CS2108 — INTRODUCTION TO MEDIA COMPUTING

PROF. NG TECK KHIM

*arsatis**

CONTENTS

*author: https://github.com/arsatis

1 INTRODUCTION

Lecture 1
10th January 2023

Key formulas used throughout the course include:

- **Euler's formula**:

$$e^{j\theta} = \cos\theta + j\sin\theta \tag{1}$$

  Note: complex numbers (e.g., $a + bi$) are written instead as $a + bj$ in this module.

- **Fourier series**:

$$\mathrm{X}(f_k) = \int_0^{\mathrm{T}} x(t)e^{-j2\pi f_k t}\,dt \tag{2}$$

- **Fourier transform**:

$$\mathrm{X}(\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t}\,dt \tag{3}$$

- **Inverse Fourier transform**:

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \mathrm{X}(\omega)e^{j\omega t}\,d\omega \tag{4}$$

2 BASIS FUNCTIONS

Lecture 2
17th January 2023

A **basis function** is an element of a particular basis for a function space. Every function in the function space can be represented as a linear combination of basis functions, just as every vector in a vector space can be represented as a linear combination of basis vectors.

Other key definitions:

- **Basis set**: the set of vectors in $\mathbb{R}^{\mathrm{N}}$ that can be used to represent all vectors in $\mathbb{R}^{\mathrm{N}}$. If $\{v_i\}_{1\leq i\leq \mathrm{N}}$ is a basis set, then any vector $x \in \mathbb{R}^{\mathrm{N}}$ can be written as

$$x = \sum_{i=1}^{\mathrm{N}} c_i v_i \tag{5}$$

i.e., the basis set is said to *span* $\mathbb{R}^{\mathrm{N}}$.

- **Orthogonal basis set**: a basis set with all vectors orthogonal to each other, i.e.:

$$v_i^{\mathrm{T}} v_j = \begin{cases} 0 & \text{if } i \neq j \\ |v_i|^2 & \text{if } i = j \end{cases}$$

  To get each c_i from equation 5, we can simply get the dot product of x with each unit vector of v_i, if all v_i are orthogonal to each other.

2.1 *Sinusoidal functions*

Key terminology:

- **Frequency** (f): the number of occurrences of a repeating event per unit of time.
- **Amplitude**: the relative strength of sound waves, which we perceive as loudness or volume.
- **Phase** (ϕ): how far a sinusoidal function is shifted horizontally from the usual position.
- **Period** (T): the time it takes for two successive crests (one wavelength) to pass a specified point.

(*Theorem 1*) Given

$$h(t) = \sin(pt + a) + \cos(qt + b) \text{ or}$$

$$h(t) = \sin(pt + a) + \sin(qt + b) \text{ or}$$

$$h(t) = \cos(pt + a) + \cos(qt + b)$$

where a, b are constants, if p/q is rational, then $h(t)$ is a periodic signal.

In other words, if either p or q is a multiple of π and the other is not, then $h(t)$ is not a periodic signal.

If $h(t)$ is periodic, then:

- **Fundamental period of** $h(t)$: $\mathrm{T}_0 = \mathrm{LCM}(\mathrm{T}_1, \mathrm{T}_2) = p' * 2\pi/p$, where:
  - $\mathrm{T}_1 = 2\pi/p$.
  - $\mathrm{T}_2 = 2\pi/q$.
  - $\mathrm{T}_1/\mathrm{T}_2 = q/p = q'/p'$.
- **Fundamental frequency of** $h(t)$: $f_0 = \mathrm{GCD}(p, q)$.

2.2 *Sinusoidal basis functions*

Sine and cosine are the basis functions for the Hilbert space of L^2 functions on the circle, and any sinusoidal functions can be derived from a linear combination of them. Specifically,

$$\text{sig}[n] = \sum_{k=0}^{K-1} (a_k \sin(2\pi f_k t_n) + b_k \cos(2\pi f_k t_n)) \tag{6}$$

where:

- f_k rep. the discretized frequency $k\Delta f$.
- t_n rep. the discretized timestep $n\Delta t$.
- a_k, b_k rep. the amplitudes of sine & cosine at frequency f_k.
- $\text{sig}[n]$ rep. some signal sampled during time t_n.
- K rep. the total number of frequencies used to generate the signal.

Even for non-zero phase, we have:

$$\sum_{k=0}^{K} A_k \sin(2\pi f_k t + \phi_k) = \sum_{k=0}^{K} A_k \left(\sin(2\pi f_k t)\cos(\phi_k) + \cos(2\pi f_k t)\sin(\phi_k)\right)$$

$$\sum_{k=0}^{K} B_k \cos(2\pi f_k t + \phi_k) = \sum_{k=0}^{K} B_k \left(\cos(2\pi f_k t)\cos(\phi_k) - \sin(2\pi f_k t)\sin(\phi_k)\right)$$

(*Theorem 2*) Let $f_m = mf_0$ and $f_n = nf_0$, where $m, n \in \mathbb{Z}$. Then,

f_0 rep. the fundamental frequency, i.e., $\gcd(f_m, f_n)$.

$$\int_0^T \sin(2\pi f_m t)\sin(2\pi f_n t)dt = \begin{cases} 0 & \text{if } m \neq n \\ \frac{T}{2} & \text{if } m = n \end{cases}$$

$$\int_0^T \cos(2\pi f_m t)\cos(2\pi f_n t)dt = \begin{cases} 0 & \text{if } m \neq n \\ \frac{T}{2} & \text{if } m = n \end{cases}$$

$$\int_0^T \sin(2\pi f_m t)\cos(2\pi f_n t)dt = 0$$

In other words, sine and cosine are always orthogonal to each other, and orthogonal to themselves when their frequencies are not the same.

However, the frequencies must be a multiple of the fundamental frequency.

2.3 *Fourier series*

Since sine and cosine form an orthogonal basis set for the space of sinusoidal functions, we could retrieve each a_k, b_k by performing a dot product of the signal (i.e., equation 6) with each $\cos(2\pi f_k t)$ and $\sin(2\pi f_k t)$ for all f_k.

