." — Numerical Recipes

---

**Applications.**

- Optics, acoustics, quantum physics, telecommunications, radar, control systems, signal processing, speech recognition, data compression, image processing, seismology, mass spectrometry
- Digital media.  [DVD, JPEG, MP3, H.264]
- Medical diagnostics.  [MRI, CT, PET scans, ultrasound]
- Numerical solutions to Poisson’s equation.
- Shor’s quantum factoring algorithm.
- ...

"The FFT is one of the truly great computational developments of the 20th century. It has changed the face of science and engineering so much that it is not an exaggeration to say that life as we know it would be very different without the FFT." — Charles van Loan
Fast Fourier Transform: Brief History

**Gauss (1805, 1866).** Analyzed periodic motion of asteroid Ceres.

**Runge-König (1924).** Laid theoretical groundwork.

**Danielson-Lanczos (1942).** Efficient algorithm, x-ray crystallography.

**Cooley-Tukey (1965).** Monitoring nuclear tests in Soviet Union and tracking submarines. Rediscovered and popularized FFT.

Importance not fully realized until advent of digital computers.

---

**A Modest PhD Dissertation Title**

"New Proof of the Theorem That Every Algebraic Rational Integral Function In One Variable can be Resolved into Real Factors of the First or the Second Degree."

— PhD dissertation, 1799 the University of Helmstedt

---

Polynomials: Coefficient Representation

**Polynomial.** [coefficient representation]

\[ A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1} \]
\[ B(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_{n-1} x^{n-1} \]

**Add.** \(O(n)\) arithmetic operations.

\[ A(x) + B(x) = (a_0 + b_0) + (a_1 + b_1) x + \cdots + (a_{n-1} + b_{n-1}) x^{n-1} \]

**Evaluate.** \(O(n)\) using Horner’s method.

\[ A(x) = a_0 + (a_1 + x) (a_2 + x) \cdots (a_{n-1} + x(a_{n-1} + x)) \cdots ) \]

**Multiply (convolve).** \(O(n^2)\) using brute force.

\[ A(x) \times B(x) = \sum_{j=0}^{2n} c_j x^j, \quad \text{where } c_j = \sum_{j=0}^{n} a_j b_{n-j} \]

---

Polynomials: Point-Value Representation

**Fundamental theorem of algebra.** [Gauss, PhD thesis] A degree \(n\) polynomial with complex coefficients has exactly \(n\) complex roots.

**Corollary.** A degree \(n-1\) polynomial \(A(x)\) is uniquely specified by its evaluation at \(n\) distinct values of \(x.\)
Polynomials: Point-Value Representation

**Polynomial.** [point-value representation]

\[ A(x) : (x_0, y_0), \ldots, (x_n, y_n) \]
\[ B(x) : (x_0, z_0), \ldots, (x_n, z_n) \]

**Add.\ O(n) arithmetic operations.**

\[ A(x) + B(x) : (x_0, y_0 + z_0), \ldots, (x_n, y_n + z_n) \]

**Multiply (convolve).\ O(n), but need 2n-1 points.**

\[ A(x) \times B(x) : (x_0, y_0 \times z_0), \ldots, (x_{2n-1}, y_{2n-1} \times z_{2n-1}) \]

**Evaluate.\ O(n^2) using Lagrange’s formula.**

\[
A(x) = \sum_{i=0}^{n} y_i \prod_{j=0}^{n} \frac{(x-x_j)}{(x_i-x_j)}
\]

Converting Between Two Representations: Brute Force

**Coefficient \(\Rightarrow\) point-value.** Given a polynomial \(a_0 + a_1 x + \ldots + a_n x^{n-1}\), evaluate it at \(n\) distinct points \(x_0, \ldots, x_n\).

