3F1, Signals and Systems

PART V.2

Fast Fourier Transform and Multidimensional Transform

Fulvio Forni (f.forni@eng.cam.ac.uk)

November 12, 2018

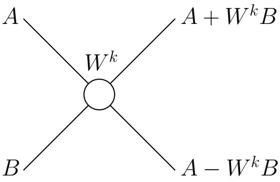
Fast Fourier transform

- **FFT = fast DFT (for large data set)**.
- The discovery of the FFT: *J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex Fourier series. Mathematics of Computation, 19:297301, 1965.*
- A paper by Gauss published posthumously in 1866 (and dated 1805) contains a "splitting" technique that forms the basis of modern FFT algorithms.

- **DFT complexity $\simeq N^2$. FFT complexity $\simeq N \log_2(N)$.**
- For $N = 1024$, FFT is 205 times faster.

- Many different FFT algorithms.
- We consider the basic "radix-2" algorithm (N power of 2).

- FFT relies on redundancy in the calculation of the basic DFT
  $\rightarrow$ reuse past computations.
- FFT is a recursive algorithm. Repeatedly rearranges the problem into simpler subproblems of half the size
  $\rightarrow$ logarithmic complexity, restriction to $N = 2^M$ horizon.
- Basic recursive "butterfly" structure



DFT:

$$\bar{x}_p := \sum_{k=0}^{N-1} x_k e^{-j\frac{2\pi}{N}pk} = b(p,N)'x$$

where $x = [x_0, \ldots, x_{N-1}]'$ and $b(p,N) = [e^{-j\frac{2\pi}{N}p0} \ldots e^{-j\frac{2\pi}{N}(N-1)k}]'$

Recursion: split the summation into two parts

$$\begin{aligned}
\bar{x}_p &= \sum_{k=0}^{\frac{N}{2}-1} x_{2k} e^{-j\frac{2\pi}{N}(2k)p} + \sum_{k=0}^{\frac{N}{2}-1} x_{2k+1} e^{-j\frac{2\pi}{N}(2k+1)p} \\
&= \underbrace{\sum_{k=0}^{\frac{N}{2}-1} x_{2k} e^{-j\frac{2\pi}{(N/2)}kp}}_{=:A_p} + \underbrace{e^{-j\frac{2\pi}{N}p}}_{=:W^p} \underbrace{\sum_{k=0}^{\frac{N}{2}-1} x_{2k+1} e^{-j\frac{2\pi}{(N/2)}kp}}_{=:B_p} \\
&= A_p + W^p B_p \qquad A_p \text{ and } B_p \text{ are DFT too.}
\end{aligned}$$

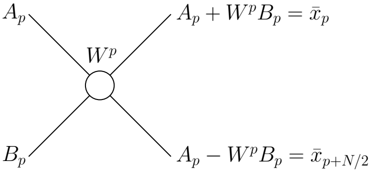
$$b(p,N)'x = A_p + W^p B_p = b(p,N/2)'x_A + W^p b(p,N/2)'x_B$$

where $x_A = [x_0, x_2, \ldots, x_{N-2}]'$ and $x_B = [x_1, x_3, \ldots, x_{N-1}]'$.

$$
\begin{aligned}
\bar{x}_p &= \underbrace{\sum_{k=0}^{\frac{N}{2}-1} x_{2k} e^{-j\frac{2\pi}{(N/2)}kp}}_{A_p} + \underbrace{e^{-j\frac{2\pi}{N}p}}_{W^p} \underbrace{\sum_{k=0}^{\frac{N}{2}-1} x_{2k+1} e^{-j\frac{2\pi}{(N/2)}kp}}_{B_p} \\
&= A_p + W^p B_p
\end{aligned}
$$

