CS 473 – Medical Image Processing

Contents

Unit 1: Basic Image Processing - Lesson 01 - Introduction to CS 473/673

"medical image processing"
Goal: to give an overview of the topics the course will cover, as well as the structure of the course.

Medical Image Processing

- Medical image sources (CT, MRI, PET, US)
- Image and signal processing
- Image enhancement (denoising, deblurring)
- Registration (aligning images)
- Segmentation (tissue classification)
- Reconstruction

Registration

- Automatically align

Functional MRI

- MRI gets brighter in some parts of the image over time (brain)
- Stimulus on/off – brighter/darker

Digital subtraction: image, subtract something from it

- E.g. only want blood vessels

Multimodal fusion:

- T1 MRI: bone dark
- CT: right: bone light

Forensic Identification

- 2,1
- 4,5

FI (2)

- Uses fourier transform

Segmentation

- Red outlines the liver
- Left image is coloured to get right image

Volume studies

- White is cerebral spinal fluid
- BPF decreases with age

Lesson 02 – Images in Matlab

Images in Matlab

To look at how images are stored especially in matlab.

Storing Digital Images

Images are just arrays of numbers

- 2D, 3D or higher
- Colour (3 images, one for each colour channel, Red, Green, Blue… "R, G, B")

Each element of the array is called a **pixel**, which comes from **picture element**. For a 3D array (representing a variable), the elements are called **voxels**.

One disk, or in memory, the data is stored in a 1D array, so we need a convention to convert high-dimensional arrays to 1D arrays.

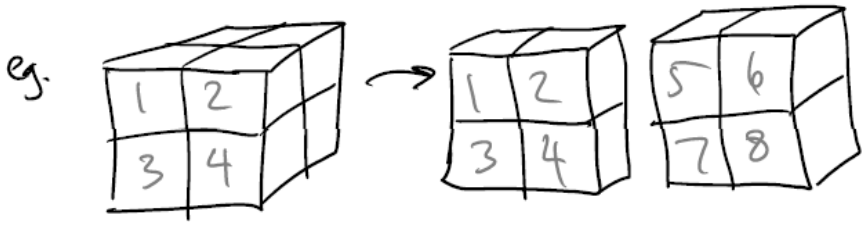
- In C++, 2D array is array of pointers

A common method:

<image: flattening matrix into 1D array, in C/C++>

<MATLAB: does indexing column-wise, e.g. [1 5 9 13 2 6 10 14…]

Storing a volume is simply a matter of storing many consecutive 2D images (slices), one image after the next.



[1 3 2 4 5 7 6 8…]

Hence, to read a volume into memory…

```
for z = 1 to Z
    for col = 1 to C (# cols)
        for row = 1 to R
            f(row, col, z) ←read value
        next row
    next col
next z
```

reading "depth"-wise: 1,5,… not cache efficient
slice to slice is better
this algorithm is done MATLAB's way

Colour

We will deal almost entirely with graylevel images in this course (one value per pixel/voxel). However, some applications save graylevel images as colour images with 3 identical colour channels.

Be aware of this, and make sure you aren't inadvertently using colour data.

MATLAB Coordinates

Something that makes coordinates awkward is the fact that MATLAB indexes its arrays using **(row, col)** format (like one would when specifying an element of a matrix).

e.g. $f = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix}$ element (2,3) is 6

But if we think of *f* as an image in the xy-plane using (x, y) coordinates, then (2,3) gives us **5.**

Moreover, MATLAB's indexing is **base-1**. The first element in an array has index **1**. In C and C++ (among others), indexing starts at **0**.

We can think of these differences as a coordinate system transformation.

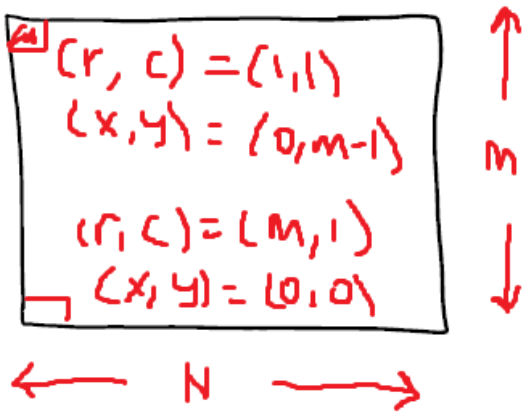
Coordinate System Transformations

It is not hard to see that we can convert between

(row, col) ←→ (x, y)

MATLAB index coordinates ←→ Cartesian

Suppose our image matrix has M rows and N columns.



$$\begin{cases} x = col - 1 \\ y = M - row \end{cases}$$

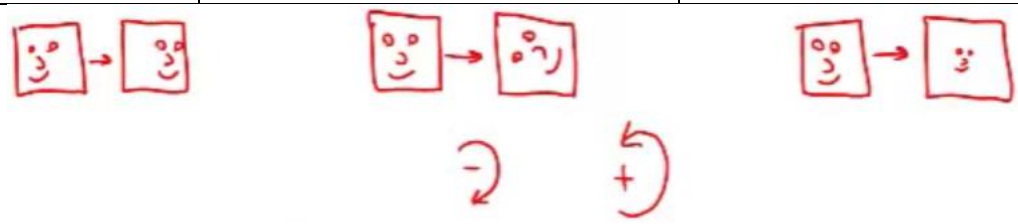
$$\begin{cases} row = M - y \\ col = x + 1 \end{cases}$$

Lesson 03 – Spatial Transformations

Goal: to learn how to represent affine transformations in an efficient way.

Images (and volumes) can be thought of as fields (or functions) of space. We will talk a lot about transformations of these spatial domains. Some examples are: **translation, rotation, scaling, etc.**

Translation	Rotation	Scaling
u = x + a **v = y + b** **translation of x, y to u, v**	**u = xcos(Θ) – ysin(Θ) (+ sin(Θ))?** **v = xsin(Θ) + ycos(Θ) (- sin(Θ))?**	**u = s_1x** **v = s_2x** **s_1 and s_2 are scalars**



Negative rotation: clockwise; Positive rotation: counter clockwise

Homogeneous Coordinates

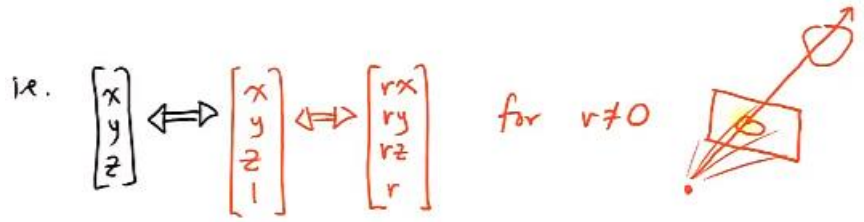
Notice above that rotation and scaling can each be written as matrix-vector products.

$\vec{u} = \begin{bmatrix} cos\Theta & -sin\Theta \\ -sin\Theta & cos\Theta \end{bmatrix}\vec{x}$, **Rotation**

$\vec{u} = \begin{bmatrix} s_1 & 0 \\ 0 & s_2 \end{bmatrix}\vec{x}$ **scaling**

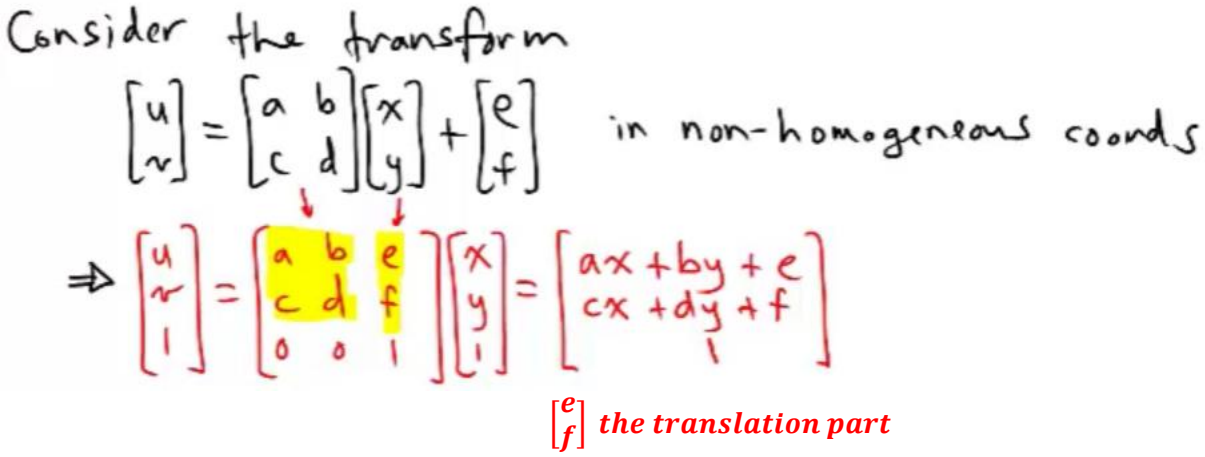
So called **"affine" transformations**

However, translation cannot. But we can embed translations into a matrix-vector product if we bump up the dimension by 1.



$$\begin{bmatrix} x \\ y \end{bmatrix} \leftrightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \leftrightarrow \begin{bmatrix} rx \\ ry \\ r \end{bmatrix}, r \neq 0$$

We call this augmented coordinate system **"Homogeneous coordinates"**.



$\begin{bmatrix} e \\ f \end{bmatrix}$ ***the translation part***

Affine part left (a, b, c, d), non-affine part right (e, f)

Example:
Rotate by 30 degrees, then translate by $[2\ 1]^T$. rotation matrix is

$$\boldsymbol{R_{30^\circ}} = \begin{bmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{\sqrt{3}}{2} \end{bmatrix}$$

Thus, P = $\begin{bmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} & 2 \\ \frac{1}{2} & \frac{\sqrt{3}}{2} & 1 \\ 0 & 0 & 1 \end{bmatrix}$

Affine part applied first, then translated.

Composition of Transformations

Why do we care about representing translations as a matrix-vector product?

Because we can do this:

- Two (or more) affine transforms can be combined into one affine transform by simple matrix multiplication.

Consider the composite transform:

1. P_1 ≡ rotate by $30°$
2. P2 ≡ translate by $\begin{bmatrix} 2 \\ 1 \end{bmatrix}$

$$P_1 = \begin{bmatrix} cos30 & -sin30 & 0 \\ sin30 & cos30 & 0 \\ 0 & 0 & 1 \end{bmatrix}, P_2 = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

Composite transform is

$$P_2P_1 = \begin{bmatrix} \boldsymbol{1} & \boldsymbol{0} & \boldsymbol{2} \\ \boldsymbol{0} & \boldsymbol{1} & \boldsymbol{1} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{bmatrix} \begin{bmatrix} \boldsymbol{cos30} & \boldsymbol{-sin30} & \boldsymbol{0} \\ \boldsymbol{sin30} & \boldsymbol{cos30} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{bmatrix}$$
$$= \begin{bmatrix} \boldsymbol{cos30} & \boldsymbol{-sin30} & \boldsymbol{2} \\ \boldsymbol{sin30} & \boldsymbol{cos30} & \boldsymbol{1} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{bmatrix}$$

Now suppose we add a 3rd transform to the sequence:

3. P_3 ≡ rotate by $45°$, then translate by $\begin{bmatrix} -1 \\ 1 \end{bmatrix}$

$$P_3 = \begin{bmatrix} \boldsymbol{cos45} & \boldsymbol{-sin45} & \boldsymbol{-1} \\ \boldsymbol{sin45} & \boldsymbol{cos45} & \boldsymbol{1} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{bmatrix}$$

The transform resulting from the sequential composition of all 3 can be written

$$P = \boldsymbol{P_3P_2P_1}$$

<MATLAB exercise L03_transforms.m> 19:14

P_all_three =

0.2588	-0.9659	1.1213
0.9659	0.2588	0.2929
0	0	1.0000

Decomposing Affine Transforms

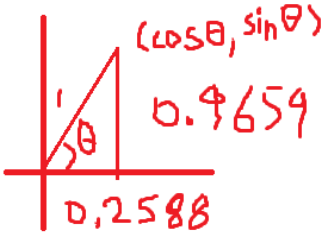
We can decompose a transformation into a single rotation and translation (& scale, skew [shear], etc.).

In the above case

$$\begin{bmatrix} cos\Theta & -sin\Theta & t1 \\ sin\Theta & cos\Theta & t2 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.2588 & -0.9659 & 1.1213 \\ 0.9659 & 0.2588 & 0.2929 \\ 0 & 0 & 1.0000 \end{bmatrix}$$

In quadrants, these are where only sine, all (cos, sin, tan), tan, cosine are positive.

- ⇨ $cos\Theta$ = 0.2588 (1,1)
  $sin\Theta$ = 0.9659 acosd(0.2588) = -75 deg (not other quadrant)
  t_1 = 1.1213
  t_2 = 0.2929

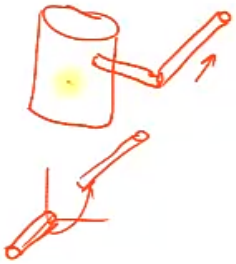


Thus $P_3P_2P_1$ is equivalent to

- Rotation by $-75°$, followed by
- Translation of $\begin{bmatrix} 1.1213 \\ 0.2929 \end{bmatrix}$

In graphics, transformation to put object in right spot

- Each object has orientation in three space
- Apply translations and rotations
- Global reference frame – body to arm to actuator
  - Add little reference pieces



It is often helpful to think of these transforms in terms of their affine part and translation part.

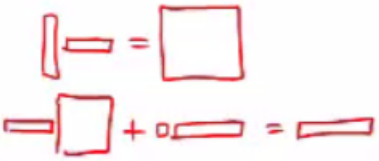
$$P = \left[\begin{array}{c|c} A & T \\ \hline 0 & 1 \end{array}\right] \quad A \in \mathbb{R}^{dxd} \quad T \in \mathbb{R}^{dx1}$$

0: row vect.

d=2 for 2D
d=3 for 3D
etc.

T: translation vector

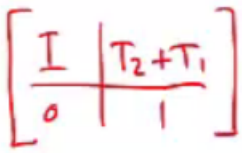
0: row vector of zeros



Combining Simple Transforms

Translations:

$$\left[\begin{array}{c|c} I & T_1 \\ \hline 0 & 1 \end{array}\right]\left[\begin{array}{c|c} I & T_2 \\ \hline 0 & 1 \end{array}\right] = \begin{bmatrix} I^2 + T_2 0 & IT_2 + T1 * 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} I & T_2 + T_1 \\ 0 & 1 \end{bmatrix}$$



Thus, combining translations simply **adds** the translations.

Inverse of translation T_1 is translation by **$-T_1$**

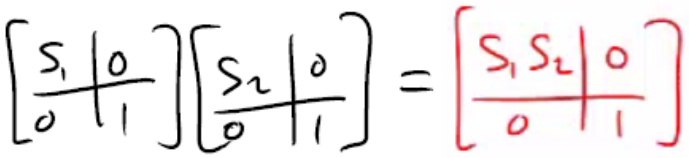
Rotations:

$$\left[\begin{array}{c|c} R_{\theta_1} & 0 \\ \hline 0 & 1 \end{array}\right]\left[\begin{array}{c|c} R_{\theta_2} & 0 \\ \hline 0 & 1 \end{array}\right] = \left[\begin{array}{c|c} R_{\theta_1} R_{\theta_2} & 0 \\ \hline 0 & 1 \end{array}\right] = \left[\begin{array}{c|c} R_{\theta_1+\theta_2} & 0 \\ \hline 0 & 1 \end{array}\right]$$

Thus, combining rotations simply **adding** their angles.

Inverse of rotation by Θ is: **rotation by -Θ**

Scaling:



Thus, combining scales simply **multiplies** them.
Inverse of scaling by S is: **scaling by 1/s**

CAUTION: be aware of the complexities that arise when you are combining different kinds of simple transforms.
Example: translation by $\begin{bmatrix}2\\1\end{bmatrix}$, rotate by $30°$, translation by $\begin{bmatrix}-2\\-1\end{bmatrix}$

P = 

Notice that the translations do not undo each other **because there is a rotation (or any other transform) between each other**.

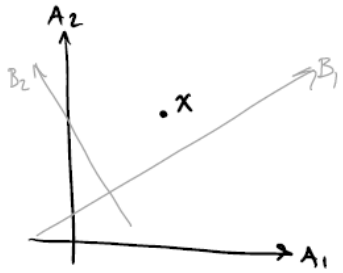
Converting Between Coordinate Systems

Suppose we have a point x and two different coordinate systems (CS) labeled A and B.

x_A = coordinates of x with respect to CS A

x_B = coordinates of x with respect to CS B

e.g.



We want to be able to convert x_A and x_B (and vice versa)

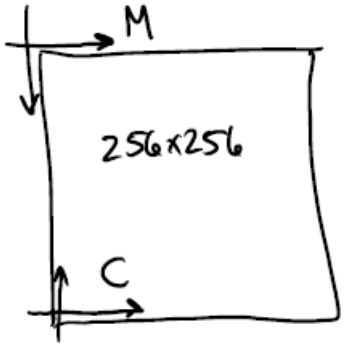
$$P_{A\to B}\, x_A = x_B$$

$$P_{AB}\, x_A = x_B$$

Magic Rule: to get the transform $P_{A->B}$, we can simply use the same transform that moves the B axes onto the A axes using A's coordinate system.

That is, suppose the B axes are an object in A's coordinate space. Compose the transform that will overlay the B axes on the A axes.

Example: Cartesian to MATLAB index coordinates. (50:32)



To convert from x_c (Cartesian) to x_M (MATLAB index), we move M axes onto C axes.

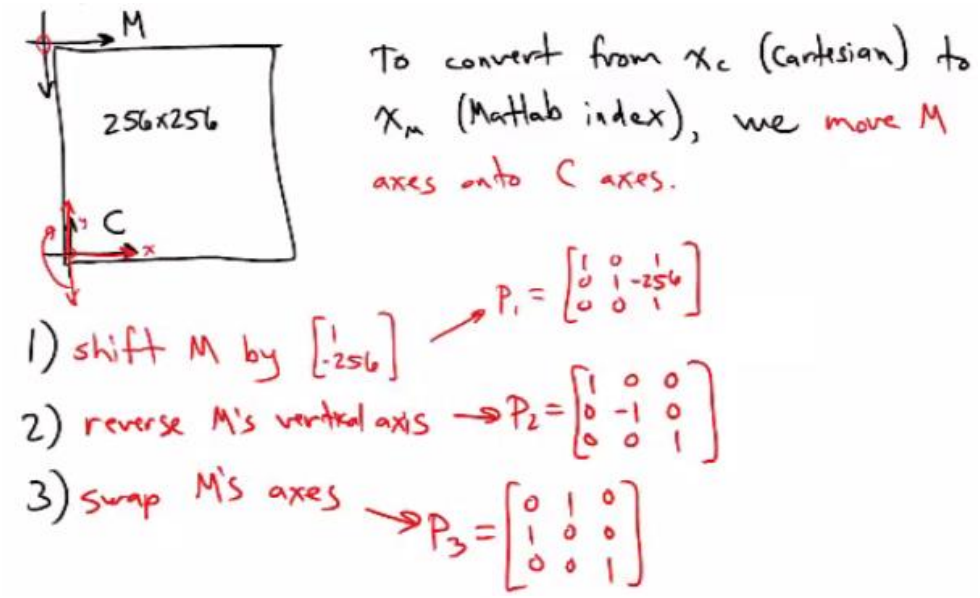
1. shift M by $\begin{bmatrix} 1 \\ -256 \end{bmatrix} \rightarrow P1 = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & -256 \\ 0 & 0 & 1 \end{bmatrix}$ initially 2d,
2. rotate by $90° \rightarrow P2 = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
3. ---

$$P_{C->M} = P_3 P_2 P_1 = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & -256 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 256 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad \begin{bmatrix} 0 & -1 & 256 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 255 \\ 255 \\ 1 \end{bmatrix} (Cartesian)$$

Check: $\begin{bmatrix} 0 \\ 0 \end{bmatrix}_c \rightarrow \begin{bmatrix} 256 \\ 1 \\ 1 \end{bmatrix}$ $\quad\quad$ $\begin{bmatrix} 255 \\ 255 \end{bmatrix}_c \rightarrow \begin{bmatrix} 1 \\ 256 \\ 1 \end{bmatrix}$

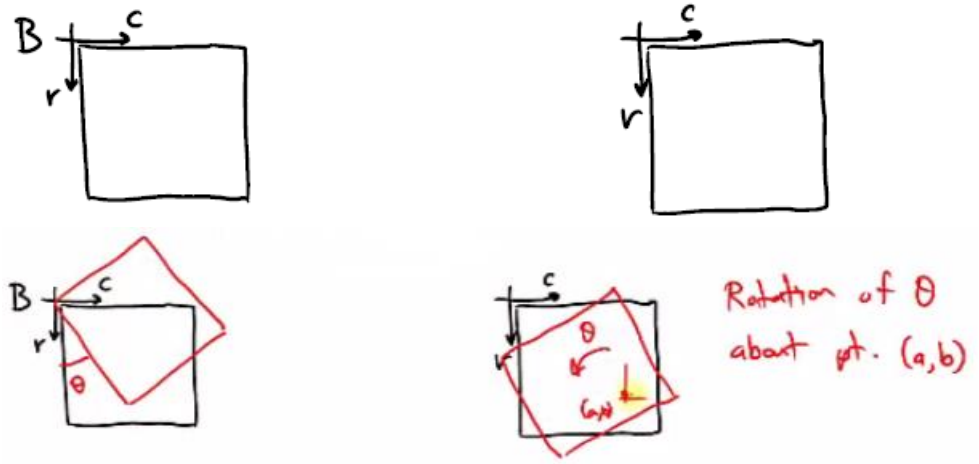
MATLAB -> Cartesian => inverse of $P_{C->M}$

(Alternate)



Converting Transforms between Coordinate Systems

Suppose you want to rotate an image around a specified point with coordinates (a, b). You already know how to build a rotation matrix, but it rotates about the origin.



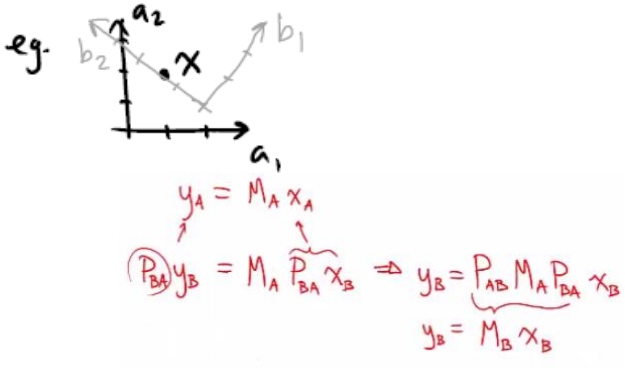
The Theory

Given two coordinate systems A and B, and a transform matrix M_A, expressed with respect to CS A, find the equivalent transform matrix with respect to CS B, M_B.

Let P_{AB} be the change-of-basis matrix that converts from A coordinates to B coordinates.

i.e. $x_B = P_{AB}x_A$

$$b1 = \begin{bmatrix}0\\0\end{bmatrix}$$
$$b2 = \begin{bmatrix}0\\0\end{bmatrix}$$



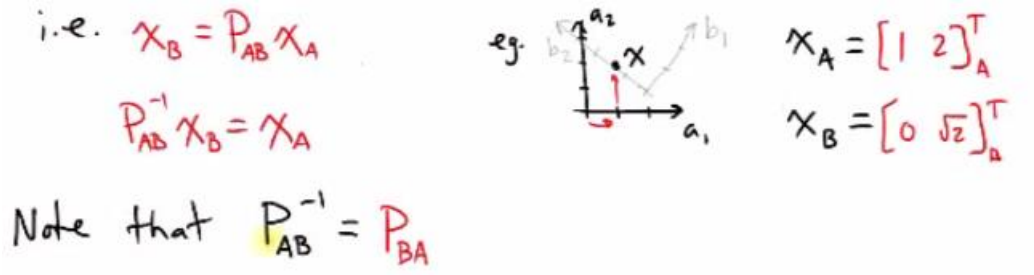
Note that $P_{AB}^{-1} = P_{BA}$

Thus, $\boxed{x_B = P_{AB}x_A}$ and $\boxed{x_A = P_{BA}x_B}$.

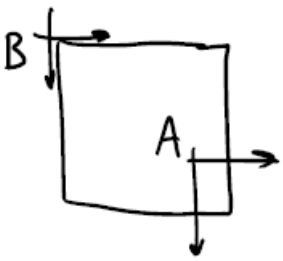
The transform M_A is applied to vectors in CS A.

i.e. $y_A = M_A x_A$

$$P_{BA}y_B = M_A P_{BA} x_B$$
$$y_B = P_{BA}^{-1} M_A P_{BA} x_B$$



Hence, $M_B = P_{AB} M_A P_{BA} = P_{BA}^{-1} M_A P_{BA}$

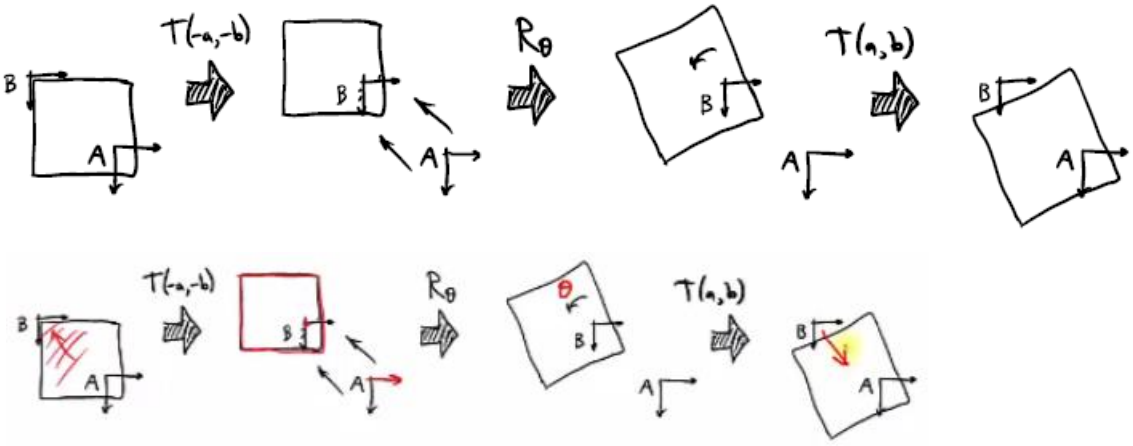


Back to our rotation example:

$M_A = R_\Theta$ (rotation about A's origin)

P_{AB} = T(a,b) Translation by (a,b)

Hence, M_B = T(a,b) R_Θ T(-a,-b)



**Personal note: notes/pictures are doubled – duplicated by hand then copied pictures/notes from video

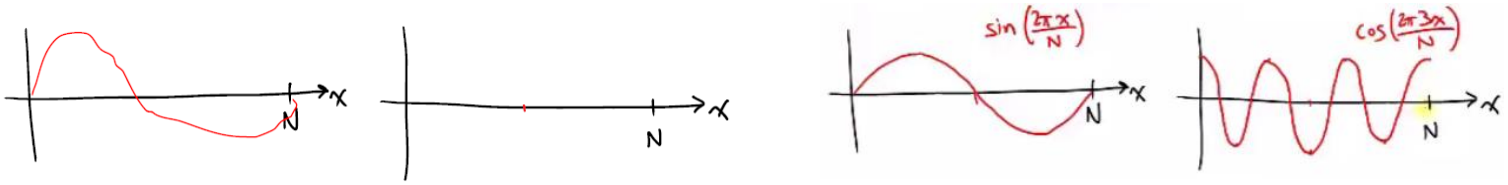
Lesson 04 – Fourier Series

Fourier Series

Goal: to introduce the Fourier transform and review complex numbers. The Fourier transform is a fundamental not only to image processing, but it also plays a special role in medical imaging.

Consider the trigonometric functions of x, $\sin\left(\frac{2\pi kx}{N}\right)$ and $\cos\left(\frac{2\pi kx}{N}\right)$ $\quad k, N \in Z$

They repeat when x increases by $\frac{N}{K}$, or they repeat **k times $0 \leq x \leq N$**



Theorem: suppose f is a "nice" N-periodic function. There exists coefficients a_k and b_k such that

$$f(x) = a_0 + \sum_{k=1}^{\infty}\left[a_k \cos\left(\frac{2\pi kx}{N}\right) + b_k \sin\left(\frac{2\pi kx}{N}\right)\right]$$

This is known as a Fourier Series.

In practice, we approximate f with a truncated Fourier Series,

$$f(x) = a_0 + \sum_{k=1}^{m}\left[a_k \cos\left(\frac{2\pi kx}{N}\right) + b_k \sin\left(\frac{2\pi kx}{N}\right)\right] \qquad (*)$$

f: R -> R

Instead of treating the a's and b's separately, we can use the more sophisticated and compact complex notation,

$$f(x) = \sum_{k=-m}^{m} c_k\left(\cos\left(\frac{2\pi kx}{N}\right) + i\sin\left(\frac{2\pi kx}{N}\right)\right)$$

f: R-> C

Notice the sum is now from –m to m. Here's why. If $f(x) \in R$, then we need to make sure all the imaginary parts cancel out.

$$f(x) = a_0 + \sum_{k=1}^{m}\left[c_k \cos\frac{2\pi kx}{N} + ic_k \sin\frac{2\pi kx}{N} + c_{-k}\cos\frac{-2\pi kx}{N} + ic_{-k}\sin\frac{-2\pi kx}{N}\right]$$

$$a_0 + \sum_{k=1}^{m}\left[(c_k + c_{-k})\cos\frac{2\pi kx}{N} + i(c_k - c_{-k})\sin\frac{2\pi kx}{N}\right]$$

Comparing to **(*)** above:

$$\begin{cases} a_k = c_k + c_{-k} & ① \\ b_k = i(c_k - c_{-k}) & ② \end{cases}$$

(1) + i(2) => $a_k + ib_k = 2c_{-k} => c_{-k} = \frac{a_k + ib_k}{2}$

(1) – i(2) => $a_k - ib_k = 2c_k => c_k = \frac{a_k - ib_k}{2}$

Notice, then, that $c_k = \bar{c}_{-k}$ (complex conjugates)

We have 2m terms in truncated Fourier Series.

- In ak, bk relations are real valued if f is real(?)

Conjugate symmetry – if function you're computing the FT for is real-valued, then the coefficients exhibit conjugate symmetry.

Lesson 05 – Fourier Transform

Fourier Transform

Goal: to introduce some of the basic methods for the Fourier transform.

Fourier Transform (continuous domain)

Let $f: R \to R$ (i.e. f(x))

Its Fourier transform (FT) is defined as:

$$F(\omega) = \mathcal{F}\{f(x)\}(\omega) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i \omega x} dx \leftarrow \textit{Fourier basis function; } \omega \textit{ to specify}$$

(F: Fourier coefficients; ω, λ for frequency; x, y spatial variables; f,g spatial signals; script F for functional notation for FT)

- Like an inner product
  - f signal, and complex exponentials => how much of sin, cos of given frequency do we find in F?

F(w) is a frequency decomposition (tells you how much of each frequency is present in F)… a different way of representing the same signal f.

The FT is invertible.

$$f(x) = \mathcal{F}^{-1}\{F(\omega)\}(x) = \int_{-\infty}^{\infty} F(\omega) e^{2\pi i \omega x} d\omega$$

Discrete Fourier Transform (DFT)

When both f(x) and F(w) are sampled,

$$\text{i.e. } f_n, n = 0, \ldots, N-1 \Rightarrow [f_0 \; f_1 \; f_2 \; \ldots \; f_{N-1}]$$

$$F_k, k = 0, \ldots, N-1 \Rightarrow [F_0 \; F_1 \; F_2 \; \ldots \; F_{N-1}]$$

Will use m, n for spatial indices; k, l for frequency

DFT maps between these two spaces

Then we have the Discrete Fourier Transform (DFT) and its inverse:

$$F_k = \sum_{n=0}^{N-1} f_n e^{\frac{-2\pi i n k}{N}}, k = 0, \ldots, N-1$$

$$f_n = \frac{1}{N} \sum_{k=0}^{N-1} F_k e^{\frac{2\pi i n k}{N}}, n = 0, \ldots, N-1$$

1/N because all of the sums. Also, $c_k \to F_k$

Fast Fourier Transform (FFT)

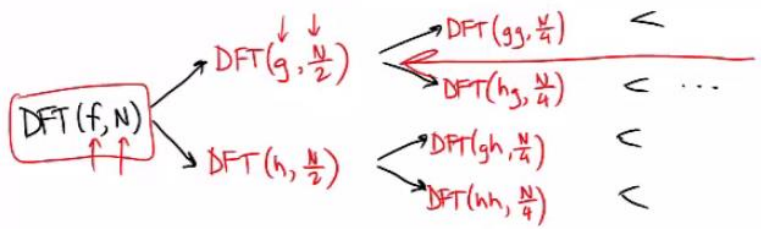
Computing all N Fourier coefficients directly using the formula for F_k above would take **$O(N^2)$** flops.

A divide-and-conquer method turns out to be faster. It's called the Fast Fourier Transform.

Briefly, it decomposes the length-N DFT into two N/2-length DFTs. This process recursively decomposes the DFT until it arrives at arrays of length 1… those are easy. The whole process take **O(NlogN)** flops for a 1D array of length N.

(what about for 2D or 3D?) **NxN -> $O(N^2 \log N)$**

Given f_n, n = 0, …, N – 1, we recombine the elements to get g_n and h_n, n = 0,…,N/2 – 1

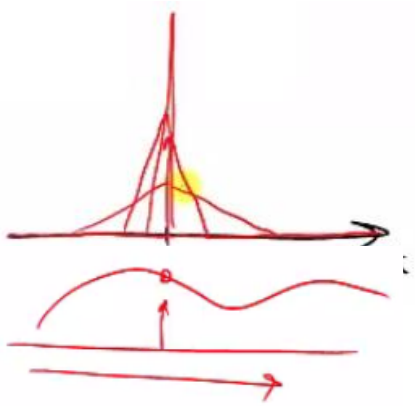


Definition: Dirac delta function

$$\delta(x) = \begin{cases} \infty & if\ x = 0 \\ 0 & otherwise \end{cases}$$

$$s.t. \int_{-\infty}^{\infty} \delta(x)\, dx = 1$$

$$and \int_{-\infty}^{\infty} \delta(x-c) f(x)\, dx = f(c)$$



- in integral, think of triangle area = 1, make thinner, but limit still = 1
- at limit = 0, infinity
- 2nd integral: shifted

Theorem: $\int_{-\infty}^{\infty} e^{2\pi i \omega x}\, dx = \delta(\omega)$

Proof: not in this course.

We can use the Dirac delta function to prove that $\mathscr{F}$ and $\mathscr{F}^{-1}$ are inverses.

Proof: (19:00)

$\mathcal{F}^{-1}\{\mathcal{F}\{f(x)\}(\omega)\}(s)$ = f(s)

$$= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x) e^{-2\pi i \omega x} dx\, e^{2\pi i \omega s} d\omega$$

$$= \int_{-\infty}^{\infty}\left[\int_{-\infty}^{\infty} f(x) e^{2\pi i \omega (s-x)} d\omega\right] dx$$

$$= \int_{-\infty}^{\infty} f(x)\delta(x-c)\, dx = f(s) \quad \blacksquare$$

2D Fourier Transform

Consider the function f: R^2 -> R

The 2D FT is defined as: (ω, λ) two frequency variables

$$F(\omega, \lambda) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y) e^{-2\pi i(\omega x + \lambda y)} dx\, dy$$

$$\uparrow (\omega, \lambda) \cdot (x, y)$$

Notice that the FT is ~~separable~~ independent:

$$F(\omega, \lambda) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y) e^{-2\pi i \omega x} dx\, e^{-2\pi i \lambda y} dy$$

$$= \int_{-\infty}^{\infty}\underbrace{\left[\int_{-\infty}^{\infty} f(x,y) e^{-2\pi i \omega x} dx\right]}_{Apply\ FT\ along\ x-dim} e^{-2\pi i \lambda y} dy$$

Apply FT along y-dim

Thus, and N-D FT can be done using 1D FTs along each of the N dimensions.

Lesson 06 – Convolution

Convolution

Goal: to define convolution, see what it does, and find out how it can be done using the Fourier transform.

Definition: the convolution between two functions f(x) and g(x) is an integral of the form

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

For discrete signals f_n, g_n,

$$(f * g)_n = \sum_{k=0}^{N-1} f_k \, g_{n-k} \qquad n = 0, \ldots, N-1$$

Theorem: (continuous-domain version)

Let f(x) and g(x) be functions, and let

$$F(\omega) = \mathcal{F}\{f(x)\}(\omega) \quad \text{and} \quad G(\omega) = \mathcal{F}\{g(x)\}(\omega)$$

Be their Fourier transforms. Then

$$\mathcal{F}\{(f * g)(x)\}(\omega) = \mathcal{F}\{f(x)\}(\omega)\, \mathcal{F}\{g(x)\}(\omega)$$
$$= F(\omega)\, G(\omega)$$

Proof:

$$\mathcal{F}\{(f*g)(x)\}(\omega) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(\tau) g(x - \tau) d\tau \, e^{-2\pi i \omega x} dx$$

Change of variables: let y = x - τ => x = y + τ

dx = dy

$$= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(\tau) g(y) d\tau \, e^{-2\pi i \omega (y+\tau)} dy$$

$$= \int_{-\infty}^{\infty} f(\tau) e^{-2\pi i \omega \tau} d\tau \int_{-\infty}^{\infty} g(y) e^{-2\pi i \omega y} dy$$

separate the τ's from the y's

$$= F(\omega)\, G(\omega)$$

QED

Note: one can just as easily prove the theorem.

$$\mathcal{F}^{-1}\{(F*G)(\omega)\}(x) = f(x)\, g(x)$$

convolution in frequency ⇔ convolution in spatial

Therefore, convolution in one domain is equivalent to element-wise multiplication in the other domain.

$\mathcal{F}\{f*g\}(\omega) = F(\omega)\, G(\omega)$

conv. in spatial domain ⇔ mult. in freq. domain

$\mathcal{F}^{-1}\{F*G\}(x) = f(x)\, g(x)$

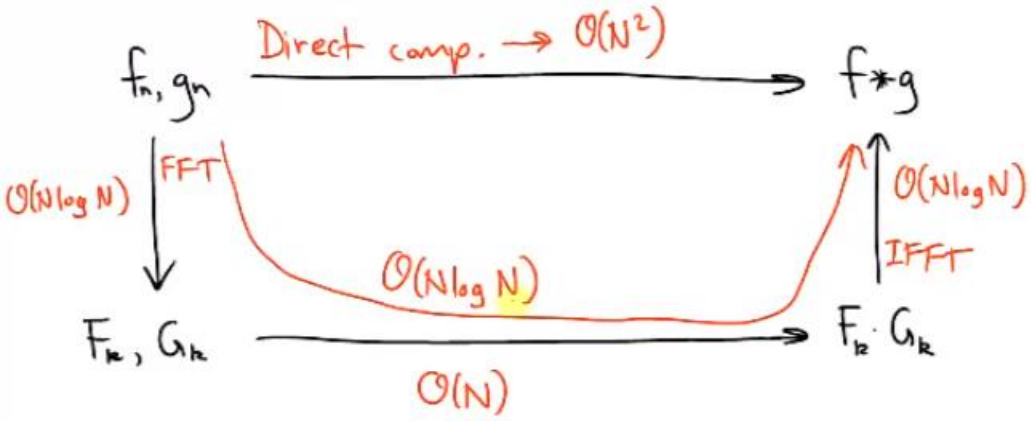
conv. in freq. domain ⇔ mult. in spatial domain

Theorem: (discrete-domain version)
Let f_n and g_n, n = 0, …, N – 1, be two discrete functions (signals).

$$\mathcal{F}\{(f*g)_n\}_k = F_k G_k$$

Question: why bother using the FT to compute convolution? Sure, it may be cool, but isn't it more work?

Consider the number of flops (floating-point operations) to compute (f*g) using the two methods.



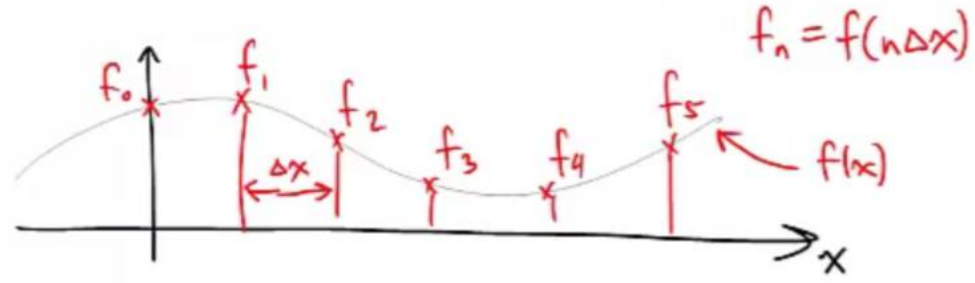
Thus, as N gets large, using the DFT is more efficient.
What about convolving NxN images, f_{mn} and g_{mn}, where m = 0, …, N – 1 and n = 0, …, N – 1?

$\rightarrow O(N^2 log N)\ flops \qquad (opposed\ to\ O(N^4))$

Lesson 07 – Sampling Theory

Sampling Theory

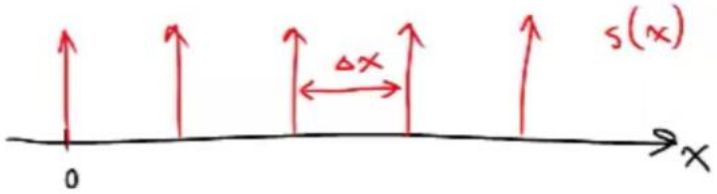
Goal: images can be thought of as sampled versions of 2D functions. We want to develop a framework for understanding the relationship between a function and its samples.



Consider a continuous-domain function, $\boldsymbol{f}: \boldsymbol{R} \to \boldsymbol{R}$, and samples of it, $\boldsymbol{f_n}: \boldsymbol{Z} \to \boldsymbol{R}$

To sample f, we multiply by the Shah "comb" function.

$$s(x) = \sum_{n \in Z} \delta(x)$$



Then, the sampled version of f can be written

$$\bar{f}(x) = f(x)\, s(x) = f(x) \sum_{n \in \mathbb{Z}} \delta(x - n\Delta x)$$

Consider the FT of f(x).

$$\mathcal{F}\{\bar{f}(x)\}(\omega) = \mathcal{F}\{s(x)\, f(x)\}(\omega) = (S * F)(\omega)$$

What is S(w)?

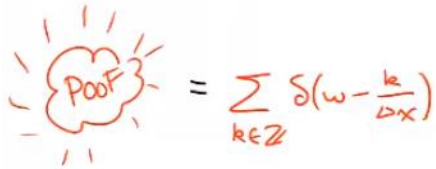
$$S(\omega) = \mathcal{F}\{s(x)\}(\omega)$$

$$= \int_{-\infty}^{\infty} s(x) e^{-2\pi i \omega x} dx$$

$$= \int_{-\infty}^{\infty} \sum_{n} \delta(x - n\Delta x)\, e^{-2\pi i \omega x} dx$$

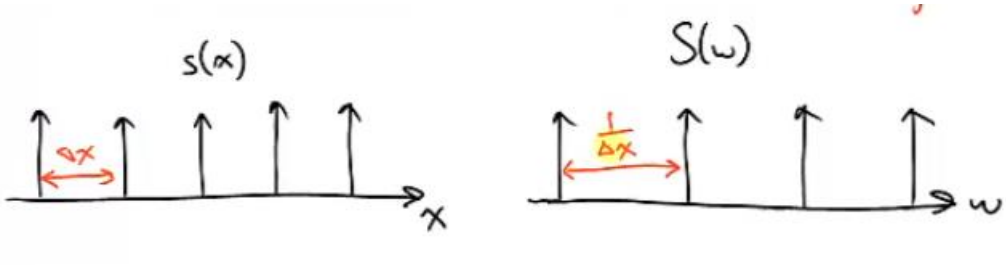
$$= \sum_{n} \int_{-\infty}^{\infty} \delta(x - n\Delta x)\, e^{-2\pi i \omega x} dx \quad \left(after\ swapping\ \Sigma\ and\ \int\ \right)$$

$$= \sum_{n \in Z} \underbrace{e^{-2\pi i \boxed{\omega(n\Delta x)}}}\, dx \qquad = 1\ when\ \omega\Delta x = k \in Z \Rightarrow w = \frac{k}{\Delta x}, k \in Z$$

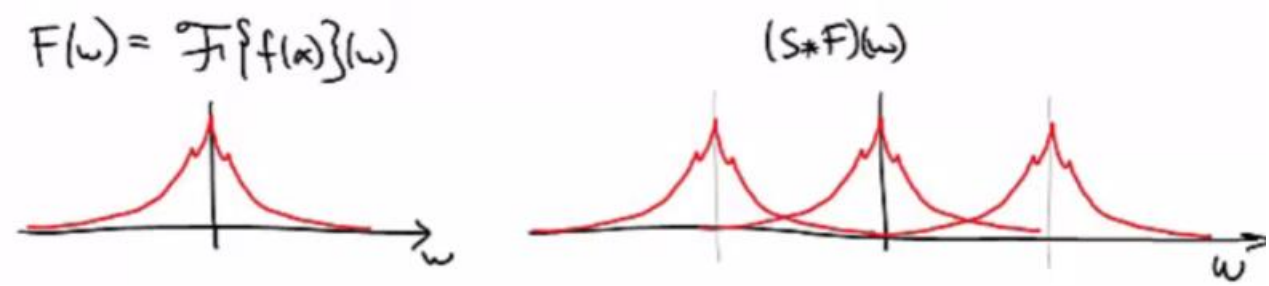


$$= \sum_{k \in \mathbb{Z}} \delta\left(\omega - \frac{k}{\Delta x}\right)$$

Thus, the FT of the Shah function is **also a Shah function, but with different spacing**.

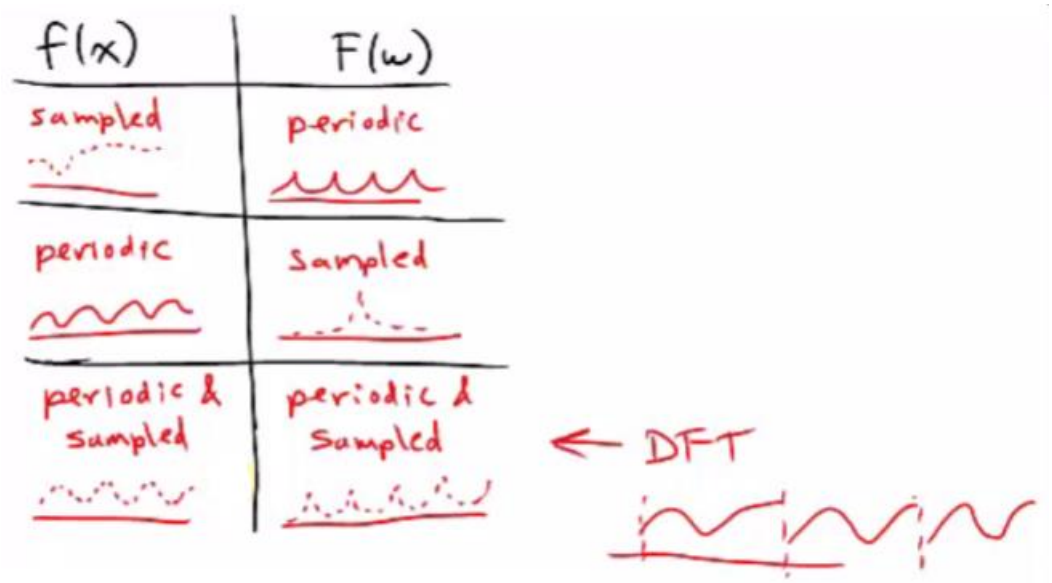


Back to $\mathcal{F}\{f(x)\,s(x)\}(\omega) = (S * F)(\omega)$



So, sampling f gives us **a periodic FT**.

Likewise, a similar derivation can be used to show that a periodic f yields **a discrete FT**.



Lesson 08 – Aliasing

Goal: to observe and understand the issues when sampling an image

Demo: Moiré video

What's going on here?

Moiré effect

Suppose we have this signal:

Suppose we sub-sample it by a factor of 3:

↑lots of overlap

The overlap between the copies of the Fourier repetitions is called aliasing. The coefficients are added and can't be separated to get the correct Fourier Transform.

Nyquist-Shannon Sampling Theorem

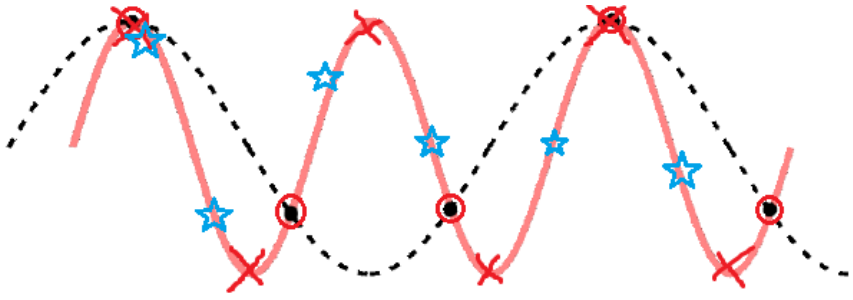
To avoid aliasing, your sampling frequency must be at least double the highest frequency in your signal.

e.g. sample @ 10 samples/cm, spatial sampling spacing of 0.5 mm

if you want to sample @ 10 cycles/mm, need spatial sample spacing of at least 0.5 mm apart

circle: different trig function that passes through those points

sample accurately: double the highest frequency

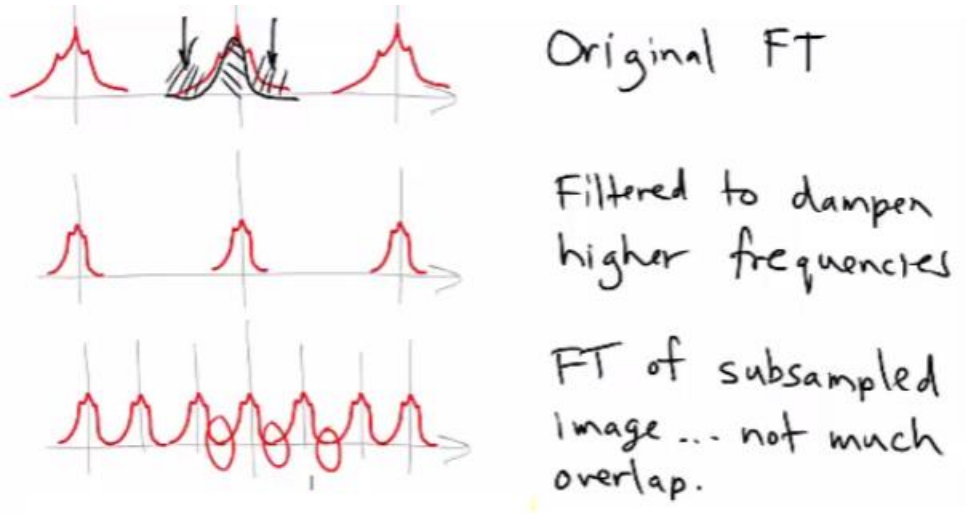


Demo: vertical stripes being produced – then sampled every 5

- Original: fairly high frequency; every fifth sample: black bars – looks like lower frequency
- Downsampling to subsample – artifacts will appear => aliasing

Filtering

If we want to subsample an image, but avoid Moiré artifacts, we can filter the image first. Filtering is multiplying the Fourier coefficients to adjust the relative contributions of the different frequencies. In our case, we need to dampen the higher frequencies so they don't overlap so much.