\[
\begin{pmatrix}
    y_0 \\
    y_1 \\
    y_2 \\
    \vdots \\
    y_{n-1}
\end{pmatrix} = \begin{pmatrix}
    1 & x_0 & x_0^2 & \ldots & x_0^{n-1} \\
    1 & x_1 & x_1^2 & \ldots & x_1^{n-1} \\
    1 & x_2 & x_2^2 & \ldots & x_2^{n-1} \\
    \vdots & \vdots & \vdots & \ddots & \vdots \\
    1 & x_{n-1} & x_{n-1}^2 & \ldots & x_{n-1}^{n-1}
\end{pmatrix} \begin{pmatrix}
    a_0 \\
    a_1 \\
    a_2 \\
    \vdots \\
    a_{n-1}
\end{pmatrix}
\]

**Running time.\ O(n^2) for matrix-vector multiply (or n Horner’s).**

Converting Between Two Representations: Fast Multiplication

**Tradeoff.** Fast evaluation or fast multiplication. We want both!

<table>
<thead>
<tr>
<th>representation</th>
<th>multiply</th>
<th>evaluate</th>
</tr>
</thead>
<tbody>
<tr>
<td>coefficient</td>
<td>(O(n^3))</td>
<td>(O(n))</td>
</tr>
<tr>
<td>point-value</td>
<td>(O(n))</td>
<td>(O(n^2))</td>
</tr>
</tbody>
</table>

**Goal.** Efficient conversion between two representations \(\Rightarrow\) all ops fast.

\[
\begin{pmatrix}
    a_0, a_1, \ldots, a_{n-1}
\end{pmatrix} \rightarrow \begin{pmatrix}
    (x_0, y_0), \ldots, (x_n, y_n)
\end{pmatrix}
\]

Converting Between Two Representations: Brute Force

**Point-value \(\Rightarrow\) coefficient.** Given \(n\) distinct points \(x_0, \ldots, x_n\) and values \(y_0, \ldots, y_n\), find unique polynomial \(a_0 + a_1 x + \ldots + a_n x^{n-1}\), that has given values at given points.

\[
\begin{pmatrix}
    y_0 \\
    y_1 \\
    y_2 \\
    \vdots \\
    y_{n-1}
\end{pmatrix} = \begin{pmatrix}
    1 & x_0 & x_0^2 & \ldots & x_0^{n-1} \\
    1 & x_1 & x_1^2 & \ldots & x_1^{n-1} \\
    1 & x_2 & x_2^2 & \ldots & x_2^{n-1} \\
    \vdots & \vdots & \vdots & \ddots & \vdots \\
    1 & x_{n-1} & x_{n-1}^2 & \ldots & x_{n-1}^{n-1}
\end{pmatrix} \begin{pmatrix}
    a_0 \\
    a_1 \\
    a_2 \\
    \vdots \\
    a_{n-1}
\end{pmatrix}
\]

\[
\text{Vandermonde matrix is invertible iff } x_i \text{ distinct}
\]

**Running time.\ O(n^2) for Gaussian elimination.**

or \(O(n^{2.376})\) via fast matrix multiplication
Divide-and-Conquer

Decimation in frequency. Break up polynomial into low and high powers.
- \( A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7. \)
- \( A_{\text{low}}(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3. \)
- \( A_{\text{high}}(x) = a_4 + a_5 x + a_6 x^2 + a_7 x^3. \)
- \( A(x) = A_{\text{low}}(x) + x^4 A_{\text{high}}(x). \)

Decimation in time. Break polynomial up into even and odd powers.
- \( A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7. \)
- \( A_{\text{even}}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^4. \)
- \( A_{\text{odd}}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^4. \)
- \( A(x) = A_{\text{even}}(x^2) + x A_{\text{odd}}(x^2). \)

Coefficient to Point-Value Representation: Intuition

Coefficient \(\Rightarrow\) point-value. Given a polynomial \(a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}\), evaluate it at \(n\) distinct points \(x_0, \ldots, x_{n-1}\).

