Algorithms
COMP3121/3821/9101/9801
5. THE FAST FOURIER TRANSFORM

School of Computer Science and Engineering
University of New South Wales Sydney

Coefficient vs value representation of polynomials

- Every polynomial $P(x)$ of degree n is uniquely determined by its values at any $n+1$ distinct input values $x_0, x_1, \ldots, x_n$:

$$P(x) \leftrightarrow \{(x_0, P(x_0)), (x_1, P(x_1)), \ldots, (x_n, P(x_n))\}$$

- If $P(x) = p_0 + p_1 x + \ldots + p_{n-1}x^{n-1} + p_n x^n$, evaluation of $P(x)$ at points $x_0, x_1, \ldots, x_n$ can be represented in a matrix form by a single matrix-vector multiplication:

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^n \\ 1 & x_1 & x_1^2 & \ldots & x_1^n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & \ldots & x_n^n \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_n \end{pmatrix} = \begin{pmatrix} P(x_0) \\ P(x_1) \\ \vdots \\ P(x_n) \end{pmatrix}. \quad (1)$$

- It can be shown that if x_i are all distinct, then this matrix is invertible. (It is called the *Vandermonde* matrix)

Coefficient vs value representation of polynomials - ctd.

- Thus, if all x_i are distinct, given values $P(x_0), P(x_1), \ldots, P(x_n)$ the coefficients $p_0, p_1, \ldots, p_n$ are uniquely determined:

$$\begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_n \end{pmatrix} = \begin{pmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^n \\ 1 & x_1 & x_1^2 & \ldots & x_1^n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & \ldots & x_n^n \end{pmatrix}^{-1} \begin{pmatrix} P(x_0) \\ P(x_1) \\ \vdots \\ P(x_n) \end{pmatrix} \tag{2}$$

- Equations (1) and (2) show how we can commute between:
  1. a representation of a polynomial $P(x)$ via the vector of its coefficients $P(x) \leftrightarrow \langle p_0, p_1, \ldots, p_{n-1}, p_n \rangle$ such that $P(x) = p_0 + p_1 x + \ldots + p_{n-1} x^{n-1} + p_n x^n$;
  2. a representation of a polynomial $P(x)$ via a set of $n+1$ ordered pairs of distinct inputs x_i and the values $P(x_i)$ of such a polynomial at these inputs $P(x) \leftrightarrow \{(x_0, P(x_0)), (x_1, P(x_1)), \ldots, (x_n, P(x_n))\}$

Our strategy to multiply polynomials fast:

- Given two polynomials of degree $\leq n$,

$$P_A(x) = a_n x^n + \ldots + a_0 \qquad P_B(x) = b_n x^n + \ldots + b_0$$

1. convert them into value representation at $2n+1$ distinct points $x_0, x_1, \ldots, x_{2n}$:

$$P_A(x) \leftrightarrow \{(x_0, P_A(x_0)), (x_1, P_A(x_1)), \ldots, (x_{2n}, P_A(x_{2n}))\}$$
$$P_B(x) \leftrightarrow \{(x_0, P_B(x_0)), (x_1, P_B(x_1)), \ldots, (x_{2n}, P_B(x_{2n}))\}$$

2. multiply them point by point using $2n+1$ multiplications:

$$\Big\{(x_0, \underbrace{P_A(x_0)P_B(x_0)}_{P_C(x_0)}), \quad (x_1, \underbrace{P_A(x_1)P_B(x_1)}_{P_C(x_1)}), \ldots, (x_{2n}, \underbrace{P_A(x_{2n})P_B(x_{2n})}_{P_C(x_{2n})})\Big\}$$

3. Convert such value representation of $P_C(x)$ to its coefficient form

$$P_C(x) = c_{2n}x^{2n} + c_{2n-1}x^{2n-1} + \ldots + c_1 x + c_0;$$

Key Question:

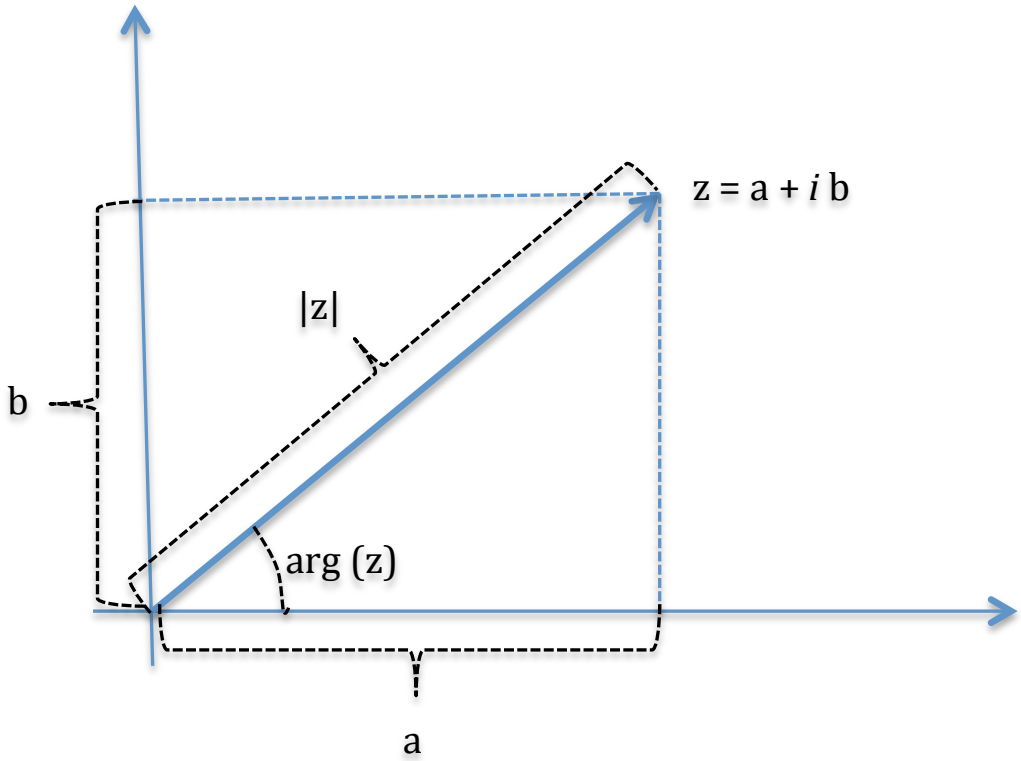
What values should we take for $x_0, \ldots, x_{2n}$ to avoid "explosion" of size when we evaluate x_i^n while computing $P_A(x_i) = a_0 + a_1 x_i + \ldots + a_n x_i^n$?

Complex numbers revisited

Complex numbers $z = a + ib$ can be represented using their *modulus* $|z| = \sqrt{a^2 + b^2}$ and their *argument*, $\arg(z)$, which is an angle taking values in $(-\pi, \pi]$ and satisfying:

$$z = |z|e^{i \arg(z)} = |z|(\cos \arg(z) + i \sin \arg(z)),$$

see figure below.

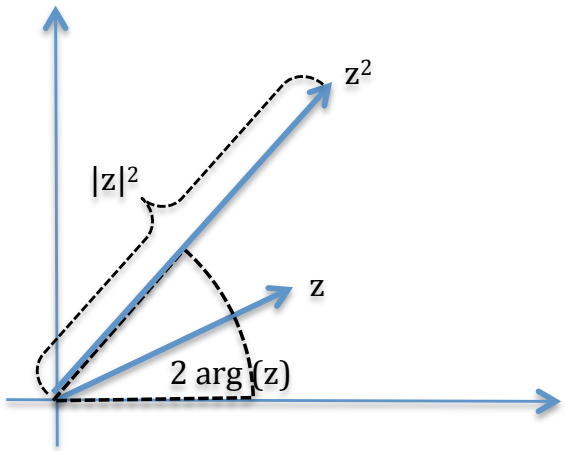


Complex numbers revisited

Recall that

$$z^n = \left(|z|e^{i\ \arg(z)}\right)^n = |z|^n e^{i\,n\,\arg(z)} = |z|^n(\cos(n\arg(z)) + i\sin(n\arg(z))),$$

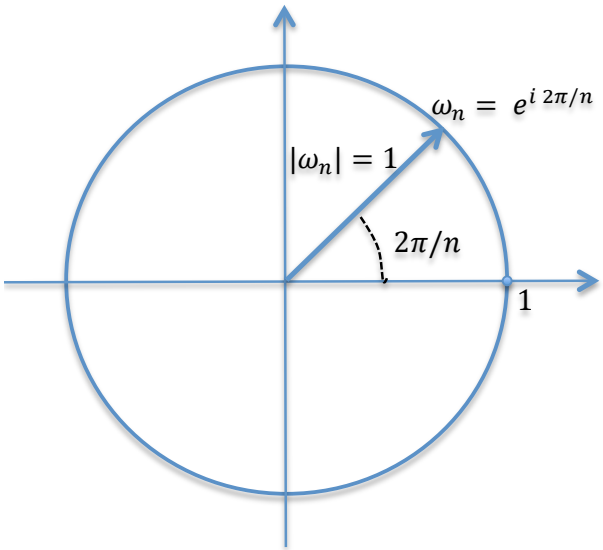
see the figure.

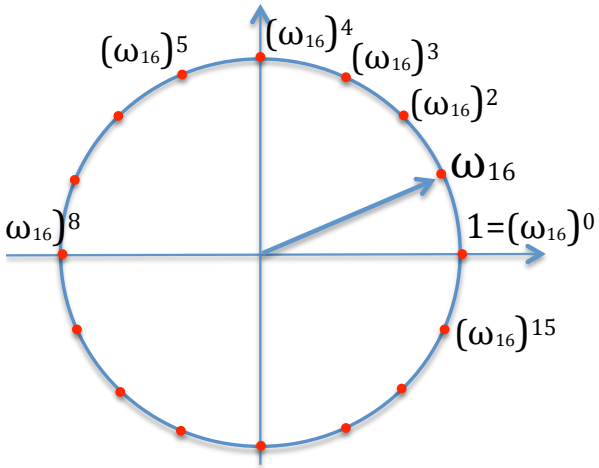


Complex roots of unity

- *Roots of unity of order* n are complex numbers which satisfy $z^n = 1$.
- If $z^n = |z|^n(\cos(n \arg(z)) + i \sin(n \arg(z))) = 1$ then

  $|z| = 1$ and $n \arg(z)$ is a multiple of 2π;
- Thus, $n \arg(z) = 2\pi k$, i.e., $\arg(z) = \dfrac{2\pi k}{n}$
- We denote $\omega_n = e^{i\,2\pi/n}$; such ω_n is called a *primitive root of unity of order* n.





Roots of unity of order 16

- A root of unity ω of order n is *primitive* if all other roots of unity of the same order can be obtained as its powers ω^k.

Complex roots of unity

- For $\omega_n = e^{i\,2\pi/n}$ and for all k such that $0 \leq k \leq n-1$,

$$((\omega_n)^k)^n = (\omega_n)^{n\,k} = ((\omega_n)^n)^k = 1^k = 1.$$

- Thus, $\omega_n^k = (\omega_n)^k$ is also a root of unity (and it can be shown that it is primitive just in case k is relatively prime with n).
- Since ω_n^k are roots of unity for $k = 0, 1, \ldots, n-1$ and there are at most n distinct roots of unity of order n (i.e., solutions to the equation $x^n - 1 = 0$) we conclude that every root of unity of order n must be of the form ω_n^k.
- For the product of any two roots of unity ω_n^k and ω_n^m of the same order we have $\omega_n^k \omega_n^m = \omega_n^{k+m}$.
- If $k + m \geq n$ then $k + m = n + l$ for $l = (k + m) \bmod n$ and we have $\omega_n^k \omega_n^m = \omega_n^{k+m} = \omega_n^{n+l} = \omega_n^n \omega_n^l = 1 \cdot \omega_n^l = \omega_n^l$ where $0 \leq l < n$.
- Thus, product of any two roots of unity of the same order is just another root of unity of the same order.

Complex roots of unity

- So in the set of all roots of unity of order n, i.e., $\{1, \omega_n, \omega_n^2, \ldots, \omega_n^{n-1}\}$ we can multiply any two elements or raise an element to any power without going out of this set.
- Note that this is not true for addition, i.e., the sum of two roots of unity is NOT another root of unity!

- A most important property of the roots of unity is:

The Cancellation Lemma

For all integers $k > 0$, $m \geq 0$, $n \geq 0$, we have $\omega_{kn}^{km} = \omega_n^m$.

Proof.

$$\omega_{kn}^{km} = (\omega_{kn})^{km} = (e^{i\frac{2\pi}{kn}})^{km} = e^{i\frac{2\pi km}{kn}} = e^{i\frac{2\pi m}{n}} = (e^{i\frac{2\pi}{n}})^m = \omega_n^m$$

□

- Thus, in particular, $(\omega_{2n}^k)^2 = \omega_{2n}^{2k} = (\omega_{2n}^2)^k = \omega_n^k$;
- So the squares of the roots of unity of order $2n$ are just the roots of unity of order n.