Filter: multiply by other Fourier coefficients – things get scaled down outside region

- Middle region stays big

Information is lost, but better to lose them than have them be misinterpreted

- Adjusting the frequency content in order to aliasing (in this context)

L08_Aliasing.m

- Original: bunch of bricks – high frequency pattern
- Subsampled: Moiré effect – artifact patterns
    - High-frequencies being misinterpreted, turn to low frequency

GaussianBlur: takes image and number – multiplies frequency domain (Fourier coefficients by a bell curve); e.g. in diagram above

- Has far fewer pixels then above
- Lot of high frequency info lost
- But pattern is lessened – aliasing is lessened
- Input 3: more information lost – bricks “disappear”

<u>Strobe Effect</u>

(A related phenomenon)

Lesson 09 – Properties of the FT

Goal: to survey some of the properties of the Fourier transform that are useful in image processing (beyond the convolution theorem).

Demo: L09_wavefront.m

$$\mathcal{F}\{\beta f(x)\}(\omega) = \beta \mathcal{F}\{f(x)\}(\omega)$$

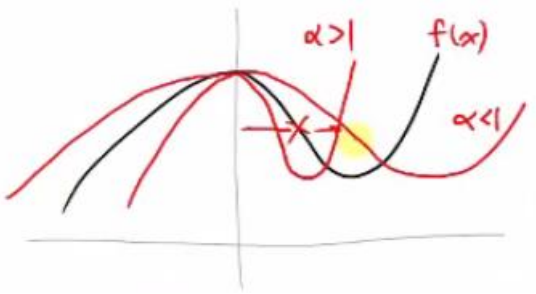
Scale Property

What happens to F(w) if we scale f?

Let $\bar{f}(x) = f(\alpha x)$

$$\bar{F}(w) = \int_{-\infty}^{\infty} \bar{f}(x)\, e^{-2\pi\omega x} dx$$

$$= \int_{-\infty}^{\infty} f(\alpha x)\, e^{-2\pi\omega x} dx$$



Change of variables: y = αx => dy = αdx => dy/α = dx

Note: lots of Fourier proofs involve a change in variables.

$$\bar{F}(w) = \frac{1}{\alpha}\int_{-\infty}^{\infty} f(y)\, e^{-2\pi\frac{\omega y}{\alpha}} dy = \frac{1}{\alpha} F\left(\frac{\omega}{\alpha}\right)$$

Thus, $\mathcal{F}\{f(\alpha x)\}(\omega) = \frac{1}{\alpha}\mathcal{F}\{f(x)\}\left(\frac{\omega}{\alpha}\right)$

So, as f(x) stretches, F(w) contracts and shrinks (and vice versa).

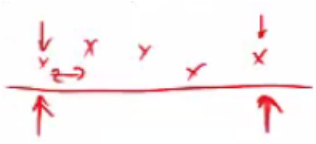
Example: consider the Shah function with spacing Δx, s(x). Recall the spacing of S(w) = $\mathcal{F}${s(x)}(w)

Exercise: what is the spacing of s(αx), α≠0? (17:06)

What is the spacing of S(αx)?

There is also a scale property for the DFT, though the interpretation is different. Suppose f_n and F_k are each an array of N numbers. What do those numbers mean? What points in the spatial/frequency domain do those values correspond to?

$$F_k = \sum_{n=0}^{N-1} f_n e^{-2\pi i \frac{nk}{N}}$$



Look at nk/N, when it goes from 0 to 1

One period… $0 \le \frac{nk}{N} < N$ (*)

Notice we can introduce an arbitrary spatial scale, $0 \le \frac{nk}{N} < 1 \Rightarrow 0 \le \frac{nL}{N} < L$

$0 \le \_\_\_ < 2\ km$ $\Rightarrow 0 \le x < L$

We compensate by scaling the frequency variable,

(*) => $0 \le \frac{nL}{N}\frac{k}{L} < N$

i.e. x = $\frac{nL}{N}, for\ n = 0, \dots, N-1$

$\omega = \frac{k}{L}, for\ k = 0, \dots, N-1$

Thus, if $x \in \{\frac{0L}{N}, \frac{L}{N}, \frac{2L}{N}, \frac{3L}{N}, \dots, \frac{(N-1)L}{N}\}$ upper limit of last term is L, "field of view" FOV or period

Then $\omega \in \{\frac{0}{L}, \frac{1}{L}, \frac{2}{L}, \frac{3}{L}, \dots, \frac{N-1}{L}\}$

In general, $(FOV_x)(\Delta\omega)(FOV_\omega)(\Delta x) = 1$

$(FOV_x)(\Delta\omega) \to 1$ $\quad$ $(FOV_\omega)(\Delta x) \to 1$

$(L)\left(\frac{1}{L}\right) = 1$ $\quad$ $\left(\frac{N}{L}\right)\left(\frac{L}{N}\right) = 1$

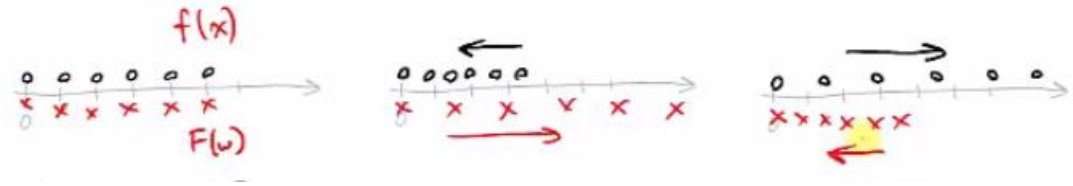
Equivalently, we could use

$$x \in \{\frac{0}{L}, \frac{1}{L}, \frac{2}{L}, \dots, \frac{N-1}{L}\}$$

$$\omega \in \{\frac{0L}{N}, \frac{L}{N}, \frac{2L}{N}, \dots, \frac{(N-1)L}{N}\}$$

Exercise: verify that this sampling obeys the discrete scaling rule.

Visual examples: For 6-vectors (N = 6)…



2nd: contract top, expand bottom

3rd: top - increase sample space in spatial domain, decrease sample space in frequency domain

⇨ Inverse relationship

Side-Effect: scaling using padding and cropping, and the DFT

Consider this situation

- Pad f (add samples), then apply DFT
  Where will the samples be in the frequency domain?

In other words, padding in one domain can be thought of as **supersampling** in the other

Demo: image scaling using the DFT L09_Scaling

Shift Property

Consider f_n and its DFT F_k, n, k = 0, …, N – 1. Let $g_n = f_{n-d}$ (a shifted version of f_n)

$$G_k = \sum_{n=0}^{N-1} g_n e^{-2\pi i \frac{nk}{N}}, \quad k = 0, \dots, N-1$$

$$= \sum_{n=0}^{N-1} f_{n-d} e^{-2\pi i \frac{nk}{N}}, \quad k = 0, \dots, N-1 \; change\ of\ vars:$$

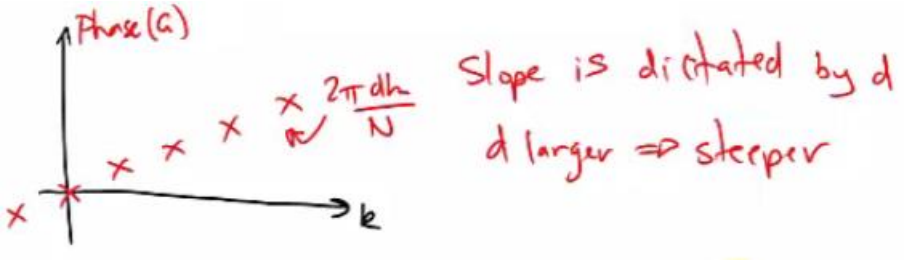
Let m = n – d, n = m + d

$$= \sum_{m=-d}^{N-1-d} f_m e^{-2\pi i \frac{(m+d)k}{N}} \leftarrow sinusoidal; periodic$$

$$= \boxed{\sum_{m=0}^{N-1} f_m e^{-2\pi i \frac{mk}{N}}} e^{-2\pi i \frac{dk}{N}}$$

$$= e^{-2\pi i \frac{dk}{N}} F_k \quad \to \frac{2\pi dk}{N}$$

That is, shifting f by d samples is equivalent to multiplying F by a "phase ramp"



"Phase" because it only influences the phase of the Fourier coefficients.
This property is also used quite often in MRI.
Demo: Fourier Shift – L09_Fourier_shift.m – **MORE NOTES**

Rotational Invariance

If R is a rotation matrix, then $\bar{f}(x) \equiv f(R\vec{x})$ is a rotated version of f. What does $\mathcal{F}\{\bar{f}(\vec{x})\}$(w) look like?

As it turns out $\mathcal{F}\{f(R\vec{x})\}(\omega) = \boldsymbol{F(R\vec{\omega})}$

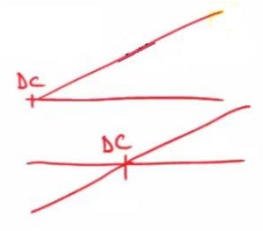
We won't prove this here, but it's quite simple.

Thus, rotating f is equivalent to **rotating F by the same amount**.

This works for the FT, and also for the DFT with a few minor caveats.

Demo: Fourier rotation – L09_Fourier_rotation.m – **MORE NOTES**

- Resampling artifact in rotation
- Sampling best when thing is smooth

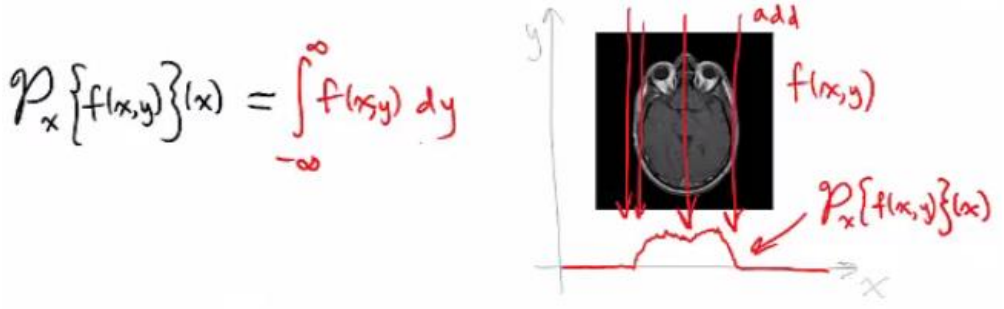


Fourier Projection Theorem

(important for CT (especially), MRI reconstruction)

Suppose you have $f: R^2 \to C$, and $F(\omega, \lambda) = \mathcal{F}\{f(x,y)\}(w,\lambda)$. (Cartesian x, y)

Consider the Radon projection of onto the x-axis, (we use MATLAB coords)



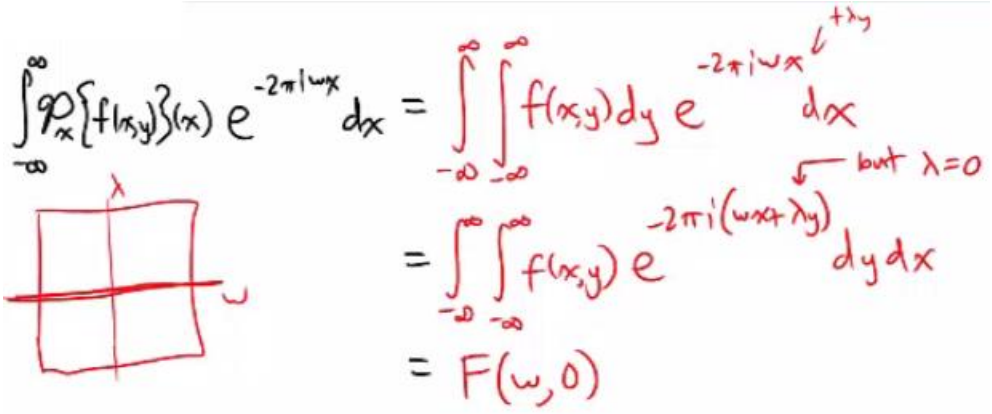
$$P_x\{f(x,y)\}(x) = \int_{-\infty}^{\infty} f(x,y)dy$$

For no apparent reason, let's take the 1D FT of P_X.

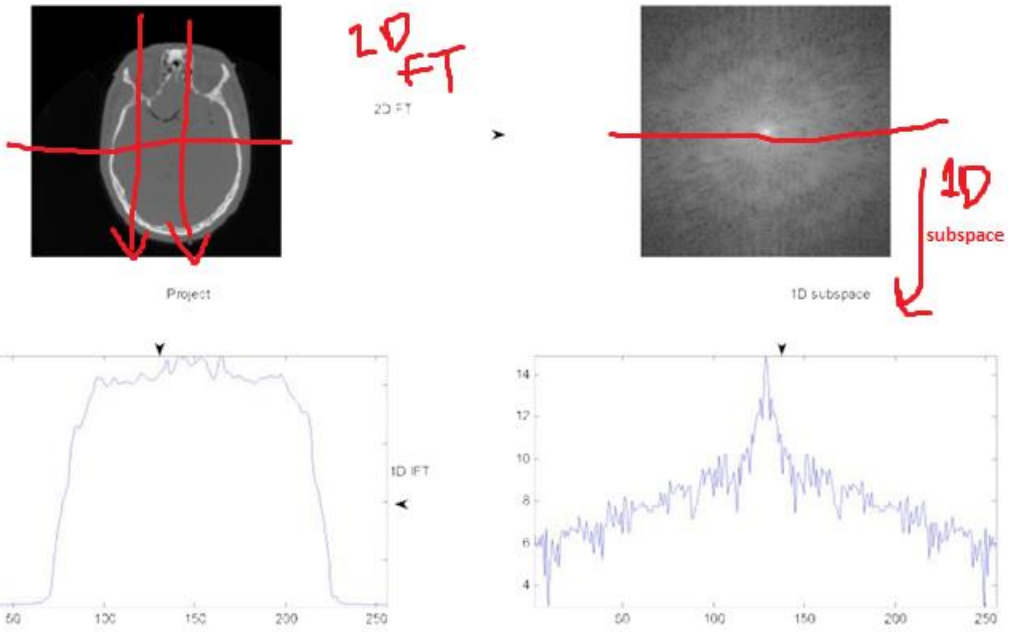
$$\int_{-\infty}^{\infty} P_x\{f(x,y)\}(x)\, e^{-2\pi\omega x} dx = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)dy\, e^{-2\pi i\omega x} dx \leftarrow +\lambda y \ (in\ exponent)$$

$$= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)dy\, e^{-2\pi i(\omega x+\lambda y)} dydx \leftarrow (but\ \lambda = 0)$$

$$= F(\omega, 0)$$



Thus, the 1D-FT of the projection is the same as the 1D line of Fourier coefficients taken from the 2D-FT.
In a picture…



This works for a projection along any direction, as long as the subspace in the frequency domain has the same orientation.

Lesson 10 – Interpolation and Resampling

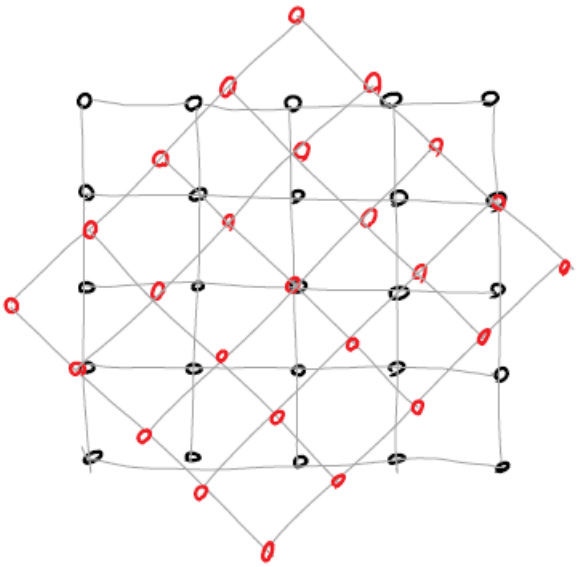
Goal: to learn when and how to resample an image.

Resampling

Suppose we want to rotate an image by 45°.

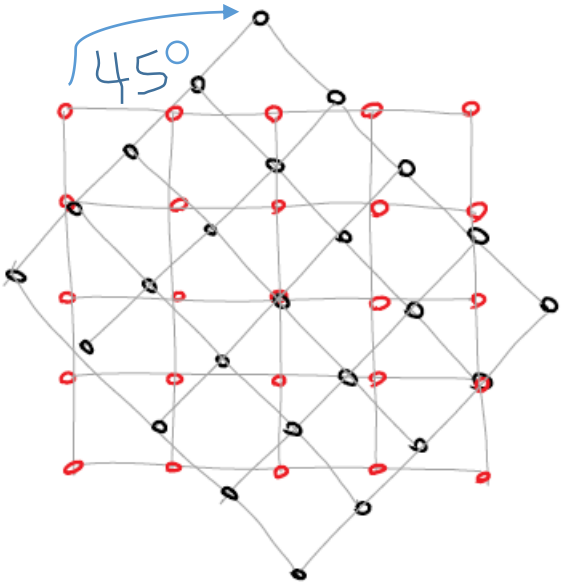
(black: pre-transform pixels, red: post-transform)

Method 1: for each pre-transform pixel, rotate it 45° to its post-transform location.



Problem: **pixels tend to land between post-transform grid elements, not on top of existing pixels – not nicely on the grid**

Method 2: for each post-transform pixel, counter-rotate it 45° to its pre-transform location.



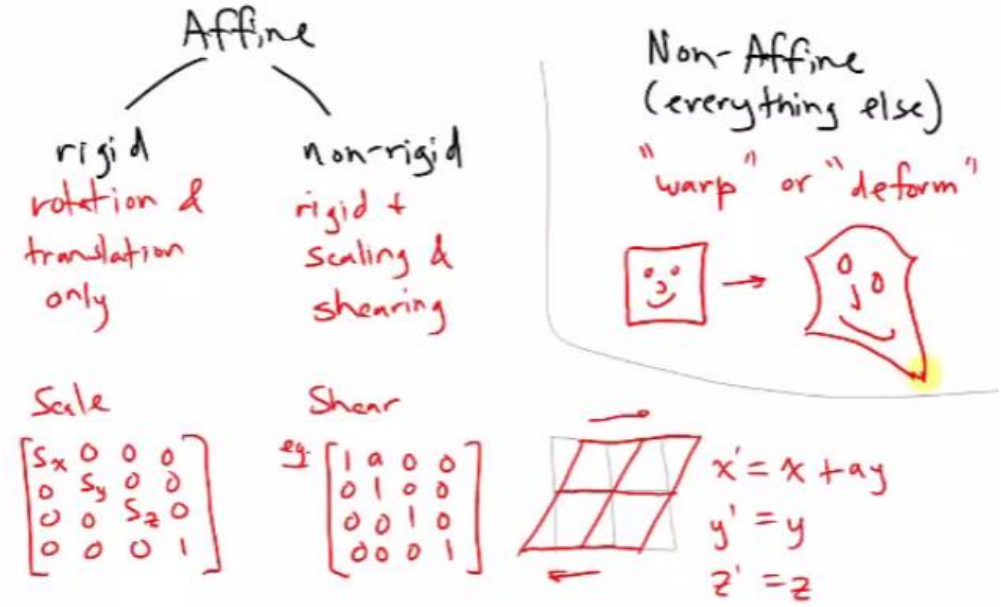
Problem: **pixels tend to come from locations between pre-transform grid elements**

The vast majority use Method 2. The reason becomes more obvious when you consider non-affine transformations.

Pseudocode: to rotate an image f by 45°

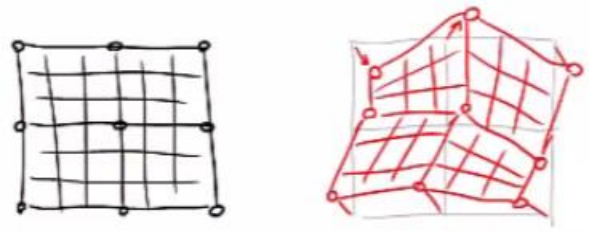
```
for each (r, c) in g (post-transform image)
   rotate  (r, c) by -45° -> (r', c')
   sample f at (r', c') -> g(r, c)
next (r, c)
```

Types of Spatial Transforms



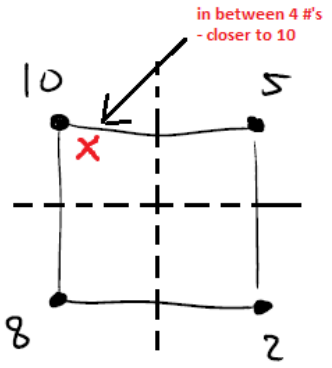
There are many ways to represent non-affine (a.k.a. “non-linear”) transformations.
Ultimately, one needs to define a displacement map… a set of vectors defining how each pixel moved. To implement it, we need to know the inverse of the deformation. For each post-transform pixel, we need a displacement vector that tells us **where it came from**.
6° of freedom in 3d affine translation (rigid)
12° in non-rigid -> 12° in affine
Non-affine: infinite degrees

Piecewise Linear
The displacement of a grid of control points defines the transform. Between those control points, we use a linear transformation. (9 control points)



Basis Functions
Use a linear combination of mathematical functions to define the x- and y- displacements.
e.g. trig, polynomials (splines, etc.), Gaussians, etc.

Interpolation
This is the operation of estimating the value of an image between its pixels.

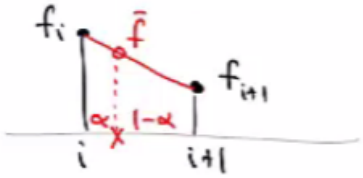


Nearest Neighbour
The simplest method is to round to the nearest sample, and use its value. The problem is that this creates discontinuities in the result.
e.g.

Linear Interpolation

You've heard of this before

i.e.

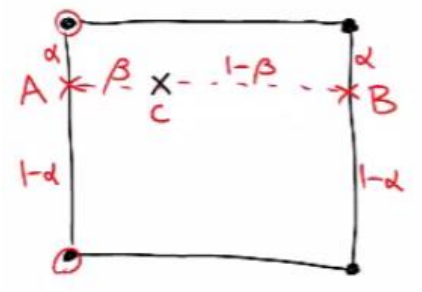


$$y - y_0 = m(x - x_0)$$

$$\bar{f} = \alpha f_{i+1} + (1-\alpha) f_i$$

it's a weighted sum ("convex combination" in fact) of the two nearest values.

What about in 2D, for images?



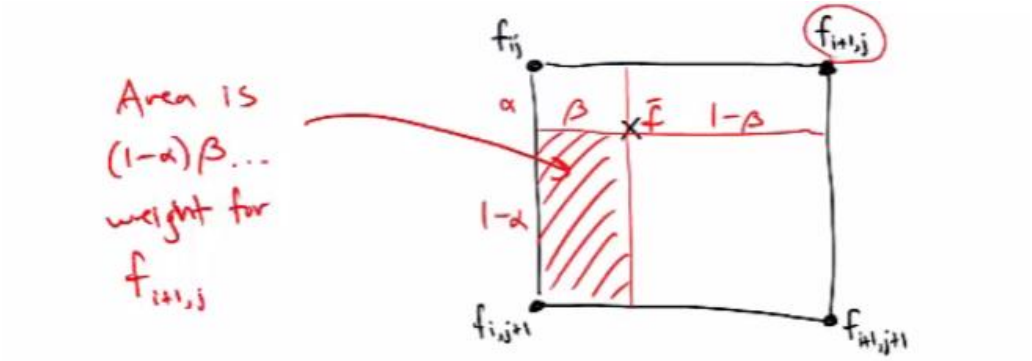
Use 1D linear interpolation along each dimension sequentially

- Compute A & B
- Compute C
- Why not the other way? Horizontal then vertical? Both give same answer

This is called "bilinear interpolation"

In 3D, "trilinear interpolation"

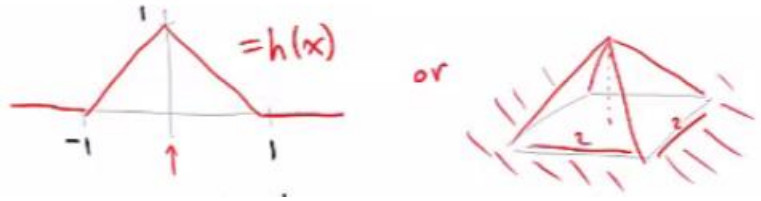
Trick: weight each sample by the area (length, volume) of its opposite region.



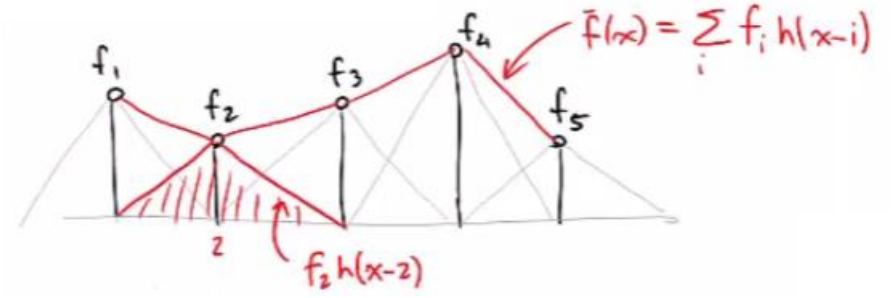
$$\bar{f} = f_{ij}(1-\alpha)(1-\beta) + f_{i+1,j}(1-\alpha)\beta + f_{i,j+1}\alpha(1-\beta) + f_{i+1,j+1}\alpha\beta$$

Convolution

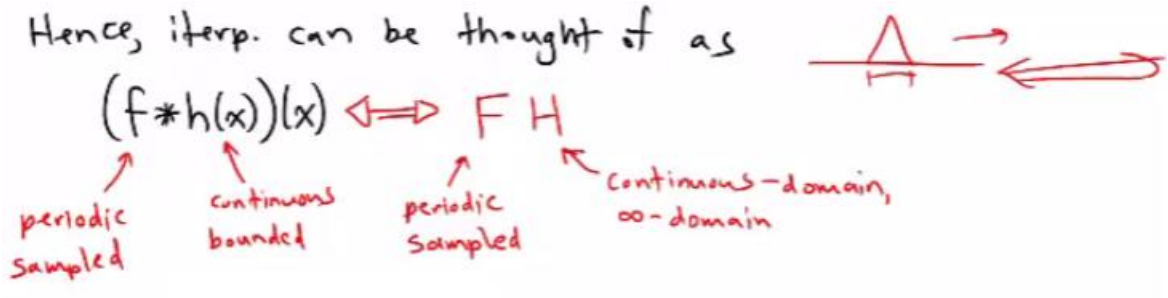
Another way to think of (bi)linear interpolation is as convolving the samples with a continuous-domain interpolation kernel.



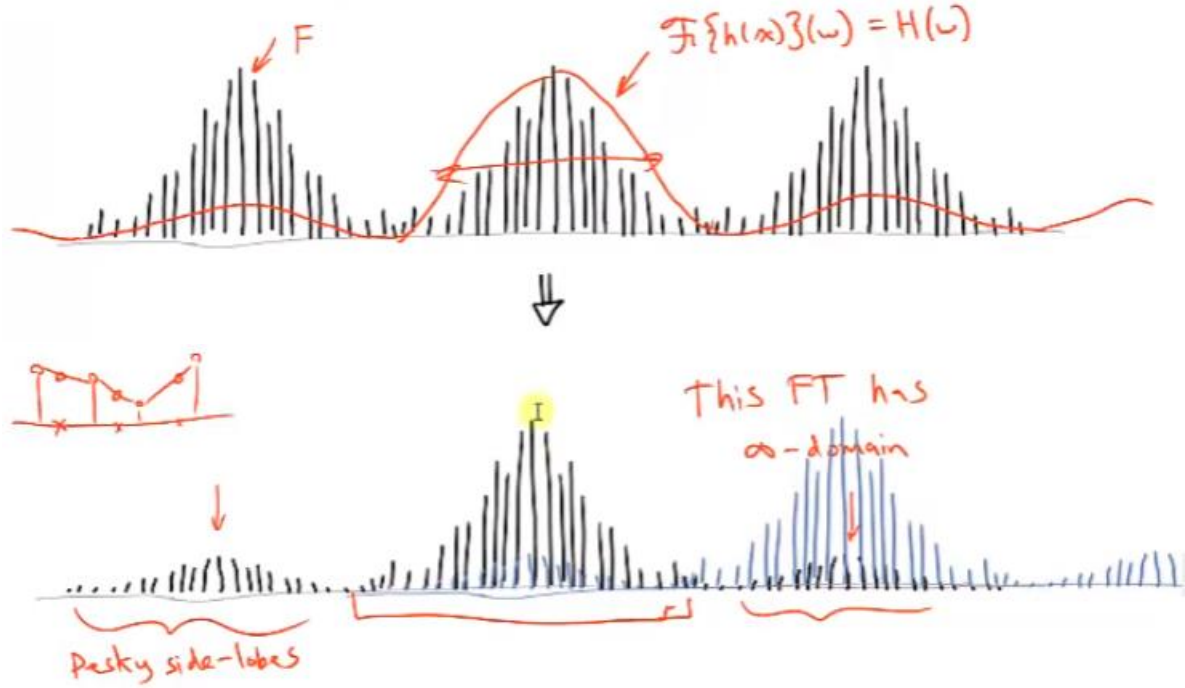
This is equivalent to placing a weighted copy of h(x) at every sample.



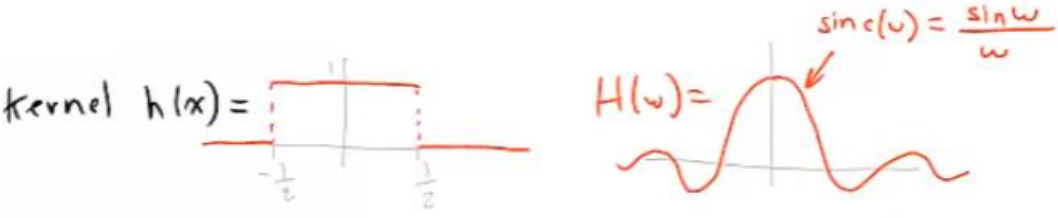
Sampling Theory (revisited)



* info lost when interpolating



But, since we sample (f*h(x)), its FT becomes periodic, and those pesky side-lobes interfere with adjacent copies. This is another example of aliasing and results in **a fundamental loss of information**.
For nearest-neighbour, the side-lobes are even bigger than for linear interpolation.



There are lots of other kernels we can use.
But, there is a tradeoff:

- You want a large kernel because the scaling property says that spatially wide functions have **a tighter FT, and the side lobes will be smaller (accuracy)**
- You want a small kernel so it is **cheaper** to interpolate (**efficiency**)

Unit 2: Medical Imaging Modalities - Lesson 11 – Modalities

There are different kinds of medical images. We refer to each source of medical image as a **modality**.
We will study some in detail (**x-ray, MRI, CT**) while others will be mentioned briefly (**PET, SPECT, VS, MEG**).

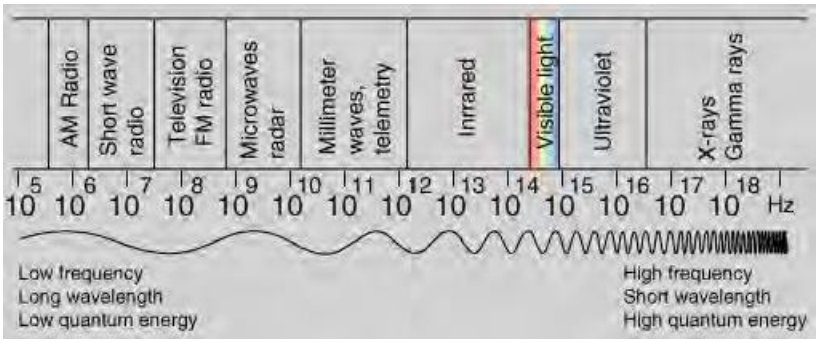
Each modality results from a different phenomenon of physics, and offers doctors (radiologists) a different view of the patient. And each modality has its risks, costs, and benefits. All these factors have to be considered when choosing to image someone

Lesson 12 – X-Ray

Goal: to learn the physics behind x-rays, some of its uses and limitations.
Note: much of the content here can be found in section 2.1 of Alexei Ramotar's MMath thesis http://hdl.handle.net/10012/2914)

What are x-rays?

X-rays are a form of electromagnetic radiation, in the same spectrum as visible light and radio waves.

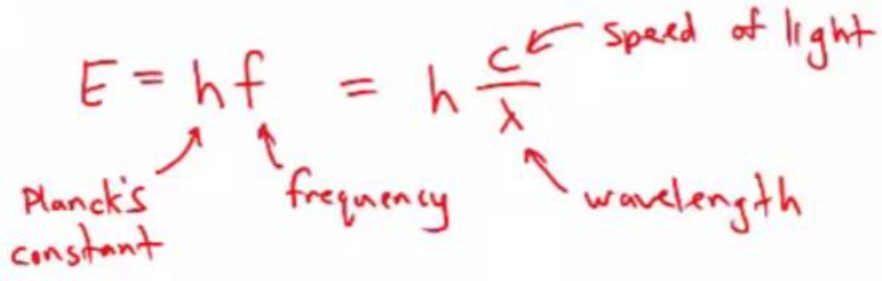


In 1895, Wilhelm Roentgen discovered these rays that could pass through wood and human tissues. He called them "x-rays", the "x" being a place-holder for the unknown.

X-Ray Physics

Like light, x-rays can be thought of as a **particle or wave**.

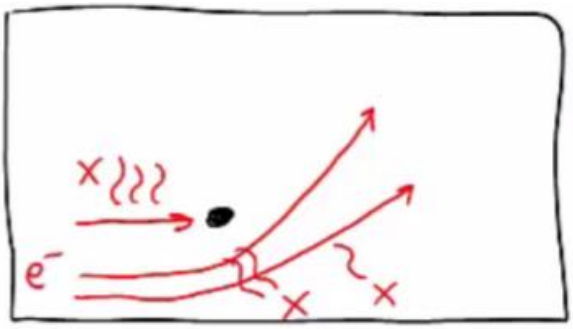
Energy of an x-ray photon:



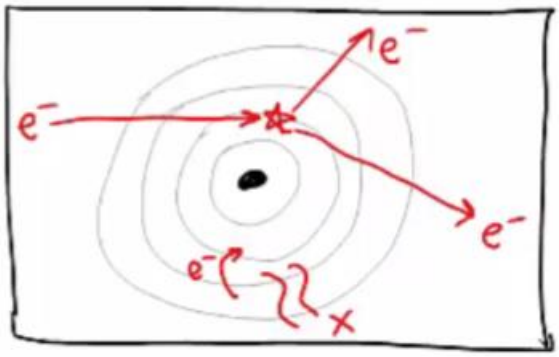
Producing X-Rays

To make x-rays, you can shoot high-energy electrons into matter. We will discuss 2 such interactions:

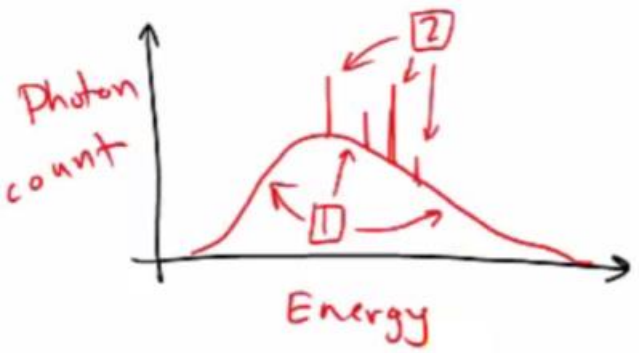
1. Bremsstrahlung Radiation
   - **Electron is decelerated by the charge of the nucleus**
   - **Energy of photon depends on charge of the nucleus and on trajectory of e^- (and its kinetic energy)**



2. Characteristic radiation
   - **Knocks an electron out of its shell, causing a cascade of e^- dropping shells**
   - **Energy of photon depends on the binding energies of the resulting e^- shells (a discrete set of quantities)**



Resulting X-Ray Spectrum:

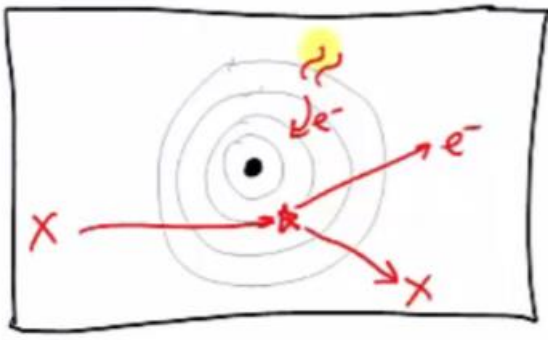


In medical-imaging applications, x-rays are sent into tissue (the tissue is “irradiated”)

X-Ray Interactions with Matter

We will discuss 3 common interactions:

1. Photoelectric Effect (Einstein Nobel Prize 1921)

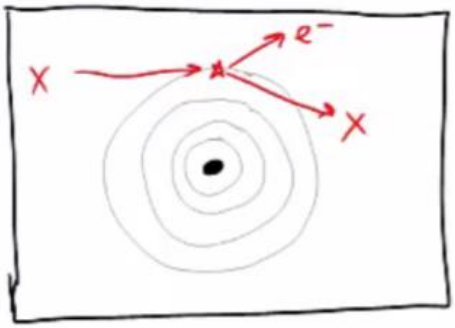


- An x-ray photon ejects an **electron from its shell**
- Probability of interaction **P α Z^3 <- atomic # of atom (proportional to atomic # cubed)**

This effect is the principal effect that makes x-ray useful. Different **tissues** have different concentrations of **atoms with high atomic number**, so have different **probabilities for this interaction**. Thus, different tissues have different **x-ray attenuation (to make smaller) properties** x-ray images are essentially x-ray **attenuation maps**.

➔ Some photons going in, fewer going out

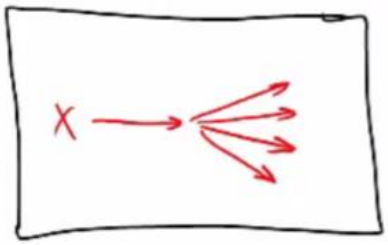
➔ Attenuation: how much the beam decreased



2. Compton interaction
   - Eject **electron in outer shell, so x-ray keeps most of its energy**.
   - But **x-ray changes direction**
   - Probability of this interaction **P α electron density**
   - Photons do not ionize – not enough energy
   - Radiation every
     - Ionizing radiation => problem

Since tissues tend to have little variation in **electron density**, this interaction does not give us information about what is inside the body **-> nuisance**

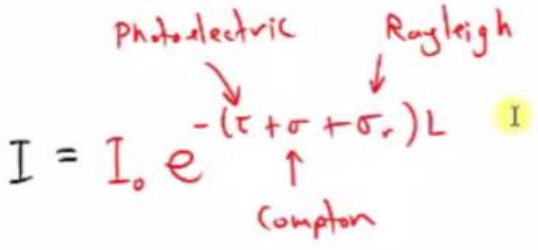
3. Rayleigh (coherent) scattering

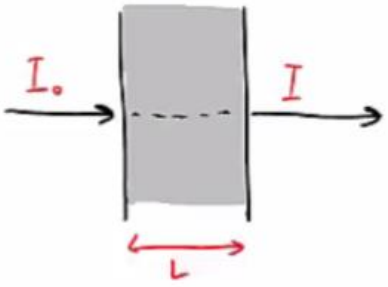


- **No transfer of energy**
- **But a change in direction**

This interaction also does not give us anatomical information **-> nuisance**

X-Ray Attenuation





Or, more simply,

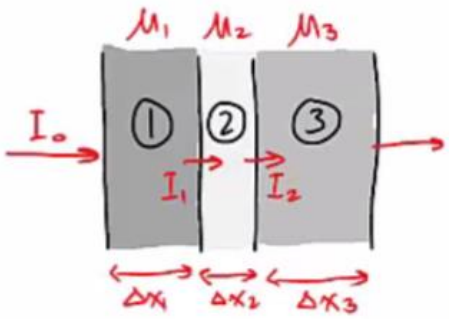
I = $I_0 e^{-\mu L}$ ("Beer-Lambert Law")

Now let's put a few layers together

Exiting ①

$$I_1 = I_0 e^{-\mu_1 \Delta x_1}$$

↑ this is the intensity that enters ②



$$I_2 = I_1 e^{-\mu_2 \Delta x_2}$$

$$I_3 = I_2 e^{-\mu_3 \Delta x_3}$$

Chain them all together…

$$I = I_0 e^{-\mu_1 \Delta x_1} e^{-\mu_2 \Delta x_2} e^{-\mu_3 \Delta x_3}$$

$$= I_0 e^{-(\mu_1 \Delta x_1 + \mu_2 \Delta x_2 + \mu_3 \Delta x_3)}$$

$$= I_0 e^{-\sum_n \mu_n \Delta x_n}$$

Take the limit as Δx -> 0

$$I = I_0 e^{-\int \mu(s)\, ds}$$

↑ attenuation at s

I/I_0 is intensity

Thus $\boldsymbol{\ln\left(\frac{I_0}{I}\right) = -\int \mu(s) ds}$ **Radon Transform**

The beam intensity (# of x-ray photons) that transmits through the tissue and emerges on the other side encodes attenuation information for its entire path.

The problem with x-ray images is that **all structures are projected on top of each other, superimposed on the image**.

- Ionizing radiation

<u>Hounsfield Units</u>

There is an absolute scale for x-ray attenuation coefficients.

$$H = \frac{\mu - \mu_{water}}{\mu_{water}} x1000\ Hu$$

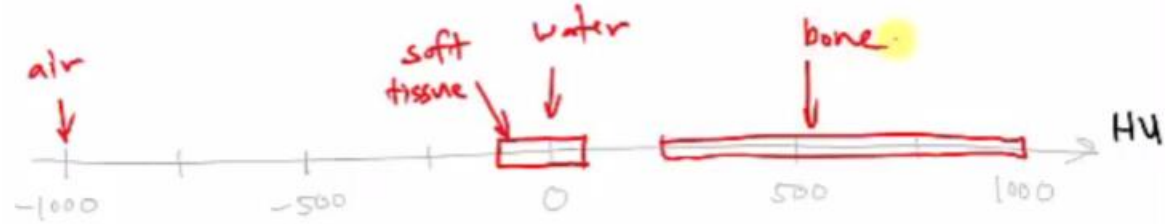
(normalized to water)

e.g. H_{air} = -1000 Hu

$H_{soft\ tissue}$ = -100 to 60 Hu

H_{water} = 0 Hu

H_{bone} = 250 to 1000 Hu

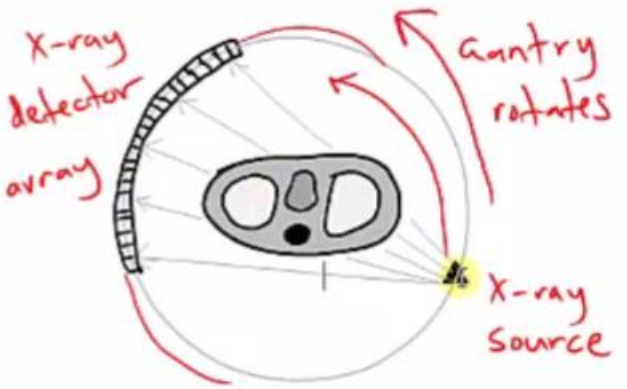


Lesson 13 – Computed Tomography (CT)

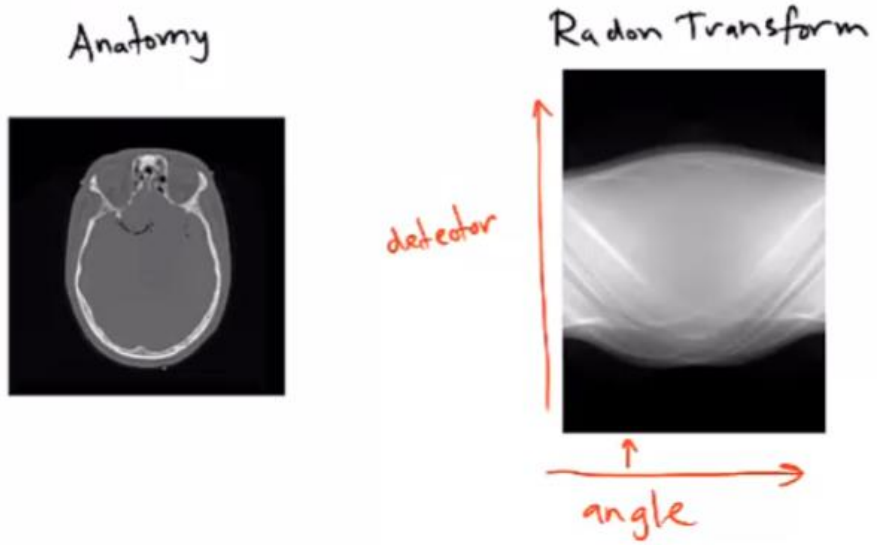
Goal: to find out how CT works, and how it is used (its pros and cons).

All known as **CAT scan (computer-assisted/aided tomography)**.

The idea is to resolve a single slice of an object using many x-ray projections.



As the gantry rotates, the scanner collects a 1D x-ray at each angle.



Projecting up, left, down (from bottom of scan) => radon transform

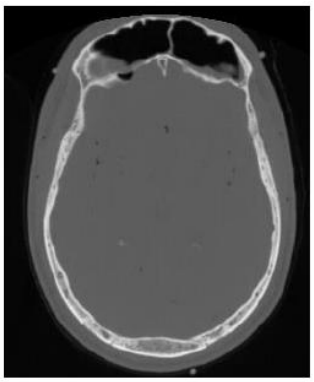
Properties of CT

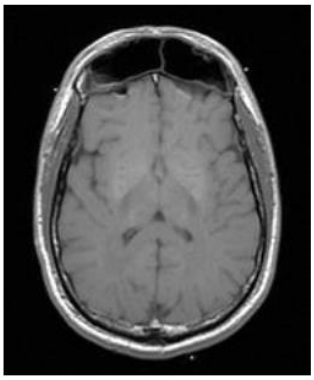
Some things to consider when choosing an imaging modality.

Contrast

CT is great for distinguishing **bone from soft tissue**, but not too good for distinguishing **soft tissue apart**.

*what's light, dark – want to differentiate



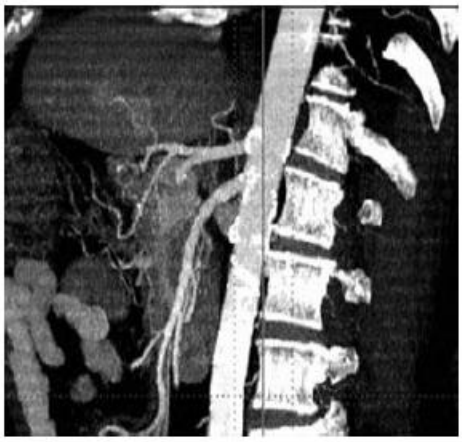


bone, fat, brain tissue

Bone tissue, nasal cavity (black), soft tissue

Contrast Agents

Substances can be introduced to the body to add contrast. These are called contrast agents. They are especially useful for visualizing **vasculature (arteries and veins)**.

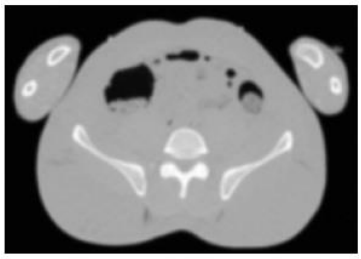


Without contrast agent with contrast agent (brighter) (aorta now highlighted)

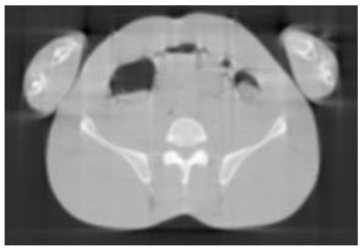
Higher attenuation in …?

Motion Artifacts

It takes a few seconds to acquire one slice of CT data. If the patient moves during that acquisition, the resulting Radon transform will be inconsistent, and the reconstructed image will contain errors.



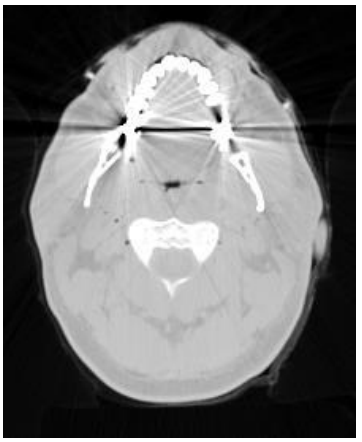
If the patient motion is known, a lot of the artifacts can be corrected during the reconstruction. There are also some automatic methods that try to sharpen the image by guessing the motion.



(top: motion free, bottom: patient moved 3 degrees halfway through the scan)

Dense Object Shadowing

Highly radio-opaque object in the field of view do not fit the model assumed by the imaging and reconstruction process. These include metal objects like dental fillings, bullet, artificial joints, etc.



(hips, fillings, bullet)

(fillings, x-rays going through body not attenuating properly)

(waves “emanating” from fillings)

Ionizing Radiation

X-rays are a form of ionizing radiation. Ionizing radiation is radiation with high enough energy that electrons can be ejected out of their orbitals, creating ions. These ions, in large numbers, can tissue damage & damage DNA.

For these reasons, we actively limit the amount of ionizing radiation a person gets (the government keeps track of all your x-rays, CT scans, etc., and sets yearly and lifetime limits).

Recall: **risk vs benefit**

Speed

The speed of a CT scan is really only limited by **how fast the x-ray source can be moved (and by x-ray flux)**. After all, the x-rays themselves move at the speed of light. So, although patient motion can be an issue, CT is among the best for speed.

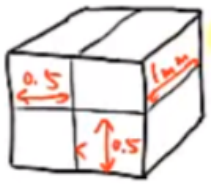
Quantitative

The intensity values are in Hounsfield units, which is an absolute intensity scale.

Recall: H_{air} = -1000 HU, H_{water} = 0 HU

This helps when processing the images. For example, we know the range for bones, so we can do a pretty good job of deciphering which voxels are bone without even looking at the image.

Resolution

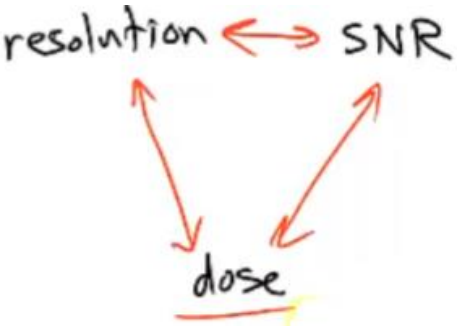


CT scanners designed for human diagnostic imaging are generally capable of producing images with voxels of size **0.5 mm x 0.5 mm 1 mm**

Typical image size is **512 x 512 pixels**.

Tradeoffs

We could achieve higher resolution and higher signal-to-noise ratio (SNR) if we could increase the x-ray dose. So CT scans are carefully designed with all 3 factors in mind to minimize the dose, while still acquiring images that service their diagnostic purpose.



Lesson 14 – PET

Goal: to find out what a PET scan is, how it works (briefly), and what it is used for.

PET stands for **positron emission tomograph**.

Similar to CT in that the scanner detects radiation using a ring of detectors.

Different than a CT scanner because the radiation is **emitted from inside the body**, rather than being **transmitted through the body (as in CT)**.

PET is a **functional imaging modality**. That means it is used to observe **where** in the body a particular function is occurring. For example, one might want to know where rapid tissue growth is taking place (to locate growing tumors). This is different from viewing the tissue itself.