Redundancy: the DFT is periodic in the frequency domain

$$
\begin{aligned}
\bar{x}_{p+N/2} &= \underbrace{\sum_{k=0}^{\frac{N}{2}-1} x_{2k} e^{-j\frac{2\pi}{(N/2)}k(p+N/2)}}_{x_{2k} e^{-j\frac{2\pi}{(N/2)}kp}} + \underbrace{e^{-j\frac{2\pi}{N}(p+N/2)}}_{-e^{-j\frac{2\pi}{N}p}} \underbrace{\sum_{k=0}^{\frac{N}{2}-1} x_{2k+1} e^{-j\frac{2\pi}{(N/2)}k(p+N/2)}}_{x_{2k+1} e^{-j\frac{2\pi}{(N/2)}kp}} \\
&= A_p - W^p B_p
\end{aligned}
$$



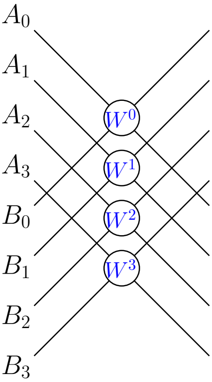
recursion + redundancy:
from N DFT to $2 \times N/2$ DFT and reuse past computation

- $\bar{x}_p = b(p, N)'x = A_p + W^p B_p$
- $\bar{x}_{p+N/2} = b(p + N/2, N)'x = A_p - W^p B_p$
- $A_p = b(p, N/2)'x_A$ where $x_A = [x_0, x_2, \ldots, x_{N-2}]'$
- $B_p = b(p, N/2)'x_B$ where $x_B = [x_1, x_3, \ldots, x_{N-1}]'$
- $W = e^{-j\frac{2\pi}{N}}$

- **Complexity:** DFT requires $2N + 2N$ operations (sums and products) to compute $\bar{x}_p$ and $\bar{x}_{p+N/2}$. The new method requires N operations for A_p, N operations for B_p and $2 + 2$ additional sums and product products (for $\bar{x}_p$ and $\bar{x}_{p+N/2}$ respectively).

