FFT: an example

Outline

- DFT: evaluate a polynomial at n special points;
- FFT: an efficient implementation of DFT;
- Applications of FFT: multiplying two polynomials (and multiplying two n-bits integers); time-frequency transform; solving partial differential equations;

DFT: Discrete Fourier Transform

- DFT evaluates a polynomial $A(x) = a_0 + a_1 x + \ldots + a_{n-1}x^{n-1}$ at n distinct points $1, \omega, \omega^2, \ldots, \omega^{n-1}$, where $\omega = e^{\frac{2\pi}{n}i}$ is the n-th complex root of unity.
- Thus, it transforms the complex vector $a_0, a_1, \ldots, a_{n-1}$ into another complex vector $y_0, y_1, \ldots, y_{n-1}$, where $y_i = A(w^i)$, i.e.,

$$
\begin{array}{lllll}
y_0 = a_0 & + a_1 & + a_2 & \ldots & + a_{n-1} \\
y_1 = a_0 & + a_1\omega^1 & + a_2\omega^2 & \ldots & + a_{n-1}\omega^{n-1} \\
\ldots & \ldots & \ldots & \ldots & \ldots \\
y_{n-1} = a_0 & + a_1\omega^{n-1} & + a_2\omega^{2(n-1)} & \ldots & + a_{n-1}\omega^{(n-1)^2}
\end{array}
$$

- Matrix form:

$$
\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega^1 & \omega^2 & \ldots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \ldots & \omega^{2(n-1)} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \ldots & \omega^{(n-1)^2} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix}
$$

FFT: a fast way to implement DFT [Cooley-Tukey 1965]

- Direct matrix-vector multiplication requires $O(n^2)$ operations when using the Horner's method, i.e.,

$$A(x) = a_0 + x(a_1 + x(a_2 + \ldots + xa_{n-1})).$$

- FFT: reduce $O(n^2)$ to $O(n \log_2 n)$ using divide-and-conqueror technique.
- Note: The idea of FFT was proposed by Cooley and Tukey in 1965 when analyzing earth-quake data, but the idea can be dated back to F. Gauss.

Application: time-frequency transform

DFT: time-domain vs. frequency-domain

- DFT, denoted as $\mathbf{X} = \mathcal{F}\{\mathbf{x}\}$, transforms a sequence of N complex numbers $x_0, x_1, ..., x_{N-1}$ (time-domain) into a N-periodic sequence of complex numbers $X_0, X_1, ..., X_{N-1}$ (frequency-domain):

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi}{N} ikn}, \quad k = 0, 1, ..., N-1$$

- Here, X_k encodes both amplitude and phase of a sinusoidal component $e^{-\frac{2\pi}{N} kni}$ of the function x_n (the sinusoid's frequency is k **cycles per** N **samples**).
- **Inverse transform** of DFT:

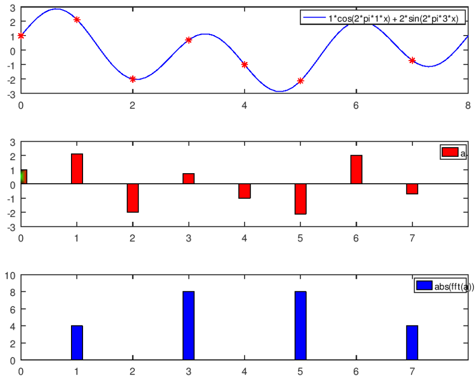
$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{\frac{2\pi}{N} ikn}$$

An interpretation of DFT is that its inverse transform is the **discrete analogy** of the formula for **a Fourier series**:

$$f(x) = \sum_{n=-\infty}^{+\infty} F_n e^{inx}, \; F_n = \frac{1}{2\pi} \int_{-\infty}^{\infty} f(x) e^{-inx} \mathrm{d}x$$

FFT: an example

```
N = 8;
t = 0:1/N:1-1/N;
a = 1*cos(2*pi*1*t) + 2*sin(2*pi*3*t);
Freq = 0:N-1;
bar( Freq, abs(fft(a)), "b", 0.2 );
```



y_1 encodes amplitude and phase of a sinusoidal component

- $y_1 = a_0 + \ a_1\omega^1 + \ a_2\omega^2 + \ldots + \ a_7\omega^7$ computes the direct product of two vectors: the input data $a_0, a_1, ..., a_7$, and a sinusoid signal $1, \omega^1, \omega^2, ..., \omega^7$. Here $\omega = e^{\frac{2\pi}{8}i}$ and thus the sinusoid has frequency of 1 cycle per 8 samples.
- The direct product of vectors is essentially a discrete analogy of the integral of two sinusoids

