University of California, Los Angeles
UCLA Department of Electrical And Computer Engineering

EE211-Image Processing

Spring 2019

Lecture 1

Date: April 2, 2019 **Scribe: Lucas He**

1 Basics of Signals

1.1 Signal

The first question that students may ask is that *what is signal?*

- **Definition:** Signal is a function that varies in time, space or other dimensions. It usually contains the information about the behavior or attributes of certain phenomenon.
- **Examples:** The temperature of **Boston Logan International Airport** for each month in 2018 is a discrete signal.

1.2 Analog vs Digital

The difference between **analog signal** and **digital signal** is that analog signal represents continuous values; at any given time it represents a real number within continuous range of values, while the latter represents a sequence of discrete values; at a given time it can only take one of a finite number of values.

1.3 Conversion Between Analog and Digital Signals

- Analog to Digital Converter(ADC): A device that converts analog signal to digital by **sampling** and **quantization.**
- Digital to Analog Converter(DAC): A device that converts an abstract finite-precision number into a physical quantity (e.g., a voltage or a pressure).

1.4 The Scope of This Course

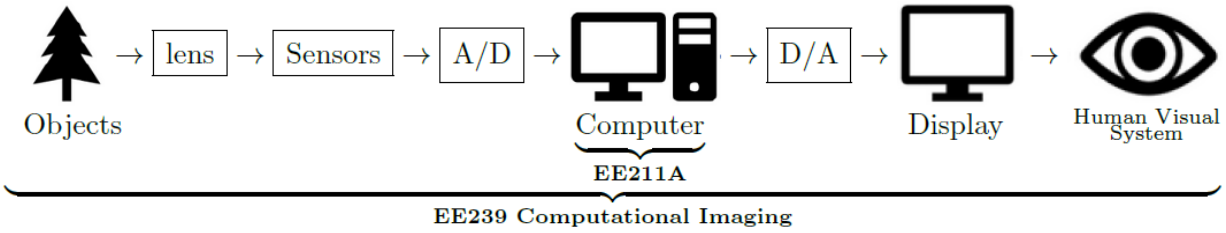


Figure 1: Scope of the course

1.5 Separable Function

2-D Step Function

$$u(n_1, n_2) = \begin{cases} 1 & n_1 \geq 0, n_2 \geq 0 \\ 0 & o.w. \end{cases}$$

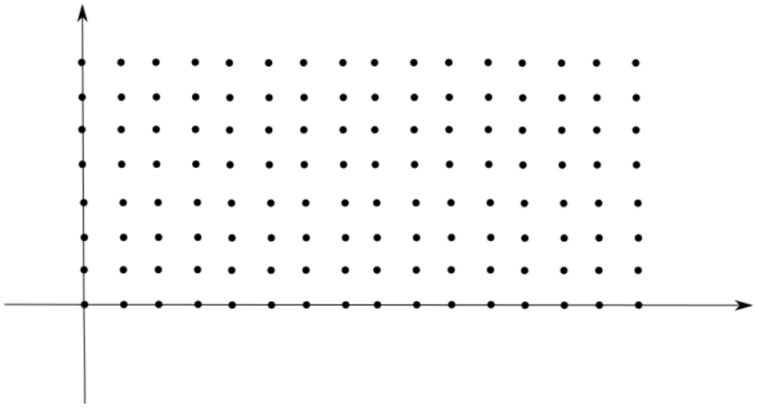


Figure 2: 2D Step Function

Goal: Write 2D signal as separable 1D signals to reduce computational complexity (N^2 vs order 2N operations).

2-D Impulse Function

$$\delta(n_1, n_2) = \delta(n_1) \times \delta(n_2)$$

Figure 3: 2D Step Function

Check Your Understanding: Is the 2D step function separable?

Yes. The 2D step function can be separated as outer product of two 1D step functions.

2 Complex exponentials

$$x(n_1, n_2) = e^{j\omega_1 n_1} \times e^{j\omega_2 n_2},$$
$$e^{j\omega_1 n_1} e^{j\omega_2 n_2} \rightarrow LSIsystem \rightarrow Ae^{j\omega_1 n_1} e^{j\omega_2 n_2} e^{j\phi},$$
$$x(n_1, n_2) = e^{j\omega_1 n_1} e^{j\omega_2 n_2} = cos(\omega_1 n_1 + \omega_2 n_2) + jsin(\omega_1 n_1 + \omega_2 n_2).$$

Check Your Understanding: Is the complex exponential periodic in the frequency domain?

Yes.

$$e^{j(\omega_1+2\pi)n_1} e^{j(\omega_2+2\pi)n_2} = e^{j\omega_1 n_1} e^{j\omega_2 n_2}.$$

Check Your Understanding: Prove whether the complex exponential is periodic in space.

$$e^{j\omega_1(n_1+2\pi)} e^{j\omega_2(n_2+2\pi)} = e^{j(\omega_1 n_1)} e^{j(\omega_2 n_2)}.$$

3 2D System

3.1 Definition

A 2D system can be defined as follows:

$$x(n_1, n_2) \rightarrow \tau(.) \rightarrow y(n_1, n_2) = \tau[x(n_1, n_2)],$$

where $y(n_1, n_2)$ is the output of the system, $x(n_1, n_2)$ is the input of the system, and $\tau(.)$ is the 2D system.

3.2 Some Examples of 2D System

- **Example 1:** Linear Polarity Operator (Inverse the color of the image)

  $$y(n_1, n_2) = \tau[x(n_1, x_2)] = 255 - x(n_1, n_2).$$

- **Example 2:** Image Enhancement (Increase contrast of an image)

  $$y(n_1, n_2) = \tau[x(n_1, x_2)] = g(n_1, n_2),$$

  where $g(.)$ is called a tone mapping function, which can map a pixel value to another pixel value.

- **Example 3:** Smooth Operator (Make the pixel in the image smooth)

  $$y(n_1, n_2) = \tau[x(n_1, x_2)] = Average[Neighbor(x(n_1, n_2))].$$

3.3 Properties of 2D Systems

For a 2D system, we used several properties as follows to describe the system:

- Stability $\rightarrow$ *e.g.(bounded - input, bounded - output)BIBO System*
- Memoryless
- Causality
- **Linearity***
- **Space Invariance***

In this class we are focusing the last two properties: **Linearity** and **Space Invariance**. A system which meets these two properties is called **Linear Shift-Invariant (LSI)** system.

4 LSI System

4.1 Linearity

Considering a system, $\tau[x(n_1, n_2)]$ and two input signal $x_1(n_1, n_2)$ and $x_2(n_1, n_2)$. If the system follows the following requirement:

$$\tau[\alpha x_1(n_1, n_2) + \beta x_2(n_1, n_2)] = \alpha\tau[x_1(n_1, n_2)] + \beta\tau[x_2(n_1, n_2)].$$

Check Your Understanding: Can we prove linear system have homogeneity?
Considering we have the input signal $x_1(n_1, n_2)$. Thus, we have:

$$\alpha x_1(n_1, n_2) + \alpha x_1(n_1, n_2) = 0.$$

Then we can expand the $\tau[0]$ as follows:

$$\tau[0] = \tau[\alpha x_1(n_1, n_2) + \alpha x_1(n_1, n_2)] = \alpha\tau[x_1(n_1, n_2)] - \alpha\tau[x_1(n_1, n_2)].$$

Thus, we can prove that every linear system have homogeneity.

Check Your Understanding: Is image polarity a linear system?
No. Image polarity is $\tau[x(n_1, n_2)] = 255 - x(n_1, n_2)$, and $\tau[0] = 255$. Thus, we can know that image polarity do not have homogeneity. Since we know that every linear system have homogeneity, we can conclude that image polarity is not a linear system.

4.2 Space Invariance

Considering a system, $\tau[x(n_1, n_2)]$ so that the output will be $y(n_1, n_2) = \tau[x(n_1, n_2)]$. If the system follows the following requirement:

$$y(n_1 - k_1, n_2 - k_2) = \tau[x(n_1 - k_1, n_2 - k_2)].$$

Then the system is called **space invariance**

Check Your Understanding: Is image polarity a space invariance system?
Yes.

$$y(n_1, n_2) = \tau[x(n_1, n_2)] = 255 - x(n_1, n_2).$$

Thus, we can write the following:

$$\tau[x(n_1 - k_1, n_2 - k_2)] = 255 - x(n_1 - k_1, n_2 - k_2) = y(n_1 - k_1, n_2 - k_2).$$

Thus, the image polarity is a space invariance system.

4.3 Magic of LSI System

The LSI system has a lot of great characteristic. One of the best characteristic is that we can use a delta function impulse to get the impulse response.

$$\delta(n_1, n_2) \xrightarrow{\text{LSI}} h(n_1, n_2) \equiv ImpulseResponse.$$

Thus, if we want to estimate the impulse response of an camera, we can just use the camera to take a picture of dot light, and the resulting image will be impulse response of the camera.

University of California, Los Angeles
UCLA Department of Electrical And Computer Engineering

EE211-Image Processing

Spring 2019

Lecture 2

Date: April 4, 2019 **Scribe: VMG Summer Team**

1 Review

1.1 Complex Exponential

The expression for the complex exponential is:

$$x(n_1, n_2) = e^{j\omega_1 n_1} e^{j\omega_2 n_2} = cos(\omega_1 n_1 + \omega_2 n_2) + jsin(\omega_1 n_1 + \omega_2 n_2).$$

Any given complex exponential passed through a LSI system can be expressed in the form:

$$y(n_1, n_2) = T\{x(n_1, n_2)\} = Ax(n_1, n_2)e^{j\phi}.$$

Question: Is the complex exponential periodic in frequency?
Answer: Yes
Proof: $e^{j(\omega_1+2\pi)n_1} e^{j(\omega_2+2\pi)n_2} = e^{j\omega_1 n_1} e^{j\omega_2 n_2}$.

2 2D Systems

$$y(n_1, n_2) = T\{x(n_1, n_2)\},$$

where T denotes a transform. Examples of a transformation are

- Image Polarity: $y(n_1, n_2) = 255 - x(n_1, n_2)$.
- Image enhancement
- Smoothing $y(n_1, n_2) = Average(Neighborhood(x(n_1, n_2)))$.

3 Properties of 2D Systems

- Stability
- Memory
- Causality
- Linearity
- Spacial Invariance

1. Linearity: Linear systems comply with

$$T\{\alpha_1 x_1(n_1, n_2) + \alpha_2 x_2(n_1, n_2)\} = \alpha_1 T\{x_1(n_1, n_2)\} + \alpha_2 T\{x_2(n_1, n_2)\},$$

where a_1 and a_2 are constants.
Homogeneity: $T\{\alpha x(n_1, n_2)\} = \alpha T\{x(n_1, n_2)\}$.

Question: Can we prove linear systems have homogeneity?
Answer: Yes. Substitute $a_2 = 0$ into the definition of linear systems (Eq. 1).
Question: Is image polarity a linear system?
Answer: No.

2. Space Invariance: Space invariant systems comply with:

$$T\{x(n_1 - k_1, n_2 - k_2)\} = y(n_1 - k_1, n_2 - k_2),$$

where k_1 and k_2 are constants.
Question: Is image polarity a space invariant system?
Answer: Yes.

4 Magic of LSI Systems

4.1 LSI System - Linear Space Invariance System

We define the impulse response of a LSI system as:

$$h(n_1, n_2) = T\{\delta(n_1, n_2)\}.$$

Then for an input $x(n_1, n_2)$ and output $y(n_1, n_2)$ of LSI system, the following expression holds:

$$\begin{aligned} y(n_1, n_2) &= T\{(x(n_1, n_2)\}, \\ &= x(n_1, n_2) * *h(n_1, n_2), \\ &= \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} x(k_1, k_2) h(n_1 - k_1, n_2 - k_2), \\ &= h(n_1, n_2) * *x(n_1, n_2). \end{aligned}$$

where ** denotes 2D convolution. Note that convolution is **commutative**.
Proof.
Step 1: $x(n_1, n_2)$ can be decomposed into a series of scaled and shifted $\delta(n_1, n_2)$,

$$x(n_1, n_2) = \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} x(k_1, k_2) \delta(n_1 - k_1, n_2 - k_2).$$

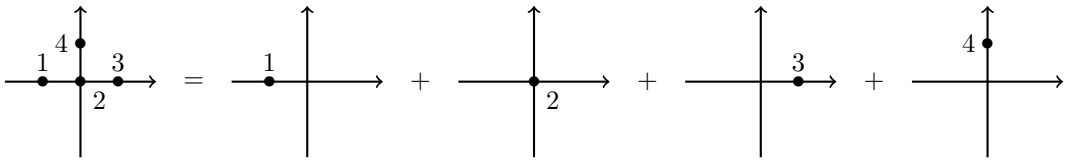


Figure 1: A graphical example of $s(n_1, n_2)$, and its decomposition.

