Chapter 4 The Discrete Fourier Transform

Recall The Fourier Transform of a function $f(t)$ is

$$F(t) = \int_{-\infty}^{\infty} f(s)\, e^{-2\pi tsi} ds$$

and Fourier series

$$f(t) \;\tilde{}\; \sum_{k=0}^{\infty} \left(a_k \cos(2k\pi t) + b_k \sin(2k\pi t)\right)$$

The Fourier Series transform infinitely discrete signal stream $\{a_n, b_n\}$ to a continuous function in so-called the frequency domain. We also recall that Z-transform of x_n at $z = e^{-2\pi t}$ is

$$\sum_{k=0}^{\infty} x_k z^{-k} = \sum_{k=0}^{\infty} x_k \left(e^{-2\pi t}\right)^{-k} = \sum_{k=0}^{\infty} x_k e^{2\pi tk} = \sum_{k=0}^{\infty} x_k \left(\cos(2k\pi t) + i \sin(2k\pi t)\right)$$

- ## Section 4.1 Real-time Processing
  - Consider finite data set $x = \{x_0, x_1, ..., x_{n-1}\}$
  - In Chap 3, we learned that a causal filter F is basically a convolution $y = h * x$. In many applications, the original signal x is transmitted through a channel (noise). At the other end a distorted signal y is received. The question is how to recover x from y if we have knowledge about the noise. In other words, we want to solve x from equation $y = h * x$ with given h. This can be

achieved by applying $Z - transform : Z(y) = Z(h)\, Z(x)$. So $x = Z^{-1}(Z(y)/Z(h))$. But this process is very time consuming. It is difficult, if not impossible, to process in real time.

– Here we introduce another tool called "Discrete Fourier Transform" DFT that can be done in real time

– DFT of a finite signal stream x is a sequence of the frequency-domain objects

$$\hat{x} = F(x) = \{\hat{x}_0, \hat{x}_1, ..., \hat{x}_{n-1}\}, \quad \xi = e^{\frac{2\pi}{n}i} \text{ (n-th complex root of } 1\text{)}$$

$$\hat{x}_k = \sum_{j=0}^{n-1} x_j \xi^{-jk} = \sum_{j=0}^{n-1} x_j e^{-\frac{2\pi jk}{n}i}$$

$$= \sum_{j=0}^{n-1} x_j \left(\cos\left(\frac{2\pi jk}{n}\right) - i \sin\left(\frac{2\pi jk}{n}\right)\right)$$

– Recall that matrix multiplication: for $A = (a_{ij})_{m\times n}$, $B = (b_{ij})_{n\times p}$, $AB = (c_{ij})_{m\times p}$, where

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}.$$

– Using matrix form, let F be the $n \times n$ symmetric matrix

$$F = [f_{ij}]_{ij}, \; f_{ij} = \xi^{-(i-1)(j-1)}$$

$$F = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \xi^{-1} & \xi^{-2} & \cdots & \xi^{-(n-1)} \\ 1 & \xi^{-2} & \xi^{-4} & \cdots & \xi^{-2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \xi^{-(n-1)} & \xi^{-2(n-1)} & \cdots & \xi^{-(n-1)^2} \end{bmatrix}, \ \hat{y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \\ \vdots \\ \hat{y}_n \end{bmatrix}, x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{n-1} \end{bmatrix}$$

and consider the matrix equation

$$\hat{y} = Fy$$

– One can verify that, for $k = 0, 1, ..., n-1,$ (note that the k-th component of vector x is $(x)_k = x_{k-1}$

$$\hat{y}_{k+1} = \sum_{l=1}^{n} f_{k+1,l} y_l = \sum_{l=1}^{n} \xi^{-k(l-1)} y_l \overset{l-1=j}{=} \sum_{j=0}^{n-1} \xi^{-kj} x_j = \hat{x}_k$$

– The DFT may be written in matrix form $\hat{x} = Fx$

$$\begin{bmatrix} \hat{x}_0 \\ \hat{x}_1 \\ \hat{x}_2 \\ \vdots \\ \hat{x}_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \xi^{-1} & \xi^{-2} & \cdots & \xi^{-(n-1)} \\ 1 & \xi^{-2} & \xi^{-4} & \cdots & \xi^{-2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \xi^{-(n-1)} & \xi^{-2(n-1)} & \cdots & \xi^{-(n-1)^2} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{n-1} \end{bmatrix}$$

– Note if $\xi^\alpha \neq 1$

$$\sum_{j=0}^{n-1} \xi^{ja} = \sum_{j=0}^{n-1} (\xi^a)^j = \frac{1-\xi^{na}}{1-\xi^\alpha} = 0 \quad (\xi^n = 1)$$

