Machine Learning for Signal Processing

Representing Signals: Images and Sounds

Class 4. 10 Sep 2013

Instructor: Bhiksha Raj

Administrivia

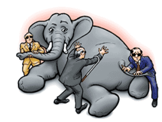
- Basics of probability: Will not be covered
  - Several very nice lectures on the net
- Things to know:
  - Basic probability, Bayes rule
  - Probability distributions over discrete variables
  - Probability density and Cumulative density over continuous variables
    - Particularly Gaussian densities
  - Moments of a distribution
  - What is independence
  - Nice to know
    - What is maximum likelihood estimation
    - MAP estimation

Representing Data

- The first and most important step in processing signals is representing them appropriately

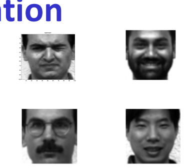
Representing an Elephant

- It was six men of Indostan,
  To learning much inclined,
  Who went to see the elephant,
  (Though all of them were blind),
  That each by observation
  Might satisfy his mind.
- The first approached the elephant,
  And happening to fall
  Against his broad and sturdy side,
  At once began to bawl:
  "God bless me! But the elephant
  Is very like a wall!"
- The second, feeling of the tusk,
  Cried: "Ho! What have we here,
  So very round and smooth and sharp?
  To me 'tis very clear,
  This wonder of an elephant
  Is very like a spear!"
- The third approached the animal,
  And happening to take
  The squirming trunk within his hands,
  Thus boldly up and spake:
  "I see," quoth he, "the elephant
  Is very like a snake!"
- The fourth reached out an eager hand,
  And felt about the knee.
  "What most this wondrous beast is like
  Is might plain," quoth he;
  "Tis clear enough the elephant
  Is very like a tree."
- The fifth, who chanced to touch the ear,
  Said: "E'en the blindest man
  Can tell what this resembles most:
  Deny the fact who can,
  This marvel of an elephant
  Is very like a fan."
- The sixth no sooner had begun
  About the beast to grope,
  Than seizing on the swinging tail
  That fell within his scope,
  "I see," quoth he, "the elephant
  Is very like a rope."
- And so these men of Indostan
  Disputed loud and long,
  Each in his own opinion
  Exceeding stiff and strong.
  Though each was partly right,
  All were in the wrong.

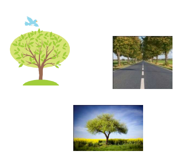


Representation

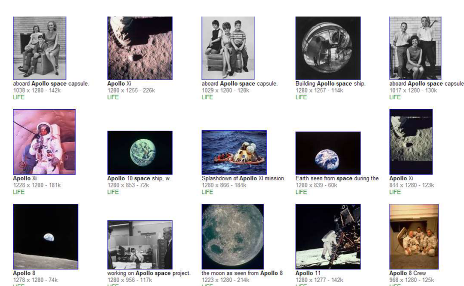
- Describe these images
  - Such that a listener can visualize what you are describing



- More images



Still more images

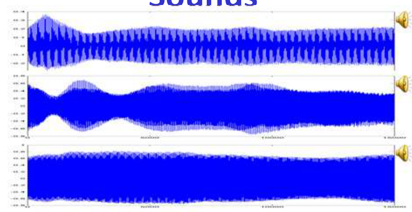


How do you describe them?

Representation

- Pixel-based descriptions are uninformative
- Feature-based descriptions are infeasible in the general case

Sounds



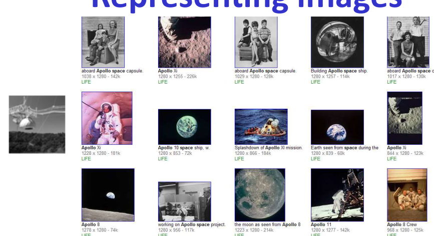
- Sounds are just sequences of numbers
- When plotted, they just look like blobs
  - Which leads to "natural sounds are blobs"
    - Or more precisely, "sounds are sequences of numbers that, when plotted, look like blobs"
  - Which wont get us anywhere

Representation

- Representation is description
- But in compact form
- Must describe the salient characteristics of the data
  - E.g. a pixel-wise description of the two images here will be completely different

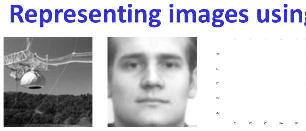
- Must allow identification, comparison, storage, reconstruction..

Representing images



- The most common element in the image: background
  - Or rather large regions of relatively featureless shading
  - Uniform sequences of numbers

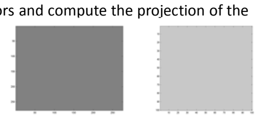
Representing images using a "plain" image



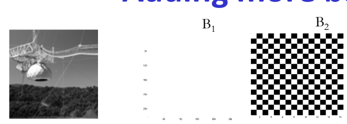
$$B = \begin{bmatrix} 1 \\ 1 \\ . \\ 1 \end{bmatrix} \quad \text{Image} = \begin{bmatrix} pixel\,1 \\ pixel\,2 \\ . \\ pixel\,N \end{bmatrix}$$

- Most of the figure is a more-or-less uniform shade
  - Dumb approximation – a image is a block of uniform shade
    - Will be mostly right!
- How to compute the "best" description? Projection
  - Represent the images as vectors and compute the projection of the image on the "basis"

$$BW \approx \text{Image}$$
$$W = pinv(B)\,\text{Image}$$
$$PROJECTION = BW = B(B^T B)^{-1} B^T.\text{Image}$$



Adding more bases



$$\begin{array}{cc} & \begin{array}{cc} B_1 & B_2 \end{array} \\ B = & \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ 1 & 1 \\ 1 & -1 \\ 1 & 1 \end{bmatrix} \end{array}$$

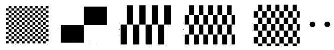
- Lets improve the approximation
- Images have some fast varying regions
  - Dramatic changes
  - Add a second picture that has very fast changes
    - A checkerboard where every other pixel is black and the rest are white

$$\text{Image} \approx w_1 B_1 + w_2 B_2$$

$$W = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \qquad B = [B_1 \;\; B_2]$$

$$BW \approx \text{Image}$$
$$W = pinv(B)\text{Image}$$
$$PROJECTION = BW = B(B^T B)^{-1} B^T.\text{Image}$$

Adding still more bases



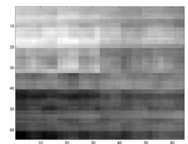
- Regions that change with different speeds

$Image \approx w_1 B_1 + w_2 B_2 + w_3 B_3 + \ldots$

$$W = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \cdot \\ \cdot \end{bmatrix} \qquad B = [B_1 \ B_2 \ B_3]$$

$$BW \approx Image$$

$$W = pinv(B)Image$$

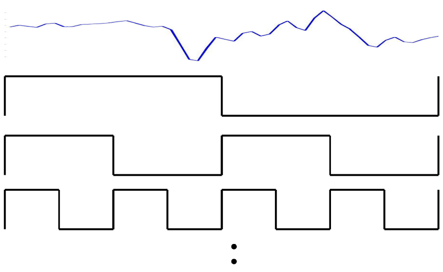


Getting closer at 625 bases!

Representation using checkerboards

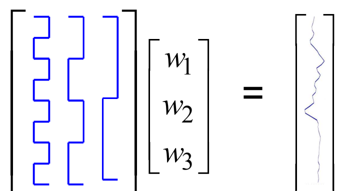
- A "standard" representation
  - Checker boards are the same regardless of the picture you're trying to describe
    - As opposed to using "nose shape" to describe faces and "leaf colour" to describe trees.
- Any image can be specified as (for example) 0.8*checkerboard(0) + 0.2*checkerboard(1) + 0.3*checkerboard(2) ..
- The definition is sufficient to reconstruct the image to some degree
  - Not perfectly though

What about sounds?



- Square wave equivalents of checker boards

Projecting sounds



$Signal \approx w_1 B_1 + w_2 B_2 + w_3 B_3$

$$W = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} \qquad B = [B_1 \ B_2 \ B_3]$$

$$BW \approx Signal$$

$$W = pinv(B)Signal$$

$$PROJECTION = BW = (B.pinv(B)).Signal$$

General Philosophy of Representation

- Identify a set of *standard structures*
  - E.g. checkerboards
  - We will call these "bases"
- Express the data as a weighted combination of these bases
  - $X = w_1 B_1 + w_2 B_2 + w_3 B_3 + \ldots$
- Chose weights w_1, w_2, w_3.. for the best representation of X
  - I.e. the error between X and $\Sigma_i w_i B_i$ is minimized
  - The error is generally chosen to be $||X - \Sigma_i w_i B_i||^2$
- The weights w_1, w_2, w_3.. fully specify the data
  - Since the bases are known beforehand
  - Knowing the weights is sufficient to reconstruct the data

Bases requirements

- Non-redundancy
  - Each basis must represent information *not* already represented by other bases
  - I.e. bases must be orthogonal
    - $<B_i, B_j> = 0$ for i != j
  - Mathematical benefit: can compute $w_i = <B_i, X>$
- Compactness
  - Must be able to represent most of X with fewest bases
  - Completeness: For D-dimensional data, need no more than D bases

Bases based representation

$$\begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix}$$

- Place all bases in basis matrix B

$$BW \approx X$$
$$W = Pinv(B)X$$

- For orthogonal bases

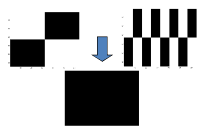
$$w_i = \frac{< B_i, X >}{\| B_i \|^2}$$

Bases based representation

- Challenge: Choice of appropriate bases

Why checkerboards are great bases

- We cannot explain one checkerboard in terms of another
  - The two are orthogonal to one another!
- This means we can determine the contributions of individual bases separately
  - Joint decomposition with multiple bases gives the same result as separate decomposition with each
  - This never holds true if one basis can explain another