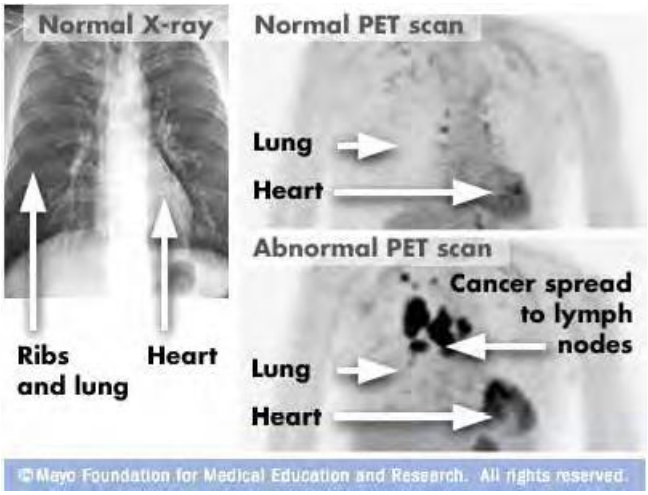
How does it work?

A small amount of positron-emitting radioisotope is injected into the subject. This radioisotope is bound to a metabolite that is used in the function we wish to observe. For example, glucose can be tagged to produce

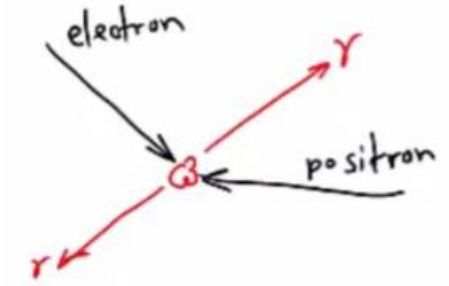
fluoro-2-deoxy-D-glucose (FDG)

The FDG will migrate to the sites where glucose is being consumed rapidly. Malignant tumors require a lot of glucose to grow, and tend to cause a locally high concentration of FDG. PET imaging can estimate the concentration of the radioisotope in tomographic sections (slices).

High concentration is shown dark



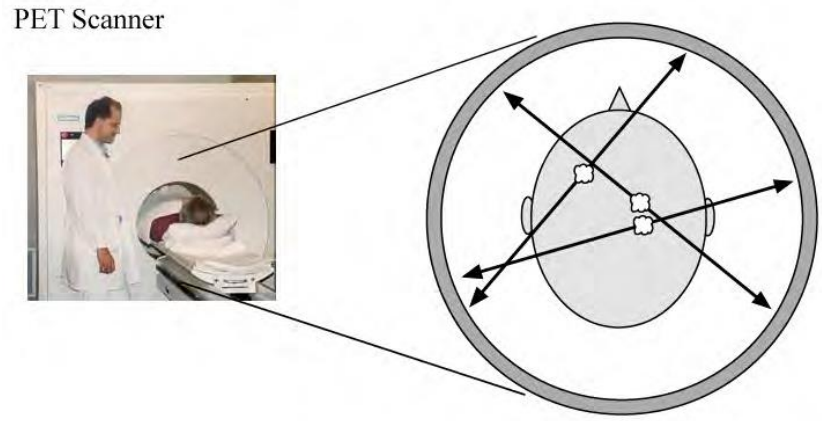
As the radioisotope decays, it gives off positrons. Each positron collides with an electron, and they **annihilate** each other. But the event releases two **gamma rays (particles) in opposite directions**. These rays are each **511 keV**.



The scanner listens for 2 gamma rays being **acquired at the same time** these 2 rays likely originated from the same annihilation event. Hence, we know that the radioisotope was somewhere along the line connecting the two detectors.

*don't know where gamma goes, but rays move in opposite direction

The raw data that comes out of a PET scanner is very similar in nature to the Radon transform in CT.



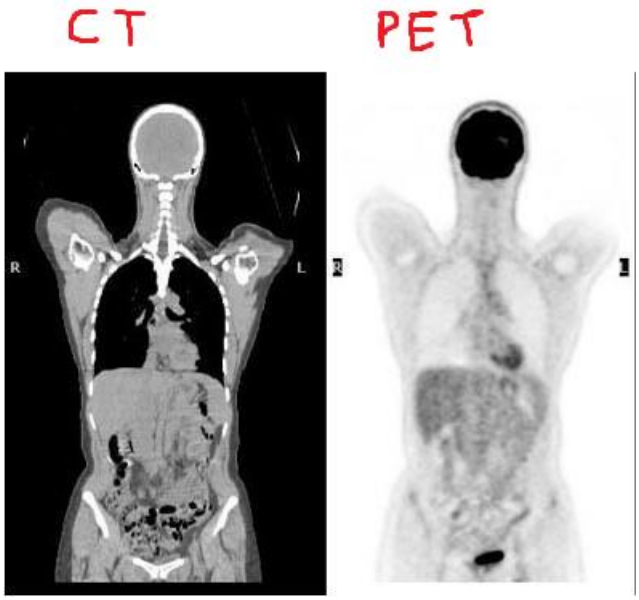
Properties of PET

Speed

PET scans take a long time (30 minutes to 3 hours), so patient motion is a problem. After that time, the radioisotope has decayed to a very low level; the half-life of FDG is 108 minutes.

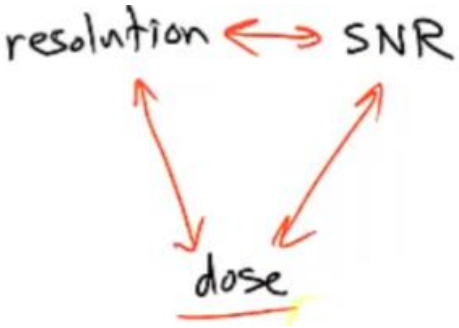
Ionizing Radiation

As the gamma rays (ionizing radiation) exit the body, they encounter tissues. Just like in normal x-rays, these gamma rays can be **absorbed**, and the probability depends on the tissue type (e.g. bone absorbs better than fat). To properly reconstruct the isotope concentration, one needs to know the map of attenuation coefficients – that's essentially what a CT scan gives. So a PET scan is always accompanied by a CT scan. In fact, many PET scanners have built-in CT scanners



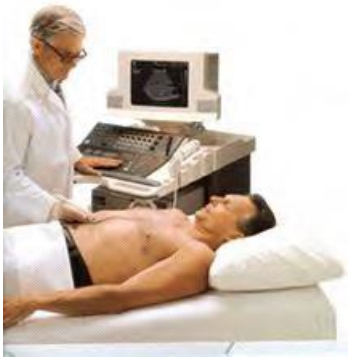
Tradeoffs

Just like in CT



Lesson 15 – Ultrasound

Goal: to find out how ultrasound works (briefly, what it is used for), and what are its advantages and disadvantages. Ultrasound (US) uses **soundwaves** above the audible frequency. These sound waves reflect off **tissue interfaces** (e.g. at the interface between heart muscle and blood). An US transducer sends these waves into the body, **then listens for echoes**. The longer it takes an echo to arrive, the **further away** the reflecting interface.



Properties of Ultrasound

Safe

There are no known dangers to US, so it is used frequently, e.g. to monitor fetal development.

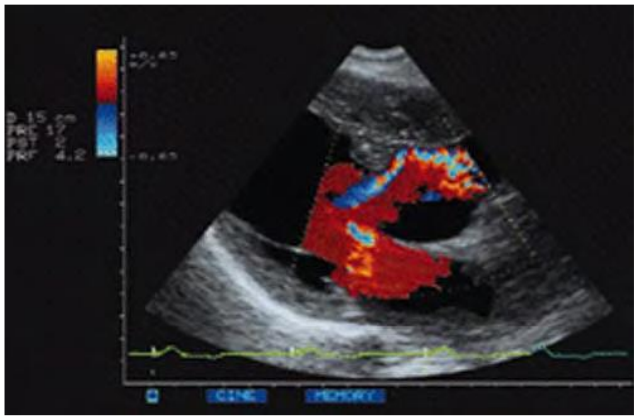
Cheap

US scanners are small and relatively cheap:

- Approximately $20000
- CT scanner, approx. $500000
- MRI scanner, approx. $4000000

Speed (Real-Time)

US images are displayed in real-time, as a video. You can actually watch things moving, like a heart pumping or a fetus shifting (heart_tongue video)



Doppler Imaging

US is also capable of detecting the motion of objects (or blood) toward or away from the transducer. US the velocities can then be rendered as colours in the video (red and blue as opposite directions). This is helpful for diagnosing heart valve regurgitation (leaking).
(Doppler video)

Noisy

US images are notoriously noisy, with lots of “salt and pepper” speckle noise. Because of this, single static images can be difficult to interpret. The noise hast less of an impact when viewed as a video.

- ⇨ “smoothing” through time

Shadows

Dense structures do not transmit sound very well, so it can be difficult to image some parts of the body.
e.g.
- US cannot be used for brain imaging unless the skull is opened
- Also, the heart must be viewed between ribs

Other Modalities

MEG – Magneto Encephalography (brain electric field)
EEG – Electro Encephalography (brain waves (current))
OCT – Optical Coherence Tomography
SPECT – Single Photon Emission Computed Tomography

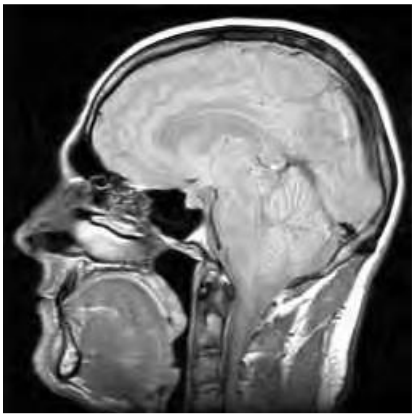
Lesson 16 – MRI Physics

Basic Physics of MRI (Magnetic Resonance Imaging)

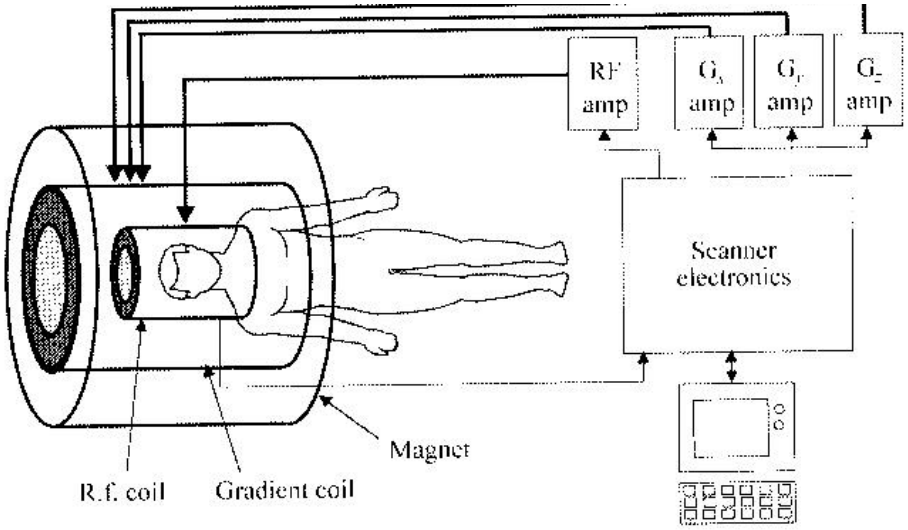
Superconductor -> cold => less resistance to build a current
Emergency “quenching” button vents coolant

What is MRI good at?

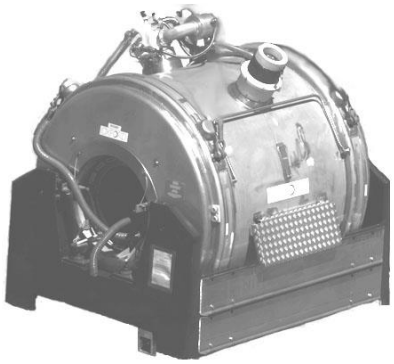
Among other things (to be discussed later), TISSUE CONTRAST!



Necessary Equipment

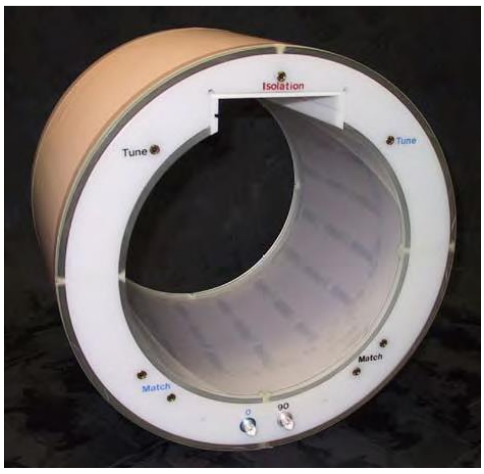


Magnet



Gradient Coil

RF Coil

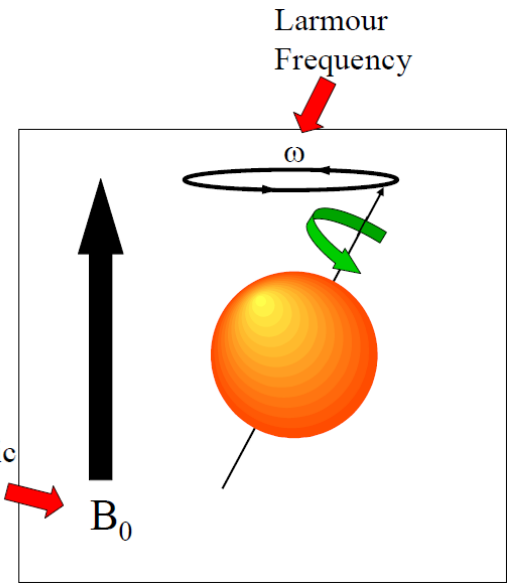


Dipole Spin

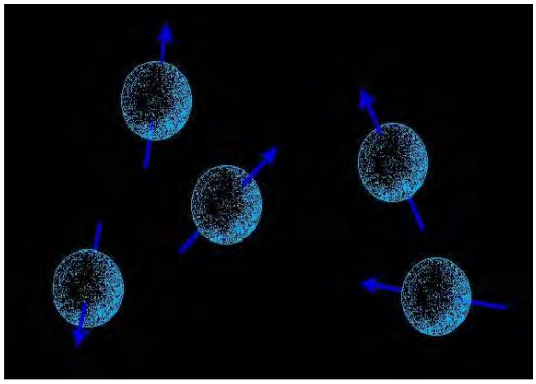
Magnetic moment

$$\frac{d\vec{\mu}}{dt} = \gamma\vec{\mu} \times \vec{B}_0$$

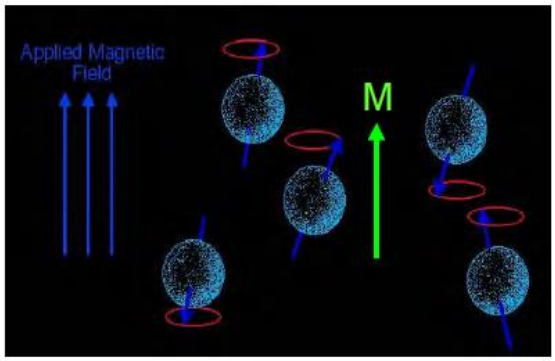
$$\Rightarrow \ \omega = \gamma B_0$$



Net Magnetization Vector

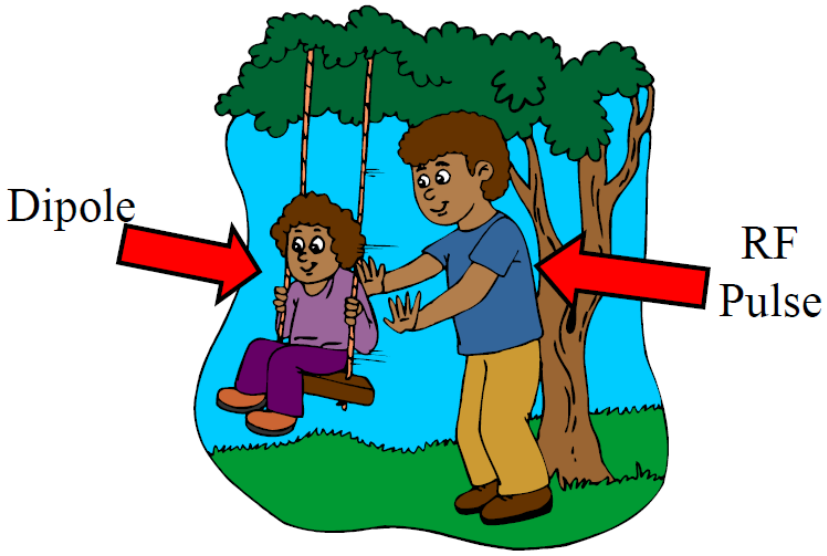


Spins orient themselves randomly

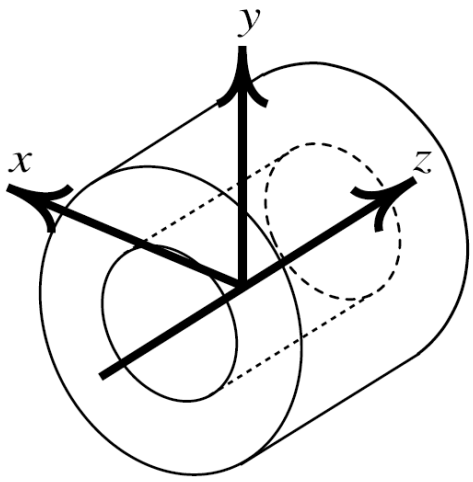


But in a magnetic field, they orient either “spin-up” or “spin-down”, and result in a net magnetization in a given neighbourhood.

Nuclear Magnetic Resonance (NMR)



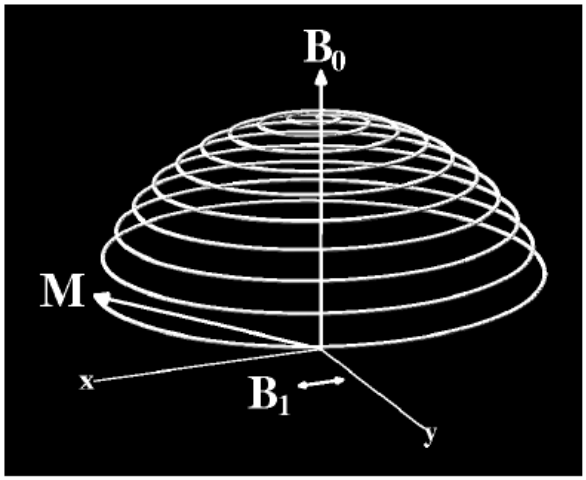
Scanner Coordinate System

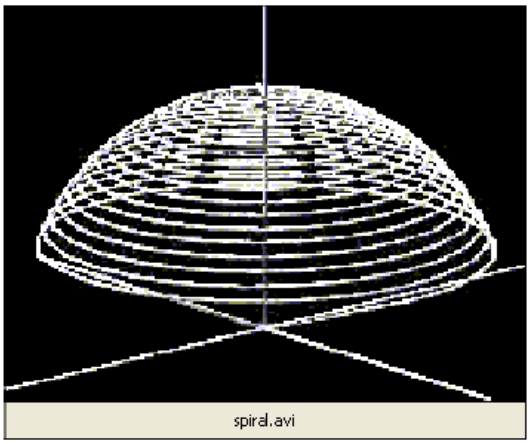


bore of scanner

Nuclear Magnetic Resonance (NMR)

By pulsing a magnetic field perpendicular to $\boldsymbol{B}_0$, we can move $\boldsymbol{M}$ out of alignment with $\boldsymbol{B}_0$. This pulsing magnetic field is called the RF (radiofrequency) pulse, and pulses at the Larmour frequency.





- Pulsing and resonance
    - Push out of equilibrium

<u>Relaxation</u>

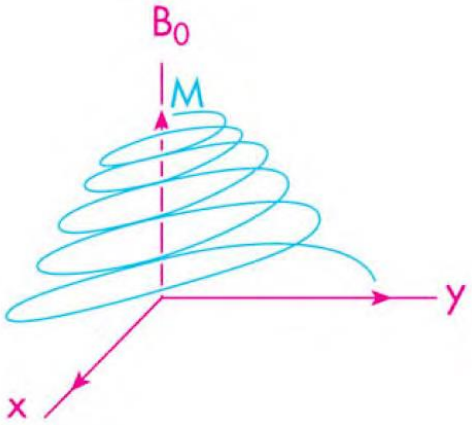
$$\frac{d\vec{M}}{dt} = \gamma\vec{M} \times \vec{B}_0 - \frac{1}{T_2}\left(M_x\vec{i} + M_y\vec{j}\right) - \frac{1}{T_1}\left(M_z - M_0\right)\vec{k}$$

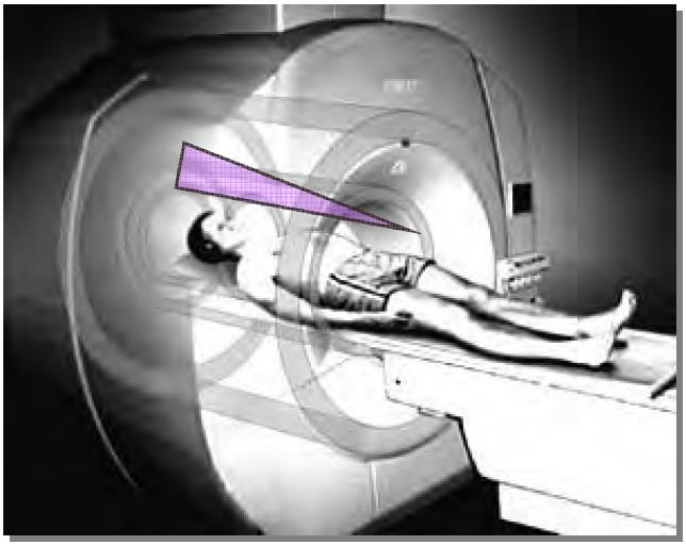
Transverse component spirals down to nothing. (Unit 2 S31)

$$M_{xy}(x,y,t) = ce^{-i\gamma B_0 t}e^{\frac{-t}{T_2}}e^{-i(k_x x + k_y y)}$$

Longitudinal component relaxes back to the state $\boldsymbol{M_0}$.

$$M_z(t) = M_0\left(1 - c_1 e^{-\frac{t}{T_1}}\right)$$

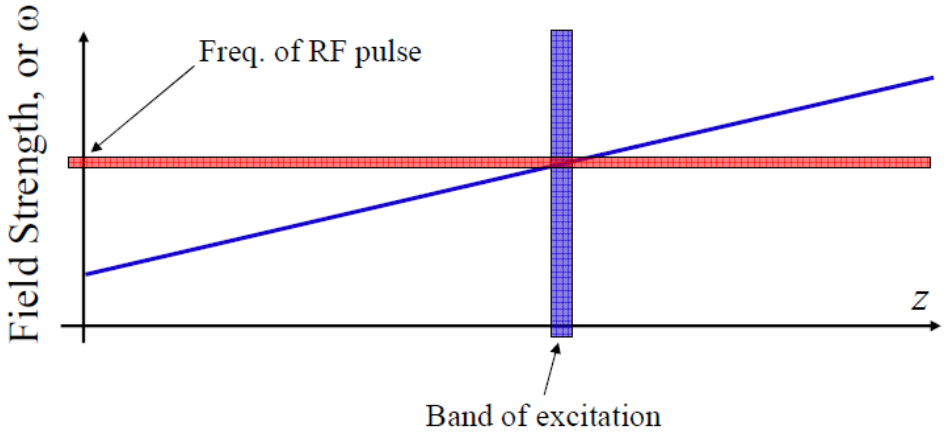


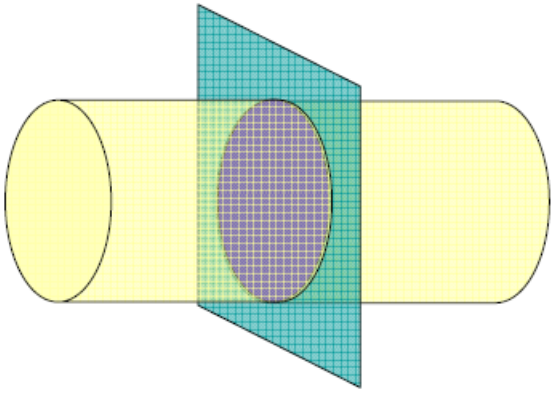


Paul C Lauterbur and Peter Mansfield, Nobel Prize 2003 in Physiology or Medicine

Slice Selection

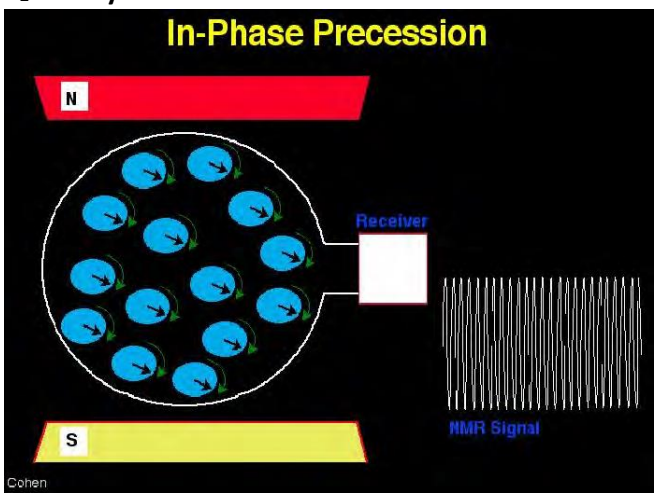
Adding a gradient to the magnetic field causes a gradient in the Larmour frequency. The frequency of the RF pulse will only excite a thin band of matter.

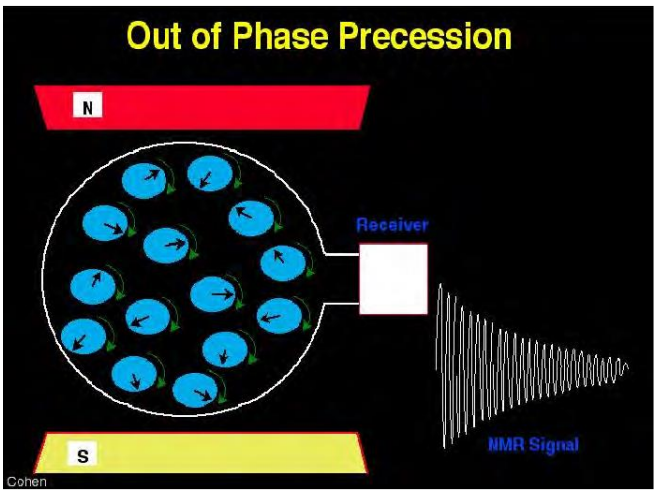




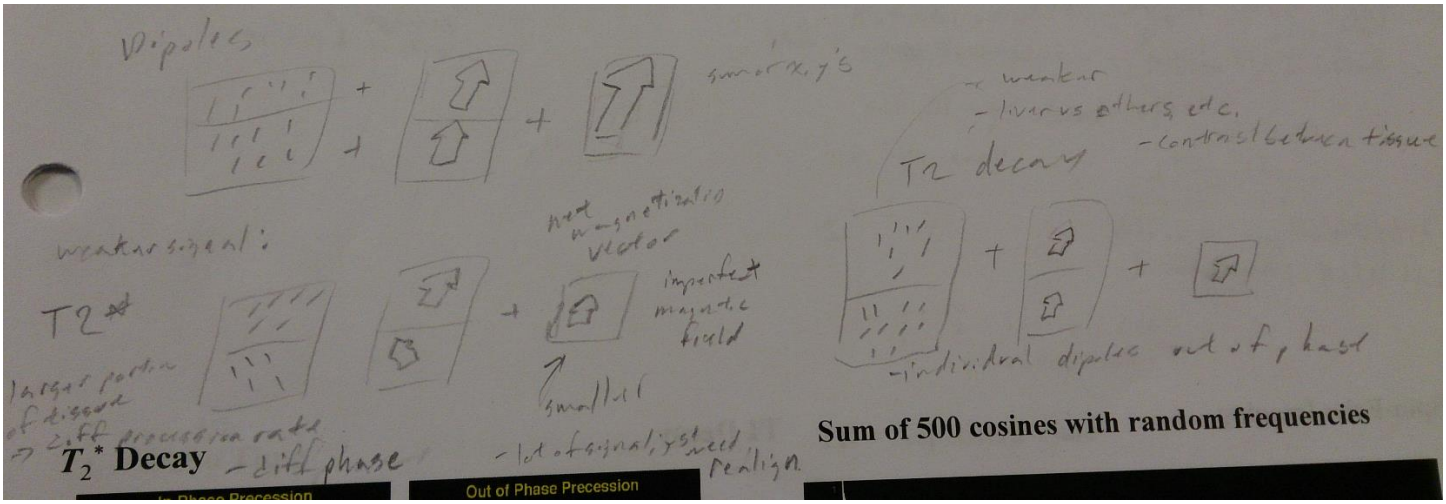
Once a slice is excited, the magnetization vectors in that slice are coherent (synchronized) and create a signal.

T_2^* Decay

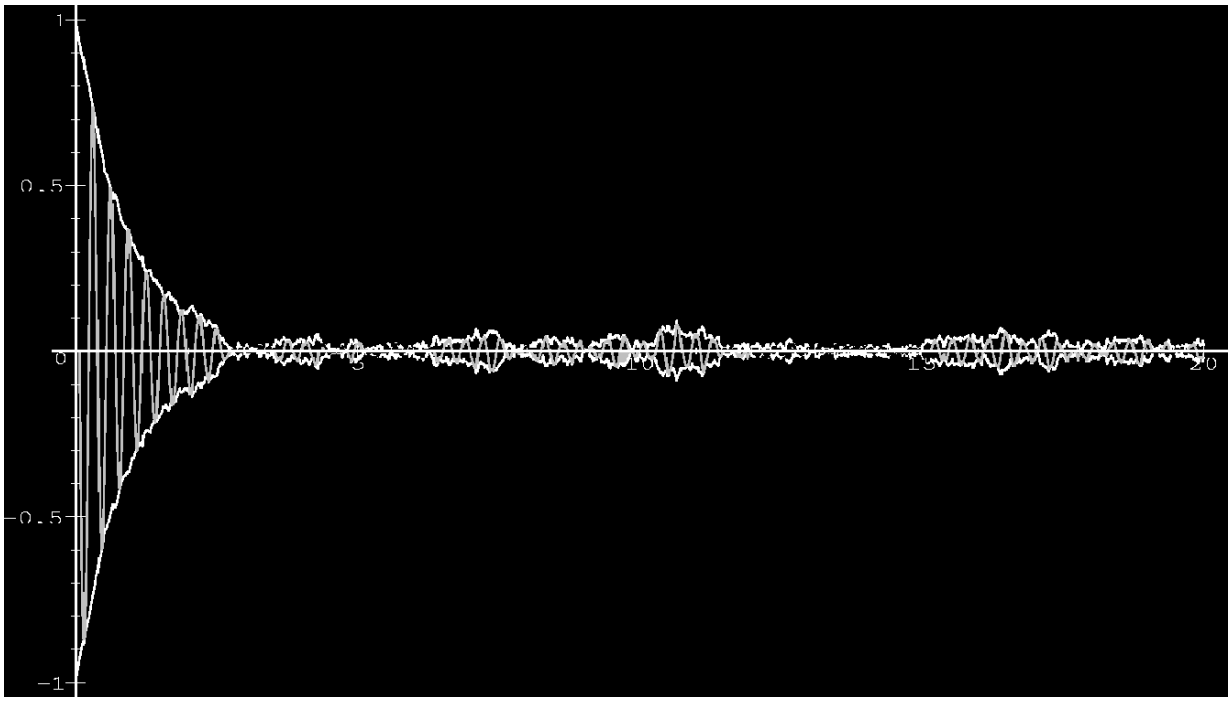




- Protons precess at slightly different frequencies because of
  1. Random fluctuations in the local field at the molecular scale (T2) and
  2. Larger scale variations in the magnetic field (T2*)
- Over time, the frequency differences lead to different phases between the net magnetization vectors (think of a bunch of blocks running at different rates – at first they are synchronized, but over time, they get more and more out of sync until they are random)
- As protons get out of phase, the transverse magnetization decays



Sum of 500 cosines with random frequencies

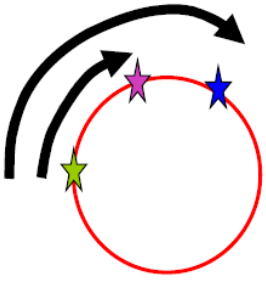


Hahn Spin Echo: Retrieving Lost Signal

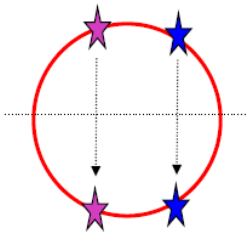
Problem: M_{xy} rotates at different rates in different spots

Solution: take all of the M_{xy}'s that are ahead and make them get behind (in phase) the slow ones

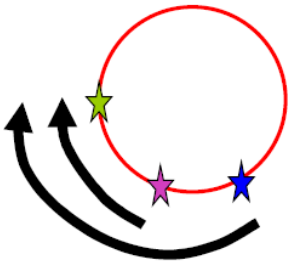
- After a while, fast ones catch up to slow ones => re-phased!



Fast & slow runners



Magically "beam" runners across track

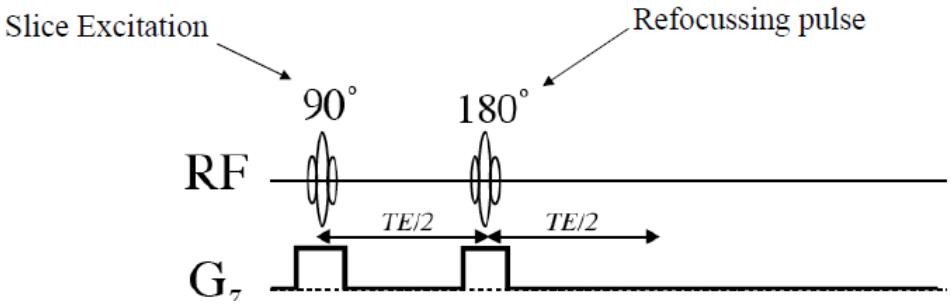


Let them run the same time as before

MRI strong magnet

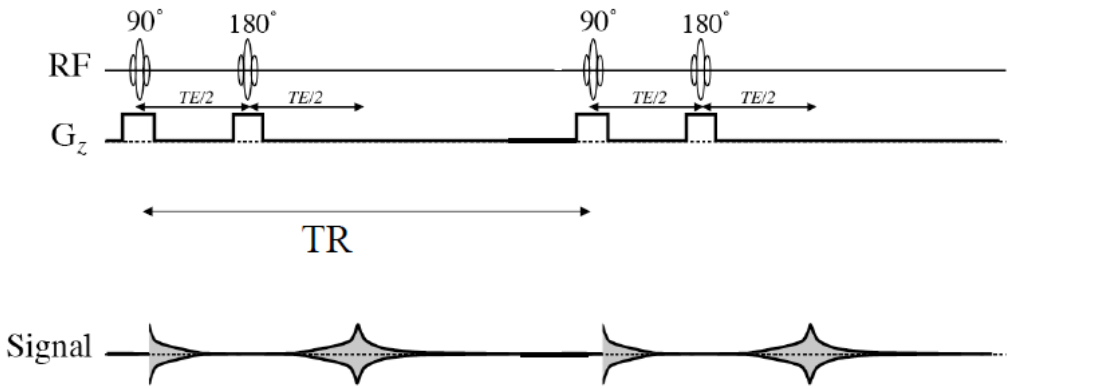
5-gauss line – measure of field strength

Spin-Echo Imaging



At refocusing impulse, dipoles flipped

[dephasing] ^ re-aligns, measure output here for signal



T1 Decay

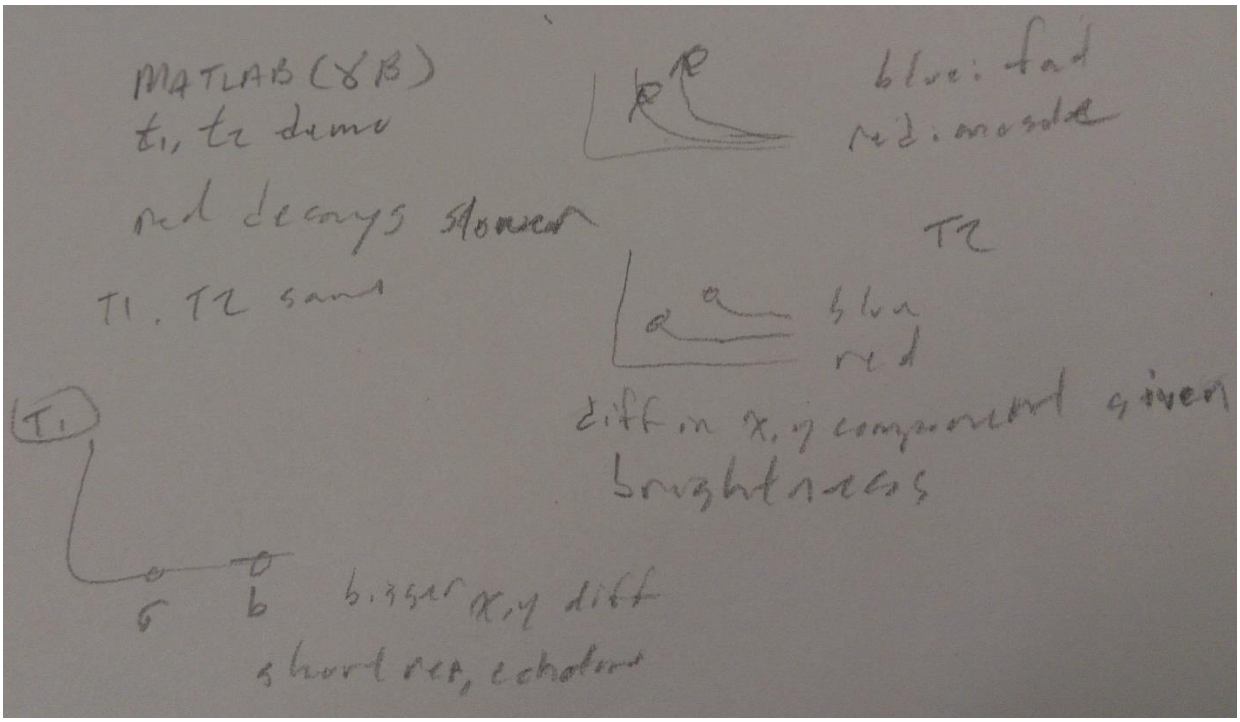
Thermal jostling of molecules disperses energy. In the presence of heavy molecules such as large protein molecules, this energy dispersal is quite rapid. For this reason, tissues such as fat, and those found in organs like the liver and kidneys, have short T1 times. Dipoles in light, mobile molecules tend to hang onto this energy longer. Materials such as cerebrospinal fluid (CSF) and water have relatively long T1 times.

T2 Decay

The decay of the transverse component is mostly dependent on local magnetic inhomogeneities. Dipoles precessing at slightly different frequencies interfere with each other, and can exchange energy. The spins quickly become phased. If the molecules can move around, like in water, the net magnetic field that a particular molecule experiences is averaged and therefore the magnetic field inhomogeneities are somewhat smoothed out. For this reason, the T2 time of aqueous materials is longer than that of fixed or elastic tissues.

Tissue Decay Constants at 1.5 Tesla

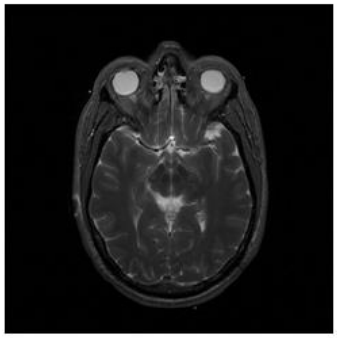
Tissue	T_1 (msec)	T_2 (msec)
CSF	2400	160
White Matter	780	90
Gray Matter	900	100
Muscle	870	45
Liver	500	40
Fat	260	80



TE = Echo Time

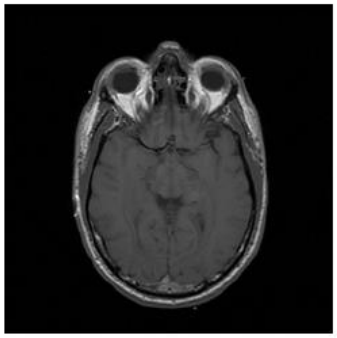
When TE is long, dipoles in tissues with long T2 decay times (e.g. CSF) don't decay as much as dipoles in tissues with short T2 decay times (e.g. muscle).

T2-weighted



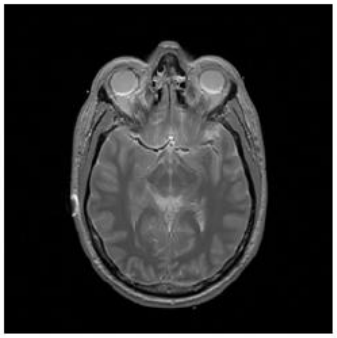
^ liquid = bright

T1-weighted



^fat is brightest – behind eye
Fat relax to z-axis fastest

PD-weighted



proton-density (water-based protons)

Echo time, repetition time

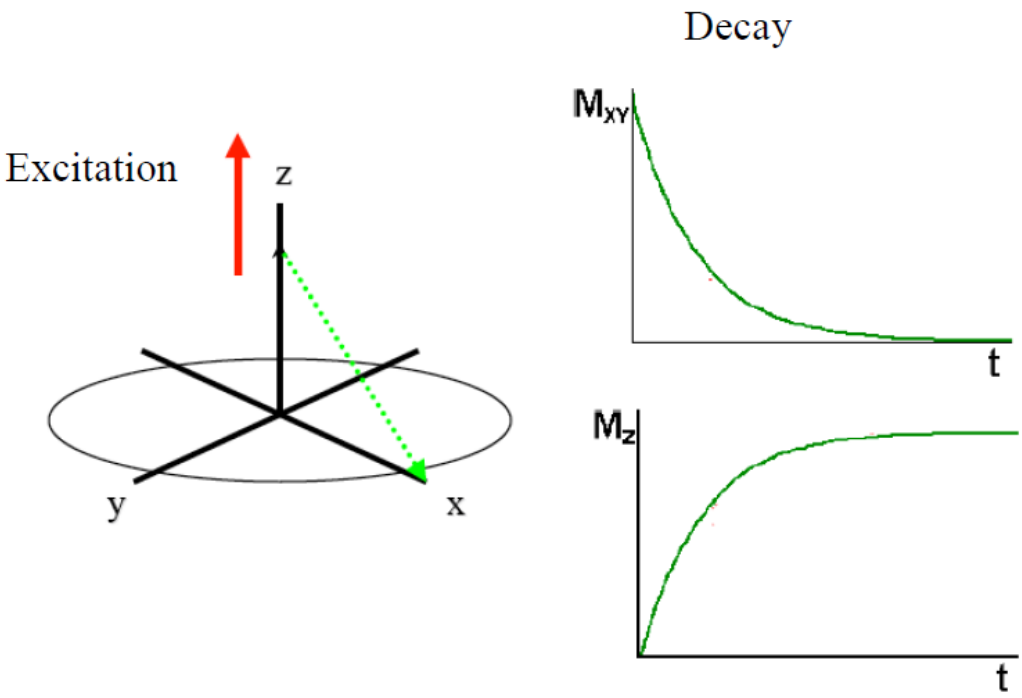
TR = Repetition Time

When TR is short, the dipoles in tissue with long T1 times (e.g. liquids) don't have a chance to relax the full length of M_0. This makes those tissues darker in the corresponding images, called "T1-weighted" images.

Longer T2 time => stay in phase longer

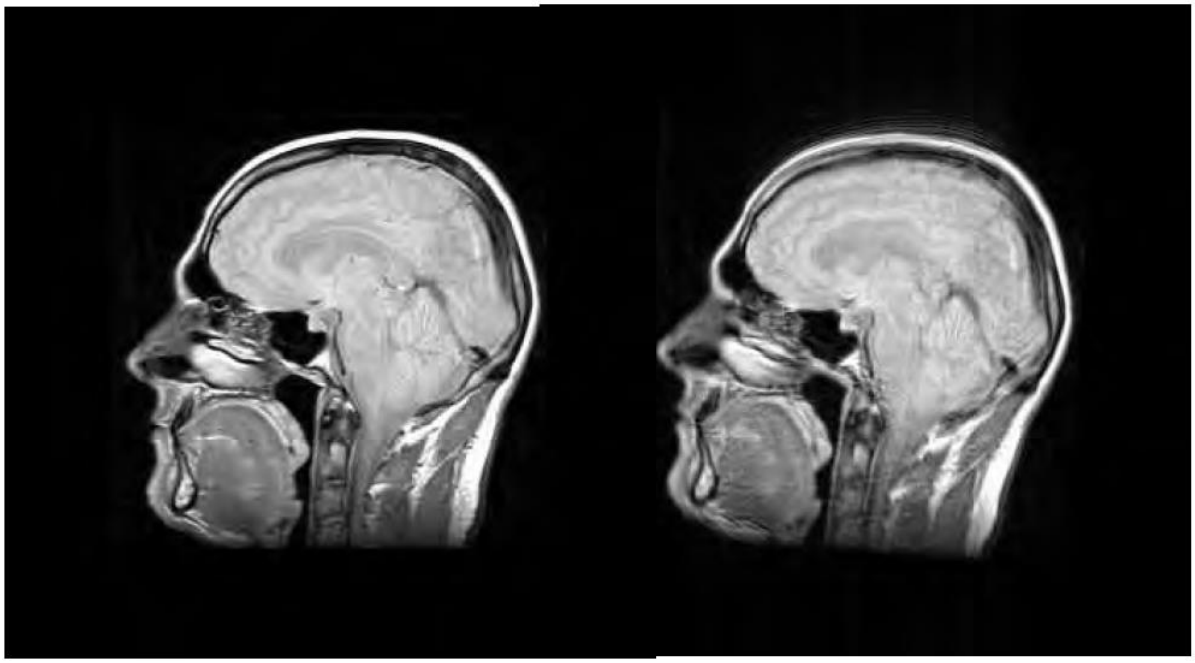
MRI (part 2)

Magnetic Decay



MRI Artifacts

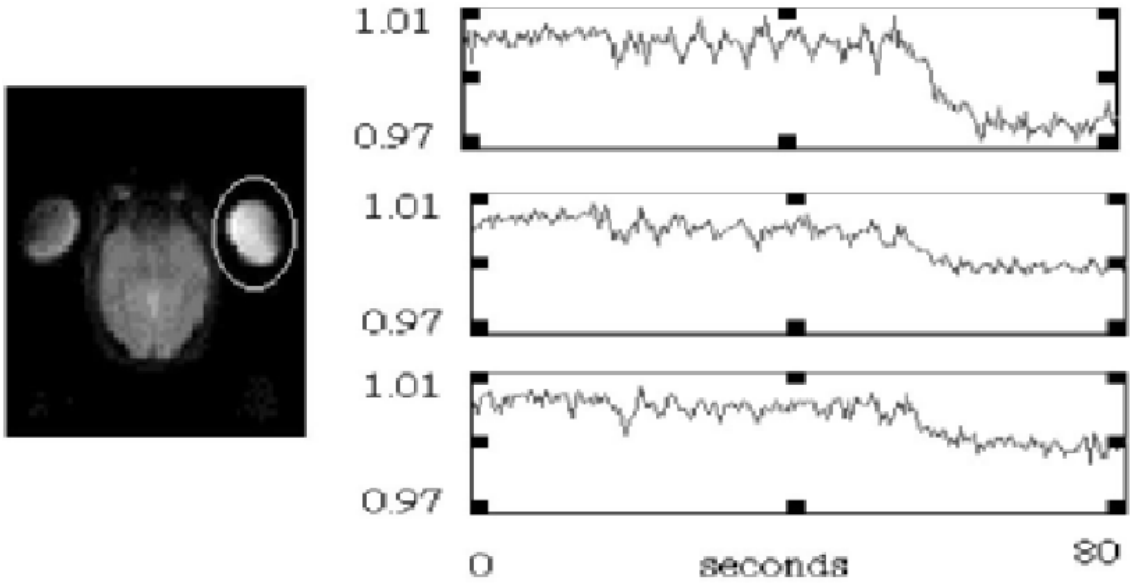
Intra-scan Motion



Physiological Artifacts

Breathing motion of a patient's chest affects the magnet field, and changes the signal measured in a water bottle beside the patient's head.

<Holds breath>



Right: bottle of water

Water giving signal – magnetic field strength changes closer to lungs than head

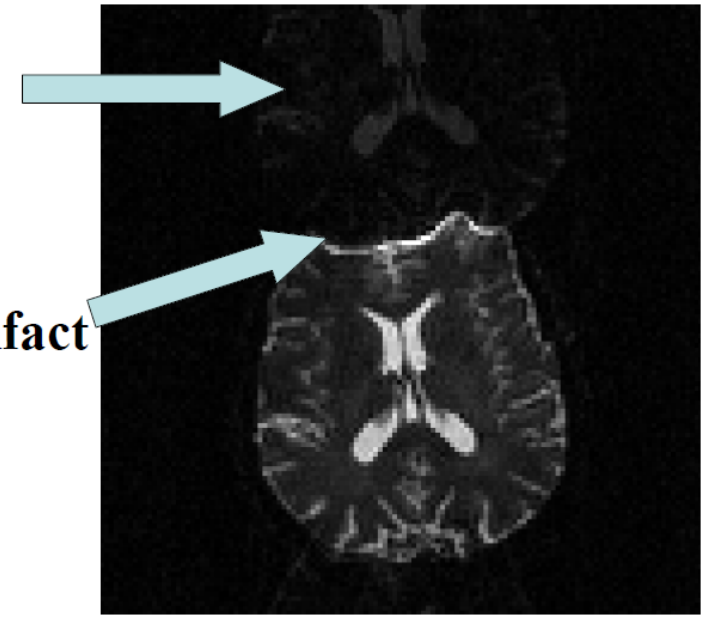
Ghosting – Susceptibility Artifact

Ghosting

Imperfect acquisition timing can lead to ghosting, a low-intensity repetition of the main image.

Susceptibility Artifact

Objects in the MR scanner disrupt the B_0 magnetic field, especially when tissue right next to air (eg. Sinus cavities, or metal objects).



Speed of MRI

It takes time to collect all that data. It takes a finite amount of time for the dipoles to relax close to their steady states. A typical 256x256x129 T1-weighted dataset takes over 5 minutes to collect. Hence, motion is **BIG** problem in MRI.

Resolution

A typical MRI is multiple slices of 256x256 pixels. For a head scan, voxels are often ~1mm.

Tradeoffs

Like many imaging modalities, MRI involves a tradeoff between imaging time, image resolution, and signal-to-noise ratio.