The Discrete Fourier Transform

- Let $\mathbf{a} = \langle a_0, a_1, \ldots, a_{n-1} \rangle$ be a sequence of n real or complex numbers.
- We can form the corresponding polynomial $P_A(x) = \sum_{j=0}^{n-1} a_j x^j$,
- We can evaluate it at all complex roots of unity of order n, i.e., we compute $P_A(\omega_n^k)$ for all $0 \leq k \leq n-1$.

Discrete Fourier Transform (DFT)

The **Discrete Fourier Transform (DFT)** of a sequence $\mathbf{a} = \langle a_0, a_1, \ldots, a_{n-1} \rangle$ is the sequence of values $\mathbf{A} = \langle A_0, A_1, \ldots, A_{n-1} \rangle = \langle P_A(1), P_A(\omega_n), P_A(\omega_n^2), \ldots, P_A(\omega_n^{n-1}) \rangle$.

- The DFT $\mathbf{A}$ of a sequence $\mathbf{a}$ can be computed VERY FAST using a divide-and-conquer algorithm called the **Fast Fourier Transform**.

New way for fast multiplication of polynomials

- If we multiply a polynomial

$$P_A(x) = a_0 + \ldots + a_{n-1}x^{n-1}$$

of degree $n-1$ with a polynomial

$$P_B(x) = b_0 + \ldots + b_{m-1}x^{m-1}$$

of degree $m-1$ we get a polynomial

$$C(x) = P_A(x)P_B(x) = c_0 + \ldots + c_{m+n-2}x^{m+n-2}$$

of degree $n-1+m-1 = m+n-2$ with $m+n-1$ coefficients.

- To uniquely determine such a polynomial $C(x)$ of degree $m+n-2$ we need $m+n-1$ many values.

- Thus, we will evaluate both $P_A(x)$ and $P_B(x)$ at all the roots of unity of order $n+m-1$ (instead of at $-(n-1), \ldots, -1, 0, 1, \ldots, m-1$ as we would in Karatsuba's method!)

New way for fast multiplication of polynomials

- Note that we defined the DFT of a sequence of length n as the values of the corresponding polynomial of degree $n-1$ at the n roots of unity of order n, i.e., ω_n^k $(0 \leq k \leq n-1)$.
- So the DFT of a sequence **a** is another sequence **A** of exactly the same length; for simplicity, we do not consider an operation which would evaluate a polynomial of degree $n-1$ at m roots of unity of order $m \neq n$.
- For that reason, in order to be able to call a function for evaluating the DFT and since we need $n+m-1$ values of $P_C(x) = P_A(x)P_B(x)$, we pad **a** with $m-1$ zeros at the end, $(a_0, a_1, \ldots, a_{n-1}, \underbrace{0, \ldots, 0}_{m-1})$ to make it of length $m+n-1$, and similarly we pad **b** with $n-1$ zeros at the end, $(b_0, b_1, \ldots, b_{m-1}, \underbrace{0, \ldots, 0}_{n-1})$ to also obtain a sequence of length $n+m-1$.

New way for fast multiplication of polynomials

- We can now compute the DFTs of the two (0 padded) sequences:

$$DFT(\langle a_0, a_1, \ldots, a_{n-1}, \underbrace{0, \ldots, 0}_{m-1}\rangle) = \langle A_0, A_1, \ldots, A_{n+m-2}\rangle$$

and

$$DFT(\langle b_0, b_1, \ldots, b_{m-1}, \underbrace{0, \ldots, 0}_{n-1}\rangle) = \langle B_0, B_1, \ldots, B_{n+m-2}\rangle$$

- For each k we multiply the corresponding values $A_k = P_A(\omega^k_{n+m-1})$ and $B_k = P_B(\omega^k_{n+m-1})$, thus obtaining

$$C_k = A_k B_k = P_A(\omega^k_{n+m-1}) P_B(\omega^k_{n+m-1}) = P_C(\omega^k_{n+m-1})$$

- We then use the inverse transformation for DFT, called IDFT, to recover the coefficients $\langle c_0, c_1, \ldots, c_{n+m-1}\rangle$ of the product polynomial $P_C(x)$ from the sequence $\langle C_0, C_1, \ldots, C_{n+m-1}\rangle$ of its values $C_k = P_C(\omega^k_{n+m-1})$ at the roots of unity of order $n + m - 1$.

New way for fast multiplication of polynomials

$$P_A(x) = a_0 + \ldots + a_{n-1}x^{n-1} + 0 \cdot x^n + \ldots + 0 \cdot x^{n+m-2};$$

$$P_B(x) = b_0 + \ldots + b_{m-1}x^{m-1} + 0 \cdot x^m + \ldots + 0 \cdot x^{n+m-2}$$

$\Downarrow$ DFT $\qquad\qquad$ $\Downarrow$ DFT

$$\{P_A(1), P_A(\omega_{n+m-1}), \ldots, P_A(\omega_{n+m-1}^{n+m-2})\}; \quad \{P_B(1), P_B(\omega_{n+m-1}), \ldots, P_B(\omega_{n+m-1}^{n+m-2})\}$$

$\Downarrow$ multiplication

$$\Big\{ \underbrace{P_A(1)P_B(1)}_{P_C(1)}, \quad \underbrace{P_A(\omega_{n+m-1})P_B(\omega_{n+m-1})}_{P_C(\omega_{n+m-1})}, \ldots, \underbrace{P_A(\omega_{n+m-1}^{n+m-2})P_B(\omega_{n+m-1}^{n+m-2})}_{P_C(\omega_{n+m-1}^{n+m-2})} \Big\}$$

$\Downarrow$ IDFT

$$P_C(x) = P_A(x) \cdot P_B(x) = \sum_{j=0}^{n+m-2} c_j x^j = \sum_{j=0}^{n+m-2} \Big(\underbrace{\sum_{i=0}^{j} a_i b_{j-i}}_{c_j} \Big) x^j$$

Fast multiplication of polynomials

- For simplicity of notation let us assume that polynomials $P_A(x)$ and $P_B(x)$ are of the same degree $n-1$.
- In fact, if they are not, we can pad the one of a lower degree $m < n$ with higher degrees coefficients 0 to make it appear also of degree $n-1$.
- Multiplying $2n-1$ values of $P_A(\omega_{2n-1}^k)$ and $P_B(\omega_{2n-1}^k)$ is done in linear time.

- So we have to find an efficient way to compute DFT and IDFT.

- For each fixed k we need to evaluate

$$P_A(\omega_{2n-1}^k) = a_0 + a_1\omega_{2n-1}^k + a_2\omega_{2n-1}^{2k} + \ldots + a_{n-1}\omega_{2n-1}^{(n-1)k}$$

$$P_B(\omega_{2n-1}^k) = b_0 + b_1\omega_{2n-1}^k + b_2\omega_{2n-1}^{2k} + \ldots + b_{n-1}\omega_{2n-1}^{(n-1)k}$$

- We could precompute all of the values ω_{2n-1}^k, but, by brute force, for each k we would have to do $n-1$ multiplications of the form $a_m \cdot \omega_{2n-1}^{km}$, for $1 \leq m < n-1$.

- Thus, since k ranges from 1 to $2n-1$ ($k=0$ does not involve any multiplications) and since m ranges from 1 to $n-1$ ($m=0$ also does not involve a multiplication), we would have to do $O((2n-1)(n-1)) = O(n^2)$ multiplications.

- **Can we do it faster??**This is precisely what the **Fast Fourier Transform (FFT)** does; it computes the values $P_A(\omega_n^k)$ for all k such that $0 \leq k < n$ in $\mathbf{O(n \log n)}$ time.

The Fast Fourier Transform (FFT)

- Let $P_A(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$;
  - We can assume that n is a power of 2—otherwise we can pad $P_A(x)$ with zero coefficients until its number of coefficients becomes equal to the nearest power of 2.
  - Exercise: Show that for every n which is not a power of two the smallest power of 2 larger or equal to n is smaller than $2n$.
  - *Hint:* consider n in binary. How many bits does the nearest power of two have?

The Fast Fourier Transform (FFT)

- **The main idea of the FFT algorithm:** divide-and-conquer by splitting the polynomial into the even powers and the odd powers:

$$\begin{aligned} P_A(x) &= (a_0 + a_2x^2 + a_4x^4 + \ldots + a_{n-2}x^{n-2}) + (a_1x + a_3x^3 + \ldots a_{n-1}x^{n-1}) \\ &= a_0 + a_2x^2 + a_4(x^2)^2 + \ldots + a_{n-2}(x^2)^{\frac{n}{2}-1} \\ &\qquad + x\left(a_1 + a_3x^2 + \ldots + a_{n-1}(x^2)^{\frac{n}{2}-1}\right) \end{aligned}$$

- Let us define

$$\begin{aligned} \mathbf{a}^{[0]}(y) &= a_0 + a_2y + a_4y^2 + \ldots + a_{n-2}y^{\frac{n}{2}-1} \\ \mathbf{a}^{[1]}(y) &= a_1 + a_3y + a_5y^2 + \ldots + a_{n-1}y^{\frac{n}{2}-1} \end{aligned}$$

- Then $P_A(x) = \mathbf{a}^{[0]}(x^2) + x\,\mathbf{a}^{[1]}(x^2)$
- Note that the number of coefficients of the polynomials $\mathbf{a}^{[0]}(y)$ and $\mathbf{a}^{[1]}(y)$ is $n/2$ each, while the number of coefficients of the polynomial $P_A(x)$ is n. Thus, the number of coefficients of each of the two polynomials $\mathbf{a}^{[0]}(y)$ and $\mathbf{a}^{[1]}(y)$ is only one half of the number of coefficients of the polynomial $P_A(x)$.

The Fast Fourier Transform (FFT)

- **Problem of size n:**
  Evaluate a polynomial with n coefficients at n many roots of unity.

- **Problem of size $n/2$:**
  Evaluate a polynomial with $n/2$ coefficients at $n/2$ many roots of unity.

- We reduced evaluation of our polynomial $P_A(x)$ with n coefficients at inputs $x = \omega_n^0,\ x = \omega_n^1,\ x = \omega_n^2, \ldots, x = \omega_n^{n-1}$ to evaluation of two polynomials $\mathbf{a}^{[0]}(y)$ and $\mathbf{a}^{[1]}(y)$ each with $n/2$ coefficients, at points $y = x^2$ for the same values of inputs x.

- However, as x ranges through values $\{\omega_n^0, \omega_n^1, \omega_n^2, \ldots, \omega_n^{n-1}\}$, the value of $y = x^2$ ranges through $\{\omega_{\frac{n}{2}}^0, \omega_{\frac{n}{2}}^1, \omega_{\frac{n}{2}}^2, \ldots, \omega_{\frac{n}{2}}^{n-1}\}$, and **there are only $n/2$ distinct such values**.

- Once we get these $n/2$ values of $\mathbf{a}^{[0]}(x^2)$ and $\mathbf{a}^{[1]}(x^2)$ we need n additional multiplications with numbers ω_n^k to obtain the values of

$$\begin{aligned} P_A(\omega_n^k) &= \mathbf{a}^{[0]}((\omega_n^k)^2) + \omega_n^k \cdot \mathbf{a}^{[1]}((\omega_n^k)^2) \\ &= \mathbf{a}^{[0]}(\omega_{n/2}^k) + \omega_n^k \cdot \mathbf{a}^{[1]}(\omega_{n/2}^k). \end{aligned}$$

- Thus, we have reduced a problem of size n to two such problems of size $n/2$, plus a linear overhead.