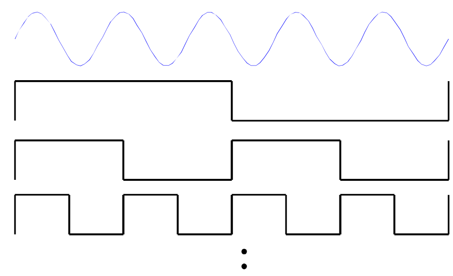


$$B = \begin{array}{cc} B_1 & B_2 \\ \end{array}$$
$$B = \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ 1 & 1 \\ 1 & -1 \\ 1 & 1 \end{bmatrix}$$

$$\mathrm{Image} \approx w_1 B_1 + w_2 B_2$$
$$W = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \qquad B = [B_1 \;\; B_2]$$

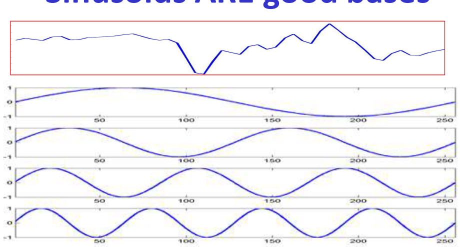
$$w_i = \frac{< B_i, \mathrm{Image} >}{\| B_i \|^2}$$

Checker boards are not good bases



- Sharp edges
  - Can *never* be used to explain rounded curves

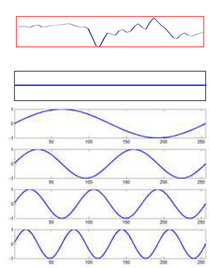
Sinusoids ARE good bases



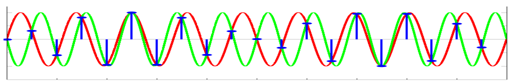
- They are orthogonal
- They can represent rounded shapes nicely
  - Unfortunately, they cannot represent sharp corners

What are the frequencies of the sinusoids

- Follow the same format as the checkerboard:
  - DC
  - The entire length of the signal is one period
  - The entire length of the signal is two periods.
- And so on..
- The k-th sinusoid:
  - F(n) = sin(2πkn/N)
    - N is the length of the signal
    - k is the number of periods in N samples

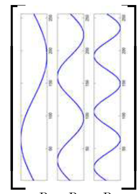


How many frequencies in all?

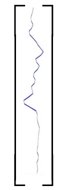


- A max of L/2 periods are possible
- If we try to go to (L/2 + X) periods, it ends up being identical to having (L/2 – X) periods
  - With sign inversion
- Example for L = 20
  - Red curve = sine with 9 cycles (in a 20 point sequence)
    - Y(n) = sin(2π9n/20)
  - Green curve = sine with 11 cycles in 20 points
    - Y(n) = -sin(2π11n/20)
  - The blue lines show the actual samples obtained
    - These are the only numbers stored on the computer
    - This set is the same for both sinusoids

How to compose the signal from sinusoids



$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} =$$



$$Signal \approx w_1B_1 + w_2B_2 + w_3B_3 \qquad W = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} \qquad B = [B_1\ B_2\ B_3]$$

$$BW \approx Signal$$

$$W = pinv(B)Signal$$

$$PROJECTION = BW = B(B^TB)^{-1}B.Signal$$

- The sines form the vectors of the projection matrix
  - Pinv() will do the trick as usual

How to compose the signal from sinusoids

$$\begin{bmatrix} \sin(2\pi.0.0/L) & \sin(2\pi.1.0/L) & . & . & \sin(2\pi.(L/2).0/L) \\ \sin(2\pi.0.1/L) & \sin(2\pi.1.1/L) & . & . & \sin(2\pi.(L/2).1/L) \\ . & . & . & . & . \\ . & . & . & . & . \\ \sin(2\pi.0.(L-1)/L) & \sin(2\pi.1.(L-1)/L) & . & . & \sin(2\pi.(L/2).(L-1)/L) \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ . \\ . \\ w_{L/2} \end{bmatrix} = \begin{bmatrix} s[0] \\ s[1] \\ . \\ . \\ s[L-1] \end{bmatrix}$$

L/2 columns only

$$Signal \approx w_1B_1 + w_2B_2 + w_3B_3$$

$$W = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} \qquad B = [B_1\ B_2\ B_3]$$

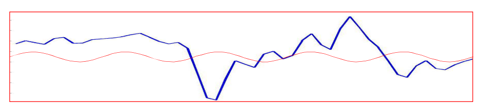
$$Signal = \begin{bmatrix} s[0] \\ s[1] \\ . \\ s[L-1] \end{bmatrix}$$

$$BW \approx Signal$$

$$W = pinv(B)Signal$$

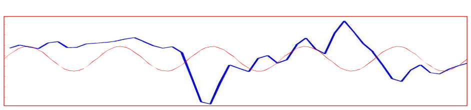
- The sines form the vectors of the projection matrix
  - Pinv() will do the trick as usual

Interpretation..



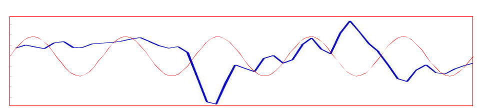
- Each sinusoid's amplitude is adjusted until it gives us the least squared error
  - The amplitude is the weight of the sinusoid
- This can be done independently for each sinusoid

Interpretation..



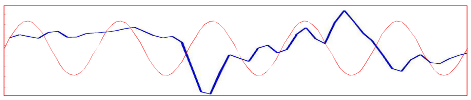
- Each sinusoid's amplitude is adjusted until it gives us the least squared error
  - The amplitude is the weight of the sinusoid
- This can be done independently for each sinusoid

Interpretation..



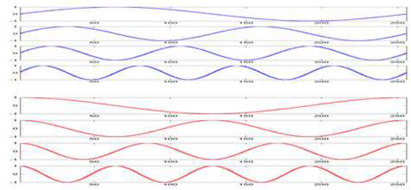
- Each sinusoid's amplitude is adjusted until it gives us the least squared error
  - The amplitude is the weight of the sinusoid
- This can be done independently for each sinusoid

Interpretation..



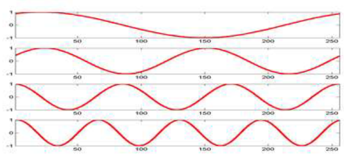
- Each sinusoid's amplitude is adjusted until it gives us the least squared error
  - The amplitude is the weight of the sinusoid
- This can be done independently for each sinusoid

Sines by themselves are not enough



- Every sine starts at zero
  - Can never represent a signal that is non-zero in the first sample!
- Every cosine starts at 1
  - If the first sample is zero, the signal cannot be represented!

The need for phase



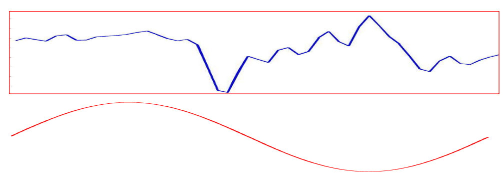
Sines are shifted: do not start with value = 0

- Allow the sinusoids to move!

$$signal = w_1 \sin(2\pi kn / N + \phi_1) + w_2 \sin(2\pi kn / N + \phi_2) + ...$$

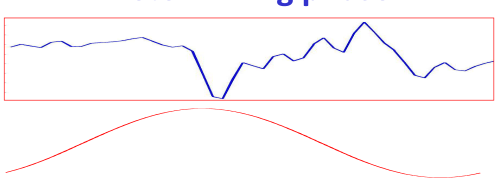
- How much do the sines shift?

Determining phase



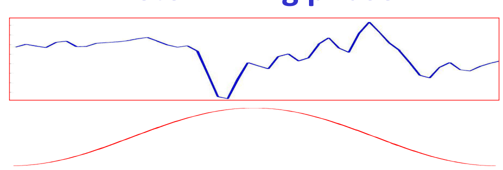
- Least squares fitting: move the sinusoid left / right, and at each shift, try all amplitudes
  - Find the combination of amplitude and phase that results in the lowest squared error
- We can still do this separately for each sinusoid
  - The sinusoids are still orthogonal to one another

Determining phase



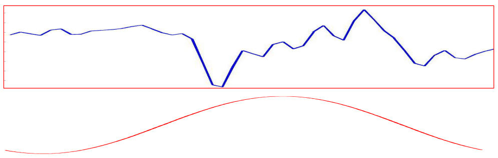
- Least squares fitting: move the sinusoid left / right, and at each shift, try all amplitudes
  - Find the combination of amplitude and phase that results in the lowest squared error
- We can still do this separately for each sinusoid
  - The sinusoids are still orthogonal to one another

Determining phase



- Least squares fitting: move the sinusoid left / right, and at each shift, try all amplitudes
  - Find the combination of amplitude and phase that results in the lowest squared error
- We can still do this separately for each sinusoid
  - The sinusoids are still orthogonal to one another

Determining phase



- Least squares fitting: move the sinusoid left / right, and at each shift, try all amplitudes
  - Find the combination of amplitude and phase that results in the lowest squared error
- We can still do this separately for each sinusoid
  - The sinusoids are still orthogonal to one another

The problem with phase

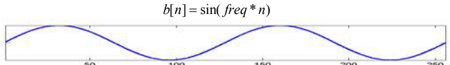
$$\begin{bmatrix} \sin(2\pi.0.0/L+\phi_0) & \sin(2\pi.1.0/L+\phi_1) & . & . & \sin(2\pi.(L/2).0/L+\phi_{L/2}) \\ \sin(2\pi.0.1/L+\phi_0) & \sin(2\pi.1.1/L+\phi_1) & . & . & \sin(2\pi.(L/2).1/L+\phi_{L/2}) \\ . & . & . & . & . \\ . & . & . & . & . \\ \sin(2\pi.0.(L-1)/L+\phi_0) & \sin(2\pi.1.(L-1)/L+\phi_1) & . & . & \sin(2\pi.(L/2).(L-1)/L+\phi_{L/2}) \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ . \\ . \\ w_{L/2} \end{bmatrix} = \begin{bmatrix} s[0] \\ s[1] \\ . \\ . \\ s[L-1] \end{bmatrix}$$

L/2 columns only

- This can no longer be expressed as a simple linear algebraic equation
  - The "basis matrix" depends on the unknown phase
    - I.e. there's a component of the basis itself that must be estimated!
- Linear algebraic notation can only be used if the bases are *fully* known
  - *We can only (pseudo) invert a known matrix*