Specifically, we rep. the amplitude of the k^{th} frequency in the signal as:

$$
\begin{aligned}
\mathrm{X}(f_k) &= \int_0^{\mathrm{T}} \mathrm{sig}[n]\cos(2\pi f_k t)dt - j\int_0^{\mathrm{T}} \mathrm{sig}[n]\sin(2\pi f_k t)dt && \Rrightarrow \text{rep. as complex numbers} \\
&= \int_0^{\mathrm{T}} \mathrm{sig}[n]e^{-j2\pi f_k t}dt && \Rrightarrow \text{use Euler's formula (eq. 1)} \\
& && \Rrightarrow \text{this is Fourier series (eq. 2)} \\
&= \int_0^{\mathrm{T}} \left[\sum_{m=0}^{\mathrm{K}-1} (a_m \sin(2\pi f_m t_n) + b_m \cos(2\pi f_m t_n))\right] e^{-j2\pi f_k t}dt && \Rrightarrow \text{expand with eq. 6} \\
&= \int_0^{\mathrm{T}} \left[\sum_{k=0}^{\mathrm{K}-1} \mathrm{A}_m \sin(2\pi f_m t + \phi_m)\right] e^{-j2\pi f_k t}dt && \Rrightarrow \text{rep. cosine as shifted sine} \\
&= -\frac{\mathrm{T}}{2}\mathrm{A}_k e^{-j\phi_k} \\
&= \frac{\mathrm{T}}{2}\mathrm{A}_k\left(\sin(\phi_k) - j\cos(\phi_k)\right)
\end{aligned}
$$

We represent the amplitude of each frequency as a complex number because of the following desirable properties:

$$|\mathrm{X}(f_k)| = \frac{\mathrm{T}}{2}\mathrm{A}_k \tag{7}$$

$$\theta = \tan^{-1}\left(\frac{-\cos(\phi_k)}{\sin(\phi_k)}\right) \tag{8}$$

3 FOURIER TRANSFORM

Lecture 3
31st January 2023

We can treat non-periodic signals as a periodic signal with period $\mathrm{T} = \infty$. Then, we can apply the same technique as above to retrieve the frequencies present in the non-periodic signal. This is known as **Fourier transform**.

In other words, instead of integrating over the fundamental period T in equation 2, we integrate over the period $-\infty$ to ∞ (as seen in equation 3).

3.1 *Inverse Fourier transform*

In contrast to Fourier transform, **inverse Fourier transform** brings the frequency domain representation of signals back to its time domain representation. The equations for each are as follows:

$$\mathrm{X}(\omega) = \int_{-\infty}^{\infty} x(t) e^{-j\omega t}\, dt \qquad \text{Fourier transform}$$

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \mathrm{X}(\omega) e^{j\omega t}\, d\omega \qquad \text{inverse Fourier transform}$$

where $\omega = 2\pi f$.

3.2 *Discretization*

The formula for Fourier transform can be discretized to enable numerical computations.

We can discretize both time and frequency using:

$$t = n\Delta t = n\frac{\mathrm{T}}{\mathrm{N}}$$

$$\omega = k\omega_0 = k2\pi f_0 = k2\pi\frac{1}{\mathrm{T}}$$

where N rep. the total number of points sampled within one period.

$$\begin{aligned}
\mathrm{X}(\omega) &= \int_{-\infty}^{\infty} x(t) e^{-j\omega t}\, dt \\
&= \sum_{n=-\infty}^{\infty} x[n] e^{-j\omega n\Delta t} \Delta t && \text{discretization of } t \\
&= \frac{\mathrm{T}}{\mathrm{N}} \sum_{n=-\infty}^{\infty} x[n] e^{-j\omega \frac{n\mathrm{T}}{\mathrm{N}}} && \text{DTFT} \\
\mathrm{X}[k] &= \frac{\mathrm{T}}{\mathrm{N}} \sum_{n=-\infty}^{\infty} x[n] e^{-j(k2\pi\frac{1}{\mathrm{T}})\frac{n\mathrm{T}}{\mathrm{N}}} && \text{discretization of } f \\
&= \frac{\mathrm{T}}{\mathrm{N}} \sum_{n=-\infty}^{\infty} x[n] e^{-j2\pi k\frac{n}{\mathrm{N}}} && \text{DFT}
\end{aligned}$$

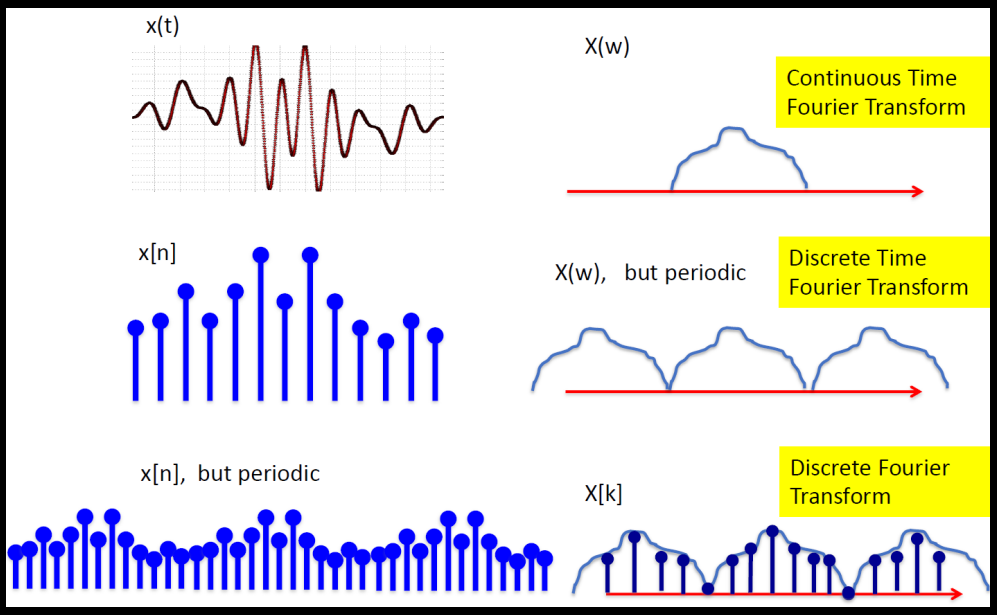
With this discretization, it can be shown that $\mathrm{X}[k]$ is periodic with period N.