Divide. Break polynomial up into even and odd powers.
- \( A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7. \)
- \( A_{\text{even}}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^4. \)
- \( A_{\text{odd}}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^4. \)
- \( A(x) = A_{\text{even}}(x^2) + x A_{\text{odd}}(x^2). \)

Intuition. Choose four complex points to be \(\pm 1, \pm i\).
- \( A(1) = A_{\text{even}}(1) + i A_{\text{odd}}(1). \)
- \( A(-1) = A_{\text{even}}(-1) - i A_{\text{odd}}(-1). \)
- \( A(i) = A_{\text{even}}(-1) + i A_{\text{odd}}(1). \)
- \( A(-i) = A_{\text{even}}(1) - i A_{\text{odd}}(-1). \)

Discrete Fourier Transform

Coefficient \(\Rightarrow\) point-value. Given a polynomial \(a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}\), evaluate it at \(n\) distinct points \(x_0, \ldots, x_{n-1}\).

Key idea. Choose \(x_j = \omega^j \) where \(\omega\) is principal \(n\)th root of unity.

Can evaluate polynomial of degree \(\leq n\) at 4 points by evaluating two polynomials of degree \(\leq \frac{n}{2}\) at 1 point.
**Roots of Unity**

**Def.** An $n^{th}$ root of unity is a complex number $x$ such that $x^n = 1$.

**Fact.** The $n^{th}$ roots of unity are: $\omega^n, \omega^1, \ldots, \omega^{n-1}$ where $\omega = e^{2\pi i/n}$.

**Pf.** $(\omega^n)^n = (e^{2\pi i/n})^n = (e^{2\pi i})^n = (-1)^n = 1$.

**Fact.** The $\frac{1}{2}n^{th}$ roots of unity are: $\nu^n, \nu^1, \ldots, \nu^{n-2}$ where $\nu = \omega^2 = e^{4\pi i/n}$.

**FFT Algorithm**

```plaintext
fft(n, a_0, a_1, ..., a_{n-1}) {
    if (n == 1) return a_0
    (a_0, a_1, ..., a_{n/2-1}) ← FFT(n/2, a_0, a_2, ..., a_{n-2})
    (d_0, d_1, ..., d_{n/2-1}) ← FFT(n/2, a_1, a_3, ..., a_{n-1})

    for k = 0 to n/2 - 1 {
        $\omega^k \leftarrow e^{2\pi ik/n}$
        $Y_k \leftarrow a_k + \omega^k d_k$
        $Y_{k+n/2} \leftarrow a_k - \omega^k d_k$
    }

    return (Y_0, Y_1, ..., Y_{n-1})
}
```

**Fast Fourier Transform**

**Goal.** Evaluate a degree $n$-1 polynomial $A(x) = a_0 + ... + a_{n-1}x^n$ at its $n^{th}$ roots of unity: $\omega^n, \omega^1, \ldots, \omega^{n-1}$.

**Divide.** Break up polynomial into even and odd powers.
- $A_{even}(x) = a_0 + a_1x + a_3x^2 + ... + a_{n-2}x^{n-2}$.
- $A_{odd}(x) = a_1 + a_2x + a_4x^2 + ... + a_{n-1}x^{n-1}$.
- $A(x) = A_{even}(x^2) + x A_{odd}(x^2)$.

**Conquer.** Evaluate $A_{even}(x)$ and $A_{odd}(x)$ at the $\frac{1}{2}n^{th}$ roots of unity: $\nu^n, \nu^1, \ldots, \nu^{n-2}$.

**Combine.**
- $A(\omega^k) = A_{even}(\omega^k) + \omega^{k\cdot n} A_{odd}(\nu^k)$, $0 \leq k < n/2$
- $A(\omega^{k\cdot n}) = A_{even}(\nu^k) - \omega^{k\cdot n} A_{odd}(\nu^k)$, $0 \leq k < n/2$

**Theorem.** FFT algorithm evaluates a degree $n$-1 polynomial at each of the $n^{th}$ roots of unity in $O(n \log n)$ steps.

