Complex numbers

Any polynomial $p(x)$ of degree $d$ ought to have $d$ roots. (I.e., $p(x) = 0$ should have $d$ solutions.)

But the equation

$$x^2 + 1 = 0 \quad (\ast)$$

has no solutions at all if we restrict our attention to real numbers.

Introduce a special symbol $i$ to stand for a solution to $(\ast)$. Then $i^2 = -1$ and $(\ast)$ has the required two solutions, $i$ and $-i$.

Adding $i$ allows all polynomial equations to be solved! Indeed a polynomial of degree $d$ has $d$ roots (taking account of multiplicities). This is the Fundamental Theorem of Algebra.

Roots of Unity

In particular,

$$x^n = 1$$

has $n$ solutions in the complex numbers. They may be written

$$1, \omega_n, \omega_n^2, \ldots, \omega_n^{n-1}$$

where $\omega_n$ is the principal $n$th root of unity:

$$\omega_n = \cos(2\pi/n) + i\sin(2\pi/n), \quad \text{(**)}.$$  

Convention: from now on $\omega_n$ denotes the principal $n$th root of unity given by $(\ddagger)$.

Note: $e^{iu} = \cos u + i\sin u$ so $\omega_n = e^{2\pi i/n}$.

8th Roots of Unity

```
8\omega_0 = 1
2\pi/8
8\omega = e^i*2\pi/8
= i
= (1+i)/sqrt(2)
= (cos (2 pi/8), i*sin(2 pi/8))
```

"Wheel" representation of 8th roots-of-unity (complex plane)).

"Wheel" representation for any $n$ (then $\omega_n$ found at angle $2\pi/n$).
The Discrete Fourier Transform (DFT)

**Instance** A sequence of \( n \) complex numbers

\[
a_0, a_1, a_2, \ldots, a_{n-1},
\]

\( n \) is a Power-of-2.

**Output** The sequence of \( n \) complex numbers

\[
A(1), A(\omega_n), A(\omega_n^2), \ldots, A(\omega_n^{n-1})
\]

obtained by evaluating the polynomial

\[
A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}
\]
at the \( n \)th roots of unity.

The DFT is a fingerprint of size \( n \) of a polynomial.

**CLASS QUESTION:** It’s not the only fingerprint (why?)

Divide-and-Conquer

We are interested in evaluating:

\[
A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1},
\]

\( n \) a Power-of-2. Put

\[
A_{\text{even}}(y) = a_0 + a_2 y + \cdots + a_{n-2} y^{n/2-1},
\]

\[
A_{\text{odd}}(y) = a_1 + a_3 y + \cdots + a_{n-1} y^{n/2-1},
\]

so that

\[
A(x) = A_{\text{even}}(x^2) + x A_{\text{odd}}(x^2).
\]

To evaluate \( A(x) \) at the \( n \)th roots of unity, we need to evaluate \( A_{\text{even}}(y) \) and \( A_{\text{odd}}(y) \) at the points \( 1, \omega_n^2, \omega_n^4, \ldots, \omega_n^{2(n-1)} \).

We’ll show now that these are DFTs. (wrt \( n/2 \))

**Motivation for algorithms for DFT/Inverse DFT**

**Direct.** Signal processing: mapping between time and frequency domains.

**Indirect.** Subroutine in numerous applications, e.g., multiplying polynomials or large integers, cyclic string matching, etc.

It is important, therefore to find the fastest method. There is an obvious \( \Theta(n^2) \) algorithm. Can we do better?

YES! Really cool algorithm (Fast Fourier Transform (FFT)) runs in \( O(n \lg n) \) time. Published by Cooley & Tukey in 1965 - basics known by Gauss in 1805!

Used in *every* Digital Signal Processing application. Probably the most important algorithm of today. We will show how to apply FFT to do polynomial multiplication in \( O(n \lg n) \) (not most common application, but cute).

**Key Facts**

Assuming \( n \) is even:

- \( \omega_n^2 = (e^{\frac{2\pi i}{n}})^2 = e^{\frac{2\pi i}{n^2}} = \omega_{n/2} \), and
- \( \omega_n^{n/2} = (e^{\frac{2\pi i}{n}})^{n/2} = e^{\pi i} = -1 \).

Thus we have the following relationships between \( \omega_n \) and \( \omega_{n/2} \):

\[
\begin{array}{ccccccc}
\omega_n^1 & \cdots & \omega_n^{n/2} & \omega_n^n & \omega_n^{n+2} & \cdots & \omega_n^{2(n-1)} \\
\| & \cdots & & & & & \\
\omega_{n/2}^1 & \cdots & \omega_{n/2}^{n/2-1} & 1 & \omega_{n/2}^{n/2+1} & \cdots & \omega_{n/2}^{n/2+1}
\end{array}
\]

So evaluating \( A_{\text{odd}}(x) \), \( A_{\text{even}}(x) \) at \( \omega^2 \) for all \( n \)th-roots-of-unity (in order to implement (#)), is TWO “sweeps” of evaluating \( A_{\text{odd}}(x) \), \( A_{\text{even}}(x) \) at the \( n/2 \)th-roots.
“Divide”: a warning

In performing the “Divide” part of Divide-and-Conquer to DFT, it was important that the “Divide” was based on odd/even.