$x(n_1, n_2)$ in Fig.1 can be decomposed as $x(n_1, n_2) = x(0,0)\delta(n_1, n_2) + x(-1,0)\delta(n_1 + 1, n_2) + x(1,0)\delta(n_1 - 1, n_2) + x(0,1)\delta(n_1, n_2 - 1)$.
Step 2: Using linear (Eq. 1) and space invariant (Eq. 2) properties

$$\begin{aligned} y(n_1, n_2) &= T\{(x(n_1, n_2)\}, \\ &= T\left\{ \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} x(k_1, k_2)\delta(n_1 - k_1, n_2 - k_2) \right\}, \\ &= \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} x(k_1, k_2) T\{\delta(n_1 - k_1, n_2 - k_2)\}, \\ &= \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} x(k_1, k_2) h(n_1 - k_1, n_2 - k_2), \\ &= h(n_1, n_2) * * x(n_1, n_2). \end{aligned}$$

4.2 Example 2D convolution by picture

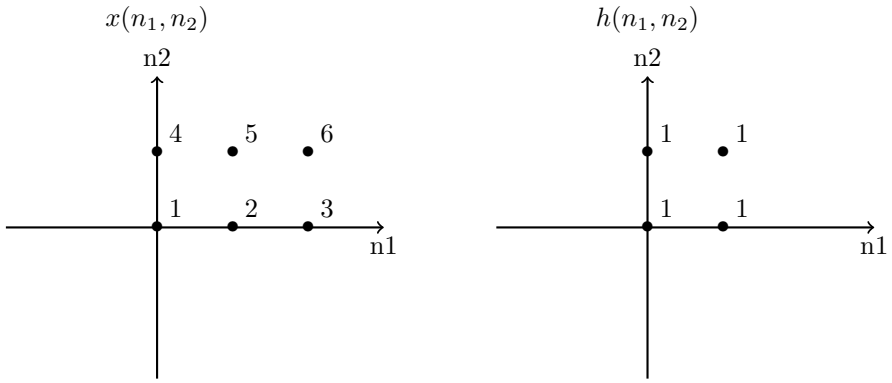


Figure 2: Example of $x(n_1, n_2)$ and $h(n_1, n_2)$.

Question: Compute $y(n_1, n_2) = x(n_1, n_2) * * h(n_1, n_2)$, as shown in Fig. 2.
Answer: Using the definition of convolution. (Eq. 3)
Step 1: Flip $h(n_1, n_2)$ and around (0,0) to get $h(-k_1, -k_2)$

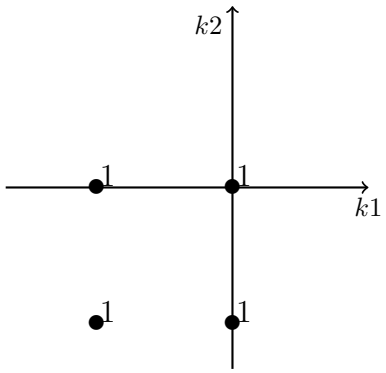


Figure 3: $h(-k_1, -k_2)$

Step 2: Shift $h(-k_1, -k_2)$ by (n_1, n_2) to get $h(n_1 - k_1, n_2 - k_2)$. Calculate $y(n_1, n_2) = h(n_1 - k_1, n_2 - k_2) * x(k_1, k_2)$,

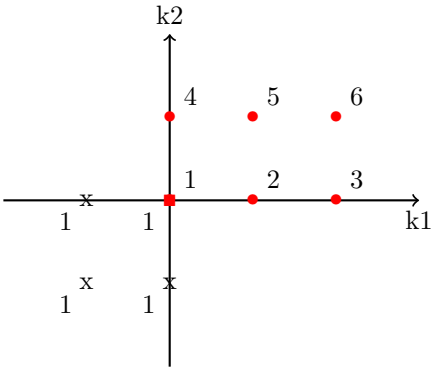


Figure 4: $h(n_1 - k_1, n_2 - k_2)$ is shown with black cross. $x(k_1, k_2)$ is shown with red dot. There is zero shift, i.e. $n_1 = n_2 = 0$, and y(0,0) = 1.

Step 3: Repeat Step 2. Shift with all possible values of (n_1, n_2) and compute $y(n_1, n_2)$.

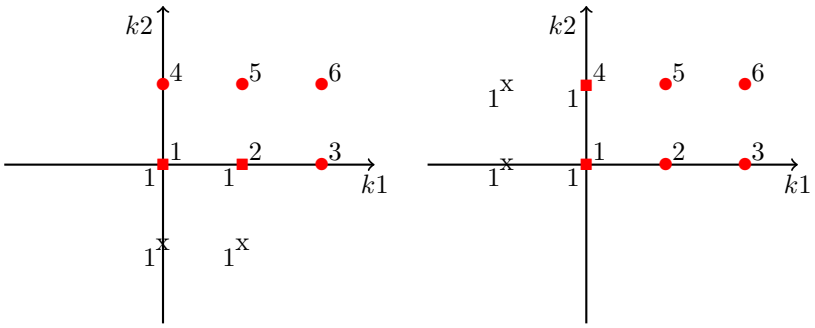


Figure 5: $h(n_1 - k_1, n_2 - k_2)$ is shown with black cross. $x(k_1, k_2)$ is shown with red dot. On the left, $n_1 = 1, n_2 = 0$, and $y(1, 0) = 3$. On the right, $n_1 = 0, n_2 = 1$, and $y(0, 1) = 5$

Result

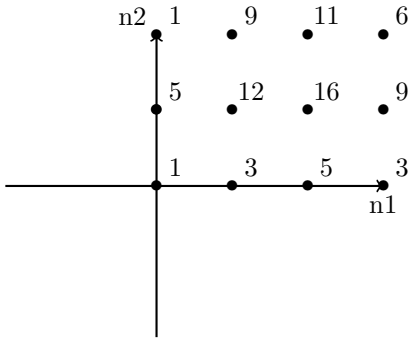


Figure 6: $y(n_1, n_2)$

Note that the support of $y(n_1, n_2)$ is different from that of $x(n_1, n_2)$ or $h(n_1, n_2)$.
In general:

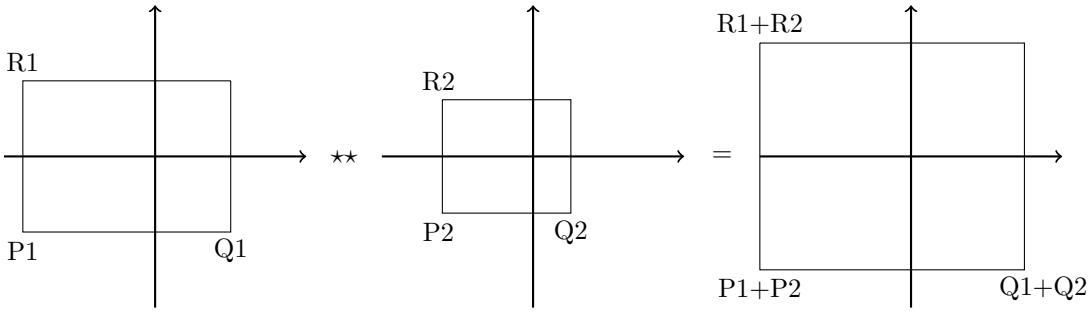


Figure 7: The relationship between the support of the result of the convolution and the support of two inputs.

University of California, Los Angeles
UCLA Department of Electrical And Computer Engineering

EE211-Image Processing

Spring 2019

Lecture 3

Date: April 9, 2019 **Scribe: VMG Summer Team**

1 Filters

1.1 Gaussian Filter Expression

The Gaussian filter is a 2D operator which is often used for smoothing the image and remove the noise. The expression can be written as:

$$I^{'}(x) \propto \sum_{x_i \in \mathcal{N}} I(x_i) g(\|x_i - x\|_2).$$

In this equation, $\mathcal{N}$ stands for the neighborhood of the pixel x_i, $I(x_i)$ is the original pixel in the original image, $g(...)$ is the Gaussian operator, which can be expressed as:

$$g(...) = exp\left(-\frac{\|x_i - x\|_2}{\sigma}\right).$$

In the expression we can see that the Gaussian filter gives more weight to the pixels near the neighborhood of the target pixel and less weight to the pixels far away. And increasing the filter size will make the filter more diffuse. Thus we can use the information in the pixels near by to create a new pixel in the new image.

1.2 Bilateral Filter Expression

In order to denoise the image while keeping the edges, we will use bilateral filter. The expression of bilateral filter is as below:

$$I^{'}(x) = \sum_{x_i \in \mathcal{N}} I(x_i) r(\left\|I(x_i) - I(x)\right\|_2) g(\|x_i - x\|_2),$$

where $g(...)$ is the domain kernel and $r(...)$ is the range kernel, which can be expressed as:

$$r(...) = exp\left(-\frac{\left\|I(x_i) - I(x)\right\|_2}{\sigma}\right).$$

We can see that if the larger difference between $I(x)$ and $I(x_i)$, the lower weight it will contribute to the new pixel, which will help us keep the edge.

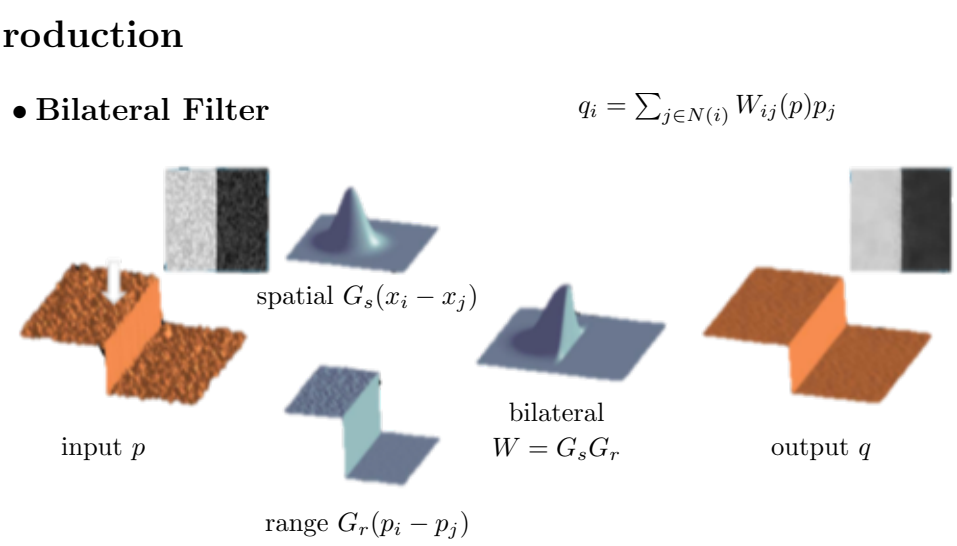


Figure 1: A visualization slide from EECV10 "Guided Image Filtering" Kaiming He et al.

2 Eigenfunctions of LSI Systems

If we have input:

$$x(n_1, n_2) = e^{j(\omega_1, n_1 + \omega_2, n_2)}.$$

The impulse response of the LSI system is $h(n_1, n_2)$, then the output of the LSI system will be:

$$y(n_1, n_2) = e^{j(\omega_1, n_1 + \omega_2, n_2)} \circledast h(n_1, n_2),$$

where $h(n_1, n_2)$ is the point spread function of the LSI system. Then if we take a look at the result of the convolution:

$$\begin{aligned} y(n_1, n_2) &= \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} e^{j\omega_1(n_1-k_1)} e^{j\omega_2(n_2-k_2)} h(n_1, n_2), \\ &= e^{j\omega_1 n_1} e^{j\omega_2 n_2} \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} h(k_1, h_2) e^{-j\omega_1 k_1} e^{-j\omega_2 k_2}. \end{aligned}$$

Where $H(\omega_1, \omega_2)$, as indicated in the highlighted part, is the frequency response of the LSI system at ω_1, ω_2. $x(n_1, n_2)$ can be called the eigenfunction of the LSI system.

3 2D Fourier Transform

Fourier Transform

$$X(\omega_1, \omega_2) = \sum_{n_1=\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} x(n_1, n_2) e^{j\omega^{-j\omega_2 n_2}}.$$

Inverse Fourier Transform

$$x(n_1, n_2) = \frac{1}{4\pi^2} \int \int_{-\pi}^{\pi} X(\omega_1, \omega_2) e^{j\omega_1 n_1} e^{j\omega_2 n_2} d\omega_1 d\omega_2.$$

Proof.

$$\begin{aligned}
x(n_1, n_2) &= \frac{1}{4\pi^2} \int \int_{-\pi}^{\pi} X(\omega_1, \omega_2) e^{j\omega_1 n_1} e^{j\omega_2 n_2} d\omega_1 d\omega_2, \\
&= \frac{1}{4\pi^2} \int \int_{-\pi}^{\pi} \sum_{n_1'=\infty}^{\infty} \sum_{n_2'=-\infty}^{\infty} x(n_1', n_2') e^{-j\omega_1 n_1} e^{-j\omega_2 n_2} e^{j\omega_1 n_1} e^{j\omega_2 n_2} d\omega_1 d\omega_2, \\
&= \frac{1}{4\pi^2} \sum_{n_1'=\infty}^{\infty} \sum_{n_2'=-\infty}^{\infty} x(n_1', n_2') \int \int_{-\pi}^{\pi} e^{-j\omega_1 (n_1' - n_1)} e^{-j\omega_2 (n_2' - n_2)} d\omega_1 d\omega_2, \\
&= \frac{1}{4\pi^2} x(n_1, n_2) 4\pi^2 \delta(n_1' - n_1, n_2' - n_2), \\
&= x(n_1, n_2).
\end{aligned}$$

3.1 Properties of 2D Fourier Transform

Periodic

$$X(\omega_1, \omega_2) = X(\omega_1 + 2\pi, \omega_2 + 2\pi).$$

Shifting

$$x(n_1 - m_1, n_2 - m_2) \longleftrightarrow e^{-j\omega_1 m_1} e^{-j\omega_2 m_2} X(\omega_1, \omega_2).$$

This indicates that shifting image does not lead to the shift frequency.