$$\frac{4N}{2N+2+2} \simeq 2 \text{ times faster}$$

Example $N = 8$



$A_0 + W^0 B_0 = \bar{x}_0$

$A_1 + W^1 B_1 = \bar{x}_1$

$A_2 + W^2 B_2 = \bar{x}_2$

$A_3 + W^3 B_3 = \bar{x}_3$

$\bar{x}_p$

$A_0 - W^0 B_0 = \bar{x}_4$

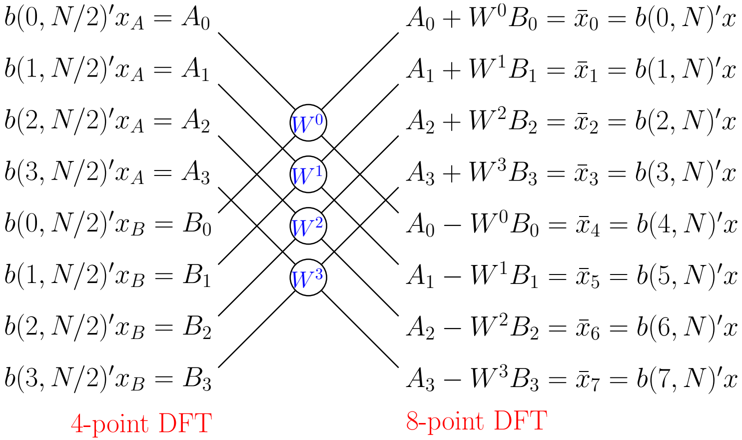
$A_1 - W^1 B_1 = \bar{x}_5$

$A_2 - W^2 B_2 = \bar{x}_6$

$A_3 - W^3 B_3 = \bar{x}_7$

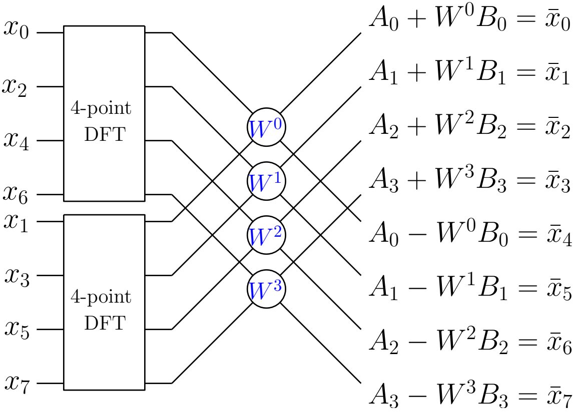
$\bar{x}_{p+N/2}$

Example $N = 8$



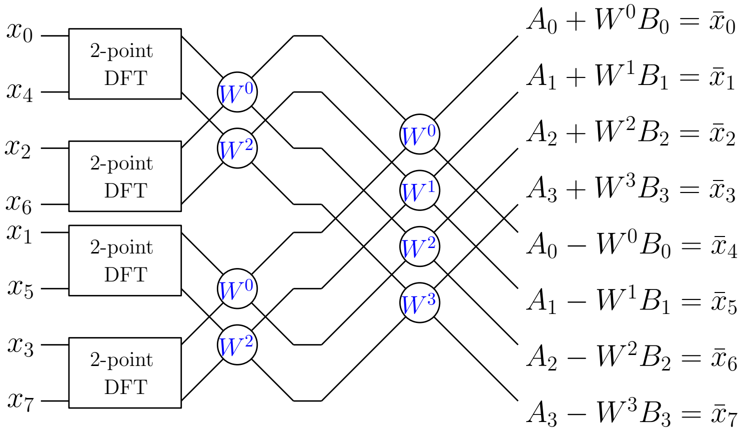
where $x = [x_0, \ldots, x_{N-1}]'$, $x_A = [x_0, x_2, \ldots, x_{N-2}]'$, $x_B = [x_1, x_3, \ldots, x_{N-1}]'$

Example $N = 8$



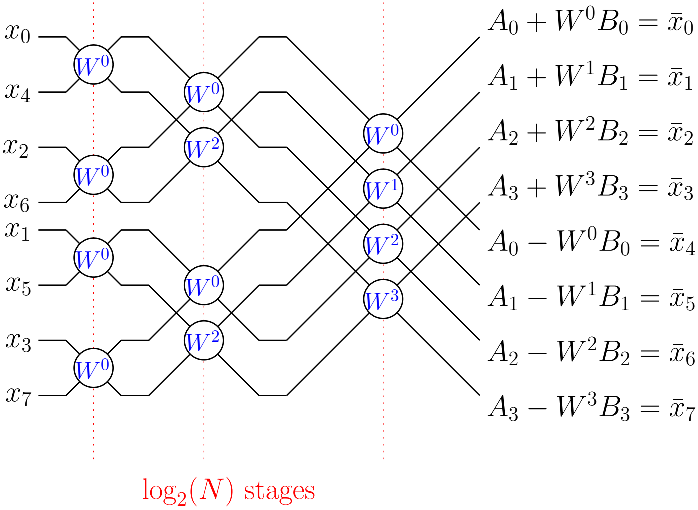
if DFT_N can be obtained as $2 \times DFT_{N/2}$
then DFT_N can be obtained as $4 \times DFT_{N/4}$,
then ... recursion (reducing complexity!)

Example $N = 8$



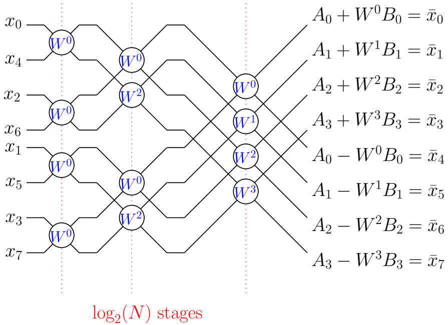
if DFT_N can be obtained as $2 \times DFT_{N/2}$
then DFT_N can be obtained as $4 \times DFT_{N/4}$,
then ... recursion (reducing complexity!)