Suppose we had instead partitioned $A(x)$ into small/larger terms:

$$A_{\text{small}}(y) = a_0 + a_1 y + \cdots + a_{n/2-1}y^{n/2-1}$$

$$A_{\text{big}}(y) = a_{n/2} + a_{n/2+1} y + \cdots + a_{n-1}y^{n/2-1}$$

Then we would have

$$A(x) = A_{\text{small}}(x) + x^{n/2} A_{\text{big}}(x).$$

However, to evaluate $A(x)$ at the $n$th roots of unity, we would need to evaluate $A_{\text{small}}(y)$ and $A_{\text{big}}(y)$ at all of the $n$th roots of unity.

So for recursive calls: we would reduce the degree of the polynomial (to $n/2 - 1$), but would NOT reduce the “number of roots”. We would lose the relationship between degree of poly. and number of roots, which is CRUCIAL.

Key Facts (cont’d)

$$A(1) = A_{\text{even}}(1) + 1 \cdot A_{\text{odd}}(1)$$

$$A(\omega_n) = A_{\text{even}}(\omega_n^2) + \omega_n A_{\text{odd}}(\omega_n^2) = A_{\text{even}}(\omega_{n/2}) + \omega_n A_{\text{odd}}(\omega_{n/2})$$

$$A(\omega_n^2) = A_{\text{even}}(\omega_{n/2}^2) + \omega_n^2 A_{\text{odd}}(\omega_{n/2}^2)$$

$$\vdots$$

$$A(\omega_n^{n/2-1}) = A_{\text{even}}(\omega_{n/2}^{n/2-1}) + \omega_n^{n/2-1} A_{\text{odd}}(\omega_{n/2}^{n/2-1})$$

The $x$ co-efficient on $xA_{\text{odd}}(x^2)$ of (#) stays positive until $x = \omega_n^{n/2}$.

Key Facts (cont’d)

$$A(\omega_n^{n/2}) = A_{\text{even}}(1) - 1 \cdot A_{\text{odd}}(1)$$

$$A(\omega_n^{n/2+1}) = A_{\text{even}}(\omega_n^{n/2}) - \omega_n A_{\text{odd}}(\omega_n^{n/2})$$

$$\vdots$$

$$A(\omega_n^{n-1}) = A_{\text{even}}(\omega_n^{n/2-1}) - \omega_n^{n/2-1} A_{\text{odd}}(\omega_n^{n/2-1})$$

From $\omega_n^{n/2}$ on, the $x$ co-efficient of $xA_{\text{odd}}(x^2)$ of (#) is negative.
We will use this negative relationship (with the $j < n/2$ case) on lines 8., 9. of our pseudocode.

The Fast Fourier Transform (FFT)

$$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1}x^{n-1},$$

assume $n$ is a power of 2. Compute

$$A(1), A(\omega_n), A(\omega_n^2), \ldots, A(\omega_n^{n-1}),$$

as follows:

1. If $n = 1$ then $A(x)$ is a constant so task is trivial. Otherwise split $A$ into $A_{\text{even}}$ and $A_{\text{odd}}$.
2. By making two recursive calls compute the values of $A_{\text{even}}(y)$ and $A_{\text{odd}}(y)$ at the $(n/2)$ points $1, \omega_{n/2}, \omega_{n/2}^2, \ldots, \omega_{n/2}^{n/2-1}$.
3. Compute the values (*) by using the equation

$$A(x) = A_{\text{even}}(x^2) + xA_{\text{odd}}(x^2).$$
Algorithm \( \text{FFT}_n(\langle a_0, \ldots, a_{n-1} \rangle) \)

1. if \( n = 1 \) then return \( \langle a_0 \rangle \)
2. else
3. \( \omega_n \leftarrow e^{2\pi i / n} \)
4. \( \omega \leftarrow 1 \)
5. \( \langle y_0^{\text{even}}, \ldots, y_{n/2-1}^{\text{even}} \rangle \leftarrow \text{FFT}_{n/2}(\langle a_0, a_2, \ldots, a_{n-2} \rangle) \)
6. \( \langle y_0^{\text{odd}}, \ldots, y_{n/2-1}^{\text{odd}} \rangle \leftarrow \text{FFT}_{n/2}(\langle a_1, a_3, \ldots, a_{n-1} \rangle) \)
7. for \( k \leftarrow 0 \) to \( n/2 - 1 \) do
   8. \( y_k \leftarrow y_k^{\text{even}} + \omega y_k^{\text{odd}} \)
   9. \( y_{k+n/2} \leftarrow y_k^{\text{even}} - \omega y_k^{\text{odd}} \)
10. \( \omega \leftarrow \omega \omega_n \)
11. return \( \langle y_0, \ldots, y_{n-1} \rangle \)