**Running time.**

$$T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = \Theta(n \log n)$$

**FFT Summary**

- **Coefficient representation:** $a_0, a_1, ..., a_{n-1}$
- **Point-value representation:** $(\omega^0, y_0), ..., (\omega^{n-1}, y_{n-1})$
Inverse FFT

Point-value ⇒ coefficient. Given \(n\) distinct points \(x_0, \ldots, x_{n-1}\) and values \(y_0, \ldots, y_{n-1}\), find unique polynomial \(a_0 + a_1x + \ldots + a_{n-1}x^{n-1}\), that has given values at given points.

\[
\begin{bmatrix}
a_0 \\
a_1 \\
a_2 \\
\vdots \\
a_{n-1}
\end{bmatrix} =
\begin{bmatrix}
1 & 1 & 1 & \cdots & 1 \\
1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\
1 & \omega^2 & \omega^4 & \cdots & \omega^{2(n-1)} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \omega^{n-1} & \omega^{2(n-1)} & \cdots & \omega^{(n-1)(n-1)}
\end{bmatrix}^{-1}
\begin{bmatrix}
y_0 \\
y_1 \\
y_2 \\
\vdots \\
y_{n-1}
\end{bmatrix}
\]

Inverse DFT

Fourier matrix inverse \((F_n)^{-1}\)
Inverse DFT

Claim. Inverse of Fourier matrix \( F_n \) is given by following formula.

\[
G_n = \frac{1}{n} \begin{bmatrix}
1 & 1 & 1 & \cdots & 1 \\
\omega^{-1} & \omega^{-2} & \omega^{-3} & \cdots & \omega^{-(n-1)} \\
\omega^{-2} & \omega^{-3} & \omega^{-5} & \cdots & \omega^{-(2n-1)} \\
\omega^{-3} & \omega^{-5} & \omega^{-9} & \cdots & \omega^{-(3n-1)} \\
\vdots & \vdots & \vdots & \cdots & \vdots \\
\omega^{-(n-1)} & \omega^{-(2n-1)} & \omega^{-(3n-1)} & \cdots & \omega^{-(n^2-n-1)}
\end{bmatrix}
\]

\( \frac{1}{\sqrt{n}} F_n \) is unitary

Consequence. To compute inverse FFT, apply same algorithm but use \( \omega^{-1} = e^{-2\pi i/n} \) as principal \( n^{th} \) root of unity (and divide by \( n \)).

Inverse FFT: Algorithm

```c
ifft(n, a0, a1, ..., an-1) {
    if (n == 1) return a0

    (e0, e1, ..., en/2-1) ← FFT(n/2, a0, a2, a4, ..., an-2)
    (d0, d1, ..., dn/2-1) ← FFT(n/2, a1, a3, a5, ..., an-1)

    for k = 0 to n/2 - 1 {
        \( \omega^k \leftarrow \omega^{2\pi i k/n} \)
        \( Y_{k+n/2} \leftarrow (e_k + \omega^k d_k) / \sqrt{n} \)
        \( Y_{k+n/2} \leftarrow (e_k - \omega^k d_k) / \sqrt{n} \)
    }

    return (Y0, Y1, ..., Yn-1)
}
```

Inverse FFT: Proof of Correctness

Claim. \( F_n \) and \( G_n \) are inverses.

Pf.

\[
(F_n G_n)_{k,k'} = \frac{1}{n} \sum_{j=0}^{n-1} \omega^{k j} \omega^{-j k'} = \begin{cases} 1 & \text{if } k = k' \\ 0 & \text{otherwise} \end{cases}
\]

Summation lemma. Let \( \omega \) be a principal \( n^{th} \) root of unity. Then

\[
\sum_{j=0}^{n-1} \omega^{k j} = \begin{cases} n & \text{if } k \equiv 0 \mod n \\ 0 & \text{otherwise} \end{cases}
\]

Pf.

1. If \( k \) is a multiple of \( n \) then \( \omega^k = 1 \implies \text{series sums to } n \).
2. Each \( n^{th} \) root of unity \( \omega^k \) is a root of \( x^n - 1 = (x - 1)(1 + x + x^2 + \cdots + x^{n-1}) \).
3. If \( \omega^k \neq 1 \) we have: \( 1 + \omega^k + \omega^{k2} + \cdots + \omega^{k(n-1)} = 0 \implies \text{series sums to } 0 \).

