

# Moving into the Frequency Domain

Frequency domains can be obtained through the transformation from one (**Time** or **Spatial**) domain to the other (**Frequency**) via

- **Discrete Cosine Transform (DCT)**— Heart of **JPEG** and **MPEG Video**, (alt.) MPEG Audio. (**New**)
- **Fourier Transform (FT)** — **MPEG Audio** (**Recap From CM0268**)

**Note:** We mention some image (and video) example in this section as DCT (in particular) but also the FT is commonly applied to filter multimedia data.



Back

Close

# 1D Example

Lets consider a 1D (e.g. Audio) example to see what the different domains mean:

Consider a complicated sound such as the noise of a car horn. We can describe this sound in two related ways:

- Sample the amplitude of the sound many times a second, which gives an approximation to the sound as a function of time.
- Analyse the sound in terms of the pitches of the notes, or frequencies, which make the sound up, recording the amplitude of each frequency.

[Back](#)[Close](#)

# An 8 Hz Sine Wave

In the example (next slide):

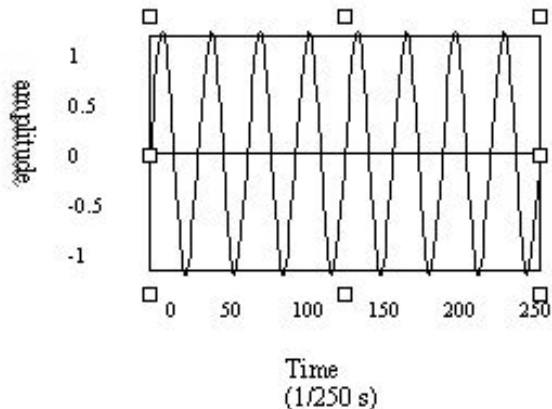
- A signal that consists of a sinusoidal wave at 8 Hz.
- 8 Hz means that wave is completing 8 cycles in 1 second
- The **frequency** of that wave (8 Hz).
- From the frequency domain we can see that the composition of our signal is
  - one wave (one peak) occurring with a frequency of 8 Hz
  - with a magnitude/fraction of 1.0 i.e. it is the whole signal.



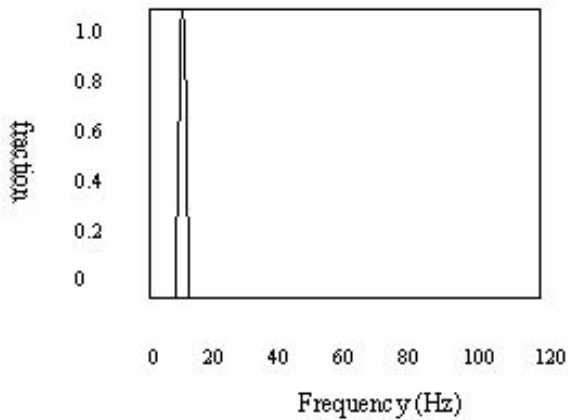
Back

Close

# An 8 Hz Sine Wave (Cont.)



Time  
Domain



Frequency  
Domain



Back

Close

## 2D Image Example

Now images are no more complex really:

- Brightness along a line can be recorded as a set of values measured at equally spaced distances apart,
- **Or** equivalently, at a set of spatial frequency values.
- Each of these frequency values is a **frequency component**.
- An image is a 2D array of pixel measurements.
- We form a 2D grid of spatial frequencies.
- A given frequency component now specifies what contribution is made by data which is changing with specified  $x$  and  $y$  direction spatial frequencies.



Back

Close

# What do frequencies mean in an image?

- Large values at **high** frequency components then the data is changing rapidly on a short distance scale.

*e.g.* a page of text

- Large **low** frequency components then the large scale features of the picture are more important.

*e.g.* a single fairly simple object which occupies most of the image.



Back

Close

# The Road to Compression

How do we achieve compression?

- Low pass filter — ignore high frequency noise components
  - Only store lower frequency components
- High Pass Filter — Spot Gradual Changes
  - If changes to low Eye does not respond so ignore?

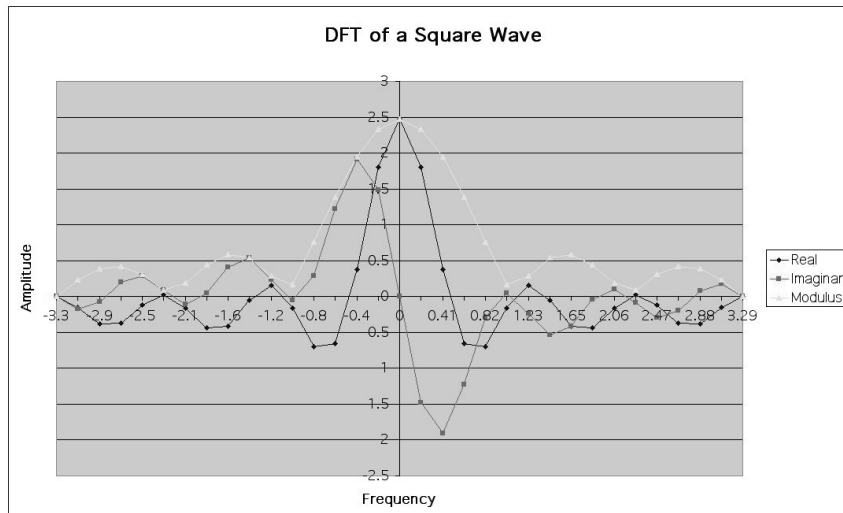


Back

Close

# Visualising Frequency Domain Transforms

- Any function (signal) can be decomposed into purely sinusoidal components (sine waves of different size/shape)
- When added together make up our original signal.
- **Fourier transform** is the tool that performs such an operation