Modulation

$$X(\omega_1 - \theta_1, \omega_2 - \theta_2) \longleftrightarrow x(n_1, n_2) e^{j\theta_1 n_1 + j\theta_2 n_2}.$$

Hermitian Property

When the image is real, we have

$$|X(\omega_1, \omega_2)| = |X(-\omega_1, -\omega_2)|, \;\; \angle X(\omega_1, \omega_2) = -\angle X(-\omega_1, -\omega_2).$$

3.2 Parseval's

$$\sum_{n_1} \sum_{n_2} |x(n_1, n_2)|^2 = \frac{1}{4\pi^2} \int \int_{-\pi}^{\pi} |X(\omega_1, \omega_2)|^2 d\omega_1 d\omega_2.$$

4 Example

Given an impulse response of an LSI system, where $h(0,0) = \frac{1}{3}$, $h(0,1) = h(1,0) = h(0,-1) = h(-1,0) = \frac{1}{6}$. Find $H(\omega_1, \omega_2)$ in frequency domain. (Hint: $e^{ja} + e{-}ja = 2\cos(a)$)
Answer:

$$\begin{aligned} H(\omega_1,\omega_2) &= h(0,0) + h(-1,0)e^{j\omega_1} + h(1,0)e^{-j\omega_1} + h(0,-1)e^{j\omega_2} + h(0,1)e^{-j\omega_2}, \\ &= \frac{1}{3} + \frac{1}{6}e^{j\omega_1} + \frac{1}{6}e^{-j\omega_1} + \frac{1}{6}e^{j\omega_2} + \frac{1}{6}e^{-j\omega_2}, \\ &= \frac{1}{3}(1 + \cos\omega_1 + \cos\omega_2). \end{aligned}$$

From the above equation, we have $H(0,0) = 1$ and $H(-\pi, \frac{\pi}{2}) = 0$. Hence, it can be seen as a Low Pass Filter.

University of California, Los Angeles
UCLA Department of Electrical And Computer Engineering

EE211-Image Processing

Spring 2019

Lecture 4

Date: April 11, 2019 **Scribe: VMG Summer Team**

1 Quick Review

Question: Suppose a kernel $h(n_1, n_2) = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$, and $H(\omega_1, \omega_2)$ is its Fourier transformations in frequency domain. Is $H(\omega_1, \omega_2)$ real?
Answer: According to Fourier transformation:

$$H(\omega_1, \omega_2) = \sum_{n_1=0}^{\infty} \sum_{n_2=0}^{\infty} h(n_1, n_2) e^{-j\omega_1 n_1} e^{-j\omega_2 n_2}.$$

We apply $h(n_1, n_2)$ into this formula, so that:

$$\begin{aligned} H(\omega_1, \omega_2) &= 9 - [e^{-j\omega_1} + e^{-j\omega_2} + e^{j\omega_1} + e^{j\omega_2} + e^{-j(\omega_1+\omega_2)} + e^{j(\omega_1+\omega_2)} + e^{-j(\omega_1-\omega_2)} + e^{j(\omega_1-\omega_2)}], \\ &= 9 - 2cos(\omega_1) - 2cos(\omega_2) - 2cos[(\omega_1 + \omega_2)] - 2cos[(\omega_1 - \omega_2)]. \end{aligned}$$

As a result, $H(\omega_1, \omega_2)$ is real. **In fact, if the signal input is even and symmetric, the frequency response of it is bounded to be real.** From the result, we could also see this kernel $h(n_1, n_2)$ is a high-pass filter.

2 Magic of LSI

2.1 Convolution theorem

Theorem 0.1. *Under suitable conditions the Fourier transform of a convolution of two signals is the pointwise product of their Fourier transforms.*

$$\begin{array}{ccccc} x(n_1, n_2) & \rightarrow & \boxed{h(n_1, n_2)} & \rightarrow & y(n_1, n_2) = x(n_1, n_2) * *h(n_1, n_2) \\ \updownarrow & & \updownarrow & & \updownarrow \\ X(\omega_1, \omega_2) & \rightarrow & \boxed{H(\omega_1, \omega_2)} & \rightarrow & Y(\omega_1, \omega_2) = X(w\omega_1, \omega_2) H(\omega_1, \omega_2) \end{array}$$

Proof. We suppose $T[\cdot]$ is a LSI system,

$$
\begin{aligned}
y(n_1, n_2) &= T[x(n_1, n_2)], \\
&= T\left[\frac{1}{4\pi^2}\iint_{-\pi}^{\pi} X(\omega_1, \omega_2) e^{j\omega_1 n_1} e^{j\omega_2 n_2} d\omega_1 d\omega_2\right], \\
&= \frac{1}{4\pi_2}\iint_{-\pi}^{\pi} X(\omega_1, \omega_2) T[e^{j\omega_1 n_1} e^{j\omega_2, n_2}] d\omega_1 d\omega_2, \\
&= \frac{1}{4\pi^2}\iint_{-\pi}^{\pi} X(\omega_1, \omega_2) H(\omega_1, \omega_2) e^{j\omega_1 n_1} e^{j\omega_2 n_2} d\omega_1 d\omega_2, \\
y(n_1, n_2) &= \frac{1}{4\pi^2}\iint_{-\pi}^{\pi} Y(\omega_1, \omega_2) e^{j\omega_1 n_1} e^{j\omega_2 n_2} d\omega_1 d\omega_2.
\end{aligned}
$$

As a result, $Y(\omega_1, \omega_2) = X(\omega_1, \omega_2)H(\omega_1, \omega_2)$, which shows the convolution theorem.

Note that we are using the property of LSI system from (4) to (5), the property of eigens of LSI from (6) to (7) (*See more explanation for this from previous lectures*) and IFT definition of $y(n_1, n_2)$ for (7). So that, $y(n_1, n_2) \leftrightarrow Y(\omega_1, \omega_2) = X(\omega_1, \omega_2)H(\omega_1, \omega_2)$.

2.2 Convolutions in Digital Image Processing

Question: If $x = \begin{bmatrix}1 & 8 & 3 & 2 & 5\end{bmatrix}$, $h = \begin{bmatrix}3 & 5 & 2\end{bmatrix}$, what is $y = conv(x, h)$?

Answer:

Solution 1: By using the formula of convolution:

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

We first reverse $h = \begin{bmatrix}2 & 5 & 3\end{bmatrix}$ to $h' = \begin{bmatrix}3 & 5 & 2\end{bmatrix}$.

We move the kernel as a sliding window to make multiplication to x pointwisely.

$y = conv(x, h) = \begin{bmatrix}3 & 29 & 51 & 37 & 29 & 10\end{bmatrix}$.

Solution 2: The convolution operation can be constructed as a matrix multiplication, where one of the inputs is converted into a Toeplitz matrix. For example, the convolution of y = h**x can be expressed by:

$$
y^T = \begin{bmatrix}h1 & h2 & h3 & ... & hm\end{bmatrix}
\begin{bmatrix}
x_1 & x_2 & x_3 & ... & x_n & 0 & 0 & 0 & ... & 0 \\
0 & x_1 & x_2 & x_3 & ... & x_n & 0 & 0 & ... & 0 \\
0 & 0 & x_1 & x_2 & x_3 & ... & x_n & 0 & ... & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & ... & \vdots & \vdots & ... & 0 \\
0 & ... & 0 & 0 & x_1 & ... & x_{n-2} & x_{n-1} & x_n & \vdots \\
0 & ... & 0 & 0 & 0 & x_1 & ... & x_{n-2} & x_{n-1} & x_n
\end{bmatrix}.
$$

In our example,

$$
y^T = \begin{bmatrix}3 & 5 & 2\end{bmatrix}
\begin{bmatrix}
1 & 8 & 3 & 2 & 5 & 0 & 0 \\
0 & 1 & 8 & 3 & 2 & 5 & 0 \\
0 & 0 & 1 & 8 & 3 & 2 & 5
\end{bmatrix}
= \begin{bmatrix}3 & 29 & 51 & 37 & 31 & 29 & 10\end{bmatrix}.
$$

2.3 Deconvolutions in digital image processing

Solution 1: Naive solution:

$$\begin{aligned} F(y) &= F(x) * F(h), \\ F(x) &= \tfrac{F(y)}{F(h)}, \\ x &= F^{-1}\left(\tfrac{F(y)}{F(h)} - \tfrac{F(e)}{F(h)}\right). \end{aligned}$$

However, when we take noise into consideration, we will find that the error of deconvolutions by naive way is large:

$$\begin{aligned} y &= x * h + e, \\ F(y) &= F(x) * F(h) + F(e), \\ F(x) &= \tfrac{F(y)}{F(h)} - \tfrac{F(e)}{F(h)}, \\ x &= F^{-1}\left(\tfrac{F(y)}{F(h)} - \tfrac{F(e)}{F(h)}\right), \\ x &= x^* - \tfrac{F(e)}{F(h)}. \end{aligned}$$

Because of the high frequency of noise, the term $\frac{F(e)}{F(h)}$ is going to be large. To solve this problem, we have Wiener Deconvolution.

Solution 2: Wiener deconvolution is an application of the Wiener filter to the noise problems inherent in deconvolution. It works in the frequency domain, attempting to minimize the impact of deconvoluted noise at frequencies which have a poor signal-to-noise ratio.

$$x_{est} = F^{-1}\left(\frac{|F(h)|^2}{|F(h)|^2 + \frac{1}{SNR(\omega)}} * \frac{F(y)}{F(h)}\right),$$

$$SNR(\omega) = \frac{mean\ \ signal\ \ at\ \ \omega}{noise\ \ std\ \ at\ \ \omega} (SNR \propto \frac{1}{\omega^2}).$$

We can see that Wiener deconvolution restrains the side effect from high frequencies noise, which can also be written as an objection function $min_x \|y - h * x\|^2 + \|\Delta x\|^2$ and we will talk about it in the following chapter.

University of California, Los Angeles
UCLA Department of Electrical And Computer Engineering

EE211-Image Processing

Spring 2019

Lecture 5

Date: April 16, 2019 **Scribe: VMG Summer Team**

1 Review: Deconvolution

1.1 Noiseless Systems

$$x(n_1, n_2) \to h(n_1, n_2) \to y(n_1, n_2).$$

By Convolution Theorem,

$$Y(\omega_1, \omega_2) = H(\omega_1, \omega_2) \cdot X(\omega_1, \omega_2),$$
$$\therefore X(\omega_1, \omega_2) = \frac{Y(\omega_1, \omega_2)}{H(\omega_1, \omega_2)},$$
$$\therefore \hat{X}(\omega_1, \omega_2) = \mathcal{F}^{-1}\left(\frac{Y(\omega_1, \omega_2)}{H(\omega_1, \omega_2)}\right).$$

The above equation is how we solve the inverse problem of de-convolution.

1.2 Systems with Noise

Such systems have an additional noise component represented by $e(n_1, n_2)$.

$$x(n_1, n_2) \to h(n_1, n_2) \to y(n_1, n_2) \leftarrow e(n_1, n_2).$$

By Convolution Theorem,

$$Y(\omega_1, \omega_2) = H(\omega_1, \omega_2) \cdot X(\omega_1, \omega_2) + \mathcal{F}\Big[e(n_1, n_2)\Big],$$
$$\therefore X(\omega_1, \omega_2) = \frac{Y(\omega_1, \omega_2)}{H(\omega_1, \omega_2)} - \frac{\mathcal{F}\Big[e(n_1, n_2)\Big]}{H(\omega_1, \omega_2)},$$
$$\hat{X}_{noise}(\omega_1, \omega_2) = \hat{X}_{noiseless}(\omega_1, \omega_2) - \mathcal{F}^{-1}\left(\frac{\mathcal{F}\Big[e(n_1, n_2)\Big]}{H(\omega_1, \omega_2)}\right).$$

The second term represents the error due to noise.
The problem with the above approach is that for low-pass filters, the term $H(\omega_1, \omega_2)$ tends to 0 as ω_1 and ω_2 increase. This phenomenon is also called 'De-convolution Blow-Up'. Thus, we abstract the problem differently as follows:

$$\hat{X}(\omega_1, \omega_2) = \frac{1}{H(\omega_1, \omega_2)}\Big[Y(\omega_1, \omega_2) - E(\omega_1, \omega_2)\Big].$$

The term $\frac{1}{H(\omega_1, \omega_2)}$ is the Inverse Filter $\approx G(\omega_1, \omega_2)$

1.3 Solutions

The different solutions for this problem are:

1. Naive or Unregulated or Pseudo-inverse De-convolution
2. Weiner Deconvolution

$$G(\omega_1, \omega_2) = \frac{1}{H(\omega_1, \omega_2)} \left[\frac{|H(\omega_1, \omega_2)|^2}{|H(\omega_1, \omega_2)|^2 + \frac{1}{SNR(\omega_1, \omega_2)}} \right],$$

where,

$$SNR = \frac{\text{Signal Spectral Density } @(\omega_1, \omega_2)}{\text{Noise Spectral Density } @(\omega_1, \omega_2)}.$$

Weiner De-convolution has the benefit that it acts as a damping factor for noisy frequencies, thereby reducing the contribution of noise. However, it also blurs the image slightly as it removes the high-frequency components, thus the high-frequency details are lost.