Similarly, inverse Fourier transform can also be discretized, and the resulting $x[n]$ is also periodic with period N. Specifically, the discrete Fourier transform (DFT) pair is:

$$\mathrm{X}[k] = \sum_{n=0}^{\mathrm{N}-1} x[n] e^{-j2\pi k \frac{n}{\mathrm{N}}} \qquad \text{DFT}$$

$$x[n] = \frac{1}{\mathrm{N}} \sum_{k=0}^{\mathrm{N}-1} \mathrm{X}[k] e^{j2\pi k \frac{n}{\mathrm{N}}} \qquad \text{iDFT}$$

Note that when the time axis is discretized, the frequency axis is periodic (vice versa).



3.3 *Impulse*

Lecture 4
7th February 2023

The **Dirac delta function** is also known as:

- **Impulse**, has infinite height, infinitesimally-small width, and the area under the impulse = 1 on a continuous time axis.
  - Continuous time Fourier transform for impulse train:

$$p(t) = \sum_{n=-\infty}^{\infty} \delta(t - n\mathrm{T_S}) \Leftrightarrow \mathrm{P}(\omega) = \sum_{k=-\infty}^{\infty} \omega_0 \delta(\omega - k\omega_0)$$

  - Continuous time Fourier transform for rectangular function:

$$\text{rect}(t) = \begin{cases} 1 & \text{if } -0.5 \leq t \leq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

$$x(t) = \text{rect}(t) \Leftrightarrow \text{X}(\omega) = \frac{2\sin\left(\frac{\omega}{2}\right)}{\omega} = \text{sinc}\left(\frac{\omega}{2\pi}\right)$$

- **Kronecka delta**, $\delta[n] = \begin{cases} 1 & \text{if } n = 0 \\ 0 & \text{otherwise} \end{cases}$ on a discrete time axis.
  - Discrete Fourier transform pairs:

$$\sum_{m=-\infty}^{\infty} \delta[n + m\text{N}] = \begin{cases} 1 & \text{if } n = 0, \pm\text{N}, \pm 2\text{N}, \ldots \\ 0 & \text{otherwise} \end{cases}$$

$$\cos\left(\frac{2\pi}{\text{N}} k_0 n\right) \Leftrightarrow \frac{\text{N}}{2}(\delta[k - k_0] + \delta[k + k_0])$$

$$\sin\left(\frac{2\pi}{\text{N}} k_0 n\right) \Leftrightarrow \frac{j\text{N}}{2}(-\delta[k - k_0] + \delta[k + k_0])$$

Some important features associated with impulses include:

- Time domain uniform $\Leftrightarrow$ frequency domain is an impulse.

  Frequency domain uniform $\Leftrightarrow$ time domain is an impulse.
- Time domain is a train of impulses $\Leftrightarrow$ frequency domain is a train of impulses.

If a uniform/flat shape in one domain is padded with zeros, it will take the shape of a sinc function in the other domain.

impulse train: impulses repeating with a nonzero period.

This similarity across frequency/time domain also applies to Gaussian functions.

Important Fourier transform properties include:

- **Linearity**:

$$\mathcal{F}\{ax[n] + by[n]\} = a\text{X}[k] + b\text{Y}[k] \tag{9}$$

- **Shift property**: a shift in one domain results in a phase shift in the other domain:

$$\mathcal{F}\{x[n - m]\} = \text{X}[k]e^{\frac{-j2\pi km}{\text{N}}} \qquad \text{time shift} \tag{10}$$

$$\mathcal{F}\left\{x[n]e^{\frac{j2\pi nm}{\text{N}}}\right\} = \text{X}[k - m] \qquad \text{frequency shift}$$

  - **Shift invariant property**: if there is a time shift in the signal, its magnitude does not change, i.e.:

$$|\text{X}[k]_{\text{shift}}| = |\text{X}[k]| \tag{11}$$

- If x is real, then X exhibits complex conjugate symmetry:

$$\mathrm{X}[k] = \mathrm{X}^*[-k]$$

  - X^* rep. the complex conjugate of X.
  - Note that $|\mathrm{X}| = |\mathrm{X}^*|$.

3.4 *Frequency resolution*

Lecture 5
14th February 2023

When displaying FFT results in Matlab, print the log of the results instead of the results directly.

When passing a vector of length N to FFT, we will obtain a vector of the same length as output. How do we interpret the vector?

Let Δf rep. each *frequency step*. We can determine the magnitude of Δf in Hertz with the formula:

$$\Delta f = \frac{f_s}{\mathrm{N}} \tag{12}$$

where:

- f_s rep. the sampling frequency, and
- N rep. the number of points sent as input to FFT.

4 CONVOLUTION

Throughout this topic, we will assume that we are dealing with **linear time-invariant systems**.