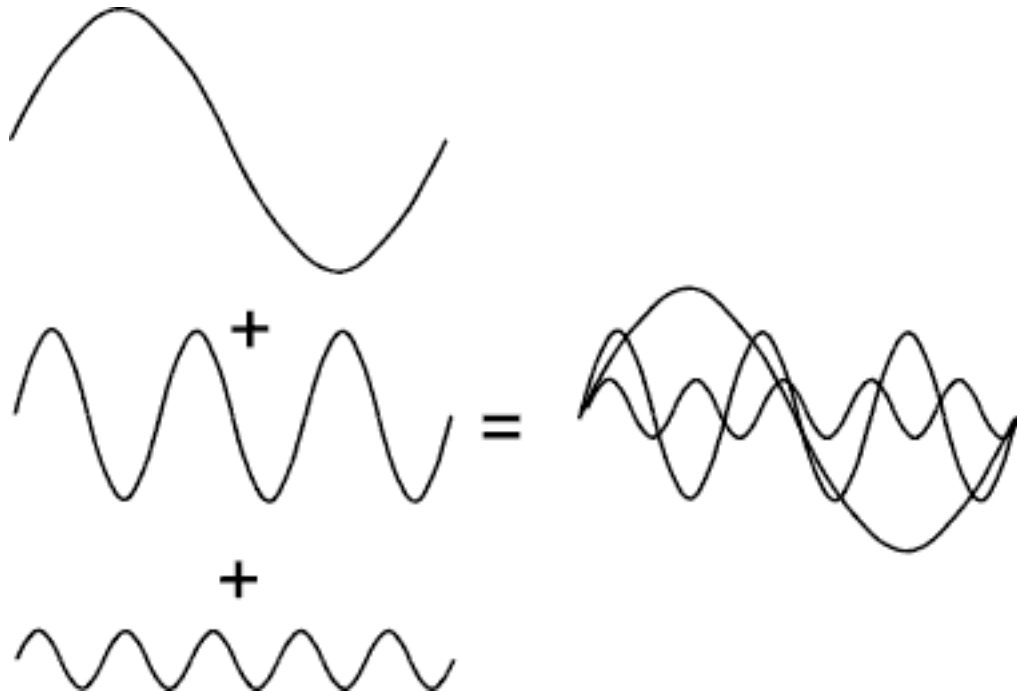


Back

Close

# Summing Sine Waves

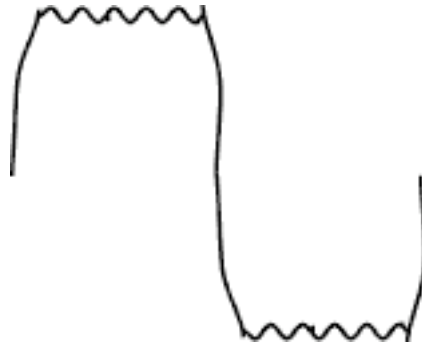
Digital signals are composite signals made up of many sinusoidal frequencies



# Summing Sine Waves to give a Square(ish) Wave

We can take the previous example a step further:

- A 200Hz digital signal (square(ish) wave) may be composed of 200, 600, 1000, 1400, 1800, 2200, 2600, 3000, 3400 and 3800 sinusoidal signals which sum to give:



# So What Does All This Mean?

Transforming a signal into the frequency domain allows us

- **To see what sine waves make up our underlying signal**
- **E.g.**
  - One part sinusoidal wave at 50 Hz and
  - Second part sinusoidal wave at 200 Hz.
- More complex signals will give more complex graphs but the idea is exactly the same.
- Filtering now involves **attenuating** or **removing** certain frequencies — easily performed.
- The graph of the frequency domain is called the frequency spectrum — **more soon**

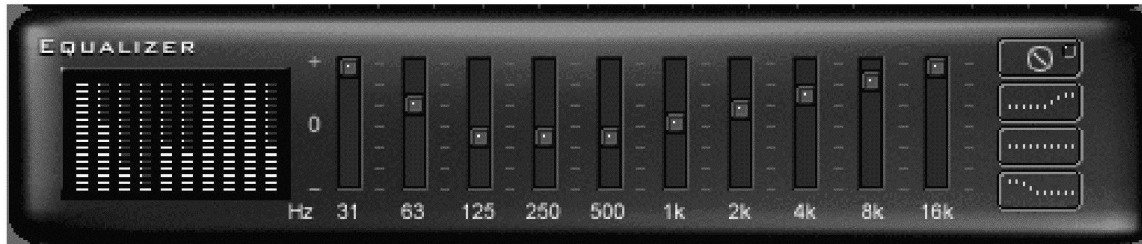


Back

Close

# Visualising Frequency Domain: Think Graphic Equaliser

An easy way to visualise what is happening is to think of a graphic equaliser on a stereo system (or some software audio players, *e.g. iTunes*).



# Fourier Theory

The tool which converts a spatial (real space) description of audio/image data into one in terms of its frequency components is called the **Fourier transform**

The new version is usually referred to as the **Fourier space description** of the data.

We then essentially process the data:

- *E.g.* for **filtering** basically this means attenuating or setting certain frequencies to zero

We then need to convert data back to real audio/imagery to use in our applications.

The corresponding **inverse** transformation which turns a Fourier space description back into a real space one is called the **inverse Fourier transform**.