2 Discrete World Of Linear Algebra

$$y[n_1, n_2] \rightarrow \overrightarrow{y}.$$

Organising the matrix as a row/column vector,

$$\overrightarrow{y} = \Big[y(1,1), y(2,1), ..., y(n,1), y(1,2), y(2,2), ..., y(N,2), ..., y(N,N) \Big]^T.$$

$$\overrightarrow{y} = H\overrightarrow{x} + \overrightarrow{e}.$$

2.1 An example in 1D:

$$y = H \cdot x.$$

Let $h = \begin{bmatrix} a & b & c \end{bmatrix}^T$ and $x = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}^T$ Then, we form H as a Circular Matrix instead of the Toeplitz matrix as:

$$H = \begin{bmatrix} a & b & c \\ b & c & a \\ c & a & b \end{bmatrix}$$

3 Preview: Regularization/Priors/Optimization

3.1 Case I: $\vec{y} = H\vec{x}$

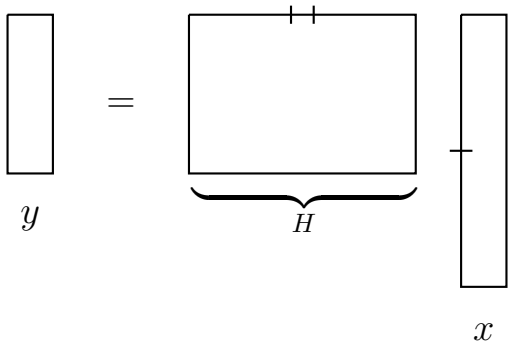


Figure 1: Matrix representation

If the size of $\vec{x}$ is bigger than H, then, we pad H.
Suppose we have y and H, then we can estimate $\hat{x}$ as below.

$$\hat{x} = argmin_x \|y - Hx\|_2^2 .$$

Check Your Understanding: What would be the closed form solution for $\hat{x}$?

Hints:

$$\nabla_x[x^T A] = A,$$
$$\nabla_x[x^T c] = c,$$
$$\nabla_x[x^T Ax] = 2Ax \text{ (if A is symmetric)},$$

$$\nabla_x \|y - Hx\|_2^2 = 0,$$
$$\nabla_x (y - Hx)^T (y - Hx) = 0,$$
$$\nabla_x [y^T y - x^T H^T y - yHx + x^T H^t Hx] = 0,$$
$$-H^T y - H^T y + 2H^T H\hat{x} = 0,$$
$$-H^T y + H^T H\hat{x} = 0,$$

$$\therefore \hat{x} = (H^T H)^{-1}(H^T y).$$

3.2 Case II: Tikhonov Solution

$$\hat{x} = argmin_x \left(\|y - Hx\|_2^2 + \lambda \|\rho x\|_2^2 \right).$$

Here, in this rigid regression form the first term represent the model while the second term represent the prior. And, λ is a hyper-parameter.

Check Your Understanding: Closed form solution for Tikhonov problem.

$$\begin{aligned}
\nabla_x(\|y - Hx\|_2^2 + \lambda\|px\|_2^2) &= 0, \\
\nabla_x[y^T y - x^T H^T y - yHx + x^T H^T Hx + \lambda x^T x] &= 0, \\
\nabla_x[y^T y - x^T H^T y - yHx + x^T (H^T H + 1\lambda)x] &= 0, \\
-2H^T y + 2(H^T H + 1\lambda)\hat{x} &= 0,
\end{aligned}$$

$$\therefore \hat{x} = (H^T H + 1\lambda)^{-1}(H^T y).$$

The Wiener filter can be implemented using Tikhonov solution. By adding Gamma function, we can represent it as a Wiener deconvolution.

3.3 Case III: Sparsity

$$\hat{x} = argmin_x(\|y - Hx\|_2^2 + \lambda\|\bigtriangledown x\|_1).$$

The second term behaves as an edge detection operator. We are trying to minimize the second term and thereby remove the edges. This produces a low-frequency solution.

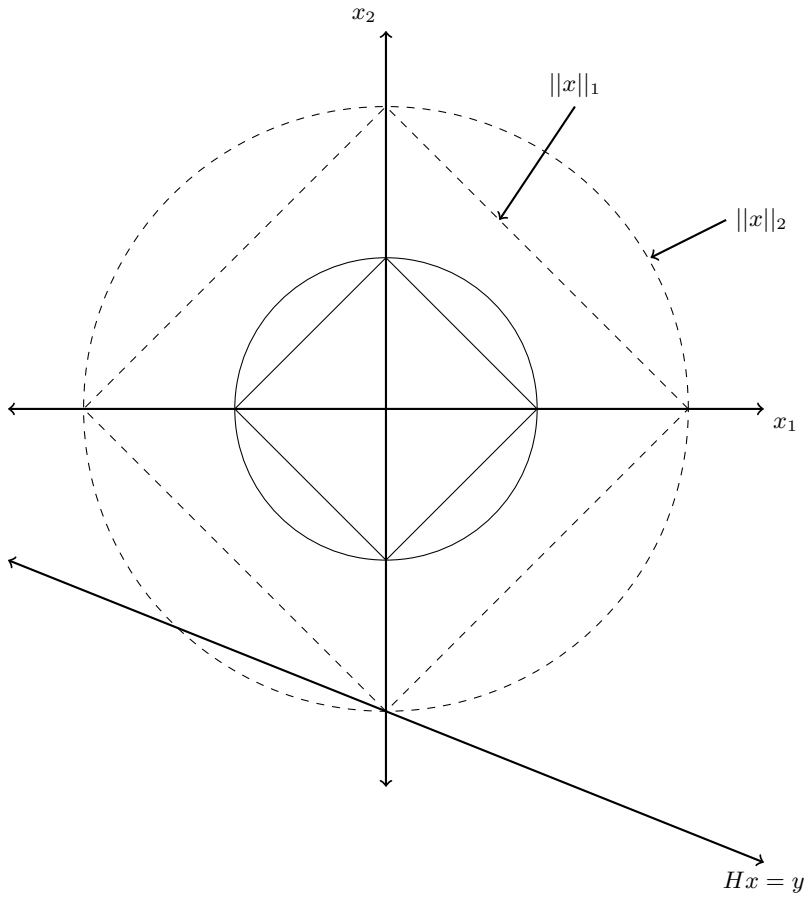


Figure 2: L1 and L2 norm

L2-norm ball

$$\|x\|_2 = \sqrt{x_1^2 + x_2^2 + ... + x_n^2}.$$

L1-norm diamond

$$\|x\|_1 = |x_1| + |x_2| + ... + |x_n|.$$

$$\|x\|_2 \Leftarrow 1 \Rightarrow \text{L2 Ball}$$
$$|x| \Leftarrow 1 \Rightarrow \text{L1 Diamond}$$

For $Hx = y$ without regularization, we could pick any point in the line. But when we apply regularization, we would get an unique answer. The L2 regularization grows the L2 ball and finds a point where it intersects with the L1 diamond. Since, the diamond is pointy, it generally intersects near the axes. Hence, this causes the solution to be sparse in nature.

University of California, Los Angeles
UCLA Department of Electrical And Computer Engineering

EE211-Image Processing

Spring 2019

Lecture 6

Date: April 18, 2019 **Scribe: Brian Chap**

1 Wiener Deconvolution

Check your understanding: Weiner deconvolution can be defined as

1. Removing noisy high frequency data from the inverse filter process
2. Finding a low-frequency solution model that fits the data

So can the Weiner filter (formula below) be written as a convex optimization problem?

$$\hat{x}_{est} = F^{-1}\left(\left[\frac{|F(n)|^2}{|F(n)|^2 + \frac{1}{SNR(\omega)}}\right] \cdot \frac{F(y)}{F(h)}\right).$$

Yes, this can be solved by gradient descent to get a closed form solution. The convex optimization problem (L2 regularization) is written below:

$$\hat{x} = argmin \|y - Hx\|^2 + \lambda \|\nabla x\|^2.$$

When we add a regularization term ∇x, the high frequency terms (edges) will be penalized, which promotes a low frequency solution. In other words, the edge operation is high frequency. Minimizing the norm of the edge operation says that the edges should be small.

Check your understanding: Why does the L1-norm not give us a low frequency solution?

Wiener deconvolution is naive deconvolution with the regularization term added to it. Gradient regularization is still not completely accurate. Sometimes, blurry images have frequencies that are neither too high nor too low and therefore they are not removed. This leads to the output of gradient deconvolution being dark and still removes noise. This still remains better than naive deconvolution which amplifies noise. Ringing artifacts remain in the image after regularization is done, because regularization reduces larger changes but does not remove/affect the smaller changes.

1.1 Deconvolution Comparison

When the input is noisy, naive deconvolution is bad. Gradient regularization does better, but it still has stipple effects (dots) because the dots fit in the equation. The blurry image itself has dots which get retained because they are of medium frequency technically.

How to prove gradient regularization is actually removing noise and naive deconvolution is amplifying noise? ⇒ We feed an image which is purely noise and look at the difference. Naive deconvolution will output noise, where as gradient regularization will clean the noise.

How do we measure the PSF? ⇒ Take a picture of a dot. Note that phone camera benchmarks (e.g. DxOMark) evaluate the PSF (in addition to other things).

Blind Deconvolution Consider the equation

$$x * h = y,$$

where x is the input, h is the point spread function (PSF) and y is the output. How do we solve for x, if we do not have the point spread function? (It is worth noting that generating the image of the black hole used concepts such as blind convolution.) We understand that the shake of the camera (which causes the PSF) is unknown in nature as the movement of the camera could be in any direction. The output i3mage y is the result of this shaky motion of the camera. So there are many solutions, but which one do we choose?

One solution is to use priors and regularization. For example, we know that images have a certain "natural" look. Images that look like a natural image have a lot of low frequency content and little high frequency content. We also know a few things about the PSF. For example, we know that the PSF must be sparse because the motion of the camera won't cover every pixel. Also, we know that the PSF must be positive because we are using a camera to take pictures. With this knowledge, we can arrive at a good solution.

1.2 Image Compression

There are 2 types of image compression: lossless (e.g. PNG) and lossy (e.g. JPEG). We will explore JPEG here. The general steps of JPEG compression are:

1. Cut the image into 8 by 8 blocks. Subtract the average of the block so that the mean of the block is zero.
2. Perform the discrete cosine transform (DCT). (The DCT is similar to a Fourier Transform.)
3. Use a hard-coded quantiztion matrix to quantize the DCT coefficients. This is essentially a compression or sparse representation.

To expand on step 2, we had an orchestra analogy in class. Say that we had a trumpet playing at a constant Hertz. Let's also add a monotone DC component. The signal might look like this:

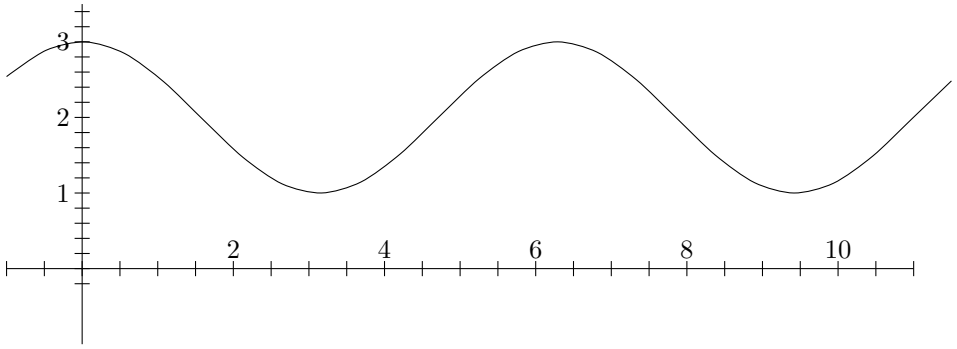


Figure 1: Orchestra Signal

It's possible to discretize this signal and represent it as a matrix like this:

$$\begin{bmatrix} 3 & 2 & 1 & 2 & 3 & 2 & 1 & 2 & 3 & ... \end{bmatrix}$$

However, a more sparse way is to use a DCT matrix composed of signals of various frequencies. For example, the DCT matrix could be the below figure:

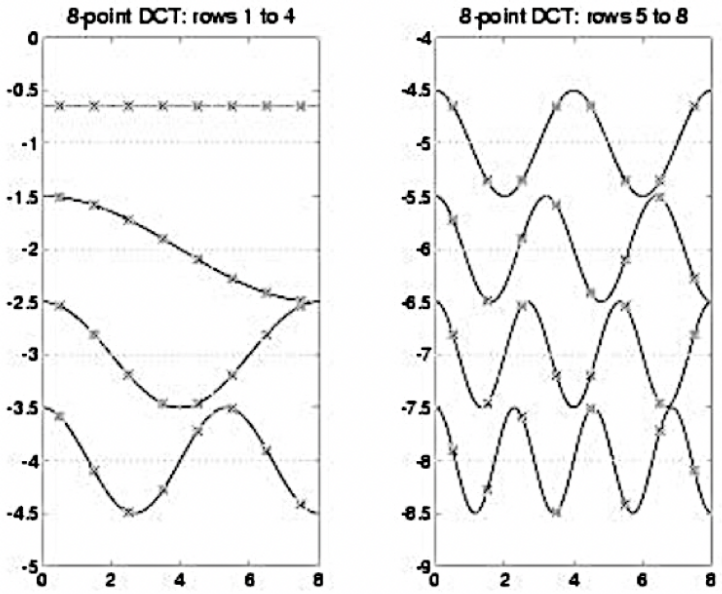
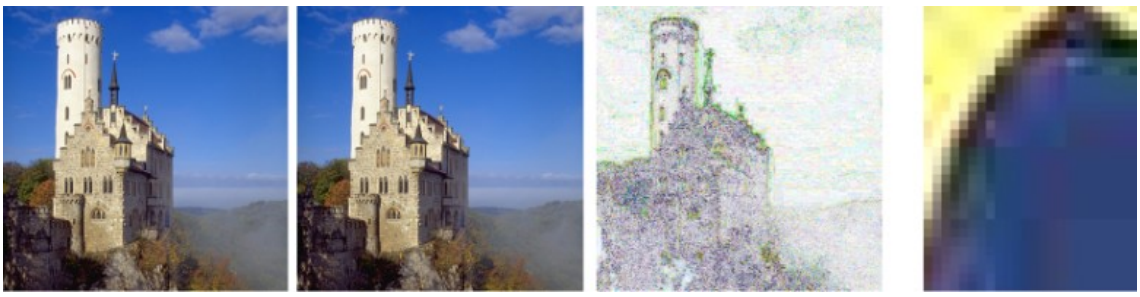


Figure 2: DCT Matrix Composed of Various Frequencies

Since our signal is composed of a DC signal and a single frequency, we could write our representation as

$$\begin{bmatrix} a & 0 & 0 & \dots & 0 & b & 0 & \dots & 0 & 0 \end{bmatrix}$$

where a and b are constants. Thus, this matrix representation is much more sparse.