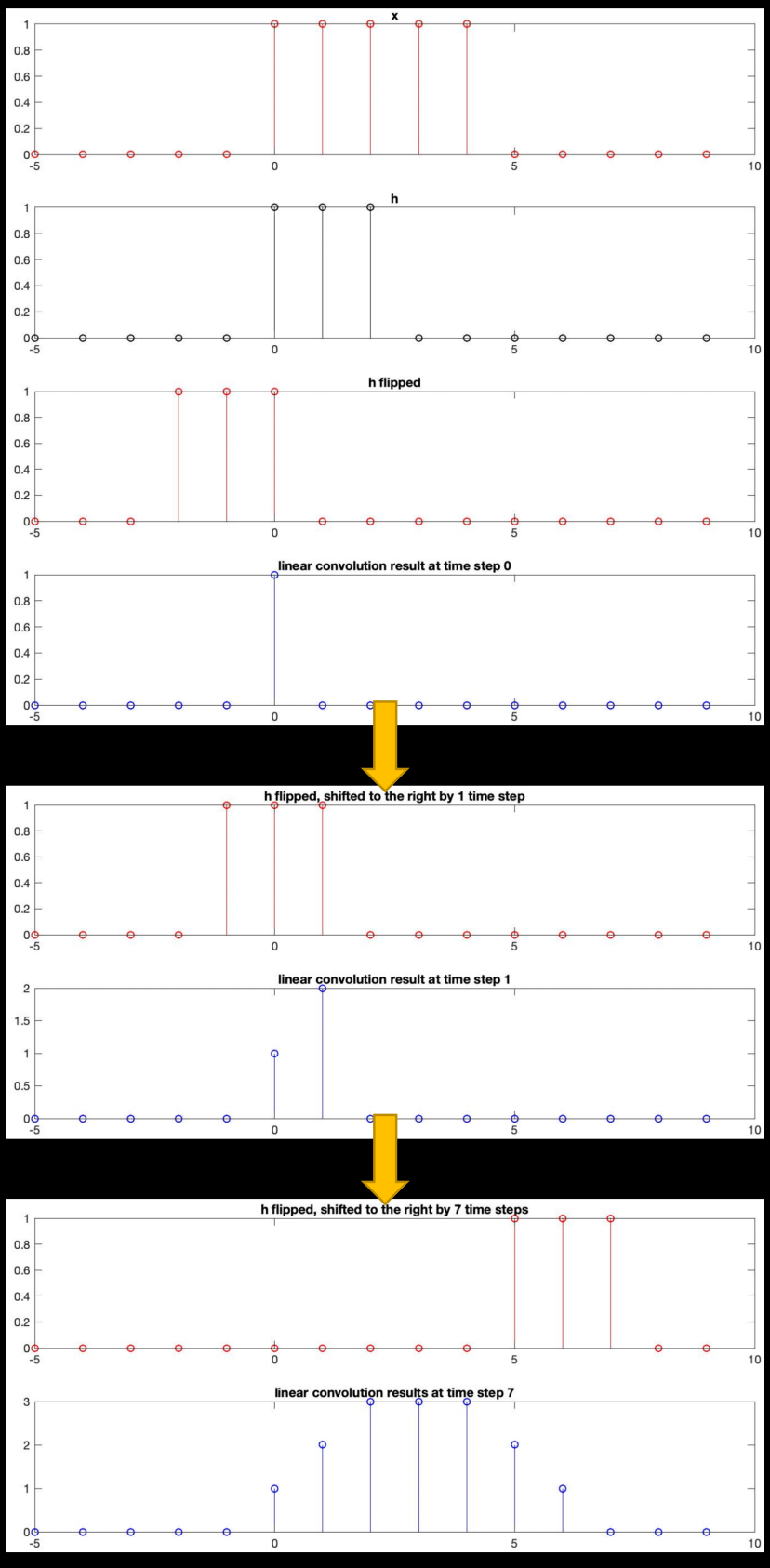
- **Linear** systems: systems whose outputs for a linear combination of inputs are the same as a linear combination of individual responses to those inputs.
- **Time-invariant** systems: systems where the output does not depend on when an input was applied.

In audio, the **kernel** is the **impulse response** (e.g., of a gong being hit).

Convolution is the process of sliding a window (i.e., *kernel*) over a signal, and replacing each point in the signal with a weighted sum of its neighbours.

- The *kernel* defines the weights at each time point.
- Convolution can be used to explain real-world phenomena, e.g., what we hear when we hit a gong multiple times (1D conv), or why a picture is blurred when we take a photograph (2D conv).

In **linear convolution**, the kernel is first flipped, and the values in the kernel gets "shifted out" at each time point.


x
h
h flipped
linear convolution result at time step 0
h flipped, shifted to the right by 1 time step
linear convolution result at time step 1
h flipped, shifted to the right by 7 time steps
linear convolution results at time step 7

Since multiplication in the frequency domain is equivalent to convolution in the time domain, i.e.:

$$g[n] = h[n] * x[n] \Leftrightarrow \mathrm{G}[k] = \mathrm{H}[k]\mathrm{X}[k]$$

where:

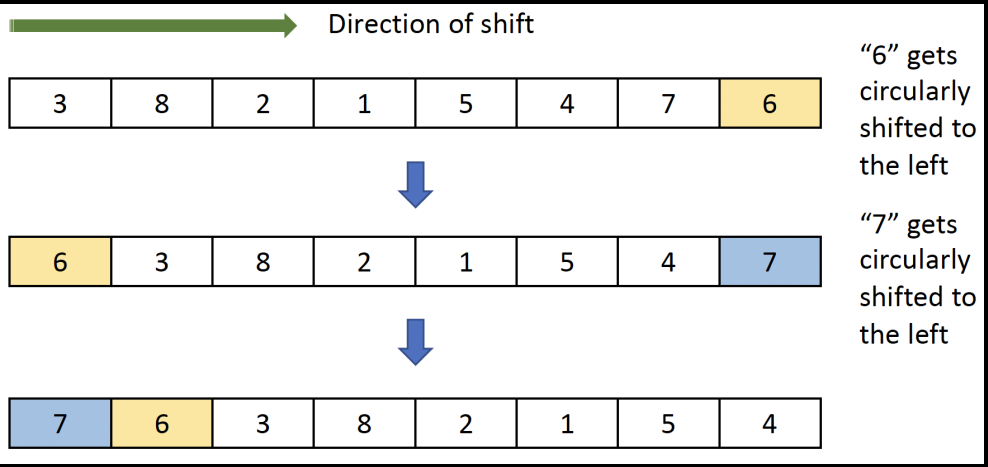
- $*$ rep. convolution (not multiplication).
- g rep. the result of the convolution of x with h.
- G rep. the Fourier transform of g.

Thus, convolution can be done in the frequency domain using FFT. Specifically, in Matlab:

```
g(t) = ifft(fft(h(t)) .* fft(x(t)))
```
(13)

However, note that FFT performs **circular convolution** instead.