Note that $\xi^\alpha \neq 1$ for $0 < \alpha < n$. So

$$\sum_{j=0}^{n-1} \xi^{ja} = \begin{cases} n & \text{if } \xi^a = 1 \ (or \ \ a = kn) \\ 0 & otherwise \end{cases}$$

– Note that $\bar{\xi} = e^{-\frac{2\pi}{n}i}$. Now the (i, j)-entry if $F\bar{F}$ is

$$\left(F\bar{F}\right)_{ij} = \sum_{k=1}^{n} f_{ik}\bar{f}_{kj} = \sum_{k=1}^{n} \xi^{-(i-1)(k-1)}\xi^{(k-1)(j-1)}$$
$$= \sum_{k=1}^{n} \xi^{(j-i)(k-1)} = \begin{cases} n & \text{if } \ i = j \\ 0 & otherwise \end{cases}$$

So $F\bar{F} = nI$, or $F^{-1} = \bar{F}/n$,

$$x = \frac{1}{n}\bar{F}\hat{x} = \frac{1}{n}\bar{F}\hat{x} = \frac{1}{n}\sum_{j=0}^{n-1} \hat{x}_j \xi^{jk}$$

- Section 4.2 Properties of DFT
  - DFT is linear bijective maps from C^n to itself
  - Circular convolution of two n-tuples h and $x : z = h * x = \{z_0, z_1, ..., z_{n-1}\}$

$$\begin{aligned} z_k &= \sum_{j=0}^{n-1} h_j x_{k-j} = h_0 x_k + h_1 x_{k-1} + ... + h_k x_0 + h_{k+1} x_{n-1} + ... + h_{n-1} x_{k+1} \\ &= (h_0, h_1, ..., h_{n-1}) \circ (x_k, x_{k-1}, x_{k-2,}..., x_0, x_{n-1}, x_{n-2,}..., x_{k+1}) \end{aligned}$$

where the subscripts are calculated module n (i.e., $x_{-1} = x_{n-1}, x_{-2} = x_{n-2}, ...$).

  - Note that for $w = x * h$

$$w_k = \sum_{j=0}^{n-1} x_j h_{k-j} = x_0 h_k + x_1 h_{k-1} + ... + x_k h_0 + x_{k+1} h_{n-1} + ... + x_{n-1} h_{k+1}$$

So

$$h * x = x * h.$$

– Another way to look at $z = h * x$: $z_k = h_0 x_k + ... + h_k x_0 \ + h_{k+1} x_{n-1} + ... + h_{n-1} x_{k+1}$

$$\left\{ h_0, h_1, \overrightarrow{...}, h_k, \vdots\ h_{k+1}, \overrightarrow{...}, h_{n-1} \right\}$$
$$\left\{ x_0, x_1, \underleftarrow{...}, x_k, \vdots\ x_{k+1}, \underleftarrow{...}, x_{n-1} \right\}$$

– Example: Given $x = \{1, 1, 2\}$, $y = \{-1, 3, 4\}$.

$$\left\{ \ 1, \vdots\ 1, 2 \right\}, \ \left\{ 1, 1, \vdots\ 2 \right\}, \ \left\{ \ 1, 1, 2 \right\}$$
$$\left\{ -1, \vdots\ 3, 4 \right\}, \ \left\{ -1, 3, \vdots\ 4 \right\}, \ \left\{ -1, 3, 4 \right\}$$

So $x * y = \{9, 10, 5\}$

– One may extend x periodically to all integer m : $x_m = x_{m+kn}$, and extend h_k by 0 for $m \geq n$. With this extension, circular convolution is the same as the discrete convolution defined in Chapter 3. To see this, we write

$$\tilde{h} = \{h_0, h_1, ..., h_{n-1}, 0, 0, ...\} ,$$
$$\tilde{x} = \{\tilde{x}_0, \tilde{x}_1 . \tilde{x}_2, ...\}$$
$$= \{x_0, x_1, ..., x_{n-1}, x_0, x_1, ..., x_{n-1}, x_0, x_1, ..., x_{n-1}, ...\}$$
$$\tilde{x}_k = x_{k-jn} \ \text{ for } k > n.$$

Let $\tilde{h} * \tilde{x} = \{u_0, u_1, u_2, ...u_{n-1}, ...\}$ .Then

$$u_{n+k} = \sum_{j=0}^{n+k} h_j \tilde{x}_{n+k-j} = \sum_{j=0}^{n-1} h_j \tilde{x}_{n+k-j} = \sum_{j=0}^{n-1} h_j x_{k-j} = z_k$$