Why is this blocky?

Figure 3: JPEG Compression

In the above example, the difference between the original image and the compressed image are the high frequency components. We observe blockness in the edges as we zoom into the image. These are the artifacts of JPEG compression. This is because the DCT is a real transform (not complex), and therefore the phase information is lost during compression.

JPEG tends to introduce a few kinds of distortions:

1. General loss of sharpness and oscillations around high-contrast edges. These are due to approximating sharp transitions with smooth functions (cosines); you see them as small "dots" or "halos" around the edges and they are particularly easy to see in images of text or hand-drawings. Consider using the floating-point discrete cosine transform to fix this. (It may increase the precision of the transform, but file saving will take longer.)

2. Blocking structure. Since the image is processed in separate 8x8 blocks (or bugger in case of chromatic downsampling), block edges become visible at high compression ratios.

3. Loss of color detail. Depending on saving parameters, the program may aggressively "downsample" (reduce the resolution of) chromatic channels.

University of California, Los Angeles
UCLA Department of Electrical And Computer Engineering

EE211-Image Processing

Spring 2019

Lecture 7

Date: April 23, 2019 **Scribe: Akshat Bansal**

1 Basis

Definition: N linearly independent vectors spanning $\mathbb{R}^n$

Example: Standard Basis

Consider the vectors e_1 and e_2 depicted below. These form basis in $\mathbb{R}^2$ if...

$$\exists \ \alpha, \beta \ \mid \ \alpha e_1 + \beta e_2 = y \forall y \in \mathbb{R}^2.$$

Or as a system of equations...

$$y = Dx = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

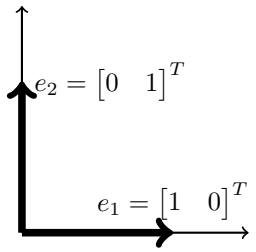
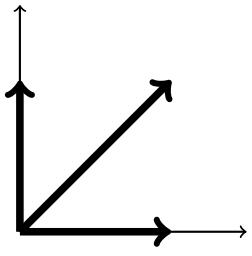


Figure 1:

In the system of equations above, D is a *dictionary matrix*, whose columns for a subspace of *range(D)*. Any $y \in \mathbb{R}^n$ can be represented by some $D \in \mathbb{R}^{m \times n}$ if D spans $\mathbb{R}^n$. In this context, sparsity occurs for x vectors with zero components, i.e. where not all basis vectors are required.

Overcomplete Dictionary: spans $\mathbb{R}^n$, but not a basis because all the columns aren't linearly independent. In other words, matrix D that spans $\mathbb{R}^n$ is not full rank.

Example:



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

Figure 2:

Dictionary Learning: Finding a basis where the representation of some measurements are sparse.

Check Your Understanding: Consider canonical basis for $\mathbb{R}^2$ discussed above...

- Can we *sparsely* represent $\hat{y} = \begin{bmatrix} 7.312 & 0 \end{bmatrix}^T$? **Yes.**
- What about $\hat{y} = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$? **No...but yes with another basis, i.e.** $D = \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$

1.1 Linear Algebra Review: Linear Inverse Problems, $b = Ax$

- **Fully Determined System** $\Rightarrow$ M = N , x* = $A^{-1}b$

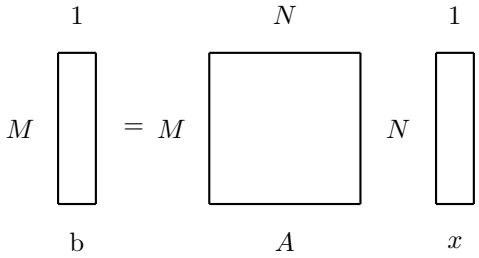


Figure 3:

- **Over Determined System** $\Rightarrow$ M > N , x* = $\underset{x}{argmin} \|Ax - b\|_2^2 = (A^T A)^{-1} A^T b$

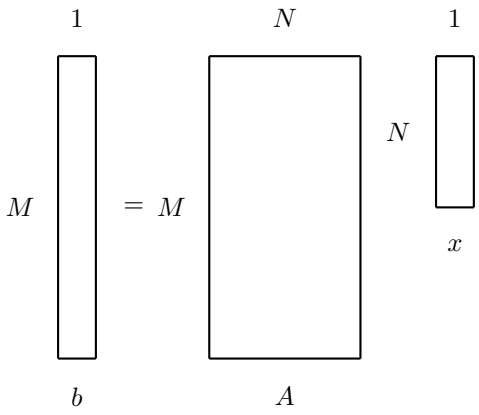


Figure 4:

- **Under Determined System** $\Rightarrow$ M << N , ∞ solutions if $b \in range(A)$, else none (if infinite solutions, we can use regularization to find the correct one!)

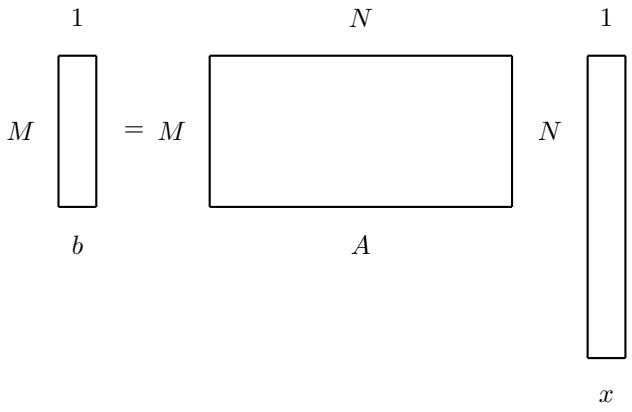


Figure 5:

1.2 Linear Algebra Review: p-norms

- Definition: $\|x\|_p = (\sum x^p)^{(1/p)}$
- Isometry Property: $\|\alpha x\| p = \alpha \|x\|_p$

1.3 Optimization Review: l_2 Regularization

$$\min_x J(x) \mid Ax = b,$$
$$J(x) = \|x\|_2 .$$

Visualization: l_2 ball in $\mathbb{R}^2 \rightarrow \|x\| \frac{2}{2} = c$, find minimum c for intersection...

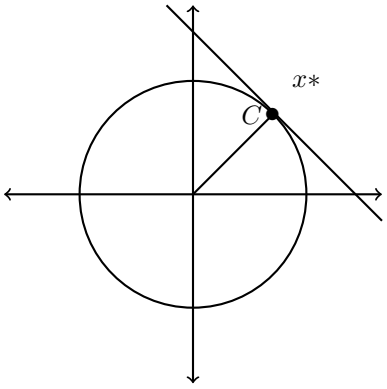


Figure 6:

Check Your Understanding: Taking the l_2 Norm...

$$x = \begin{bmatrix} 3 & 4 & 0 \end{bmatrix}^T \rightarrow \|x\|_2 = ? \quad \|x\|_2 = 5.$$

1.4 l_2 Regularization as a Linear Program

Problem: $\min \|x\|_2 \ s.t. \ Ax = b$ How can we solve? One method, look at slack and write as Lagrange problem...

$$\begin{aligned} &\underset{x,\lambda}{argmin}\ \mathcal{L}(x,\lambda) = \|x\|_2 + \lambda^T(Ax-b), \\ &s.t.\ kkt\ 1\ \rightarrow \frac{\delta}{\delta x}\mathcal{L}(x,\lambda) = 0, \\ &s.t.\ kkt\ 2\ \rightarrow \frac{\delta}{\delta x}\mathcal{L}(x,\lambda) = 0. \end{aligned}$$

(i.e. at optimal point, neither x nor λ would perturb optimum)

Solution: (b/c convex, gradients can be written explicitly)

$$\begin{aligned} &\nabla_x \mathcal{L} = x + A^T\lambda = 0, \\ &\rightarrow x = -A^T\lambda, \\ &\nabla_\lambda \mathcal{L} = (Ax-b) = 0, \\ &\rightarrow A(-A^T\lambda) - b = 0, \\ &\Rightarrow \lambda = -(AA^T)^{-1}b \quad \text{(i.e. how much 'slack' on our constraint)}, \\ &\Rightarrow x^* = A^T(AA^T)^{-1}b \quad \text{(i.e. point where constraint intersects with } l_2 \text{ ball, shown above)}. \end{aligned}$$

Check Your Understanding: Compare with solution to Tikhonov regularization...

$$\begin{aligned} &\hat{x} = argmin_x(\|y - Hx\|_2^2 + \lambda\|\rho x\|_2^2), \\ &\hat{x} = (H^TH + 1\lambda)^{-1}(H^Ty). \end{aligned}$$

1.5 Intro to Sparsity

Definition: N-Sparse $\rightarrow$ N non-zero elements in vector.
Example: 3-sparse

$$\begin{bmatrix}1\\1\\1\end{bmatrix} \quad \begin{bmatrix}1\\1\\1\\0\\0\\...\\0\end{bmatrix}$$

Sparsity in a Linear Program

Problem: $min\|x\|_0 \ s.t. \ Ax = b$, where $\|x\|_0$ = non-zero's in x...

Uh Oh! Combinatorial optimization $\rightarrow$ no explicit derivative!
Can we find a *relaxation* of the problem?

Check Your Understanding: Is the l_0 norm a norm or does it violate the properties of norm?

1. $(\sum x^0)^{\frac{1}{0}} = (\sum x^0)^{\infty}$ [undefined]
2. $\|\alpha x_0\| = (\sum x^0)^{\frac{1}{0}} \neq \alpha\|x_0\|$ [no isometry]

Norm Comparision:

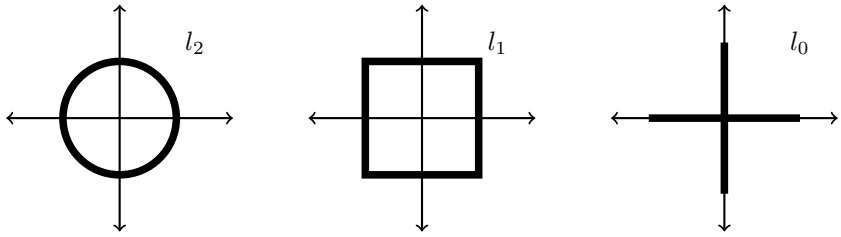


Figure 7:

Relaxation I. $min\|x\|_0 \ \ s.t. \ \|Ax - b\|_2 \leq \epsilon$ (i.e. relax constraint)

Relaxation II. $x* = \underset{x}{argmin}\|Ax - b\|_2 \ \ s.t. \ \|x\|_0 \leq S$ (i.e relax sparsity)

Example: Using sparsity in $\mathbb{R}^2$

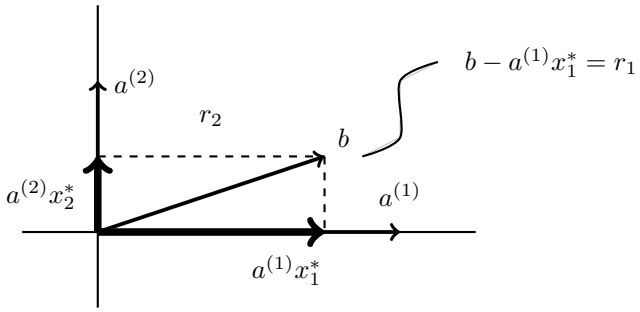


Figure 8:

---

let $x \in \mathbb{R}^2$, $S = 1$ **define** $r = residual$, $v_i \equiv i^{th}$ component of vector v
search all possible x^* $s.t.$ $r_1 = b - a^{(1)}x_1^*$ is minimized
search all possiblex^* $s.t.$ $r_2 = b - 1^{(2)}x_2^*$ is minimized

if $r_1 \leq r_2$ **then**
 | $x^* = [x_1^* \quad 0]^T$
end
else
 | $x^* = [0 \quad x_2^*]^T$
end

See figure above. Next class, we'll fully explain O.M.P.