The Fast Fourier Transform (FFT)—a simplification

- Note that by the Cancelation Lemma $\omega_n^{\frac{n}{2}} = \omega_{2\frac{n}{2}}^{\frac{n}{2}} = \omega_2 = -1$
- Furthermore

$$\omega_n^{\frac{n}{2}+k} = \omega_n^{\frac{n}{2}}\,\omega_n^k = \omega_2\omega_n^k = -\omega_n^k \tag{3}$$

$$(\omega_n^k)^2 = \omega_{\frac{n}{2}}^k \tag{4}$$

$$(-\omega_n^k)^2 = (-1)^2(\omega_n^k)^2 = \omega_{\frac{n}{2}}^k \tag{5}$$

- We can now simplify the evaluation of $P_A(\omega_n^k)$ for $0 \le k < n/2$ as follows:

$$\begin{aligned} P_A(\omega_n^k) &= \mathbf{a}^{[0]}((\omega_n^k)^2) + \omega_n^k \mathbf{a}^{[1]}((\omega_n^k)^2) \\ &= \mathbf{a}^{[0]}(\omega_{n/2}^k) + \omega_n^k \mathbf{a}^{[1]}(\omega_{n/2}^k) \qquad \text{Using Eq. (4)} \end{aligned}$$

- We can now simplify the evaluation of $P_A(\omega_n^{\frac{n}{2}+k})$ for $n/2 \le \frac{n}{2} + k < n$ as follows:

$$\begin{aligned} P_A(\omega_n^{\frac{n}{2}+k}) &= \mathbf{a}^{[0]}((\omega_n^{\frac{n}{2}+k})^2) + \omega_n^{\frac{n}{2}+k}\mathbf{a}^{[1]}((\omega_n^{\frac{n}{2}+k})^2) \\ &= \mathbf{a}^{[0]}((-\omega_n^k)^2) - \omega_n^k \mathbf{a}^{[1]}((-\omega_{n/2}^k)^2) \qquad \text{Using Eq. (3)} \\ &= \mathbf{a}^{[0]}(\omega_{n/2}^k) - \omega_n^k \mathbf{a}^{[1]}(\omega_{n/2}^k) \qquad \text{Using Eq. (5)} \end{aligned}$$

The Fast Fourier Transform (FFT)—a simplification

- So we can replace evaluations of

$$\begin{aligned} P_A(\omega_n^k) &= \mathbf{a}^{[0]}((\omega_n^k)^2) + \omega_n^k \mathbf{a}^{[1]}((\omega_n^k)^2) \\ &= \mathbf{a}^{[0]}(\omega_{n/2}^k) + \omega_n^k \mathbf{a}^{[1]}(\omega_{n/2}^k) \end{aligned}$$

  for $k = 0$ to $n - 1$

  with such evaluations only for $k = 0$ to $n/2 - 1$

  and just let for $k = 0$ to $n/2 - 1$

$$\begin{aligned} P_A(\omega_n^{\frac{n}{2}+k}) &= \mathbf{a}^{[0]}((\omega_n^k)^2) - \omega_n^k \mathbf{a}^{[1]}((\omega_n^k)^2) \\ &= \mathbf{a}^{[0]}(\omega_{n/2}^k) - \omega_n^k \mathbf{a}^{[1]}(\omega_{n/2}^k) \end{aligned}$$

- We can now write a pseudo-code for our FFT algorithm:

FFT algorithm

`fft(a)`

$n \leftarrow \text{length}[\mathbf{a}]$

if $n = 1$ **then return a**

$\mathbf{a}^{[0]} \leftarrow (a_0, a_2, \ldots a_{n-2})$

$\mathbf{a}^{[1]} \leftarrow (a_1, a_3, \ldots a_{n-1})$

$y^{[0]} \leftarrow$ `fft(`$\mathbf{a}^{[0]}$`)`

$y^{[1]} \leftarrow$ `fft(`$\mathbf{a}^{[1]}$`)`

$\omega \leftarrow 1$

for $k = 0$ **to** $\frac{n}{2} - 1$ **do**

$A_k \leftarrow y_k^{[0]} + \omega \cdot y_k^{[1]}$

$A_{\frac{n}{2}+k} \leftarrow y_k^{[0]} - \omega \cdot y_k^{[1]}$

$\omega \leftarrow \omega \cdot e^{i\frac{2\pi}{n}}$

return A

How fast is the Fast Fourier Transform?

- We have recursively reduced evaluation of a polynomial $P_A(x)$ with n coefficients at n roots of unity of order n to evaluations of two polynomials $A^{[0]}(y)$ and $A^{[1]}(y)$, each with $n/2$ coefficients, at $n/2$ many roots of unity of order $n/2$.
- Once we get these $n/2$ values of $\mathbf{a}^{[0]}(y)$ and $\mathbf{a}^{[1]}(y)$ we need $n/2$ additional multiplications to obtain the values of

$$\underbrace{P_A(\omega_n^k)}_{A_k} = \underbrace{\mathbf{a}^{[0]}(\omega_{n/2}^k)}_{y_k^{[0]}} + \omega_n^k \underbrace{\mathbf{a}^{[1]}(\omega_{n/2}^k)}_{y_k^{[1]}} \tag{6}$$

and

$$\underbrace{P_A(\omega_n^{\frac{n}{2}+k})}_{A_{\frac{n}{2}+k}} = \underbrace{\mathbf{a}^{[0]}(\omega_{n/2}^k)}_{y_k^{[0]}} - \omega_n^k \underbrace{\mathbf{a}^{[1]}(\omega_{n/2}^k)}_{y_k^{[1]}} \tag{7}$$

for all $0 \leq k < n/2$

- Thus, we have reduced a problem of size n to two such problems of size $n/2$, plus a linear overhead.
- Consequently, our algorithm's run time satisfies the recurrence

$$T(n) = 2\,T\left(\frac{n}{2}\right) + c\,n$$

- The Master Theorem gives $T(n) = \Theta(n \log n)$.

Matrix representation of polynomial evaluation

- Evaluation of a polynomial $P_A(x) = a_0 + a_1 x + \ldots + a_{n-1}x^{n-1}$ at roots of unity ω_n^k of order n can be represented in the matrix form as follows:

$$\begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega_n & \omega_n^2 & \ldots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^{2\cdot 2} & \ldots & \omega_n^{2\cdot(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \ldots & \omega_n^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} P_A(1) \\ P_A(\omega_n) \\ P_A(\omega_n^2) \\ \vdots \\ P_A(\omega_n^{n-1}) \end{pmatrix} = \begin{pmatrix} A_0 \\ A_1 \\ A_2 \\ \vdots \\ A_{n-1} \end{pmatrix}$$

- The FFT is just a method of replacing this matrix-vector multiplication taking n^2 many multiplications with an $n \log n$ procedure.
- From $P_A(1) = P_A(\omega_n^0), \quad P_A(\omega_n), \; P_A(\omega_n^2), \ldots, P_A(\omega_n^{n-1})$, we get the coefficients from

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega_n & \omega_n^2 & \ldots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^{2\cdot 2} & \ldots & \omega_n^{2\cdot(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \ldots & \omega_n^{(n-1)(n-1)} \end{pmatrix}^{-1} \begin{pmatrix} P_A(1) \\ P_A(\omega_n) \\ P_A(\omega_n^2) \\ \vdots \\ P_A(\omega_n^{n-1}) \end{pmatrix}$$

Another remarkable feature of the roots of unity:

- To obtain the inverse of the above matrix, all we have to do is just change the signs of the exponents and divide everything by n:

$$\begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n & \omega_n^2 & \dots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^{2\cdot 2} & \dots & \omega_n^{2\cdot(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \dots & \omega_n^{(n-1)(n-1)} \end{pmatrix}^{-1} =$$

$$\frac{1}{n}\begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n^{-1} & \omega_n^{-2} & \dots & \omega_n^{-(n-1)} \\ 1 & \omega_n^{-2} & \omega_n^{-2\cdot 2} & \dots & \omega_n^{-2\cdot(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega_n^{-(n-1)} & \omega_n^{-2(n-1)} & \dots & \omega_n^{-(n-1)(n-1)} \end{pmatrix}$$

To see this, note that if we compute the product

$$\begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n & \omega_n^2 & \dots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^{2\cdot 2} & \dots & \omega_n^{2\cdot(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \dots & \omega_n^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n^{-1} & \omega_n^{-2} & \dots & \omega_n^{-(n-1)} \\ 1 & \omega_n^{-2} & \omega_n^{-2\cdot 2} & \dots & \omega_n^{-2\cdot(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega_n^{-(n-1)} & \omega_n^{-2(n-1)} & \dots & \omega_n^{-(n-1)(n-1)} \end{pmatrix}$$

the (i, j) entry in the product matrix is equal to a product of i^{th} row and j^{th} column:

$$\begin{pmatrix} 1 & \omega_n^i & \omega_n^{2\cdot i} & \dots & \omega_n^{i\cdot(n-1)} \end{pmatrix} \begin{pmatrix} 1 \\ \omega_n^{-j} \\ \omega_n^{-2j} \\ \vdots \\ \omega_n^{-(n-1)j} \end{pmatrix} = \sum_{k=0}^{n-1} \omega_n^{ik}\omega_n^{-jk} = \sum_{k=0}^{n-1} \omega_n^{(i-j)k}$$

We now have two possibilities:

1. $i = j$: then
$$\sum_{k=0}^{n-1} \omega_n^{(i-j)k} = \sum_{k=0}^{n-1} \omega_n^0 = \sum_{k=0}^{n-1} 1 = n;$$

2. $i \neq j$: then $\sum_{k=0}^{n-1} \omega_n^{(i-j)k}$ represents a geometric series with the ratio ω_n^{i-j} and thus
$$\sum_{k=0}^{n-1} \omega_n^{(i-j)k} = \frac{1 - \omega_n^{(i-j)n}}{1 - \omega_n^{i-j}} = \frac{1 - (\omega_n^n)^{i-j}}{1 - \omega_n^{i-j}} = \frac{1-1}{1 - \omega_n^{i-j}} = 0$$

So,

$$\begin{pmatrix} 1 & \omega_n^i & \omega_n^{2\cdot i} & \dots & \omega_n^{i\cdot(n-1)} \end{pmatrix} \begin{pmatrix} 1 \\ \omega_n^{-j} \\ \omega_n^{-2j} \\ \vdots \\ \omega_n^{-(n-1)j} \end{pmatrix} = \sum_{k=0}^{n-1} \omega_n^{(i-j)k} = \begin{cases} n & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{8}$$

So we get:

$$\begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n & \omega_n^2 & \dots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^{2\cdot 2} & \dots & \omega_n^{2\cdot(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \dots & \omega_n^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n^{-1} & \omega_n^{-2} & \dots & \omega_n^{-(n-1)} \\ 1 & \omega_n^{-2} & \omega_n^{-2\cdot 2} & \dots & \omega_n^{-2\cdot(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega_n^{-(n-1)} & \omega_n^{-2(n-1)} & \dots & \omega_n^{-(n-1)(n-1)} \end{pmatrix}$$

$$= \begin{pmatrix} n & 0 & 0 & \dots & 0 \\ 0 & n & 0 & \dots & 0 \\ 0 & 0 & n & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & n \end{pmatrix} = n \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{pmatrix}$$

i.e.,

$$\begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n & \omega_n^2 & \dots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^{2\cdot 2} & \dots & \omega_n^{2\cdot(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \dots & \omega_n^{(n-1)(n-1)} \end{pmatrix}^{-1} = \frac{1}{n} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n^{-1} & \omega_n^{-2} & \dots & \omega_n^{-(n-1)} \\ 1 & \omega_n^{-2} & \omega_n^{-2\cdot 2} & \dots & \omega_n^{-2\cdot(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega_n^{-(n-1)} & \omega_n^{-2(n-1)} & \dots & \omega_n^{-(n-1)(n-1)} \end{pmatrix}$$

- We now have

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n & \omega_n^2 & \dots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^{2\cdot 2} & \dots & \omega_n^{2\cdot(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \dots & \omega_n^{(n-1)(n-1)} \end{pmatrix}^{-1} \begin{pmatrix} P_A(1) \\ P_A(\omega_n) \\ P_A(\omega_n^2) \\ \vdots \\ P_A(\omega_n^{n-1}) \end{pmatrix} =$$

$$= \frac{1}{n} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n^{-1} & \omega_n^{-2} & \dots & \omega_n^{-(n-1)} \\ 1 & \omega_n^{-2} & \omega_n^{-2\cdot 2} & \dots & \omega_n^{-2\cdot(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega_n^{-(n-1)} & \omega_n^{-2(n-1)} & \dots & \omega_n^{-(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} P_A(1) \\ P_A(\omega_n) \\ P_A(\omega_n^2) \\ \vdots \\ P_A(\omega_n^{n-1}) \end{pmatrix}$$

- This means that to covert from the values

$$\langle P_A(1), P_A(\omega_n), P_A(\omega_n^2), \dots, P_A(\omega_n^{n-1}) \rangle = \langle A_0, A_1, A_2, \dots, A_{n-1} \rangle$$

back to the coefficient form

$$P_A(x) = a_0 + a_1 x + a_2 x^2 + a_{n-1} x^{n-1}$$

we can use **the same** FFT algorithm with the only change that:

1. the root of unity ω_n is replaced by $\overline{\omega_n} = \omega_n^{-1} = e^{-i\frac{2\pi}{n}}$,
2. the resulting output values are divided by n.

