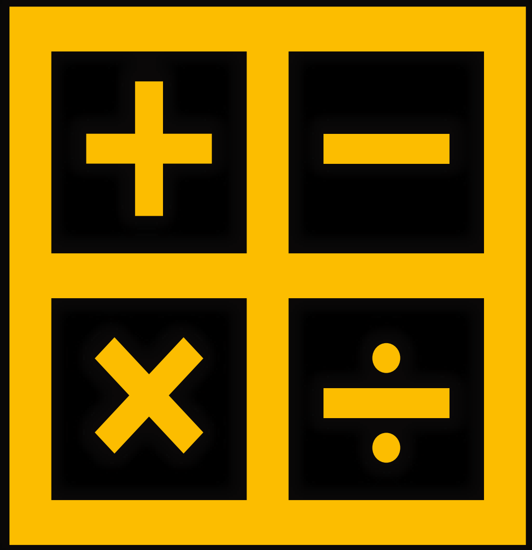
The Fourier Transform
and
Fast Arithmetic



What is the Fourier Transform?

- At a basic level the "*Fourier Transform*" is an operation that maps between n-vectors of Complex values and n-vectors of Complex values. That is,

$$\mathbf{F} : C^n \leftrightarrow C^n$$

- In *electronics* it is often a useful method of translating between the so-called "*spatial*" and "*frequency*" domains of a *signal*, eg the *amplitude of a wave* and its *frequency* so providing a powerful tool in *signal processing* for reducing *noise* and *distortion*.
- In *CS* it offers one technique for *Image Compression* and, in the "*Discrete Form*", is the basis for the fastest known *Integer Multiplication Algorithm*: the *Schönhage-Strassen Method* (1973).

Fourier Transform – Formal Definition

- Given

$$\underline{x} = \langle x_0, x_1, \dots, x_{n-1} \rangle \in C^n$$

- The Fourier Transform, $\mathbf{F}(\underline{x})$, is $\underline{y} = \langle y_0, y_1, \dots, y_{n-1} \rangle \in C^n$:

$$y_t = \sum_{k=0}^{n-1} x_k e^{\frac{-2\pi i t k}{n}}$$

- writing: $\omega_k \stackrel{\text{def}}{=} e^{\frac{-2\pi i k}{n}}$ this is the same as:

$$y_t = \sum_{k=0}^{n-1} x_k \omega_k^t$$

Fourier Transform – Important Aspects

- The object ω_k is a *Primitive n'th root of unity*: $\omega_k^n = 1$.
- We can describe the computation as a *Matrix-vector product*:

$$\begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} (\omega_0)^0 & (\omega_1)^0 & \cdots & (\omega_{n-1})^0 \\ (\omega_0)^1 & (\omega_1)^1 & \ddots & (\omega_{n-1})^1 \\ \vdots & \vdots & \ddots & \vdots \\ (\omega_0)^{n-1} & (\omega_1)^{n-1} & \cdots & (\omega_{n-1})^{n-1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{pmatrix}$$

- More succinctly: $\underline{y} = \mathbf{M_F}\underline{x}$

Fourier Transform – Useful Properties

- It is *easily invertible*: given $\underline{y} \in C^n$ we can find $\underline{x} \in C^n$ with $\underline{y} = \mathbf{F}(\underline{x})$ by computing $\underline{x} = \frac{\mathbf{F}(\underline{y})}{n}$.
- It defines a *Linear Transformation*:

$$\mathbf{F}\left(\alpha\underline{x} + \underline{y}\right) = \alpha\mathbf{F}(\underline{x}) + \mathbf{F}\left(\underline{y}\right)$$