Check Your Understanding: Tabletop Touch Sensor

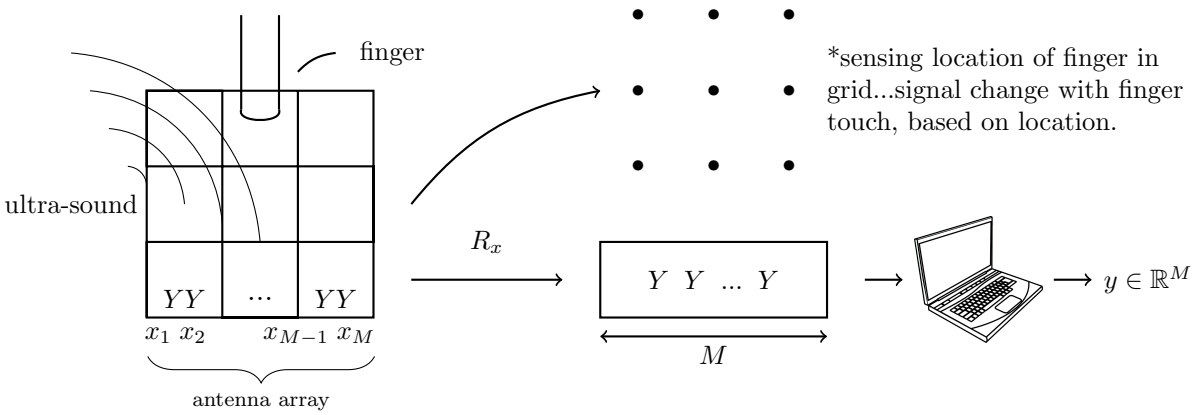


Figure 9:

Measurement Dictionary: $D = \begin{bmatrix} y_1 & y_2 & ... & y_9 \end{bmatrix} \in \mathbb{R}^9$ b/c 9×9 $grid$

Sparsity Constraint: Because there is 1 finger, we expect a 1-sparse solution.

Problem and Method: To find the finger position, compute the similarity score between $\hat{y}$ (i.e. measured) and 9 columns in dictionary $\Rightarrow \underset{index}{max}[D^T \hat{y}]$

What if there are 2 fingers? We find top 2 indexes... essentially the basic matching pursuits algorithm! But there is something better...

Next Class: Orthogonal Matching Pursuit

University of California, Los Angeles
UCLA Department of Electrical And Computer Engineering

EE211-Image Processing

Spring 2019

Lecture 8

Date: April 30, 2019 **Scribe: Irfan Syed**

1 Definitions:

- $\|x\|_0 - \ell_0$ is the "l-zero norm" or the number of non-zero entries in x. Note that this is not a true norm as it doesn't follow the rules of norms.
- $\|x\|_F -$ Frobenius norm is a matrix norm of an $m \times n$ matrix A defined as the square root of the sum of the absolute squares of its elements.

$$\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2}.$$

Note that this is equal to $\|A\|_F = \sqrt{Tr(AA^H)}$ where Tr is the matrix trace and A^H is the Hermitian (conjugate transpose) of A.

- The matrix trace (Tr) is the sum of diagonal elements in a matrix $Tr(A) = \sum_{i=1}^{n} a_{ii}$

2 Finding a sparse representation of a signal

Solving $\min \|x\|_0 \quad s.t. \;\; Ax = b$

Claim: this is a trivial problem to solve, if looking for a 1-sparse solution (1-sparse meaning 1 non-zero entry in x)

---

for *j = 1: numcol* **do**
  //numcol = number of columns in A
  $y_j = a_j^T b$ // this is the similarity value of jth column
end
$x_j = A^{T}_{j} b$ for jth column with maximum similarity value
$x_j = 0$ for all other values of j

Algorithm 1: $min\|Ax - b\|_2 \quad s.t. \;\; \|x\|_0 = 1$

Solving this problem for different levels of sparsity is more difficult. A naive approach for S = 2 would be to take the rst and second highest similarity scores as shown below.

for *j = 1: numcol* **do**
  //numcol = number of columns in A
  $y_j = a_j^T b$ // this is the similarity value of jth column
end
$x_j = A^{T}{}^{b}$ for jth column with maximum similarity value $x'_j = A'^{T}_j b$ for the jth column with second highest similarity $x_j = 0$ for all other values of j

Algorithm 2: $min\|Ax - b\|_2 \;\; s.t. \;\; \|x\|_0 < 2$

However, this algorithm will only work well if columns in A, A_j^T are orthogonal to each other. Therefore, this is not a great assumption, and is denitely not scaleable as S grows. This leads to a new algorithm called orthogonal matching pursuit or OMP for short.

Here is a fail example adding to explain the notation why non-orthogonal may not work for this algorithm.

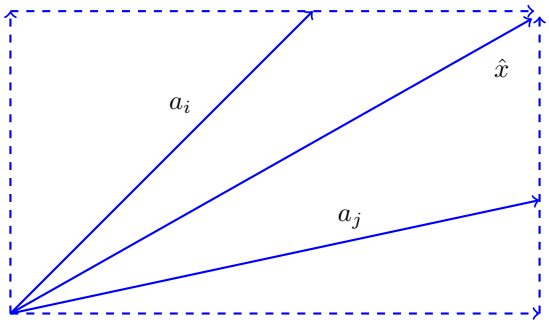


Figure 1:

In the figure, we can see there is basis vector $[a_i, a_j]$, and according to previous algorithm, there is a b vector which can make $\hat{x} = [a_i * b, a_j * b]$. If two vectors are orthogonal to each other, it will perform like dot line in the gure. However, two vectors may have multiple overlapping gures, and during the algorithm, we only measure the similarity for each separately. Finally, the $\hat{x}$ will be larger than the original one.

- $r \leftarrow b$ // r is the residual of b that needs to be accounted for still
- $\Omega_{new} = \{\Phi\}$ // Ω is the columns of A, we are taking a subset of these and putting them in a new set

while $\|x\|_0 \leq S$ **do**
  for *all* $j \in \Omega$ **do**
    Find the 1-sparse solution
    $x_j = a_j^T r$ // compute projection
    $i = argmax_{j \notin \Omega} |x_j|$ // take max for $j \in \Omega$
    // Add column to set, find new residual value, break or repeat
    $\Omega_{new} \leftarrow \Omega_{new} \cup \Omega_i$ // add max column to new set
    $x^*_\Omega = argmin\|A_\Omega X_\Omega - b\|_2^2$ // restricted least squares, can solve via pseudo inverse
    $r \leftarrow b - A_\Omega x_{\Omega *}$ // if $|r| <$ tolerance $\rightarrow$ break
  end
end

Algorithm 3: $min\|Ax - b\|_2 \;\; s.t. \;\; \|x\|_0 < S$ (OMP)

This solves the non-blind problem: where A is already known. This algorithm can be improved by limiting the search region to be close to a previous column (known as structured sparsity). This can be useful in situations like wide-band radar where the best solutions are likely to be nearby. Some other applications of OMP are *super resolution, compressed sensing, signal reconstruction,* and *denoising.*

Previously using OMP, we were trying to nd sparse representation of A using one measurement of B. To bring this problem in machine learning perspective, we will use multiple measurement. This means that b becomes B, a set of training data, and x becomes X, a set of measured data.//

$$\begin{aligned}
&\min_x \left\| Ax^{(1)} - b^{(1)} \right\|_2 \quad s.t. \quad \left\| x^{(1)} \right\|_0 \leq S, \\
&\min_x \left\| Ax^{(2)} - b^{(2)} \right\|_2 \quad s.t. \quad \left\| x^{(2)} \right\|_0 \leq S, \\
&\qquad\qquad\qquad\qquad\qquad\qquad . \\
&\qquad\qquad\qquad\qquad\qquad\qquad . \\
&\qquad\qquad\qquad\qquad\qquad\qquad . \\
&\min_x Ax^{(N)} - b^{(N)}{}_2 \quad s.t. \quad \left\| x^{(N)} \right\|_0 \leq S.
\end{aligned}$$

Above optimization problems can be collectively written in matrix format as:

$$\min_X \|AX - B\|_F^2 \quad s.t. \quad \left\| x^{(i)} \right\|_0 \leq S \; \forall_i.$$

Now, to solve a blind problem (like dictionary learning), we need to optimize over A also. Our new problem becomes:

$$\min_{A,X} \|AX - B\|_F^2 \quad s.t. \quad \left\| x^{(i)} \right\|_0 \leq S \; \forall_i.$$

3 Dictionary Learning:

It is a branch of signal process and machine learning that aims at finding a frame (called a dictionary) in which some training data admits a sparse representation.

Example scenario

Consider a picture of a person. In one small section, or patch, the shoulder is captured. This patch will have energy compacted into low frequencies, and high frequencies will be mostly zeros. If our matrix A (in Ax = b, where x is the data, and b is our measurement) is proportional to frequency content, then we can say the patch admits a sparse representation. Note that a noisy image patch has energy across both low and high frequencies and thus can't be compacted into low frequency bands, so in this way, the patch does not admit sparse representation in matrix A.

Dictionary learning uses training data, a collection of noiseless patches from images to create a matrix A that will inherently remove noise and other unwanted signals from an image. This is because the noise and signal will live in dierent subspaces if the data is trained only on the signal. Since it is hard to represent noise sparsely, sparse representations get rid of noise.

In other words, when you solve a sparse least squares problem, your solution will not reconstruct the signal perfectly. The errors will be where the signal includes portions that can't be sparsely represented by A, and as it is discussed above, this portion is what the noise is. Therefore, the noise will not be reconstructed in the output image.

3.1 Dictionary Learning Process

- Step 1: Learn A^* based on clean data set. [**Dictionary Learning**]

- Step 2:

  (a) $\tilde{b}$ (noisy measurement) given with a watermark

  (b) find a sparse representation via matching pursuit algorithms [**OMP**]

$$\hat{x} = \underset{x}{argmin} ||\tilde{b} - A^* x||_2 \; s.t. \; ||x||_0 \leq s.$$

- Step 3: Reconstruction $b_{denoised} = A^* \hat{x}$

University of California, Los Angeles
UCLA Department of Electrical And Computer Engineering

EE211-Image Processing

Spring 2019

Lecture 9

Date: May 2, 2019 **Scribe: Jack Tran**

1 Dictionary Learning

1.1 Problem Formulation

$$\underset{A,x}{argmin}\|Ax - B\|_F^2 \, s.t. \left\|x^{(i)}\right\|_0 \leq S \quad \forall i.$$

1.2 Solution 1: MOD (Method of Optimal Directions)

Steps

Step 1: Fix A, solve for x.

Step 2: Fix x, solve for A.

Step 3: Repeat until convergence.

Check Your Understanding: How to implement the above steps?

Step 1: Any pursuit algorithm e.g. OMP.

Step 2: Linear regression. (Because there are no sparsity constraints.)

$$\mathbf{A*} = \mathbf{Bx^T}(\mathbf{xx^T})^{\mathbf{-1}}.$$

1.3 Solution 2: K-SVD (Naive)

Formulation: K-SVD is a solution to learn A

$$\|\mathbf{B} - \mathbf{Ax}\|_F^{\mathbf{2}} = \sum_{\mathbf{i}} \left\| b^{(\mathbf{i})} - \mathbf{Ax}^{(\mathbf{i})} \right\|_{\mathbf{2}}^{\mathbf{2}}.$$

1.4 Steps

Step 1: Initialize the dictionary.

Step 2: Break into sub-problems and solve for x^i [Method: pursuit algorithms].

Step 3: Given x, update A atom-by-atom (col-by-col)

$$\begin{aligned}\|B - Ax\|_F^2 &= \left\| B - \sum_{j=1}^{k} a_j x_T^j \right\|_F^2, \\ &= \left\| B - \left(\sum_{j \neq k} a_j x_T^j \right) - a_k x_T^k \right\|_F^2, \\ &= \left\| E_k - a_k x_T^k \right\|_F^2.\end{aligned}$$

We want to find

$$\underset{a_k}{argmin} \left\| E_k - a_k x_T^k \right\|_F^2.$$

Notation

- a_j is jth column of A.
- x_T^j is jth row of x (jth column of x^T)
- $\mathbf{E_k}$ is the error term due to removing the k-th column.

$$\mathbf{E}_k = B - \left(\sum_{j \neq k} a_j x_T^j \right).$$

Check Your Understanding: Find pseudo code to decompose M into rank 3 matrix

$$\begin{aligned}&SVD(M) \rightarrow [U, D, V^T], \\ &\tilde{D} = D \odot M, \\ &\tilde{M} = U \tilde{D} V^T M.\end{aligned}$$

Question: Why does SVD not work for rank 1 approximation of E_k?
Answer: x_T^k might not be sparse.

1.5 Solution 3: K-SVD (Pseudocode)

Step 1. Initialize A (perhaps with the output of a CNN.

Step 2. Sparse Coding of X: Use OMP (or some other algorithm) to optimize the following objective $\forall$ i:

$$\underset{x^{(i)}}{min} \left\| b^{(i)} - Ax^{(i)} \right\|_2^2 s.t. \left\| x^{(i)} \right\|_0 \geq S.$$

Step 3. Dictionary Update. Update A, column-by-column (atom-by-atom).

for $i \in 1, 2, ..., K$ **do**

Set $\omega_k = \{i | x_T^k(i) \neq 0\}$

Set $E_k = \mathbf{B} - \sum_{j \neq k} a_j x_T^j$

Set $\tilde{E}_k = \Omega[E_k]$ where Ω is the truncation operator that only keeps columns that are in ω_k

Set[U, D, V^T]=SVD($\tilde{E}_k$)

Set $a_k = U_1, \tilde{x}_T^k = [v_1 \cdot D_{1,1}]^T$

end

Repeat steps 2 and 3 until convergence.