Inverse Fast Fourier Transform (IFFT):

```
ifft*(A)
    n ← length[A]
    if n = 1 then return A
    A^[0] ← (A_0, A_2, ... A_{n-2})
    A^[1] ← (A_1, A_3, ... A_{n-1})
    y^[0] ← ifft*(A^[0])
    y^[1] ← ifft*(A^[1])
    ω ← 1
    for k = 0 to n/2 − 1 do
        a_k ← y_k^[0] + ω · y_k^[1]
        a_{n/2+k} ← y_k^[0] − ω · y_k^[1]
        ω ← ω · e^{−i 2π/n}                    /* ⇐  different from FFT */
    return y

ifft(A)                                        /* ⇐  different from FFT */
    return ifft*(A / length(A))
```

More on DFT

- We have followed the textbook (CLRS).
- However, what CLRS calls DFT, namely, the sequence

$$\langle P_A(\omega_n^0), P_A(\omega_n^1), P_A(\omega_n^2), \ldots, P_A(\omega_n^{n-1}) \rangle$$

  is usually considered the Inverse Discrete Fourier Transform (IDFT) of the sequence of the coefficients

$$\langle a_0, a_1, a_2, \ldots, a_{n-1} \rangle$$

  of the polynomial $P_A(x)$;
- Instead,

$$\langle P_A(\omega_n^0), P_A(\omega_n^{-1}), P_A(\omega_n^{-2}), \ldots, P_A(\omega_n^{-(n-1)}) \rangle$$

  is considered the "forward operation" i.e., the DFT.
- Taking this as the "forward operation" has an important conceptual advantage and is used more often than the textbook's choice, especially in electrical engineering literature.

More on DFT: Another "tweak"

$$\begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n & \omega_n^2 & \dots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^{2\cdot 2} & \dots & \omega_n^{2\cdot(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \dots & \omega_n^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n^{-1} & \omega_n^{-2} & \dots & \omega_n^{-(n-1)} \\ 1 & \omega_n^{-2} & \omega_n^{-2\cdot 2} & \dots & \omega_n^{-2\cdot(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega_n^{-(n-1)} & \omega_n^{-2(n-1)} & \dots & \omega_n^{-(n-1)(n-1)} \end{pmatrix} = n \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{pmatrix}$$

implies:

$$\begin{pmatrix} \frac{1}{\sqrt{n}} & \frac{1}{\sqrt{n}} & \frac{1}{\sqrt{n}} & \dots & \frac{1}{\sqrt{n}} \\ \frac{1}{\sqrt{n}} & \frac{\omega_n}{\sqrt{n}} & \frac{\omega_n^2}{\sqrt{n}} & \dots & \frac{\omega_n^{n-1}}{\sqrt{n}} \\ \frac{1}{\sqrt{n}} & \frac{\omega_n^2}{\sqrt{n}} & \frac{\omega_n^{2\cdot 2}}{\sqrt{n}} & \dots & \frac{\omega_n^{2\cdot(n-1)}}{\sqrt{n}} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{1}{\sqrt{n}} & \frac{\omega_n^{n-1}}{\sqrt{n}} & \frac{\omega_n^{2(n-1)}}{\sqrt{n}} & \dots & \frac{\omega_n^{(n-1)(n-1)}}{\sqrt{n}} \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{n}} & \frac{1}{\sqrt{n}} & \frac{1}{\sqrt{n}} & \dots & \frac{1}{\sqrt{n}} \\ \frac{1}{\sqrt{n}} & \frac{\omega_n^{-1}}{\sqrt{n}} & \frac{\omega_n^{-2}}{\sqrt{n}} & \dots & \frac{\omega_n^{-(n-1)}}{\sqrt{n}} \\ \frac{1}{\sqrt{n}} & \frac{\omega_n^{-2}}{\sqrt{n}} & \frac{\omega_n^{-2\cdot 2}}{\sqrt{n}} & \dots & \frac{\omega_n^{-2\cdot(n-1)}}{\sqrt{n}} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{1}{\sqrt{n}} & \frac{\omega_n^{-(n-1)}}{\sqrt{n}} & \frac{\omega_n^{-2(n-1)}}{\sqrt{n}} & \dots & \frac{\omega_n^{-(n-1)(n-1)}}{\sqrt{n}} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{pmatrix}$$

Thus, these two matrices are inverses of each other.

More on DFT

- This motivates us to "tweak" the definition of DFT:
- Given a sequence of numbers $\mathbf{a} = (a_0, a_1, \ldots, a_{n-1})$ the Discrete Fourier Transform of this sequence is the sequence of the values of the polynomial

$$P_A^*(x) = \frac{1}{\sqrt{n}} P_A(x) = \frac{1}{\sqrt{n}} \left(a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}\right)$$

for $x = \omega_n^{-k}$ for $k = 0, \ldots, n-1$; i.e., the sequence of values $P_A^*(\omega_n^{-k})$:

$$P_A^*(\omega_n^{-k}) = \frac{1}{\sqrt{n}} \left(a_0 + a_1 (\omega_n^{-k})^1 + \ldots + a_{n-1} (\omega_n^{-k})^{n-1}\right)$$

- Given a sequence of numbers $(a_0, a_1, \ldots, a_{n-1})$ the **Inverse Discrete Fourier Transform** of this sequence is the sequence of the values of the same polynomial

$$P_A^*(x) = \frac{1}{\sqrt{n}} \left(a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}\right)$$

but for $x = \omega_n^k$ for $k = 0, \ldots, n-1$; i.e., the sequence of values $P_A^*(\omega_n^k)$

$$P_A^*(\omega_n^k) = \frac{1}{\sqrt{n}} \left(a_0 + a_1 (\omega_n^k)^1 + \ldots + a_{n-1} (\omega_n^k)^{n-1}\right)$$

More on DFT

fft*(a)

$n \leftarrow \text{length}[\mathbf{a}]$
if $n = 1$ **then return a**
$\mathbf{a}^{[0]} \leftarrow (a_0, a_2, \ldots a_{n-2})$
$\mathbf{a}^{[1]} \leftarrow (a_1, a_3, \ldots a_{n-1})$
$y^{[0]} \leftarrow$ **fft***$(\mathbf{a}^{[0]})$
$y^{[1]} \leftarrow$ **fft***$(\mathbf{a}^{[1]})$
$\omega \leftarrow 1$
for $k = 0$ **to** $\frac{n}{2} - 1$ **do**
$A_k \leftarrow y_k^{[0]} + \omega \cdot y_k^{[1]}$
$A_{\frac{n}{2}+k} \leftarrow y_k^{[0]} - \omega \cdot y_k^{[1]}$
$\omega \leftarrow \omega \cdot e^{-i\frac{2\pi}{n}}$
return A

fft(a)

return **fft***$\left(\frac{\mathbf{a}}{\sqrt{\text{length}(\mathbf{a})}}\right)$

ifft*(A)

$n \leftarrow \text{length}[\mathbf{A}]$
if $n = 1$ **then return A**
$\mathbf{A}^{[0]} \leftarrow (A_0, A_2, \ldots A_{n-2})$
$\mathbf{A}^{[1]} \leftarrow (A_1, A_3, \ldots A_{n-1})$
$y^{[0]} \leftarrow$ **ifft***$(\mathbf{A}^{[0]})$
$y^{[1]} \leftarrow$ **ifft***$(\mathbf{A}^{[1]})$
$\omega \leftarrow 1$
for $k = 0$ **to** $\frac{n}{2} - 1$ **do**
$a_k \leftarrow y_k^{[0]} + \omega \cdot y_k^{[1]}$
$a_{\frac{n}{2}+k} \leftarrow y_k^{[0]} - \omega \cdot y_k^{[1]}$
$\omega \leftarrow \omega \cdot e^{i\frac{2\pi}{n}}$
return a

ifft(A)

return **ifft***$\left(\frac{\mathbf{A}}{\sqrt{\text{length}(\mathbf{A})}}\right)$

Back to fast multiplication of polynomials

$P_A(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$ $\qquad$ $P_B(x) = b_0 + b_1 x + \ldots + b_{n-1} x^{n-1}$

$\Downarrow$ DFT $\quad O(n \log n)$ $\qquad$ $\Downarrow$ DFT $\quad O(n \log n)$

$\{P_A(1), P_A(\omega_{2n-1}), P_A(\omega_{2n-1}^2), \ldots, P_A(\omega_{2n-1}^{2n-2})\}$; $\quad$ $\{P_B(1), P_B(\omega_{2n-1}), P_B(\omega_{2n-1}^2), \ldots, P_B(\omega_{2n-1}^{2n-2})\}$

$\Downarrow$ multiplication $\quad O(n)$

$\{P_A(1)P_B(1), \quad P_A(\omega_{2n-1})P_B(\omega_{2n-1}), \ldots, P_A(\omega_{2n-1}^{2n-2})P_B(\omega_{2n-1}^{2n-2})\}$

$\Downarrow$ IDFT $\quad O(n \log n)$

$$P_C(x) = \sum_{j=0}^{2n-2} \Bigg(\underbrace{\sum_{i=0}^{j} a_i b_{j-i}}_{c_j} \Bigg) x^j = \sum_{j=0}^{2n-2} c_j x^j = P_A(x) \cdot P_B(x)$$

Thus, the product $P_C(x) = P_A(x)\, P_B(x)$ of two polynomials $P_A(x)$ and $P_B(x)$ can be computed in time $O(n \log n)$.

Computing the convolution $C = A * B$

$$\mathbf{a} = \langle a_0, a_1, \ldots, a_{n-1} \rangle \qquad \mathbf{b} = \langle b_0, b_1, \ldots, b_{n-1} \rangle$$

$$\Downarrow \quad O(n) \qquad \Downarrow \quad O(n)$$

$$P_A(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1} \qquad P_B(x) = b_0 + b_1 x + \ldots + b_{n-1} x^{n-1}$$

$$\Downarrow \text{ DFT} \quad O(n \log n) \qquad \Downarrow \text{ DFT} \quad O(n \log n)$$

$$\{P_A(1), P_A(\omega_{2n-1}), P_A(\omega_{2n-1}^2), \ldots, P_A(\omega_{2n-1}^{2n-2})\}; \quad \{P_B(1), P_B(\omega_{2n-1}), P_B(\omega_{2n-1}^2), \ldots, P_B(\omega_{2n-1}^{2n-2})\}$$

$$\Downarrow \text{ multiplication} \quad O(n)$$

$$\{P_A(1)P_B(1), \quad P_A(\omega_{2n-1})P_B(\omega_{2n-1}), \ldots, P_A(\omega_{2n-1}^{2n-2})P_B(\omega_{2n-1}^{2n-2})\}$$

$$\Downarrow \text{ IDFT} \quad O(n \log n)$$

$$P_C(x) = \sum_{j=0}^{2n-2} \Bigg(\underbrace{\sum_{i=0}^{j} a_i b_{j-i}}_{c_j} \Bigg) x^j$$

$$\Downarrow$$

$$\mathbf{c} = \left\langle \sum_{i=0}^{j} a_i b_{j-i} \right\rangle_{j=0}^{j=2n-2}$$

Convolution $\mathbf{c} = \mathbf{a} * \mathbf{b}$ of sequences $\mathbf{a}$ and $\mathbf{b}$ is computed in time $O(n \log n)$.

Computing the convolution $C = A * B$

IMPORTANT: If you are using the normalised FFT which evaluates polynomials $P_A^*(x) = \frac{1}{\sqrt{N}} P_A(x)$ and $P_B^*(x) = \frac{1}{\sqrt{N}} P_B(x)$ instead of polynomial $P_A(x)$ and $P_B(x)$, then when you multiply them point-wise you get

$$P_A^*(\omega_n^k)\, P_B^*(\omega_n^k) = \frac{1}{\sqrt{N}} P_A(\omega_n^k) \frac{1}{\sqrt{N}} P_B(\omega_n^k) = \frac{1}{N} P_A(\omega_n^k) P_B(\omega_n^k)$$