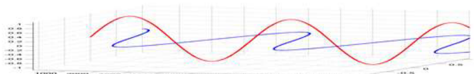
Complex Exponential to the rescue

$$b[n] = \sin(freq * n)$$



$$b[n] = \exp(j * freq * n) = \cos(freq * n) + j\sin(freq * n)$$

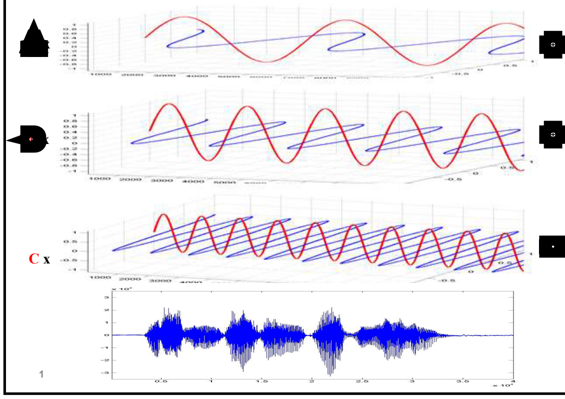
$$j = \sqrt{-1}$$



$$\exp(j * freq * n + \phi) = \exp(j * freq * n)\exp(\phi) = \cos(freq * n + \phi) + j\sin(freq * n + \phi)$$

- The cosine is the real part of a complex exponential
  - The sine is the imaginary part
- A phase term for the sinusoid becomes a multiplicative term for the complex exponential!!

Explaining with Complex Exponentials



Complex exponentials are well behaved

- Like sinusoids, a complex exponential of one frequency can never explain one of another
  - They are orthogonal
- They represent smooth transitions
- Bonus: They are *complex*
  - Can even model complex data!
- They can also model real data
  - exp(j x) + exp(-j x) is real
    - cos(x) + j sin(x) + cos(x) – j sin(x) = 2cos(x)
- More importantly
  - $\exp\left(j2\pi\frac{(L/2-x)n}{L}\right)+\exp\left(j2\pi\frac{(L/2+x)n}{L}\right)$ is real
    - The complex exponentials with frequencies equally spaced from L/2 are complex conjugates

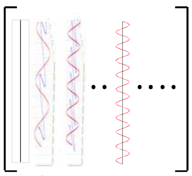
Complex exponentials are well behaved

- $\exp\left(j2\pi\frac{(L/2-x)n}{L}\right)+\exp\left(j2\pi\frac{(L/2+x)n}{L}\right)$
  - The complex exponentials with frequencies equally spaced from L/2 are complex conjugates
    - "Frequency = k" ➔ k periods in L samples

$$a\exp\left(j2\pi\frac{(L/2-x)n}{L}\right)+conjugate(a)\exp\left(j2\pi\frac{(L/2+x)n}{L}\right)$$

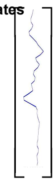
  - Is also real
  - If the two exponentials are multiplied by numbers that are conjugates of one another the result is real

Complex Exponential bases



Complex conjugates

$$\left[\ \cdots\ \right]\begin{bmatrix} w_0 \\ . \\ w_{L/2-1} \\ w_{L/2} \\ w_{L/2+1} \\ . \\ w_{L-1}\end{bmatrix} = \left[\ \ \right]$$



$w_{L/2+k} = conjugate(w_{L/2-k})$

- Explain the data using L complex exponential bases
- The weights given to the (L/2 + k)th basis and the (L/2 – k)th basis should be complex conjugates, to make the result real
  - Because we are dealing with real data
- Fortunately, a least squares fit will give us identical weights to both bases automatically; there is no need to impose the constraint externally

Complex Exponential Bases: Algebraic Formulation

$$\begin{bmatrix} \exp(j2\pi.0.0/L) & . & \exp(j2\pi.(L/2).0/L). & . & \exp(j2\pi.(L-1).0/L) \\ \exp(j2\pi.0.1/L) & . & \exp(j2\pi.(L/2).1/L). & . & \exp(j2\pi.(L-1).1/L) \\ . & . & . & . & . \\ . & . & . & . & . \\ \exp(j2\pi.0.(L-1)/L) & . & \exp(j2\pi.(L/2).(L-1)/L) & . & \exp(j2\pi.(L-1).(L-1)/L) \end{bmatrix}\begin{bmatrix} S_0 \\ . \\ S_{L/2} \\ . \\ S_{L-1}\end{bmatrix} = \begin{bmatrix} s[0] \\ s[1] \\ . \\ . \\ s[L-1]\end{bmatrix}$$

- Note that $S_{L/2+x} = \text{conjugate}(S_{L/2-x})$ for real s

Shorthand Notation

$$W_L^{k,n} = \frac{1}{\sqrt{L}}\exp(j2\pi kn/L) = \frac{1}{\sqrt{L}}\left(\cos(2\pi kn/L) + j\sin(2\pi kn/L)\right)$$

$$\begin{bmatrix} W_L^{0,0} & . & W_L^{L/2,0}. & . & W_L^{L-1,0} \\ W_L^{0,1} & . & W_L^{L/2,1}. & . & W_L^{L-1,1} \\ . & . & . & . & . \\ . & . & . & . & . \\ W_L^{0,L-1} & . & W_L^{L/2,L-1} & . & W_L^{L-1,L-1} \end{bmatrix}\begin{bmatrix} S_0 \\ . \\ S_{L/2} \\ . \\ S_{L-1}\end{bmatrix} = \begin{bmatrix} s[0] \\ s[1] \\ . \\ . \\ s[L-1]\end{bmatrix}$$

- Note that $S_{L/2+x} = \text{conjugate}(S_{L/2-x})$

A quick detour

- Real Orthonormal matrix:
  - $XX^T = X\,X^T = I$
    - But only if all entries are real
  - The inverse of X is its own transpose
- Definition: Hermitian
  - X^H = Complex conjugate of X^T
    - Conjugate of a number $a + ib = a - ib$
    - Conjugate of $\exp(ix) = \exp(-ix)$
- Complex Orthonormal matrix
  - $XX^H = X^H X = I$
  - The inverse of a complex orthonormal matrix is its own Hermitian

$W^{-1} = W^H$

$$W = \begin{bmatrix} W_L^{0,0} & . & W_L^{L/2,0}. & . & W_L^{L-1,0} \\ W_L^{0,1} & . & W_L^{L/2,1}. & . & W_L^{L-1,1} \\ . & . & . & . & . \\ . & . & . & . & . \\ W_L^{0,L-1} & . & W_L^{L/2,L-1} & . & W_L^{L-1,L-1} \end{bmatrix} \qquad W_L^{k,n} = \frac{1}{\sqrt{L}}\exp(j2\pi kn/L)$$

$$W_L^{-k,n} = \frac{1}{\sqrt{L}}\exp(-j2\pi kn/L) \qquad W^H = \begin{bmatrix} W_L^{0,0} & . & W_L^{-0,L/2}. & . & W_L^{-0,L-1} \\ W_L^{-1,0,} & . & W_L^{-1,L/2}. & . & W_L^{-1,L-1} \\ . & . & . & . & . \\ . & . & . & . & . \\ W_L^{-(L-1),0} & . & W_L^{-(L-1),L/2} & . & W_L^{-(L-1),(L-1)} \end{bmatrix}$$

- The complex exponential basis is orthogonal
  - Its inverse is its own Hermitian
  - $W^{-1} = W^H$

Doing it in matrix form

$$\begin{bmatrix} W_L^{0,0} & . & W_L^{L/2,0}. & . & W_L^{L-1,0} \\ W_L^{0,1} & . & W_L^{L/2,1}. & . & W_L^{L-1,1} \\ . & . & . & . & . \\ . & . & . & . & . \\ W_L^{0,L-1} & . & W_L^{L/2,L-1} & . & W_L^{L-1,L-1} \end{bmatrix}\begin{bmatrix} S_0 \\ . \\ S_{L/2} \\ . \\ S_{L-1}\end{bmatrix} = \begin{bmatrix} s[0] \\ s[1] \\ . \\ . \\ s[L-1]\end{bmatrix}$$

$$\begin{bmatrix} S_0 \\ . \\ S_{L/2} \\ . \\ S_{L-1}\end{bmatrix} = \begin{bmatrix} W_L^{0,0} & . & W_L^{-0,L/2}. & . & W_L^{-0,L-1} \\ W_L^{-1,0,} & . & W_L^{-1,L/2}. & . & W_L^{-1,L-1} \\ . & . & . & . & . \\ . & . & . & . & . \\ W_L^{-(L-1),0} & . & W_L^{-(L-1),L/2} & . & W_L^{-(L-1),(L-1)} \end{bmatrix}\begin{bmatrix} s[0] \\ s[1] \\ . \\ . \\ s[L-1]\end{bmatrix}$$

- Because $W^{-1} = W^H$

The Discrete Fourier Transform

$$\begin{bmatrix} S_0 \\ . \\ S_{L/2} \\ . \\ S_{L-1} \end{bmatrix} = \begin{bmatrix} W_L^{0,0} & . & W_L^{-0,L/2}. & . & W_L^{-0,L-1} \\ W_L^{-1,0,} & . & W_L^{-1,L/2}. & . & W_L^{-1,L-1} \\ . & . & . & . & . \\ . & . & . & . & . \\ W_L^{-(L-1),0} & . & W_L^{-(L-1),L/2} & . & W_L^{-(L-1),(L-1)} \end{bmatrix} \begin{bmatrix} s[0] \\ s[1] \\ . \\ . \\ s[L-1] \end{bmatrix}$$

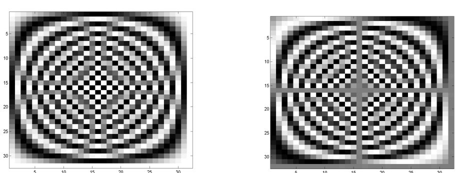
- The matrix to the right is called the "Fourier Matrix"
- The weights (S_0, S_1. . Etc.) are called the Fourier transform