Example $N = 8$



x_0
x_4
x_2
x_6
x_1
x_5
x_3
x_7
W^0
W^0
W^0
W^0
W^0
W^2
W^0
W^2
W^0
W^1
W^2
W^3
$A_0 + W^0 B_0 = \bar{x}_0$
$A_1 + W^1 B_1 = \bar{x}_1$
$A_2 + W^2 B_2 = \bar{x}_2$
$A_3 + W^3 B_3 = \bar{x}_3$
$A_0 - W^0 B_0 = \bar{x}_4$
$A_1 - W^1 B_1 = \bar{x}_5$
$A_2 - W^2 B_2 = \bar{x}_6$
$A_3 - W^3 B_3 = \bar{x}_7$
$\log_2(N)$ stages

Complexity:

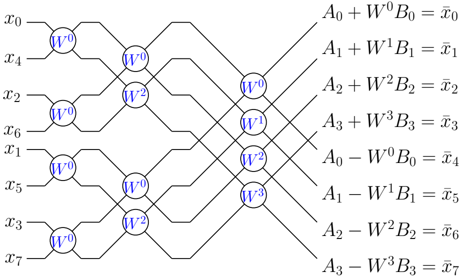


- DFT (N points): $2N^2$
  each DFT point requires $2N$ operations.

- FFT (N points): $\frac{3}{2} N \log_2(N)$
  $N/2$ products and N sum at each stage;
  $\log_2(N)$ stages to compute N samples.

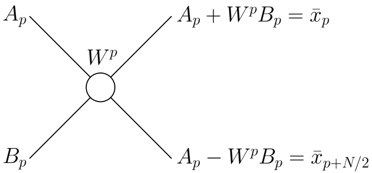
- Each stage takes data from the previous stage only.

Data shuffle:



Decimal	Binary	$\rightarrow$	Bit Reverse	Decimal
0	000	$\rightarrow$	000	0
1	001	$\rightarrow$	100	4
2	010	$\rightarrow$	010	2
3	011	$\rightarrow$	110	6
4	100	$\rightarrow$	001	1
5	101	$\rightarrow$	101	5
6	110	$\rightarrow$	011	3
7	111	$\rightarrow$	111	7

Fast inverse Fourier transform



$$x = \{x_1, \dots x_N\} \quad \xrightarrow{DFT} \quad \bar{x} = \{\bar{x}_1, \dots \bar{x}_N\}$$

FFT:

$$\bar{x} = \mathrm{FFT}(x)$$

Inverse FFT:

$$x = \frac{1}{N}\mathrm{FFT}(\bar{x}^*)^*$$

why?

Applications:

spectral analisys,

filtering (convolution/circular convolution),

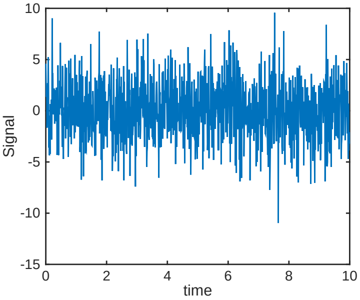
coding (mp3)

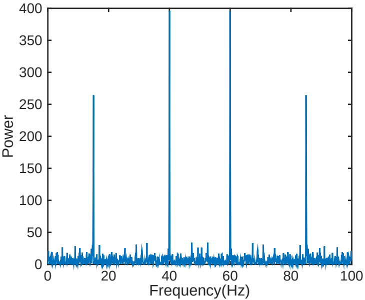
Spectral analysis [on Moodle]

The signal x is given by

- $1.3\sin(2\pi \cdot 15t)$ component at 15 Hz
- $1.7\sin(2\pi \cdot 40(t-2))$ component at 40 Hz
- gaussian noise
- sampled at 100 Hz

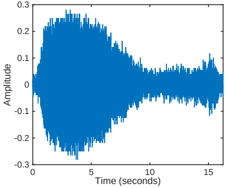
$\bar{x} = \mathrm{FFT}(x, N),\ N = 1024$