So in that sense

$$h * x = \tilde{h} * \tilde{x}$$

– Example: We know that for $x = \{1, 1, 2\}$, $y = (-1, 3, 4)$, $x * y = (9, 10, 5)$ .One can also use the discrete convolution to compute the circular convolution:

$$\tilde{x} = \{1, 1, 2, 0, 0, 0, ...\}\ ,\ \tilde{y} = \{-1, 3, 4, -1, 3, 4, -1, 3, 4, ...\}$$

$$\begin{aligned} \tilde{x} * \ \tilde{y} &= \{-1, 2, 5, (-1 + 4 + 6), (3 - 1 + 8)\ , (4 + 3 - 2)\ , ...\} \\ &= \{-1, 2, 5, 9, 10, 5, 9, 10, 5, ...\} \end{aligned}$$

– Define coordinate-wise product $\circ$

$$x \circ y = \{x_0 y_0, x_1 y_1, ..., x_{n-1} y_{n-1}\}$$

– $F(x * y) = F(x) \circ F(y)$ (or $x * y \mapsto \hat{x} \circ \hat{y}$)

– $F(x \circ y) = F(x) * F(y)/n$

– Example (Using circular convolution to modify x_k by its surroundings)

(i) Let $h = \{h_0, h_1, 0, 0, ..., 0, h_{n-1}\}$ .Then

$$z_k = \sum_{j=0}^{n-1} h_j x_{k-j} = h_0 x_k + h_1 x_{k-1} + h_{n-1} x_{k-(n-1)} = h_0 x_k + h_1 x_{k-1} + h_{n-1} x_{k+1}$$

Therefore, x_k is modified only by itself and its two immediate neighbors, if $1 \leq k \leq n-2$,i.e., x_k is not an end points. For endpoints

$$z_0 = h_0 x_0 + h_1 x_{-1} + h_{n-1} x_{-(n-1)} = h_0 x_0 + h_1 x_{n-1} + h_{n-1} x_1$$
$$z_{n-1} = h_0 x_{n-1} + h_1 x_{(n-1)-1} + h_{n-1} x_{(n-1)-(n-1)} = h_0 x_{n-1} + h_1 x_{n-2} + h_{n-1} x_0$$

So x_0 is modified by x_0, x_1,and x_{n-1} that is at the opposite side. This is called "edge effect"

(ii) Let $h = \{h_0, h_1, h_2, 0, ..., 0, h_{n-2}, h_{n-1}\}$ .Then z_k is modified by itself, two immediately before and two immediately after

- Section 4.4 The Fast Fourier Transform (http://paulbourke.net/miscellaneous/dft/)

– In DFT, each $\hat{x}_k$ requires n multiplications. So total it requires $O\left(n^2\right)$ multiplication operations

– FFT can reduce the number to $O\left(n \ln n\right)$

– The Fast Fourier Transform is an algorithm that reduces the computer implementation time

– data set of even signal $x = \{x_0, x_1, ..., x_{2m-1}\}, \quad n = 2m$

– Let $y = \{x_0, x_2, x_{4,}..., x_{n-1}\}$ be all signals with even indices, and $z = \{x_1, x_3, ..., x_{n-1}\}$ be odd-index signals. Set

$$\xi = e^{\frac{2\pi}{n}i} = e^{\frac{\pi}{m}i},\ \zeta = \xi^2 = e^{\frac{2\pi}{m}i}$$ (m-th complex root of unity)

– Then DFT of frame length $n = 2m$:

$$\begin{aligned}
\hat{x}_k = \sum_{j=0}^{n-1} x_j \xi^{-jk} &= \sum_{j=0, j=even}^{n-1} x_j \xi^{-jk} + \sum_{j=0, j=odd}^{n-1} x_j \xi^{-jk} \\
&= \sum_{l=0}^{m-1} x_{2l} \xi^{-2lk} + \sum_{l=0}^{m-1} x_{2l+1} \xi^{-(2l+1)k} \\
&= \sum_{l=0}^{m-1} y_l \zeta^{-lk} + \sum_{l=0}^{m-1} z_l \zeta^{-lk} \xi^{-k} \\
&= \hat{y}_k + \hat{z}_k \xi^{-k}, \quad k = 0, 1, ...2m-1
\end{aligned}$$

– Note that for $k = 1, 2, ..., m-1, \quad \hat{y}_k$ and $\hat{z}_k$ are DFT of frame length m,