Dangers of MRI

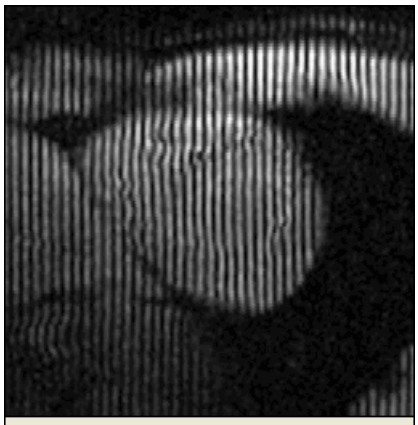
Ballistic metal objects (i.e. wrench, O2 tank – remember video)
Internal (loose) metal objects (iron shavings, IUD) could move and cause internal bleeding
Burning from

- Induction loop in anatomy (i.e. touching leg)
- Radio waves from RF pulses (heating just like a microwave oven)

Interference with implanted devices (e.g. pacemaker, hydrocephalus shunt)

Other uses of MRI

Elastography: similar to diffusion, but done while tissue is vibrated
Spin tagging: encode stripes into tissue, let the tissue movie, THEN acquire the image
MR Spectroscopy: analyzing the signal of a single voxel to determine different chemicals
Angiography: excite the blood in a slice, and then acquire a slice downstream – only blood is bright
Contrast agents: for viewing blood
Functional MRI: detecting neurological activity
Diffusion imaging: detecting preferential motion of water



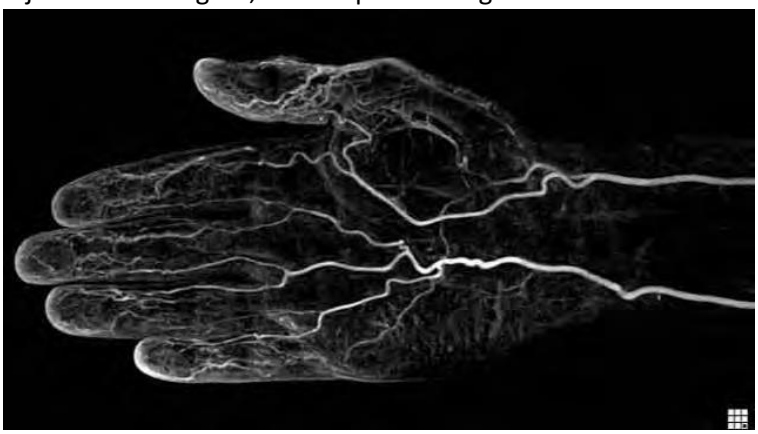
Spin Tagging

Excite stripes of tissue and image as usual

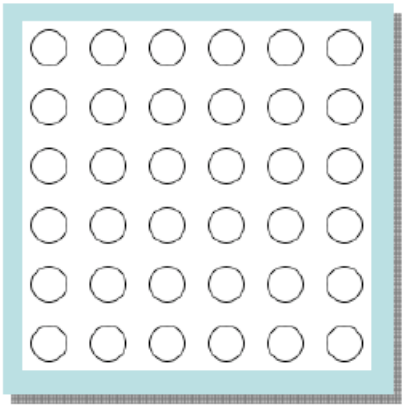
- Track how tissue deforms
- Can use for 3D interference
- E.g. horizontal – heart
    - Heart valve function

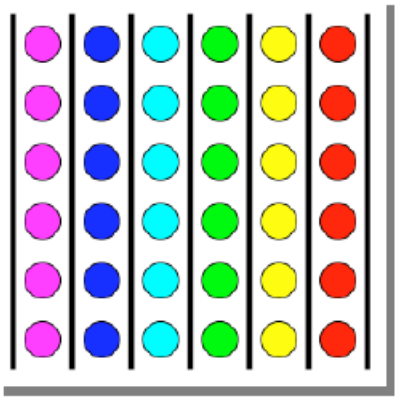
Angiography (imaging blood vessels)

Inject contrast agent, or use spin labeling

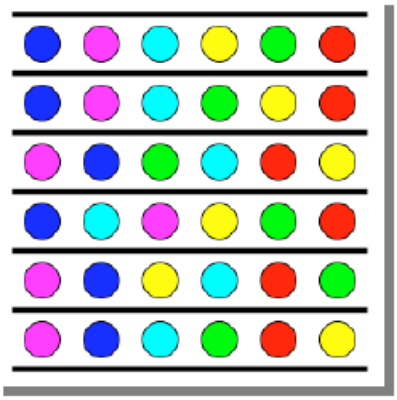


Diffusion Imaging





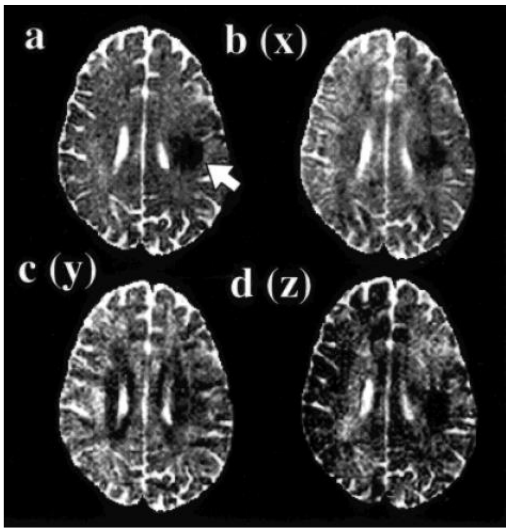
Constrained along
x-direction



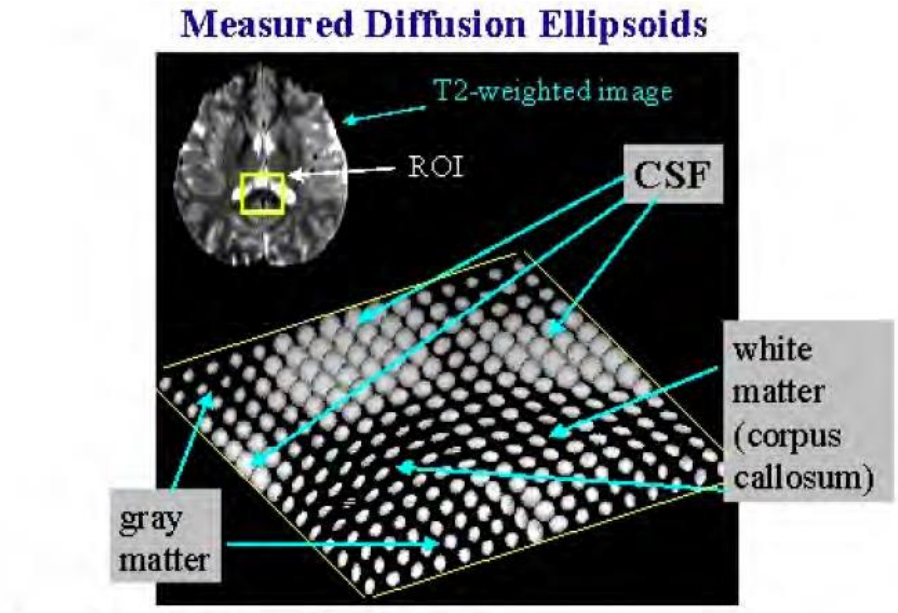
Constrained along
y-direction

Water can diffuse along fiber

diphase – they moved -> want same



Arrow: dead tissue

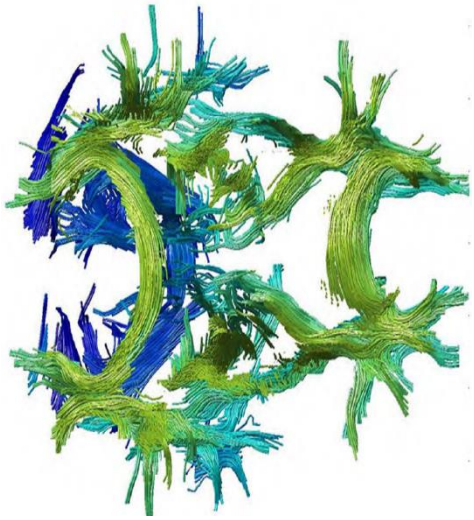


Pierpaoli et al, **Radiology** 1996; 201: 637-648.

White matter tract
Move from one side to another

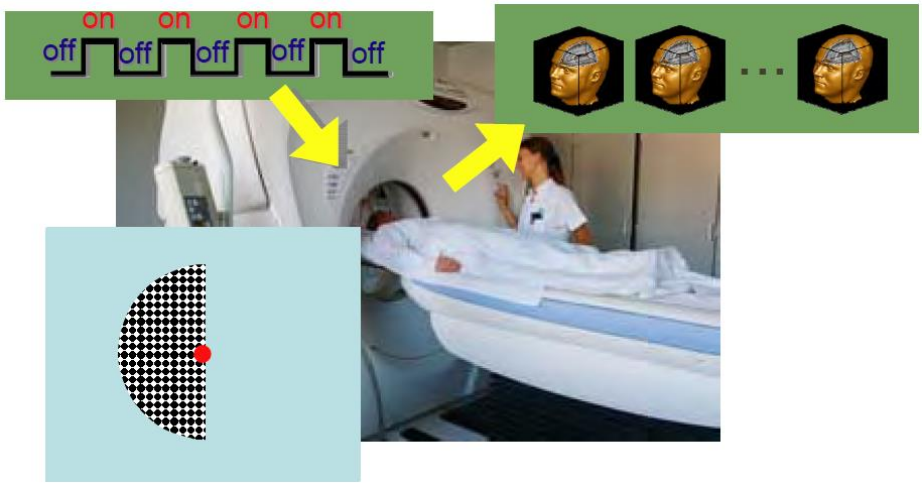
Tractography (follow diffusion tracks)

One hemisphere to another

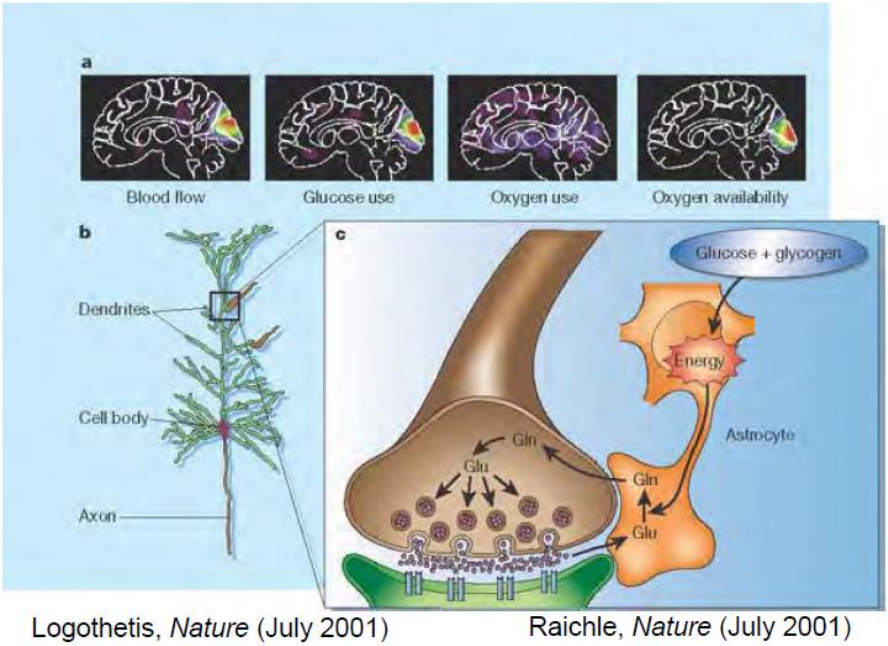


Functional MRI

Cognitive stimulus is turned on and off while snapshots are acquired

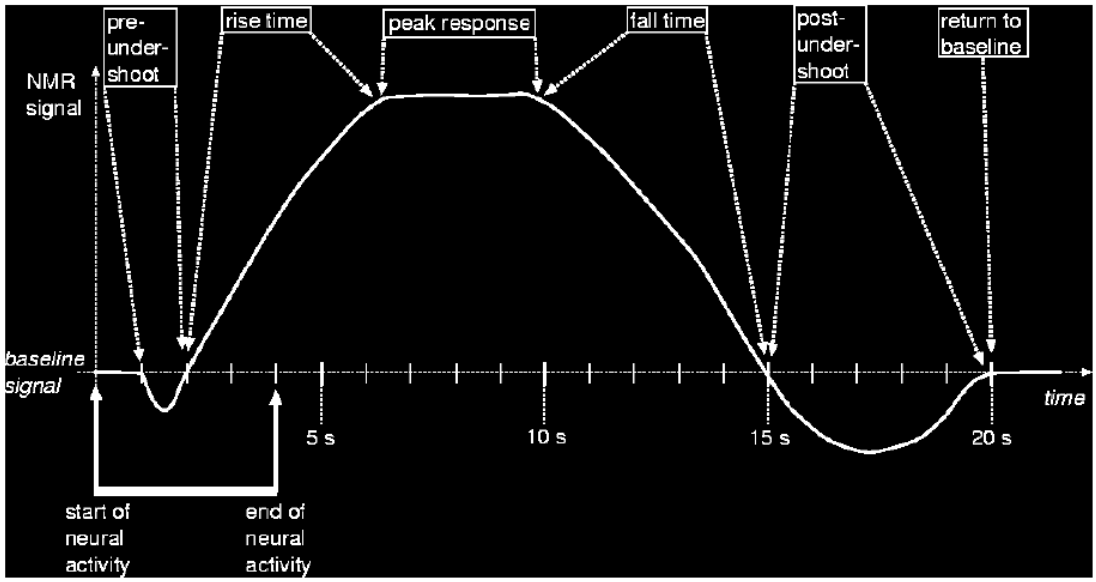


Neurological activity needs blood glucose, but not oxygen. Blood flow increases to supply glucose, resulting in a local increase in oxygenated hemoglobin, which affects the MR signal.

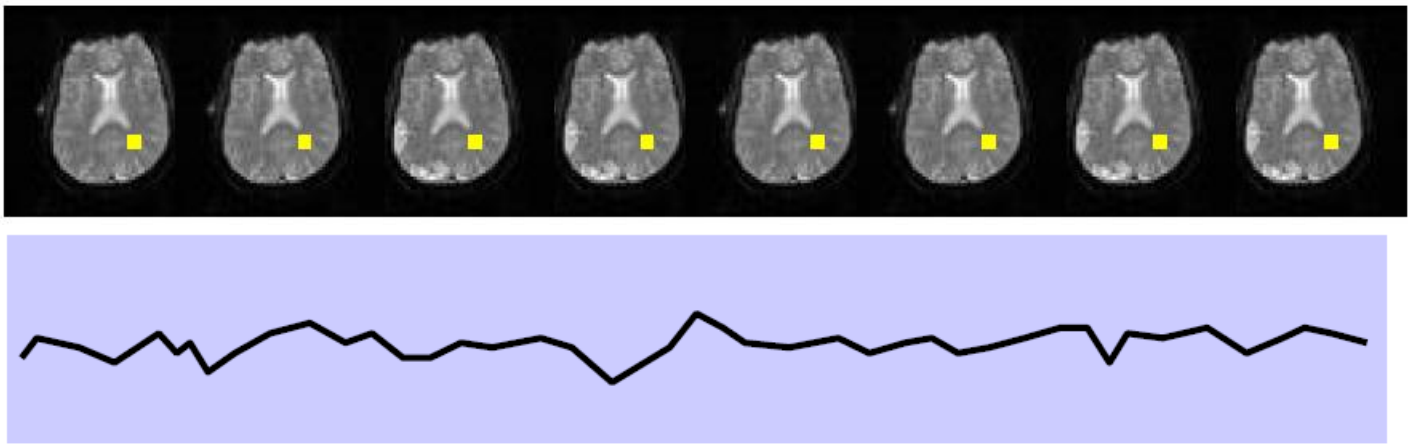


Logothetis, *Nature* (July 2001) Raichle, *Nature* (July 2001)

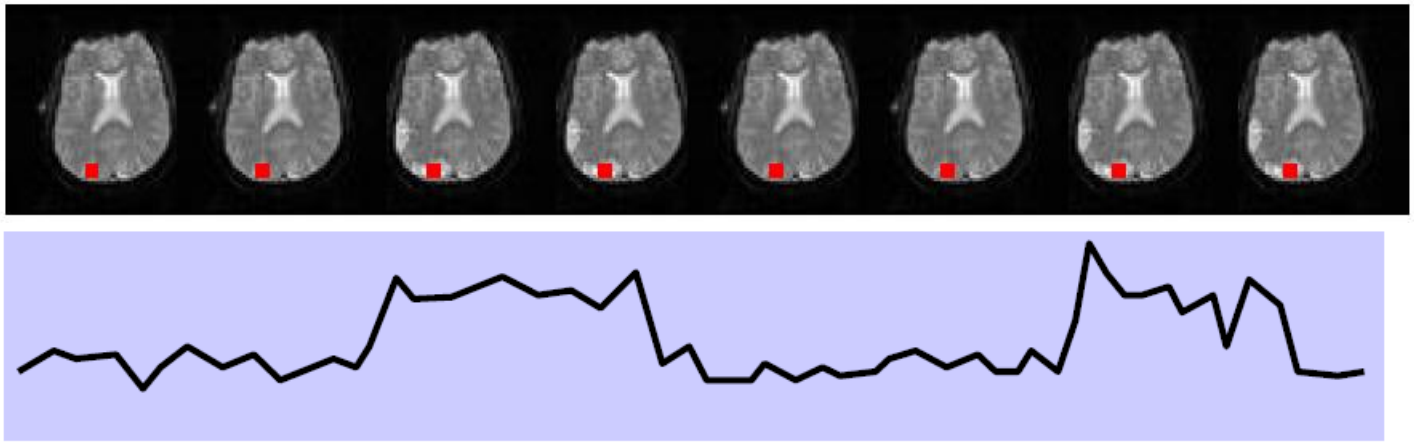
Blood Oxygen Level Dependent (BOLD) Signal



Voxel Time-Series: Inactive

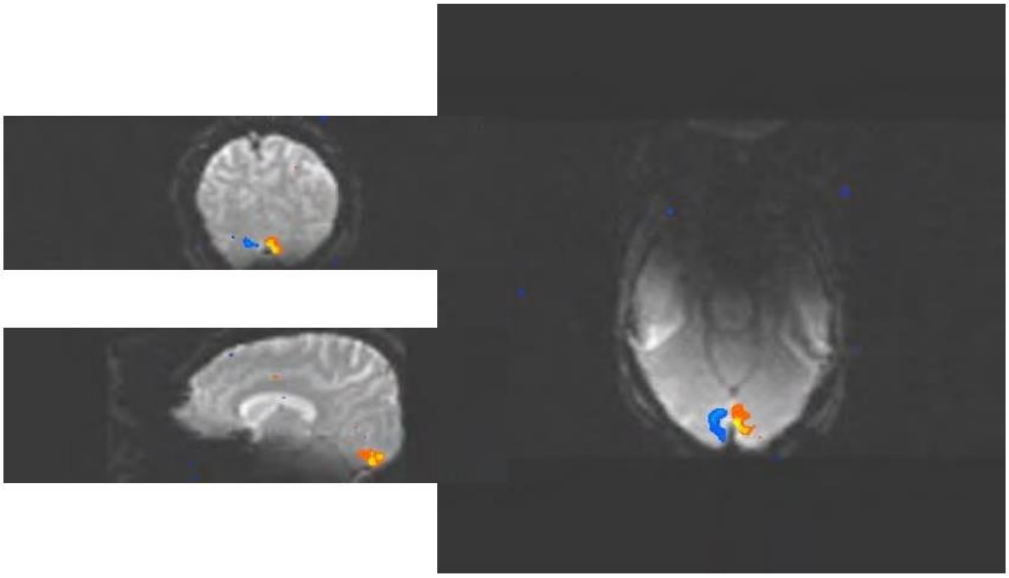


Voxel Time-Series: Active

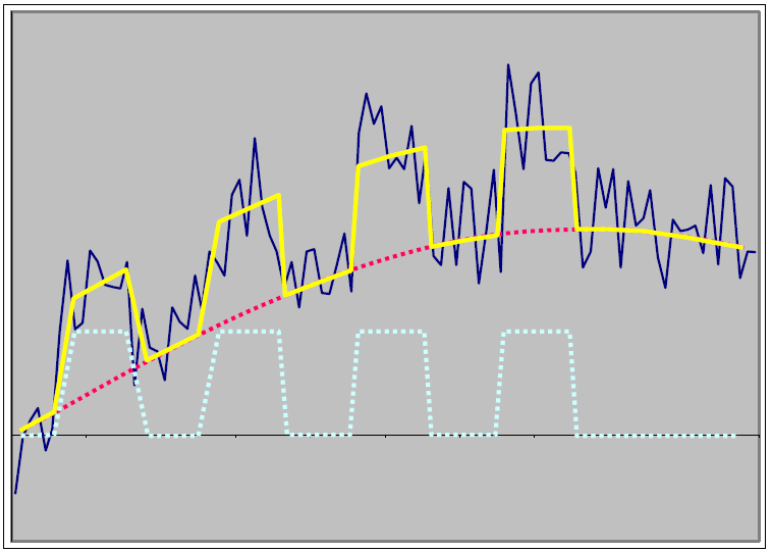


Can show that voxel is moved by stimulus – with enough stat data

Visual Stimulus (primary visual cortex)



General Linear Model



Magnet heats

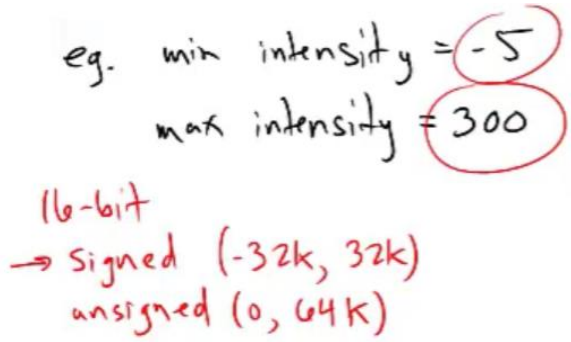
<_ stimulus on/off
Can separate signals with least squares

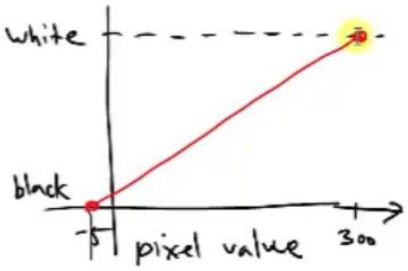
Unit 3: Image Enhancement - Lesson 19 – Contrast Enhancement

Goal: to create a clear understanding of how we can map image intensities to screen brightness

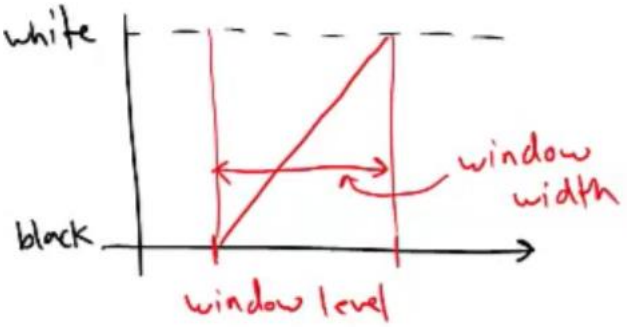
Contrast/Brightness

You have a choice of how to map image intensities to screen graylevels. MATLAB, by default, amps min to black and max to white.





In radiology circles, they often refer to this mapping as "**window width**" and "**window level**"



(ImageJ demo – ship brightness, contrast)

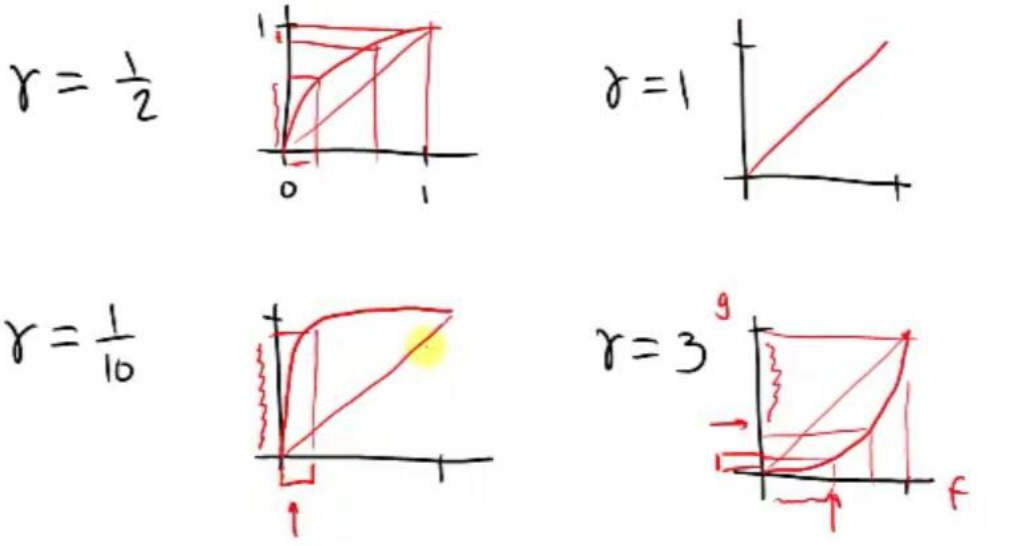
This is helpful to see structures that have similar intensities in an image that has a large intensity range.

Gamma Correction

A common way to map the image intensities is called gamma correction.

Given intensity f, remap to get g

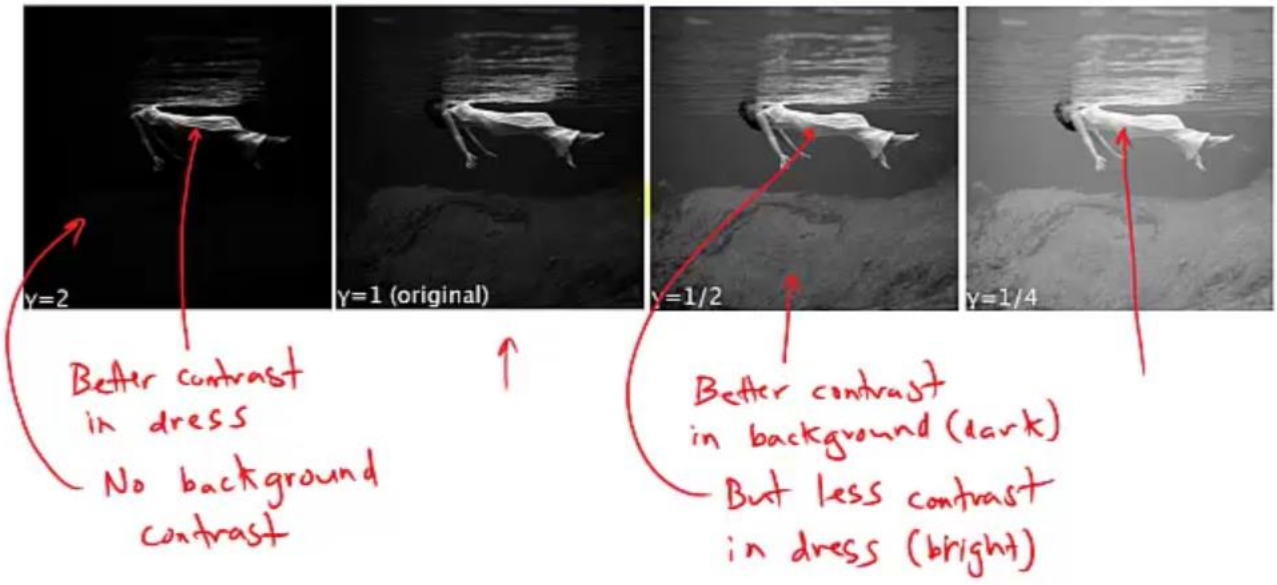
Examples:



Effect:

γ = ½ Enhances dark contrast at cost of bright contrast

γ = 2 Enhances bright contrast at cost of dark contrast



**gamma = ½: low intensity bright at cost of contrast

Gamma = ¼: bright stuff harder to see

Gamma = 2: more bright contrast, darks are harder

Gamma = 3: low intensities: black, high intensities: more spread out

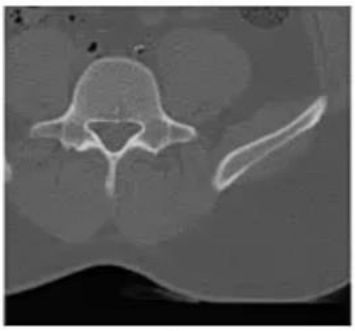
Contrast changes slop of line

Brightness changes window – horizontal translation of slope

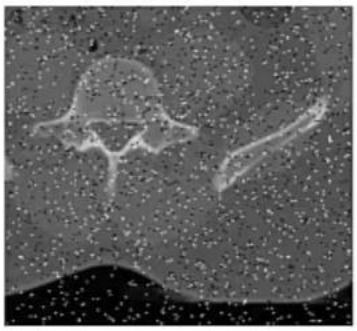
Lesson 20 – Denoising

Goal: to find out the types of compromises we use to try to remove noise from images.

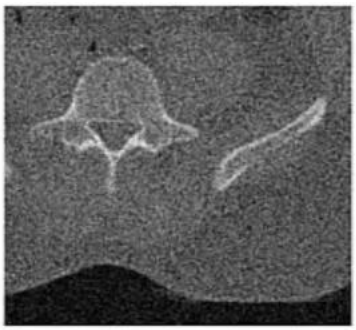
There are different types of noise in images.



Original
(8-bit image)
[0, 255]



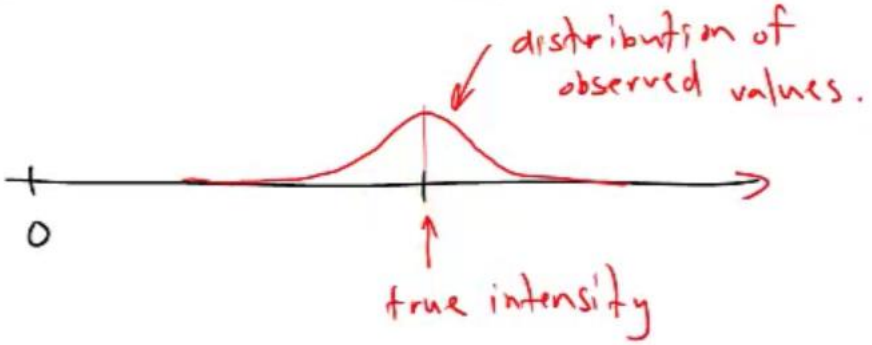
Speckle
("Salt and Pepper")
0 255



Gaussian (Normal)
Distributed
($\mu=0$, $\sigma=25$)

In a MRI, the acquired (raw) data is complex-valued. Both the real and imaginary parts have Gaussian noise. When you look at the noise in the magnitude of the reconstructed images, its distribution is called "**Rician**".

But for this course we will usually talk about Gaussian noise. Gaussian noise is typically additive.

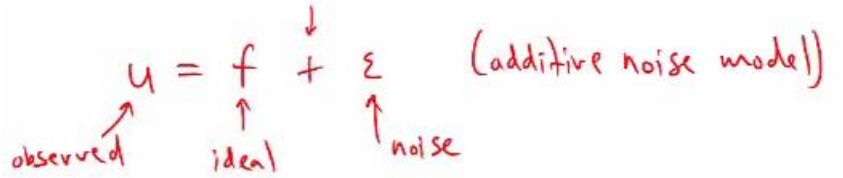


These random fluctuations are introduced into the images through a number of phenomena. Most notably:

- **Electrical noise in analog circuits**
- **Sample statistics (e.g. # of photons that hit an x-ray detector)**

Noise Removal

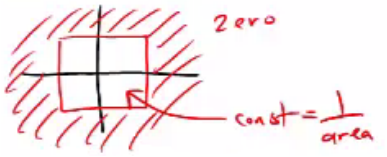
Let *f* be the true image (noise-free). What we actually observe is *u*, where



The task of noise removal is to approximate *f* given only *u* (but not ε, though you may know some statistical characteristics of ε, e.g. zero-mean Gaussian distributed)

Many approaches:

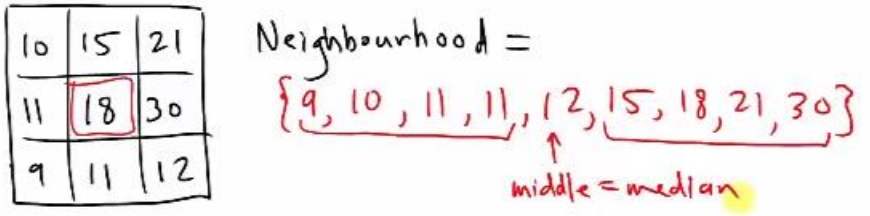
1. Local median filter is good at removing speckle
2. Windowed averaging: this is actually the same as convolving your noise image with a "window" (see below)
3. Gaussian smoothing/blurring
4. Wavelets (remove small-scale/low amplitude coefficients)
5. Non-local means



Median Filter

Look in a neighbourhood around a pixel and assign the median value.

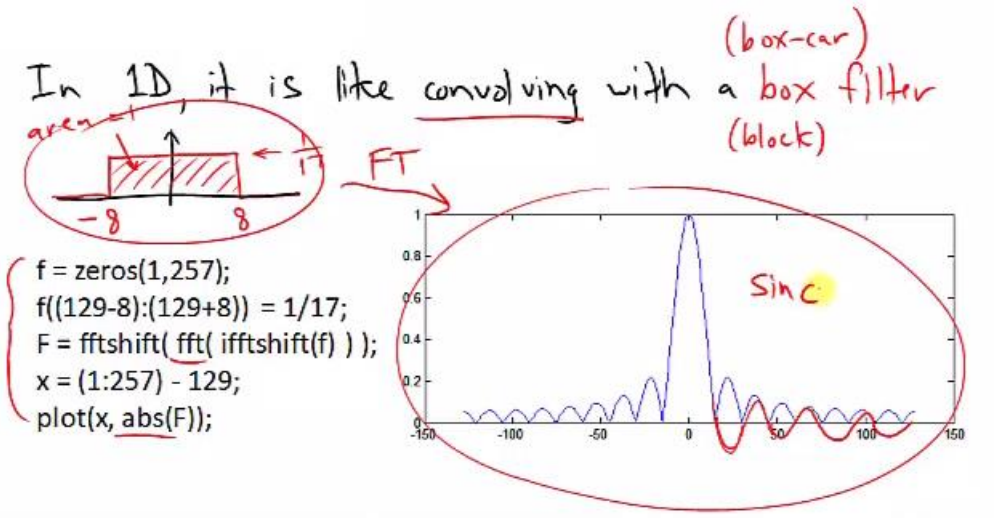
e.g



Therefore assign an intensity of **12** where the 18 is.

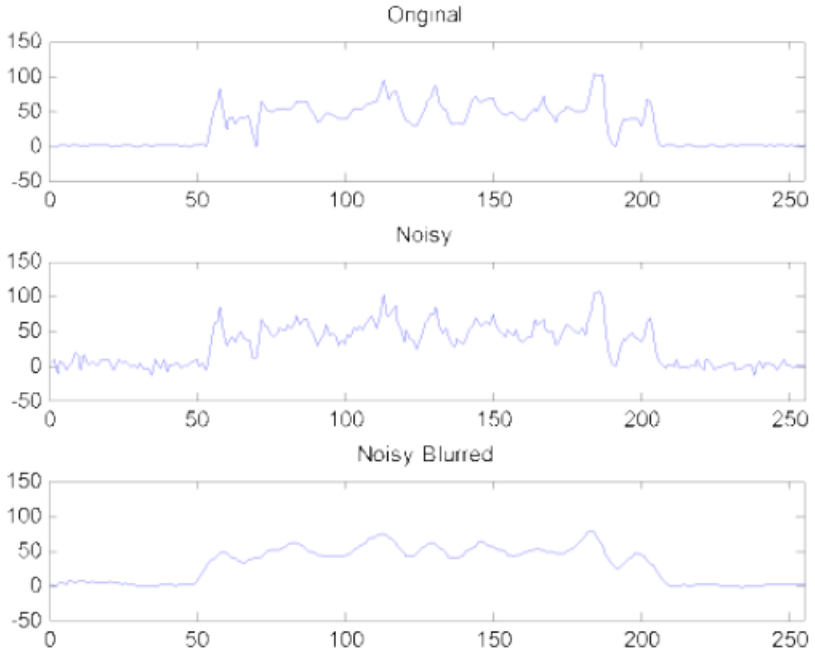
(Pelvis speckle demo) 15:00 – good for speckle noise

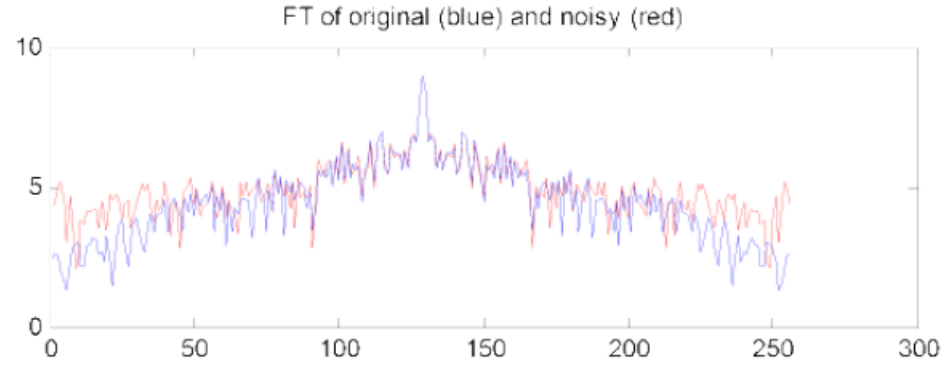
Windowed Averaging

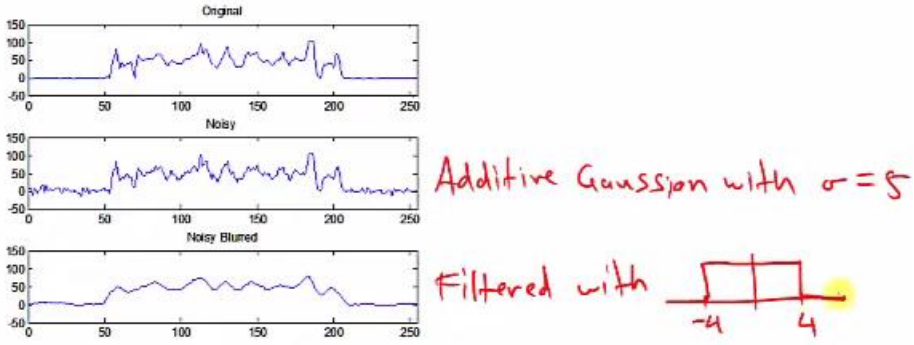


So windowed averaging is equivalent to multiplying the Fourier coefficients by the **sinc** function. Thus, the **low** frequencies are maintained, but the **high** frequencies are largely dampened.

L20_freq_of_noise.m

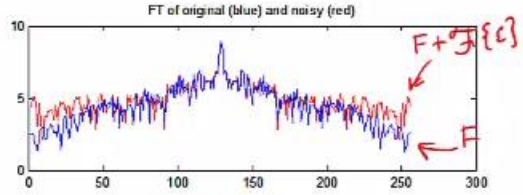






Additive Gaussian with $\sigma = 5$

Filtered with



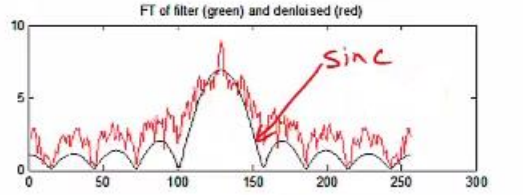
Note: FT is linear

$F = \mathcal{F}\{f\}$

$G = \mathcal{F}\{g\}$

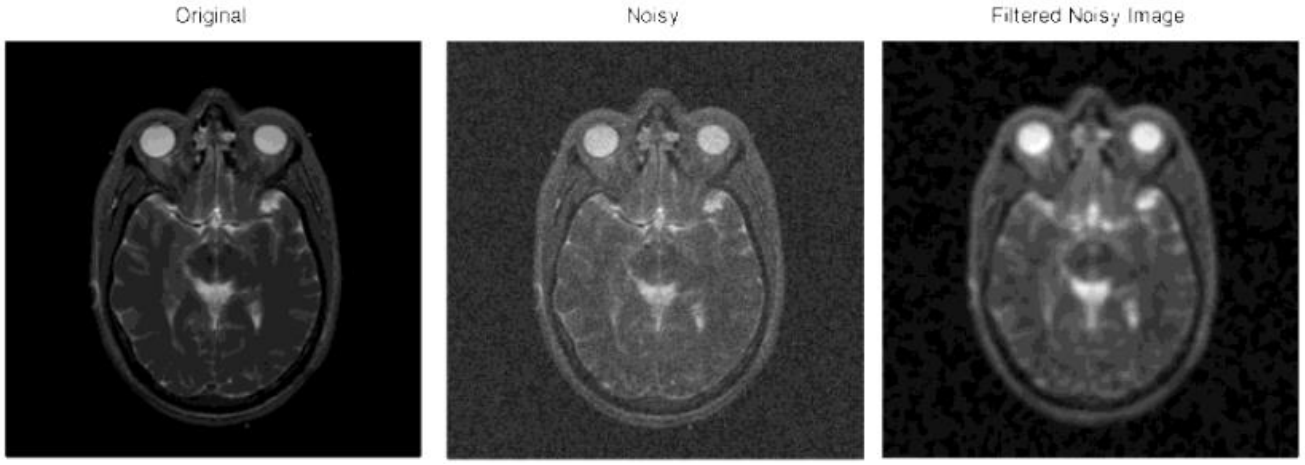
$\alpha F = \mathcal{F}\{\alpha f\}$

$F+G = \mathcal{F}\{f+g\}$

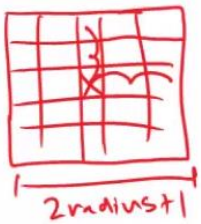


Demo part 2: want to dampen the noise
Fourier: double in, double out

How about on an image?

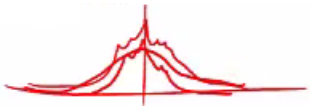


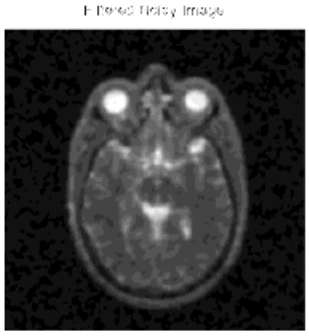
8-bit **σ = 15** **Filtered – box filter with radius 2**



Gaussian Smoothing

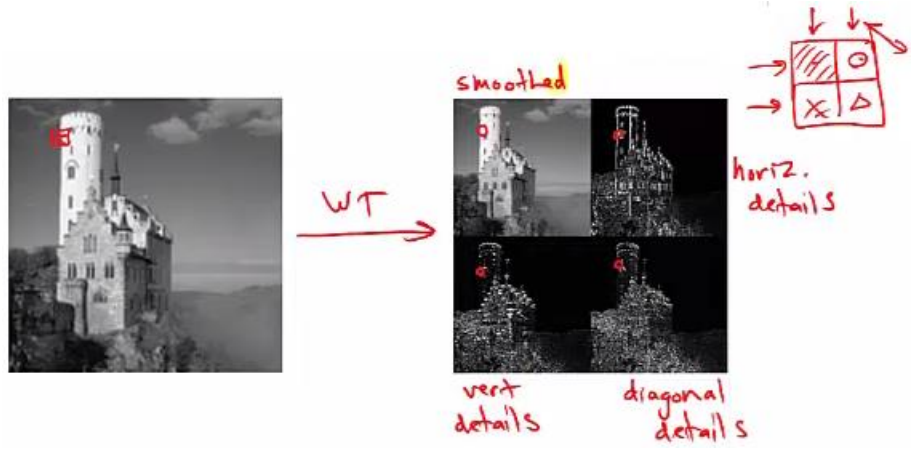
Similar to windowed averaging above, but using a Gaussian kernel. The FT of a Gaussian is a Gaussian, so the high frequencies are all but gone.



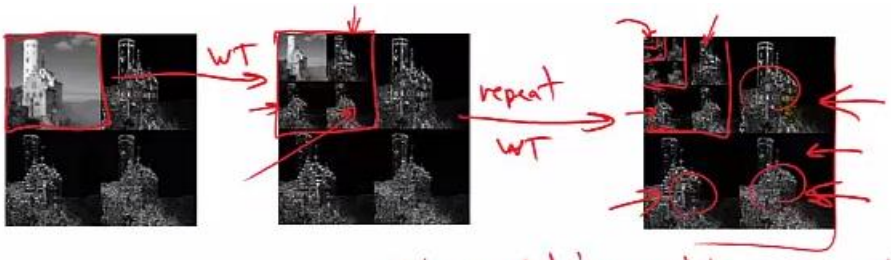


Wavelets

Unfortunately, wavelets are not part of this course. Briefly, it is a transform that applies smoothing along rows and columns, and records both the smoothed values, as well as the details that the smoothing removed.



Repeat the process on the smoothed version.



These details contain the small-scale (high-freq) info, & also a lot of the noise. Setting these coefs to zero is a noise-removal strategy.

Non-Local Means

Replace each pixel in the image with a weighted average of other pixels in similar neighbourhoods.

PSNR: peak signal-to-noise ratio. Bigger = better

$$\begin{aligned}\}_n &= \sum_{j=0}^{N-1}\left(\sum_{k=0}^{N-1} \mathbf{f}_k\, \mathbf{g}_{j-k}\right) e^{-2\pi i \frac{jn}{N}} \\ &= \sum_{j=0}^{N-1} \mathbf{f}_k\, e^{-2\pi i \frac{kn}{N}} \sum_{k=0}^{N-1} \mathbf{g}_{j-k}\, e^{-2\pi i \frac{n}{N}(j-k)} \\ &= \hat{\mathbf{f}}_n\, \hat{\mathbf{g}}_n\ .\end{aligned}$$

e above derivation assumes that f and g are

(a) Test image

$$\begin{aligned}\}_n &= \sum_{j=0}^{N-1}\left(\sum_{k=0}^{N-1} \mathbf{f}_k\, \mathbf{g}_{j-k}\right) e^{-2\pi i \frac{jn}{N}} \\ &= \sum_{j=0}^{N-1} \mathbf{f}_k\, e^{-2\pi i \frac{kn}{N}} \sum_{k=0}^{N-1} \mathbf{g}_{j-k}\, e^{-2\pi i \frac{n}{N}(j-k)} \\ &= \hat{\mathbf{f}}_n\, \hat{\mathbf{g}}_n\ .\end{aligned}$$

e above derivation assumes that f and g are

(b) Noisy, σ=11, PSNR=27.0

$$e^{-2\pi i \frac{kn}{N}} \sum$$

Original

$$e^{-2\pi i \frac{kn}{N}} \sum$$

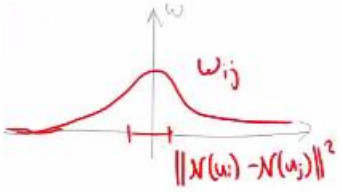
Nois

$$e^{-2\pi i \frac{kn}{N}} \sum$$

Denoised

PSNR = 27 dB **PSNR = 33.7 DB**

Recall: $u = f + \varepsilon$ (noisy image, ideal image, noise)



We replace u_i with $\tilde{u}_i$

$$\tilde{u}_i = \sum_j w_{ij}\, u_j$$ (weight of pixel j on new pixel i)

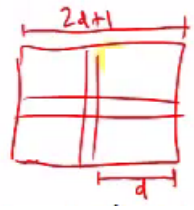
$$w_{ij} = \frac{1}{W_i} \exp\left(\frac{-1}{h^2}\|N(u_i) - N(u_j)\|^2\right)$$

$W_i = \sum_j w_{ij}$; h = parameter

To denoise a single pixel *i*, compare **$N(u_i)$** to all other neighbourhoods **$N(u_i)$**.
For a NxN image, this requires **$O(N^2)$** neighbourhood comparisons.
But there are **N^2** pixels to denoise **=> $O(N^4)$**

Suppose each neighbourhood has a radius of **d**, so contains **(2d + 1) x (2x + 1) pixels**

- ⇨ NL-Means is $O(N^4d^2)$

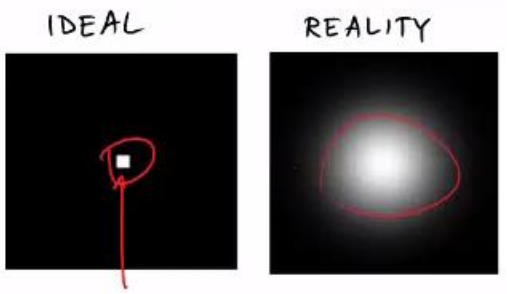


Lesson 21 – Deblurring

Goal: to demonstrate the fundamental difficulties with deblurring, but learn some techniques that we can still use to make some progress.

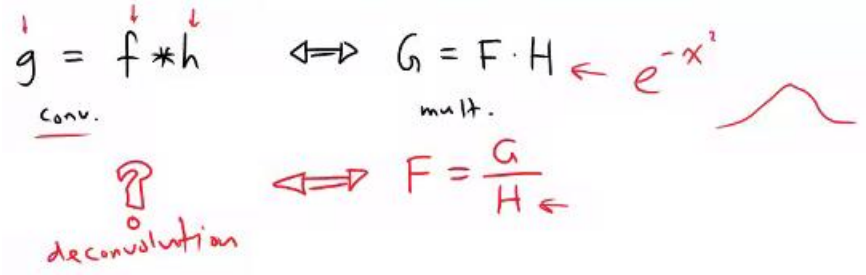
Why deblurring in medical imaging?
e.g. CT: Rayleigh scattering results in a "diffusion" of the x-ray beam.

As a result, a tightly-packed beam gets spread.

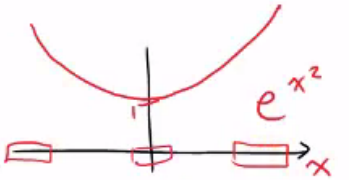


This blurred version of an ideal beam is called a "**point-spread function**" and can be thought of as a blurring kernel.

If convolving by a Gaussian blurs an image, it stands to reason that **deconvolving** deblurs an image.

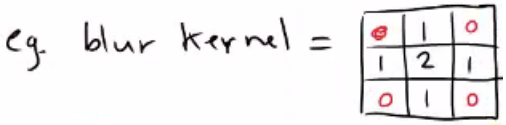


If **h** is a Gaussian, then so is **H**. and 1/H is something like this:



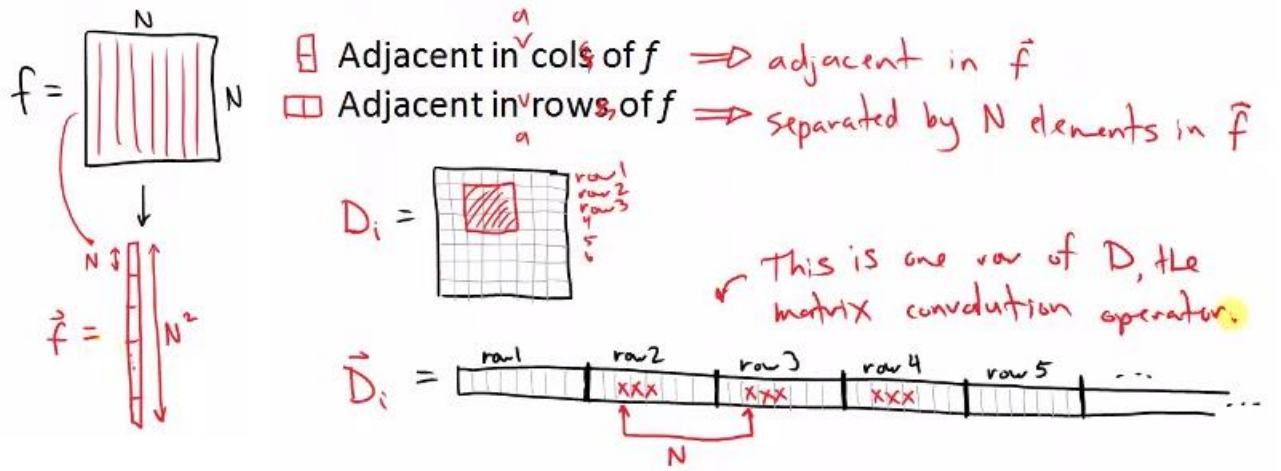
This <u>de-weights</u> the low frequencies and <u>accentuates</u> the high frequencies.
(deconvolution demo)
A problem arises when one of the Fourier coefficients of ***H*** is zero, or near zero. In such cases, and in general, deblurring is **ill-conditioned**. The resulting "deblurred" image is extremely sensitive to small perturbations. Hence, any noise in the blurry image will be <u>blown</u> up when you try to deblur it.