The Inverse Discrete Fourier Transform

$$\begin{bmatrix} W_L^{0,0} & . & W_L^{L/2,0}. & . & W_L^{L-1,0} \\ W_L^{0,1} & . & W_L^{L/2,1}. & . & W_L^{L-1,1} \\ . & . & . & . & . \\ . & . & . & . & . \\ W_L^{0,L-1} & . & W_L^{L/2,L-1} & . & W_L^{L-1,L-1} \end{bmatrix} \begin{bmatrix} S_0 \\ . \\ S_{L/2} \\ . \\ S_{L-1} \end{bmatrix} = \begin{bmatrix} s[0] \\ s[1] \\ . \\ . \\ s[L-1] \end{bmatrix}$$

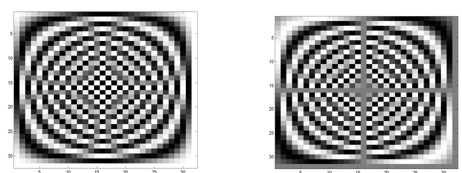
- The matrix to the left is the inverse Fourier matrix
- Multiplying the Fourier transform by this matrix gives us the signal right back from its Fourier transform

The Fourier Matrix



- Left panel: The real part of the Fourier matrix
  - For a 32-point signal
- Right panel: The imaginary part of the Fourier matrix

The FAST Fourier Transform

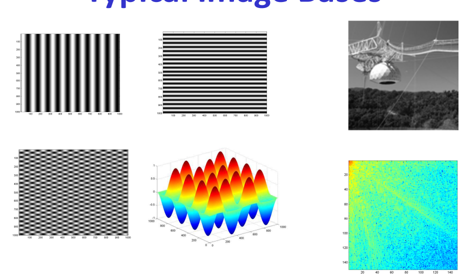


- The outcome of the transformation with the Fourier matrix is the **DISCRETE FOURIER TRANSFORM** (DFT)
- The **FAST Fourier transform** is an algorithm that takes advantage of the symmetry of the matrix to perform the matrix multiplication really fast
- The FFT computes the DFT
  - Is much faster if the length of the signal can be expressed as 2^N

Images

- The complex exponential is two dimensional
  - Has a separate X frequency and Y frequency
    - Would be true even for checker boards!
  - The 2-D complex exponential must be unravelled to form one component of the Fourier matrix
    - For a KxL image, we'd have K*L bases in the matrix

Typical Image Bases

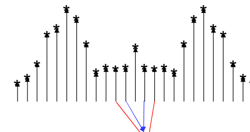


- Only real components of bases shown

DFT: Properties

- The DFT coefficients are complex
  - Have both a magnitude and a phase
  - $S_k = |S_k| \exp(-j \angle S_k)$
- Simple linear algebra tells us that
  - DFT(A + B) = DFT(A) + DFT(B)
  - The DFT of the sum of two signals is the DFT of their sum
- A horribly common approximation in sound processing
  - Magnitude(DFT(A+B)) = Magnitude(DFT(A)) + Magnitude(DFT(B))
  - Utterly wrong
  - Absurdly useful

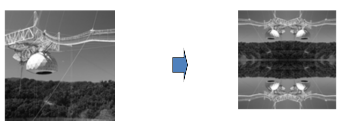
Symmetric signals



Contributions from points equidistant from L/2 combine to cancel out imaginary terms

- If a signal is (conjugate) symmetric around L/2, the Fourier coefficients are real!
  - A(L/2-k) * exp(-j *f*(L/2-k)) + A(L/2+k) * exp(-j*f*(L/2+k)) is always real if
    A(L/2-k) = conjugate(A(L/2+k))
  - We can pair up samples around the center all the way; the final summation term is always real
- Overall symmetry properties
  - If the *signal* is real, the FT is (conjugate) symmetric
  - If the signal is (conjugate) symmetric, the FT is real
  - If the signal is real and symmetric, the FT is real and symmetric

The Discrete Cosine Transform



- Compose a symmetric signal or image
  - Images would be symmetric in two dimensions
- Compute the Fourier transform
  - Since the FT is symmetric, sufficient to store only half the coefficients (quarter for an image)
    - Or as many coefficients as were originally in the signal / image

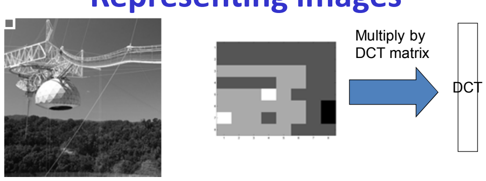
DCT

$$\begin{bmatrix} \cos(2\pi(0.5).0/2L) & \cos(2\pi.(1+0.5).0/2L) & . & . & \cos(2\pi.(L-0.5).0/2L) \\ \cos(2\pi.(0.5).1/2L) & \cos(2\pi.(1+0.5).1/2L) & . & . & \cos(2\pi.(L-0.5).1/2L) \\ . & . & . & . & . \\ . & . & . & . & . \\ \cos(2\pi.(0.5).(L-1)/2L) & \cos(2\pi.(1+0.5).(L-1)/2L) & . & . & \cos(2\pi.(L-0.5).(L-1)/2L) \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ . \\ . \\ w_{L-1} \end{bmatrix} = \begin{bmatrix} s[0] \\ s[1] \\ . \\ . \\ s[L-1] \end{bmatrix}$$

L columns

- Not necessary to compute a 2xL sized FFT
  - Enough to compute an L-sized *cosine* transform
  - Taking advantage of the symmetry of the problem
- This is the Discrete Cosine Transform

Representing images



- Most common coding is the DCT
- JPEG: Each 8x8 element of the picture is converted using a DCT
- The DCT coefficients are quantized and stored
  - Degree of quantization = degree of compression
- Also used to represent textures etc for pattern recognition and other forms of analysis

Some tricks to computing Fourier transforms

- Direct computation of the Fourier transform can result in poor representations
- Boundary effects can cause error
  - Solution : Windowing
- The size of the signal can introduce inefficiency
  - Solution: Zero padding

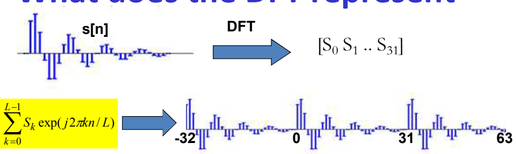
What does the DFT represent

$$\begin{bmatrix} \exp(j2\pi.0.0/L) & . & \exp(j2\pi.(L/2).0/L). & . & \exp(j2\pi.(L-1).0/L) \\ \exp(j2\pi.0.1/L) & . & \exp(j2\pi.(L/2).1/L). & . & \exp(j2\pi.(L-1).1/L) \\ . & . & . & . & . \\ . & . & . & . & . \\ \exp(j2\pi.0.(L-1)/L) & . & \exp(j2\pi.(L/2).(L-1)/L) & . & \exp(j2\pi.(L-1).(L-1)/L) \end{bmatrix} \begin{bmatrix} S_0 \\ . \\ S_{L/2} \\ . \\ S_{L-1} \end{bmatrix} = \begin{bmatrix} s[0] \\ s[1] \\ . \\ . \\ s[L-1] \end{bmatrix}$$

$$s[n] = \sum_{k=0}^{L-1} S_k \exp(j2\pi kn/L)$$

- The IDFT can be written formulaically as above
- There is no restriction on computing the formula for n < 0 or n > L-1
  - Its just a formula
  - But computing these terms behind 0 or beyond L-1 tells us what the signal composed by the DFT looks like outside our narrow window

What does the DFT represent



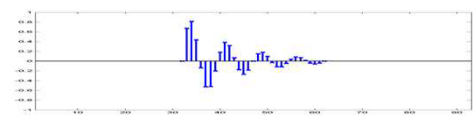
$$s[n] = \sum_{k=0}^{L-1} S_k \exp(j2\pi kn/L)$$

- If you extend the DFT-based representation beyond 0 (on the left) or L (on the right) it repeats the signal!
- So what does the DFT really mean

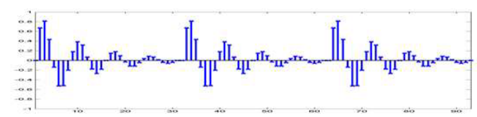
What does the DFT represent

- The DFT represents the properties of the infinitely long repeating signal that you can generate with it
  - Of which the observed signal is ONE period
- This gives rise to some odd effects

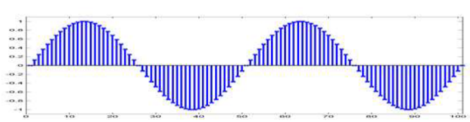
The discrete Fourier transform



- The discrete Fourier transform of the above signal actually computes the properties of the periodic signal shown below
  - Which extends from –infinity to +infinity
  - The period of this signal is 32 samples in this example

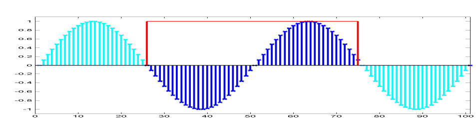


Windowing



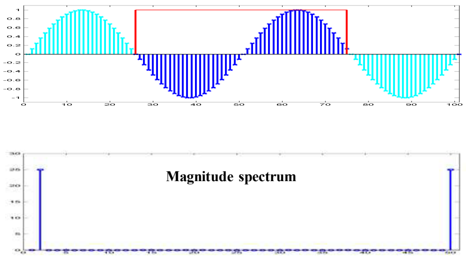
- The DFT of one period of the sinusoid shown in the figure computes the spectrum of the entire sinusoid from –infinity to +infinity

Windowing



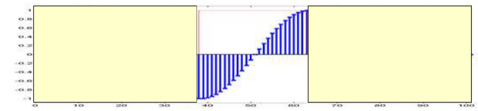
- The DFT of one period of the sinusoid shown in the figure computes the spectrum of the entire sinusoid from –infinity to +infinity