Back

Close

# Fourier Transform

## 1D Case (e.g. Audio Signal)

Considering a continuous function  $f(x)$  of a single variable  $x$  representing distance.

The Fourier transform of that function is denoted  $F(u)$ , where  $u$  represents spatial frequency is defined by

$$F(u) = \int_{-\infty}^{\infty} f(x)e^{-2\pi i x u} dx.$$

**Note:** In general  $F(u)$  will be a **complex** quantity *even though* the original data is purely **real**.

The meaning of this is that not only is the magnitude of each frequency present important, but that its phase relationship is too.



Back

Close

# Inverse 1D Fourier Transform

The inverse Fourier transform for regenerating  $f(x)$  from  $F(u)$  is given by

$$f(x) = \int_{-\infty}^{\infty} F(u) e^{2\pi i x u} du,$$

which is rather similar, except that the **exponential term has the opposite sign.** – not negative



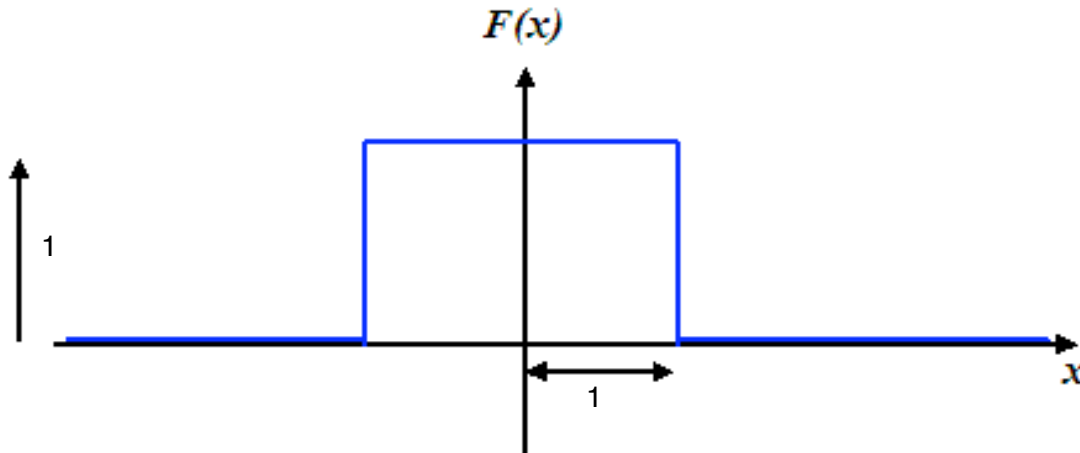
Back

Close

# Example Fourier Transform

Let's see how we compute a Fourier Transform: consider a particular function  $f(x)$  defined as

$$f(x) = \begin{cases} 1 & \text{if } |x| \leq 1 \\ 0 & \text{otherwise,} \end{cases}$$



So its Fourier transform is:

$$\begin{aligned}
 F(u) &= \int_{-\infty}^{\infty} f(x)e^{-2\pi i x u} dx \\
 &= \int_{-1}^1 1 \times e^{-2\pi i x u} dx \\
 &= \frac{1}{2\pi i u} (e^{2\pi i u} - e^{-2\pi i u}) \\
 &= \frac{\sin 2\pi u}{\pi u}.
 \end{aligned}$$

In this case  $F(u)$  is **purely real**, which is a consequence of the original data being symmetric in  $x$  and  $-x$ .

A graph of  $F(u)$  is shown overleaf.

This function is often referred to as the Sinc function.

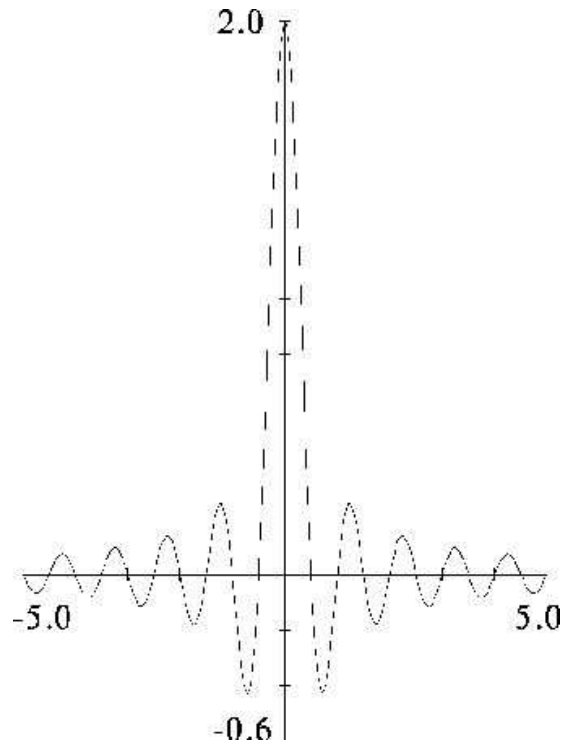


Back

Close

# The Sinc Function

The Fourier transform of a top hat function:



Back

Close

## 2D Case (e.g. Image data)

If  $f(x, y)$  is a function, for example the brightness in an image, its Fourier transform is given by

$$F(u, v) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) e^{-2\pi i(xu+yv)} dx dy,$$

and the inverse transform, as might be expected, is

$$f(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(u, v) e^{2\pi i(xu+yv)} du dv.$$



Back

Close

# But All Our Audio and Image data are Digitised!!

Thus, we need a *discrete* formulation of the Fourier transform:

- Which takes such regularly spaced data values, and
- Returns the value of the Fourier transform for a set of values in frequency space which are equally spaced.