– For $d = m + k, \;\; k = 0, 1, ..., m - 1$, since $\zeta^m = \xi^{2m} = 1, \; \xi^m = e^{-\pi i} = -1$, we see

$$\zeta^{-ld} = \zeta^{-l(m+k)} = \zeta^{-lm}\zeta^{-lk} = \zeta^{-lk}$$
$$\xi^{-d} = \xi^{-m-k} = -\xi^{-k}$$

Therefore

∗ For $k = 0, 1, ..., m - 1$

$$\hat{x}_k = \hat{y}_k + \hat{z}_k\xi^{-k},$$
$$\hat{x}_{m+k} = \hat{y}_k + \hat{z}_k\xi^{-(m+k)} = \hat{y}_k - \hat{z}_k\xi^{-k}$$

∗ where $\hat{y}_k$ and $\hat{z}_k$ are DFT of frame length m

∗ So FFT algorithm reduces a DFT of frame length $2m$ to the sum of two DFT of frame length m.

∗ Total number of multiplications required: $2\,(m)^2 = n^2/2$,half as much as DFT.

∗ If $n = 2^a$,then apply one step of FFT, the number of multiplications reduces to

$$\frac{n^2}{2} = 2^{2a-1}$$

– Recall that DFT has matrix form: $\hat{x} = Fx$

– FFT basically is a matrix factorization: Let F_n be the DFT matrix of frame length n.Then

$$F_n = B_n \begin{bmatrix} F_m & 0 \\ 0 & F_m \end{bmatrix} P_n$$

where

$$B_n = \begin{bmatrix} I_m & D_m \\ I_m & D_m \end{bmatrix}, \quad D_m = diag\left\{1, \xi_n^{-1}, \xi_n^{-2}, ..., \xi_n^{-(m-1)}\right\}, \; \xi_n = e^{-\dfrac{2\pi}{n}}$$

and P_n is a permutation matrix of $0's$ and $1's$ such that

$$P_n x = \begin{bmatrix} y \\ z \end{bmatrix}, \quad y \text{ even index term, } z \text{ odd index term}$$
$$= [x_0, x_2, x_4, ..., x_{2m}, x_1, x_3, x_{5,}..., x_{2m-1}]$$

– $n = 4, 6$

$$P_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad P_6 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

– If $n = 2m = 4l$, then

$$F_n = B_n \begin{bmatrix} F_m & 0 \\ 0 & F_m \end{bmatrix} P_n$$

$$= B_n \begin{bmatrix} B_m \begin{bmatrix} F_l & 0 \\ 0 & F_l \end{bmatrix} P_m & 0 \\ 0 & B_m \begin{bmatrix} F_l & 0 \\ 0 & F_l \end{bmatrix} P_m \end{bmatrix} P_n$$

$$= B_n \begin{bmatrix} B_m & 0 \\ 0 & B_m \end{bmatrix} \begin{bmatrix} F_l & 0 & 0 & 0 \\ 0 & F_l & 0 & 0 \\ 0 & 0 & F_l & 0 \\ 0 & 0 & 0 & F_l \end{bmatrix} \begin{bmatrix} P_m & 0 \\ 0 & P_m \end{bmatrix} P_n$$

– Note that

$$B_2 = F_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

– If $n = 2^a$, then we may continue this kind of factorization:

$$F_n = B_n \begin{bmatrix} B_{n/2} & 0 \\ 0 & B_{n/2} \end{bmatrix} \cdots \begin{bmatrix} B_4 & 0 & \cdots & 0 \\ 0 & B_4 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & B_4 \end{bmatrix} \begin{bmatrix} F_2 & 0 & 0 & \cdots & 0 \\ 0 & F_2 & 0 & \cdots & 0 \\ 0 & 0 & F_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & F_2 \end{bmatrix} Q_n$$

where Q_n is the product of permutation matrices

$$Q_n = \begin{bmatrix} P_4 & 0 & \cdots & 0 \\ 0 & P_4 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & P_4 \end{bmatrix} \cdots \begin{bmatrix} P_{n/2} & 0 \\ 0 & P_{n/2} \end{bmatrix} P_n$$

– Q_n depends only on $n = 2^a$. It is basically re-arrangement of x. It can be pre-coded and stored for use with using computing power.