- In circular convolution, the values in the kernel gets *circularly shifted* instead of being "shifted out".



Therefore, to implement linear convolution, the input array has to be padded with $m + n - 1$ 0s before being passed into FFT.

5 SAMPLING

Lecture 6
28th February 2023

Due to the duality of Fourier transform, multiplication in the time domain is also equivalent to convolution in the frequency domain.

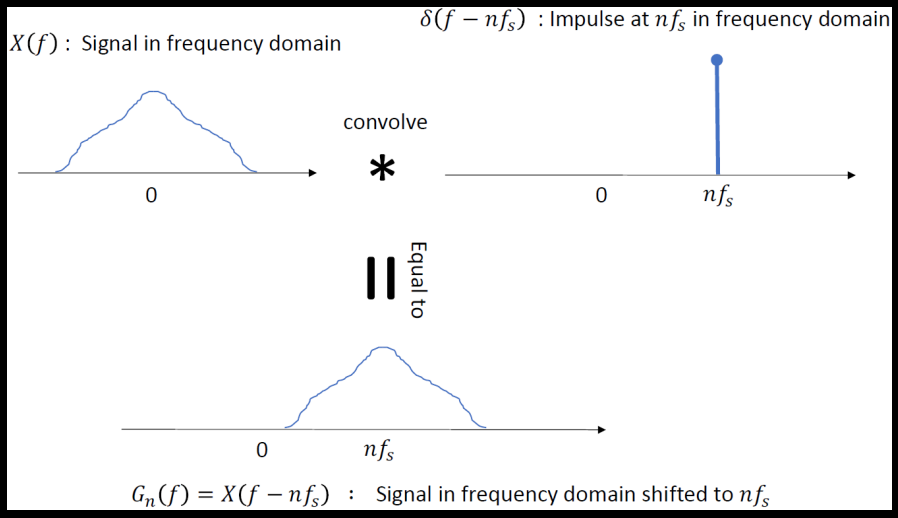
Since the sampling of a signal is represented by (pointwise) multiplication of the signal with an impulse train in the time domain, the same sampling

operation can be represented by the convolution of the signal and impulse train in the frequency domain. Convolving the signal $X(f)$ with an impulse at nf_s gives a replication of the signal at nf_s:

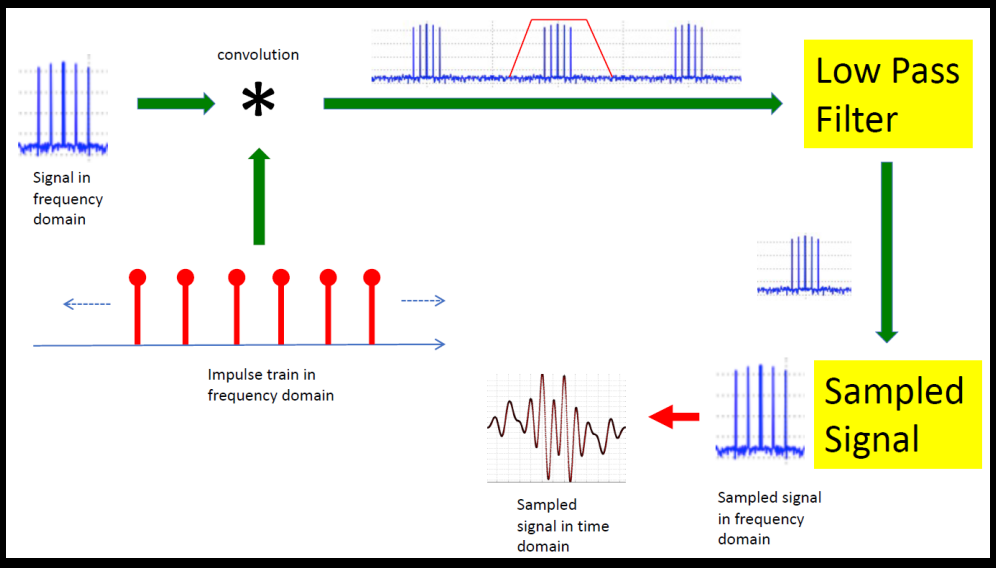
$$G(f) = \sum_{n=-\infty}^{\infty} X(f) * \delta(f - nf_s)$$

$$= \sum_{n=-\infty}^{\infty} \int_{-\infty}^{\infty} X(f - \tau)\delta(\tau - nf_s)d\tau$$

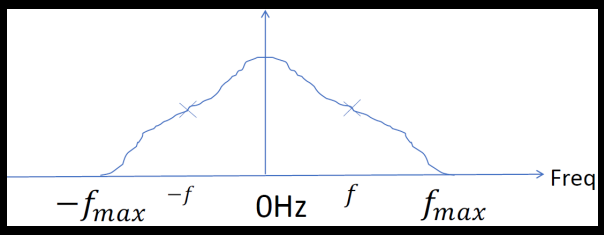
$$= \sum_{n=-\infty}^{\infty} X(f - nf_s)$$



Thus, when convolving a signal with an impulse train in the frequency domain, the result is the original signal centered at each impulse on the impulse train. Subsequently, a **low pass filter** can be applied to filter out higher frequencies.



Note that for real signals, their Fourier transform exhibits conjugate symmetry, i.e., $X(f) = X^*(-f)$. Thus, the magnitude of Fourier transform for real signals is always symmetrical along the y-axis.