This is done quite naturally by replacing the integral by a summation, to give the *discrete Fourier transform* or DFT for short.

In 1D it is convenient now to assume that  $x$  goes up in steps of 1, and that there are  $N$  samples, at values of  $x$  from 0 to  $N - 1$ .



Back

Close

# 1D Discrete Fourier transform

So the DFT takes the form

$$F(u) = \frac{1}{N} \sum_{x=0}^{N-1} f(x) e^{-2\pi i x u / N},$$

while the inverse DFT is

$$f(x) = \sum_{u=0}^{N-1} F(u) e^{2\pi i x u / N}.$$

**NOTE:** Minor changes from the continuous case are a factor of  $1/N$  in the exponential terms, and also the factor  $1/N$  in front of the forward transform which does not appear in the inverse transform.



# 2D Discrete Fourier transform

The 2D DFT works is similar. So for an  $N \times M$  grid in  $x$  and  $y$  we have

$$F(u, v) = \frac{1}{NM} \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} f(x, y) e^{-2\pi i(xu/N + yv/M)},$$

and

$$f(x, y) = \sum_{u=0}^{N-1} \sum_{v=0}^{M-1} F(u, v) e^{2\pi i(xu/N + yv/M)}.$$



Back

Close

# Balancing the 2D DFT

Often  $N = M$ , and it is then more convenient to redefine  $F(u, v)$  by multiplying it by a factor of  $N$ , so that the forward and inverse transforms are more symmetrical:

$$F(u, v) = \frac{1}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x, y) e^{-2\pi i(xu+yv)/N},$$

and

$$f(x, y) = \frac{1}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} F(u, v) e^{2\pi i(xu+yv)/N}.$$



Back

Close

# Visualising the Fourier Transform

Back to Fourier Transform for Moment:

- It's useful to visualise the Fourier Transform
- Standard tools
- Easily plotted in MATLAB



Back

Close

# The Magnitude Spectrum of Fourier Transform

Recall that the Fourier Transform of even our **real** audio/image data is always **complex**.

- How can we visualise a complex data array?

**Compute the absolute value of the complex data:**

$$|F(k)| = \sqrt{F_R^2(k) + F_I^2(k)} \text{ for } k = 0, 1, \dots, N - 1$$

where  $F_R(k)$  is the real part and  $F_I(k)$  the imaginary part of the  $N$  sampled Fourier Transform,  $F(k)$ .

This is called the **magnitude spectrum** of the Fourier Transform

**Easy in MATLAB:** `Sp = abs(fft(X,N))/N;`

(Normalised form)



# The Phase Spectrum of the Fourier Transform

The Fourier Transform also represent phase, the **phase spectrum** is given by:

$$\varphi = \arctan \frac{F_I(k)}{F_R(k)} \text{ for } k = 0, 1, \dots, N - 1$$

**Easy in MATLAB:** `Ph = angle(fft(X));`  
(in radians)



Back

Close

# Relating a Sample Point to a Frequency Point

When plotting graphs of *FT Spectra* and doing other FT processing we may wish to plot the  $x$ -axis in Hz (Frequency) rather than sample point number  $k = 0, 1, \dots, N - 1$

There is a simple relation between the two:

The sample points go in steps  $k = 0, 1, \dots, N - 1$

For a given sample point  $k$  the frequency relating to this is given by:

$$f_k = k \frac{f_s}{N}$$

where  $f_s$  is the *sampling frequency* and  $N$  the number of samples.

Thus we have equidistant frequency steps of  $\frac{f_s}{N}$  ranging from 0 Hz to  $\frac{N-1}{N} f_s$  Hz



Back

Close

# MATLAB Fourier Frequency Spectra Example

The following code ([fourierspectraeg.m](#)):

```

N=16;
x=cos(2*pi*2*(0:1:N-1)/N)';

figure(1)
subplot(3,1,1);stem(0:N-1,x,'.');
axis([-0.2 N -1.2 1.2]);
legend('Cosine signal x(n)');
ylabel('a');
xlabel('n \rightarrow');

X=abs(fft(x,N))/N;
subplot(3,1,2);stem(0:N-1,X,'.');
axis([-0.2 N -0.1 1.1]);
legend('Magnitude spectrum |X(k)|');
ylabel('b');
xlabel('k \rightarrow');

N=1024;
x=cos(2*pi*(2*1024/16)*(0:1:N-1)/N)';

FS=40000;
f=(0:N-1)/N*FS;
X=abs(fft(x,N))/N;
subplot(3,1,3);plot(f,X);
axis([-0.2*44100/16 max(f) -0.1 1.1]);
legend('Magnitude spectrum |X(f)|');
ylabel('c');
xlabel('f in Hz \rightarrow');

figure(2)
subplot(3,1,1);
plot(f,20*log10(X./(0.5)));
axis([-0.2*44100/16 max(f) ...
-45 20]);
legend('Magnitude spectrum |X(f)| ...
in dB');
ylabel('|X(f)| in dB \rightarrow');
xlabel('f in Hz \rightarrow');