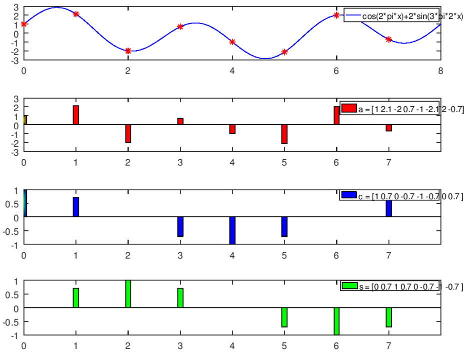
$$\int_0^{2\pi} \cos mx \cdot \sin nx \mathrm{d}x$$

- The orthogonality of sinusoids states that

$$\int_0^{2\pi} \cos mx \cdot \cos nx \mathrm{d}x = \begin{cases} \frac{\pi}{n} & m = n \\ 0 & m \neq n \end{cases}$$

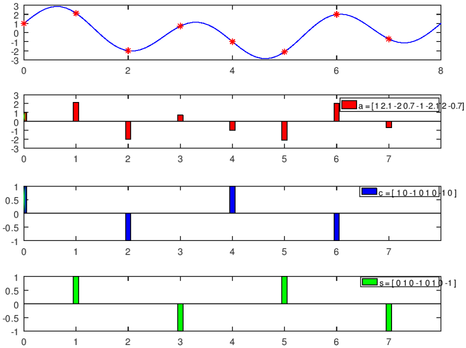
- Thus $y_1 = 0$ if the input data $a_0, a_1, ..., a_7$ does not consist of a sinusoidal component of frequency 1 cycle per 8 samples.

- As $1, \omega^1, \omega^2, ..., \omega^7$ represents a sinusoidal signal of frequency 1 cycle per 8 samples, the direct product $y_1 = a_0 + \; a_1\omega^1 + \; a_2\omega^2 + \ldots + \; a_7\omega^7$ could detect the existence of such sinusoidal component in the input data $a_0, a_1, ..., a_7$.
- ```
  a = [1 2.1 -2 0.7 -1 -2.1 2 -0.7]
  c = [1 0.7 0 -0.7 -1 -0.7 0 0.7 ]
  s = [0 0.7 1 0.7 0 -0.7 -1 -0.7 ]
  ```
  ```
  sqrt( (a*c')∧2 + (a*s')∧2 ) = 4
  ```


  ```

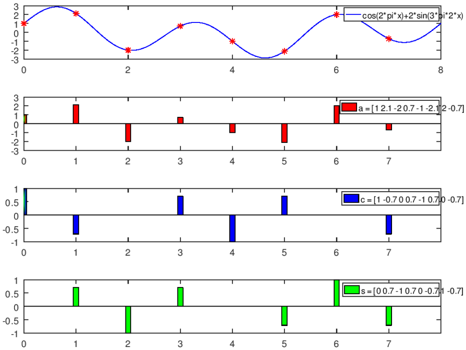
Calculation of y_2

- $1, \omega^2, \omega^4, ..., \omega^{14}$ represents a sinusoidal signal of frequency 2 cycles per 8 samples, and the existence of such sinusoidal component in the input data $a_0, a_1, ..., a_7$ is encoded by the direct product $y_2 = a_0 + \ a_1\omega^2 + \ a_2\omega^4 + \ldots + \ a_7\omega^{14}$.
- ```
  a = [1 2.1 -2 0.7 -1 -2.1 2 -0.7]
  c = [ 1 0 -1 0 1 0 -1 0 ]
  s = [ 0 1 0 -1 0 1 0 -1 ]
  sqrt( (a*c')∧2 + (a*s')∧2 ) = 0
  ```


  ```

Calculation of y_3

- $1, \omega^3, \omega^6, ..., \omega^{21}$ represents a sinusoidal signal of frequency 3 cycles per 8 samples, and the existence of such sinusoidal component in the input data $a_0, a_1, ..., a_7$ is encoded by the direct product $y_3 = a_0 + \ a_1\omega^3 + \ a_2\omega^6 + \ldots + \ a_7\omega^{21}$.
- ```
  a = [1 2.1 -2 0.7 -1 -2.1 2 -0.7]
  c = [1 -0.7 0 0.7 -1 0.7 0 -0.7]
  s = [0 0.7 -1 0.7 0 -0.7 1 -0.7]
  sqrt( (a*c')∧2 + (a*s')∧2 ) = 8
  ```


  ```