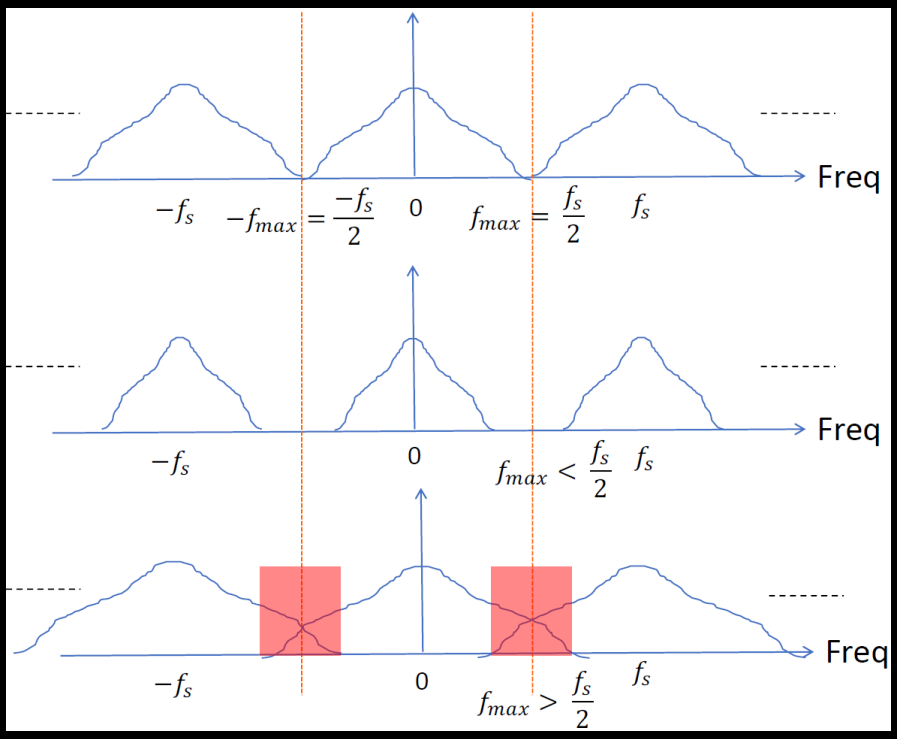
5.1 *Aliasing*

Aliasing is an effect that causes different signals to become indistinguishable when sampled. To prevent aliasing, the highest frequency component of a signal should be less than half of the sampling frequency, i.e.,

$$f_{\text{max}} < \frac{f_s}{2}$$

if $f_{\text{max}} = f_s/2$, there will be no aliasing only if the low pass filter has vertical cutoffs (instead of slanted cutoffs), which is practically impossible to produce.

If frequencies higher than $\frac{f_s}{2}$ are present, aliasing would occur. A *low pass filter* can be used to filter out such frequencies to eliminate aliasing.



Therefore, if the highest frequency component in a signal is f_{max}, we will have to sample the signal with sampling frequency $f_s > 2f_{\text{max}}$, also known as the **Nyquist rate**.

6 2D FOURIER TRANSFORM

Lecture 7
7^{th} March 2023

Note that vertical strips correspond to horizontal frequencies, and vice versa.

FFT can also be applied to images. In images,

- Frequency is expressed as **cycles per meter**.
- Sampling frequency is expressed as **samples/pixels per meter**.
- Frequency resolution is similarly given by

$$\Delta f = \frac{f_s}{N}$$

  where N rep. the number of pixels along the horizontal/vertical axis.

The discrete Fourier transform pair is:

$$G[k,l] = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} g[u,v] e^{-j2\pi\left(\frac{ku}{M} + \frac{lv}{N}\right)} \qquad \text{DFT}$$

$$g[u,v] = \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} G[k,l] e^{j2\pi\left(\frac{ku}{M} + \frac{lv}{N}\right)} \qquad \text{iDFT}$$

For dimensions $D \geq 2$, FFT is performed on one dimension at a time.

6.1 *2D Convolution*

In the image domain, the point spread function is similar to *impulse response* in the audio domain, and it serves as the kernel for the convolution process.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	2	0	0	0	0	0	0	0
0	0	0	0	0	0	2	2	2	0	0	0	0	0	0
0	0	0	0	0	2	2	2	2	2	0	0	0	0	0
0	0	0	0	0	0	2	2	2	0	0	0	0	0	0
0	0	0	0	0	0	0	2	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

0	0	2	0	0	0	0	2	0	0	0	0	2	0	0
0	2	2	2	0	0	2	2	2	0	0	2	2	2	0
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
0	2	2	2	0	0	2	2	2	0	0	2	2	2	0
0	0	2	0	0	0	0	2	0	0	0	0	2	0	0
0	0	2	0	0	0	0	2	0	0	0	0	2	0	0
0	2	2	2	0	0	2	2	2	0	0	2	2	2	0
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
0	2	2	2	0	0	2	2	2	0	0	2	2	2	0
0	0	2	0	0	0	0	2	0	0	0	0	2	0	0
0	0	2	0	0	0	0	2	0	0	0	0	2	0	0
0	2	2	2	0	0	2	2	2	0	0	2	2	2	0
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
0	2	2	2	0	0	2	2	2	0	0	2	2	2	0
0	0	2	0	0	0	0	2	0	0	0	0	2	0	0

Figure 1: Left: linear convolution; Right: circular convolution.

Similar to audio, **linear convolution** in images assumes zero values everywhere beyond the target image, whereas **circular convolution** assumes that the image is repeated in each direction beyond the image.

In Matlab, `fft2` (i.e., 2D FFT) performs **circular convolution** instead of linear convolution.

We can perform 2D (circular) convolution in Matlab in a similar way as before:

```
fftKern = fft2(kern5, 5, 5);        kern5 is a 5x5 kernel
   fftP = fft2(p, 5, 5);            p is the image to perform FFT on
      c = ifft2(fftKern .* fftP);
```

To implement linear convolution, the input matrix has to be padded with 0s (i.e., apply a windowing function) before being passed into FFT, to reduce frequency artifacts due to mismatches at image borders.

Note that to achieve best efficiency in the computation of FFT, it is best to have the dimensions of the matrix to be powers of 2 (i.e., by padding with 0s).

7 DATA COMPRESSION

Lecture 8
14th March 2023

Data compression is a technique to reduce the size of data, to save data storage space and reduce transmission bandwidth. Data compression can be either **lossy** or **lossless**.

7.1 *Lossy compression*

Lossy compression attempts to reduce file sizes while keeping perceivable quality intact (i.e., degradation that is still acceptable to consumers). Specifically, they are based on the following principle:

"Sacrifice the part of the signal which is least critical to human perception."