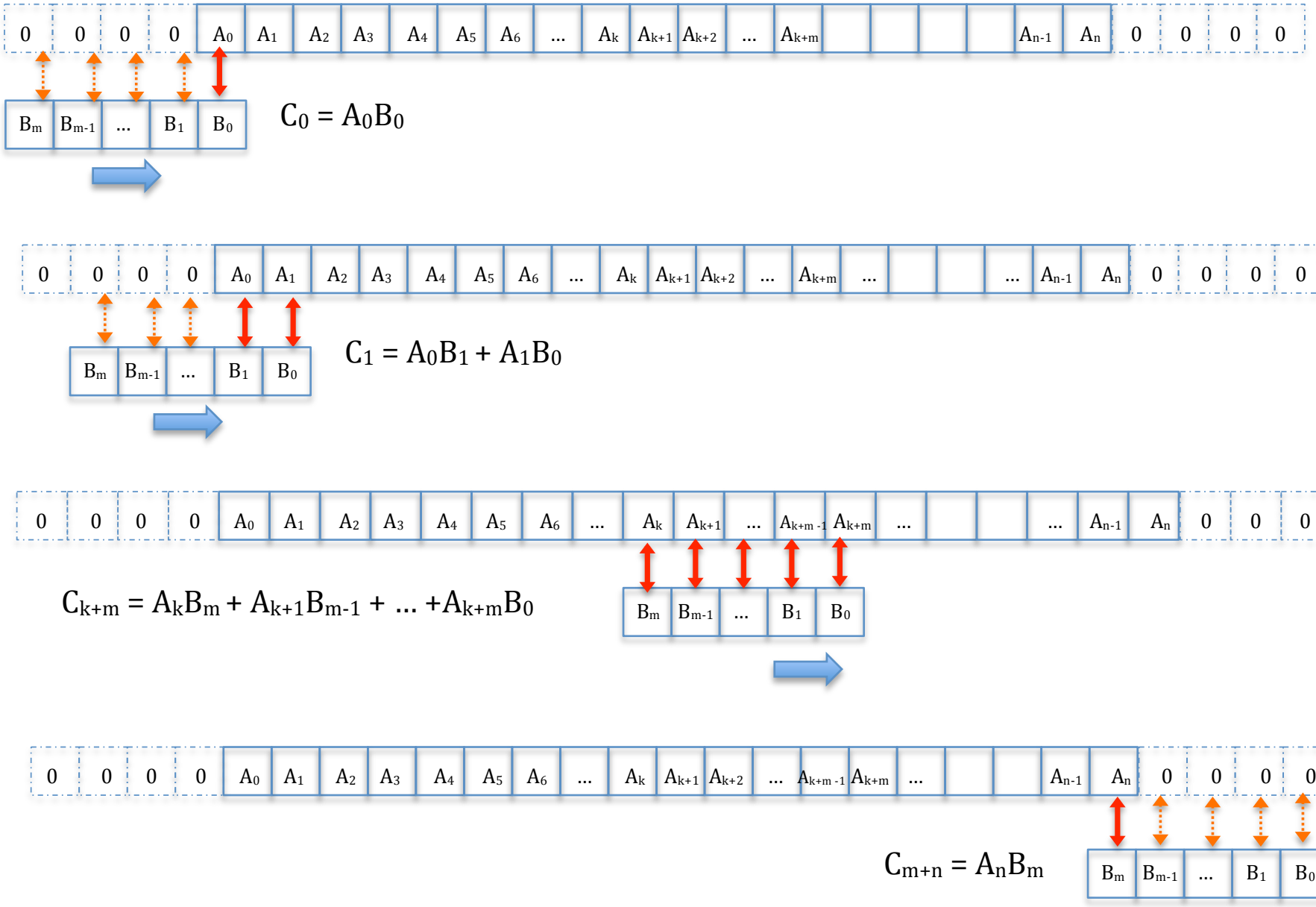
which the Inverse FFT would multiply with another $\frac{1}{\sqrt{N}}$ so your convolution will end up scaled by $\frac{1}{\sqrt{N}}$. To avoid this, if you are using the normalised FFT to compute the DFT, the correct formula for computing a convolution via such FFT is

$$\mathbf{a} * \mathbf{b} = \sqrt{N} \cdot \text{IFFT}(\text{FFT}(\mathbf{a}) \cdot \text{FFT}(\mathbf{b}) \cdot).$$

This is how e.g., *Mathematica* evaluates convolution.

Also, convolution is defined for sequences of numbers **a** and **b**, and some of these numbers can be 0, including those at the right end of **a** or **b**. The convolution of a sequence **a** of length n with a sequence **b** of length m is defined to always be a sequence of length $n + m - 1$. Thus, if the largest power in $P_C(x)$ with a non zero coefficient is less than $n + m - 2$ (because some of the right end terms of **a** or **b** were all consecutive 0's), the product polynomial $P_C(x) = P_A(x)\, P_B(x)$ is padded with zero coefficients times powers of x up to $2n - 2$, so that the total number of coefficients in $P_C(x)$ is exactly $n + m - 1$.

Visualizing Convolution $\mathbf{c} = \mathbf{a} * \mathbf{b}$



Applications of Convolution

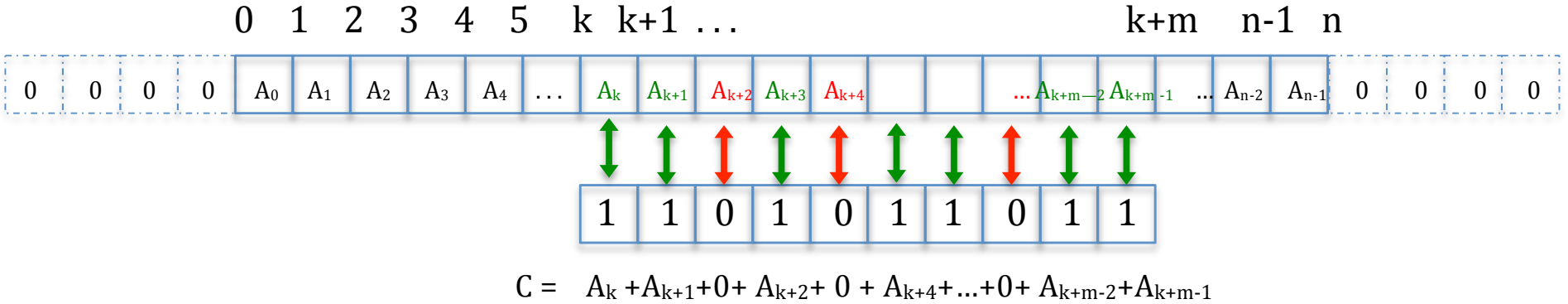
- The degree of similarity of two strings $\mathbf{a}$ and $\mathbf{b}$ of the same length n can be taken to be the number of places i where $\mathbf{a}[i] = \mathbf{b}[i]$.
- Assume you are given two binary strings $\mathbf{a}$ and $\mathbf{b}$; $\mathbf{a}$ has n^2 bits, the other, shorter string $\mathbf{b}$ has n bits.
- Your task is to find all substrings of consecutive bits in $\mathbf{a}$ which are of length n that are the most similar to string $\mathbf{b}$.
- Your algorithm should run in time $O(n^2 \log n)$. Note that the brute force algorithm runs in time $O(n^3)$.
- *Hint:* Replace $0's$ everywhere in both $\mathbf{a}$ and $\mathbf{b}$ with -1, to obtain sequences $\mathbf{a}'$ and $\mathbf{b}'$. Look at the convolution $\mathbf{c} = \mathbf{a}' * \widetilde{\mathbf{b}}'$ where $\widetilde{\mathbf{b}}'$ is the mirror image of $\mathbf{b}'$ (i.e., $\widetilde{\mathbf{b}}'(i) = \mathbf{b}'(n-1-i)$ for all $0 \leq i \leq n-1$).

  Find indices j such that $\mathbf{c}(j) = \max\{\mathbf{c}(m) : 0 \leq m \leq 2n-2\}$.

  How can you use such indices to find the substrings of $\mathbf{a}$ of the kind required?

Applications of Convolution

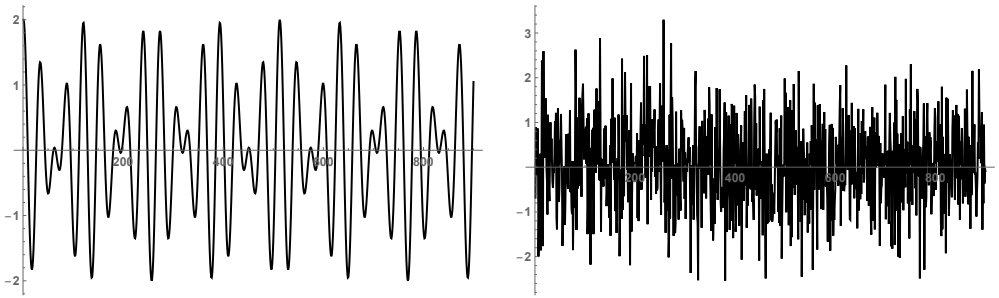
- Assume you are given a map of a straight sea shore of length n meters as a sequence on n numbers such that a_i is the number of fish between i^{th} meter of the shore and $(i+1)^{th}$ meter, $0 \leq i \leq n-1$. You also have a net of length m meters but unfortunately it has holes in it. Such a net is described as a sequence of m ones and zeros, where 0's denote where the holes are. If you throw such a net starting at meter k and ending at meter $k+m$, then you will catch only the fish in one meter stretches of the shore where the corresponding bit of the net is 1; see the figure.



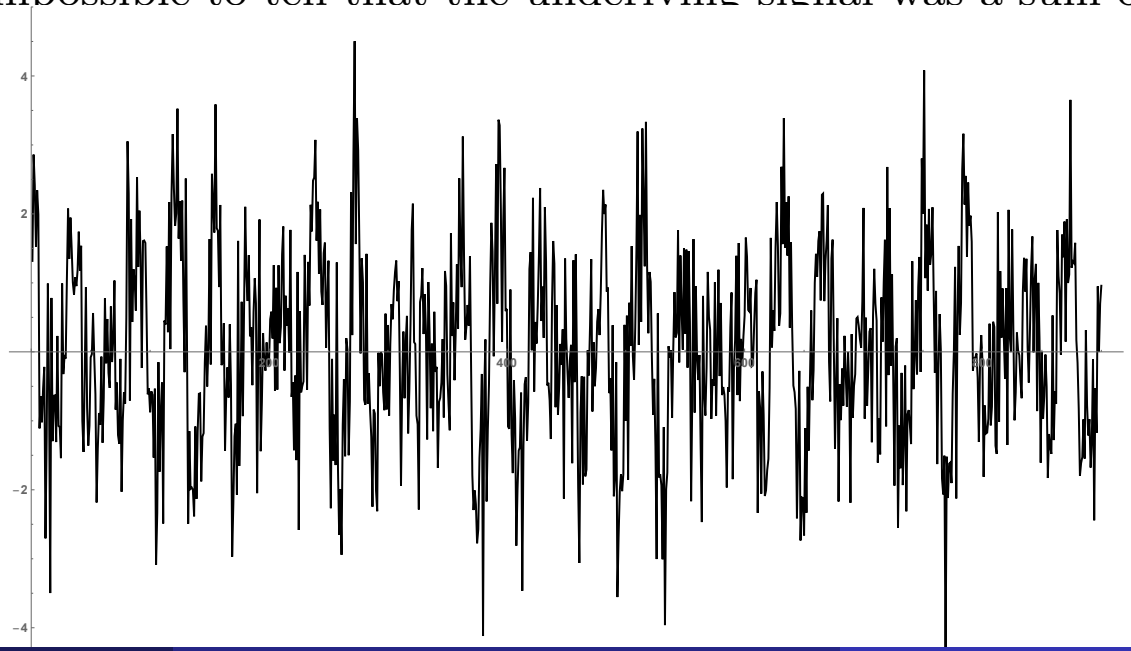
Find the spot where you should place the left end of your net in order to catch the largest possible number of fish using an algorithm which runs in time $O(n \log n)$.

Applications of Convolution—Moving Average Smoothing

- Let us define a simple signal by $s(t) = \cos\left(\frac{2\pi \cdot 26}{1024}\ t\right) + \cos\left(\frac{2\pi \cdot 34}{1024}\ t\right)$, and compute its values for $t = 1 \ldots 900$, thus obtaining sequence $\langle s(i) \ : \ 1 \leq i \leq 900 \rangle$.
- Using *Mathematica's* random number generator, let us produce a Gaussian white noise signal of the same length, of mean 0 and variance 1:

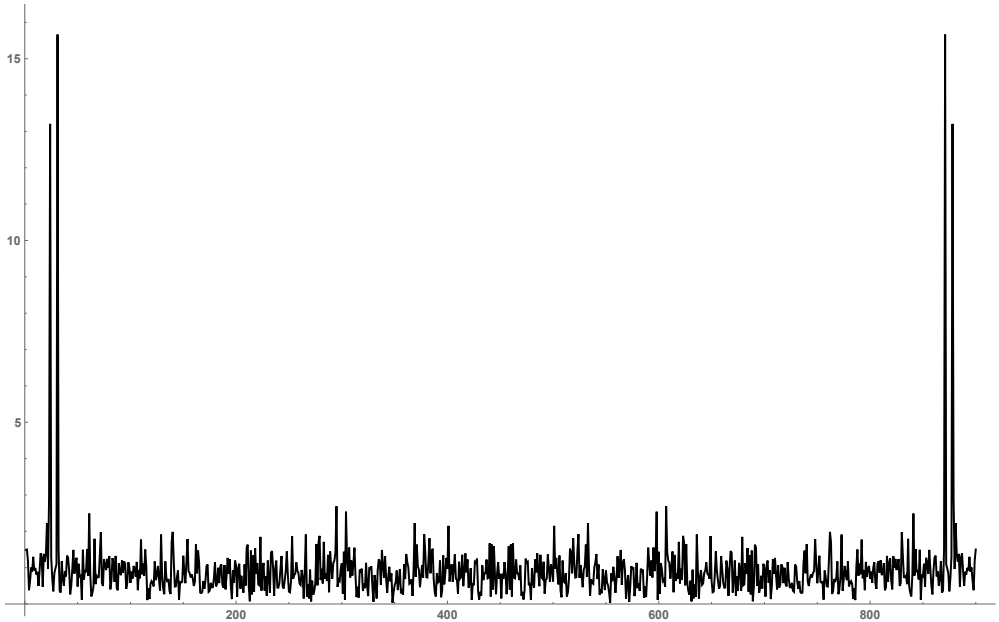


- We add noise to the signal to obtain a very noisy signal $\langle ns(i) \ : \ 1 \leq i \leq 900 \rangle$; notice that it is now impossible to tell that the underlying signal was a sum of two sine waves:



Applications of Convolution—Moving Average Smoothing

- However, if we plot the absolute value of the DFT of such a noisy signal it is pretty clear that it has two sine waves:



- Note that for two sine waves we get 4 peaks. This is because we need two complex exponentials per each cosine wave, in order to cancel out the imaginary parts:

$$\cos\left(\frac{2\pi \cdot 26}{1024}\, t\right) = \frac{1}{2}\left(\mathrm{e}^{i\,\frac{2\pi\cdot 26}{1024}\,t} + \mathrm{e}^{-\,i\,\frac{2\pi\cdot 26}{1024}\,t}\right) = \frac{1}{2}\left(\mathrm{e}^{i\,\frac{2\pi\cdot 26}{1024}\,t} + \mathrm{e}^{i\,\frac{2\pi\cdot(1024-26)}{1024}\,t}\right)$$
$$= \frac{1}{2}\left(\mathrm{e}^{i\,\frac{2\pi\cdot 26}{1024}\,t} + \mathrm{e}^{i\,\frac{2\pi\cdot 998}{1024}\,t}\right)$$

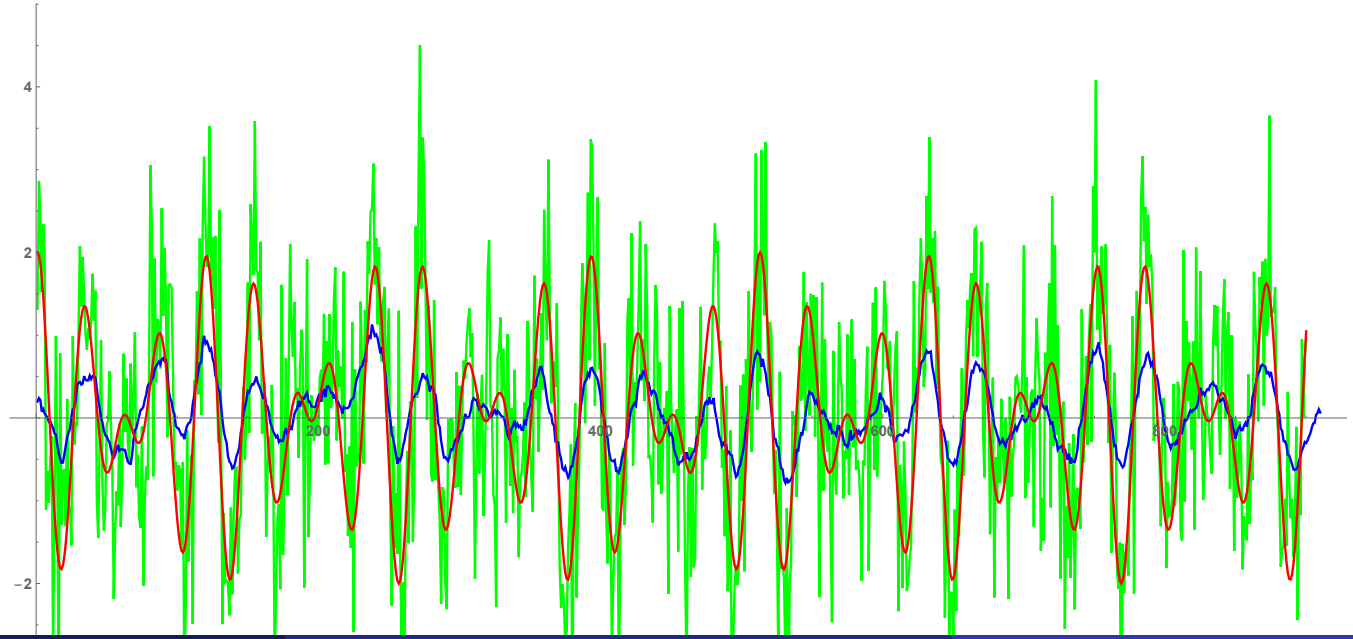
- So $\cos\left(2\pi \cdot 26/1024\, t\right)$ has two peaks: $\widehat{A}_{26}$ and $\widehat{A}_{998}$.

Applications of Convolution—Moving Average Smoothing

- We can smooth such a noisy signal using a *Moving Average*, by replacing every noisy value $ns(i)$ with the mean value of $2M+1$ values of the noisy signal,

$$ms(i) = \frac{1}{2M+1} \sum_{k=-M}^{M} ns(i+k)$$

(setting $ns(i) = 0$ for $i < 0$ and for $i > 900$). The degree of smoothing depends on M; larger values of M produce more smoothing but also distort more the underlying "clean signal" $s(i)$. Here we choose $M = 10$. The clean signal is in red, the noisy signal in green and the signal smoothed via the Moving Average is in blue; plots are produced using the *Mathematica* software, free to students of UNSW, go to the IT website to download it.



Applications of Convolution—Moving Average Smoothing

- How do we compute the Moving Average of a signal?
- Applying $ms(i) = \frac{1}{2M+1} \sum_{k=-M}^{M} ns(i+k)$ directly to a signal with N many data points would involve $(2M+1)N$ additions and N divisions.
- Much faster way, involving only $2M+1+2N$ additions and N divisions computes only the first value $ms(1)$ directly, using $2M+1$ additions and one division and then proceeds by recursion by letting
  $ms(i+1) = ms(i) + (ns(i+1+M) - ns(i-M))/(2M+1)$.
- This is correct because

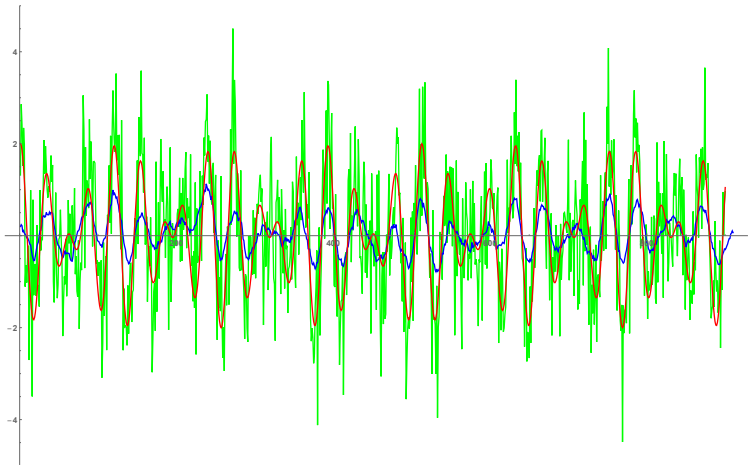
$$ms(i+1) = \frac{ns(i+1-M) + \ldots + ns(i+1) + \ldots + ns(i+1+M)}{2M+1}$$

$$= \frac{(ns(i-M) + ns(i+1-M) + \ldots + ns(i+1) + \ldots + ns(i+M)) + ns(i+1+M) - ns(i-M)}{2M+1}$$

$$= ms(i) + \frac{ns(i+1+M) - ns(i-M)}{2M+1}$$

Applications of Convolution—Moving Average Smoothing

- However, the signal obtained by smoothing via a Moving Average does not look like a particularly good replica of the original, uncorrupted signal:



- Can we do better?
- Note that the Moving Average gives equal weight of $\frac{1}{2M+1}$ to all of $2M+1$ many samples which are averaged:

$$ms(i) = \frac{1}{2M+1} \sum_{k=-M}^{M} ns(i+k) = \sum_{k=-M}^{M} \frac{1}{2M+1} \cdot ns(i+k)$$

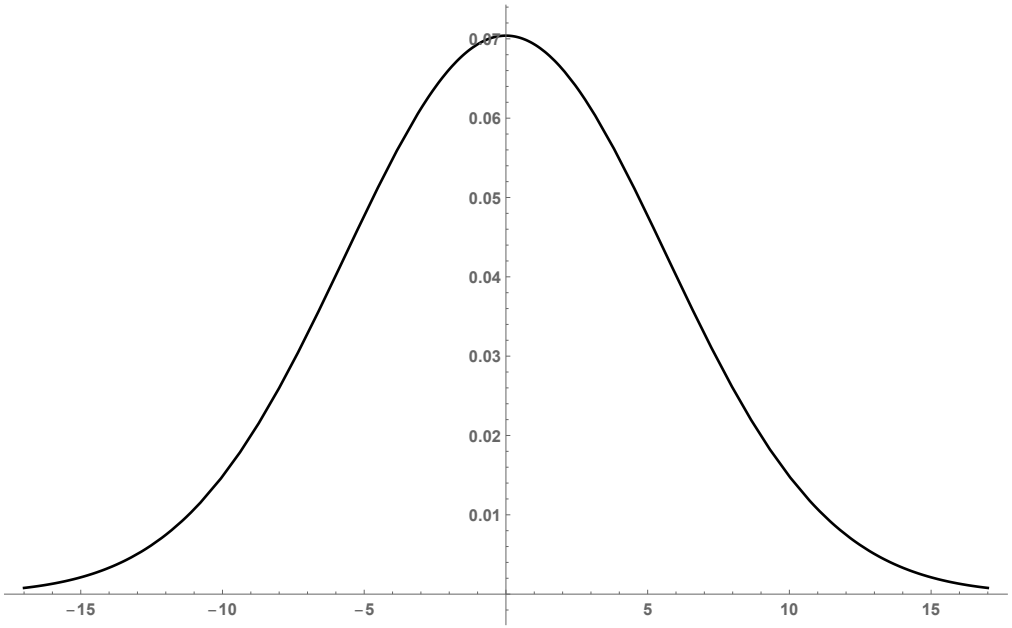
- Maybe there are better weights $w_{-M}, w_{-M+1}, \ldots, 0, \ldots, w_{M-1}, w_M$, which get smaller as you get away of the central point 0,
  $w_{-M} < w_{-M+1} < \ldots < w_{-1} < w_0 > w_l > \ldots > w_{M-1} > w_M$?

Applications of Convolution—Gaussian Smoothing

- Gaussian is just the normal distribution with probability density given by

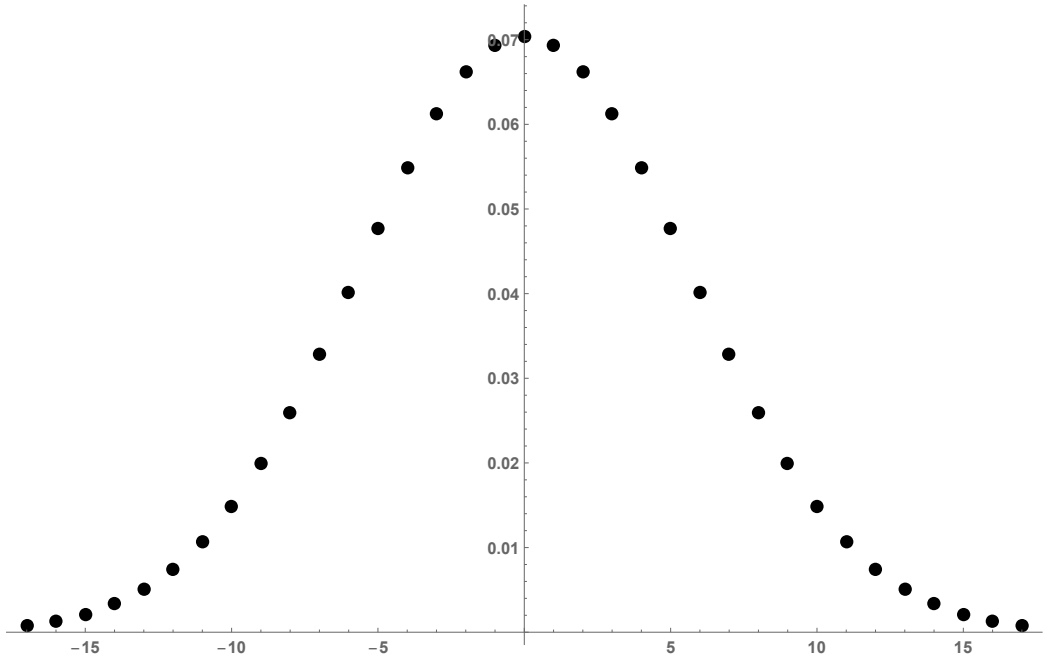
$$g(t) = \frac{1}{\sqrt{2\pi\, v}} \mathrm{e}^{-\frac{t^2}{2v}}$$

- It "dies off" almost completely whithin the interval $[-3\sqrt{v}, 3\sqrt{v}]$:



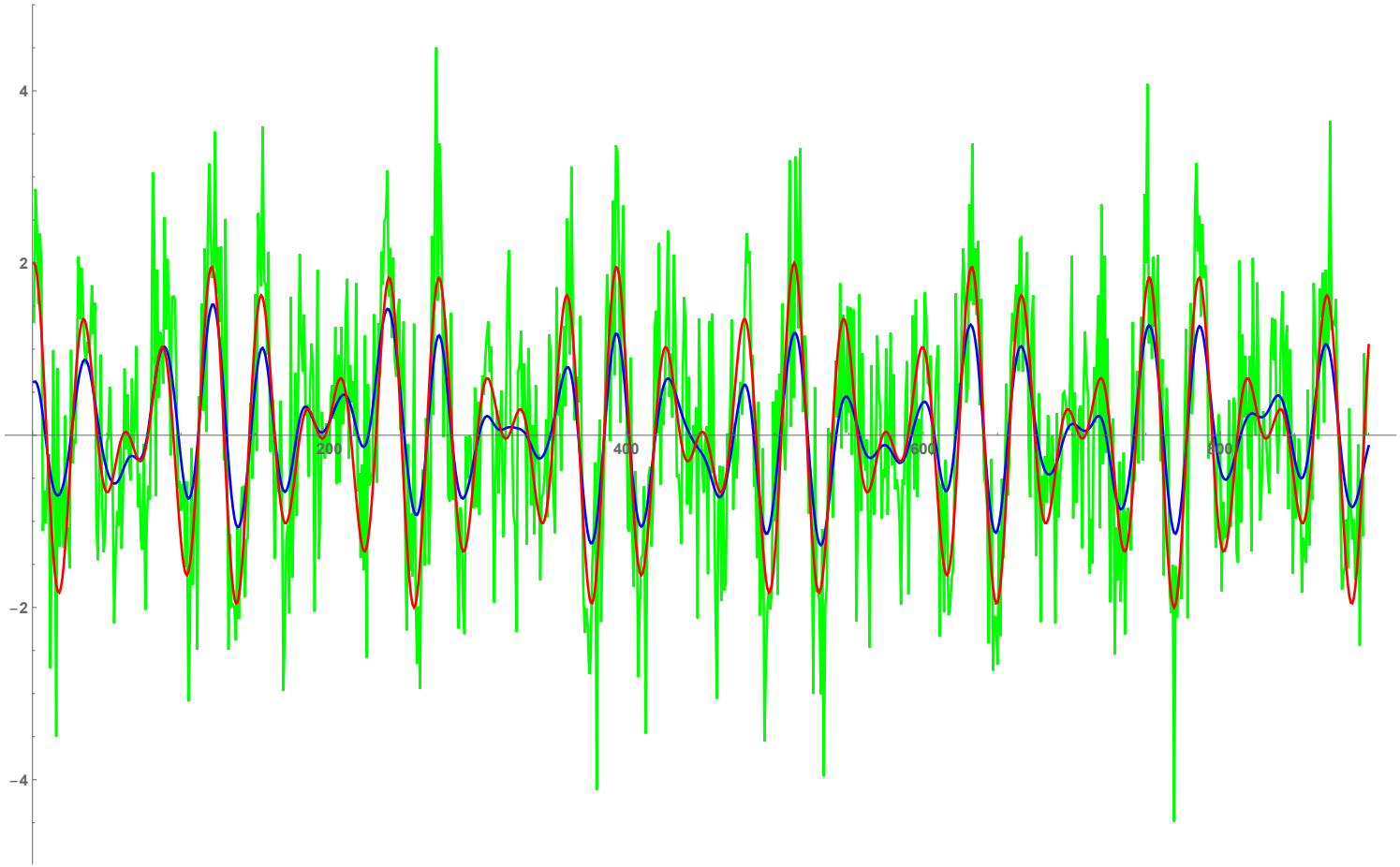
Applications of Convolution—Gaussian Smoothing

- We choose v such that $3\sqrt{v} = 17$ and evaluate such a Gausian at integers $-17 \ldots 17$ and take these values as weights for our smoothing; thus, $w_k = \frac{1}{\sqrt{2\pi\, v}} \mathrm{e}^{-\frac{k^2}{2v}}$. If k ranges between $-3\sqrt{v}$ and $3\sqrt{v}$, then these weights sum up almost exactly to 1, but you can normalise them to sum up exactly to 1 by replacing each w_k with $w_k / \sum_{m=-17}^{17} w_m$.



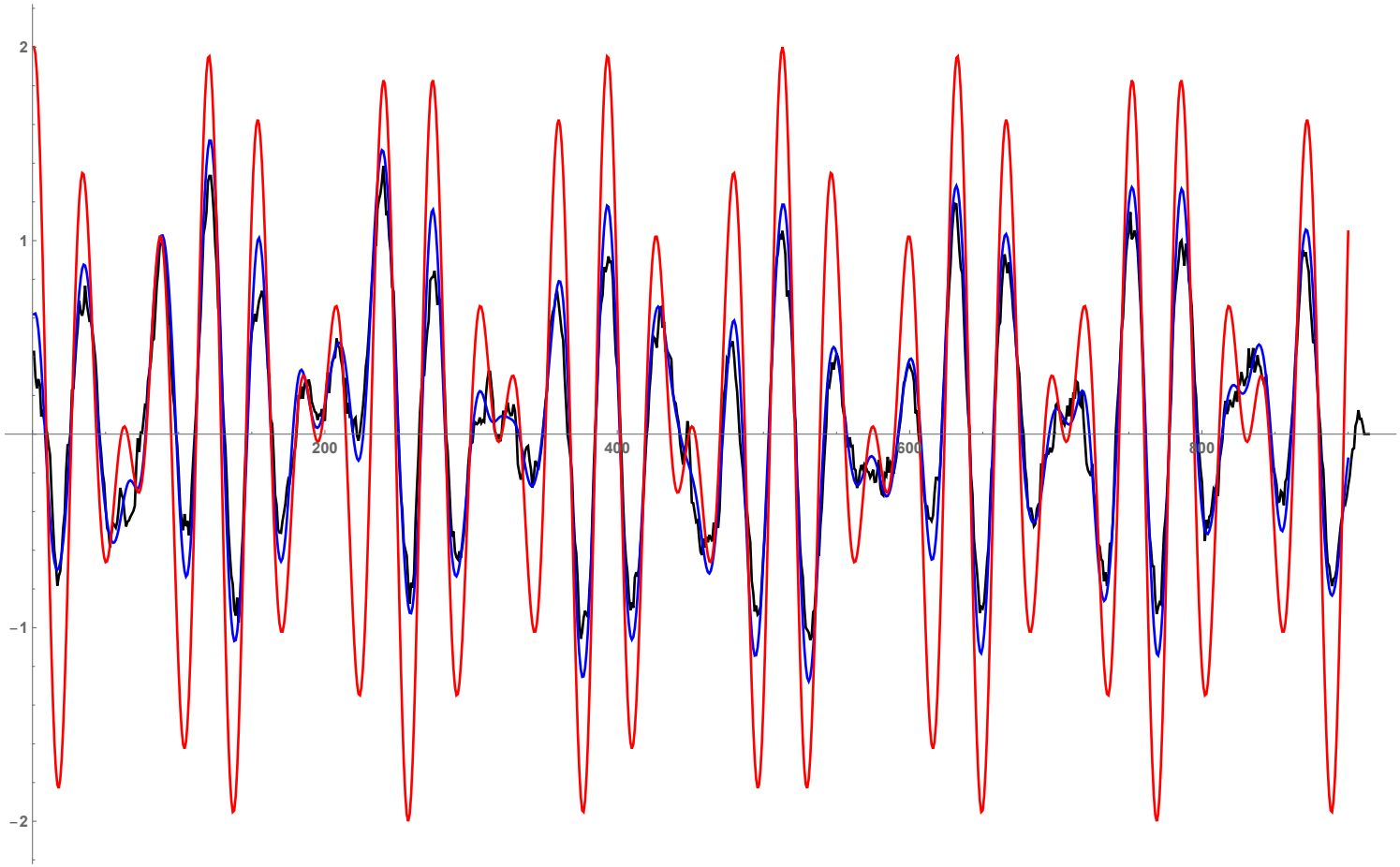
Applications of Convolution—Gaussian Smoothing

- We now produce a smoothing $gs(i)$ of the noisy signal $ns(i)$ by letting $gs(i) = \sum_{k=-17}^{17} w_k \, ns(i-k)$:



Applications of Convolution—Gaussian Smoothing

- We now compare the Moving Average smoothing (black) of our noisy signal with its Gaussian smoothing (blue); the clean signal is shown in red:



- We see that the Gaussian smoothing produces better results. But how do we compute it efficiently?

Applications of Convolution—Gaussian Smoothing

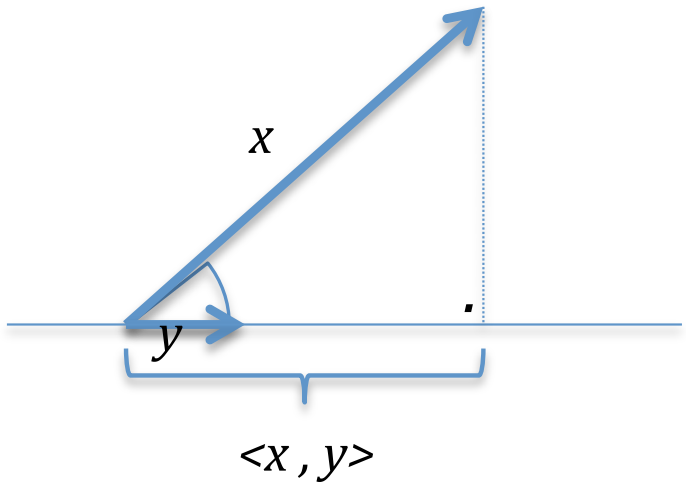
- Since the weights are not equal, the old trick of computing the smoothing at point $i+$ from the smoothed value at point i by dropping the first term of the sum and adding the new term clearly no longer works, and applying the formula $gs(i) = \sum_{k=-M}^{M} w_k \, ns(i-k)$ to N values of the noisy signal would result in about $(2M+1)N$ multiplications.
- However, notice that $gs(i) = \sum_{k=-M}^{M} w_k \, ns(i-k)$ is simply a convolution of the noisy signal with the vector of weights! And since the weights are symmetric, we do not even have to flip them.
- Thus, the Gaussian smoothing is simply the convolution $ns * W$ where $W = \langle w_{-M}, w_{-M+1}, \ldots, w_{-1}, 0, w_1, \ldots, w_{M-1}, w_M \rangle$ with $w_k = \frac{1}{\sqrt{2\pi\, v}} \mathrm{e}^{-\frac{k^2}{2v}}$ where the variance v is chosen so that $M = \lceil 3\sqrt{v} \rceil$, and ns is the sequence of N noisy samples of the signal we want to smooth.
- If we evaluate the convolution using the FFT, then we make only $O((M+N)\log_2(M+N))$ multiplications, which is faster than $(2M+1)N$ multiplications of the brute force.
- All of the above calculations can be found in a Mathematica file "convolution.nb" at the course website. Mathematica is a wonderful software free to the UNSW students which you can download from the UNSW IT website.

Optional Material: Interpretation of DFT

- The **scalar product** (also called *dot* product) of two vectors with real coordinates, $\vec{x} = (x_0, x_1, \ldots, x_{n-1})$ and $\vec{y} = (y_0, y_1, \ldots, y_{n-1})$, denoted by $\langle \vec{x}, \vec{y} \rangle$ is defined as

$$\langle \vec{x}, \vec{y} \rangle = \sum_{j=0}^{n-1} x_j y_j$$

- How to think about the scalar product of two vectors? Let $\vec{x}$ be an arbitrary vector and let $\vec{y}$ be a vector of unit length; then $\langle \vec{x}, \vec{y} \rangle$ is simply the length of the orthogonal projection of vector $\vec{x}$ onto the line to which vector $\vec{y}$ belongs:



- If $\vec{y}$ is not of unit length, then to get $\langle \vec{x}, \vec{y} \rangle$ the value of such a projection has to be multiplied by the length of vector $\vec{y}$.