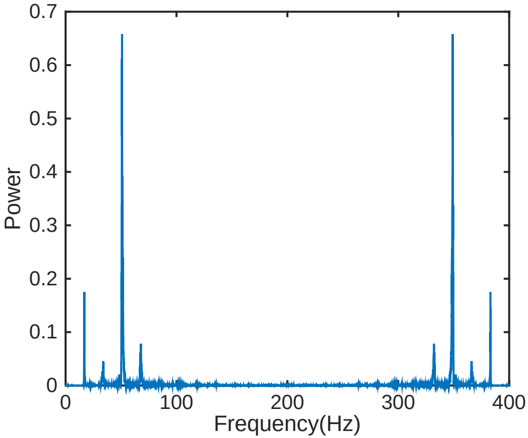




Spectral analysis: whale vocalization [on Moodle]

Pacific blue whale vocalization recorded off the coast of California. From Cornell University Bioacoustics Research Program.





Coding: mp3

Is a lossy compression scheme: information is discarded. How to decide what data to through away?

Spectrum analysis and limitations of human hearing ‣ Auditory masking

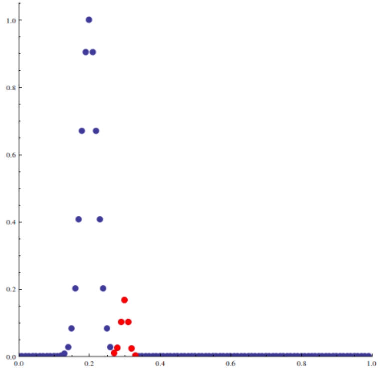
<u>Encode</u>

- FFT
- Frequency masking to approximate spectrum

<u>Decode</u>

- Inverse FFT

JPEG is similar...



Multidimensional filtering: transform and convolution

Multidimensional z-transform

- 1D:

$$X(z) = \sum_{k=0}^{\infty} x(k) z^{-k}$$

- 2D:

$$X(z_1, z_2) = \sum_{k_1=0}^{\infty} \sum_{k_2=0}^{\infty} x(k_1, k_2) z_1^{-k_1} z_2^{-k_2}$$

Discrete time Fourier transform

$$\bar{x}(\omega_1, \omega_2) = \sum_{k_1=0}^{\infty} \sum_{k_2=0}^{\infty} x(k_1, k_2) e^{-jT(\omega_1 k_1 + j\omega_2 k_2)}$$

Discrete Fourier transform

$$\bar{x}(p_1, p_2) = \sum_{k_1=0}^{N-1} \sum_{k_2=0}^{N-1} x(k_1, k_2) e^{-j\frac{2\pi}{N}(p_1 k_1 + p_2 k_2)}$$

2D signals $\{x_{n_1,n_2}\}$ and $\{y_{n_1,n_2}\}$

Linearity

$$a\{x_{n_1,n_2}\} + b\{y_{n_1,n_2}\} \xrightarrow{\mathcal{Z}} aX(z_1, z_2) + bY(z_1, z_2)$$

Convolution

$$\{x_{n_1,n_2}\}*\{y_{n_1,n_2}\} = \sum_{k_1=0}^{\infty} \sum_{k_2=0}^{\infty} x_{n_1-k_1,n_2-k_2} y_{k_1,k_2} \xrightarrow{\mathcal{Z}} X(z_1, z_2)y(z_1, z_2)$$

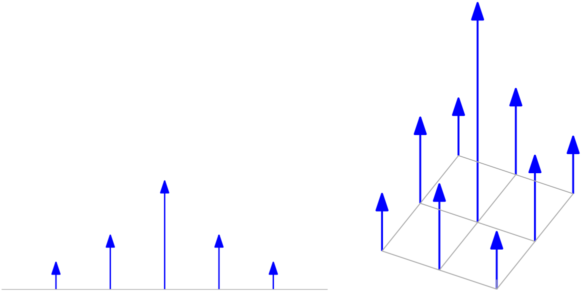
Shift

$$\{x_{m_1 \pm k_1, m_2 \pm k_2}\} \xrightarrow{\mathcal{Z}} z_1^{\pm m_1} z_2^{\pm m_2} X(z_1, z_2) + \text{initial conditions}$$

like 1D...