```

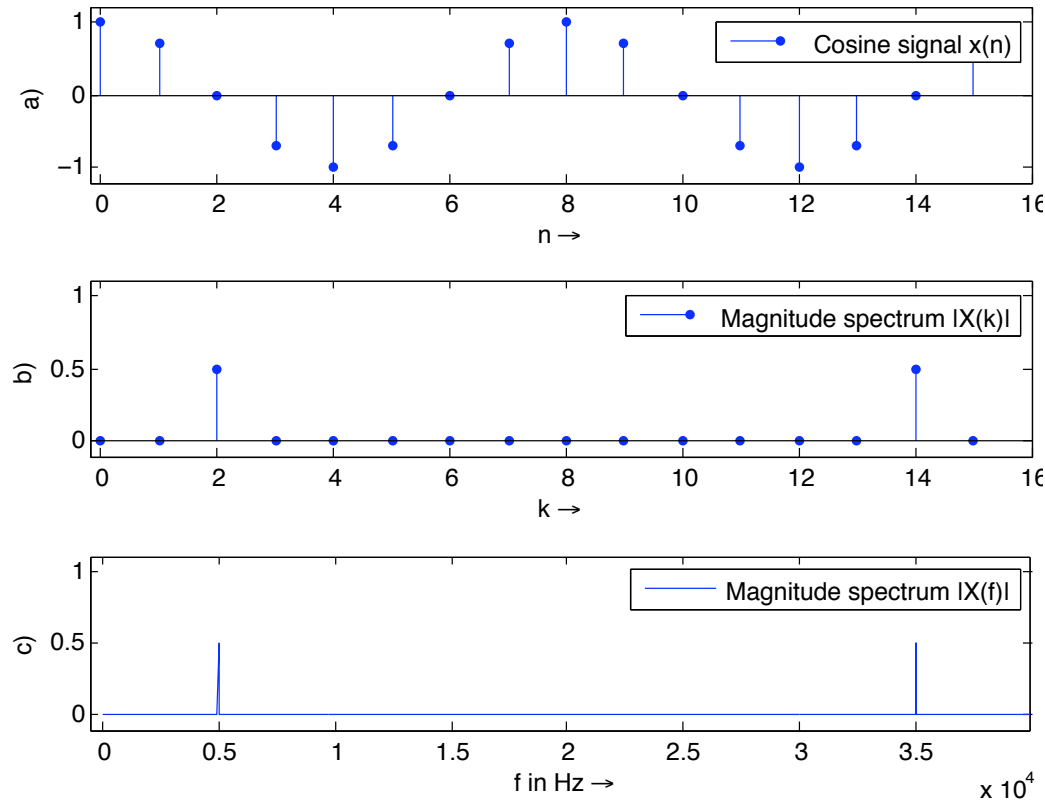


Back

Close

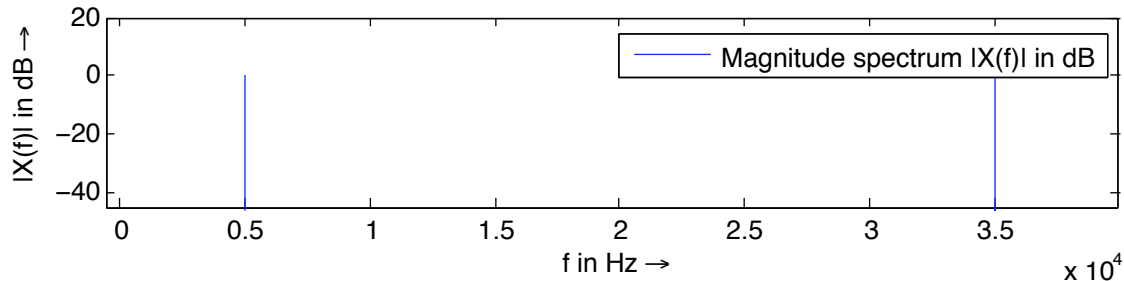
# MATLAB Fourier Frequency Spectra Example (Cont.)

The above code produces the following:



# Magnitude Spectrum in dB

**Note:** It is common to plot both spectra magnitude (also frequency ranges not show here) on a dB/log scale:  
(Last Plot in above code)



Back

Close

# Time-Frequency Representation: Spectrogram

It is often useful to look at the frequency distribution over a short-time:

- Split signal into  $N$  segments
- Do a **windowed** Fourier Transform
  - Window needed to reduce *leakage* effect of doing a short sample FFT.
  - Apply a Blackman, Hamming or Hanning Window
- MATLAB function does the job: `Spectrogram` — see `help spectrogram`
- See also MATLAB's `specgramdemo`



Back

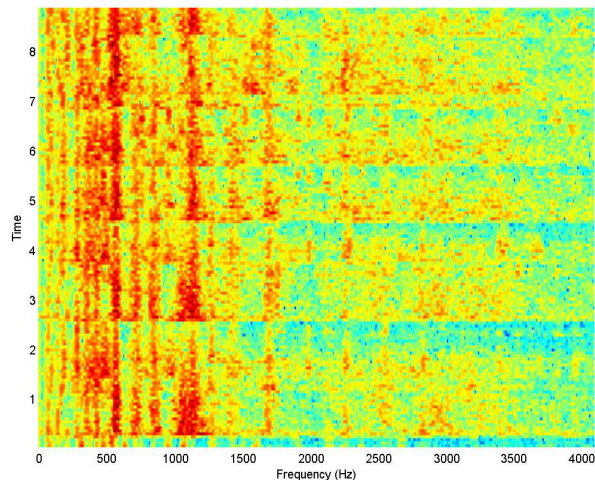
Close

# MATLAB Example

The code:

```
load('handel')
[N M] = size(y);
figure(1)
spectrogram(y, 512, 20, 1024, Fs);
```

Produces the following:



# The Discrete Cosine Transform (DCT)

## Relationship between DCT and FFT

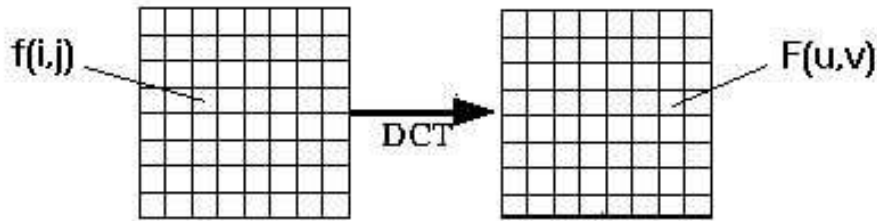
**DCT (Discrete Cosine Transform)** is actually a *cut-down* version of the Fourier Transform or the Fast Fourier Transform (FFT):

- Only the **real** part of FFT
- Computationally simpler than FFT
- DCT — Effective for Multimedia Compression
- DCT **MUCH** more commonly used (than FFT) in Multimedia Image/Video Compression — **more later**
- Cheap MPEG Audio Variant — **more later**

[Back](#)[Close](#)

# Applying The DCT

- Similar to the discrete Fourier transform:
  - it transforms a signal or image from the spatial domain to the frequency domain
  - DCT can approximate lines well with fewer coefficients



- Helps separate the image into parts (or spectral sub-bands) of differing importance (with respect to the image's visual quality).



# 1D DCT

For  $N$  data items 1D DCT is defined by:

$$F(u) = \left(\frac{2}{N}\right)^{\frac{1}{2}} \sum_{i=0}^{N-1} \Lambda(u) \cdot \cos \left[ \frac{\pi \cdot u}{2 \cdot N} (2i + 1) \right] f(i)$$

and the corresponding **inverse** 1D DCT transform is simple  $F^{-1}(u)$ , i.e.:

$$\begin{aligned} f(i) &= F^{-1}(u) \\ &= \left(\frac{2}{N}\right)^{\frac{1}{2}} \sum_{u=0}^{N-1} \Lambda(u) \cdot \cos \left[ \frac{\pi \cdot u}{2 \cdot N} (2i + 1) \right] F(u) \end{aligned}$$

where

$$\Lambda(\xi) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for } \xi = 0 \\ 1 & \text{otherwise} \end{cases}$$



Back

Close

## 2D DCT

For a 2D  $N$  by  $M$  image 2D DCT is defined :

$$F(u, v) = \left(\frac{2}{N}\right)^{\frac{1}{2}} \left(\frac{2}{M}\right)^{\frac{1}{2}} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} \Lambda(u) \cdot \Lambda(v) \cdot \cos \left[ \frac{\pi \cdot u}{2 \cdot N} (2i + 1) \right] \cos \left[ \frac{\pi \cdot v}{2 \cdot M} (2j + 1) \right] \cdot f(i, j)$$

and the corresponding **inverse** 2D DCT transform is simple  $F^{-1}(u, v)$ , i.e.:

$$\begin{aligned} f(i, j) &= F^{-1}(u, v) \\ &= \left(\frac{2}{N}\right)^{\frac{1}{2}} \sum_{u=0}^{N-1} \sum_{v=0}^{M-1} \Lambda(u) \cdot \Lambda(v) \cdot \cos \left[ \frac{\pi \cdot u}{2 \cdot N} (2i + 1) \right] \cdot \cos \left[ \frac{\pi \cdot v}{2 \cdot M} (2j + 1) \right] \cdot F(u, v) \end{aligned}$$

where

$$\Lambda(\xi) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for } \xi = 0 \\ 1 & \text{otherwise} \end{cases}$$



Back

Close

# Performing DCT Computations

The basic operation of the DCT is as follows:

- The input image is N by M;
- $f(i,j)$  is the intensity of the pixel in row  $i$  and column  $j$ ;
- $F(u,v)$  is the DCT coefficient in row  $u_i$  and column  $v_j$  of the DCT matrix.
- For JPEG image (and MPEG video), the DCT input is usually an 8 by 8 (or 16 by 16) array of integers.  
This array contains each image window's respective colour band pixel levels;



Back

Close

# Compression with DCT

- For most images, much of the signal energy lies at low frequencies;
  - These appear in the upper left corner of the DCT.
- Compression is achieved since the lower right values represent higher frequencies, and are often small
  - Small enough to be neglected with little visible distortion.



Back

Close

# Computational Issues (1)

- Image is partitioned into 8 x 8 regions — The DCT input is an 8 x 8 array of integers. **Why 8 x 8?**
- An 8 point DCT would be:

$$F(u, v) = \frac{1}{4} \sum_{i,j} \Lambda(u) \cdot \Lambda(v) \cdot \cos \left[ \frac{\pi \cdot u}{16} (2i + 1) \right] \cdot \cos \left[ \frac{\pi \cdot v}{16} (2j + 1) \right] f(i, j)$$

where

$$\Lambda(\xi) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for } \xi = 0 \\ 1 & \text{otherwise} \end{cases}$$

- The output array of DCT coefficients contains integers; these can range from -1024 to 1023.



## Computational Issues (2)

- Computationally easier to implement and more efficient to regard the DCT as a set of **basis functions**
  - Given a known input array size (8 x 8) can be precomputed and stored.
  - Computing values for a convolution mask (8 x 8 window) that get applied
    - \* Sum values  $x$  pixel the window overlap with image apply window across all rows/columns of image
  - The values as simply calculated from DCT formula.