Optional Material: Interpretation of DFT

- If the coordinates of our vectors are complex numbers, then the scalar product of such two vectors is defined as

$$\langle \vec{x}, \vec{y} \rangle = \sum_{j=0}^{n-1} x_j \overline{y_j}$$

- Recall that $\overline{z}$ denotes the complex conjugate of z, j.e., $\overline{a + i\ b} = a - i\ b$.
- The **norm** of a vector $\vec{x} = (x_0, x_1, \ldots, x_{n-1})$ is defined as

$$\|\vec{x}\| = \sqrt{\langle \vec{x}, \vec{x} \rangle} = \sqrt{\sum_{j=0}^{n-1} x_j \overline{x_j}} = \sqrt{\sum_{j=0}^{n-1} |x_j|^2}$$

- Note that the complex conjugation ensures that $x_j\, \overline{x_j} = |x_j|^2$ is always a non-negative number, because if $x_j = a + i\, b$ then

$$x_j \overline{x_j} = (a + i\, b)\overline{(a + i\, b)} = (a + i\, b)(a - i\, b) = a^2 + b^2 = |a + i\, b|^2 = |x_j|^2.$$

Optional Material: Interpretation of DFT

- Note that

$$\overline{\omega_n^k} = \overline{e^{i\frac{2\pi k}{n}}} = \overline{\cos\frac{2\pi k}{n} + i\sin\frac{2\pi k}{n}} = \cos\frac{2\pi k}{n} - i\sin\frac{2\pi k}{n}$$
$$= \cos\frac{-2\pi k}{n} + i\sin\frac{-2\pi k}{n} = e^{-i\frac{2\pi k}{n}} = \omega_n^{-k}$$

- Thus, what we had before,

$$\begin{pmatrix} 1 & \omega_n^k & \omega_n^{2\cdot k} & \dots & \omega_n^{k\cdot(n-1)} \end{pmatrix} \begin{pmatrix} 1 \\ \omega_n^{-m} \\ \omega_n^{-2m} \\ \vdots \\ \omega_n^{-(n-1)m} \end{pmatrix} = \sum_{j=0}^{n-1} \omega_n^{(k-m)j} = \begin{cases} n & \text{if } k = m \\ 0 & \text{if } k \neq m \end{cases} \tag{9}$$

simply means that for $k \neq m$ vectors $\begin{pmatrix} 1, & \omega_n^k, & \omega_n^{2\cdot k}, & \dots, & \omega_n^{k\cdot(n-1)} \end{pmatrix}$ and $\begin{pmatrix} 1, & \omega_n^m, & \omega_n^{2\cdot m}, & \dots, & \omega_n^{m\cdot(n-1)} \end{pmatrix}$ are mutually orthogonal and that their norm is equal to $\sqrt{n}$.

Optional Material: Interpretation of DFT

- If we define $\vec{e_k} = \frac{1}{\sqrt{n}}\left(1, \ \omega_n^{k\cdot 1}, \ \omega_n^{k\cdot 2}, \ \ldots \ , \omega_n^{k\cdot(n-1)}\right)$ then
  $\|\vec{e_k}\| = \sqrt{\langle \vec{e_k}, \vec{e_k}\rangle} = 1$ and $\langle \vec{e_k}, \vec{e_m}\rangle = 0$ for $k \neq m$

- Thus, the set of vectors $\{\vec{e_0}, \vec{e_1}, \ldots, \vec{e_n}\}$ is an orthonormal basis of the vector space of all complex valued sequences of length n.
- Let $\vec{A} = (a_0, a_1, a_2, \ldots, a_{n-1})$; then for $P_A^*(x) = \frac{1}{\sqrt{n}}\left(a_0 + a_1 x + \ldots + a_{n-1}x^{n-1}\right)$ we have

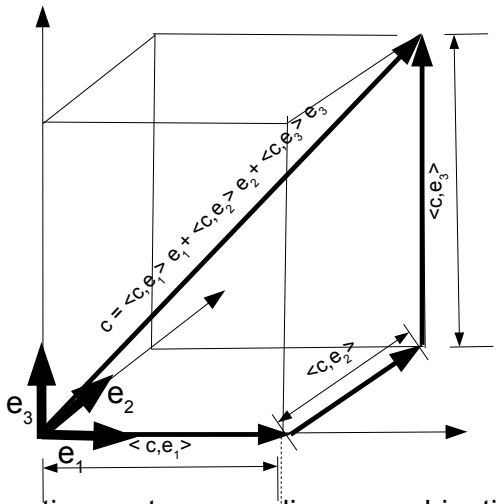
$$\begin{aligned}
\widehat{A}_k &= P_A^*(\omega_n^{-k}) \\
&= \frac{1}{\sqrt{n}} P_A(\omega_n^{-k}) \\
&= \frac{a_0}{\sqrt{n}}(\omega_n^{-k})^0 + \frac{a_1}{\sqrt{n}}(\omega_n^{-k})^1 + \frac{a_2}{\sqrt{n}}(\omega_n^{-k})^2 + \ldots + \frac{a_{n-1}}{\sqrt{n}}(\omega_n^{-k})^{n-1} \\
&= a_0\frac{(\overline{\omega_n^k})^0}{\sqrt{n}} + a_1\frac{(\overline{\omega_n^k})^1}{\sqrt{n}} + a_2\frac{(\overline{\omega_n^k})^2}{\sqrt{n}} + \ldots + a_{n-1}\frac{(\overline{\omega_n^k})^{n-1}}{\sqrt{n}} \\
&= \left\langle (a_0, a_1, a_2, \ldots, a_{n-1}), \left(\frac{(\omega_n^k)^0}{\sqrt{n}}, \frac{(\omega_n^k)^1}{\sqrt{n}}, \frac{(\omega_n^k)^2}{\sqrt{n}}, \ldots, \frac{(\omega_n^k)^{n-1}}{\sqrt{n}}\right)\right\rangle \\
&= \langle \vec{A}, \vec{e_k}\rangle
\end{aligned}$$

- Thus, the DFT of a vector $\vec{A}$ is simply the sequence of projections of $\vec{A}$ onto the basis vectors $\vec{e_k}$, $(k = 0, \ldots, n-1)$.

Optional Material: Interpretation of DFT

- In an n-dimensional vector space V with an orthonormal basis $\mathbf{B}$ every vector $\vec{A}$ can be represented as a linear combination of the basis vectors with coefficients equal to the projections of $\vec{A}$ onto the basis vectors, i.e., the scalar product $\langle \vec{A}, e_k \rangle$:

$$\vec{A} = \langle \vec{A}, \vec{e_0} \rangle \vec{e_0} + \langle \vec{A}, \vec{e_1} \rangle \vec{e_1} + \ldots + \langle \vec{A}, \vec{e}_{n-1} \rangle \vec{e}_{n-1}$$



Representing vector *c* as a linear combination of the basis vectors e_1, e_2, e_3 with projections as coefficients

Optional Material: Interpretation of DFT

- Thus, in our case

$$\begin{aligned}\vec{A} &= \langle \vec{A}, \vec{e_0} \rangle \vec{e_0} + \langle \vec{A}, \vec{e_1} \rangle \vec{e_1} + \ldots + \langle \vec{A}, \vec{e}_{n-1} \rangle \vec{e}_{n-1} \\ &= P_A^*(\omega_n^0)\vec{e_0} + P_A^*(\omega_n^{-1})\vec{e_1} + \ldots + P_A^*(\omega_n^{-(n-1)})\vec{e}_{n-1} \\ &= \widehat{A}_0\vec{e_0} + \widehat{A}_1\vec{e_1} + \ldots + \widehat{A}_{n-1}\vec{e}_{n-1}\end{aligned}$$

- Looking at the k^{th} coordinate of both the left and the right side we get

$$\begin{aligned}a_k &= \widehat{A}_0 \frac{(\omega_n^0)^k}{\sqrt{n}} + \widehat{A}_1 \frac{(\omega_n^1)^k}{\sqrt{n}} + \ldots + \widehat{A}_{n-1} \frac{(\omega_n^{n-1})^k}{\sqrt{n}} \\ &= \frac{1}{\sqrt{n}} \left(\widehat{A}_0 + \widehat{A}_1 (\omega_n^1)^k + \ldots + \widehat{A}_{n-1} (\omega_n^{n-1})^k \right) \\ &= \frac{1}{\sqrt{n}} \left(\widehat{A}_0 + \widehat{A}_1 (\omega_n^k)^1 + \ldots + \widehat{A}_{n-1} (\omega_n^k)^{n-1} \right)\end{aligned}$$

- Thus, a_k is obtained by evaluating the polynomial

$$Q(x) = \frac{1}{\sqrt{n}} \left(\widehat{A}_0 + \widehat{A}_1 x + \ldots + \widehat{A}_{n-1} x^{n-1} \right)$$

  at $x = \omega_n^k$.
- But this is precisely the Inverse Discrete Fourier Transform of the sequence

$$\left(\widehat{A}_0, \widehat{A}_1, \ldots, \widehat{A}_{n-1} \right)$$

Optional Material: Interpretation of DFT

- Let us denote the usual orthonormal basis of $\mathbb{C}^n$ by $\mathcal{B}$:

$$\vec{b}_0 = (1,0,0,0,\ldots,0),\ \vec{b}_1 = (0,1,0,0,\ldots,0),\ \ldots,\ \vec{b}_{n-1} = (0,0,0,0,\ldots,1)$$

and by $\mathcal{F}$ the basis $\mathcal{F} = \{\vec{e}_0, \vec{e}_1, \ldots, \vec{e}_{n-1}\}$ where $\vec{e_k} = \left(\frac{1}{\sqrt{n}},\ \frac{\omega_n^{k\cdot 1}}{\sqrt{n}},\ \frac{\omega_n^{k\cdot 2}}{\sqrt{n}},\ \ldots\ ,\frac{\omega_n^{k\cdot(n-1)}}{\sqrt{n}}\right)$.

- then $\vec{A} = (a_0, a_1, a_2, \ldots, a_{n-1})$

  and $\vec{A} = a_0\vec{b}_0 + a_1\vec{b}_1 + a_2\vec{b}_2 + \ldots + a_{n-1}\vec{b}_{n-1}$

  but also $\widehat{A} = (P_A^*(\omega_n^0), P_A^*(\omega_n^{-1}), \ldots, P_A^*(\omega_n^{-(n-1)})) = (\widehat{A}_0, \widehat{A}_1, \ldots, \widehat{A}_{n-1})$

  and also $\vec{A} = \widehat{A}_0\vec{e_0} + \widehat{A}_1\vec{e_1} + \ldots + \widehat{A}_{n-1}\vec{e}_{n-1}$
- Thus, DFT is just a change of basis operation: it transforms the sequence of coordinates

$$(a_0, a_1, a_2, \ldots, a_{n-1})_{\mathcal{B}}$$

  of a vector $\vec{A}$ in the basis $\mathcal{B}$ into the sequence

$$\left(P_A^*(\omega_n^0), P_A(\omega_n^{-1}), \ldots, P_A(\omega_n^{-(n-1)})\right)_{\mathcal{F}} = (\widehat{A}_0, \widehat{A}_1, \ldots, \widehat{A}_{n-1})_{\mathcal{F}}$$

  of the coordinates of the same vector in the basis $\mathcal{F}$.

Optional Material: Interpretation of DFT

- Basis $\mathcal{F}$ is very important!
- Let us look again at the equation

$$\begin{aligned} a_k &= \widehat{A}_0 \frac{(\omega_n^k)^0}{\sqrt{n}} + \widehat{A}_1 \frac{(\omega_n^k)^1}{\sqrt{n}} + \ldots + \widehat{A}_{n-1} \frac{(\omega_n^k)^{n-1}}{\sqrt{n}} \\ &= \frac{1}{\sqrt{n}} \left(\widehat{A}_0 + \widehat{A}_1 (\omega_n^k)^1 + \ldots + \widehat{A}_{n-1} (\omega_n^k)^{n-1} \right) \end{aligned}$$

- Assume that $a_k = A(k)$ is actually a sample of a signal $A(t)$ at a time instant k;
- Let us also consider $(\omega_n^k)^m = e^{i\frac{2\pi mk}{n}}$ to be a sample of the function $f_m(t) = e^{i\frac{2\pi m}{n}t}$ also at an instant k;
- Note that $f_m(t) = e^{i\frac{2\pi m}{n}t} = \cos\left(\frac{2\pi m}{n}t\right) + i\sin\left(\frac{2\pi m}{n}t\right)$;
- This means that the samples $A(k)$ of the signal $A(t)$ at instants $t = 0, 1, \ldots, n-1$ are represented as a linear combination of samples at these instants of sinusoids (also called pure harmonic oscillations) $\cos\left(\frac{2\pi m}{n}t\right)$ and $\sin\left(\frac{2\pi m}{n}t\right)$ of increasing frequencies $\frac{2\pi\cdot 1}{n}, \frac{2\pi\cdot 2}{n}, \ldots \frac{2\pi\cdot(n-1)}{n}$:

$$A(k) = \widehat{A}_0 f_0(k) + \widehat{A}_1 f_1(k) + \widehat{A}_2 f_2(k) + \ldots + \widehat{A}_{n-1} f_{n-1}(k)$$

Optional Material: Interpretation of DFT

- Thus, in a sense the absolute value $|\widehat{A}_m| = |P_A^*(\omega_n^{-m})|$ of the m^{th} coordinate $\widehat{A}_m$ in the basis $\mathcal{F}$ tells us "how much" of the sinusoids $\cos\left(\frac{2\pi m}{n}t\right)$ and $\sin\left(\frac{2\pi m}{n}t\right)$ are present in the signal $A(t)$.
- This is very important in very many fields, ranging from astronomy (detecting planets orbiting distant stars), electrical engineering (signal and image processing), to economics (looking for cycles in economic indicators).
- We call this *spectral analysis*.

That's All, Folks!!