- In images, this implies that the resultant image will be blurred.
- In audio, this implies that the higher pitches will be omitted in the resultant audio.

Many popular compression techniques are based on **Discrete Cosine Transform (DCT)**.

e.g., cosine

- **Even signals** exhibit symmetry along the y-axis, i.e., $x(t) = x(-t)$.

e.g., sine

  Odd signals exhibit anti-symmetry along the y-axis, i.e., $x(t) = -x(-t)$.

  - Intuitively, any even signal can be fully represented by a linear combination of cosine functions (and odd signals by sine functions).
  - Therefore, by manipulating input signals into even signals, we can represent the input signal with a series of cosine basis functions.

The process of manipulation is not covered in class.

- Since FT of cosine is real and FT of sine is imaginary, we can deduce that FT of even signals is real, and FT of odd signals is imaginary.
  - Since DCT employs cosine waves, the output of DCT is real (whereas the output of DFT is complex in general).
- While DCT itself is not lossy, the filtering away of high frequency components of the DCT results in information loss.
- Type-2 DCT is the most common form used for compression:

$$X[k] = \sum_{n=0}^{N-1} x[n] \cos \frac{\pi(n + \frac{1}{2})k}{N}$$

JPEG is another form of lossy compression, consisting of the following steps:

1. Colour space transformation (i.e., from RGB to YCbCr).
2. 2D DCT.
   (a) The image to be compressed is first divided into 8x8 blocks.
   (b) Each block is transformed by 2D DCT, using a set of basis functions with increasing horizontal and vertical frequencies.
3. Quantization.
   - A **quantization table** is an 8x8 matrix of integers that correspond to the results of the DCT; each entry in this table is an 8-bit integer.
   - To quantize the data, one merely divides the result of the DCT by the quantization value and keeps the integer portion of the result.
   - Typically, higher frequencies are quantized more, and lower frequencies are quantized less.

To compress, we can simply keep the results of the DCT with lower frequencies. In JPEG, different extents of quantization is employed instead.

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Figure 2: Example of a quantization table.

4. Huffman Coding: the DCT coefficients are scanned in a zig zag manner (from low to high frequencies), so that Huffman coding can achieve higher compression rates.

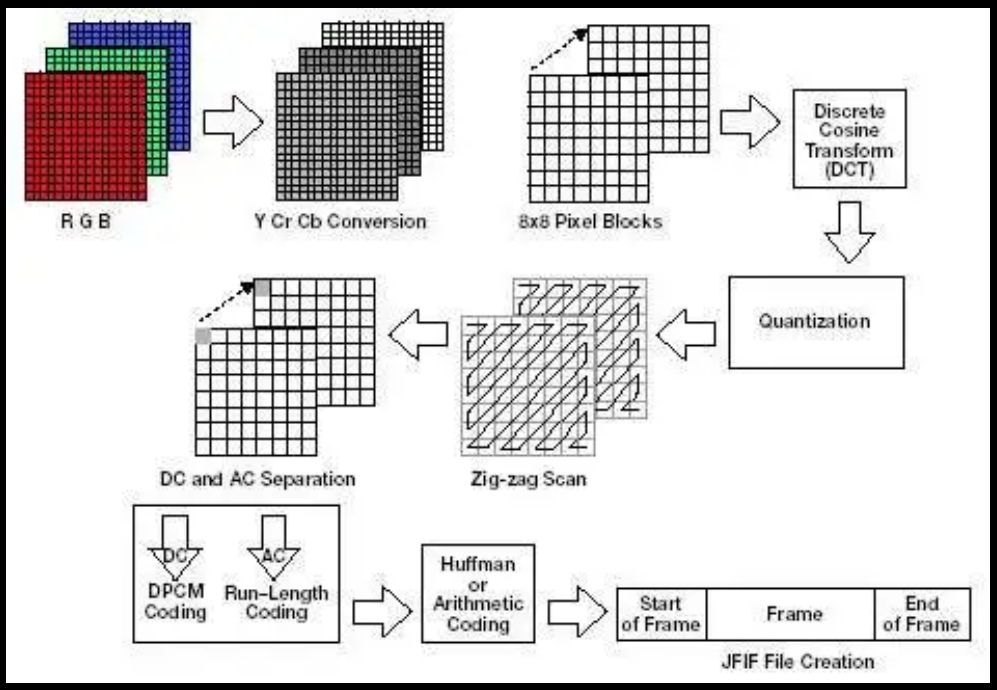


Figure 3: Overview of JPEG compression.

7.2 *Lossless compression*

Some techniques for lossless compression include:

- **Run-Length Encoding (RLE)**: sequences that display redundant data are stored as a single data value representing the repeated block and how many times it appears in the audio/image.
  - E.g., AAAABBCCCCCCCCCDEEEE → 4A2B9C1D4E.
- **Huffman coding**: variable-length strings/codes are used to represent symbols depending on how frequently they appear.
  - The idea is that symbols that are used more frequently should be shorter while symbols that appear more rarely can be longer.
  - A Huffman tree can be built using the following steps:
    1. Create a leaf node for each unique character and build a min heap of all leaf nodes.
    2. Extract two nodes with the minimum frequency from the min heap.

3. Create a new internal node with a frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its left child and the other extracted node as its right child. Add this node to the min heap.
4. Repeat steps 2 and 3 until the heap contains only one node. The remaining node is the root node and the tree is complete.

In MPEG, Huffman coding is applied in combination with RLE to the quantized DCT output coefficients (with the higher frequency components already filtered out). On the other hand, GIF and Unix's `compress` use LZW coding.