Windowing



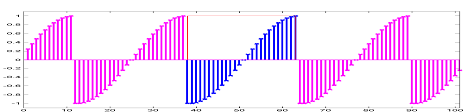
- The DFT of one period of the sinusoid shown in the figure computes the spectrum of the entire sinusoid from –infinity to +infinity
- The DFT of a real sinusoid has only one non zero frequency
  - The second peak in the figure is the "reflection" around L/2 (for real signals)

Windowing



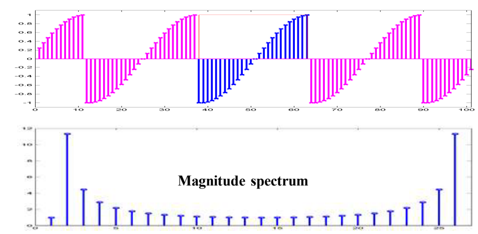
- The DFT of *any* sequence computes the spectrum for an infinite repetition of that sequence

Windowing



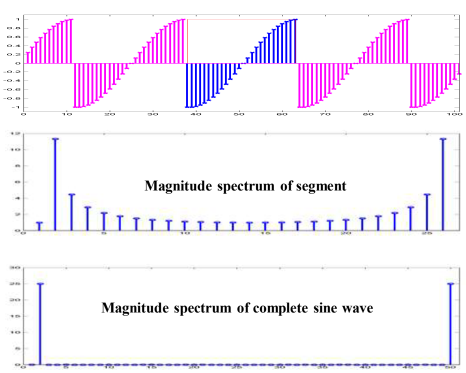
- The DFT of *any* sequence computes the spectrum for an infinite repetition of that sequence
- The DFT of a partial segment of a sinusoid computes the spectrum of an infinite repetition of that segment, and not of the entire sinusoid

Windowing

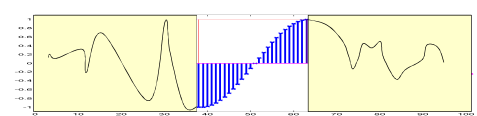


- The DFT of *any* sequence computes the spectrum for an infinite repetition of that sequence
- The DFT of a partial segment of a sinusoid computes the spectrum of an infinite repetition of that segment, and not of the entire sinusoid
- This will not give us the DFT of the sinusoid itself!

Windowing

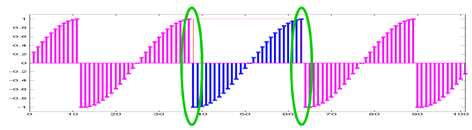


Windowing



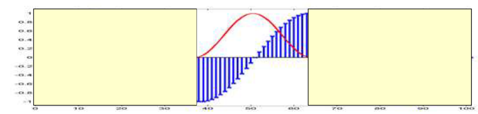
- The difference occurs due to two reasons:
- The transform cannot know what the signal actually looks like outside the observed window

Windowing



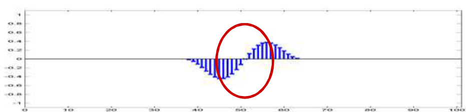
- The difference occurs due to two reasons:
- The transform cannot know what the signal actually looks like outside the observed window
- The implicit repetition of the observed signal introduces large discontinuities at the points of repetition
  - These are not part of the underlying signal
    - We only want to characterize the underlying signal
      - The discontinuity is an irrelevant detail

Windowing



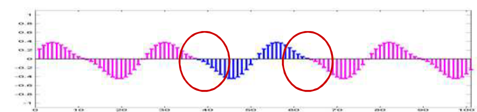
- While we can never know what the signal looks like outside the window, we can try to minimize the discontinuities at the boundaries
- We do this by multiplying the signal with a *window* function
  - We call this procedure windowing
  - We refer to the resulting signal as a "windowed" signal

Windowing



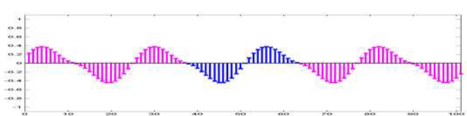
- While we can never know what the signal looks like outside the window, we can try to minimize the discontinuities at the boundaries
- We do this by multiplying the signal with a *window* function
  - We call this procedure windowing
  - We refer to the resulting signal as a "windowed" signal
- Windowing attempts to do the following:
  - Keep the windowed signal similar to the original in the central regions

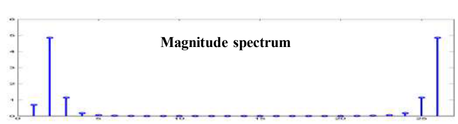
Windowing



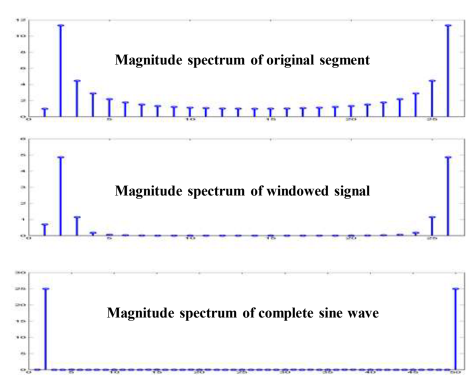
- While we can never know what the signal looks like outside the window, we can try to minimize the discontinuities at the boundaries
- We do this by multiplying the signal with a *window* function
  - We call this procedure windowing
  - We refer to the resulting signal as a "windowed" signal
- Windowing attempts to do the following:
  - Keep the windowed signal similar to the original in the central regions
  - Reduce or eliminate the discontinuities in the implicit periodic signal

Windowing

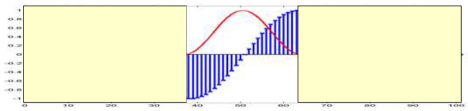




Windowing

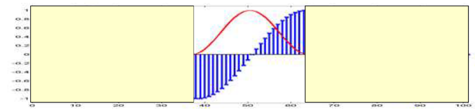


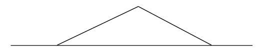
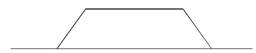
Window functions



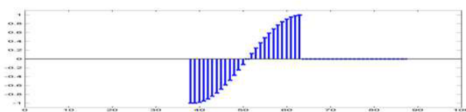
- Cosine windows:
  - Window length is M
  - Index begins at 0
- Hamming: $w[n] = 0.54 – 0.46 \cos(2\pi n/M)$
- Hanning: $w[n] = 0.5 – 0.5 \cos(2\pi n/M)$
- Blackman: $0.42 – 0.5 \cos(2\pi n/M) + 0.08 \cos(4\pi n/M)$

Window functions



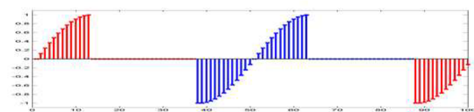
- Geometric windows:
  - Rectangular (boxcar): 
  - Triangular (Bartlett): 
  - Trapezoid: 

Zero Padding



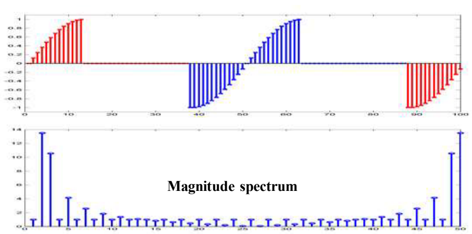
- We can pad zeros to the end of a signal to make it a desired length
  - Useful if the FFT (or any other algorithm we use) requires signals of a specified length
  - E.g. Radix 2 FFTs require signals of length 2^n i.e., some power of 2. We must zero pad the signal to increase its length to the appropriate number
- The consequence of zero padding is to change the periodic signal

Zero Padding



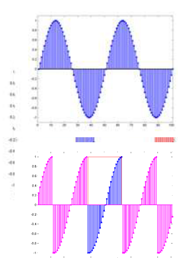
- We can pad zeros to the end of a signal to make it a desired length
  - Useful if the FFT (or any other algorithm we use) requires signals of a specified length
  - E.g. Radix 2 FFTs require signals of length 2^n i.e., some power of 2. We must zero pad the signal to increase its length to the appropriate number
- The consequence of zero padding is to change the periodic signal whose Fourier spectrum is being computed by the DFT

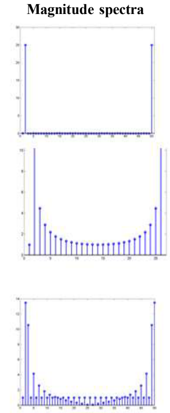
Zero Padding



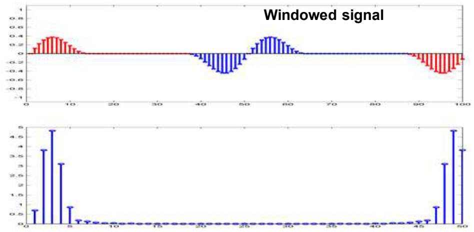
- The DFT of the zero padded signal is essentially the same as the DFT of the unpadded signal, with additional spectral samples inserted in between
  - It does not contain any additional information over the original DFT
  - It also does not contain less information

Magnitude spectra

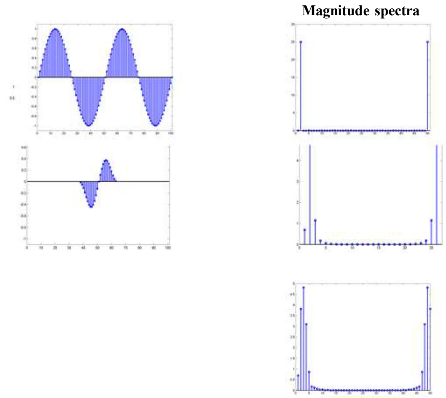




Zero Padding

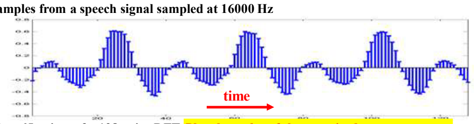


- The DFT of the zero padded signal is essentially the same as the DFT of the unpadded signal, with additional spectral samples inserted in between
  - It does not contain any additional information over the original DFT
  - It also does not contain less information