Another view: we can represent convolution operation as a matrix operator.

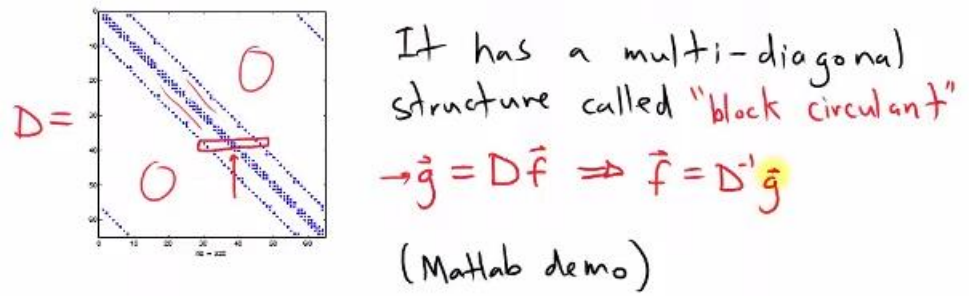


To deal with images using matrix operators, we convert the 2D image to a 1D column vector,

Placing the kernel on the image then involve pixels that are adjacent in the column, and adjacent in rows.



The matrix containing the appropriate weights is the convolution operator. We're going to use it as a blurring operator, so it is a blurring matrix.

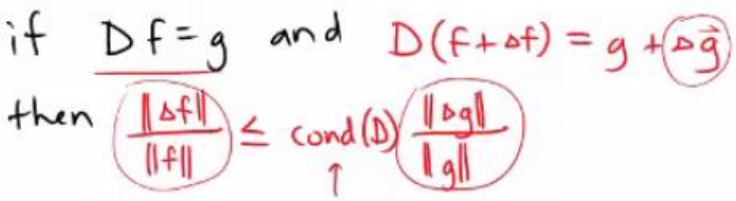


This matrix is ill-conditioned, which means its condition number is large.

$$cond(D) = K(D) = \|D\| * \|D^{-1}\| < -big\ if\ D\ nearly\ singular$$

If D is singular (no inverse) => bad

The condition number gives you an idea of how perturbations in your **blurred** image translate to perturbations in your **deblurred** image. In particular,



That is, the relative perturbation in the blurred image can be multiplied by as much as **cond(D)**. Thus, deblurring using $\boldsymbol{f = D^{-1}g}$ is not a good idea.

*cannot escape matrix ill-conditioned problem by going to the frequency domain

Instead, consider the "Reblurring" approach.

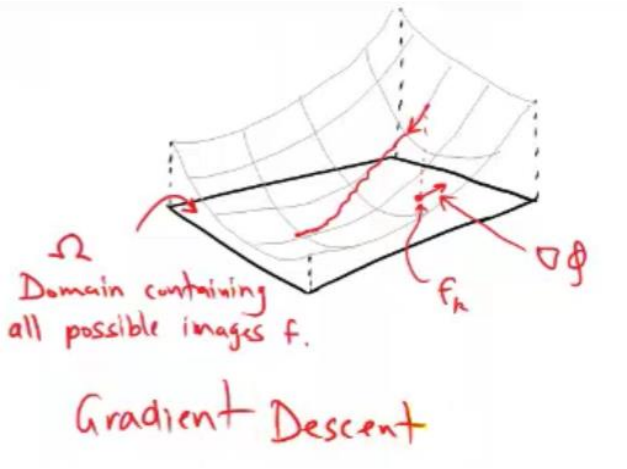
In other words, instead of trying to invert the blurring operation, we try to find an image $\tilde{f}$ that, when blurred, looks like our original blurry image, **g**. **could be many f

To minimize this, we can use an iterative gradient descent method. We want to find the vector $\tilde{f}\epsilon\Omega$ that minimizes $\varphi(\tilde{f})$. The gradient of **ϕ**, denoted **∇φ**, is a vector in **Ω** that points in the direction of greatest increase of **ϕ**.

Gradient descent means take a step from where you are in the direction opposite the gradient.

$$f_{k+1} = f_k - \beta \nabla \varphi(f_k)$$

(β: related to step size)



But how do we compute $\nabla\varphi$?

$$\varphi(f) = \|g - Df\|^2 = (g - Df)^T(g - Df)$$
$$= (g^T - f^T D^T)(g - Df)$$
$$= g^T g - g^T D^T f - f^T D^T g + f^T D^T Df$$

All scalars, so middle two are equal (transposes of each other)

$$= \|g\|^2 - 2f^T D^T g + f^T D^T Df$$

$\Rightarrow \nabla\varphi(f) = 2D^T g + 2D^T Df$ taking derivatives of f

$$= -2D^T(g - DF)$$

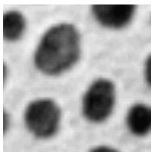
Therefore $f_{k+1} = f_k + \beta D^T(g - Df), f_0 = 0$

Hi

Lesson 22 – Edge Detection

Goal: to lay the foundation for detecting and using the edges in image content.

Edges in images are boundaries between regions with different intensities.

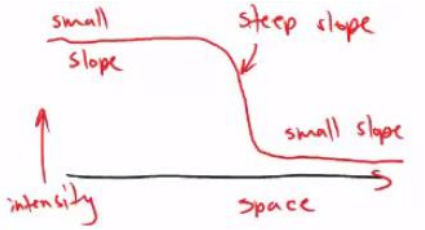
e.g. 

There are many reasons to want to locate the edges.

- Delineate organ boundaries
- Align images

Edges are also used in other image-processing operations (anisotropic diffusion, level-set segmentation, etc.)

Anatomy of an Edge



An edge is indicated by rapid change in intensity

- Gradient vector with a large magnitude.

So if we compute the gradient, we can get the edges.

Approximate Image Derivatives

Recall, for $f(x, y, z)$, $\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \\ \frac{\partial f}{\partial z} \end{bmatrix}$ gradient is a vector

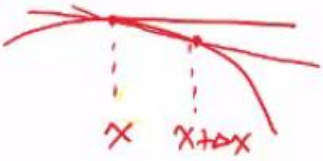
By definition,

$$\frac{\partial f}{\partial x} = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

If we approximate $f(x+\Delta x)$ using Taylor's formula,

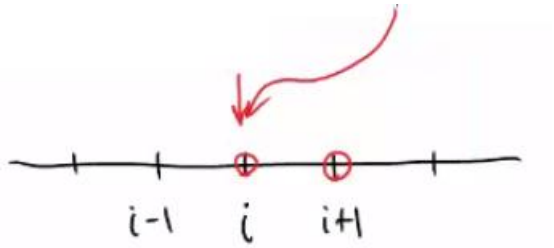
$$f(x+\Delta x) = f(x) + \frac{\partial f}{\partial x}\Delta x + \cancel{O(\Delta x^2)}$$

$$\Rightarrow \frac{\partial f}{\partial x} \approx \frac{f(x+\Delta x) - f(x)}{\Delta x}$$



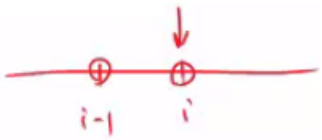
This is known as **forward differencing**. It involves the pixel where the derivative is desired, as well as the pixel in the positive direction.

$$\frac{\partial f_i}{\partial x} \approx f_{i+1} - f_i$$



The opposite is called **backward differencing**. It involves the pixel where the derivative is desired, as well as the pixel in the positive direction.

$$\frac{\partial f_i}{\partial x} \approx f_i - f_{i-1}$$



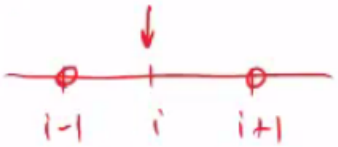
Central differencing: start with Taylor's for both

$$f(x+\Delta x) = f(x) + \frac{\partial f}{\partial x}\Delta x + O(\Delta x^2)$$

$$-\left[f(x-\Delta x) = f(x) - \frac{\partial f}{\partial x}\Delta x + O(\Delta x^2)\right]$$

$$f(x+\Delta x) - f(x-\Delta x) = 2\Delta x \frac{\partial f}{\partial x} + O(\Delta x^2)$$

$$\Rightarrow \frac{\partial f}{\partial x} \approx \frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x}$$



$$\text{or } \frac{\partial f_i}{\partial x} \approx \frac{f_{i+1} - f_{i-1}}{2}$$

Central differencing is **O(Δx²)** accurate, while forward and backward differencing are both **O(Δx).**

Image Gradient

For each pixel, compute its gradient (…)

for each pixel, compute its gradient $\left(\frac{\partial f_{ij}}{\partial r}, \frac{\partial f_{ij}}{\partial c}\right)$

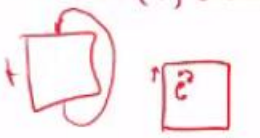

```
f = imread('t1.jpg');
f = double( f(:,:,1) );
① imshow(f,[])

dfdr = ( circshift(f,[-1 0]) - circshift(f,[1 0]) ) / 2;
② imshow(dfdr,[])

③ dfdc = ( circshift(f,[0 -1]) - circshift(f,[0 1]) ) / 2;
imshow(dfdc,[])

④ grad_mag = sqrt(dfdr.^2 + dfdc.^2);
imshow(grad_mag, [])
```

circshift(f, [down right])



① f ② $\frac{\partial f}{\partial r}$ ③ $\frac{\partial f}{\partial c}$ ④ $\|\nabla f\|$

Lesson 23 – Derivatives and Fourier Theory

Goal: to find out how image derivatives relate to the Fourier transform.

Recall the inverse Fourier transform:

$$f(x) = \int F(\omega) e^{2\pi i \omega x} d\omega$$

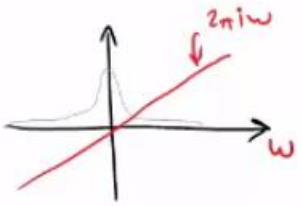
Now take its derivative:

$$\frac{df(x)}{dx} = \int F(\omega)(2\pi i \omega) e^{2\pi i \omega x} d\omega$$

$$= \int (2\pi i \omega F(\omega)) e^{2\pi i \omega x} d\omega$$

Hence, another way to compute the derivative is by manipulating the FT by

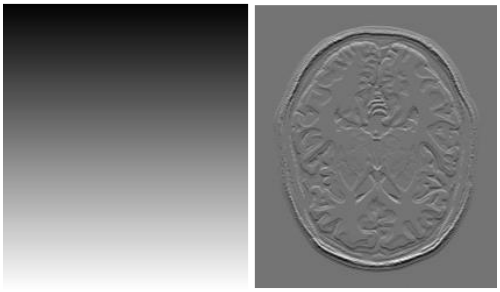
$$\frac{df(x)}{dx} = \mathcal{F}^{-1}\{2\pi i \omega F(\omega)\}$$



This can be approximated on discrete signals.

$$\frac{df(x_n)}{dx} \approx \text{IDFT}\left\{2\pi i \frac{k}{N} F_k\right\}_n$$

Example: MATLAB demo



Now consider the discrete case: Central differencing

$$\frac{\partial f_n}{\partial x} = (f_{n+1} - f_{n-1})\frac{1}{2}$$

Because the FT is linear

$$\mathcal{F}\left\{\frac{\partial f_n}{\partial x}\right\} = \left[\mathcal{F}\{f_{n+1}\} - \mathcal{F}\{f_{n-1}\}\right]\frac{1}{2}$$

$$= \left[e^{\frac{2\pi i k}{N}} F_k - e^{\frac{-2\pi i k}{N}} F_k\right]\frac{1}{2}$$

$$= \frac{F_k}{2}\left(e^{\frac{2\pi i k}{N}} - e^{\frac{-2\pi i k}{N}}\right) \quad \text{Let } \alpha = \frac{2\pi k}{N}$$

$$= \frac{F_k}{2}\left(\cos\alpha + i\sin\alpha - \cos\alpha + i\sin\alpha\right)$$

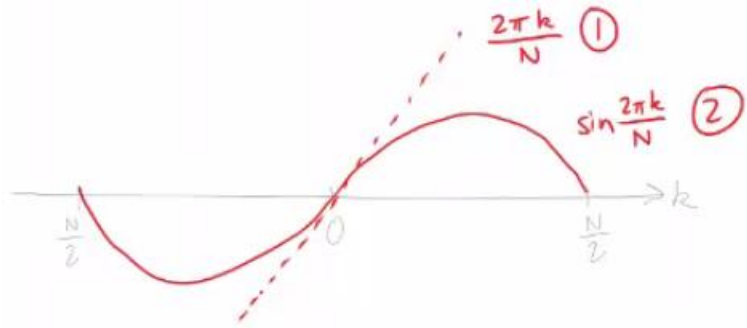
$$= i\sin\frac{2\pi k}{N} F_k$$

Summary of Derivatives and the Fourier Transform: continuous derivative -> discrete approximation

$$\text{DFT}\left\{\frac{df(x_n)}{dx}\right\}_k \approx 2\pi i \frac{k}{N} F_k \quad (1)$$

Discrete approx of derivative

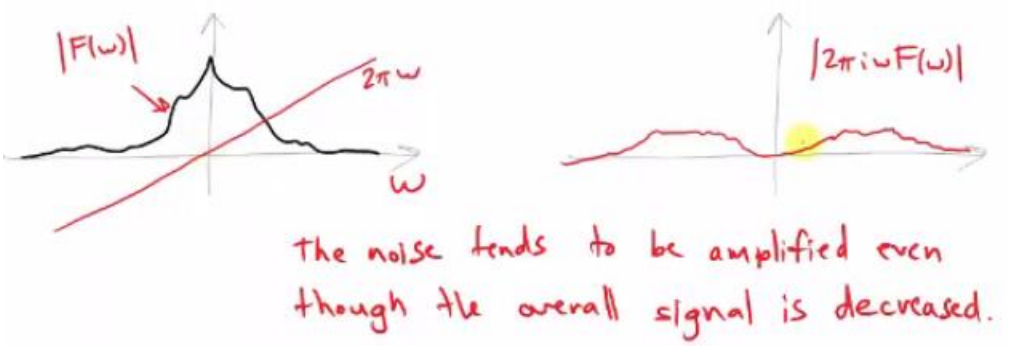
$$\text{DFT}\left\{\frac{df(x_n)}{dx}\right\}_k \approx i\sin\frac{2\pi k}{N} F_k \quad (2)$$



Can you guess how noise will impact $\frac{df(x)}{d(x)}$?

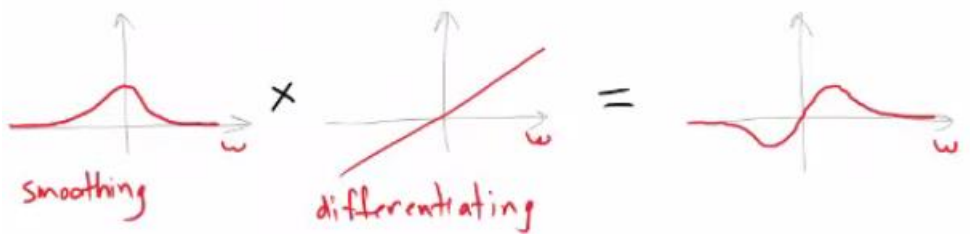
Consider $F(\omega) = \mathcal{F}\{f(x)\}(\omega)$.

The derivative is like a high-pass filter, emphasizing the high frequencies, and de-emphasizing the low frequencies.

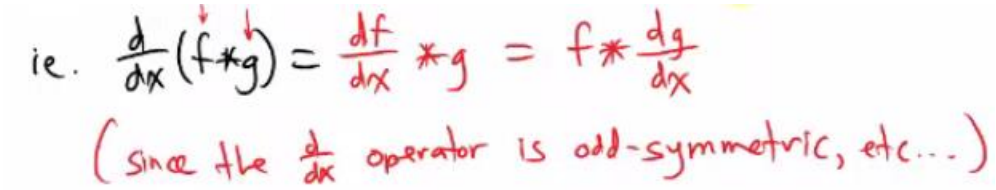


The derivative is like a high-pass filter, emphasizing the high frequencies, and de-emphasizing the low frequencies. For this reason, people often smooth the image before taking the derivative.

If we smooth with a Gaussian, then take the derivative:



Because both smoothing by convolution and differentiation are linear operations, the order they are applied does not matter.



Both can be applied by convolving f with $\frac{dg}{dx}$.

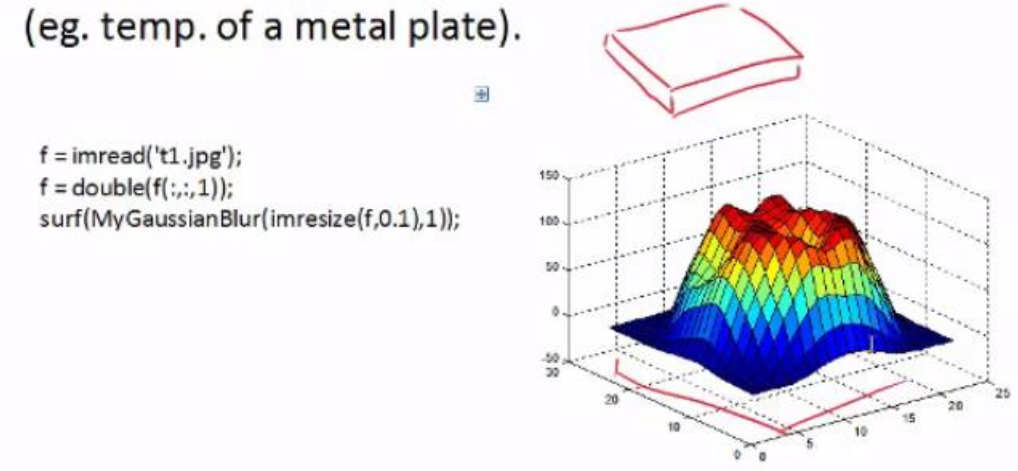
(See MATLAB demo)

Lesson 24 – Anisotropic Diffusion

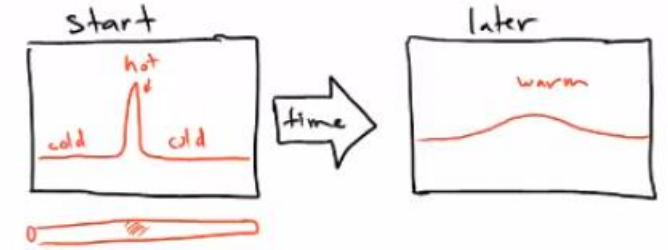
Goal: to find out how edges can be used to help in image denoising.

First thing's first: what is "diffusion" in the context of image processing?

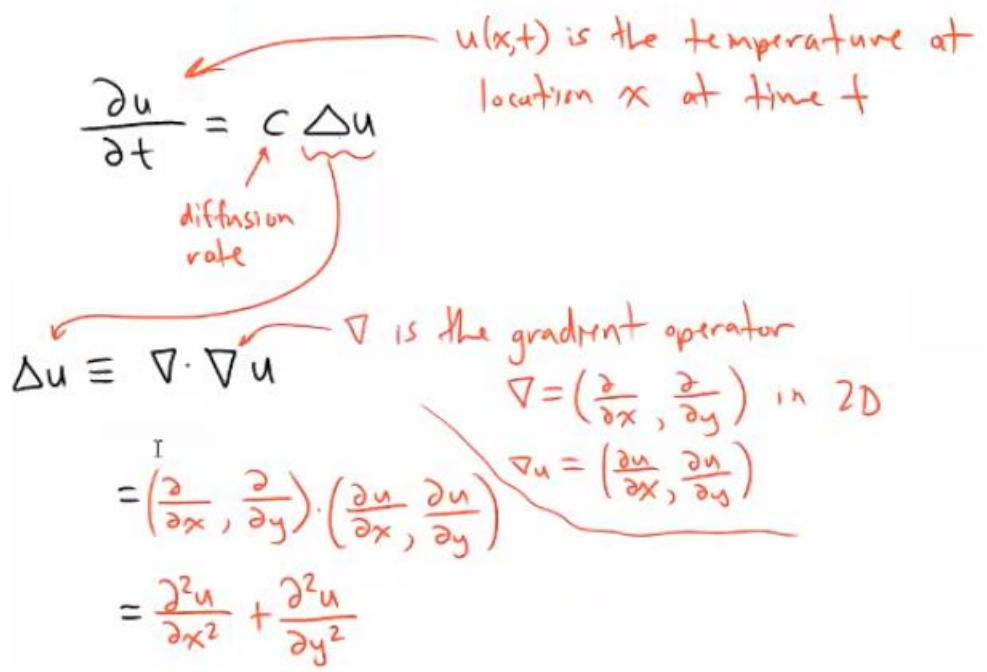
Think of an image as a function indicating the temperature distribution throughout a material (e.g. temperature of a metal plate)



It's not hard to imagine the spreading of that heat over time.

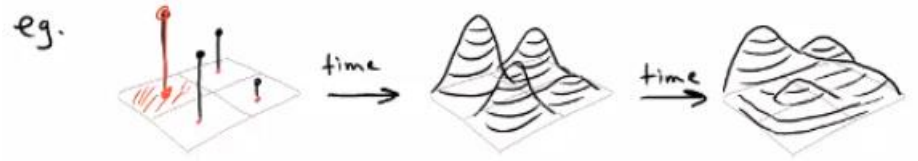


Heat diffuses over time. In particular, its time dynamics are modelled by the "heat equation", a partial differential equation (PDE),



As it turns out, if you start with a single spike of heat (and no heat elsewhere), then the solution is a time progression of Gaussian functions, with larger and larger σ as time goes on.

Any heat distribution can be represented as a combination of heat spikes. Since the differential operators are linear, the evolution of the entire heat distribution is the superposition of a bunch of spreading Gaussian functions.



Cold ^

This is equivalent to convolving with a Gaussian kernel.

⇨ **Diffusion = Blurring**

Hence, one way to blur an image is to let it evolve using the heat equation. (LHS: FD affine derivative)

$$\frac{\partial f}{\partial t} = c\,\Delta f$$

forward differencing

$$\frac{f_{mn}^{t+1} - f_{mn}^{t}}{\Delta t}$$

$$\therefore f_{mn}^{t+1} = f_{mn}^{t} + \Delta t\, c\, \Delta f$$

$$\frac{\partial}{\partial x}\left(\frac{\partial f}{\partial x}\right) + \frac{\partial}{\partial y}\left(\frac{\partial f}{\partial y}\right)$$

$$= \frac{\partial}{\partial x}\left[\frac{f_{m+1,n}^{t} - f_{m-1,n}^{t}}{2\Delta x}\right] + \frac{\partial}{\partial y}\left[\frac{f_{m,n+1}^{t} - f_{m,n-1}}{2\Delta y}\right]$$

$$= \frac{\left[\frac{f_{m+2,n}^{t} - f_{mn}^{t}}{2\Delta x}\right] - \left[\frac{f_{m+1,n}^{t} - f_{m-1,n}^{t}}{2\Delta x}\right]}{2\Delta x} + \cdots$$

$$= \frac{f_{m+2,n}^{t} - 2f_{mn}^{t} + f_{m-2,n}^{t}}{(2\Delta x)^2} + \frac{f_{m,n+2}^{t} - 2f_{mn}^{t} + f_{m,n-2}^{t}}{(2\Delta y)^2}$$

$\frac{\partial^2 f}{\partial x^2}$ $\frac{\partial^2 f}{\partial y^2}$

Apply central differencing twice in sequence.
This is the hard way.

Alternatively, we could go back to Taylor's theorem:

$$f(x+\Delta x) = f(x) + \frac{\partial f}{\partial x}\Delta x + \frac{\partial^2 f}{\partial x^2}\frac{\Delta x^2}{2} + O(\Delta x^3)$$

$$+\ f(x-\Delta x) = f(x) - \frac{\partial f}{\partial x}\Delta x + \frac{\partial^2 f}{\partial x^2}\frac{\Delta x^2}{2} + O(\Delta x^3)$$

$$f(x+\Delta x) + f(x-\Delta x) = 2f(x) + \frac{\partial^2 f}{\partial x^2}\Delta x^2 + O(\Delta x^3)$$

$$\Rightarrow \frac{\partial^2 f}{\partial x^2} \approx \frac{f(x+\Delta x) - 2f(x) + f(x-\Delta x)}{\Delta x^2}$$

This is the same as applying central differencing twice, but using Δx that is half as big.

(demo: L24 – diffusion blurring)

Anisotropic Diffusion

The diffusion process described above is isotropic. (change blurring spatially and direction)

American Heritage Dictionary – *Cite This Source*

i·so·tro·pic (ī'sə-trō'pĭk, -trŏp'ĭk) Pronunciation Key

adj. Identical in all directions; invariant with respect to direction.

One of the problems is that, while it filters out noise, it also blurs the edges of the images, thereby reducing the resolution. To get around this, we can make the rate of diffusion vary spatially, so that diffusion is slower across edges.

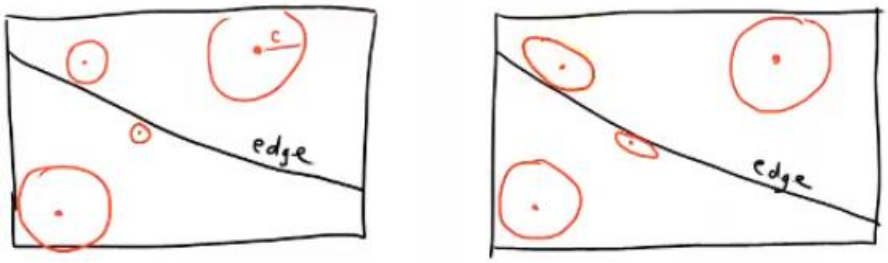
The idea behind anisotropic diffusion (AD) is to let the diffusion-rate constant *c* be a function of space so that

$$c(x) = \begin{cases} \text{large in homogeneous regions} \\ \text{small near edges} \end{cases}$$

We have two ways of using this spatially-varying *c*.

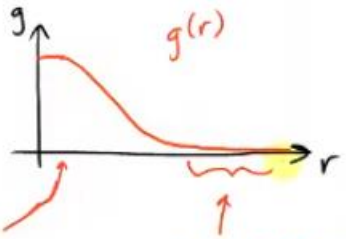
Less diffusion near an edge **not circle blurring – parallel about edge**

$$\frac{\partial f}{\partial t} = c(x)\,\Delta f \qquad \frac{\partial f}{\partial t} = \nabla \cdot (c(x)\,\nabla f)$$



We use the gradient magnitude to construct c(x). In particular, we will use

$$c(x) = g(\|\nabla f\|)$$



when $\|\nabla f\|$ is small, then diffusion is fast

when $\|\nabla f\|$ is large (near an edge), then diffusion is slow.

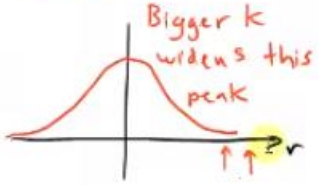
Replacing $c(x)$ with $g(\|\nabla f\|)$,

$$\frac{\partial f}{\partial t} = g(\|\nabla f\|)\,\Delta f \quad \text{or} \quad \frac{\partial f}{\partial t} = \nabla \cdot (g(\|\nabla f\|)\,\nabla f)$$

$\nabla \cdot g \nabla f + g \nabla \cdot \nabla f$

Some common g functions:

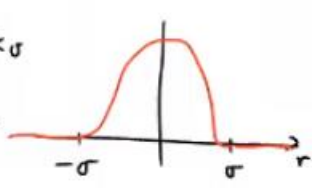
Perona & Malik's Lorentzian function $g(r) = \dfrac{1}{1 + \frac{r^2}{k^2}}$



allows more edge blur

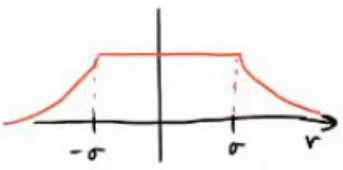
Tukey's Biweight

$$g(r) = \begin{cases} \frac{1}{2}\left(1-\left(\frac{r}{\sigma}\right)^2\right)^2 & \|r\| < \sigma \\ 0 & \text{otherwise} \end{cases}$$

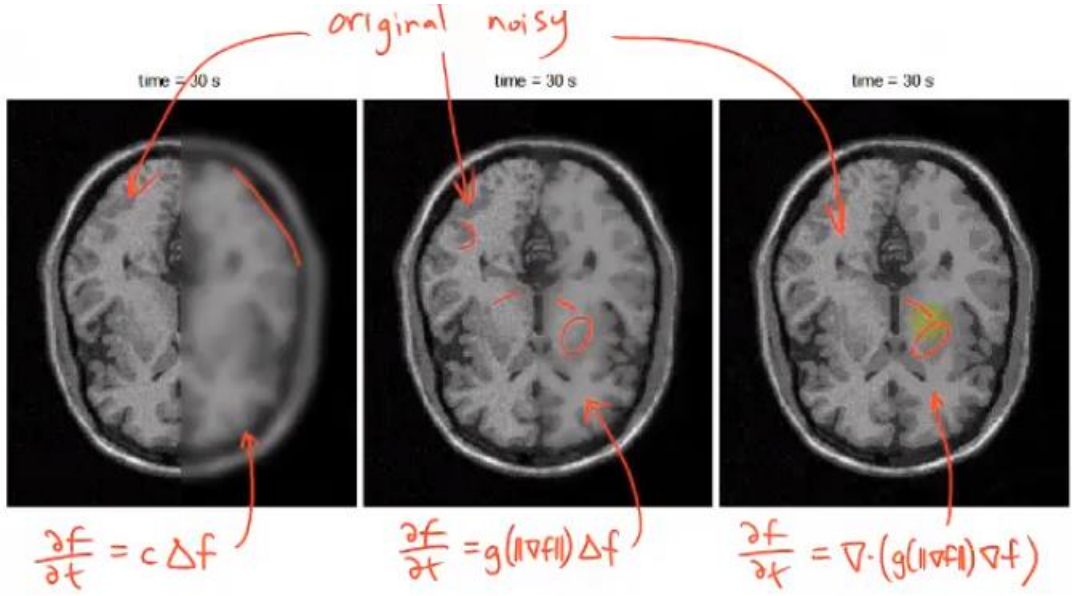


Huber's Minimax Estimator

$$g(r) = \begin{cases} \frac{1}{\sigma} & |r| \le \sigma \\ \frac{\text{sign}(r)}{r} & |r| > \sigma \end{cases}$$



Samples (MATLAB demo)



Unit 4: Image Registration - Lesson 25 – Introduction to Image Registration

Goal: to define what registration is, and survey some methods.

Formulation

You have two images (or volumes) that you wish to register (align) so that they correspond on a pixel-by-pixel basis. Let the two images be *f* and *g*.

How does one go about registering *f* and *g*?

How to move *f* into alignment with *g*?

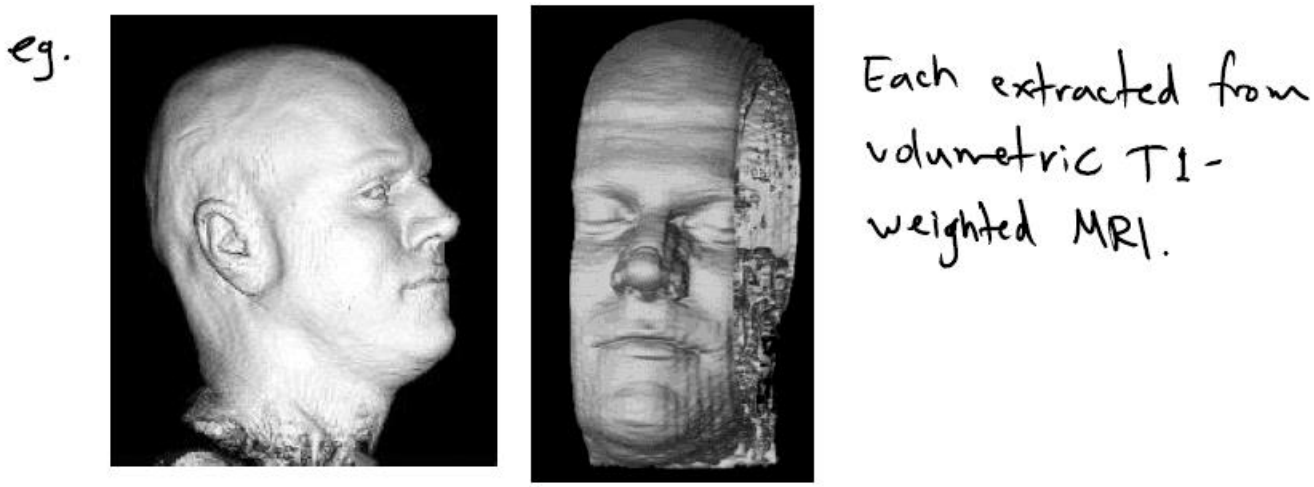
Point-Based

Someone who knows the anatomy can mark corresponding locations in both images and use those points to derive the transformation that moves *f* into alignment with *g*.

The problem is that an expert is needed, and it can be time-consuming. Also, content far from these matched points can have large registration errors.

Surface-Based

One can extract an iso-surface from each image (volume) and try to register the surfaces.

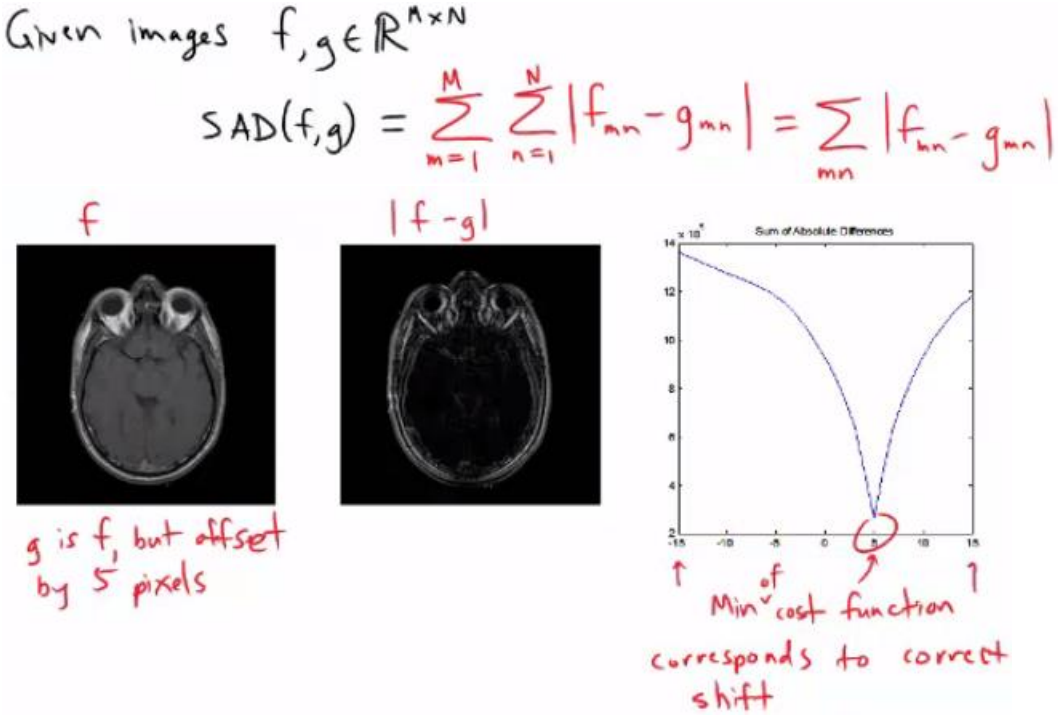


The problem is that it can be difficult to automatically extract the same surface, from both volumes.

Intensity-Based

We can derive a "goodness-of-fit" value by comparing two images on a pixel-by-pixel basis. This measure is called a **cost function** or **objective function**, and is used to gauge registration quality.

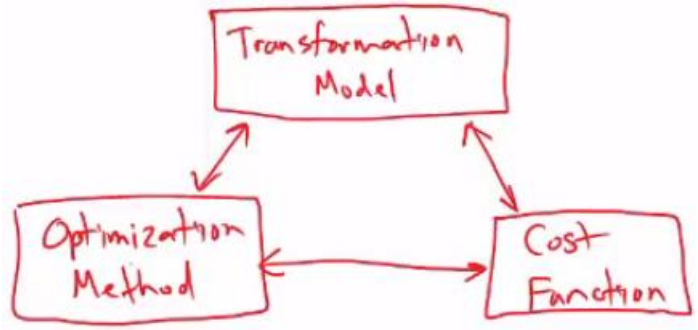
For example: Sum of Absolute Differences (SAD)



(L25 intro demo)

Registration Process

Each registration process involves an interplay between three main components:



Each choice can potentially influence the others. We will look at a number of combinations.

In general:

Let **T** represent a motion transformation

- E.g. translation only, rotation only, rigid-body, affine, non-affine, etc.
- We represent a transformed image as **Tf** . T(f)
  In this context, I call **T** an **image** operator because it applies the intended transform to the image itself.
  Alternatively, we could use the notation **f(Mx)** where M is a **coordinate** operator, applied to the pixel locations.
  Note that $\mathbf{T^{-1} \Leftrightarrow M}$ **i.e.** $\mathbf{(Tf)(x) = f(M^{-1}x)}$
  Sometimes it is helpful to be explicit about the dependence of the transform on motion parameter,
  e.g. $\mathbf{m = [\Delta row, \Delta col, \Delta\Theta]^T}$
  i.e. **T** is a function of **m**, but operates on **f**.

Let d(f, g) be a cost function that quantifies the registration quality.

e.g.

$$\boldsymbol{d(f,g) = \sum_n \sum_n |f_{mn} - g_{mn}|}$$ for SAD

Then the registration problem boils down to the optimization problem:

$$m = \underset{m}{\operatorname{argmin}}\, d(T(m)f, g)$$

Or

$$m = \underset{m}{\operatorname{argmax}}\, d(T(m)f, g)$$

$$\boldsymbol{m = \underset{m}{\operatorname{argmin}}\, d(f, T^{-1}(m)g), f\ forwards = g\ inverse}$$

Depending on the cost function. For example, we minimize SAD, but maximize correlation (later). But our goal is to find the motion parameters **m** that achieve that optimal value.

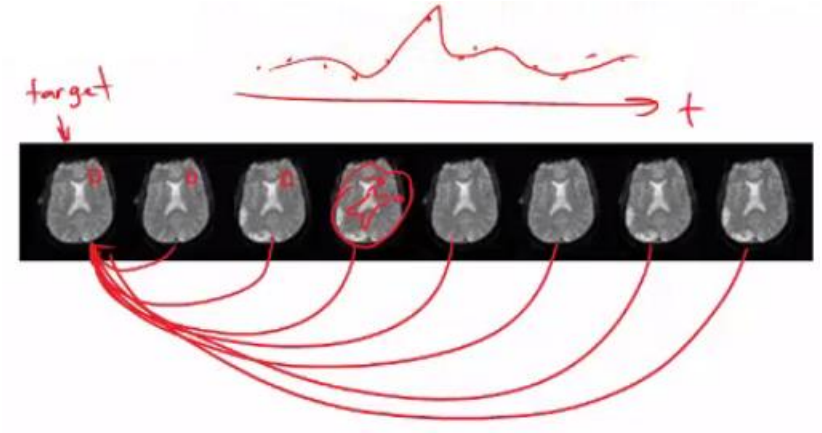
➔ Want to know: what is the transformation that aligns them?

Lesson 26 – Correlation

Goal: investigate correlation as a registration objective function, and how it can be computed.

Generally, this family of methods and cost functions is geared toward monomodal registration (aligning images of the same modality, e.g. CT to CT)

A good example is motion correction for functional MRI (FMRI). A series of MRI snapshots are taken, yielding a whole time series for each pixel. These pixel time series are statistically analyzed. However, if the snapshots are not properly aligned, the pixel time series will be mixed and disrupted. This is a monomodal registration scenario. Typically, one snapshot is chosen as the "fixed" or "reference" or "target" image, and all other images are registered to it.

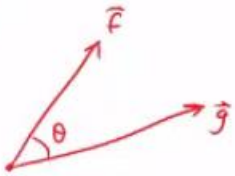


**fourth image on left: rotated, so pixels have moved – something moved into original pixel location – outlier in curve

A typical FMRI experiment can easily have 100 to 200 images, and each snapshot can also be a volume.

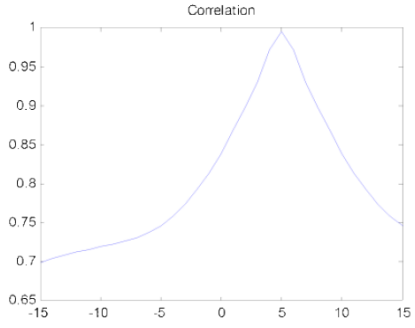
For such a monomodal registration scenario, one could use **cross-correlation** as a cost function.

$$CC(f,g) = \frac{\sum_{mn} f_{mn}\, g_{mn}}{\sqrt{\sum_{mn} f_{mn}^2 \cdot \sum_{mn} g_{mn}^2}}$$



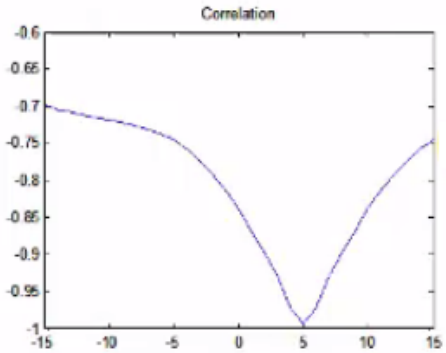
$$\frac{\bar{f}\cdot\bar{g}}{\|\bar{f}\|^2\cdot\|\bar{g}\|^2} = \boldsymbol{cost} \leftarrow \textit{angle between vectors}$$

If CC = 1 => f & g are positively correlated, i.e. f = αg for some α > 0
If CC = 0 => f & g are not correlated, i.e. f & g are independent (orthogonal)
If CC = -1 => f & g are negatively correlated, i.e. f = αg for some α < 0

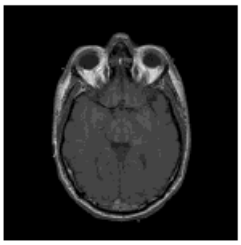


Same T1-weighted MRI used in the SAD example in L25. Max occurs at correct registration.
<-

If the image intensity is negated, the correct registration give a minimum. ->



g



f=-g

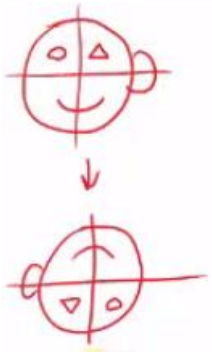
One of the great things about cross-correlation is that it can be formulated as a **convolution** and evaluated efficiently for all integer translation using the FFT.

Consider a shifted version of *f*.

$$g_{mn} = f_{m-\alpha,n-\beta} = [T(\alpha,\beta) f]_{mn}$$

alpha, beta: true shift

To find the optimal shift, we can compute $\boldsymbol{CC(T(a,b)f,g)}$ for all integer shifts $\boldsymbol{(a,b)}$ and choose the shift that gives the largest value (or absolute value).



$$CC(T(a,b)f,g) = \frac{\sum_{mn} f_{m-a,n-b}\, g_{mn}}{\sqrt{\sum_{mn} f_{mn}^2 \cdot \sum_{mn} g_{mn}^2}}$$

The numerator is almost a convolution.

Fbar: reversed version

$$\text{Let } \bar{f}_{mn} = f_{-m,-n} \Rightarrow f_{m-a,n-b} = \bar{f}_{a-m,b-n}$$

Then

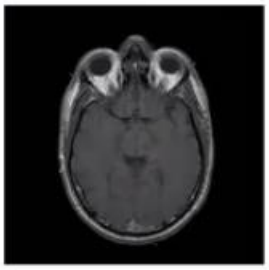
$$CC(T(a,b)f,g) = \frac{\sum_{mn} \bar{f}_{a-m,b-n}\, g_{mn}}{\sqrt{\sum_{mn} \bar{f}_{mn}^2 \cdot \sum_{mn} g_{mn}^2}}$$

$$= \frac{(\bar{f} * g)_{ab}}{\sqrt{\sum_{mn} \bar{f}_{mn}^2 \cdot \sum_{mn} g_{mn}^2}}$$

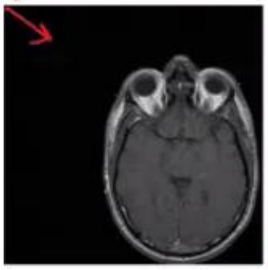
$$= \frac{1}{\sqrt{\Sigma f^2 \Sigma g^2}} \mathcal{F}^{-1}\left\{ \mathcal{F}\{\bar{f}\}_{kl}\, \mathcal{F}\{g\}_{kl} \right\}_{ab}$$

**Half shifts (17:50)?

f

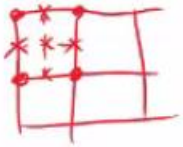


g = f shifted + noise



Maximum occurs at (a,b) that corresponds to the correct shift.

CC(T(a,b)f, g)



Lesson 27 – Least Squares

Goal: to find out how the sum of squared differences cost function is used in image registration

The "least squares" cost function, also known as the "sum of squared differences" (SSD), is a very popular cost function.

$$d_{SSD}(f, g) = \sum_{mn} (f_{mn} - g_{mn})^2$$

The reason for its popularity:

1. It is the optimal cost function in the presence of Gaussian-distributed additive noise.
2. It has an analytical solution (sort of)

Consider the rigid-body 2D transform with 3 parameters,

(r, c, Θ) = (row shift, column shift, rotation)

Using Taylor's theorem (again),

$$T(r,c,\theta)f = T(0,0,0)f + \frac{\partial T(0,0,0)f}{\partial r} r + \frac{\partial T(0,0,0)f}{\partial c} c + \frac{\partial T(0,0,0)f}{\partial \theta} \theta + \theta(r^2, c^2, \theta^2)$$

So, if r, c & θ are small,

$$T(r,c,\theta)f \overset{f}{\cong} f + \frac{\partial f}{\partial r} r + \frac{\partial f}{\partial c} c + \frac{\partial f}{\partial \theta} \theta$$

$$= f + \nabla f \begin{bmatrix} r \\ c \\ \theta \end{bmatrix}$$

where $\nabla f = \left[\frac{\partial f}{\partial r}, \frac{\partial f}{\partial c}, \frac{\partial f}{\partial \theta}\right]$.

We can write this equation for every pixel (m, n)

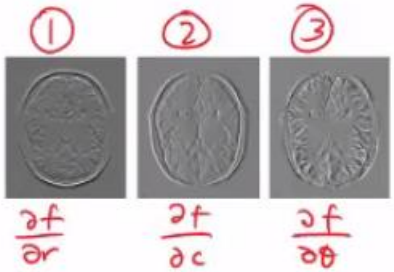
$$\left[T(r,c,\theta)f\right]_{mn} = f_{mn} + \nabla f_{mn} \begin{bmatrix} r \\ c \\ \theta \end{bmatrix} \quad m = 1, \ldots, M, \; n = 1, \ldots, N$$

If we think of our image as a column vector $\vec{f}$, then

$$T(r,c,\theta)\vec{f} = \vec{f} + \nabla \vec{f} \begin{bmatrix} r \\ c \\ \theta \end{bmatrix} \quad \text{P}$$

Now this is an $(MN) \times 3$ matrix

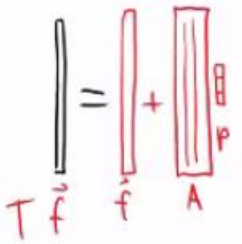
$$\nabla \vec{f} = \begin{bmatrix} \frac{\partial f_{11}}{\partial r} & \frac{\partial f_{11}}{\partial c} & \frac{\partial f_{11}}{\partial \theta} \\ \frac{\partial f_{12}}{\partial r} & & \\ \vdots & & \\ \frac{\partial f_{MN}}{\partial r} & \frac{\partial f_{MN}}{\partial c} & \frac{\partial f_{MN}}{\partial \theta} \end{bmatrix} = A$$



Then we can approximate small motions using

$$T(r,c,\theta)\,\vec{f} = \vec{f} + Ap$$

Each row of this matrix equation corresponds to one pixel



The moved $\vec{f}$ equals the original plus a linear motion term... a "nudge".

It's important to remember that **Tf** is nonlinear, and that we have merely approximated it for **small motions** (nudges).

We are looking for the motion parameters (r, c, Θ) that minimize

$$\sum_{mn}\left(T(r,c,\theta)f_{mn} - g_{mn}\right)^2 = \left\| T(r,c,\theta)\,\vec{f} - \vec{g} \right\|_2^2$$

Replacing $T(r,c,\theta)\,\vec{f}$ with our linear approx $\vec{f} + Ap$

$$\Rightarrow \min_p \left\| \vec{f} - Ap - \vec{g} \right\|_2^2$$

This is a linear least-squares problem. I'll skip the details, but we can solve it like this:
overly determined system, Moore-Penrose pseudoinverse

$$A^T\left(\vec{f} + Ap - \vec{g}\right) = \vec{0}$$

$$A^TAp = A^T(\vec{g} - \vec{f})$$

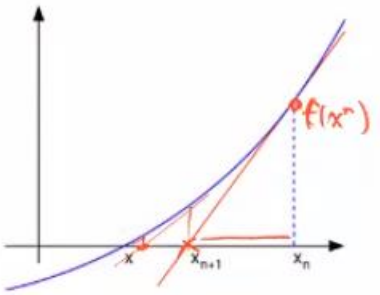
$$\rightarrow p = (A^TA)^{-1}A^T(\vec{g} - \vec{f})$$

This is a 3x3 matrix, so this system is easy to solve.

As mentioned, we are only solving a linear approximation to our nonlinear problem. How can we claim to actually solve the nonlinear problem?

Newton Method

To solve $f(x)=0$, $f:\mathbb{R}\to\mathbb{R}$,
iterate $x^{n+1} = x^n - \dfrac{f(x^n)}{f'(x^n)}$
until you reach a fixed point.



Let's derive a vector version.

We want to solve $L(p)=\vec{0}\in\mathbb{R}^N$ for $p\in\mathbb{R}^P$, where

$\to L(p) = T(p)\vec{f} - \vec{g} \approx \vec{f} + Ap - \vec{g}$

Taylor's thm... $A\in\mathbb{R}^{N\times P}$

$$L(p+\Delta p) = L(p) + \nabla L(p)\Delta p + O(\Delta p^2) = \vec{0}$$

$$\Rightarrow L(p) + A\Delta p \approx \vec{0}$$

$$A^T L(p) + A^T A\Delta p = \vec{0}\in\mathbb{R}^P$$

$$A^T A\Delta p = -A^T L(p)$$

$$\Delta p = -(A^T A)^{-1}A^T\left(T(p)\vec{f} - \vec{g}\right)$$

$$\therefore\ p^{t+1} = p^t - (A^T A)^{-1}A^T\left(T(p)\vec{f} - \vec{g}\right)$$

Note that the **A** matrices also depend on **p**.

In this iterative scheme, one starts with a guess for **p**, then computes $\boldsymbol{T(p)\vec{f}}$ and **A**. At each iteration, **p** is updated, and so is $\boldsymbol{T(p)\vec{f}}$ and **A**.