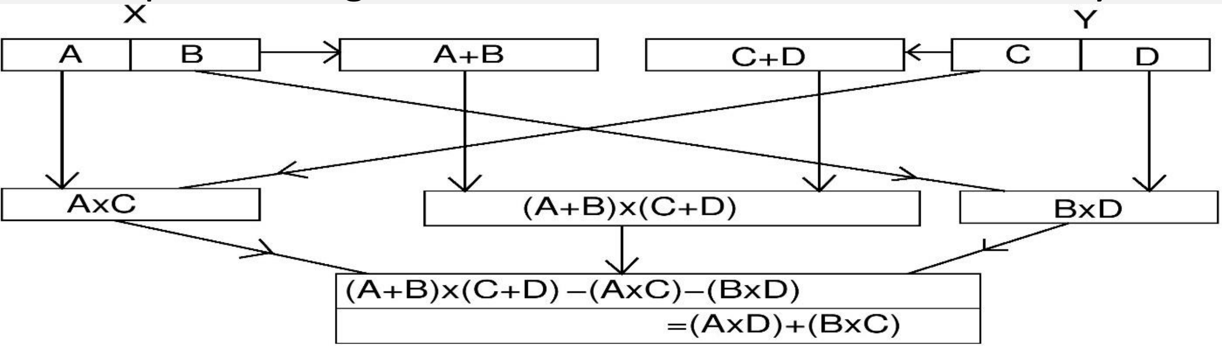
- It has a *Convolution Property* (textbook pp 230 – 231).
- This final property underpins *Fast Multiplication*.
- Here operations take place in $\mathbf{Z}_m$ (*arithmetic modulo m* for appropriate *m*), an analogue of "*Primitive Root of Unity*" exists and leads to the same structures discussed above (page 240).

Fast Multiplication – a sketch

- Traditional "*school methods*" to multiply two n digit numbers, X and Y say, take *each* of the *digits* in X and *multiply* by *each* of the *digits* in Y.
- This involves about n^2 *basic operations* (counting single digit by single digit as a "basic operation").
- We *might* do better by taking the digits in each number and writing $X = (A) \cdot 10^{n/2} + (B)$; $Y = (C) \cdot 10^{n/2} + (D)$ and using *recursion*:
$$X \cdot Y = \left(A \cdot 10^{n/2} + B\right) \cdot \left(C \cdot 10^{n/2} + D\right)$$
- If we implement the recursion "*naively*" . . .
- ***we don't do better.***

Not being “naive” – Karatsuba’s Algorithm

- In 1962 the Soviet scientist Anatoly Karatsuba found a method of implementing the recursive method more efficiently:



$$X \times Y = (A \times 10^{m} + B) \times (C \times 10^{m} + D) = (A \times C) \times 10^{(2m)} + (A \times D + B \times C) \times 10^{m} + (B \times D)$$

What Karatsuba's Method Does

- The "naïve" recursive approach makes *four calls* to compute:
$$A \cdot C \,;\, A \cdot D \,;\, B \cdot C \,;\, B \cdot D$$
- Although only numbers of "*half the length*" are involved this leads to the overall number of steps being $\sim n^2$.
- Karatsuba's Method uses only *three recursive calls*, computing:
$$A \cdot C \,;\, B \cdot D \,;\, (A + B) \cdot (C + D)$$
- The term $A \cdot D + B \cdot C$ needed is then just
$$(A + B) \cdot (C + D) - A \cdot C \; - B \cdot D$$
- The *extra addition saves one call* and the algorithm needs *only* $\sim n^{1.59}$ "*basic operations*" $\ll n^2$

Why stop at splitting in 2?

- We could try the same device of finding clever ways of saving on the "*obvious*" number of calls when *splitting* into *3, 4, 5,* …
- This very quickly becomes *onerous* and the *extra addition steps* needed rapidly *reduce its effectiveness*.
- Or we could *split numbers* into "*sections whose length depends on the length of the number itself*".
- The *Schönhage-Strassen method* divides n-digit numbers into roughly $\sqrt{n}$ parts each having $\sqrt{n}$ digits.
- The *Convolution Property* allows the multiplication to be *done quickly* via the *Fourier Transform* in $\sim n \log n \log \log n$ steps.

The Fourier Transform – Summary

- The Fourier Transform provides a powerful range of methods of importance in *electronics* and *signal processing*.
- Its basis is in the Complex Analysis notion of "*Primitive Root*".
- In *Computer Science* one of the most significant applications was in its use to develop *fast algorithms* for *Multiplication*.
- Other uses include *Image Compression* (textbook pp 231–3).
- Fast algorithms to *compute* the Fourier Transform have also been developed (textbook pp 241–5).
- In the final review of Complex Number applications in CS we look at their use in *AI and Computer Art*.