University of California, Los Angeles
UCLA Department of Electrical And Computer Engineering

EE211-Image Processing

Spring 2019

Lecture 10

Date: May 7, 2019 **Scribe: Steven Chen**

1 Motivation

Many real-world signals have a sparse representations. This is why we can perform compression, even lossy compressions like JPEG, and still "reconstruct" the original signal.However, the ubiquity of postprocessing compression raises the question — why do we bother collecting the entire signal in the first place if we're just going to compress it afterward? As such, we begin to think about ways to remove the redundancy in our data collection process and compress *while* we sense the data — this leads us into Compressed Sensing (CS).

1.1 Definition 0.1. Compressed Sensing:

A linear dimensionality reduction technique which allows the sampling rate to undershoot the Shannon-Nyquist sampling theorem. CS assumes that the signal being sampled is:

1. Spasely represented insome basis, ψ
2. Has incoherency between the smapling basis ϕ and the basis ψ, meaning that the signal is *not* sparse in ϕ. Incoherency is the principal that explains why a Dirac delta in the time domain is an even spread in the frequency domain.[1][2]

Let's consider a real-life example and what implications CS has. In seismic acquisition, it is important to have a very comprehensive idea of where and when earthquake activity is happening. Traditionally, this means that we require sensors which, both in time and location, obey the Shannon-Nyquist sampling theorem. This means that we need a lot of thoughtfully placed sensors with very high sampling rates. However, CS provides us a path to eventually reconstruct our original signal with better fidelity from sensors that violate the Shannon-Nyquist sampling theorem. [3]

2 Solving the Underdetermined Linear System

Consider a familiar sparse basis, the Fourier basis. When we aim to find DFT coefficients, we may frame and solve the problem like this:

$$\begin{aligned} y &= Ax, \\ x &= A^{-}1y, \\ &\underset{x}{argmin} \|Ax - y\|_2^2 . \end{aligned}$$

and may choose to include some regularization.

In CS, we are working with an underdetermined system (wide A). As it turns out, Occam's razor provides some insight into how to solve this optimization problem — the simplest solution is the best one. So

what's the simplest solution? Recall that there always $\exists D s.t.\ ||Dx|| \leq S$, meaning that we can always find a domain where the signal is sparse. In this case, the simplest solution is the sparse solution. This means we might want to solve this problem

$$\underset{x}{argmin} \|Ax - y\|_2^2 + \|x\|_0$$

as a solution to the underdetermined system. However, this problem is not tractable (it may take an infeasibly long time to find a solution) due to the L_0 norm.

To solve this problem, let's begin by rewriting it:

$$\min \|x\|_0 \, s.t. \quad Ax = b.$$

We **claim** that we can take the following *convex relaxation*:

$$\min \|x\|_1 \, s.t. \quad Ax = b,$$

which can be equivalently expressed [4] as

$$\min \sum yi \ \ s.t. \quad Ax = B, xi \leq yi, -xi \leq yi.$$

This problem *is* tractable and, as discussed in previous lectures, can be solved with pursuit algorithms.

3 Restricted Isometry Property

Let's prove the above claim that we use (P1) to solve (P0). Let's start with a simple pictional proof of this claim.

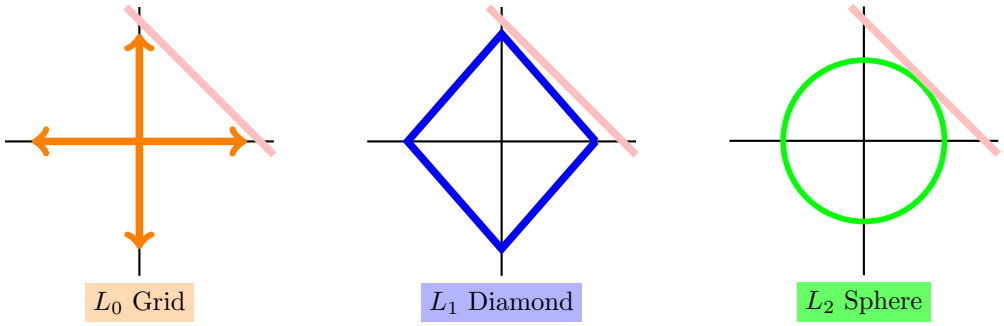


Figure 1: Pictoral proof that solving P_1 is a good approximation to P_0

More concretely, we want A to have the Restricted Isometry Property (RIP). If RIP holds, then P1 approximates P0.

3.1 Definition 0.2. Restricted Isometry Property:

We say that A has RIP with conditions k and δ_k, such that $\forall$ ——x——$_0 \leq k$ (the signal x is k-sparse),

$$(1 - \delta_k)\|x\|_2^2 \leq \|Ax\|_2^2 \leq (1 + \delta_k)\|x\|_2^2 .$$

This is equivalent to saying that A has an isometric map with x.‘

3.2 Definition 0.3. Isometry:

If A applied to vector, length is preserved. In other words, this is similar to saying that A is "pseudo"-orthogonal.

Therefore, RIP describes the orthogonal-ness of an underdermined A. Recall that an orthogonal matrix has a conditional number 1, and is very easy to invert. This gives us some intuition as to why we want A to have RIP. Additionally, we can note that if A is approximately preserving the Euclidean length of k-sparse vectors, then the set of k-sparse vectors should not be in the nullspace of A. We therefore stand a chance of finding a k-sparse solution to the problem!

How do we prove A has RIP without checking it against every possible x in the world? For A $\in \mathbb{R}^{M\times N}$, RIP typically holds for $M << N$. Luckily,$M << N$ is the situation for the underdermined system that we are working with in compressed sensing! This means that the signal size N that we wish to reconstruct is much larger than the signal size M that we sensed. [5] The size requirement follows $M = O(klog(\frac{N}{k}))$.

Consider a few examples of this sizing requirement. If we use a $k = 16$-sparse signal, we can achieve the following compression rates:

Desired N	**Required M**	**Compression Ratio**
128	30	23%
1024	60	6%
65536	120	0.2%

4 C-Almost Euclidean

Let's define some further constrainsts on our data to prove that CS works. To do this, we need to understand the following type of subspace.

4.1 Definition 0.4 (C-Almost Euclidean (C-AE) Subspace:).

A subspace $\Gamma \in \mathbb{R}^N$ is C-AE if $\forall_v \in \Gamma$,

$$\frac{1}{\sqrt{N}}\|v\|_1 \leq \|v\|_2 \leq \frac{C}{\sqrt{N}}\|v\|_1 .$$

where C is slack for the condition. In words, this means that the L_2 norms of v are close to the L_1 norms with C slack.

Examples:

1. $v = [\frac{1}{\sqrt{N}}, ..., \frac{1}{\sqrt{N}}]^T$
   $\rightarrow$ The vector is dense.
   $\rightarrow \|v\|_1 = \sqrt{N}, \|v\|_2 = 1.$
2. $v = [1, 0, ..., 0]^T$
   $\rightarrow$ The vector is sparse.
   $\rightarrow \|v\|_1 = 1, \|v\|_2 = 1.$

We eventually want to show that $N(A)$ should be a C-AE subspace, Γ. To understand why we would want the nullspace of A to have $L_1 \approx L_2$, let's look at a visual. We saw in (16) that L_1 can be an acceptably sparse substitute for L_0 because it "enforces" intersections with the solution space at the edges of the L_1-diamond. In higher dimensionality spaces, there is more opportunity to intersect the L_1-diamond in non-sparse ways. Consider the solution space shown in (17).

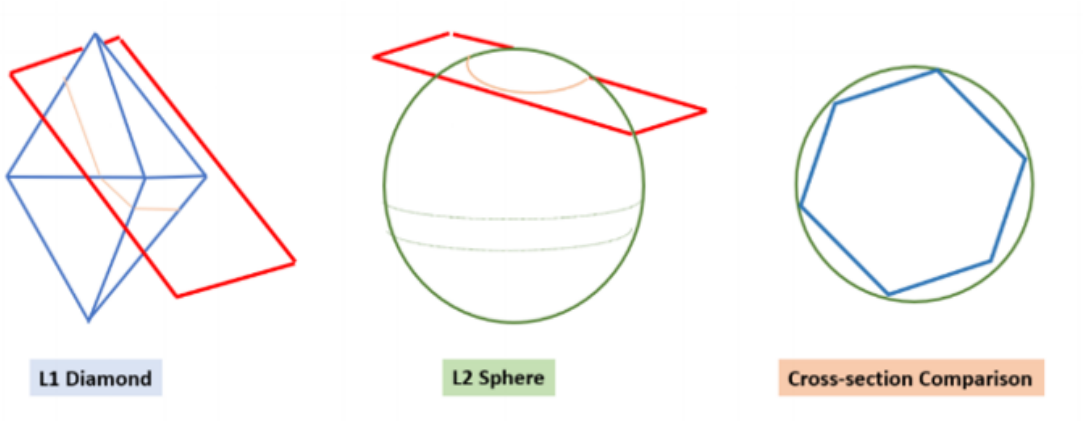


Figure 2: A solution space intersecting with the L1 diamond in a "nonsparse" way. The cross sections of the L_1 and L_2 spaces are more similar.

When the L_1 norm and the L_2 are more similar, the solution is less likely to be sparse. Therefore, we want the set of vectors with $L_1 \approx L_2$ to be in the nullspace of A! Put another way, we want $\Gamma \cap \{xs.t.\|x\|_1 \leq 1\}$ to be approximately spherical. Because there is overlap between the entire solution set and the nonsparse (undesirable) solutions, this helps to ensure that the nonsparse solutions are in the nullspace of A.

Armed with this information, we can begin to find the kernal of A and prove that it is an AE subspace.

4.2 Claim 0.1.1

$\frac{1}{\sqrt{N}}\|v\|_1 \leq \|v\|_2$ *for all* v*, all regardless of* Γ.

Proof. We use the Cauchy-Schwarz inequality for this proof.

$$\begin{aligned}
\|v\|_1 &= < v, \text{sign}(v) >, \\
&\leq \|v\|_1 \cdot \left\|\text{sign}(v)\right\|_2, \\
&= \|v\|_2 \sqrt{|\text{supp}(v)|}, \\
&\leq \|v\|_2 \sqrt{|\max(\text{supp}(v))|}, \\
&\leq \|v\|_2 \sqrt{N}, \\
\frac{1}{\sqrt{N}}\|v\|_1 &= \|v\|_2 .
\end{aligned}$$

4.3 Lemma 0.2.

If $v \in \Gamma, v \neq$, *then* $|supp(v)| \geq \frac{N}{C^2}$.

Proof. Using (50) and (0.4)

$$\begin{aligned}
\|v\|_1 \leq & \|v\|_2 \sqrt{|\text{supp}(v)|} \leq \frac{C}{\sqrt{N}} \|v\|_1 \sqrt{|\text{supp}(v)|}, \\
\|v\|_1 \leq & \frac{C}{\sqrt{N}} \|v\|_1 \sqrt{|\text{supp}(v)|}, \\
1 \leq & \frac{C}{\sqrt{N}} \sqrt{|\text{supp}(v)|}, \\
\frac{N}{C^2} \leq & |\text{supp}(v)|.
\end{aligned}$$

Recall that T is the "support set", as described in the OMP lecture.

4.4 Lemma 0.3.

No one vector in Γ *can contain overwhelming "mass". Let* $v \in \Gamma, v \neq 0$, $T \in [1, ..., N], |T| \leq \frac{S}{16}$. *Then* $||v_T||_1 \leq \frac{||v||_1}{4}$.

The intuition behind this lemma is that this sets a *concentration bound*. For example,

$$\begin{aligned}
v &= [1, 3, 5, 0, 0, 0, 0, 0, 0], \\
v_T &= [1, 3, 5].
\end{aligned}$$

This is not what we want! The vector v violates lemma (0.3). We want the "mass" of v to be distributed in an even way.

The proof of this is continued in the following lecture.

University of California, Los Angeles
UCLA Department of Electrical And Computer Engineering

EE211-Image Processing

Spring 2019

Lecture 11

Date: May 21, 2019 **Scribe: Jordan Lee**

1 Compressive Sensing: Part 2

1.1 Lemma

Let $v \in \Gamma, v \neq 0, T \in (1, 2, ..., N), |T| \leq \frac{S}{16}$, then $\|v_T\|_1 \leq \frac{\|v\|_1}{4}$.

Interpretation: All vectors in Γ have uniform mass. Γ has no imbalanced vector.

Proof. $\|v_T\|_1 \leq \sqrt{|T|}\|v_T\|_2 \leq \sqrt{|T|}\|v\|_2 \leq \frac{C\sqrt{|T|}}{\sqrt{N}}\|v\|_1 \leq \frac{C\sqrt{\frac{N}{16C^2}}}{\sqrt{N}}\|v\|\,1 = \frac{\|v\|_1}{4}$.

The first inequality follows from Cauchy-Schwarz inequality. The second inequality follows from the L_2 norm of part of a vector is less than or equal to the L_2 norm of the vector. The third inequality follows from the definition of C-AE subspace.