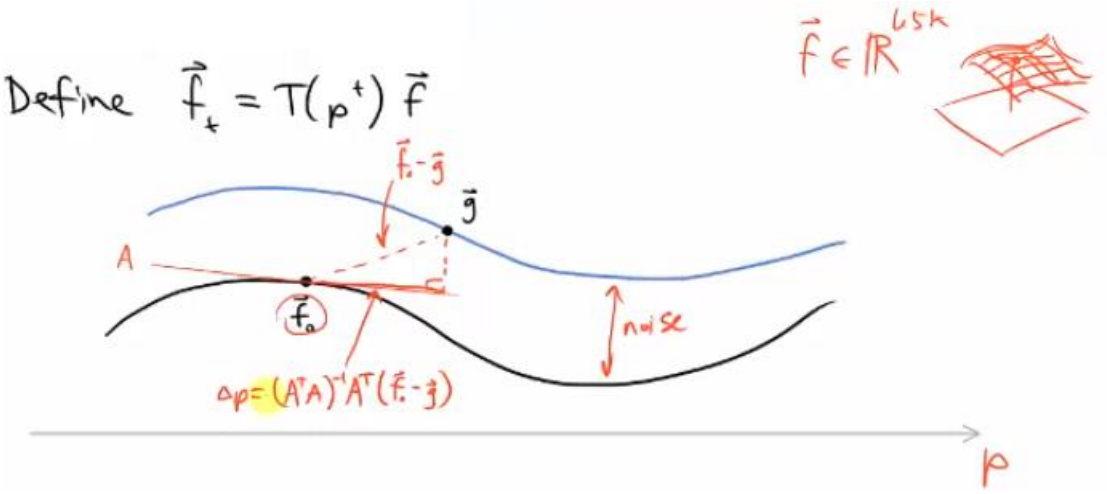
Once it converges to a fixed point, we have

$$(A^T A)^{-1}A^T\left(T(p)\vec{f} - \vec{g}\right) = \vec{0}$$

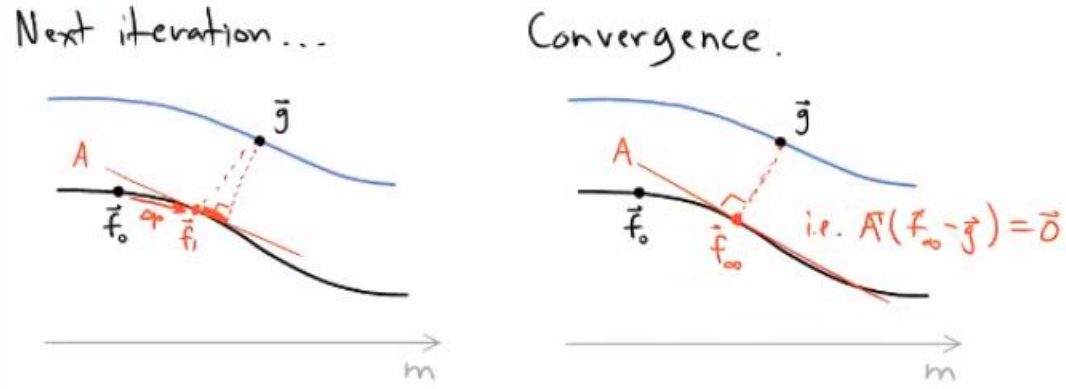
(T(p)f-g) is orthogonal to the columns of A

This is called "**fixed-point iteration**".

Geometric Interpretation



The curves represent all the different images you can get by moving $\vec{f}$ or $\vec{g}$ (i.e. by moving along the **p-axes**). I call these curves "**motion manifolds**".
Their motion manifolds are similar, but separated because of noise.



(p instead of m axes)

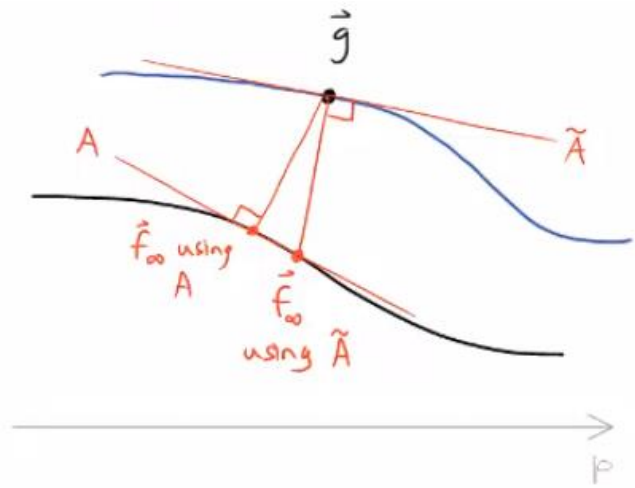
Implementation detail: you might have noticed that A has to be re-computed every iteration. **(Expensive)**
Here is a trick: instead of approximating $\boldsymbol{T(p)\vec{f}}$ and **A**.

$$T^{(-1)}(p)\vec{g} = \vec{g} - \nabla\vec{g}\,p = \vec{g} - \tilde{A}p$$

This is a slightly different $\tilde{A}$ than the one we get from approx $T(p)\vec{f}$

$$\text{Now } \tilde{A}^T(\vec{f} - T^{-1}(p)\vec{g}) = \vec{0}$$
$$\Rightarrow \tilde{A}^T(\vec{f} - (\vec{g} - \tilde{A}m)) = \tilde{A}^T(\vec{f} - \vec{g}) + A^TAp = \vec{0}$$
$$\Rightarrow p = (\tilde{A}^T\tilde{A})^{-1}\tilde{A}^T(\vec{f} - \vec{g})$$

Same as before, but with a different $\tilde{A}$.

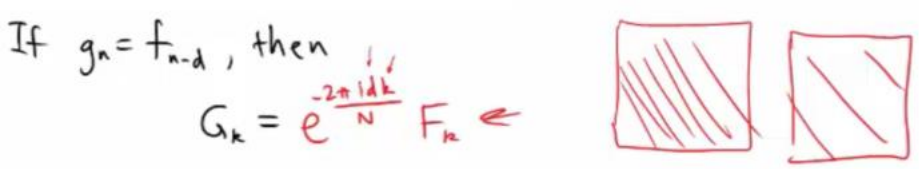


The advantage of this trick is that g does not change, so you don't have to recompute A.

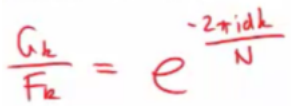
Lesson 28 – Transform-Based Registration

Goal: to see how we can take advantage of certain transformations for the purpose of registration.

Recall the Fourier Shift Theorem:



Thus, to determine the shift, you can try to find a phase ramp that relates **F_k and G_k**.



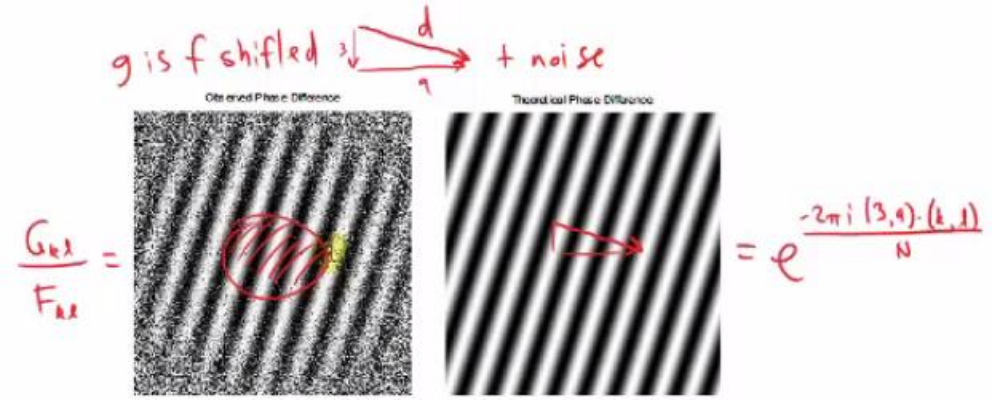
The slope of the ramp gives the relative shift **-> d**

We can get a divide-by-zero if **$F_k = 0$**

Instead, we can use the phase difference between $\boldsymbol{G_k}$ & $\boldsymbol{F_k}$, which can be done using MATLAB's **angle** function.

i.e. $\boldsymbol{angle(re^{i\theta}) = \theta}$

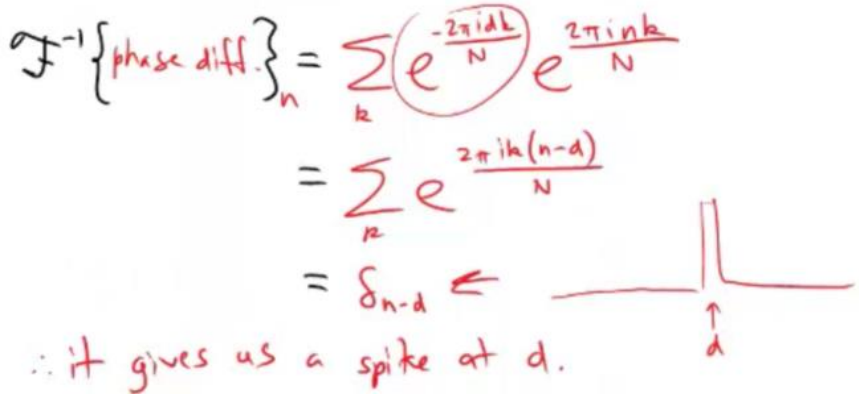
(MATLAB demo)



However, as you can see, the phase difference gets progressively noisy in the high frequencies. But you can still rely on the low-frequency coefficients to try and estimate the wave front (frequency and direction).

Phase Correlation

Instead of looking in the frequency domain to estimate the phase ramp, let's look at the spatial-domain representation of the phase difference. **(d: vector of direction of frequency)**



Limitations of Phase Difference

These methods:

- Assume that the two images have the same **intensity mappings (identity intensity map)**.
  i.e. corresponding objects in the two images have the same intensity, such as in monomodal images
- assume that the two images have the same **field of view**. The Fourier transform is a local ↔ global transform
  i.e. changing 1 pixel changes all the Fourier coefficients, and changing 1 Fourier coefficient changes all the pixels.
  Hence, there seems to be no easy way to perform this type of registration on only a subimage.
- Only find the optimal **translation**
  There is a way to deal with rotation, though…

Lesson 29 – Tricks with Transform-Based Registration

Goal: to find out how rotations and scales can be discovered from transformation-based registration methods.

Rotations

We have seen that rotations are invariant under the Fourier transform. That is, rotating an image by Θ also rotates its FT by Θ.

What if an image is rotated AND translated?

Consider the FT of $f(Rx+b)$, where $x, b \in \mathbb{R}^2$.

$$\tilde{F}(\omega) = \int f(Rx+b)\, e^{-2\pi i \omega \cdot x}\, dx \qquad e^{-\omega^T x}$$

$$\begin{bmatrix} k \\ l \end{bmatrix} \cdot \begin{bmatrix} m \\ n \end{bmatrix} = \begin{bmatrix} k & l \end{bmatrix} \begin{bmatrix} m \\ n \end{bmatrix}$$

Let $y = Rx+b \Rightarrow dy = \det\left(\frac{\partial y}{\partial x}\right) dx = \det(R)\, dx = dx$

$$\Rightarrow x = R^{-1}(y-b)$$

$$\tilde{F}(\omega) = \int f(y)\, e^{-2\pi i\, \omega^T (R^{-1}(y-b))}\, dy \qquad (\omega^T R^{-1} y)^T = y^T (R^{-1})^T \omega$$

$$= y^T R \omega$$

$$= \int f(y)\, e^{-2\pi i \left[\omega^T R^{-1} y - \omega^T R^{-1} b\right]}\, dy$$

These are scalar, so we can transpose them. Note: R is orthogonal, so $R^{-1} = R^T$.

$$= \int f(y)\, e^{-2\pi i \left[y^T R \omega - b^T R \omega\right]}\, dy$$

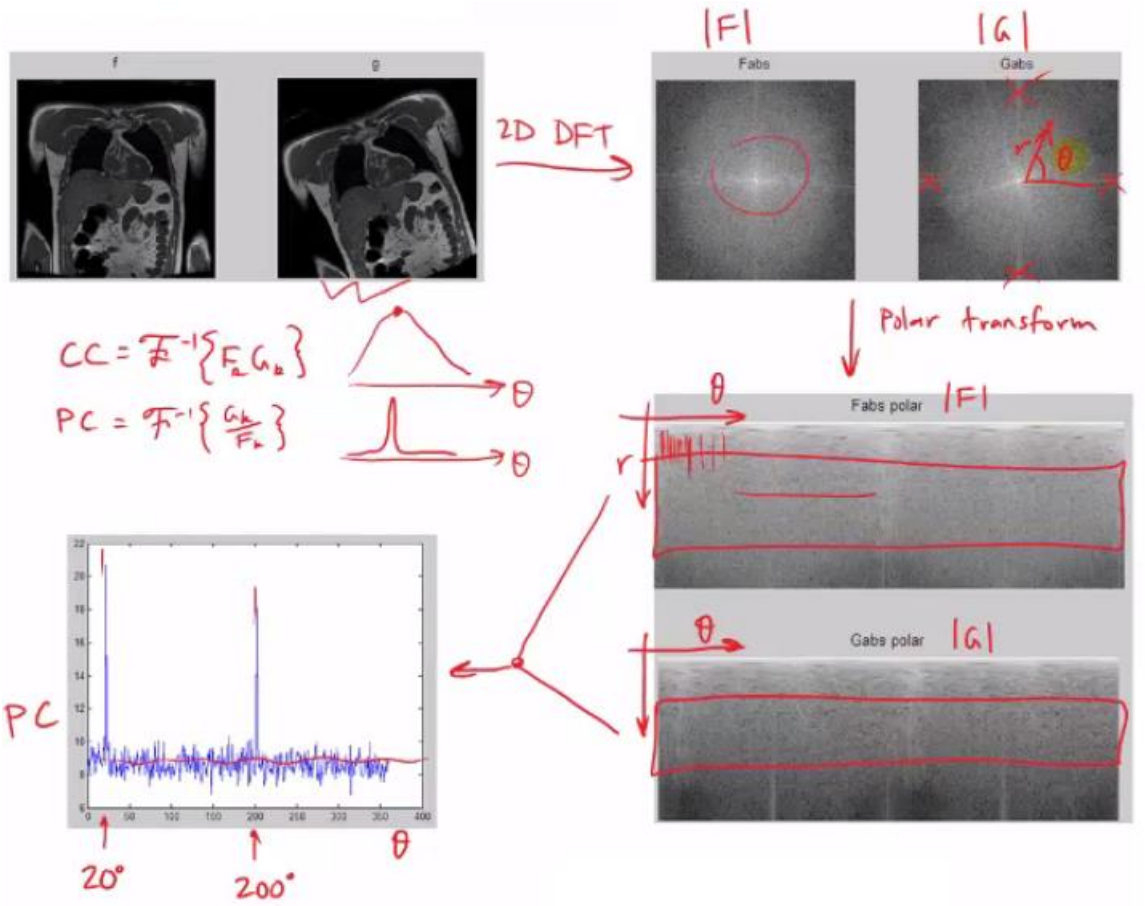
$$= e^{+2\pi i b^T R \omega} \int f(y)\, e^{-2\pi i y^T (R\omega)}\, dy$$

Phase -> $$= e^{2\pi i b^T R \omega} F(R\omega)$$

This is just phase-ramped, rotated version of F(w), but the direction of the ramp is rotated also.

Since the translation only affects the phase of the Fourier coefficients, we can look at the **modulus** of the coefficients to try to estimate rotation. Once the rotation is accounted for, we can look for the phase differences to estimate translation. In the way, rotation and translation are **decoupled** in the frequency domain.

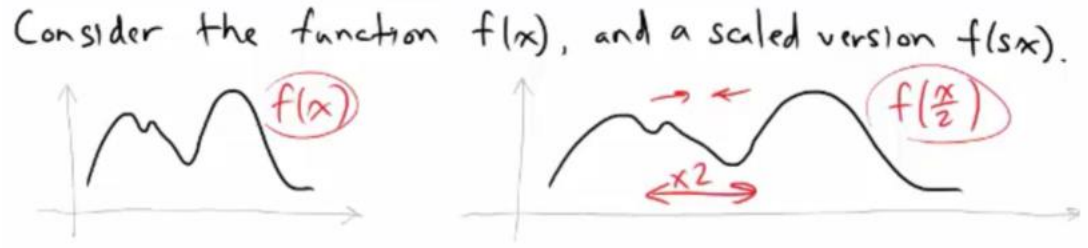
The reason that rotation and translation are decoupled in the frequency domain is that the centre of rotation is known… the origin. For 2D rotation, this turns the problem of finding the optimal rotation into a 1D shift problem; we convert our FTs into polar coordinates and look for the best shift along the Θ-axis.



Peaks at 20 degrees and 200 degrees

Notice there are two peaks. Why?

Scaling



Define $g(\log x) = f(x)$.

What is $f(\frac{x}{2})$ in terms of g?

$$f(\tfrac{x}{2}) = g(\log \tfrac{x}{2}) = g(\log x - \log 2)$$

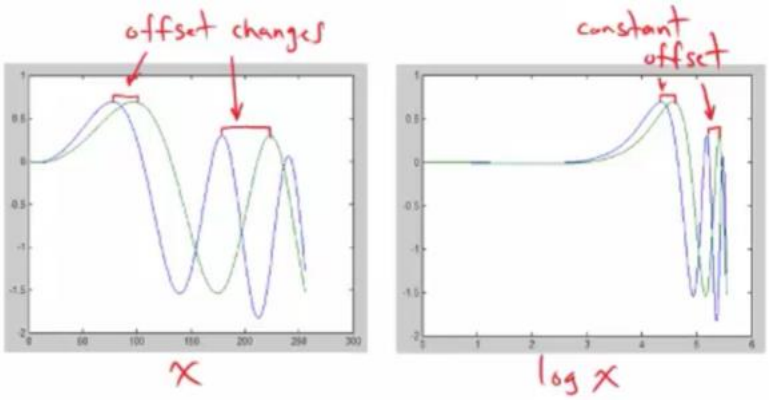
This is a shifted version of g

Hence, applying a spatial log mapping to an image turns scaling into **translation**. However, just like in rotations, the centre of scaling has to be the origin of the log mapping.

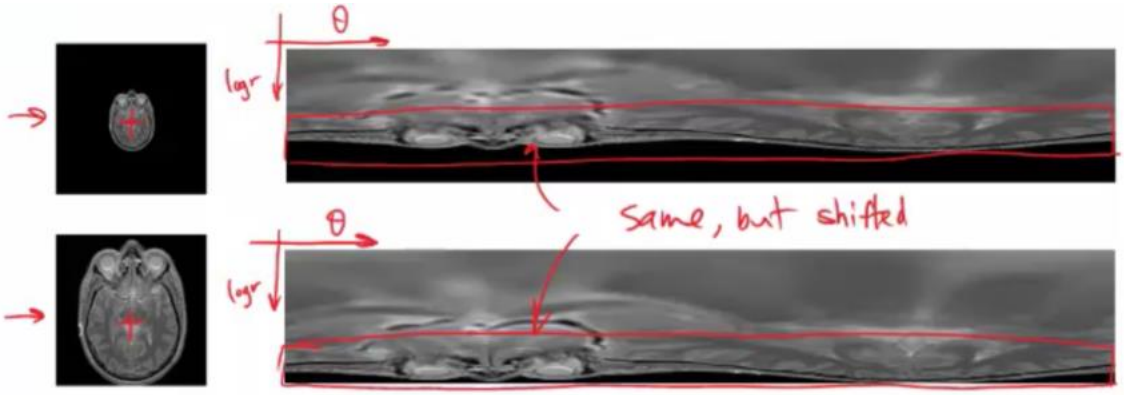
```
x = 1:256;
f = sin((x/64).^2) - x/256;
g = sin((x*0.8/64).^2) ...
    -x*0.8/256;

figure(1); plot(x,f,x,g);

figure(2); plot(log(x),f,lx,g);
```



In 2D, we can use the log-polar transform. A scale difference between the images turns into a shift along the log-radius axis.

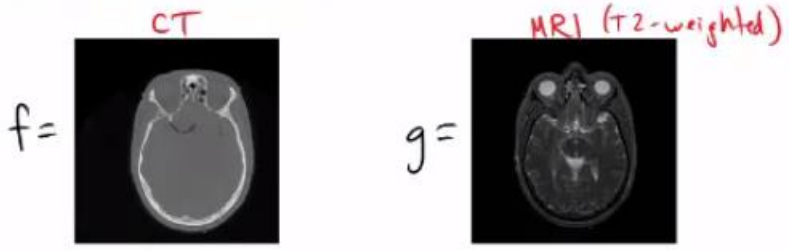


Just like for rotation, we need to know the **centre of expansion**; in this way, scaling and translation are **coupled**. However, it is easy to show that the Fourier transform is invariant under scaling, so one can do the same trick as for rotation. You can use the **modulus** of the Fourier coefficients to estimate the scale because the translation only affects the **phase**.

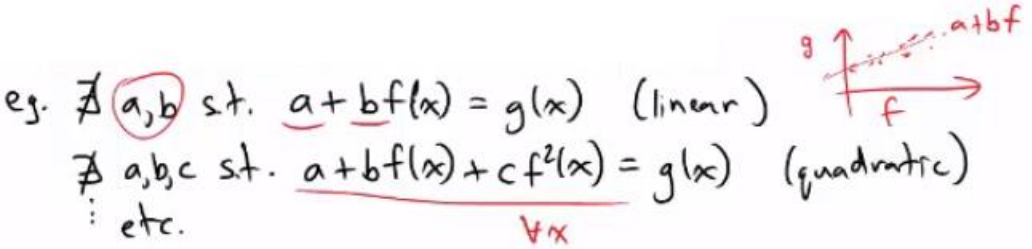
Lesson 30 – Joint Entropy Registration

Goal: an introduction to information-theoretic methods for image registration.

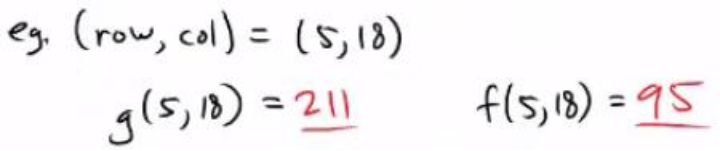
Consider the situation of registering a CT scan (f) to an MRI (g) of the same anatomy.



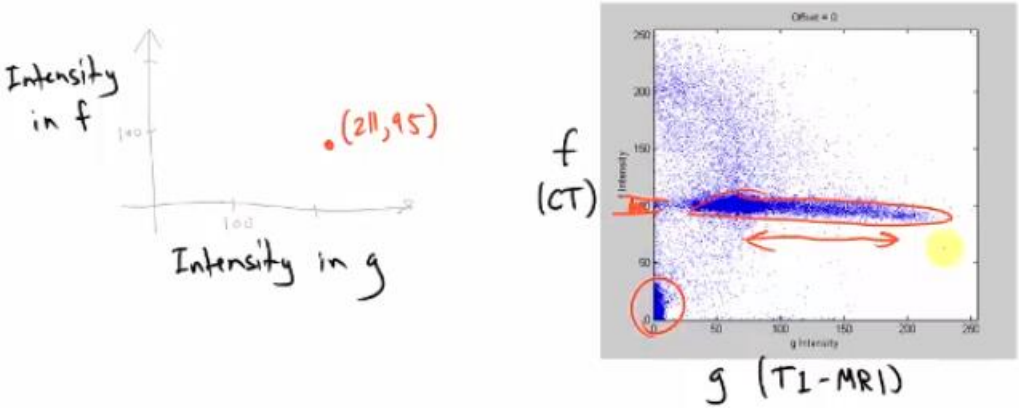
This is an example of multimodal registration.
In particular, there is no known functional relationship between the intensities of the two images.



To see this more clearly, let's look at images that are already registered. Each pixel has an intensity from each image, so each pixel can be represented as a point in 2-space.



Therefore we plot a point at (211, 95)



highlighted: tissue better in MRI

The dispersion (compactness) of the Joint Intensity Scatter Plot (JISP) can be used as a measure of registration. But how do we get a number that reflects the dispersion of the JISP?

Entropy

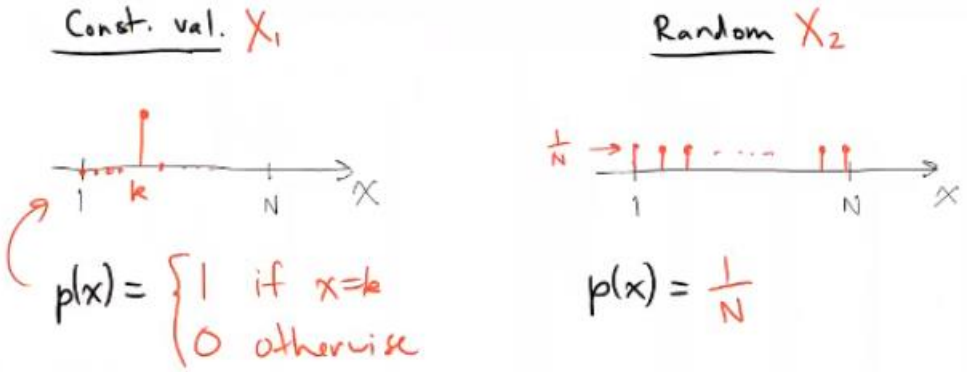
The entropy of a discrete random variable X is

$$H(x) = -\sum_i p(x_i) \log p(x_i)$$

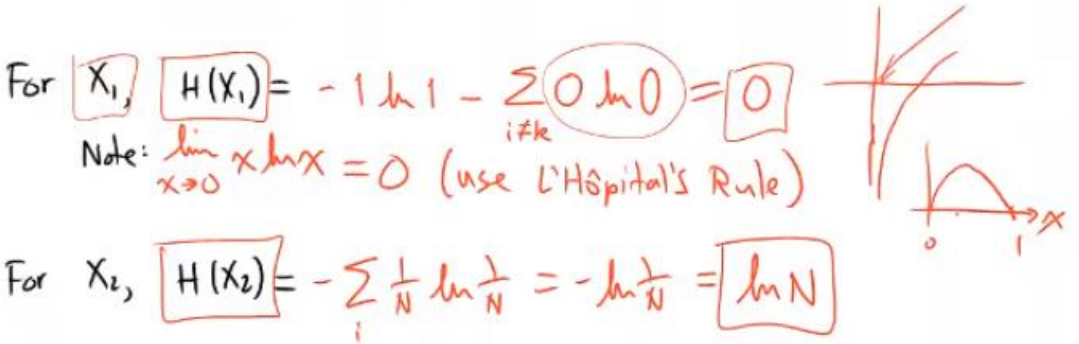
- **Where p(x) the probability of observing x**

Entropy is a measure of disorder or randomness. In particular, the more spread out a signal's histogram, the higher the entropy.

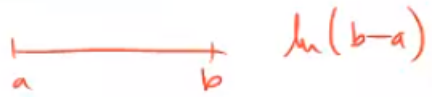
E.g. consider two signals, one with a constant value, and the other random. Here are their pdfs:



Left: use low entropy **Right: use high entropy**

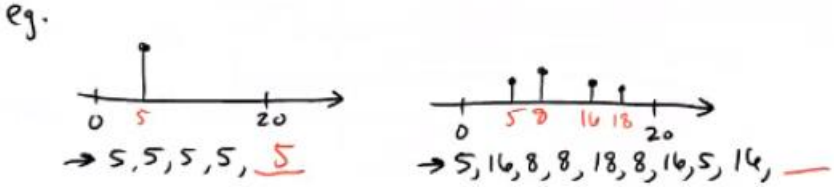


Note: for continuous-domain variables, the formula for entropy is an integral.

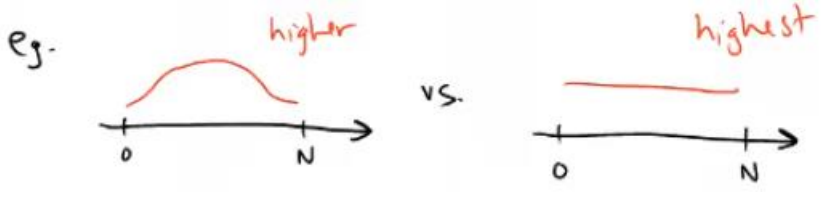


Interpretation

- A constant signal has no ~~information~~ **randomness**, and no randomness. The next sample can easily be predicted. Hence a low entropy histogram contains little information, but has only a few tight and compact peaks.

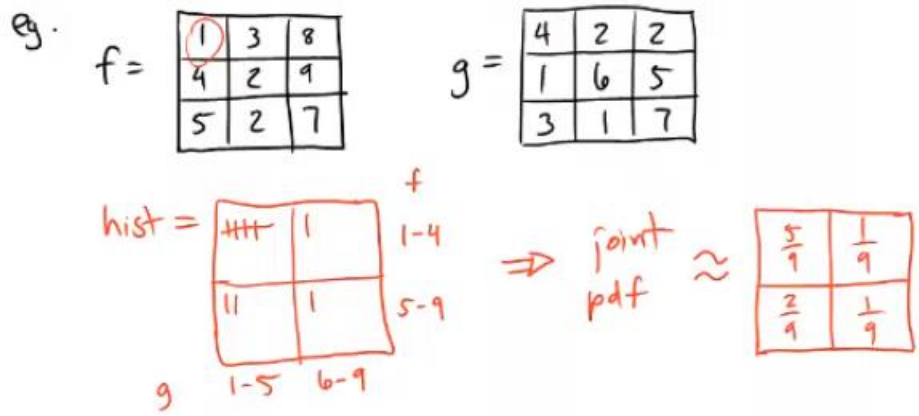


- A signal with lots of different values, each occurring with similar probability, is more random or disorderly, and is not easily predictable (in general). Hence, high entropy is associated with lots of information, and the histogram tends to be more spread out.



(L: lower entropy, R: higher)

In terms of registration, we can look at the joint histogram, a binned version of the JISP. We partition the joint intensity space into a collection of bins, and estimate the joint pdf by counting the sample frequency for each bin.



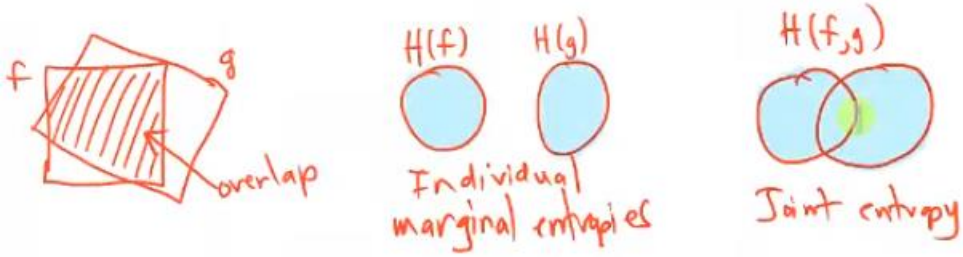
Then we evaluate the entropy of the joint histogram, and use that as a measure of registration. Thus, we can use the entropy of the joint histogram as a registration objective function. When the images are registered, the cost function will (hopefully) attain a **minimum** value.
(MATLAB demo)

We refer to the entropy of the joint histogram as "joint entropy" and denote it:

$$H(f,g) = H(X_1, X_2) = -\sum_{ij} p_{ij} \ln p_{ij}$$

(minimize + gradient descent)

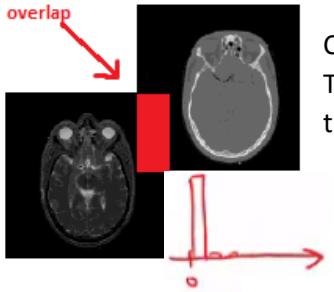
It effectively measures the amount of information in **f** or **g** within the region of overlap.



Lesson 31 – Mutual Information

Goal: when does joint entropy fail, and what other information-theoretic objective functions work better.

There is a problem with joint entropy:



Consider the case when only the background of the images overlap.
The histogram is only computed for the overlapping portion. As this region gets smaller, there are fewer samples, and it's easier to achieve a single peak in the histogram.

i.e. once it's only background, then the joint histogram will be all zero except for one bin containing the background intensities for both images.
=> tight, compact histogram
=> low joint entropy

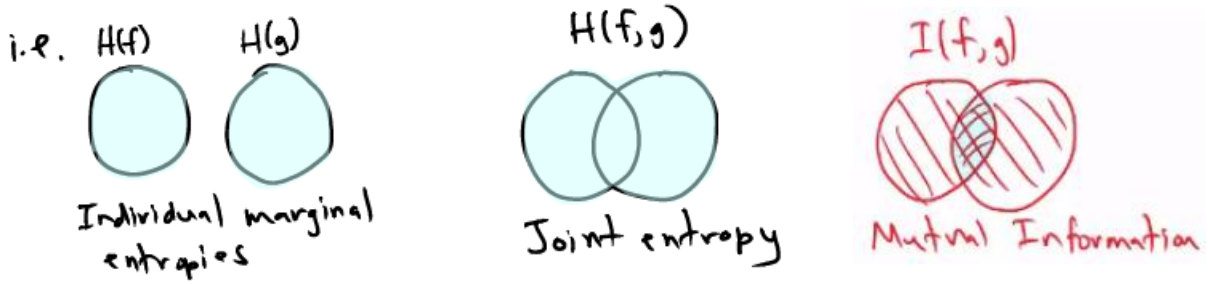
(MATLAB demo) L31_Entropy_Problem

- Start: tight on one spot
- Initially registered, then increasingly out of register
- Graph is hyperbola with two humps
- Low entropy when overlap more trivial

The cost function somehow has to also encourage nontrivial image content in the overlap region.

Mutual Information

Recall that $H(f,g)$ is the amount of information that is in either f or g or both. If you subtract $H(f,g)$ from the sum of $H(f)$ and $H(g)$, you are left with the total amount of information that both f and g share. This is called the mutual information of f and g.



Mutual Information is defined as:

$$I(f,g) = H(f) + H(g) - H(f,g)$$

Recall that

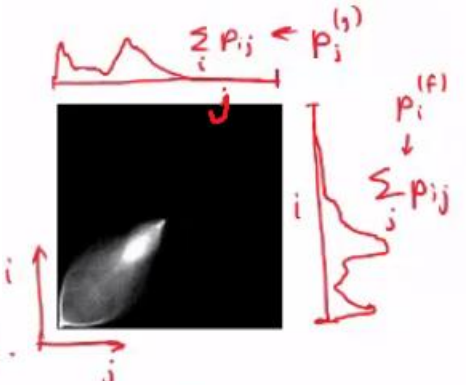
$$H(f) = -\sum_i p_i^{(f)} \ln p_i^{(f)} \qquad H(g) = -\sum_j p_j^{(g)} \ln p_j^{(g)}$$

$$H(f,g) = -\sum_i \sum_j p_{ij} \ln p_{ij}$$

Thus,

$$I(f,g) = \boxed{-\sum_i p_i^{(f)} \ln p_i^{(f)}} \boxed{-\sum_j p_j^{(g)} \ln p_j^{(g)}} + \sum_i \sum_j p_{ij} \ln p_{ij}$$

Joint histogram



Notice that

$$p_i^{(f)} = \sum_j p_{ij}$$

$$p_j^{(g)} = \sum_i p_{ij}$$

We call $p_i^{(f)}$ and $p_j^{(g)}$ the marginal probabilities.

Continuing the derivation...

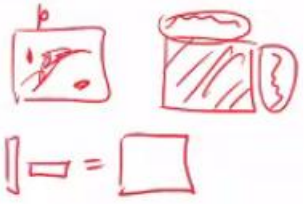
$$I(f,g) = -\sum_i p_i^{(f)} \ln p_i^{(f)} - \sum_j p_j^{(g)} \ln p_j^{(g)} + \sum_i \sum_j p_{ij} \ln p_{ij}$$

$$= -\sum_i \left[\sum_j p_{ij} \ln\left(\sum_j p_{ij}\right)\right] - \sum_j \left[\sum_i p_{ij} \ln\left(\sum_i p_{ij}\right)\right] + \sum_i \sum_j p_{ij} \ln p_{ij}$$

$\ln A - \ln B = \ln \frac{A}{B}$

$$= \sum_i \sum_j \left[p_{ij} \left(\ln p_{ij} - \ln\left(\sum_j p_{ij}\right) - \ln\left(\sum_i p_{ij}\right)\right)\right]$$

$$= \sum_i \sum_j p_{ij} \ln\left(\frac{p_{ij}}{p_i^{(f)} p_j^{(g)}}\right)$$



This is another way of writing mutual information.

Another Problem

As it turns out, MI decreases with more background. More background creates a higher peak in the histogram at (0,0).

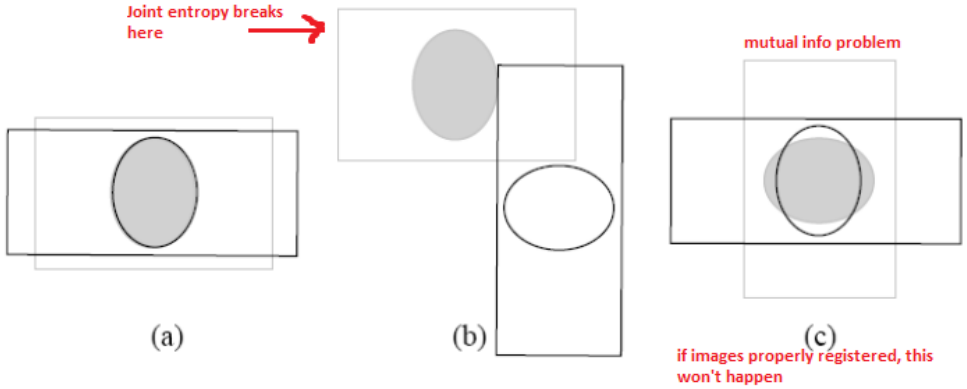


Figure 2.8: Alignment (a) is the correct alignment. However, the entropy of the joint histogram is smaller for (b) because it is calculated only on the overlapped portion. Alignment (c) shows how maximizing mutual information can be inappropriate, since including less background in the overlap increases mutual information. (Adapted with permission from Studholme *et al.* [86])

(from Jeff Orchard's PhD thesis)

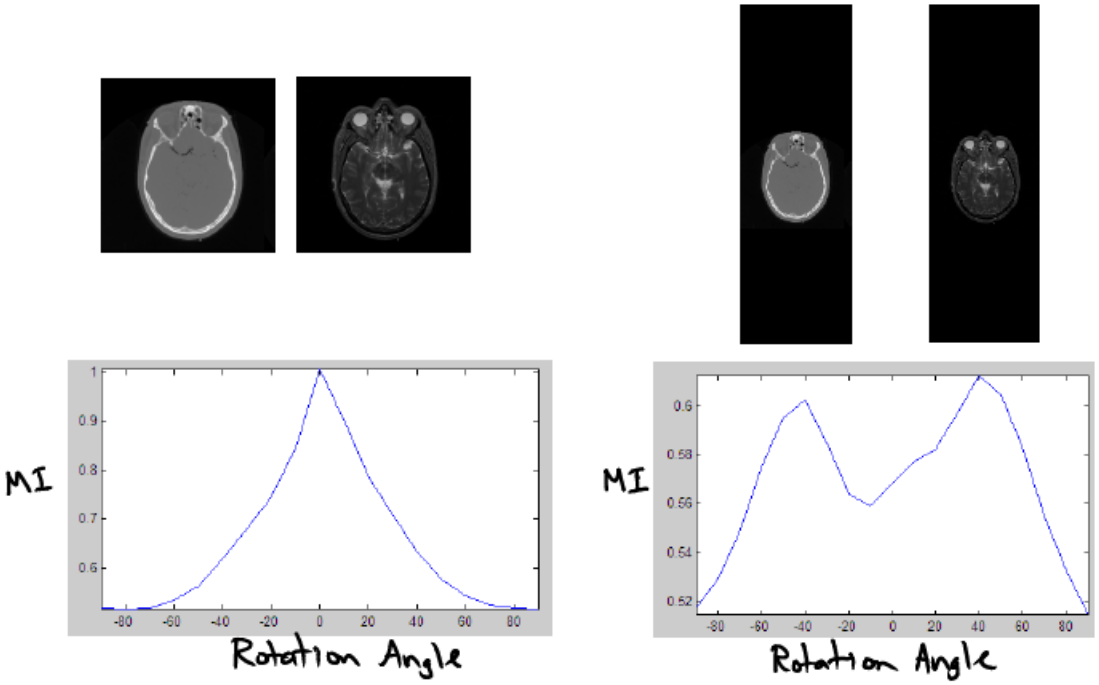
**mutual info prefers (c) over (a) – prefers incorrect registration over correct registration

- When register two images, want to minimize entropy
- Max mutual info, minimize entropy – max I(f,g), min H(f,g)

Contrast (a) and (c):
(a) has more background, so H(f) and H(g) are lower.
(c) has less background, so H(f) and H(g) are higher, but the decrease in –H(f,g) is not enough to counteract H(f) and H(g).

**(a) has a big spike at background in the histogram – makes marginal entropies lower
(MATLAB demo – L31_MI_problem)

- Padded image (more background) MI graph has two humps
- First hump is incorrect, off by 50 degrees
- Cannot blindly depend on MI



Normalized Mutual Information
To combat the problem of the amount of background influencing the cost, we have Normalized MI (NMI). There are various forms, including,

$$NMI(f,g) = \frac{\boldsymbol{H(f) + H(g)}}{\boldsymbol{H(f,g)}}$$

Or equivalently,

$$NMI(f,g) = \frac{\boldsymbol{H(f) + H(g) - H(f,g)}}{\boldsymbol{H(f,g)}} = \frac{\boldsymbol{I(f,g)}}{\boldsymbol{H(f,g)}}$$

One of them is one unit higher than the other

- MI goes down, joint entropy goes down when there's a lot of background

(MATLAB demo – L31_MI_Problem with "nmi" instead of "mi")

- Has much cleaner peak when done with large background
- Small background same

MI and NMI are routinely used to register images from different modalities. They constitute the current state-of-the-art.

Lesson 32 – Binning Discontinuities (skipped)

Goal: to appreciate the difficulties introduced by binning for the purpose of forming a histogram

(skipped)

Lesson 33 – Registration Optimization

Goal: to see what factors influence registration optimization, and how we can speed registration up.

Interpolation Effect

<skipped>

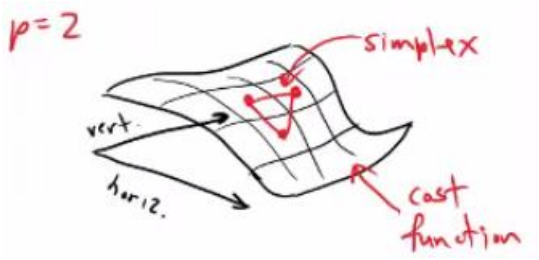
Nelder-Mead Optimization

Many optimization methods use derivatives.

- Gradient descent
- Our SSD method used a linear approximation

Not all optimization methods use derivatives. The Nelder-Mead optimization uses only function values of the cost function. It sets up a "simplex" of points in the parameter space where the cost function is evaluated. Then, these points are moved – one or two at a time – to crawl toward a better (higher or lower) solution. The simplex consists of **p+1** points, where **p** is the number of parameters you are optimization over.

For example, suppose we are looking for the optimal horizontal and vertical shift.
The simplex has 3 points, with initial positions specified manually.



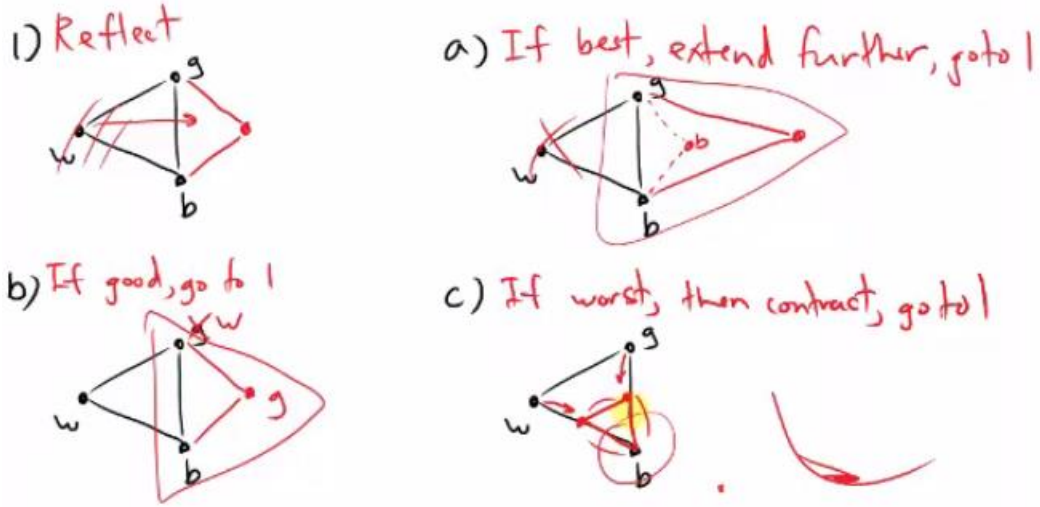
These 3 points are labelled:

w = worst **(highest cost)**
b = best **(lowest cost)**
g = good **(all others)**

Then we change the simplex using these rules:

1. Reflect *w* through the centroid (centre) of the simplex
   a. If this produces a new "best", then create new simplex by **extending** the simplex further. Go to 1.
   b. If this produces a "good" point, relabel the points. Go to 1.
   c. If this produces a new "worst", then create a new simplex by **unreflecting** and **compressing** toward the best point. Go to 1.
2. Continue until the simplex is small enough, or the difference between the best and the worst is small enough.

In pictures:



MATLAB's built-in "fminsearch" method is an implementation of Nelder-Mead.

Multiresolution Approaches

This refers to solving a **lower resolution** (and often **smaller**) version of the problem first, and then using its result as a starting point for a **higher resolution** (and usually more **expensive**) version.

This is done for two reasons:

Coarse Adjustments

On a coarse scale, adjustments are based on larger-scale structures, so there is less tendency to get stuck on **details**. Similarly, a one-pixel step in **¼** resolution is the same as a **4 pixel** step in full resolution. Hence correction of gross displacements is rapid.

Computational Complexity

Reduced-scale images have fewer pixels, so pose less of a computational burden.

These two factors complement each other. They allow for quick convergence on a coarse scale.

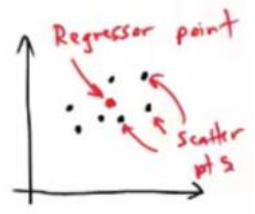
(MATLAB demo: L33_reg_optimization)

Lesson 34 – Registration by Clustering

Goal: to investigate a different way of quantifying the dispersion/compactness of the joint intensity scatter plot.

Rather than having to classify pixels into bins, it would be good to derive the cost directly from the points in the joint intensity scatter plot (JISP).
Recall that aligned images result in tighter clustering in the JISP (less dispersion).

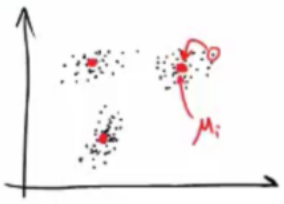
Suppose we add a regressor point to the JISP.



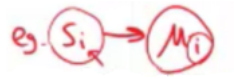
Then the cost could be computed by the **sum of square differences** of the JISP points to that regressor point.

In this way, we model a cluster of points using a point regressor. We can add multiple point regressors (PRs), one for each cluster.

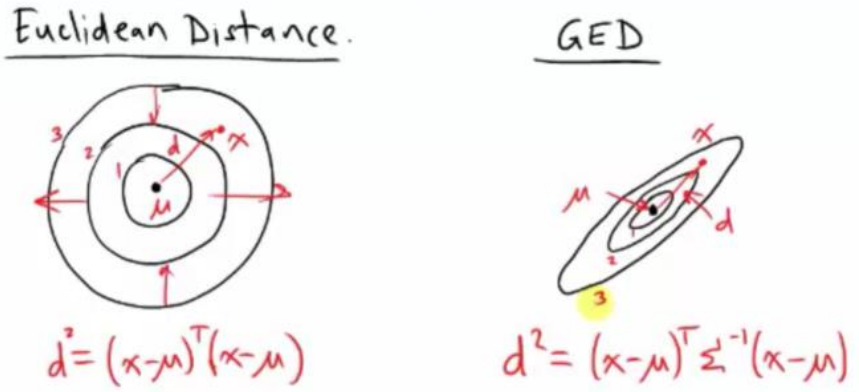
3 regressor points



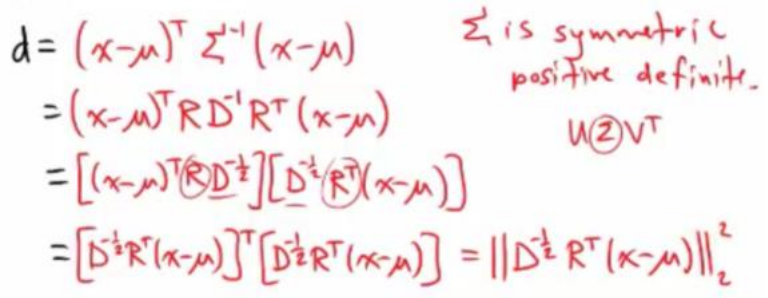
Each JISP point is **assigned** to its nearest PR, and the total cost is the sum of squares of the distances from each JISP point to its nearest PR.



Some clusters of this point are elongated:
We can model this behaviour using **Generalized Euclidean Distance (GED)**.



Σ is a covariance matrix, composed of scalings and a rotation. Notice that Euclidean distance is a special case of GED (as the name implies).



$$d = (x-\mu)^T \Sigma^{-1} (x-\mu)$$

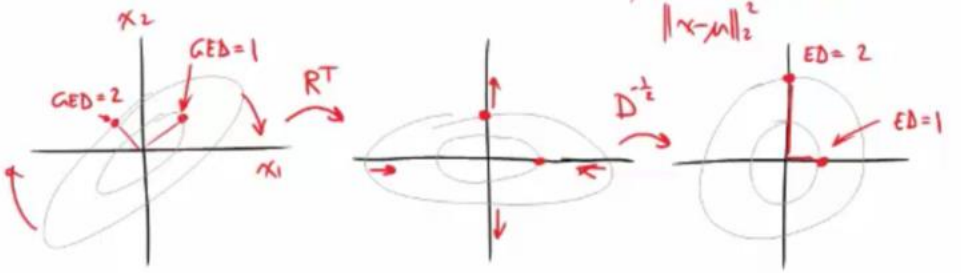
Σ is symmetric positive definite.

$$= (x-\mu)^T R D^{-1} R^T (x-\mu)$$

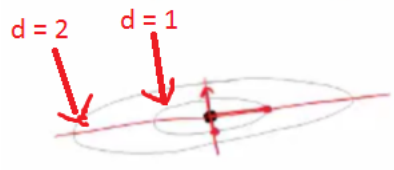
$$= \left[(x-\mu)^T R D^{-\frac{1}{2}}\right]\left[D^{-\frac{1}{2}} R^T (x-\mu)\right]$$

$$= \left[D^{-\frac{1}{2}} R^T (x-\mu)\right]^T \left[D^{-\frac{1}{2}} R^T (x-\mu)\right] = \left\| D^{-\frac{1}{2}} R^T (x-\mu) \right\|_2^2$$

$$\|x-\mu\|_2^2$$

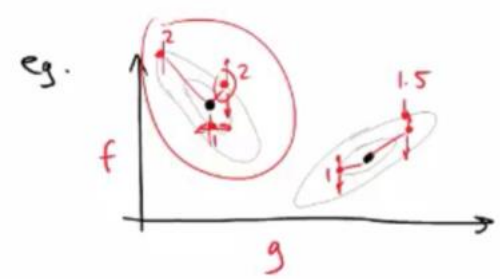


In GED, displacement along the minor axis increments distance more than displacement along the major axis.