Inverse FFT Summary

Theorem. Inverse FFT algorithm interpolates a degree \( n-1 \) polynomial given values at each of the \( n^{th} \) roots of unity in \( O(n \log n) \) steps.

\( \text{assumes } n \text{ is a power of } 2 \)
Theorem. Can multiply two degree $n-1$ polynomials in $O(n \log n)$ steps.

- $a_0, a_1, \ldots, a_{n-1}\}$
- $b_0, b_1, \ldots, b_{n-1}\}$
- $c_0, c_1, \ldots, c_{2n-2}\}$

Fastest Fourier transform in the West. [Frigo and Johnson]

Optimized C library.
- Features: DFT, DCT, real, complex, any size, any dimension.
- Won Wilkinson Prize '99.
- Portable, competitive with vendor-tuned code.
- Won Wilkinson Prize '99.
- Portable, competitive with vendor-tuned code.

Implementation details.
- Instead of executing predetermined algorithm, it evaluates your
- hardware and uses a special-purpose compiler to generate an
- optimized algorithm catered to "shape" of the problem.

- Core algorithm is nonrecursive version of Cooley-Tukey.

- $O(n \log n)$, even for prime sizes.
**Integer Multiplication, Redux**

**Integer multiplication.** Given two \( n \) bit integers \( a = a_{n-1} \ldots a_0 \) and \( b = b_{n-1} \ldots b_0 \), compute their product \( ab \).

**Convolition algorithm.**
- Form two polynomials. \( A(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1} \)
- Note: \( a = A(2), b = B(2) \).
- Compute \( C(x) = A(x)B(x) \).
- Evaluate \( C(2) = ab \).
- Running time: \( O(n \log n) \) complex arithmetic operations.

**Theory.** [Schönhage-Strassen 1971] \( O(n \log n \log \log n) \) bit operations.

**Factoring**

**Factoring.** Given an \( n \)-bit integer, find its prime factorization.

\[
2^{67} - 1 = 147573952589676412927 = 193707721 \times 761838257287
\]

\( a \) proof of Mersenne’s conjecture that \( 2^{67} - 1 \) is prime

\[
740375634795617128280467960974295731425931888892312890849
362326389727650340282662768919964196251178439958943305021
275853701189680982867331732731098309005525051168770632990
7239638078671008609696253793465056376359
\]

\( RSA-704 \)
(\$10,000 prize if you can factor)
Factoring and RSA

Primality. Given an $n$-bit integer, is it prime?

Factoring. Given an $n$-bit integer, find its prime factorization.

Significance. Efficient primality testing $\Rightarrow$ can implement RSA.

Significance. Efficient factoring $\Rightarrow$ can break RSA.

Theorem. Poly-time algorithm for primality testing.

Shor's Algorithm

Shor's algorithm. Can factor an $n$-bit integer in $O(n^3)$ time on a quantum computer.

Ramification. At least one of the following is wrong:

- RSA is secure.
- Textbook quantum mechanics.
- Extended Church-Turing thesis.

Shor's Factoring Algorithm

Period finding.

| $2^i$  | 1   | 2   | 4   | 8   | 16  | 32  | 64  | 128 | ...
|--------|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| $2^i \mod 15$ | 1   | 2   | 4   | 8   | 1   | 2   | 4   | 8   | ...
| $2^i \mod 21$ | 1   | 2   | 4   | 8   | 16  | 1   | 2   | 4   | ...

Theorem. [Euler] Let $p$ and $q$ be prime, and let $n = pq$. Then, the following sequence repeats with a period divisible by $(p-1)(q-1)$:

$x \mod n$, $x^2 \mod n$, $x^3 \mod n$, $x^4 \mod n$, ...

Consequence. If we can learn something about the period of the sequence, we can learn something about the divisors of $(p-1)(q-1)$.

Ramification. At least one of the following is wrong:

- RSA is secure.
- Textbook quantum mechanics.
- Extended Church-Turing thesis.