1.2 Theorem

If $Ax = b$, *and* $\|x\|_0 \leq \frac{S}{16} = \frac{N}{16C^2}$ *and* $\Gamma = kernel(A)$ *is C-AE, then* x *is the uniquely optimal solution to following optimization program:*

$$\underset{x}{argmin}\|x\|_1 \quad s.t. \quad Ax = b.$$

Proof. We are going to prove this theorem by using proof by contradiction. Suppose there exists another solution ω which also seems optimal. To prove the theorem, we nee to prove $\|\omega\|_1 > \|x\|_1$

Let's define a set T such that $T = supp(x)$, Recall that the support of a vector is the set consisting of all indices corresponding to nonzero entries, and by definition of the support we have $\|x_T\|_1 = \|x\|_1$ and $\|x_{\bar{T}}\|_1 = 0$
Since $\omega \neq x$, we can write:

$$\begin{aligned}\omega &= x + v,\\ \omega_T &= x_T + v_T,\\ \omega_{\bar{T}} &= x_{\bar{T}} + v_{\bar{T}}.\end{aligned}$$

Using triangle inequality, we have:

$$\|\omega_T\|_1 = \|x_T + v_T\|_1 \geq \|x_T\|_1 - \|v_T\|_1 = \|x\|_1 - \|v_T\|_1 .$$

From the definition of the support, $\|x_{\bar{T}}\|_1 = 0$ and we have:

$$\|\omega_{\bar{T}}\|_1 = \|x_{\bar{T}}\|_1 + \|v_{\bar{T}}\|_1 = \|v_{\bar{T}}\|_1 .$$

We can rewrite $\|\omega\|_1$ as following:

$$\begin{aligned}\|\omega\|_1 &= \|\omega_T\|_1 + \|\omega_{\bar{T}}\|_1, \\ &\geq \|x\|_1 - \|v_T\|_1 + \|v_{\bar{T}}\|_1.\end{aligned}$$

Add and then subtract $\|v_{\bar{T}}\|_1$ from the right hand side and use $\|v\|_1 = \|v_T\|_1 + \|v_{\bar{T}}\|_1$, we get:

$$\|\omega\|_1 > \|x\|_1 - \|v_T\|_1 + \|v_{\bar{T}}\|_1 + \|v_T\|_1 - \|v_T\|_1 = \|x\|_1 - 2\|v_T\|_1 + \|v\|_1$$

We are pretty close to what we want to prove ($\|\omega\|_1 > \|x\|_1$) and we only have to prove $-2\|v_T\|_1 + \|v\|_1$ is strictly positive. Recall that **Lemma 2** lets us relate $\|v_T\|_1$ and $\|v\|_1$ if v is in a C-AE subspace. The next step is prove that v is actually in the C-AE subspace Γ. Since ω and x are both solutions, we have

$$\begin{aligned}&A\omega \text{ and } Ax = b, \\ &\Rightarrow A(\omega - x) = Av = 0, \\ &\Rightarrow v \in \Gamma.\end{aligned}$$

Now that we have proved v is in the C-AW subspace Γ, we can use **Lemma 2:**

$$\|v_T\|_1 \leq \frac{\|v\|_1}{4}.$$

Plug this in the inequality:

$$\begin{aligned}\|\omega\|_1 &\geq \|x\|_1 - 2\|v_T\|_1 + \|v\|_1, \\ &\geq \|x\|_1 - 2(\frac{\|v\|_1}{4})_1 + \|v\|_1 = \|x\|_1 + \frac{\|v\|_1}{2}.\end{aligned}$$

Since $\omega \neq x$ we have $v \neq 0$ and $\|v\|_1 > 0$. Therefore, we have proved

$$\|\omega\|_1 > \|x\|_1.$$

University of California, Los Angeles
UCLA Department of Electrical And Computer Engineering

EE211-Image Processing

Spring 2019

Lecture 12

Date: June 4, 2019 **Scribe: Lucas He**

1 3D Captured

1.1 3D Registration

Definition: Fitting one model into the coordinate system of a model, e.g., alignment and deformation. In computer vision, we usually only consider rigid registration, in contrast to non-rigid registration.

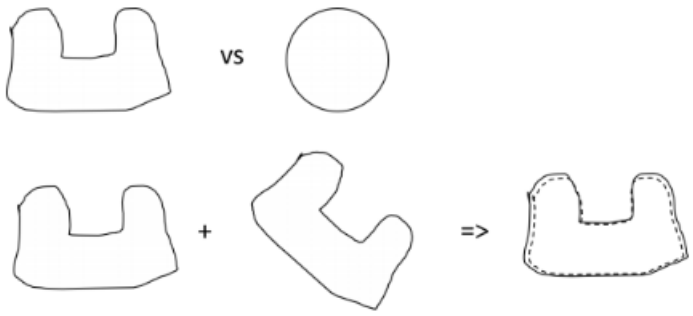


Figure 4: Non-rigid vs Rigid

2 Rotation Matrix review

2.1 2D rotation

Given A system, where p is original pixel, and p' is the rotated pixel.

$$p' = Rp, R = \begin{bmatrix} cos(\alpha) & -sin(\alpha) \\ sin(\alpha) & cos(\alpha) \end{bmatrix}.$$

Check Your Understanding: How to rotate p around another point q instead of the origin?

$$p' = R(p - q) + q.$$

Adding notes: The following figure will give a more clear illustration. The dot line represent $R(p - q)$.

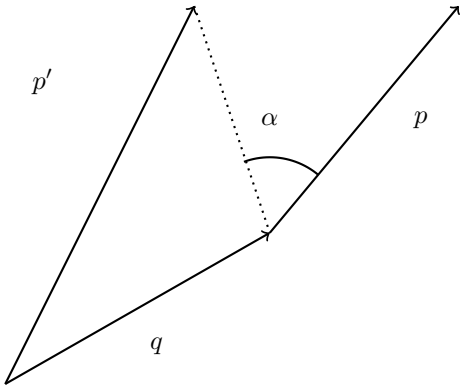


Figure 2:

2.2 3D rotation

With similar formulation as 2D,

$$p' = Rp, p' = \begin{bmatrix} p'_x \\ p'_y \\ p'_z \end{bmatrix}, p = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}.$$

the rotation matrix around each axis are the following:

$$R_z = \begin{bmatrix} cos(\alpha) & sin(\alpha) & 0 \\ sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}, R_y = \begin{bmatrix} cos(\alpha) & 0 & sin(\alpha) \\ 0 & 1 & 0 \\ -sin(\alpha) & 0 & cos(\alpha) \end{bmatrix}, R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\alpha) & -sin(\alpha) \\ 0 & sin(\alpha) & cos(\alpha) \end{bmatrix}.$$

2.3 Property of R matrix

- Orthonormal $R^T R = RR^T = I \Rightarrow R^{-1} = R^T$
- $Rv = \lambda v$, where v is eigenvector of rotation axis, and λ is eigenvalue.
- $Trace(R) = \begin{cases} 2cos\alpha & ,2D \\ 1 + 2cos\alpha & ,3D \end{cases}$

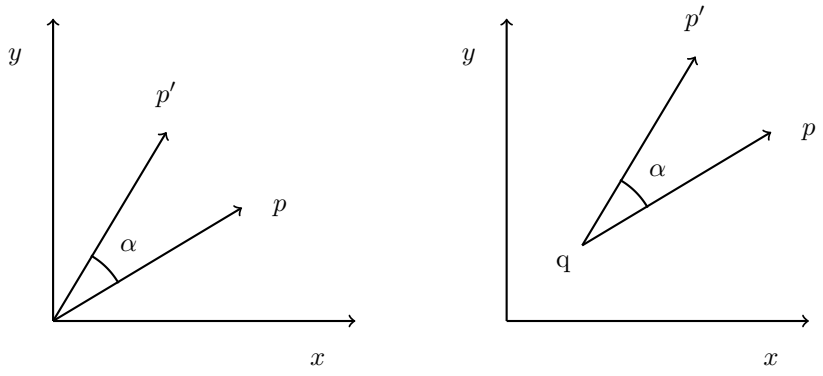


Figure 3:

3 Alignments

Input: 2 points sets
Output: 1 point set aligned to the other

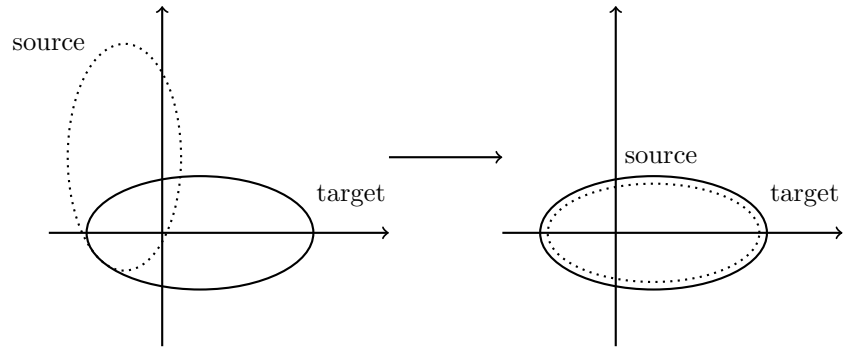


Figure 4:

4 Three methods:

- PCA: Align principle component axes
- SVD: Need knowledge of correspondence
- Iterative Closest Point (ICP): Joint alternating optimization

4.1 Method 1 PCA:

Treat images as eclipses ⇒ align source and target's centroid and principle component axes.

Let's assume source image = $\begin{bmatrix} p_S^{(1)} & ... & p_S^{(n)} \end{bmatrix}$ and target image = $\begin{bmatrix} p_T^{(1)} & ... & p_T^{(n)} \end{bmatrix}$, where upper script is the index of the point.

4.1.1 Algorithm:

Step 1. Calculate the centroid of source and target image, C_S and C_T.

Step 2. Perform translation to align centroid,

$$p_s \Leftarrow p_s + (C_T - C_S).$$

Step 3. Perform rotation respect to centroid,

$$p_s \Leftarrow (R(p_s - C_T) + C_T).$$

Step 4. Find R
Consider a simple 2x2 case, with matrices A and B whose columns are principal axes.

$$\text{If } RA = B, \text{ then } R = BA^T, \text{ since } A^{-1} = A^T.$$

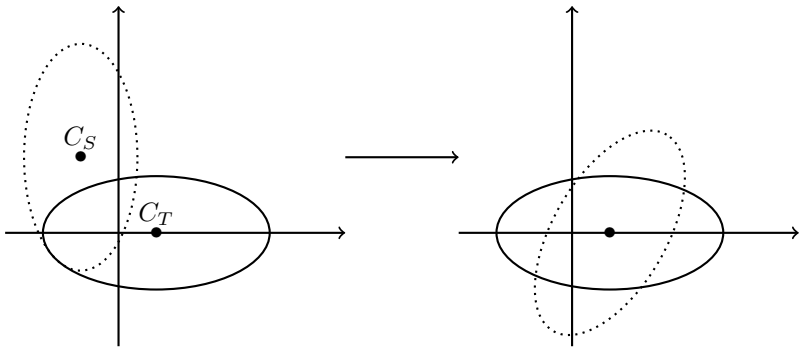


Figure 5: Translation Step

4.1.2 Problems

1. Noise will shift the centroid.
2. Symmetry will create rotation ambiguity.

4.1.3 Solution

Point Matching
Given source with $p(1)...p(n)$ with centroid location C_S, and target with $q(1)...q(n)$ with centroid location $C_T \Rightarrow$ map $q^{(i)} \leftrightarrow p^{(i)}$

4.2 Method 2 SVD:

Goal: Find R

$$\underset{R}{argmin} \sum_i \left\| q^{(i)} - p^{(i)} \right\|_2^2 = \underset{R}{argmin} \sum_i \left\| q^{(i)} - C_T + R(p^{(i)} - C_S) \right\|_2^2.$$

1. Make images zero mean,
$$P = \left[p^{(i)} - C_S, \quad ... \quad p^{(n)} - C_S \right],$$
$$Q = \left[q^{(i)} - C_T, \quad ... \quad q^{(n)} - C_T \right].$$
2. Find Cross-covariance matrix,
$$\Sigma^{(cross)} + PQ^T.$$
3. $R = VU^T$, where $U\Sigma V^T = SVD(\Sigma^{(cross)})$
Consider a simple example, where cross-covariance matrix,
$$\Sigma^{(cross)} = \begin{bmatrix} p_x^{(1)} & p_x^{(2)} \\ p_y^{(1)} & p_y^{(2)} \end{bmatrix} \begin{bmatrix} q_x^{(1)} & q_x^{(2)} \\ q_y^{(1)} & q_y^{(2)} \end{bmatrix} = \begin{bmatrix} p_x^{(1)} q_x^{(1)} + p_x^{(2)} q_x^{(2)} & \\ & p_y^{(1)} q_y^{(1)} + p_y^{(2)} q_y^{(2)} \end{bmatrix}.$$
Therefore, we need to maximize the trace: $\underset{R}{argmax} tr[RPQ^T] \triangleq tr[R\Sigma^{(cross)}]$, and the solution is $R^\star = VU^T$

4.3 Method 3 ICP:

Given uncorresponding point clouds $p^{(i)} \leftrightarrow q^{(i)}$

1. Apply PCA.
2. Guess correspondence, map $p^{(i)} \leftrightarrow q^{(i)}$ for i in $p^{(i)}$.
3. Apply SVD.
4. Loop 2 and 3 until $\epsilon = \frac{\sqrt{\sum \left\| p^{(i)} - q^{(i)} \right\|_2^2}}{n}$.