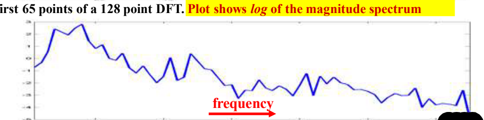


Zero padding a speech signal

128 samples from a speech signal sampled at 16000 Hz

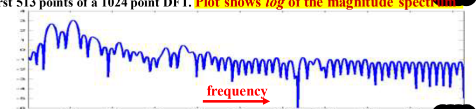


The first 65 points of a 128 point DFT. Plot shows *log* of the magnitude spectrum



[redacted]z

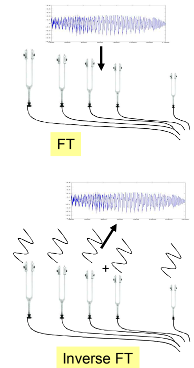
The first 513 points of a 1024 point DFT. Plot shows *log* of the magnitude spect[redacted]



[redacted]z

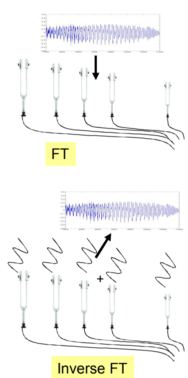
The Fourier Transform and Perception: Sound

- The Fourier transforms represents the signal analogously to a bank of tuning forks
- Our ear *has* a bank of tuning forks
- The output of the Fourier transform is perceptually very meaningful

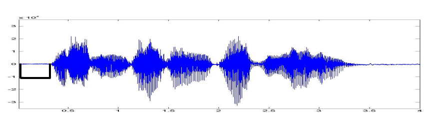


The Fourier Transform and Perception: Sound

- Processing Sound:
- Analyze the sound using a bank of tuning forks
- *Sample* the transduced output of the turning forks at periodic intervals

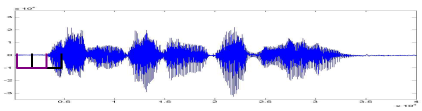


Sound parameterization



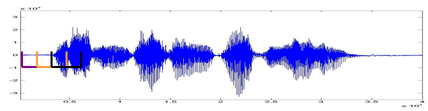
- The signal is processed in segments of 25-64 ms
  - Because the properties of audio signals change quickly
  - They are "stationary" only very briefly

Sound parameterization



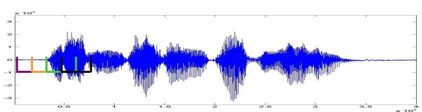
- The signal is processed in segments of 25-64 ms
  - Because the properties of audio signals change quickly
  - They are “stationary” only very briefly
- Adjacent segments overlap by 15-48 ms

Sound parameterization



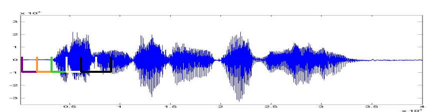
- The signal is processed in segments of 25-64 ms
  - Because the properties of audio signals change quickly
  - They are “stationary” only very briefly
- Adjacent segments overlap by 15-48 ms

Sound parameterization



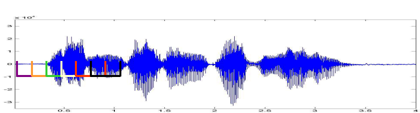
- The signal is processed in segments of 25-64 ms
  - Because the properties of audio signals change quickly
  - They are “stationary” only very briefly
- Adjacent segments overlap by 15-48 ms

Sound parameterization



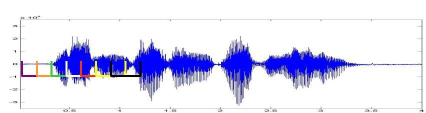
- The signal is processed in segments of 25-64 ms
  - Because the properties of audio signals change quickly
  - They are “stationary” only very briefly
- Adjacent segments overlap by 15-48 ms

Sound parameterization

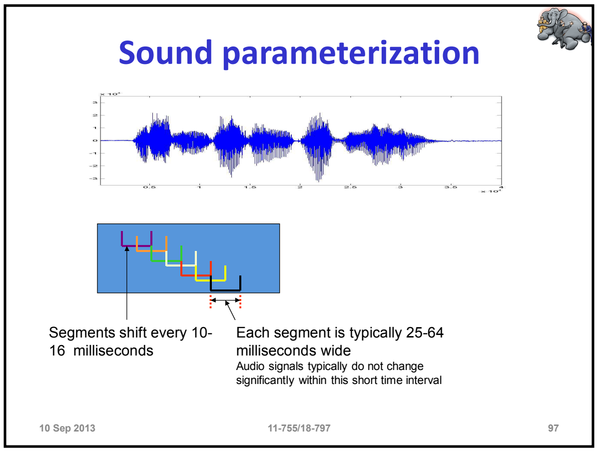


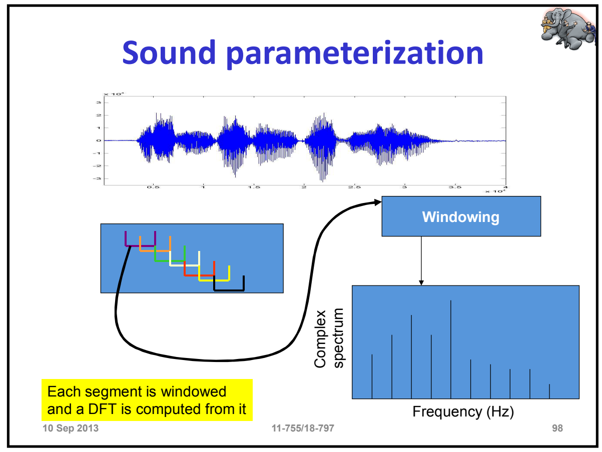
- The signal is processed in segments of 25-64 ms
  - Because the properties of audio signals change quickly
  - They are “stationary” only very briefly
- Adjacent segments overlap by 15-48 ms

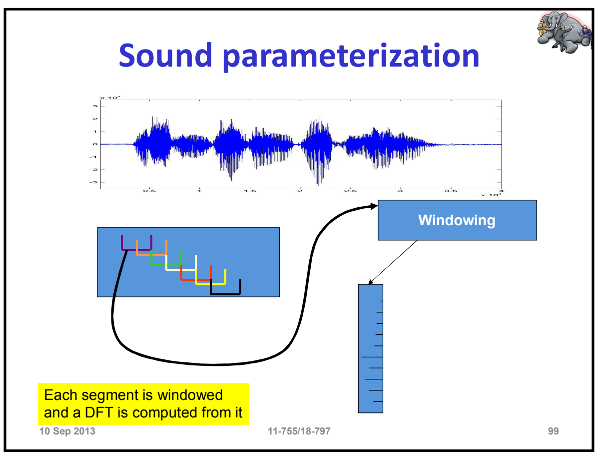
Sound parameterization

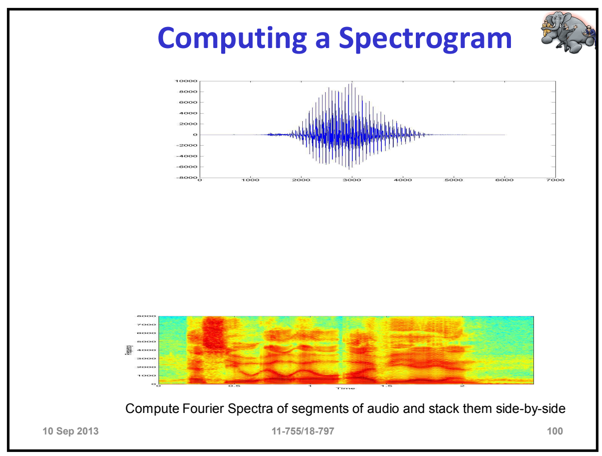


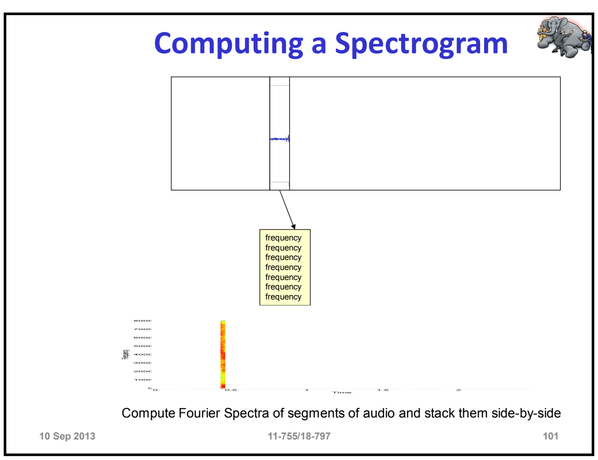
- The signal is processed in segments of 25-64 ms
  - Because the properties of audio signals change quickly
  - They are “stationary” only very briefly
- Adjacent segments overlap by 15-48 ms

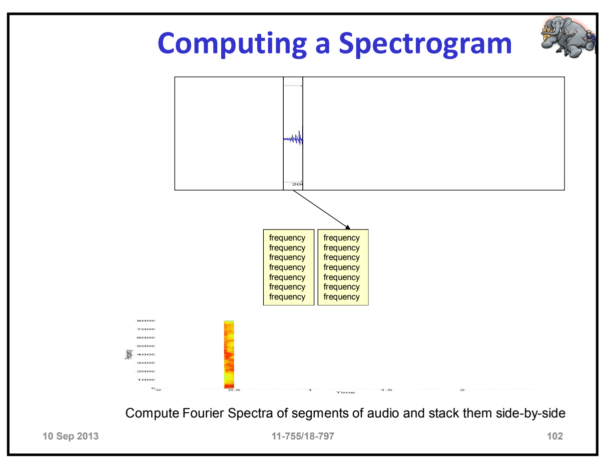

Sound parameterization
Segments shift every 10-16 milliseconds
Each segment is typically 25-64 milliseconds wide
Audio signals typically do not change significantly within this short time interval
10 Sep 2013
11-755/18-797
97


Sound parameterization
Windowing
Complex spectrum
Frequency (Hz)
Each segment is windowed and a DFT is computed from it
10 Sep 2013
11-755/18-797
98