2D filters (2D convolution)

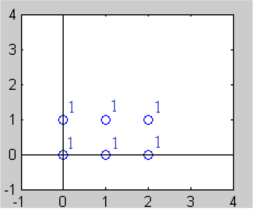
$$y(n_1, n_2) = \sum_{k_1=0}^{\infty} \sum_{k_2=0}^{\infty} h(n_1 - k_1, n_2 - k_2) x(k_1, k_2)$$



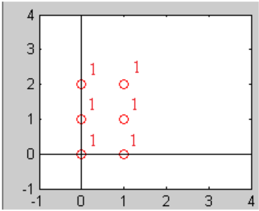
Impulse response $\{h_k\}$ vs $\{h_{k_1,k_2}\}$

2D filters (2D convolution)

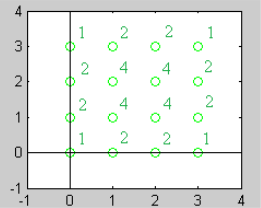
$$y(n_1, n_2) = \sum_{k_1=0}^{\infty} \sum_{k_2=0}^{\infty} h(n_1 - k_1, n_2 - k_2) x(k_1, k_2)$$



$x(n_1, n_2)$



$h(n_1, n_2)$



$x(n_1, n_2) * h(n_1, n_2)$

2D filters (2D convolution)

$$y(n_1, n_2) = \sum_{k_1=0}^{\infty} \sum_{k_2=0}^{\infty} h(n_1 - k_1, n_2 - k_2) x(k_1, k_2)$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1.5 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0.75 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

2D filters (by 2D convolution)

$$y(n_1, n_2) = \sum_{k_1=0}^{\infty} \sum_{k_2=0}^{\infty} h(n_1 - k_1, n_2 - k_2)x(k_1, k_2)$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$ Low pass

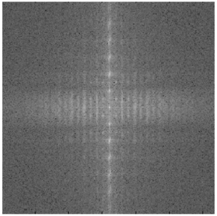
2D filters (by 2D convolution)

$$y(n_1, n_2) = \sum_{k_1=0}^{\infty} \sum_{k_2=0}^{\infty} h(n_1 - k_1, n_2 - k_2) x(k_1, k_2)$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \qquad \begin{bmatrix} 0 & -5 & 0 \\ -5 & 21 & -5 \\ 0 & -5 & 0 \end{bmatrix}$$

2D fast Fourier transform

Spectral analysis: periodicity $\rightarrow$ peaks at specific frequencies

Coding: jpeg

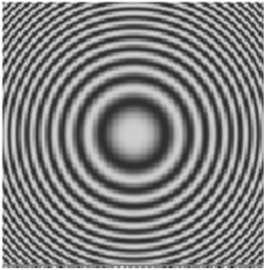
Filtering: FFT $\rightarrow$ product $\rightarrow$ inverse FFT

- Image size $N = 2^{24}$ (16 Mpixel)
- DFT $2N^2 = 2 \cdot 2^{48}$ operations
- FFT $N \log_2(N) = (2^{24}) \cdot 24 < (2^{24}) \cdot (2^5)$
- FFT is $2^{49}/2^{29} = 2^{20} > 10^6$ times faster than DFT

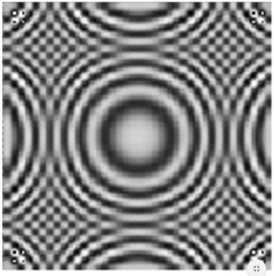
Sampling/Aliasing

(Like for 1D)

Original



Reconstruction



signal has frequencies above Nyquist limit

▸ Aliasing

...2D is conceptually quite similar to 1D.