The total cost is

$$\sum_{i \in JISP} (\textit{GED from assigned regressor})^2$$



$$\text{Cost} = 2^2 + 2^2 + 1^2 + 1^2 + 1.5^2$$

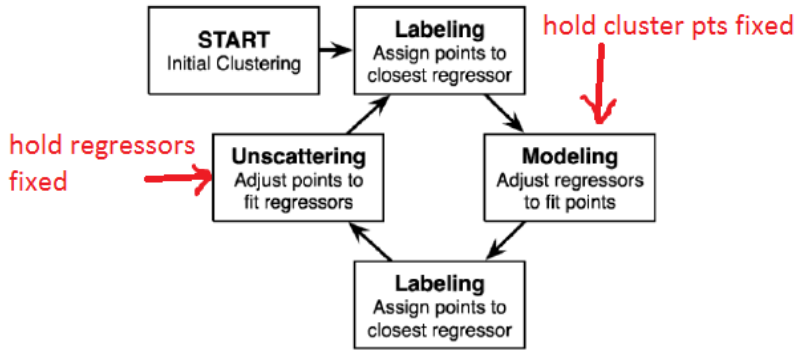
$$= 12.25$$

If one image is moved slightly (e.g. 0.01°), the JISP points move slightly, and the cost changes only slightly.

⇨ **continuous**

There are 3 processes involved in registration using JISP regressors:

- **Labelling**
- **Modelling**
- **Unscattering**



These 3 parts are iterated inside a fixed-point iteration framework.

Labelling
This is simply assigning each JISP point to its nearest regressor. -> using each RP's GED

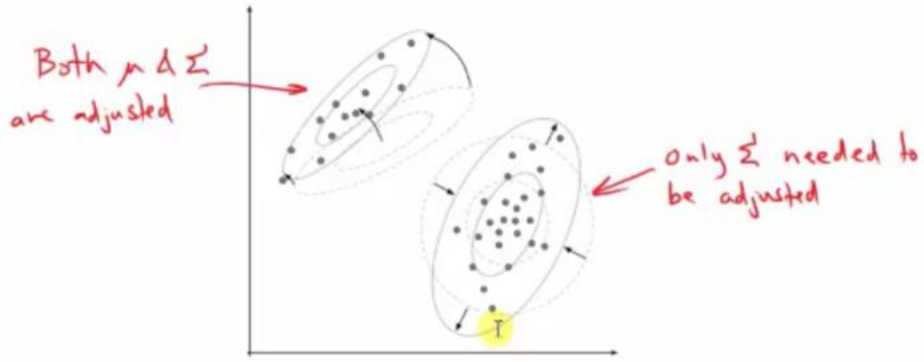
Modelling
Each set of points is approximated by a single regressor. The point of modelling is to adjust that regressor to better fit the points assigned to it.

Optimal location: **centroid of the cluster (centre of mass)**

Optimal Covariance:

$$\Sigma = \sum_i (x-\mu)(x-\mu)^T = \sum_i \| = \square$$

rank 1 matrices^ **^square matrix**



In GED, displacement along the minor axis increments distance more than displacement along the major axis.

Unscattering
We adjust the motion parameters to bring the scatter points (on average) to their regressors.

Suppose we are registering f and g.
Let $p_f = [x_f \; y_f \; \theta_f]^T$ and $p_g = [x_g \; y_g \; \theta_g]^T$
be the 2D rigid-body motion parameters for f & g.
Then our full set of motion parameters is

$$p = \begin{bmatrix} p_f \\ p_g \end{bmatrix} \quad (6\times 1 \text{ column vector})$$

$$R\vec{x} = \vec{y}$$

As usual, we linearize. For pixel i

$$f_i(p_f) \cong f_i + \frac{\partial f_i}{\partial x_f} x_f + \frac{\partial f_i}{\partial y_f} y_f + \frac{\partial f_i}{\partial \theta_f} \theta_f$$

$$g_i(p_g) \cong g_i + \frac{\partial g_i}{\partial x_g} x_g + \frac{\partial g_i}{\partial y_g} y_g + \frac{\partial g_i}{\partial \theta_g} \theta_g$$

Then, the i^{th} scatter point is

$$s_i(p) = \begin{bmatrix} f_i(p_f) \\ g_i(p_g) \end{bmatrix}$$

$$= \begin{bmatrix} f_i \\ g_i \end{bmatrix} + \begin{bmatrix} \frac{\partial f_i}{\partial x_f} & \frac{\partial f_i}{\partial y_f} & \frac{\partial f_i}{\partial \theta_f} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{\partial g_i}{\partial x_g} & \frac{\partial g_i}{\partial y_g} & \frac{\partial g_i}{\partial \theta_g} \end{bmatrix} \begin{bmatrix} p_f \\ p_g \end{bmatrix}$$

s_i ; L_i ; p (6×1)

$$= s_i + L_i p$$

Hence, our cost function can be written

$$C(p) = \sum_i \left\| D_i^{-\frac{1}{2}} R_i^T \left(\underbrace{s_i + L_i p}_{s_i(p)} - \mu_i \right) \right\|^2$$

2-vector

(only a few μ_i's – one regressor per cluster)

Instead of representing our cost function as a sum of **(the length of?)** 2-vectors squared, we can stack all the 2-vectors into one giant column vector with **2 x (# of pixels)** elements.

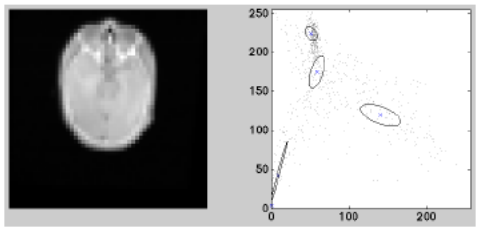
$$\left\| \begin{bmatrix} D_1^{-\frac{1}{2}} R_1^T L_1 \\ \vdots \\ D_N^{-\frac{1}{2}} R_N^T L_N \end{bmatrix} p + \begin{bmatrix} D_1^{-\frac{1}{2}} R_1^T (s_1 - \mu_1) \\ \vdots \\ D_N^{-\frac{1}{2}} R_N^T (s_N - \mu_N) \end{bmatrix} \right\|_2^2 = \left\| \bar{L} p - \bar{d} \right\|_2^2$$

Thus, we want to solve

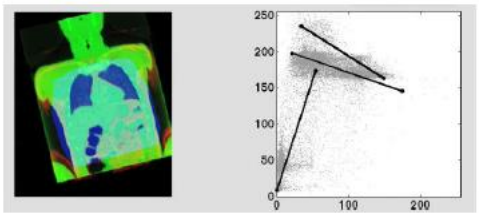
$$\min_p \left\| \bar{L} p + \bar{d} \right\|_2^2$$

This is a simple linear least-squares problem,

$$p = \underbrace{\left(\bar{L}^T \bar{L} \right)^{-1}}_{6 \times 6 \text{ matrix}} \bar{L}^T \bar{d}$$



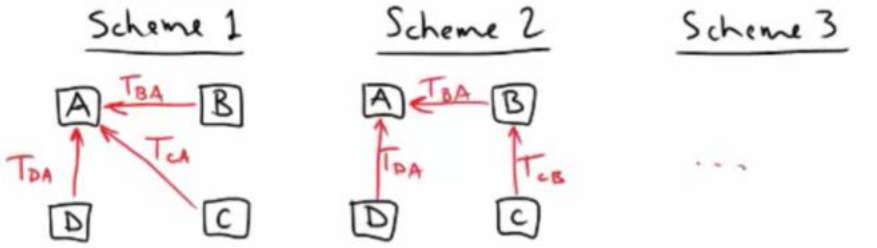
Here is a slightly different implementation that uses line segments instead of GED.



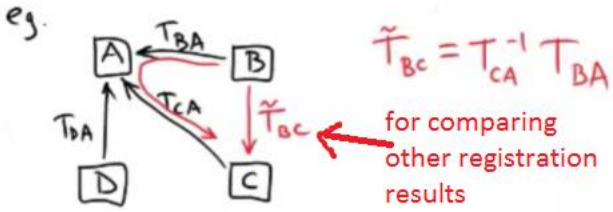
Lesson 35 – Image-Group Registration

Goal: how can we register a whole group of images together?

Suppose you have a 3 or more images that you want to register. How do you go about it?

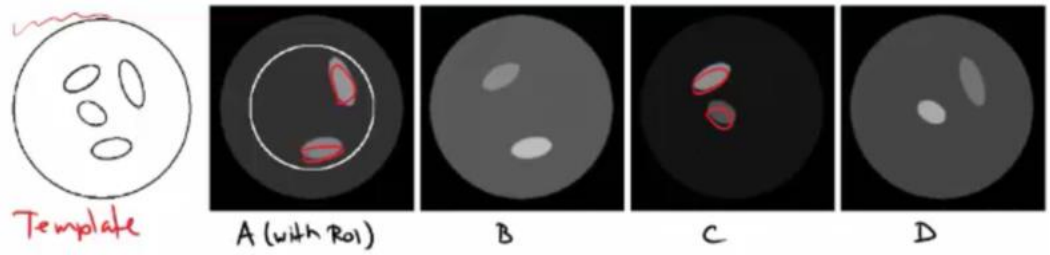


To encode the relative transformation among a set of **E** images, you need **E – 1** transformations.
Then, we can derive the transformation between any two images.



That's find any dandy, how, what happens if we register B to C? We get another estimate of T_{BC}. It's high unlikely that $\widetilde{T_{BC}}$ is exactly the same as T_{BC}. Which is more accurate? Which scheme should we use? What about all the other transforms that might bring up similar inconsistencies?

Here's an extreme (pathological) example:
Consider registering these four images, **four different modalities**



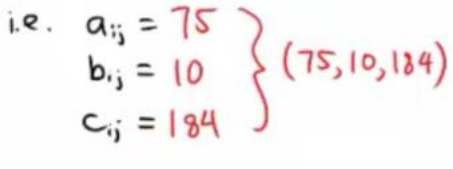
Each image contains two of 4 small ellipses. How would you register A to C? Or B to D?

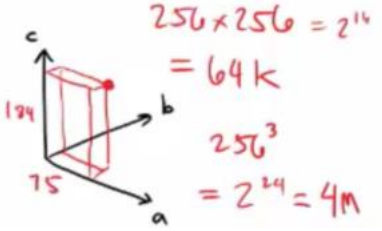
Ensemble Registration

The problem with the above scenarios is that registration is treated as a pairwise operation; it always takes the form of **"Register X to Y"**

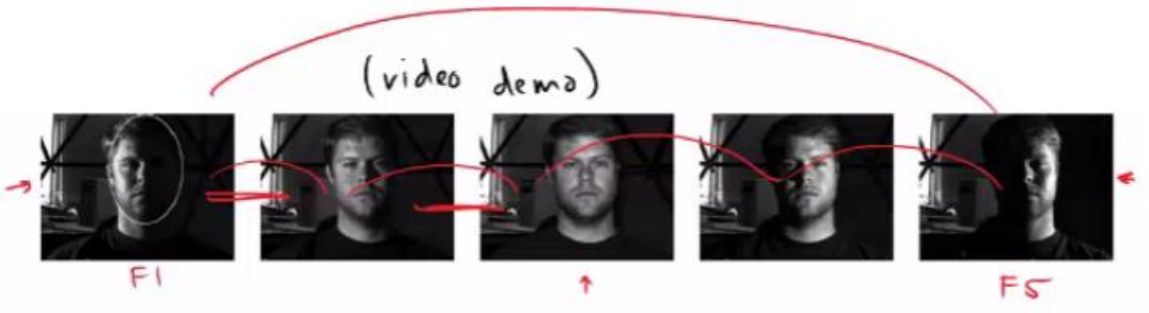
But what if we could simultaneously register multiple images?

Let's consider registering 3 images: **a, b, and c**
We can populate the joint intensity scatter plot, but this time the JISP is 3D.





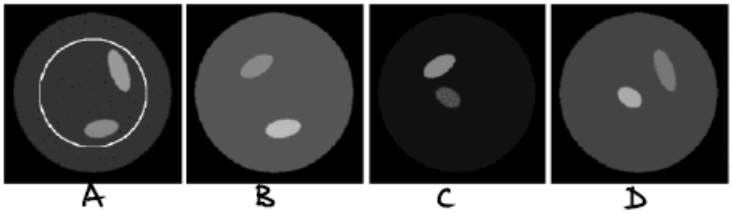
Now we place regressors in this 3D JISP, and do the same clustering process.
(video demo)



Results

Results

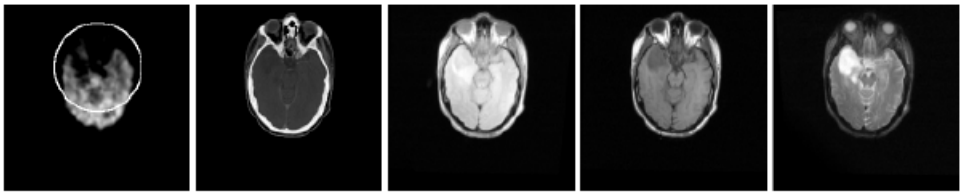
Case I:



Average Pixel Displacement (big = bad)

Method	A-B	A-C	A-D	B-C	B-D	C-D	Mean
NMI (pairwise)	0.59	4.1	0.31	0.83	7.2	4.2	2.87
Ensemble	0.54	0.67	0.32	0.31	0.53	0.64	0.50

Case II:



Method	CT-PET	CT-PD	CT-T1	CT-T2	PET-PD	PET-T1	PET-T2	PD-T1	PD-T2	T1-T2	Mean
NMI (pairwise)	6.1	7.7	2.6	8.2	7.6	5.9	6.7	6.0	4.9	7.7	6.34
Ensemble (pairwise)	1.9	0.80	1.1	1.2	2.6	2.9	2.6	0.93	1.4	2.0	1.73
Ensemble (full)	1.9	0.74	0.87	0.72	1.8	1.7	1.4	0.50	0.73	0.92	1.13

Case III:

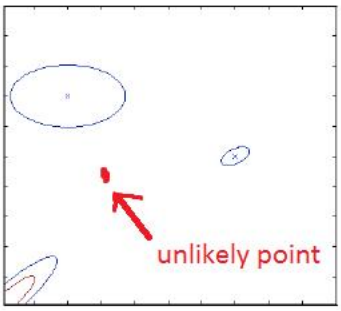
Method	F1-F2	F1-F3	F1-F4	F1-F5	F2-F3	F2-F4	F2-F5	F3-F4	F3-F5	F4-F5	Mean
NMI (pairwise)	33	96	10	41	64	89	34	49	99	15	54.6
Ensemble (pairwise)	18	21	12	15	20	21	17	17	25	14	18.0
Ensemble (full)	0.46	0.97	1.8	1.8	1.1	2.1	2.0	1.7	2.1	0.44	1.45

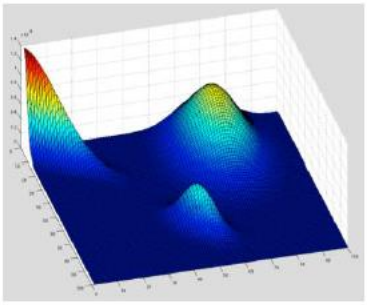
Advantages: don't have to form histograms – joint-intensity spaces of many dimensions (many images) into one optimization method.

Lesson 36 – Gaussian Mixture Models (GMM)

Goal: learn how to do clustering registration without having to label points as assigned to one regressor.

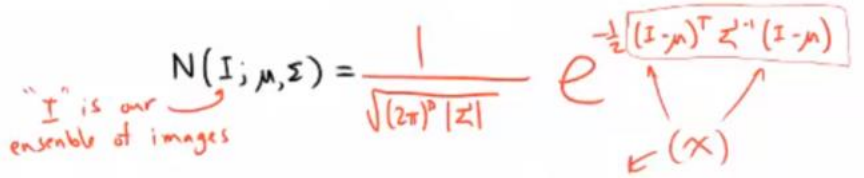
Instead of representing clusters with regressors and computing cost as a sum of squared distances, we can model the pdf (probability density function), using **Gaussian kernels** and compute our cost as the **likelihood** of observing our data.





K = 3 Gaussian components. How likely is that data with this distribution?
pdf

We denote the entire GMM as **ϕ**, representing each Gaussian component with **μ** and **Σ** (covariance matrix).

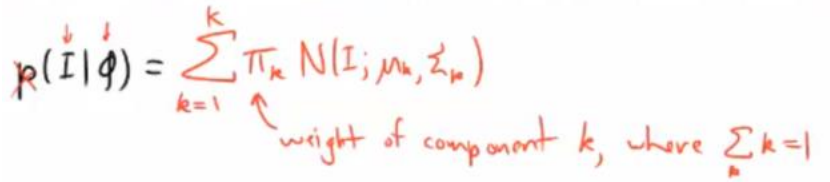


I: should be I_x, D: # dims
I_x: vector of intensities for pixel x

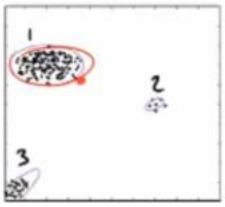
Then, the probability of observing I_x from that pdf is

$$N(I_x, \mu, \Sigma)$$

The likelihood of observing I_x from the entire GMM is



eg.

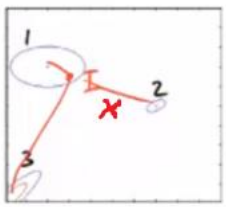


$\pi_1 = 0.8$
$\pi_2 = 0.05$
$\pi_3 = 0.15$

Generally speaking, π_k is the fraction of points that the k^{th} component explains.

In addition, each scatter point has a membership that is split among all the different Gaussian components.

eg.



"I" might have membership

$(0.7, 0.1, 0.2) = \tau_x$

Then likelihood of observing ALL the pixels is the product of their individual probabilities,

$$L(I|\phi) = \prod_x p(I_x|\phi)$$

Note that image intensities are a function of the motion parameters **Θ**. Thus, we wish to maximize

$$L(I|\theta,\phi) = \prod_x p(I_x^\theta|\phi)$$

move scatter points; gaussian pts

It is actually more convenient to consider the log-likelihood,

$$\log L(I|\theta,\phi) = \sum_x \log p(I_x^\theta|\phi)$$

Density Estimation (finding ϕ) (modeling) (find best ϕ)

Long story short: use estimation maximization (EM). Iterate between these two steps

Numerator: a problem
Denominator: all problems

Estimation step

$$\tau_{kx} = \frac{\pi_k \mathcal{N}(I_x^\theta|\phi_k)}{\sum_k \pi_k \mathcal{N}(I_x^\theta|\phi_k)}$$

Membership of pixel x to component k.

centroid weighted by mem'ship

<- rank 1 matrix weighted by π

Maximization step

$$\mu_k' = \frac{\sum_x \tau_{kx} I_x^\theta}{\sum_x \tau_{kx}}, \quad \text{Centroid of comp k.}$$

$$\Sigma_k' = \frac{\sum_x \tau_{kx} (I_x^\theta - \mu_k')(I_x^\theta - \mu_k')^T}{\sum_x \tau_{kx}} \quad \text{covariance.}$$

$$\pi_k' = \frac{\sum_x \tau_{kx}}{\sum_k \sum_x \tau_{kx}} \quad \text{Weight of comp k.}$$

Motion Adjustment (finding Θ)

Just like in calculus:

$$\frac{\partial}{\partial\theta} \log L(I^\theta|\phi) = 0$$

... crunch ... crunch ... crunch ...

$$\left(\sum_x \frac{1}{p(I_x^\theta|\phi)} \sum_{k=1}^K \pi_k \mathcal{N}_k(I_x^\theta) \frac{\partial I_x^\theta}{\partial\theta} \Sigma_k^{-1} \frac{\partial I_x^\theta}{\partial\theta}^T\right) \tilde{\theta} = \left(\sum_x \frac{1}{p(I_x^\theta|\phi)} \sum_{k=1}^K \pi_k \mathcal{N}_k(I_x^\theta) \frac{\partial I_x^\theta}{\partial\theta} \Sigma_k^{-1} (I_x^\theta - \mu_k)\right)$$

or simply $A\tilde{\theta} = b$

LHS: matrix, RHS: vector

Unit 5: Image Segmentation - Lesson 37 – Segmentation

Goal: to introduce segmentation

Red: liver, grey: lungs

Segmentation is the process of classifying pixels into groups that correspond to the same tissue type. A good example is classifying pixels as “liver or “non-liver” in this T1-weighted MRI. Or “lung” vs “non-lung” pixels.

Segmentation has many uses:

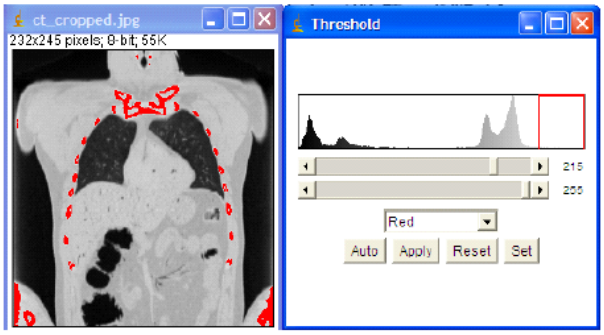
1. Measure volume of an organ
2. Render a 3D view of an organ
3. Surface-based registration

Thresholding

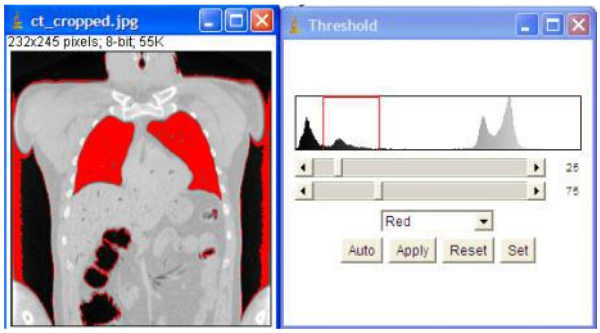
The simplest method for classifying pixels is based solely on their **intensity values**. In thresholding, you choose an upper and lower bound, and select pixels with intensities in that range.

Example: segment the bone in this CT scan. Bone is usually the brightest thing in a CT image.

ImageJ has a thresholding dialog that shows you the histogram. The red pixels are the ones between the thresholds.



How about the lungs?
No matter which thresholds we choose, some non-lung pixels get included.



Consider the histogram of an image that contains both lung and background.

There is **no intensity** that separates lung from background.

The problem is that thresholding does not take any **spatial information** into account – each pixel is evaluated **(on its intensity)** individually no matter where it is.

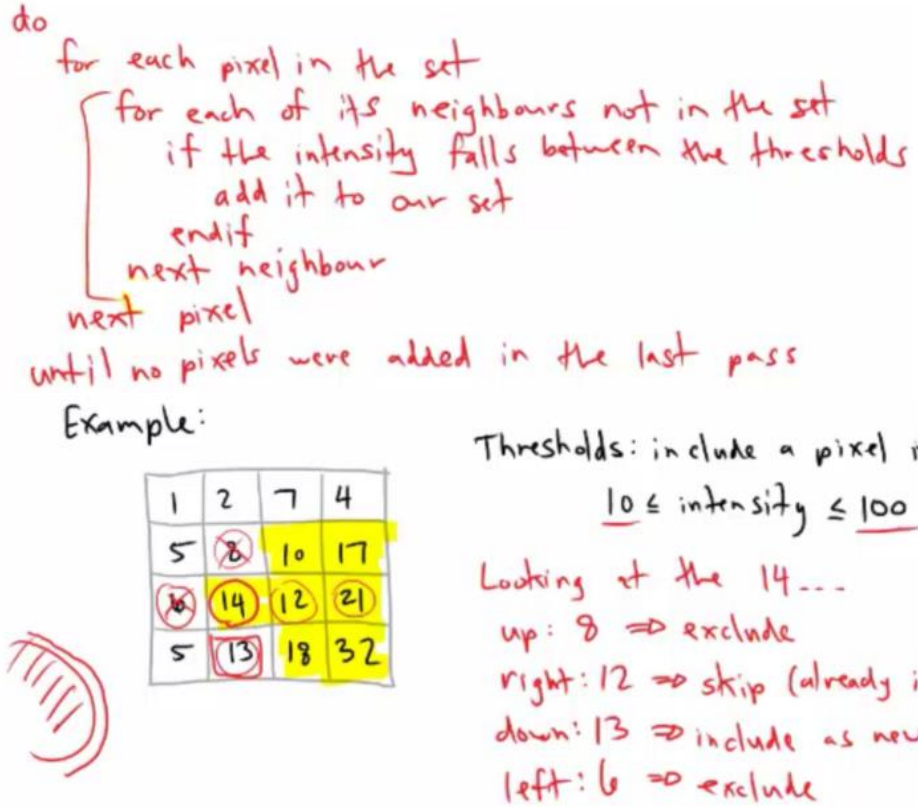
Clearly, we have to be able to build additional constraints into the segmentation process.

Lesson 38 – Region Growing

Goal: to use spatial continuity to guide our thresholding method for segmentation.

***region growing -> find fill impact**

The idea is to start with a small set of **"seed"** pixels, and grow out from them. It is an iterative process.



shaded region: already in set

(MATLAB demo – L38_region_growing)

- **Lungs fill up well**
- **Liver has leak**

You can make the method more sophisticated by coming up with different conditions for the inclusion/exclusion of pixels.

Examples:

1. Choose lower and upper thresholds based on currently-included pixels.
   i.e. compute μ and σ of selected pixels
   then low = μ – 2σ
   high = μ + 2σ
2. Use a local pixel quantity other than raw intensity

e.g. gradient magnitude, local variance, texture, etc.

Lesson 39 – k-Means Clustering

Goal: to investigate clustering as a way to perform segmentation, and the k-means method in particular.

k-means is a very simple clustering method. The dataset is represented as a scatter plot in some **feature space**. For example, a pixel could be represented by:

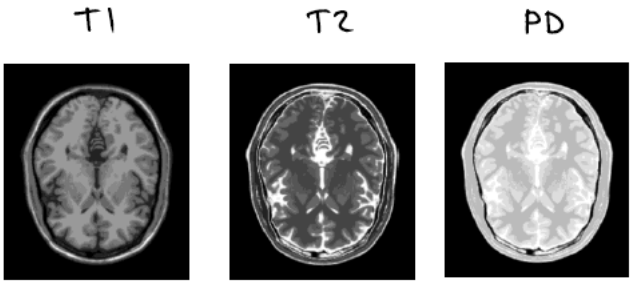
- **Intensity**
- **Gradient magnitude**
- **Laplacian**

Or, for data sets that have multiple images:

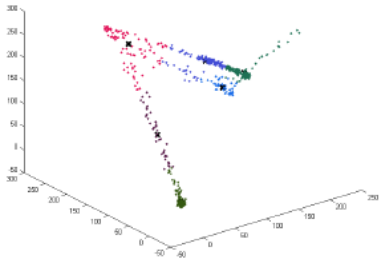
- **T1**
- **T2**
- **PD**

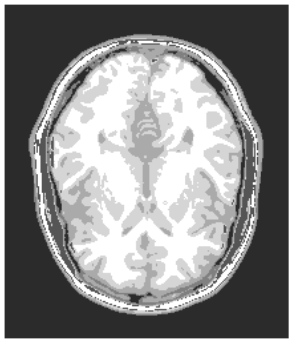
The user specifies **k**, the number of clusters he/she wants. Then here's the k-means algorithm:

1. **Randomly chose k regressors (pixels) as prototypes**
2. **Assign each scatter point ("feature vector") to its nearest prototype**
3. **Recalculate new prototypes (the mean of its members)**
4. **If the prototypes change significantly, go to step 2**



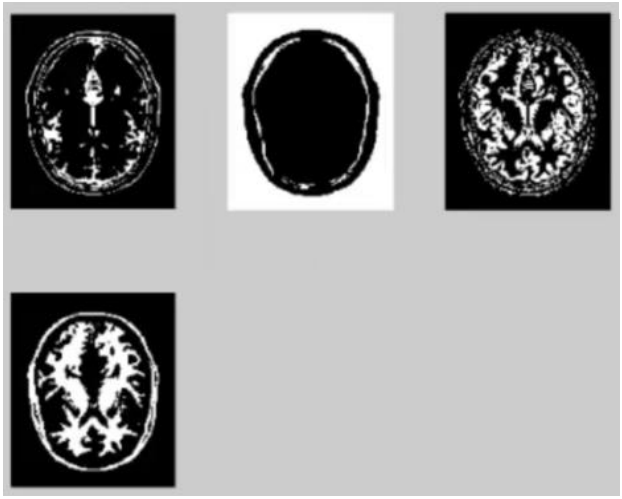
(MATLAB demo – L39_k_means)





Brain mask superimposed on brain

- **No scalp**

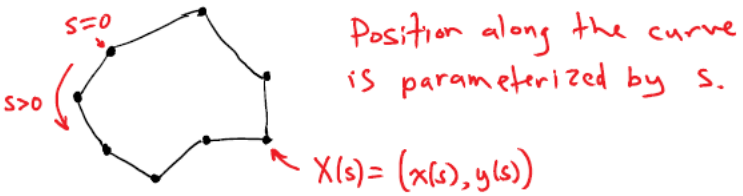


Lesson 40 – Snakes

Goal: to learn about active contours, how they can be represented, and how we can control them to perform segmentation.

An active contour is a **deformable boundary** on an image designed to converge to an anatomical boundary of some sort.

This deformable boundary is often represented by a number of nodes or vertices, a form of active contour we call **snakes**.



The nodes are moved in order to minimize a cost function. The cost function usually takes the form of an energy function.

$$E(X) = \int E_{int}(X(s))\,ds + \int E_{ext}(X(s))\,ds$$

Internal energy eg. elasticity, rigidity, etc.

External energy eg. image intensity/gradient, user constraints etc.

Internal Energy

Often modelled using

$$E_{int}(X) = \alpha \left|\frac{\partial X}{\partial s}\right|^2 + \beta \left|\frac{\partial^2 X}{\partial s^2}\right|^2$$

deriv of dist of pts

elastic term discourages stretching

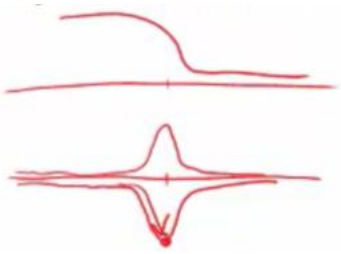
curvature term discourages bending

External Energy

Often modelled as a **"potential energy function"**, it takes on **smaller** values near **edges** or other objects of interest.

eg. $P(x,y) = -w\|\nabla(g * f)\|^2$

will stop or slow the contour at edges



w: control value

$\nabla(g * f)$: gradient magnitude

- **Like gravity, higher ball, more energy**
  - **Closer to edge, less energy. Further = more**

Energy Minimization

The procedure for finding the optimal contour **X** to minimize the energy **"functional" (E(x))** is called **variational calculus**. Just like in calculus, to find local optima of a function **f(x)** you set $\frac{df}{dx} = 0$, we set the variation of a functional to $\frac{dE}{dx} = 0$

***functional: takes function as input; in our case, a parametric curve. E(x) -> E(x(s))**

Skipping all the details, it essentially results in having to solve a PDE, in our case the Euler-Lagrange equation

$$\frac{\partial}{\partial s}\left(\alpha \frac{\partial X}{\partial s}\right) - \frac{\partial^2}{\partial s^2}\left(\beta \frac{\partial^2 X}{\partial s^2}\right) - \nabla P(X) = 0 \qquad (1)$$

Just like in anisotropic diffusion, we introduce a time variable and use it to evolve toward a **steady-state** solution of **(1)**

$$\gamma \frac{\partial x}{\partial t} = \frac{\partial}{\partial s}\left(\alpha \frac{\partial x}{\partial s}\right) - \frac{\partial^2}{\partial s^2}\left(\beta \frac{\partial^2 x}{\partial s^2}\right) - \nabla P(x)$$

Balancing-Force Formulation

Generalizing this approach to include external influences other than potential energy, we turn to a dynamic force formulation.

$$\mu \frac{\partial^2 x}{\partial t^2} = F_{damp}(x) + F_{int}(x) + F_{ext}(x) \qquad (\text{comes from } F=ma)$$

In segmentation, we often set μ to 0, and use $-d\frac{dx}{dt}$ as the viscous damping force, $F_{damp}(x)$. Thus, we get

$$\gamma \frac{\partial x}{\partial t} = F_{int}(x) + F_{ext}(x)$$

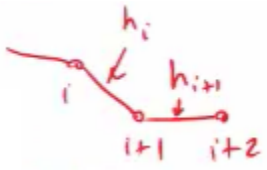
Internal Forces

As above…

$$F_{int}(x) = \frac{\partial}{\partial s}\left(\alpha \frac{\partial x}{\partial s}\right) - \frac{\partial^2}{\partial s^2}\left(\beta \frac{\partial^2 x}{\partial s^2}\right)$$

In a finite-difference framework,

$$\frac{\partial}{\partial s}\left(\alpha \frac{\partial x_i}{\partial s}\right) \approx \frac{1}{h_i^2}\left[\alpha_{i+1}\left(x_{i+1}^n - x_i^n\right) - \alpha_i\left(x_i^n - x_{i-1}^n\right)\right] \qquad (\text{assuming } h_i = h_{i+1})$$



and

$$\frac{\partial^2}{\partial s^2}\left(\beta \frac{\partial^2 x^+}{\partial s^2}\right) \approx \frac{1}{h_i^4}\left[\beta_{i-1}\left(x_{i-2}^+ - 2x_{i-1}^+ + x_i^+\right) - 2\beta_i\left(x_{i-1}^+ - 2x_i + x_{i+1}^+\right) + \beta_{i+1}\left(x_i^+ - 2x_{i+1}^+ + x_{i+2}^+\right)\right]$$

FD of 2nd derivative

β's unknown

$$X^+ = \begin{bmatrix} x_0 & y_0 \\ x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \\ x_n & y_n \end{bmatrix}$$

All these can be represented by a matrix operator

$$F_{int}(X^n) \cong AX^n$$ <- linear operator

Recall: $\gamma \frac{\partial X}{\partial t} = F_{int}(X) + F_{ext}(X)$

Thus, using finite-differencing for $\frac{\partial X}{\partial t}$, we have

$$\frac{X^n - X^{n-1}}{\Delta t} = AX^n + F_{ext}(X^{n-1})$$

Bring X^n to the left-hand side...

$$(I - \Delta t A)X^n = X^{n-1} + \Delta t F_{ext}(X^{n-1})$$

$$X^n = \underbrace{(I - \Delta t A)^{-1}}\left[X^{n-1} + \Delta t F_{ext}(X^{n-1})\right]$$

Does not change much, except that the spacing (h_i) can change.

⇒ efficient to compute LU factorization

$LU = I - \Delta t A$ $O(m^3)$ flops

↑ # of nodes

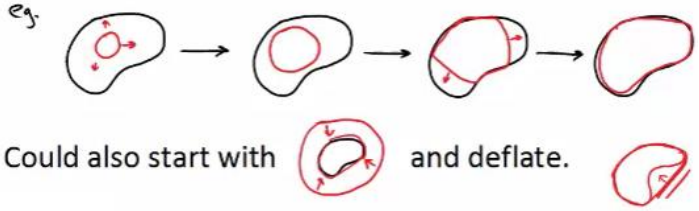
Then each iteration is $O(m^2)$ flops.

External Forces

Lots of options, limited only by your imagination. Here are some common ones.

Pressure Force:

Also called **"balloon force", grows** (or **shrinks**) the contour at a fixed rate. This allows one to start with a contour that isn't close to the desired solution.



Force pressure is defined as

$$\boldsymbol{F_p(x) = \omega N(x)}$$

where

N(x) is the outward normal vector (unit vector)

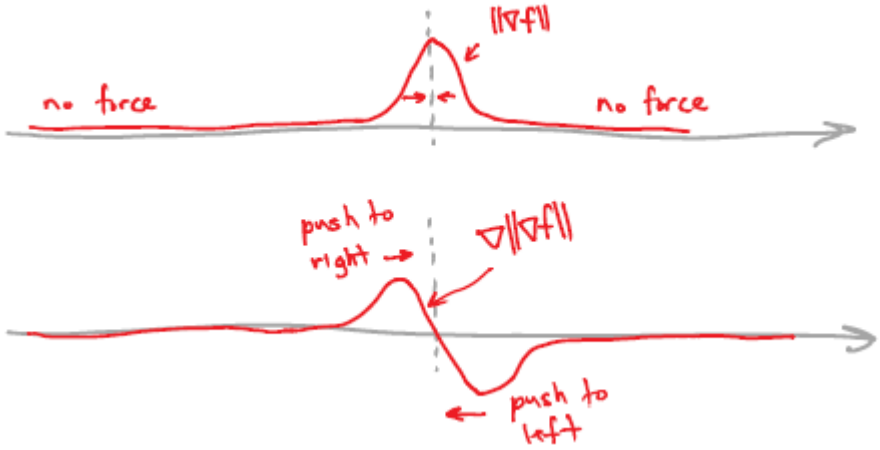
ω is positive for inflation, negative for deflation

Edge-Stopping Force:

This force is designed to stop and hold the snake on an edge.

Consider an image f, and the magnitude of its gradient, $\|\nabla f\|$

We want to use the **gradient** of $\|\nabla f\|$ to force our snake.



Hence, we get the force

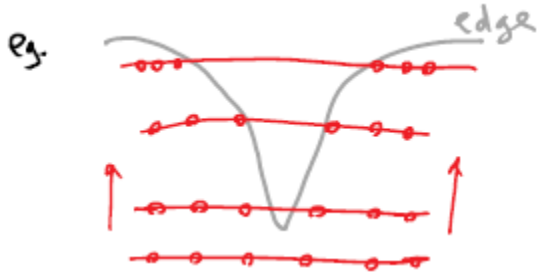
$$F_e(x) = \nabla \|\nabla f\| \quad \text{or} \quad \nabla \|\nabla f\|^2$$

It can also be helpful to use the gradient of a blurred version of f,

$$F_e(x) = \nabla \|\nabla (G * f)\|$$

Problem with Snakes

1. <u>Node spacing</u>: as the curve evolves, the points move around and can **spread apart**, especially if one is inflating a curve. Even if inflation is minimal, the points can still redistribute.



2. <u>Topological changes</u>: it can be tricky to detect and handle situations when the curve overlaps itself

For example, it might be appropriate to cleave a curve into two separate curves.

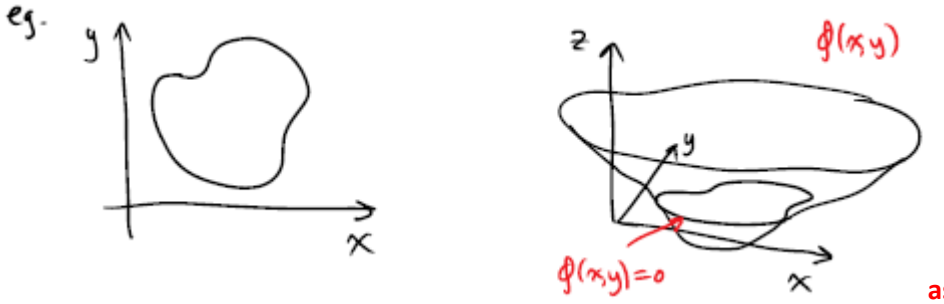
eg.

Fidgety to implement this break,
and then you have 2 curves & so on...

Lesson 41 – Introduction to Level Sets

Goal: an overview of the level set method for image segmentation.

The level set method is a different way to formulate the active contour idea. Instead of **explicitly** modelling the curve, the curve is **implicitly** modelled as a **zero level set** of a higher-dimension function. To model a curve in R^n, you use an embedding function in $R^n \rightarrow R$.



as curve changes, zero level set changes

Note: unless otherwise specified, in this course we will always assume that ϕ opens upwards **(WLOG)**.

Since $\phi: \mathbb{R}^n \rightarrow \mathbb{R}$, the curve X is given by the inverse of ϕ, $\phi^{-1}: \mathbb{R} \rightarrow \mathbb{R}^n$. That is,

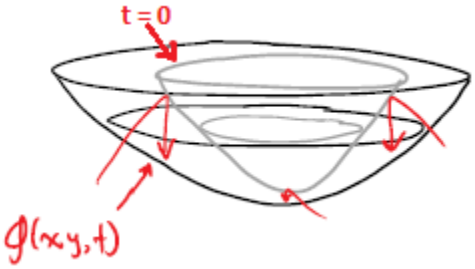
$$X = \phi^{-1}(0) \equiv \{\text{the set of all } (x,y) \text{ pts. s.t. } \phi(x,y) = 0\}$$

Now consider an evolving level set, $X(s,t) = (x(s,t), y(s,t))$

moves by embedding function changing

We can still model this with level sets: $\boldsymbol{\varphi(x,y,t)}$

As time progresses, the embedding function **ϕ** can move and change so that the zero level set takes on the desired curve



Since the curve is defined implicitly, it is easy to address topological changes.

Level Set Evolution

Consider the point $X(s,t) = (x(s,t), y(s,t))$ as the curve evolves.

Hence, $\phi(x(s,t), y(s,t), t) = 0 = \phi(x(s,t+\Delta t), y(s,t+\Delta t), t+\Delta t)$

Using a Taylor expansion,

$$\phi(x(s,t+\Delta t), y(s,t+\Delta t), t+\Delta t)$$

$$= \phi(x(s,t), y(s,t), t) + \frac{d}{dt}\phi(x(s,t), y(s,t), t)\Delta t + \cancel{O(\Delta t^2)} \quad \text{(ignore)}$$

$$= \frac{\partial \phi}{\partial t} + \frac{\partial \phi}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial \phi}{\partial y}\frac{\partial y}{\partial t}$$

$$= \cancel{\phi}^{0} + \left[\frac{\partial \phi}{\partial t} + \frac{\partial \phi}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial \phi}{\partial y}\frac{\partial y}{\partial t}\right]\Delta t \cong 0$$

$$\Rightarrow \frac{\partial \phi}{\partial t} + \frac{\partial \phi}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial \phi}{\partial y}\frac{\partial y}{\partial t} = 0$$

$$\frac{\partial \phi}{\partial t} = -\nabla\phi \cdot \left(\frac{\partial x}{\partial t}, \frac{\partial y}{\partial t}\right)$$ ← velocity of the point $(x(s,t), y(s,t))$

The velocity $(x(s,t), y(s,t))$ is the motion of the curve, and we want to manipulate it to achieve the curve we're after. We can represent our velocity as a speed **V_N** in the direction normal (orthogonal) to the curve. (note: speed **V_N** is a scalar, not a vector). The **gradient** of a function is orthogonal to its level curves. Thus, our velocity is parallel to **$\nabla\varphi$**.

So we can represent $\left(\frac{\partial x}{\partial t}, \frac{\partial y}{\partial t}\right)$ as $V_N \frac{\nabla\phi}{\|\nabla\phi\|}$

Thus, our embedding function evolves according to

$$\frac{\partial \phi}{\partial t} = -\nabla\phi \cdot \left(V_N \frac{\nabla\phi}{\|\nabla\phi\|}\right) = -V_N \frac{\|\nabla\phi\|^2}{\|\nabla\phi\|}$$

$$\Rightarrow \boxed{\frac{\partial \phi}{\partial t} = -V_N \|\nabla\phi\|} \quad (1)$$

(MATLAB demo – L40_snakes: see notes)

Lesson 42 – Speed Functions

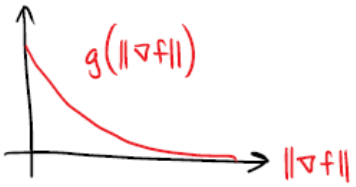
Goal: find out how to control the speed of the level set for image segmentation.

Recall that the embedding function ϕ evolves according to the PDE $\frac{\partial \phi}{\partial t} = -V_N \|\nabla \phi\|$.

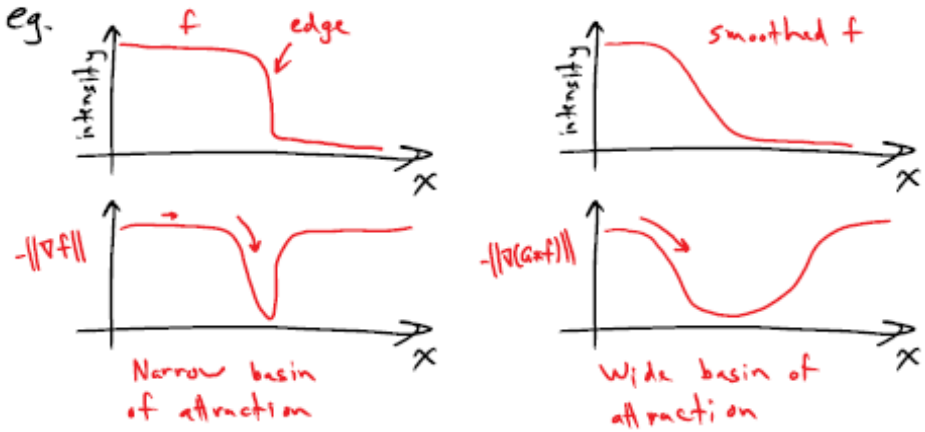
The speed function, V_N, can be any smooth scalar function.
To use level sets to segment images, we use image information (such as image gradients, etc.) to determine V_N.

Edges:
If we want the curve to stop at the strong edges, then **V_N** has to be **close to zero** when the gradient of our image is **large** (where have we seen this before?).

$g(r) = e^{-\alpha r}$



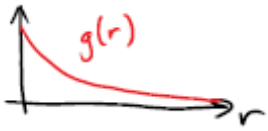
It is common to use the gradient of the **smoothed** image to widen the basin of attraction.



$\therefore V_N = g(\|\nabla(G*f)\|) = e^{-\alpha\|\nabla(G*f)\|}$

Another option:

$g(r) = \frac{1}{1+|r|}$



Curvature:

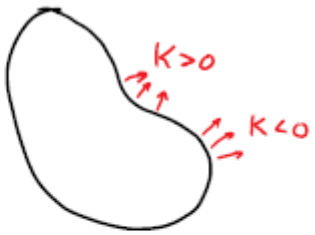
We can also add a component to the speed function that keeps the curve smooth this is similar to the rigidity term in the snakes formulation. One of its purposes is it prevent the set from leaking through tiny openings in the edge.

In **B**, the curve kept going where the edge was **absent** because the speed function only depended on the gradient magnitude. However, in **A** the curve stops because leaking through the hole would require the curve to take on a very high **curvature**, something that is discouraged if the speed function is chosen appropriately.

Curvature, **K**, can be computed from the embedded function using

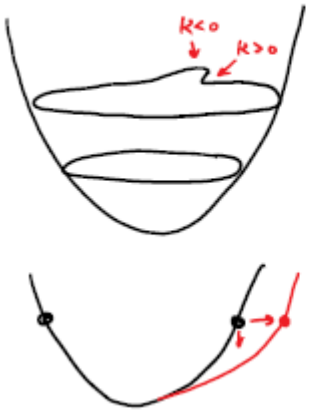
$$K = -\nabla \cdot \frac{\nabla \phi}{\|\nabla \phi\|}$$

$$= -\left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right) \cdot \underbrace{\frac{\left(\frac{\partial \phi}{\partial x}, \frac{\partial \phi}{\partial y}\right)}{\|\nabla \phi\|}}_{\text{unit normal vector}}$$



How can curvature influence ϕ? Let $V_n \propto K$.

$$\frac{\partial \phi}{\partial t} = -\varepsilon K \|\nabla \phi\|$$



If $K > 0 \Rightarrow \frac{\partial \phi}{\partial t} < 0$

Hence, ϕ decreases which pushes the curve out.

If $K < 0 \Rightarrow \frac{\partial \phi}{\partial t} > 0$

Hence, ϕ increases which pushes the curve in.

Putting these speed factors together,

$$\frac{\partial \varphi}{\partial t} = -g\left(\|\nabla(G*f)\|\right)\left(V_0 + \varepsilon\kappa\right)\|\nabla\varphi\|$$

Where

g is one of the edge-stopping functions

V_0 is a constant inflation (+) or deflation (-) speed

κ is curvature $\left(-\nabla\cdot\nabla\varphi/\|\nabla\varphi\|\right)$

ε is a chosen smoothness constant

Lesson 43 – Implementing Level Sets

Goal: to find out how to numerically solve the level-set evolution equation.

Recall the PDE that governs the evolution to the level-set embedding function,

$$\frac{\partial \phi}{\partial t} = -V_N \|\nabla \phi\|$$

Time Derivative

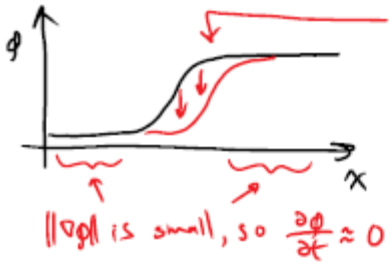
We will use **forward** differencing to give us an explicit time-stepping method.

$$\frac{\phi^{t+1} - \phi^t}{\Delta t} \cong \frac{\partial \phi}{\partial t} \Rightarrow \phi^{t+1} = \phi^t - \Delta t V_N^t \|\nabla \phi^t\|$$

Spatial Derivative

Estimating the gradient of ϕ turns out to be more problematic. There are important issues to keep in mind in order for the numerics to give good results.

Consider a wave going from left to right

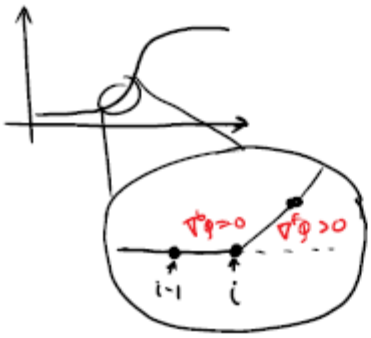


We need $\frac{\partial \phi}{\partial t} < 0$ here.

∴ To move to the right, $V_N > 0$.

If our wave travelling to the right, then we should look **"upwind" (left)** when computing our derivatives.

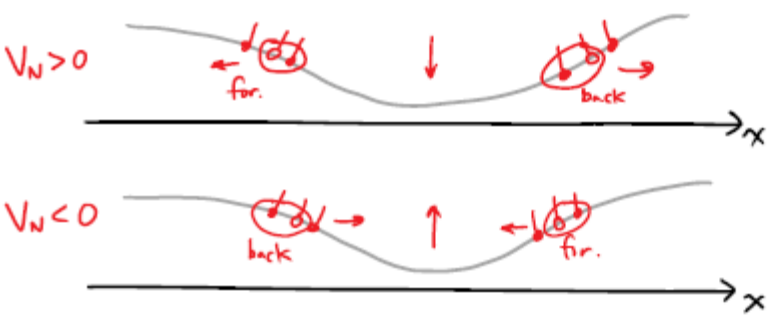
Let $\nabla^b\phi$ be the backward diff. approx. of $\frac{\partial\phi}{\partial x}$ (or $\nabla\phi$),
and $\nabla^f\phi$ be the forward diff. approx.



Using back. diff. $\nabla^b\phi = 0$
$\Rightarrow$ no change in ϕ_i (smooth landing)

Using forward diff. $\nabla^f\phi > 0$
$\Rightarrow$ ϕ_i drops down (overshoot)

$\frac{\partial\phi}{\partial x} < 0$ $\qquad$ $\frac{\partial\phi}{\partial x} > 0$



Implementation

- Start with an initial embedding function ϕ
- Compute speed function from image

$$\text{grad\_f} = \text{gradient}(f, \text{'central'})$$

$$V = \frac{1}{1 + \|\text{grad\_f}\|^2}$$

- Compute partial derivatives of ϕ using both forward and backward differencing

$$\frac{\partial \varphi}{\partial x} \begin{cases} \nabla_x^f = \text{MyDerivative}(\varphi, x, \text{'forward'}) \\ \nabla_x^b = \text{MyDerivative}(\varphi, x, \text{'backward'}) \end{cases}$$

$$\frac{\partial \varphi}{\partial y} \begin{cases} \nabla_y^f = \dots \\ \nabla_y^b = \end{cases}$$

- Combine like partial derivatives, choosing forward or backward as appropriate

$$\nabla_x^+ = \min(\nabla_x^f, 0) + \max(\nabla_x^b, 0)$$

$$\nabla_x^- = \max(\nabla_x^f, 0) + \min(\nabla_x^b, 0)$$

Likewise for ∇_y^+ and ∇_y^-.