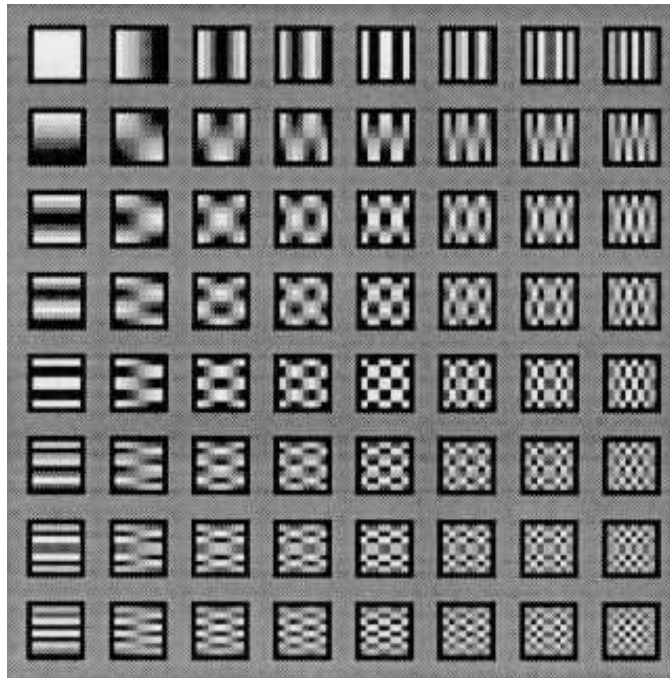


Back

Close

# Computational Issues (3)

## Visualisation of DCT basis functions

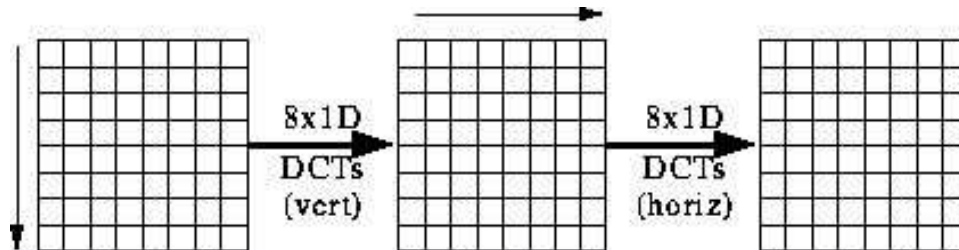


Back

Close

## Computational Issues (4)

- Factoring reduces problem to a series of 1D DCTs (No need to apply 2D form directly):
  - apply 1D DCT (Vertically) to Columns
  - apply 1D DCT (Horizontally) to resultant Vertical DCT above.
  - or alternatively Horizontal to Vertical.



Back

Close

## Computational Issues (5)

- The equations are given by:

$$G(i, v) = \frac{1}{2} \sum_i \Lambda(v) \cdot \cos \left[ \frac{\pi \cdot v}{16} (2j + 1) \right] f(i, j)$$

$$F(u, v) = \frac{1}{2} \sum_i \Lambda(u) \cdot \cos \left[ \frac{\pi \cdot u}{16} (2i + 1) \right] G(i, v)$$

- Most software implementations use fixed point arithmetic. Some fast implementations approximate coefficients so all multiplies are shifts and adds.



Back

Close

# Filtering in the Frequency Domain

FT and DCT methods pretty similar:

- DCT has less data overheads — no complex array part
- FT captures more frequency 'fidelity' (e.g. Phase).

## Low Pass Filter

*Example: Frequencies above the Nyquist Limit,*

*Noise:*

- The idea with noise smoothing is to reduce various spurious effects of a local nature in the image, caused perhaps by
  - noise in the acquisition system,
  - arising as a result of transmission of the data, for example from a space probe utilising a low-power transmitter.



Back

Close

# Frequency Space Smoothing Methods

## Noise = High Frequencies:

- In audio data many spurious peaks in over a short timescale.
- In an image means there are many rapid transitions (over a short distance) in intensity from high to low and back again or vice versa, as faulty pixels are encountered.
- **Not all high frequency data noise though!**

Therefore noise will contribute heavily to the high frequency components of the image when it is considered in Fourier space.

Thus if we reduce the high frequency components — **Low-Pass Filter**, we should reduce the amount of noise in the data.

[Back](#)[Close](#)

# (Low-pass) Filtering in the Fourier Space

We thus create a new version of the image in Fourier space by computing

$$G(u, v) = H(u, v)F(u, v)$$

where:

- $F(u, v)$  is the Fourier transform of the original image,
- $H(u, v)$  is a filter function, designed to reduce high frequencies, and
- $G(u, v)$  is the **Fourier transform of the improved image**.
- Inverse Fourier transform  $G(u, v)$  to get  $g(x, y)$  our **improved image**