The Discrete Fourier Transform

Recall

- The DFT maps a tuple \( \langle a_0, \ldots, a_{n-1} \rangle \) to the tuple \( \langle y_0, \ldots, y_{n-1} \rangle \) defined by
  \[
  y_j = \sum_{k=0}^{n-1} a_k \omega_n^{jk},
  \]
  where \( \omega_n = e^{2\pi i / n} \) is the principal \( n \)th root of unity.

- Thus for every \( n \) (power of 2) we may view \( \text{DFT}_n \) as mapping \( \mathbb{C}^n \rightarrow \mathbb{C}^n \), where \( \mathbb{C} \) denote the complex numbers.

- FFT (the Fast Fourier Transform) is an algorithm computing \( \text{DFT}_n \) in time \( \Theta(n \cdot \lg(n)) \).

The inverse DFT

\[
\text{DFT}_n : \mathbb{C}^n \rightarrow \mathbb{C}^n
\]

\[
\langle a_0, \ldots, a_{n-1} \rangle \mapsto \langle y_0, \ldots, y_{n-1} \rangle
\]

Question

Can we go back from \( \langle y_0, \ldots, y_{n-1} \rangle \) to \( \langle a_0, \ldots, a_{n-1} \rangle \)?

More precisely:

1. Is \( \text{DFT}_n \) invertible, that is, is it one-to-one and onto?
2. If the answer to (1) is ‘yes’, can we compute \( \text{DFT}_n^{-1} \) efficiently?
An alternative view on the DFT

DFTₙ is the linear mapping described by the matrix

\[
Vₙ = \begin{pmatrix}
1 & 1 & 1 & \ldots & 1 \\
1 & ωₙ & ω₂ₙ & \ldots & ωⁿ⁻¹ₙ \\
1 & ω²ₙ & ω₄ₙ & \ldots & ω²ⁿ⁻１ₙ \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & ωⁿ⁻¹ₙ & ω²ⁿ⁻¹ₙ & \ldots & ωⁿ⁻¹(n⁻¹)ₙ
\end{pmatrix}.
\]

That is, we have

\[
Vₙ \begin{pmatrix}
a₀ \\
a₁ \\
\vdots \\
aₙ⁻₁
\end{pmatrix} = \begin{pmatrix}
y₀ \\
y₁ \\
\vdots \\
yₙ⁻₁
\end{pmatrix}
\]

We will NOT actually perform the naïve matrix mult.
(we will do much better: O(n lg n))

Inverse of DFT (proof)

Verification: We must check that \( VₙV⁻¹ₙ = Iₙ \): Want \( ℓℓ \)-th entry = 1 ∀ℓ, and \( ℓj \)-th entry = 0 ∀ℓ, j with \( ℓ \neq j \).
Expanding ...

\[
(VₙV⁻¹ₙ)_{ij} = \frac{1}{n} \sum_{k=0}^{n⁻¹} ωⁿᵏⱼ \cdot \omega⁻ᵏₖ = \frac{1}{n} \sum_{k=0}^{n⁻¹} \omega^{(ℓ−j)k},
\]

\[
= \begin{cases}
1 & \text{if } ℓ = j \text{ (because } ω⁻¹ = 1) \\
0 & \text{otherwise}
\end{cases}
\]

\((VₙV⁻¹ₙ)_{ij} = 0 \text{ case uses the fact that for all } r \neq 0 \text{ (} r = (ℓ − j) \text{)}
we have \( \sum_{k=0}^{n⁻¹} ω^{rk} = 0 \).

Inverse of DFT

Claim: \( Vₙ \) is a van-der-Monde matrix and thus invertible.

Proof: Define the following “Inverse” matrix:

\[
V⁻¹ₙ = \frac{1}{n} \begin{pmatrix}
1 & 1 & 1 & \ldots & 1 \\
1 & ω⁻¹ₙ & ω⁻²ₙ & \ldots & ω⁻ⁿ⁻¹ₙ \\
1 & ω⁻²ₙ & ω⁻⁴ₙ & \ldots & ω⁻²ⁿ⁻¹ₙ \\
1 & ω⁻³ₙ & ω⁻⁶ₙ & \ldots & ω⁻³ⁿ⁻¹ₙ \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & ω⁻ⁿ⁻¹ₙ & ω⁻²ⁿ⁻¹ₙ & \ldots & ω⁻ⁿ⁻¹(n⁻¹)ₙ
\end{pmatrix}.
\]