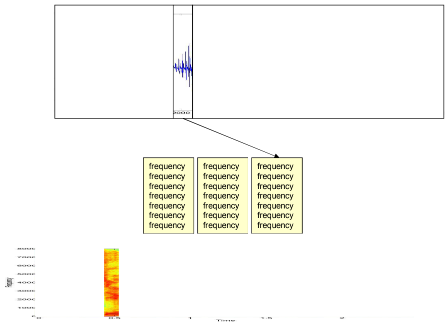

Sound parameterization
Windowing
Each segment is windowed and a DFT is computed from it
10 Sep 2013
11-755/18-797
99


Computing a Spectrogram
Compute Fourier Spectra of segments of audio and stack them side-by-side
10 Sep 2013
11-755/18-797
100


Computing a Spectrogram
frequency
frequency
frequency
frequency
frequency
frequency
frequency
Compute Fourier Spectra of segments of audio and stack them side-by-side
10 Sep 2013
11-755/18-797
101

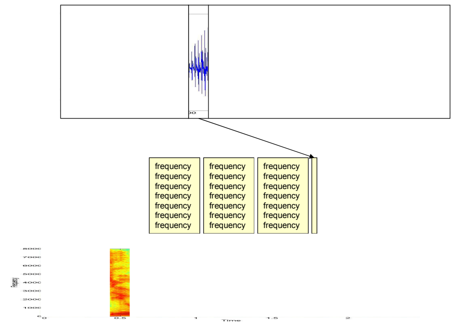

Computing a Spectrogram
frequency
frequency
frequency
frequency
frequency
frequency
frequency
frequency
frequency
frequency
frequency
frequency
frequency
frequency
Compute Fourier Spectra of segments of audio and stack them side-by-side
10 Sep 2013
11-755/18-797
102

Computing a Spectrogram



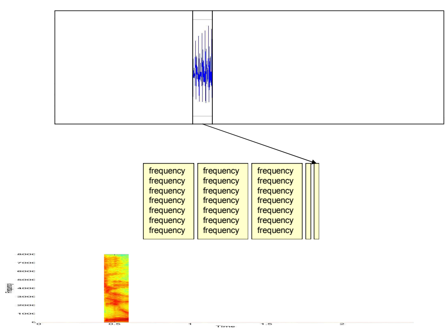
Compute Fourier Spectra of segments of audio and stack them side-by-side

Computing a Spectrogram



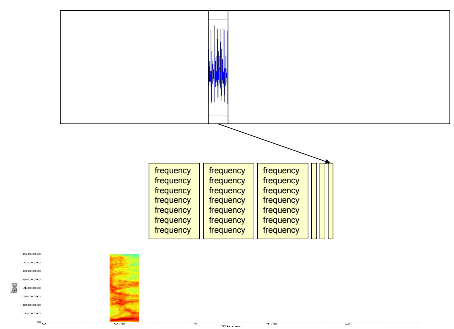
Compute Fourier Spectra of segments of audio and stack them side-by-side

Computing a Spectrogram



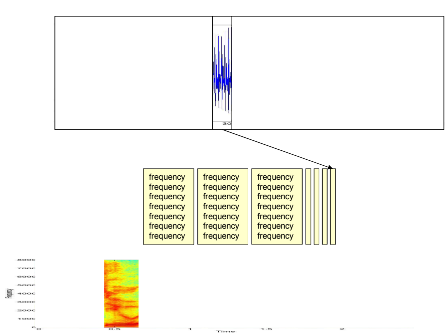
Compute Fourier Spectra of segments of audio and stack them side-by-side

Computing a Spectrogram



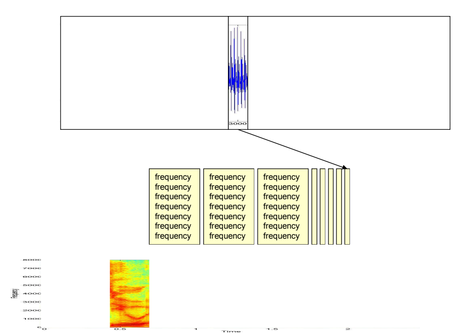
Compute Fourier Spectra of segments of audio and stack them side-by-side

Computing a Spectrogram



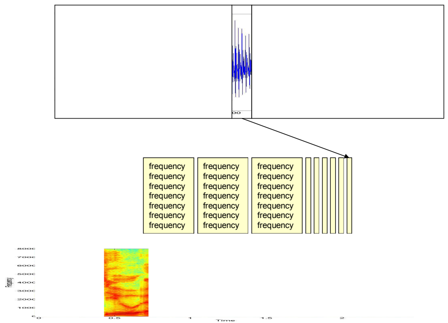
Compute Fourier Spectra of segments of audio and stack them side-by-side

Computing a Spectrogram



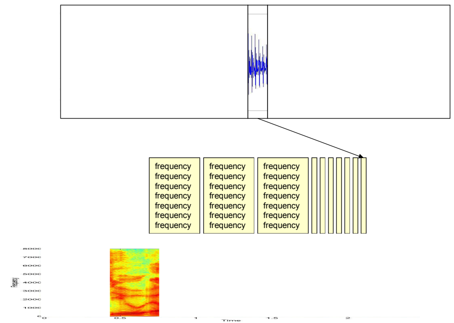
Compute Fourier Spectra of segments of audio and stack them side-by-side

Computing a Spectrogram



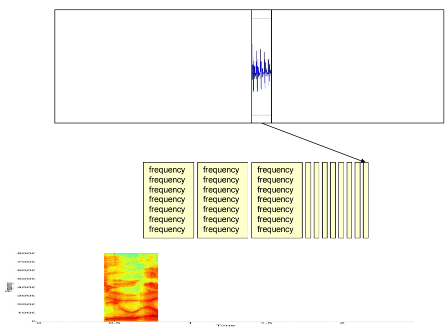
Compute Fourier Spectra of segments of audio and stack them side-by-side

Computing a Spectrogram



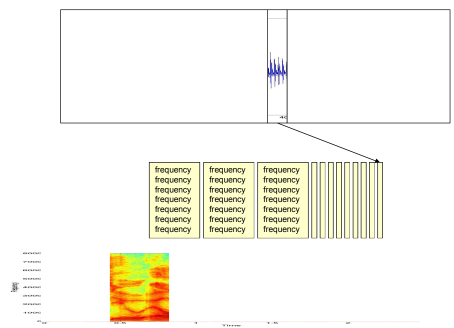
Compute Fourier Spectra of segments of audio and stack them side-by-side

Computing a Spectrogram



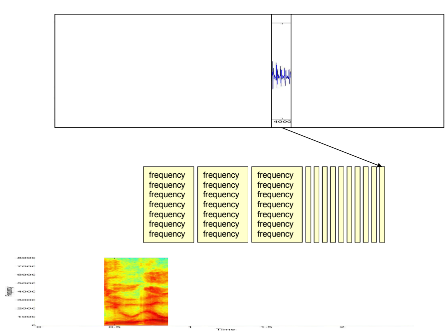
Compute Fourier Spectra of segments of audio and stack them side-by-side

Computing a Spectrogram



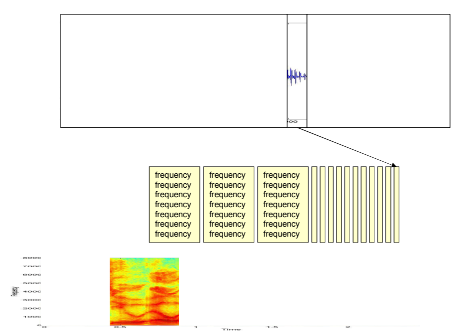
Compute Fourier Spectra of segments of audio and stack them side-by-side

Computing a Spectrogram



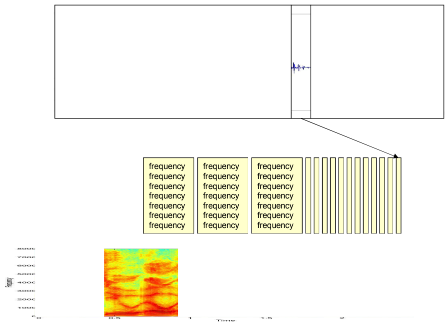
Compute Fourier Spectra of segments of audio and stack them side-by-side

Computing a Spectrogram



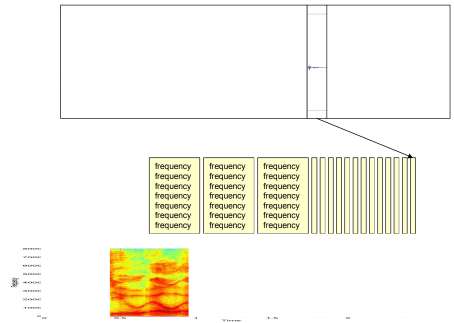
Compute Fourier Spectra of segments of audio and stack them side-by-side

Computing a Spectrogram



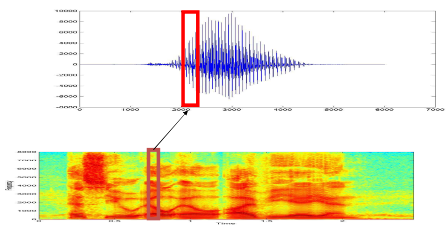
Compute Fourier Spectra of segments of audio and stack them side-by-side

Computing a Spectrogram



Compute Fourier Spectra of segments of audio and stack them side-by-side

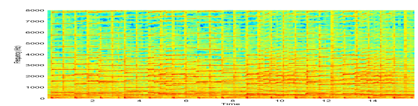
Computing the Spectrogram



Compute Fourier Spectra of segments of audio and stack them side-by-side
The Fourier spectrum of each window can be inverted to get back the signal.
Hence the spectrogram can be inverted to obtain a time-domain signal

In this example each segment was 25 ms long and adjacent segments overlapped by 15 ms

The result of parameterization



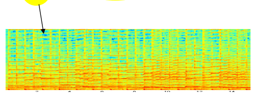
- Each column here represents the FT of a single segment of signal 64ms wide.
  - Adjacent segments overlap by 48 ms.
- DFT details
  - 1024 points (16000 samples a second).
  - 2048 point DFT – 1024 points of zero padding.
  - Only 1025 points of each DFT are shown
    - The rest are “reflections”
- The value shown is actually the magnitude of the complex spectral values
  - Most of our analysis / operations are performed on the magnitude