- Choose upwind derivative, according to speed function

$$\nabla\varphi(x,y) = \begin{cases} (\nabla_x^-, \nabla_y^-) & \text{if } V(x,y) \le 0 \\ (\nabla_x^+, \nabla_y^+) & \text{if } V(x,y) > 0 \end{cases}$$

- Take time step

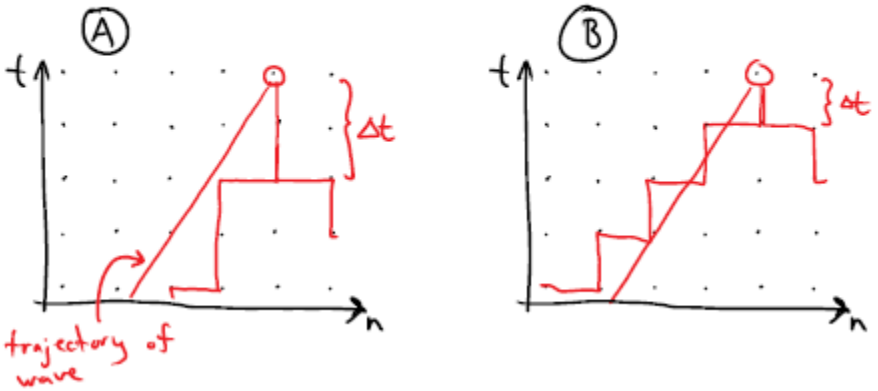
$$\varphi^{t+1}(x,y) = \varphi^t(x,y) - \Delta t\, V(x,y)\, \nabla\varphi^t(x,y)$$ **<- gradient from previous step**

CFL Condition (Courant-Friedrichs-Lewy Condition)

We need to make sure that our time steps are not too big.

As the "wind blows", we must make sure that the information keeps pace with our computations.

eg. Computing φ_n^{t+1} involves φ_n^t, φ_{n-1}^t and φ_{n+1}^t



As you go back in time, **A** shows that the basin of influence **does not include** the waves, so will be unstable. In **B**, the basis of **includes the wave**, so will be more dependable.

In general, you must choose a **Δt** small enough so that the **fastest-moving** things do not move more than **one spatial sample** per step.

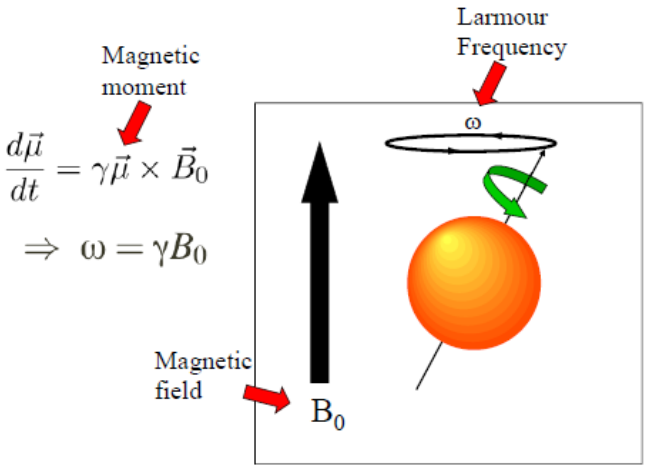
$$V_{max}\frac{\Delta t}{\Delta x} \leq 1 \quad \text{for explicit time stepping.}$$

(segment_brain demo)

Lesson 44 – Theory of MRI Reconstruction

Goal: to find how the MR signal can be turned into tomographic images.

Dipole Spin



Bloch's Equation

Net magnetization vector $\vec{M}$

$$\vec{M} = \vec{M}_x + \vec{M}_y + \vec{M}_z = M_x\vec{i} + M_y\vec{j} + M_z\vec{k}$$

Bloch's equation governs the behaviour of $\vec{M}$

$$\frac{d\vec{M}}{dt} = \gamma\vec{M} \times \vec{B}_0 - \frac{1}{T_2}\left(M_x\vec{i} + M_y\vec{j}\right) - \frac{1}{T_1}\left(M_z - M_0\right)\vec{k}$$

Larmour precession | Transverse (x-y) decay | Longitudinal (z) decay

(see page 108 of Jeff's thesis for explanation of T_1 and T_2.)

Dynamics of M_{xy}

Recall:

$$\frac{d\vec{M}}{dt} = \gamma\vec{M} \times \vec{B}_0 - \frac{1}{T_2}\left(M_x\vec{i} + M_y\vec{j}\right) - \frac{1}{T_1}\left(M_z - M_0\right)\vec{k}$$

$$\frac{d\vec{M}}{dt} = \gamma\vec{M} \times \left(\vec{B}_0 + \vec{G}\cdot\vec{x}\right) - \frac{1}{T_2}\left(M_x\vec{i} + M_y\vec{j}\right) - \frac{1}{T_1}\left(M_z - M_0\right)\vec{k}$$

We introduce a gradient in the strength of the magnetic field.
The gradient is in the direction $\vec{x}$, with no z-component.

In matrix form…

$$\frac{d\vec{M}}{dt} = -\begin{bmatrix} \frac{1}{T_2} & -\gamma\left(B_0 + \vec{G}\cdot\vec{x}\right) & 0 \\ \gamma\left(B_0 + \vec{G}\cdot\vec{x}\right) & \frac{1}{T_2} & 0 \\ 0 & 0 & \frac{1}{T_1} \end{bmatrix}\vec{M} + \frac{1}{T_1}\vec{M}_0$$

Solution for M_{xy}

T2 Relaxation

$$M_{xy}(x,y,t) = ce^{-i\gamma B_0 t}e^{\frac{-t}{T_2}}e^{-i(k_x x+k_y y)}$$

Normal precession at Larmour frequency

Frequency/Phase state (the k' s depend on the gradients)

$$k_x(t) = \int_0^t G_x(\tau)d\tau \qquad k_y(t) = \int_0^t G_y(\tau)d\tau$$

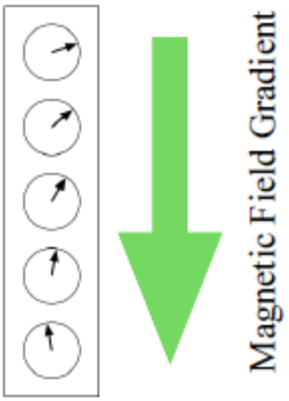
MR Signal

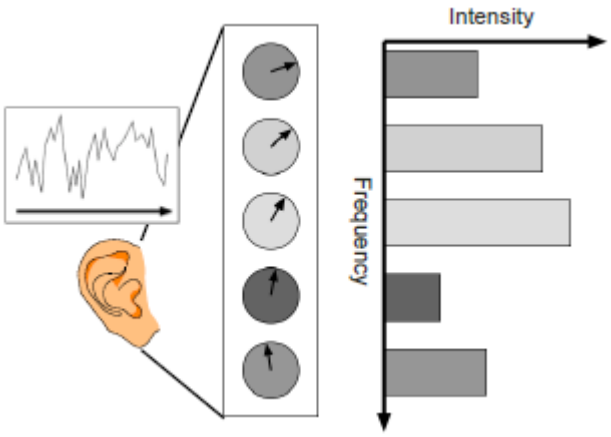
The MR signal is the sum of all the excited M_{xy}' s (ignoring t now).

$$S(k_x,k_y) = \iint M_{xy}(x,y)e^{-i(k_x x+k_y y)}dxdy$$

This looks just like a Fourier Transform, where (k_x, k_y) are the frequency variables. Thus, its inverse is just like the inverse FT,

$$M_{xy}(x,y) = \iint S(k_x,k_y)e^{i(k_x x+k_y y)}dk_x dk_y$$

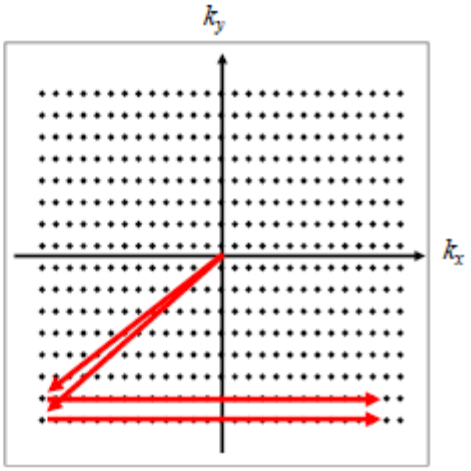




_k_-Space Traversal

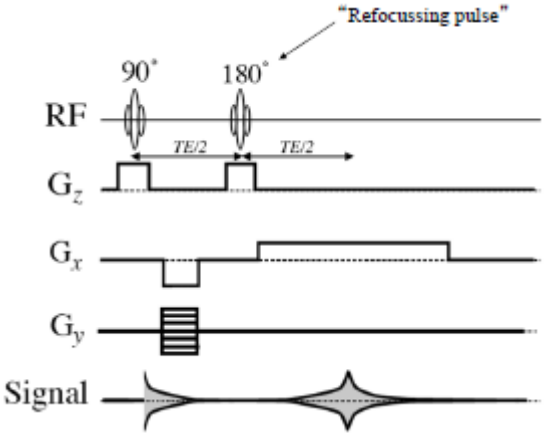
$$k_x(t) = \int_0^t G_x(\tau)d\tau$$

$$k_y(t) = \int_0^t G_y(\tau)d\tau$$



How to control the gradients

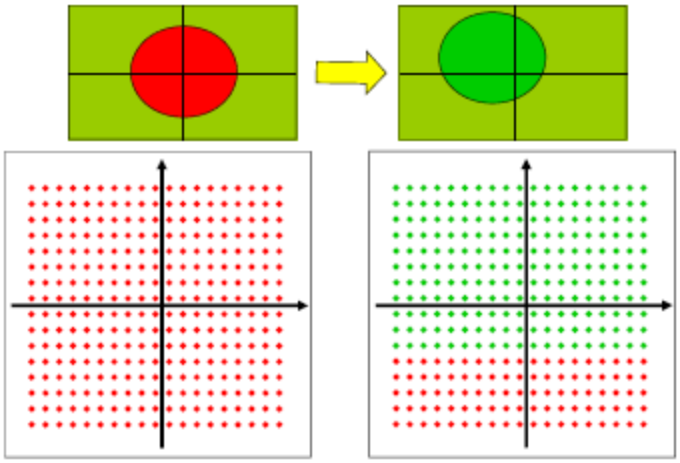
Spin-Echo Imaging



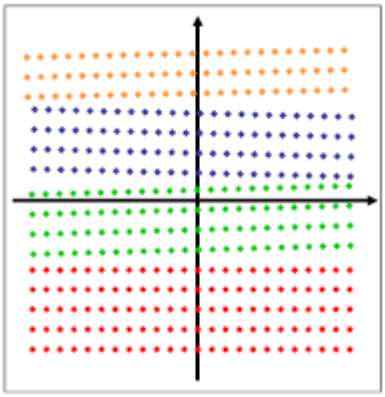
Motion Corruption

- The MR scanner collects the samples in *k*-space no matter the patients does
- If the patient moves part way through the scan, the data collected will not be consistent, so will not reconstruct to the correct image

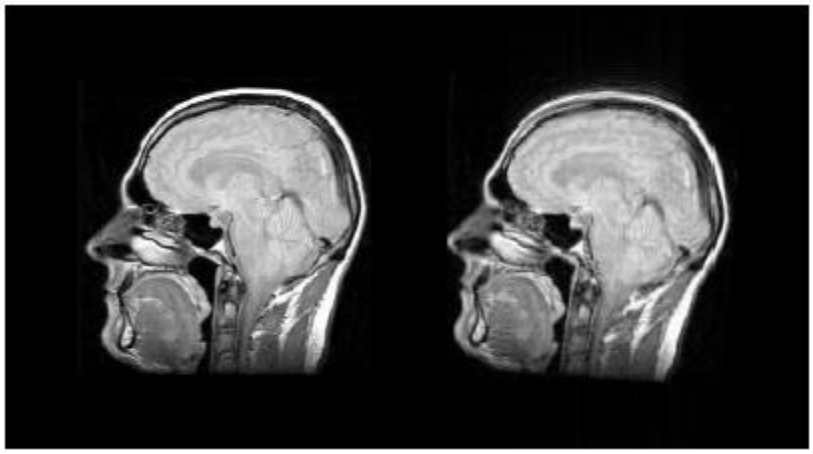
Intra-scan Motion in *k*-Space



Corrected *k*-space

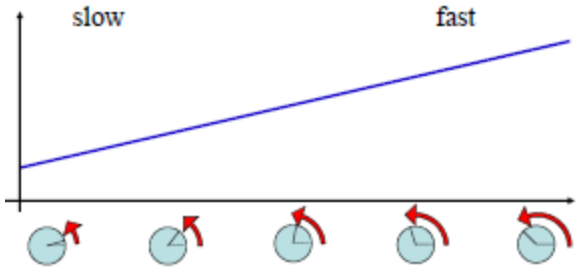


Intra-scan Motion

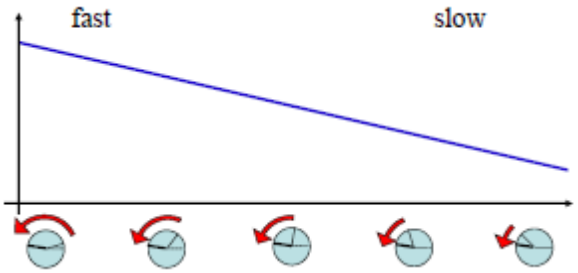


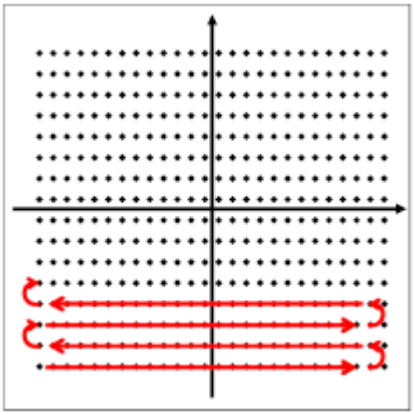
Gradient-Echo Imaging

Instead of flipping all the dipoles by 180°, reverse the phase difference to refocus the dipoles.



Gradient-Echo Imaging

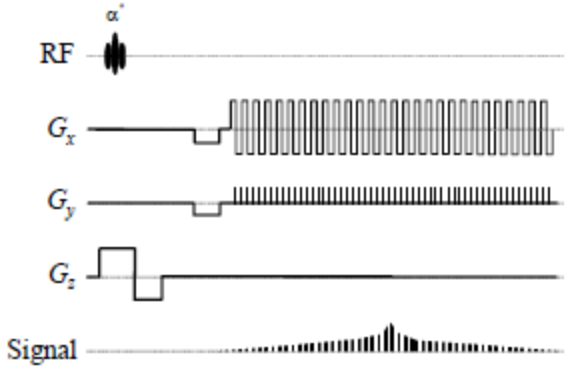




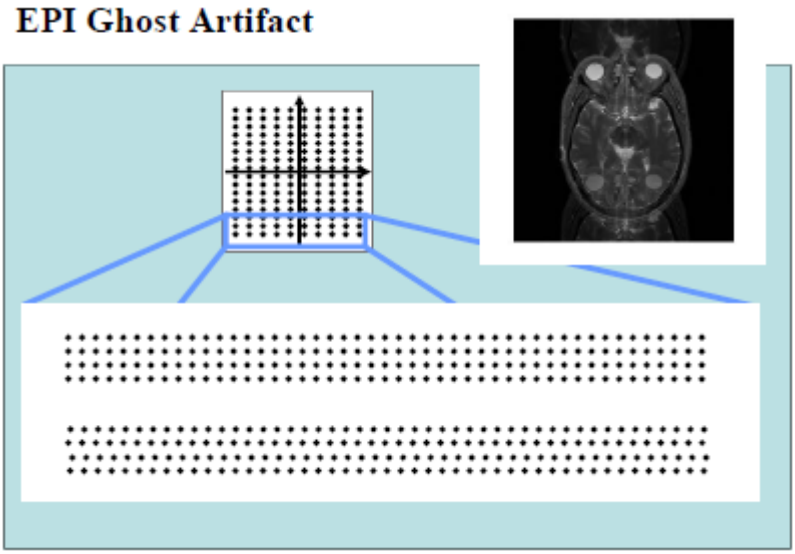
Echo-Planar Imaging (EPI)

A gradient-echo technique that allows one to collect a whole slice in one excitation. Exhibits T2* contrast, not T2 contrast.

EPI Scan Sequence



EPI Ghost Artifact

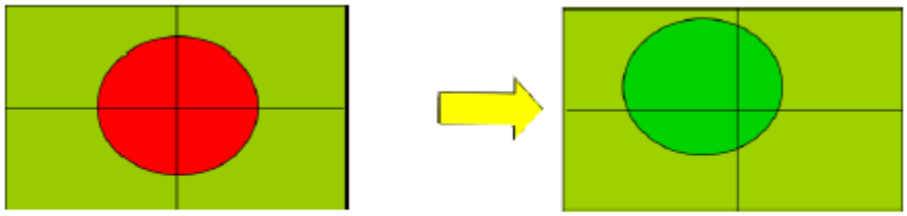


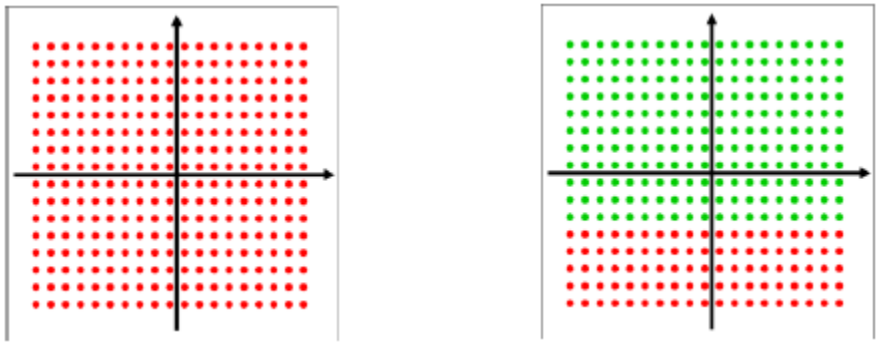
Lesson 45 – MRI Motion Compensation

Goal: to investigate how patient motion can corrupt MR images, and some methods for correcting it.

The MR scanner collects the samples in *k*-space no matter what the patient does.
If the patient moves part way through the scan, the data collected will not be consistent, so will not reconstruct to the correct image. This motion that happens during the acquisition of a single image called **intrascan** motion.

Start of scan **moves part way through scan** **end of scan**

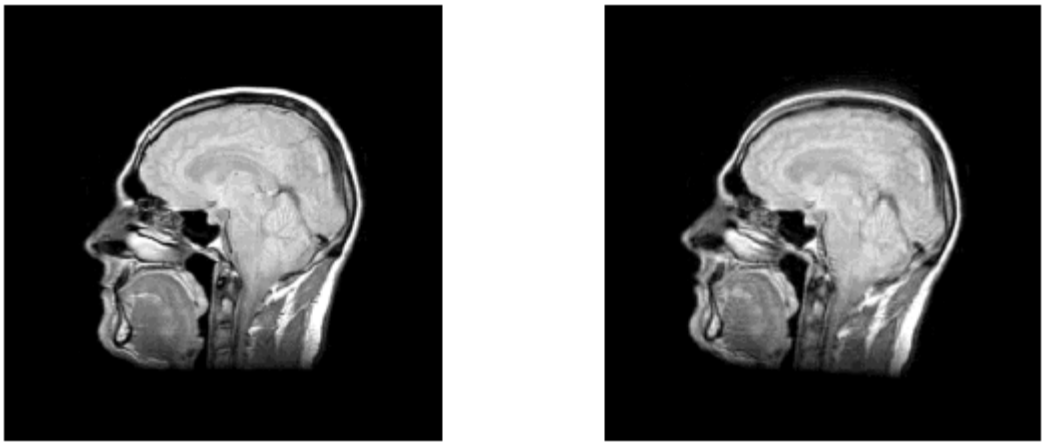




Movement here worst around DC (in between green, red)

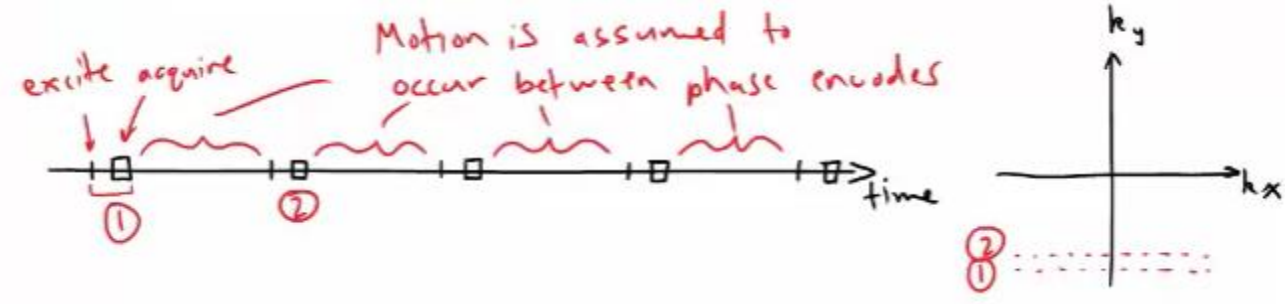
These inconsistencies cause **"motion artifacts"** in the reconstructed images.

Motion-free **Motion-corrupted**

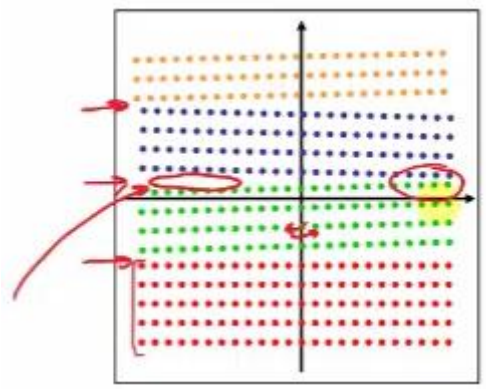


Ringing artifact on horizontal edges

Much of the effects of motion can be corrected by **post processing**. Most methods assume that motion is slow enough that motion within a single "phase encode" (row in k-space) is **negligible**.



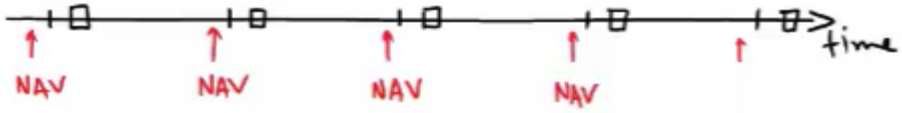
A translation in the patient causes the signal (the anatomy) to shift, which causes a phase ramp in its Fourier coefficients.
If you know the translation, you can undo it by simply changing the **phases** of the k-space samples using the appropriate **phase ramp**.

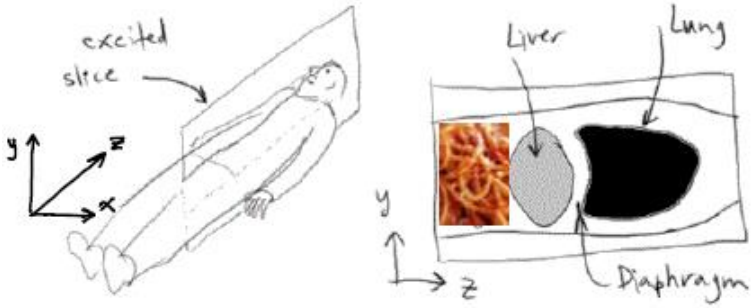


Rotation of the patient will **rotate** the corresponding content in the frequency domain.
If you know the rotations, you can compensate (to some extent) by **counter-rotating** the samples.
Information is lost if there are **under-sampled** regions. **e.g. monitor breathing, when exhale -> scan**

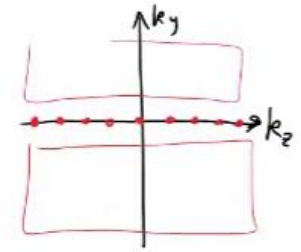
Navigator Echoes

Navigator echoes, or **NAVs**, are small sets of *k*-space samples that are acquired between imaging excitations. Their purpose is to gain an estimate of the patient's position. The echoes commonly consist of a few lines in *k*-space. These ultra-fast acquisitions are **interleaved** throughout the scanning sequence, and are useful in intra-scan motion correction because they offer a **repeated** set of measurements that can be used to **estimate position**.



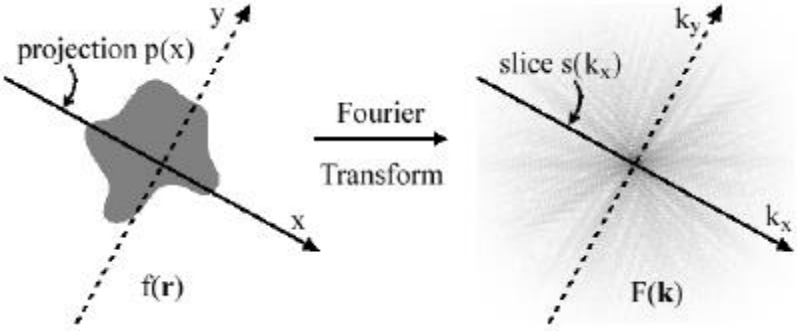


One could use all 3 gradients to acquire an image like above, but that would take too long (many excitation cycles).
Instead, what would happen if, after exciting the slice, only the **z-gradient** was used for spatial encoding?
i.e. **no y-gradient was used**

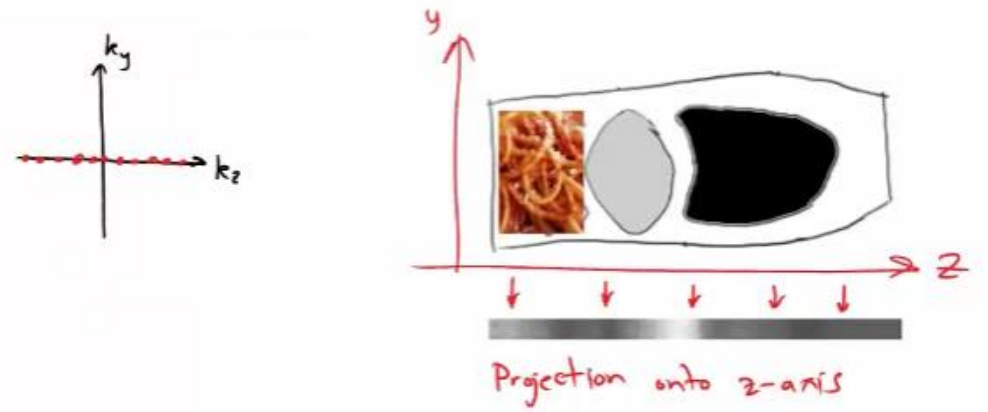


Performing the reconstruction (via a 1D IFFT), this gives a 1D signal. What does it represent?
Hint: unsampled coefficients are all zeros. What would the 2D IFFT give you?

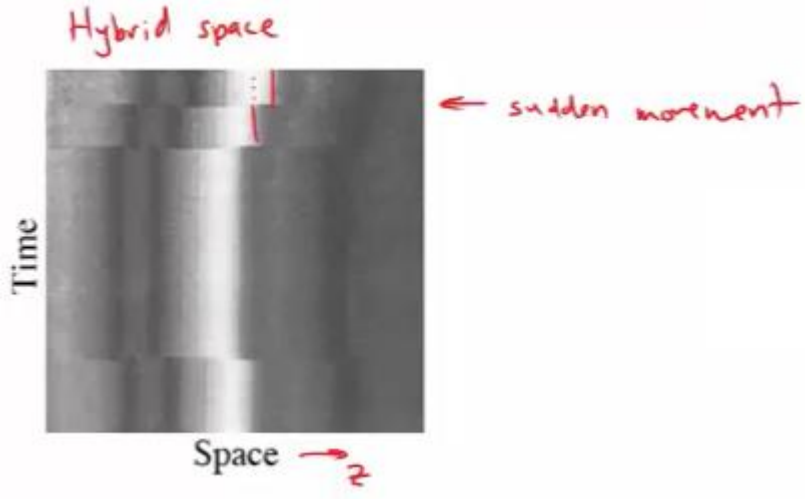
Recall the Projection Slice Theorem:



One line of *k*-space samples gives you a **projection** of the spatial-domain signal.

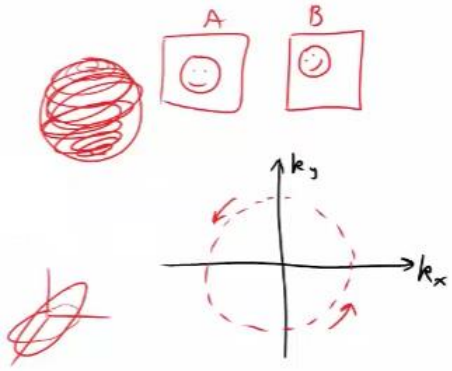


Collecting these projections during the scan can give useful information about motion during the scan.



***FT+Euclidean**

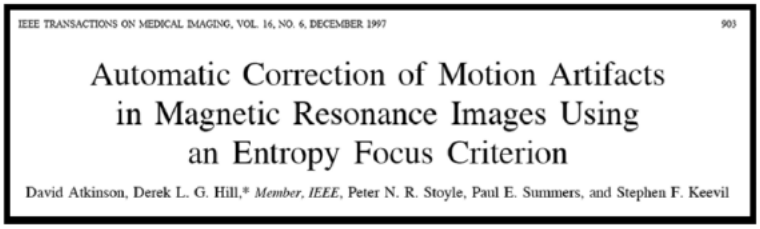
An **"orbital NAV"** collects a circle of samples in *k*-space, and allows one to determine both **translation and rotation**.



- **also spherical map**
- **3D frequency domain**
- **Have built-in detectors**

Entropy Focussing

If you do not now the patient motion, there are some methods to try to guess the motion.

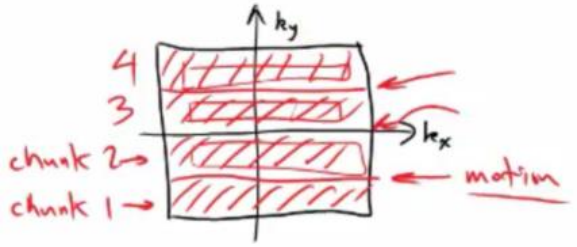


This approach is based on the fact that most movements cause artefacts to appear in the background and other dark parts of the images. These additional tend **increase** the **entropy** of the image.

If assume the motion is fairly slow, we can use a multi-resolution approach in time.

Here is the idea:

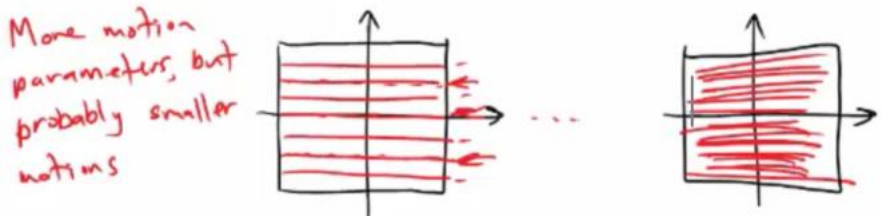
1. Group phase encodes into chronological chunks, and assume motion only occurs between these chunks.



2. Choose motion parameters, and apply the corresponding correction to each chunk. **(like gradient descent)**
3. Reconstruct the image (IFFT).
4. Compute the **entropy** of the reconstructed image.
5. Adjust the motion parameters to reduce the image **entropy**.

Optimization: minimize the image **entropy** by iterating the steps above.

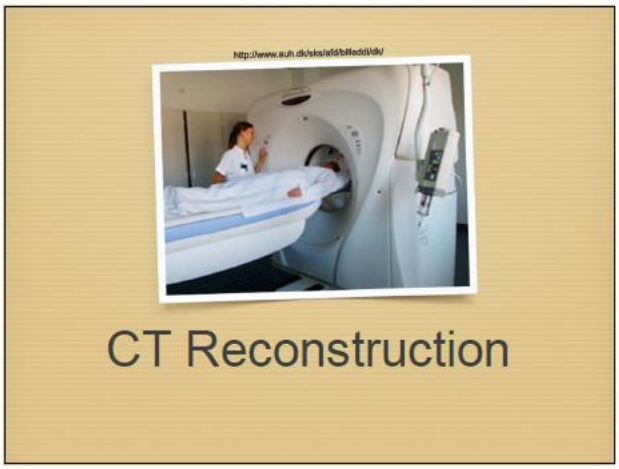
Once it's done, subdivide the temporal chunks into smaller chunks and repeat the optimization.

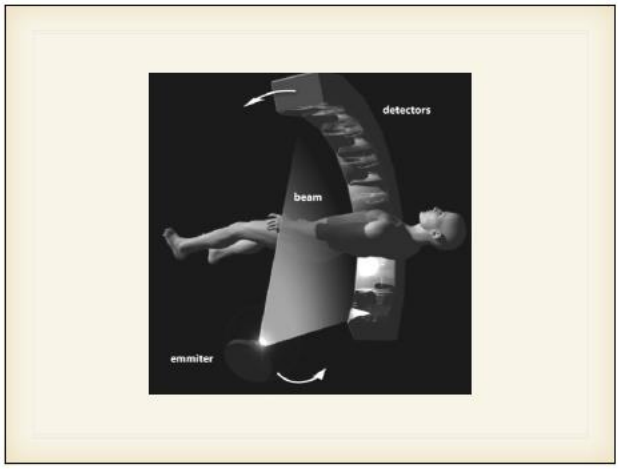


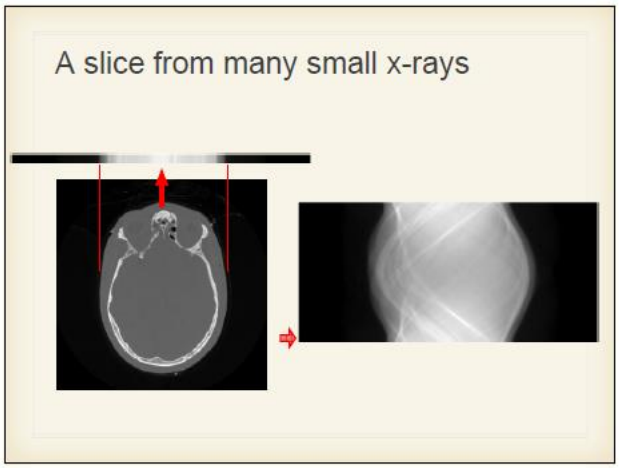
The translation corrections can be applied using Fourier Shift Theorem, and rotation corrections are implemented by rotating the *k*-space samples.

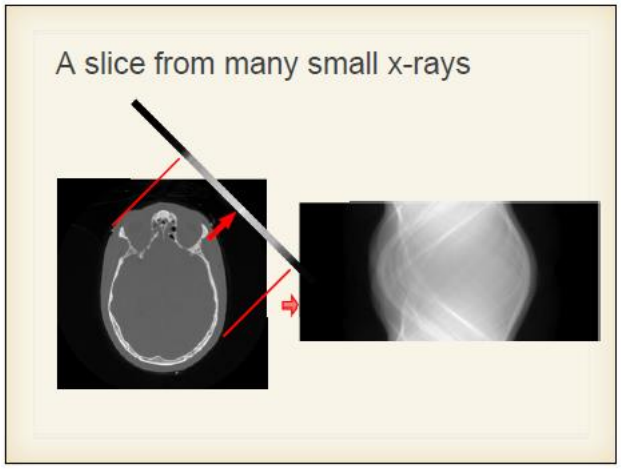
Note that the original *k*-space data is already in the **frequency** domain, so these corrections are being applied to the raw samples.

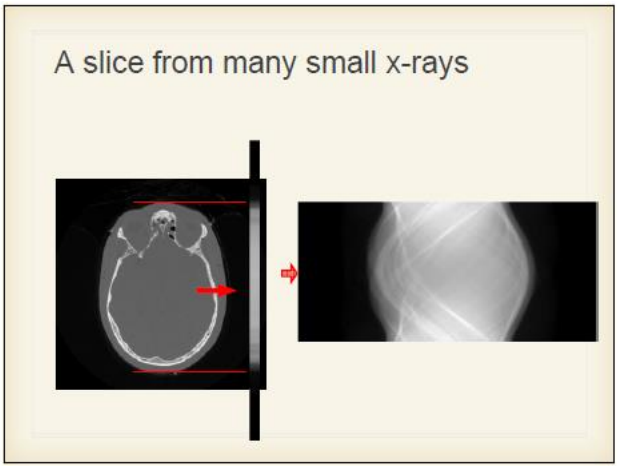
Lesson 46 – Algebraic CT Reconstruction

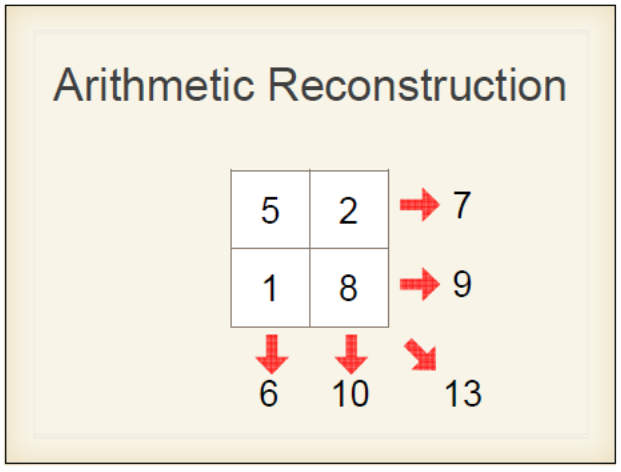












Arithmetic Reconstruction

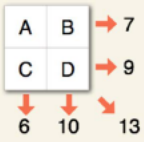
A	B	→ 7
C	D	→ 9
↓ 6	↓ 10	↘ 13

Arithmetic Reconstruction

A	B	→ 7
C	D	→ 9
↓ 6	↓ 10	↘ 13

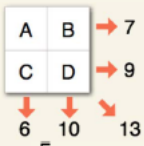
A+B=7
C+D=9
A+D=13
A+C=6
B+D=10

Arithmetic Reconstruction



$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} = \begin{bmatrix} 7 \\ 9 \\ 13 \\ 6 \end{bmatrix}$$

Arithmetic Reconstruction



$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} \approx \begin{bmatrix} 7 \\ 9 \\ 13 \\ 6 \\ 10 \end{bmatrix} ?$$

$$\mathbf{MX} \approx \mathbf{P}$$

Least-Squares Solution

$$\mathbf{MX} = \mathbf{P}$$

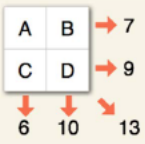
$$\mathbf{M}^{\mathrm{T}}\mathbf{MX} = \mathbf{M}^{\mathrm{T}}\mathbf{P}$$

$$\mathbf{X} = \left(\mathbf{M}^{\mathrm{T}}\mathbf{M}\right)^{-1}\mathbf{M}^{\mathrm{T}}\mathbf{P}$$

$$\text{Minimizes} \quad \|\mathbf{MX} - \mathbf{P}\|_2^2$$

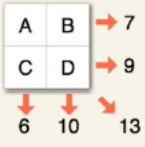
Iterative Reconstruction

A	B	
C	D	



Iterative Reconstruction

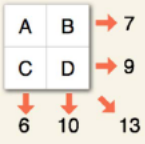
3.5	3.5	→ 7
4.5	4.5	→ 9



Distribute row-sums across each row.

Iterative Reconstruction

3.5	3.5	
4.5	4.5	
↓ 8	↓ 8	



Split new residuals evenly over columns.

Iterative Reconstruction

2.5	4.5
3.5	5.5

6 10

A	B	→ 7
C	D	→ 9
↓ 6	↓ 10	↘ 13

Split new residuals evenly over columns.

Iterative Reconstruction

2.5	4.5
3.5	5.5

↘ 8

A	B	→ 7
C	D	→ 9
↓ 6	↓ 10	↘ 13

And now the diagonal vs. off-diagonals.

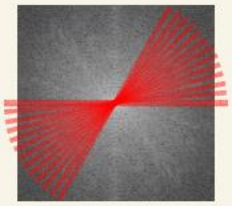
Iterative Reconstruction

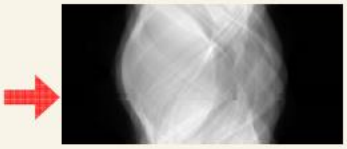
5	2
1	8

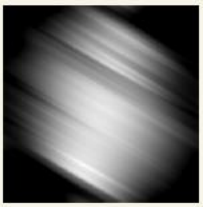
↘ 13

A	B	→ 7
C	D	→ 9
↓ 6	↓ 10	↘ 13

And now the diagonal vs. off-diagonals.







Lesson 47 – CT Back Projection

Goal: to see how the Fourier Projection Theorem can help us in CT reconstruction.

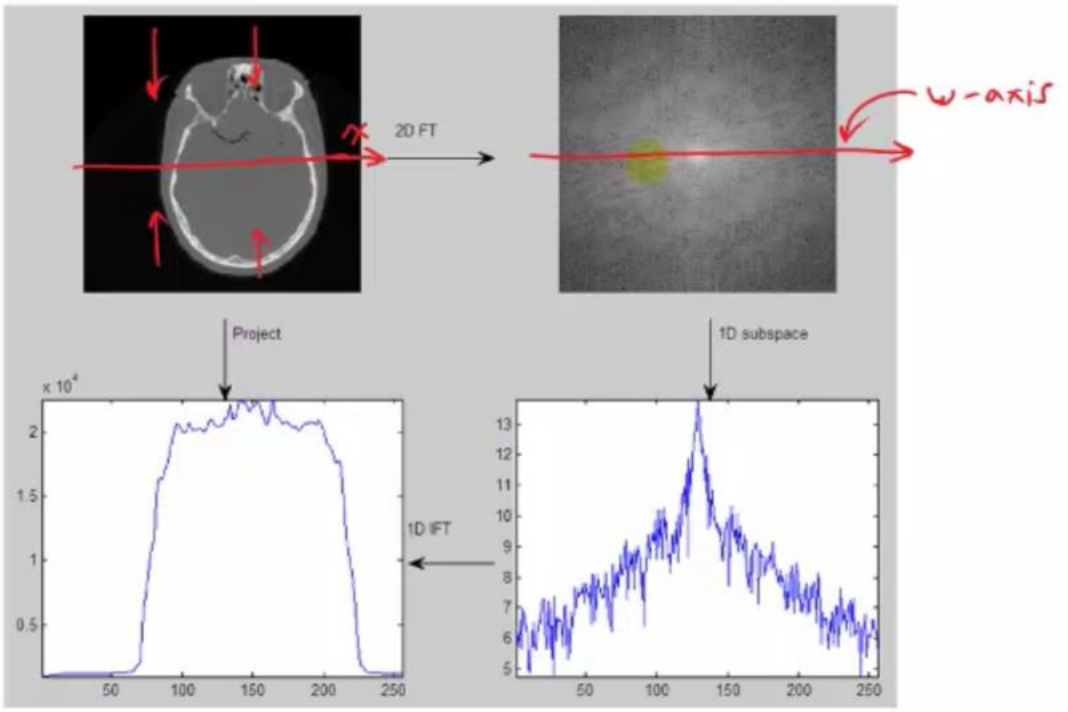
Recall the **Fourier Projection Theorem** (see the end of L09)

$$\mathcal{F}_{1D}\{p_\alpha(f(\vec{x}))(x)\}(\omega) = F(\omega, 0)$$

1D Fourier transform

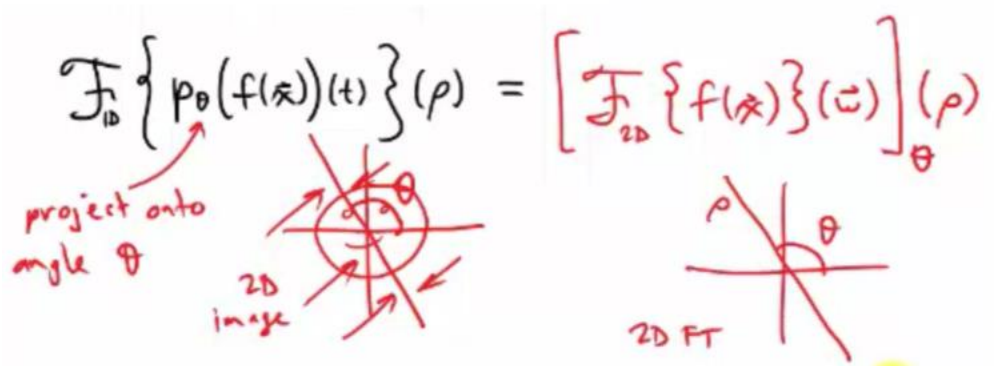
Radon proj. along y-axis is a function of x

1D subspace through the 2D Fourier transform of f.



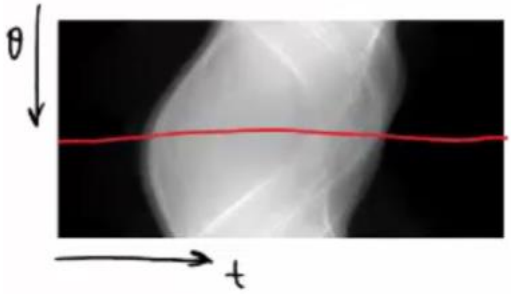
Theory of Back Projection

Since the Fourier transform is rotation invariant, we can apply the above in any direction.



Denote this as $P(\rho, \Theta)$. The scanner gives us $P_\theta(t)$ or $P(t, \Theta)$.

The spatial domain projection at angles **Θ**.



***Radon transform, take FT on row**

1D FT only

We just take the 1D-FT over **t** to get $P(\rho, \Theta)$.

Notice that this is $F(p, \Theta)$, simply a polar representation of $F(\vec{\omega})$.

Match the angle of the row

Each row corresponds to different subspace

Angle changes

Hence, we can reconstruct f by taking the inverse FT **of F**, which itself is constructed using $P(\rho, \Theta)$.

We want

$$f(x,y) = \int\int P(\rho(u,v), \theta(u,v))\, e^{2\pi i (ux+vy)}\, du\, dv$$

But this is in polar coords.

Change of variables... $u = \rho\cos\theta \quad v = \rho\sin\theta$

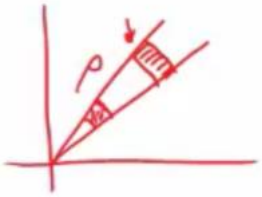
Instead of $\rho \geq 0$ and $-\pi \leq \theta < \pi$

we can integrate $-\infty < \rho < \infty$ and $0 \leq \theta < \pi$

$$f(x,y) = \int_{-\infty}^{\infty}\int_{0}^{\pi} P(\rho,\theta)\,|\rho|\, e^{2\pi i \rho(x\cos\theta + y\sin\theta)}\, d\theta\, d\rho$$

$$f(x,y) = \int_{0}^{\infty}\int_{-\pi}^{\pi} P(\rho,\theta)\, e^{2\pi i (x\rho\cos\theta + y\rho\sin\theta)}\, \rho\, d\theta\, d\rho$$

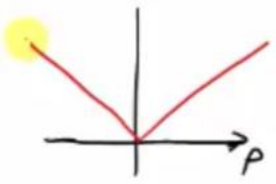
Area for $d\theta$ goes up linearly with ρ



ρ ← determinant of the Jacobian

This suggests that the way to reconstruct an image is to populate the frequency domain by adding each $\boldsymbol{P(\rho, \Theta)}$. (powerpoint demo)

Then multiply by the cone filter.



The cone filter compensates for the **higher density** of projections near the origin. It's essentially a **high pass** filter. (iradon_demo – looks better if 1 instead of 2, i.e. smaller steps)

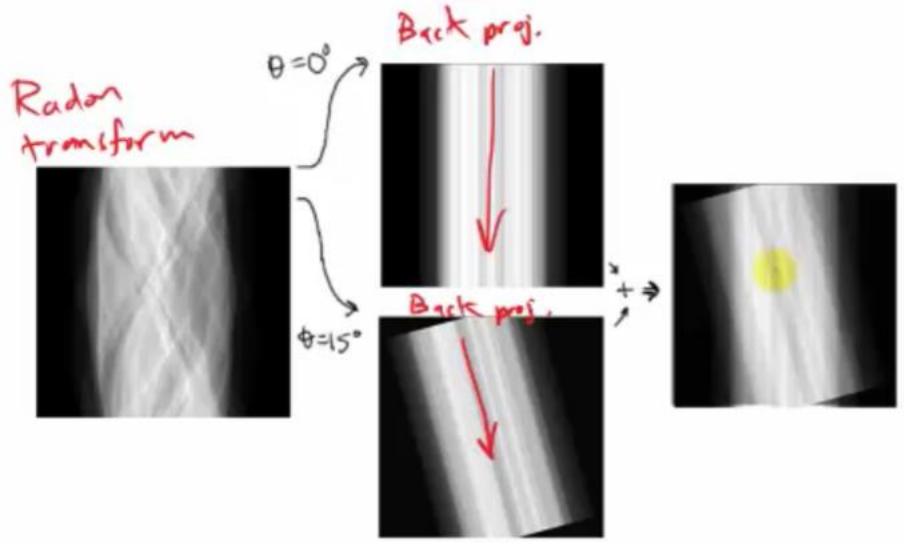
Filtered Back Projection

The method described above is called "filtered back projection". However, while this frequency-domain method works in theory, it requires **resampling** in the frequency domain, which can sometimes be **problematic**.

***resampling: assume things about underlying function – not smooth like in spatial**

Instead, we can do the equivalent operations in the spatial domain.

We take each projection and **"back project"** it. That is, we **smear** it back across the image in the **direction** it was acquired (i.e. the gantry angle).

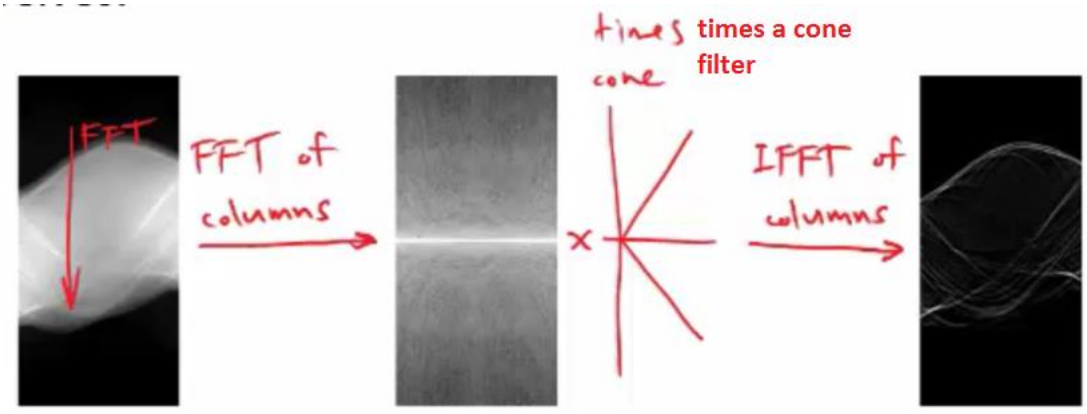


Then we simply add the back projections together – one for each projection in our Radon Transform.

The resulting image will be blurry. That's because we haven't applied the **filter** yet **(|p|)**. We can do that by multiplying the 2D FFT of our image by a cone.

frequency domain of filter

This filter can be applied in 1D to the projections themselves.



Then we do the back projection, and get a much crisper image.