- **Lempel-Ziv-Welch (LZW)** compression:

```
1:  function LZWCompress(input)
2:      initialize table with all strings of length 1
3:      s ← first letter of input
4:      while any input left do
5:          read next character c
6:          if s + c in table then
7:              s = s + c
8:          else
9:              output index of s in table
10:             append s + c to table
11:             s = c
12:     output index of s in table
```

- LZW decompression:

```
1:  function LZWDecompress(input)
2:      initialize table with all strings of length 1
3:      read first codeword p and output character corresponding to it
4:      while any codeword left do
5:          read next codeword c
6:          if c not in table then
7:              append table[p] + table[p][0] to table
8:              output table[p] + table[p][0]
9:          else
10:             append table[p] + table[c][0] to table
11:             output table[c]
12:         p = c
```

8 CORRELATION

Lecture 9
21st March 2023

Correlation is used to detect similarities (i.e., pattern recognition) between two pieces of audio/images. The underlying idea is that the dot product of two

vectors is largest when their waveforms match exactly (after normalization).

Mathematically, correlation is very similar to convolution, with the only distinction highlighted below:

- **Convolution**:
  - Continuous time:

$$g(t) = x(t) * h(t) = \int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau$$

$$G(f) = X(f)H(f)$$

  - Discrete time:

$$g[n] = x[n] * h[n] = \sum_{\tau=0}^{M-1} x[\tau]h[n-\tau]$$

$$G[k] = X[k]H[k]$$

- **Correlation**:
  - Continuous time:

$$g(t) = x(t) \otimes h(t) = \int_{-\infty}^{\infty} x(\tau)h(t+\tau)d\tau$$

$$G(f) = X^*(f)H(f)$$

  - Discrete time:

$$g[n] = x[n] \otimes h[n] = \sum_{\tau=0}^{M-1} x[\tau]h[n+\tau]$$

$$G[k] = X^*[k]H[k]$$

    where $X^*(f)$ is the complex conjugate of $X(f)$.

A MATLAB CHEATSHEET

- `diary on`: creates file named 'diary' in directory, which stores output of session.
- `whos`: shows all variables currently in memory.
- `M(:,1)`: returns the first column of matrix M.

A.1 *Tutorial 1*

- `eye(n)`: creates an $n \times n$ identity matrix.
- `rand(m, n)`: generates a random matrix of size $m \times n$.
- `inv(M)`: returns the inverse of matrix M.
- `conj(c)`: returns the complex conjugate of the complex number c.
- `[v,d] = eig(M)`: get the *eigenvector* (v) and *eigenvalue* (d) of matrix M.

Figure tools:

- `figure`: creates a new figure window using default property values.
- `plot(Y)`: plots Y against an implicit set of x-coordinates.
- `subplot(m, n, p)`: divides the current figure into an m-by-n grid and creates axes in the position specified by p.
- `stem(Y)`: plots the data sequence, Y, as stems that extend from a baseline along the x-axis.
- `axis on/off`: `off` turns off the display of the axes background
- `grid on/off`: `off` removes all grid lines from the current axes or chart.
- `title(text)`: adds the specified title to the current axes or standalone visualization.
- `print(filename, formattype)`: saves the current figure to a file using the specified file format.

Displaying text:

- `disp(X)`; displays the value of variable X without printing the variable name.
- `fprintf(formatSpec, A1...)`: formats data and displays the results on the screen.

Loops and branching:

- `for i = initVal : step : endVal`: increment i by `step` on each iteration.

- `if expression`

  ```
  statements
  elseif expression
  statements
  else
  statements
  end
  ```

A.2 *Tutorial 2*

Useful for computation of fundamental frequency.

- gcd: returns the greatest common divisors of the elements of A and B.
- `max(A)` and `max(x1, x2)`.
- `zeros(sz1,...,szN)`: returns an `sz1`-by-...-by-`szN` array of zeros.
- `ones(sz1,...,szN)`: returns an `sz1`-by-...-by-`szN` array of ones.
- `length(X)`: returns the length of the largest array dimension in X.
- `size(A)`: returns a row vector whose elements are the lengths of the corresponding dimensions of A.

A.3 *Tutorial 3*

Note: `fft` performs circular convolution.

- `fft(X)`: computes the discrete Fourier transform (DFT) of X using a fast Fourier transform (FFT) algorithm.
- `ifft(Y)`: computes the inverse discrete Fourier transform of Y using a fast Fourier transform algorithm.

A.4 *Tutorial 4*

- `audiorecorder`: object to record audio data from an input device.
- `audioplayer`: object to play audio data.
- `audiowrite(filename, y, Fs)`: writes a matrix of audio data y, with sample rate `Fs` to the file `filename`.
- `[y, Fs] = audioread(filename)`: reads data from the file `filename`, and returns sampled data y, and a sample rate for that data `Fs`.

- `playblocking(playerObj)`: plays the audio associated with audioplayer object `playerObj` from beginning to end.
- `recordblocking(recorderObj, length)`: records audio from an input device for the number of seconds specified by `length`.
- `getaudiodata(recorder)`: returns recorded audio data associated with `audiorecorder` object recorder in a double array y.
- `strcat(s1,...,sN)`: horizontally concatenates the input arguments.
- `num2str(A)`: converts a numeric array into a character array that represents the numbers.
- `nextpow2(A)`: returns the exponents for the smallest powers of 2 that satisfy $2^p \geq |A|$ for each element in A.
- `find(X)`: returns a vector containing the linear indices of each nonzero element in array X.

A.5 *Tutorial 5*

- `circshift(A, K)`: circularly shifts the elements in array A by K positions.
- `conv(u, v)`: returns the convolution of vectors u and v.
- `cconv(a, b, n)`: circularly convolves vectors a and b, n is the length of the resulting vector.

A.6 *Tutorial 6*

- `imread(filename)`: reads the image from the file specified by `filename`.
- `imshow(i)`: displays the image i in a figure.
- `imagesc(C)`: displays the data in array C as an image that uses the full range of colors in the colormap.
- `rgb2gray(RGB)`: converts the truecolor image RGB to a grayscale image.
- `fft2(X)`: returns the two-dimensional Fourier transform of a matrix; equivalent to computing `fft(fft(X).').'`.
- `ifft2(Y)`: returns the two-dimensional discrete inverse Fourier transform of a matrix.
- `fir1(n, Wn)`: uses a Hamming window to design an nth-order lowpass, bandpass, or multiband FIR filter with linear phase. The filter type depends on the number of elements of `Wn`.

* * *