We have shown DFTₙ is invertible with

\[
DFT⁻¹ₙ : \begin{pmatrix}
y₀ \\
\vdots \\
yₙ⁻₁
\end{pmatrix} \mapsto V⁻¹ₙ \begin{pmatrix}
y₀ \\
\vdots \\
yₙ⁻₁
\end{pmatrix} = \begin{pmatrix}
a₀ \\
\vdots \\
aₙ⁻₁
\end{pmatrix}.
\]

Problem

If we are were to apply \( V⁻¹ₙ \langle y₀, \ldots, yₙ⁻₁ \rangle \) directly in order to recover \( \langle a₀, \ldots, aₙ⁻₁ \rangle \), the evaluation of \( V⁻¹ₙ \langle y₀, \ldots, yₙ⁻₁ \rangle \) would take \( Θ(n^²) \) time!!!

Solution

Take another look back at the \( V⁻¹ₙ \) matrix, and see that it is more-or-less a “flipped-over” DFT.


Inverse DFT (efficient) Algorithm

ω⁻¹
n is an nth root of unity (though not the principal one). Note that

(ω⁻¹
n )j = ... if 1 ≤ i ≤ n − 1

for every 0 ≤ j < n.

Inverse FFT

- Compute DFTₙ⟨y₀, ..., yₙ₋₁⟩ (deliberately using DFT, not inverse), to obtain the result ⟨d₀, ..., dₙ₋₁⟩.
- Flip the sequence d₁, d₂, ..., dₙ₋₁ in this result (keeping d₀ fixed), then divide every term by n.

\[ a_i = \begin{cases} \frac{d_0}{n} & \text{if } i = 0 \\ \frac{d_{i-n}}{n} & \text{if } 1 \leq i \leq n - 1 \end{cases} \]

Worst-case running time is Θ(n lg(n)).

Interpolation

Theorem

Let α₀, ..., αₙ₋₁ ∈ C pairwise distinct and y₀, ..., yₙ₋₁ ∈ C. Then there exists exactly one polynomial p(X) of degree at most n − 1 such that for 0 ≤ k ≤ n − 1

p(α_k) = y_k.

Our Application! Multiplication of Polynomials

Input: p(x) = a₀ + a₁x + a₂x² + ⋯ + aₙ₋₁xⁿ⁻¹
q(x) = b₀ + b₁x + b₂x² + ⋯ + bₘ₋₁xᵐ⁻¹.

Required output:

\[ p(x)q(x) = (a₀b₀) \]
\[ + (a₀b₁ + a₁b₀)x \]
\[ + (a₀b₂ + a₁b₁ + a₂b₀)x² \]
... 
\[ + (a₀bₙ₋₁ + a₁bₙ₋₂ + a₂bₙ₋₃)xⁿ+m⁻³ \]
\[ + (a₀bₙ₋₁b₋₂ + a₁bₙ₋₂b₋₃ + a₂bₙ₋₃b₋₄)xⁿ+m⁻² \]

Naive method uses Θ(nm) arithmetic operations

CAN WE DO BETTER?
Multiplication of polynomials (cont’d)

Key idea
Let \( n' \) be the smallest power of 2 such that \( n' \geq n + m - 1 \).
Use the \( n' \)-th roots of unity as the evaluation points:
\[
\alpha_0 = 1, \quad \alpha_1 = \omega^{n'}, \quad \alpha_2 = \omega^{2n'}, \ldots, \quad \alpha_{n'-1} = \omega^{(n'-1)n'}.
\]
Then

\[\begin{align*}
\text{evaluation} &\equiv \text{DFT, and} \\
\text{interpolation} &\equiv \text{inverse DFT}
\end{align*}\]

Overall running time is
\[
\begin{align*}
\Theta(n' \log n') &= \Theta(n \log n) & \text{(FFT)} \\
+ \Theta(n') &= \Theta(n) & \text{(pointwise multiplication)} \\
+ \Theta(n' \log n') &= \Theta(n \log n) & \text{(inverse FFT)} \\
\end{align*}
\]

\[\Theta(n \log n)\]

Reading Assignment

Fast Fourier Transform, by M. Cryan, notes handed out today.

[CLRS] (2nd and 3rd ed) Section 30.2 and 30.3.

Problems

1. Exercise 30.2-2 of [CLRS].
2. Let \( f(x) = 3 \cos(2x) \). For \( 0 \leq k \leq 3 \), let \( a_k = f(2\pi k/4) \). Compute the DFT of \( \langle a_0, \ldots, a_3 \rangle \).

Do the same for \( f(x) = 5 \sin(x) \).
3. Exercise 30.2-3 of [CLRS].
4. Exercise 30.2-7 of [CLRS].