– Note that in the decomposition of F_n, it is the product of a total of a matrices that are (2×2) $-$block-diagonal. So total number of multiplication operations for $n = 2^a$:

$$a\,(2n) = 2n \log_2 n$$

- Section 4.5 Imaging Processing

- Two dimensional DFT
  - For a two-dimensional data stream $X = [x_{ij}]_{m \times n}$,its DFT $\hat{X} = [\hat{x}_{ij}]_{m \times n}$ is

$$\hat{x}_{ij} = \sum_{p,q=1}^{m,n} x_{pq} \xi_m^{=(p-1)(i-1)} \xi_n^{=(q=1)(j=1)}, \quad \xi_k = e^{\frac{2\pi}{k} i}$$ is the primitive kth root of unity

  - $\hat{X} = F_m X F_n, \quad F_k = [f_{ij}]_{k \times k}, \quad f_{ij} = \xi_k^{-(i-1)(j-1)}$
  - For $H = [h_{ij}]_{m \times n}$, $X = [x_{ij}]_{m \times n}$,the $2d$ circular convolution is defined as $Y = H * X = [y_{ij}]$

$$y_{ij} = \sum_{p,q=1}^{m,n} h_{pq} x_{(i-p+1),(j-q+1)},$$

    where $x_{-p,-q} = x_{m-p,n-q}$.

  - In particular, in the summation of y_{ij}, the contribution of $x_{i,j+1}$ is when

$$\begin{aligned} i - p + 1 = i, i \pm m \quad &\Longrightarrow p = 1, \\ j - q + 1 = j + 1, j + 1 \pm n \quad &\Longrightarrow q = n \end{aligned}$$

    i.e.,

$$h_{pq} x_{(i-p+1),(j-q+1)} = h_{1,n} x_{i,j+1}$$

- Properties:
  - Inverse: $X = (F_m)^{-1}\hat{X}(F_n)^{-1} = \bar{F}_m\hat{X}\bar{F}_n/(mn)$

$$x_{ij} = \frac{1}{mn}\sum_{p,q=1}^{m,n}\hat{x}_{pq}\xi_m^{(p-1)(i-1)}\xi_n^{(q=1)(j=1)}$$

  - $X * Y \rightarrow \hat{X} \circ \hat{Y}$ (entry-wise product: if $X = [x_{ij}]$, $Y = [y_{ij}]$, then $X \circ Y = [x_{ij}y_{ij}]$
  - $\hat{X} \circ \hat{Y} \rightarrow X * Y/(mn)$
  - local modification and Edge Effect: Let $Y = H * X$ where

$$H = \begin{bmatrix} \begin{bmatrix} K_{1,1} & K_{1,2} \\ K_{2,1} & K_{2,2} \end{bmatrix} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \begin{bmatrix} L_{n-1,n-1} & L_{(n-1),n} \\ L_{n,(n-1)} & L_{n,n} \end{bmatrix} \end{bmatrix}$$

    Then $y_{ij} = K_{1,1}x_{i,j}+$surrounding

- Image Processing (black & White):
  - An B&W image file consists of $m \times n$ pixels $X = [x_{ij}]_{m\times n}$. Each pixel x_{ij} is a integer intensity representing a grey scale $from$ 0 to $W.$

– For instance $m = n = 128$, $W = 255$,provide recognizable image of human face

– Enhance contrast, brightness, redeye correction, etc., may be realized by a filter, or circular convolution

$$Y = H * X = [y_{ij}],$$
$$y_{ij} = \sum_{p,q=1}^{m,n} h_{pq} x_{(i-p),(j-q)}$$

it modifies the pixel (i,j) according to its nearby pixels with certain weight H if

$$H = \begin{bmatrix} K_{r\times s} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & L_{r\times s} \end{bmatrix}$$

– However, this also could cause "edge effects" at the borders by the pixel at the opposite edge. This edge effects may be corrected by cropping.

- Deblurring: Suppose that an image file X is transmitted through a channel (such image scanning). This image is filtered by H to become a blurred image $Y = H * X$. It is very time consuming to recover the original image X by solving a huge system of equations. With DFT, we compute

$$\hat{Y} = \hat{H} \circ \hat{X} = \left[\hat{h}_{ij}\hat{x}_{ij}\right]$$

then solve

$$\hat{x}_{ij} = \frac{\hat{y}_{ij}}{\hat{h}_{ij}}$$

The original image pixel at (i,j) can be recovered by inverse DFT

$$\begin{aligned} x_{ij} &= \frac{1}{mn} \sum_{p,q=1}^{m,n} \hat{x}_{pq} \xi_m^{(p-1)(i-1)} \xi_n^{(q=1)(j=1)} \\ &= \frac{1}{mn} \sum_{p,q=1}^{m,n} \frac{\hat{y}_{pq}}{\hat{h}_{pq}} \xi_m^{(p-1)(i-1)} \xi_n^{(q=1)(j=1)} \end{aligned}$$

– Example in 72 (see also project 4.15)

- Homework: 4.3,4.4,4.8, 4.12, 4.13
- Project: 4.9, 4.15