Magnitude and phase

?

$$S_k = |S_k| \exp(j.phase(S_k))$$

$$\begin{bmatrix} W_L^{0,0} & . & W_L^{L/2,0} & . & W_L^{L-1,0} \\ W_L^{0,1} & . & W_L^{L/2,1} & . & W_L^{L-1,1} \\ . & . & . & . & . \\ . & . & . & . & . \\ W_L^{0,L-1} & . & W_L^{L/2,L-1} & . & W_L^{L-1,L-1} \end{bmatrix} \begin{bmatrix} S_0 \\ . \\ S_k \\ . \\ S_{L-1} \end{bmatrix} = \begin{bmatrix} s[0] \\ s[1] \\ . \\ . \\ s[L-1] \end{bmatrix}$$

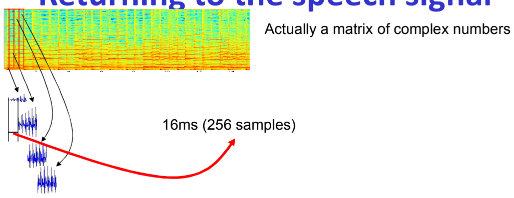


- All the operations (e.g. the examples shown in the previous class) are performed on the magnitude
- The phase of the complex spectrum is needed to invert a DFT to a signal
  - Where does that come from?
- Deriving phase is a serious, not-quite solved problem.

Phase

- Common tricks: Obtain the phase from the original signal
  - Sft = DFT(signal)
  - Phase1 = phase(Sft)
    - Each term is of the form real + j imag
    - For each element, compute arctan(imag/real)
  - Smagnitude = magnitude(Sft)
    - For each element compute Sqrt(real*real + imag*imag)
  - ProcessedSpectrum = Process(Smagnitude)
  - New SFT = ProcessedSpectrum*exp(j*Phase)
  - Recover signal from SFT
- Some other tricks:
  - Compute the FT of a different signal of the same length
  - Use the phase from that signal

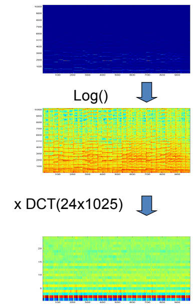
Returning to the speech signal



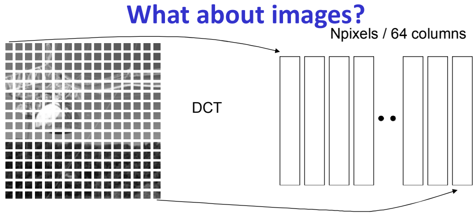
- For each complex spectral vector, compute a signal from the inverse DFT
  - Make sure to have the complete FT (including the reflected portion)
- If need be window the retrieved signal
- Overlap signals from adjacent vectors in exactly the same manner as during analysis
  - E.g. If a 48ms (768 sample) overlap was used during analysis, overlap adjacent segments by 768 samples

Additional tricks

- The basic representation is the magnitude spectrogram
- Often it is transformed to a *log* spectrum
  - By computing the log of each entry in the spectrogram matrix
  - After processing, the entry is exponentiated to get back the magnitude spectrum
    - To which phase may be factored in to get a signal
- The log spectrum may be "compressed" by a dimensionality reducing matrix
  - Usually a DCT matrix

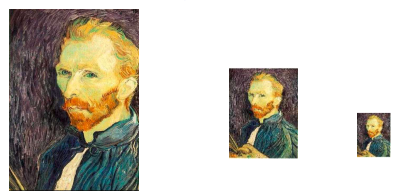


What about images?



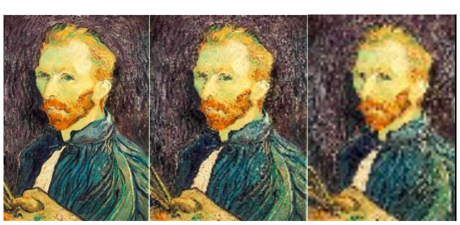
- DCT of small segments
  - 8x8
  - Each image becomes a matrix of DCT vectors
- DCT of the image
- **Pyramid representations**
- Haar transform (checkerboard)
- Various wavelet representations
  - Gabor wavelets
- *Or data-driven representations*
  - Eigen faces, SIFT

Downsampling-based representations



- Downsampling an example
  - Trying to reduce size by factor of 4 each time
    - Select every alternate sample row-wise and column-wisee
  - What exactly did we capture?
    - Clue : Results are horrible.

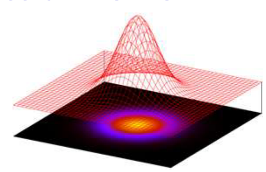
Downsampling-based representations



- Nasty aliasing effects!

The Gaussian Kernel

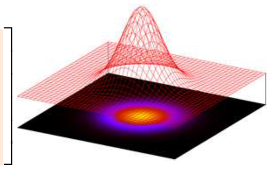
$$\begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_N \end{bmatrix}$$



- A two-dimensional image of a Gaussian
- Characterized by
  - Center (mean)
  - Standard deviation σ (assumed same in both directions)
    - I.e. sphereical Gaussian
- The image can be represented by a vector

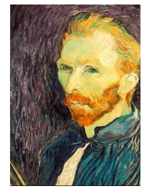
The Gaussian Kernel matrix

$$G = \begin{bmatrix} g_{11} & g_{12} & \cdots & g_{1N} \\ g_{21} & g_{22} & \cdots & g_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ g_{N1} & g_{N2} & \cdots & g_{NN} \end{bmatrix}$$



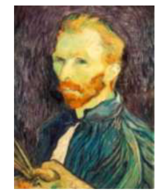
- Each column is one Gaussian
  - Representing a Gaussian centered at one of the pixels in the image
- As many columns as pixels
  - Also as many rows as pixels

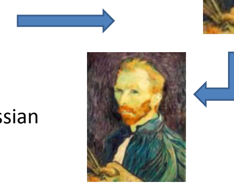
Downsampling-based representations



G X

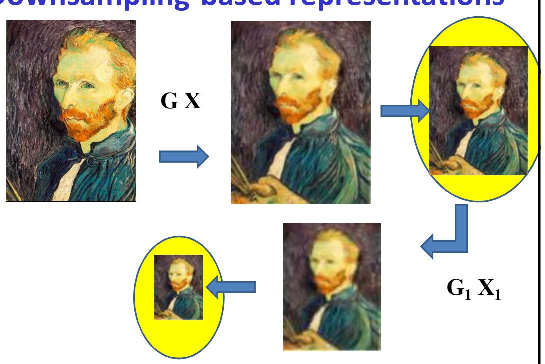
$$X = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_N \end{bmatrix}$$



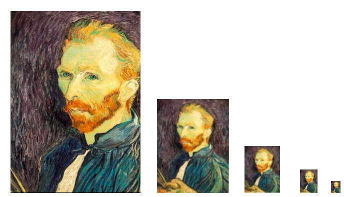


- Transform with Gaussian kernel matrix
- Then downsample

Downsampling-based representations

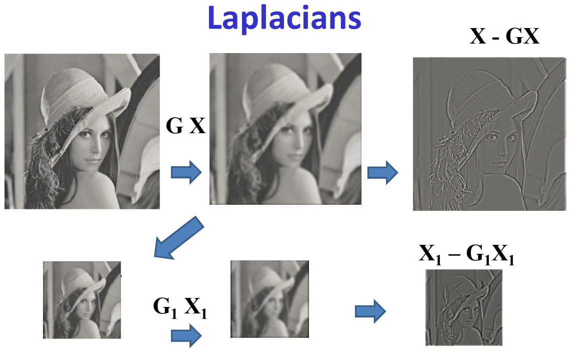


The Gaussian Pyramid

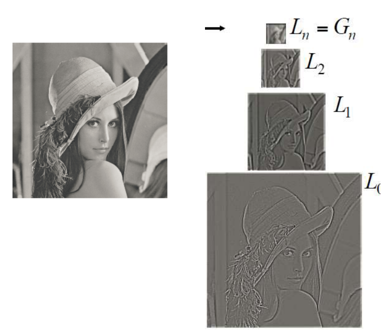


- Successive smoothing and scaling
- The entire collection of images is the Gaussian pyramid

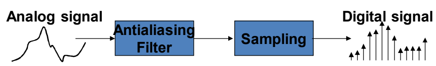
Laplacians



Laplacian Pyramid

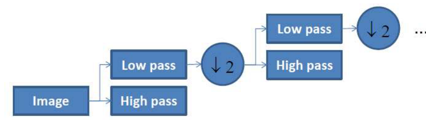


Remember..



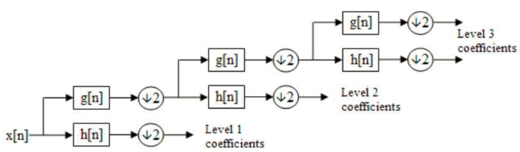
- The Gaussian is an anti-aliasing filter
- The Gaussian pyramid is the *low-pass filtered* version of the image
- The Laplacian pyramid is the *high-pass filtered* version of the image

The Gaussian/Laplacian Decomposition



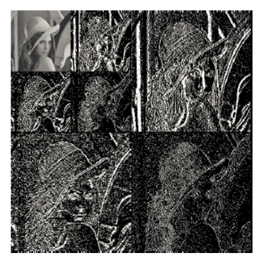
- Each low-pass filtered image is downsampled
- The process is recursively performed

The discrete wavelet transform



- Very similar in structure
- But the bases at each scale are orthogonal to bases at other scales
  – As opposed to a Gaussian kernel matrix

Haar Wavelets



- We have already encountered Haar wavelets

Other characterizations

- Content-based characterizations
  – E.g. Hough transform
    - Captures linear arrangements of pixels
  – Radon transform
  – SIFT features
  – Etc.
- Will revisit in homework..