**Note:** Discrete Cosine Transform approach identical, sub. FT with DCT

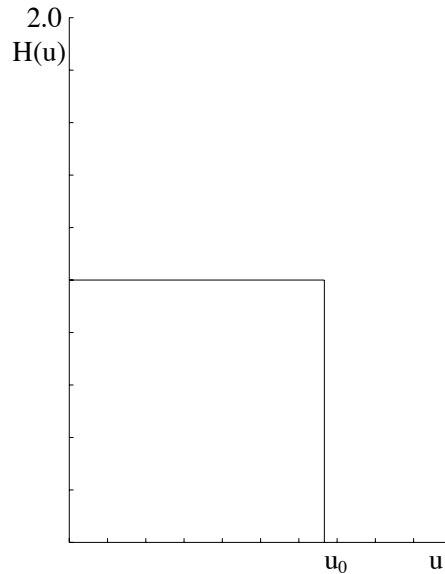


Back

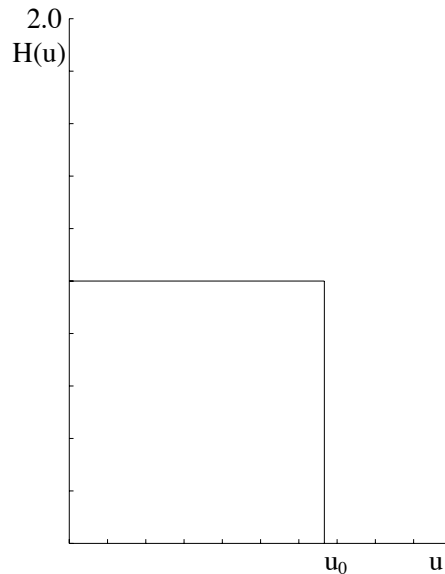
Close

# Ideal Low-Pass Filter

The simplest sort of filter to use is an *ideal low-pass filter*, which in one dimension appears as :



## Ideal Low-Pass Filter (Cont.)



This is a top hat function which is 1 for  $u$  between 0 and  $u_0$ , the *cut-off frequency*, and zero elsewhere.

- So All frequency space information above  $u_0$  is thrown away, and all information below  $u_0$  is kept.
- A **very simple** computational process.

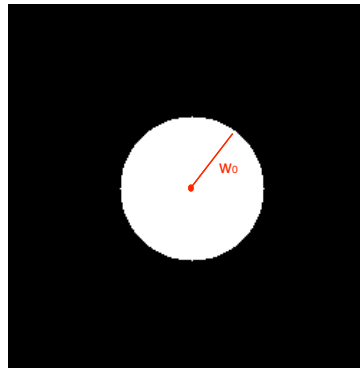
# Ideal 2D Low-Pass Filter

The two dimensional analogue of this is the function

$$H(u, v) = \begin{cases} 1 & \text{if } \sqrt{u^2 + v^2} \leq w_0 \\ 0 & \text{otherwise,} \end{cases}$$

where  $w_0$  is now the cut-off frequency.

Thus, all frequencies inside a radius  $w_0$  are kept, and all others discarded.



Back

Close

# Not So Ideal Low-Pass Filter?

The problem with this filter is that as well as the noise:

- In audio: plenty of other high frequency content
- In Images: edges (places of rapid transition from light to dark) also significantly contribute to the high frequency components.

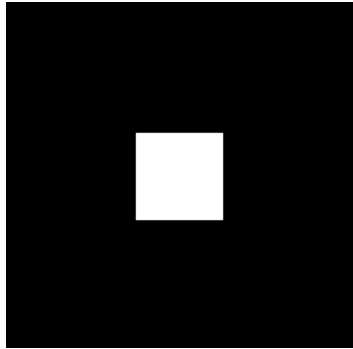
Thus an ideal low-pass filter will tend to *blur* the data:

- High audio frequencies become muffled
- Edges in images become blurred.

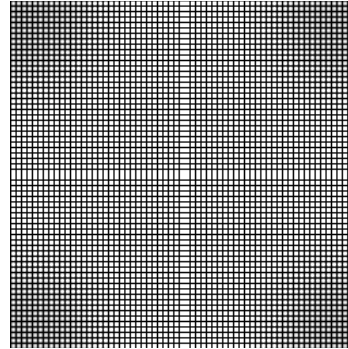
The lower the cut-off frequency is made, the more pronounced this effect becomes in *useful data content*



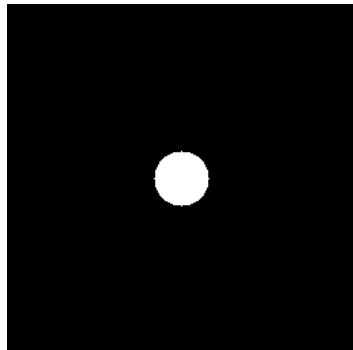
# Ideal Low Pass Filter Example 1



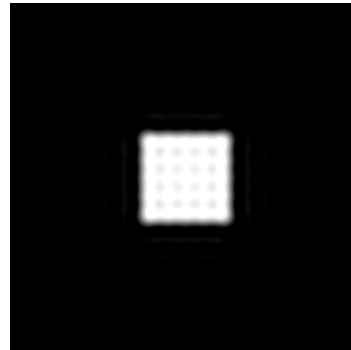
(a) Input Image



(b) Image Spectra



(c) Ideal Low Pass Filter



(d) Filtered Image



# Ideal Low-Pass Filter Example 1 MATLAB Code

## low pass.m:

```
% Create a white box on a black background image
M = 256; N = 256;
image = zeros(M,N)
box = ones(64,64);
%box at centre
image(97:160,97:160) = box;

% Show Image

figure(1);
imshow(image);

% compute fft and display its spectra

F=fft2(double(image));
figure(2);
imshow(abs(fftshift(F)));
```



Back

Close

# Ideal Low-Pass Filter Example 1 MATLAB Code (Cont.)

```
%compute Ideal Low Pass Filter
u0 = 20; % set cut off frequency

u=0:(M-1);
v=0:(N-1);
idx=find(u>M/2);
u(idx)=u(idx)-M;
idy=find(v>N/2);
v(idy)=v(idy)-N;
[V,U]=meshgrid(v,u);
D=sqrt(U.^2+V.^2);
H=double(D<=u0);

% display
figure(3);
imshow(fftshift(H));

% Apply filter and do inverse FFT
G=H.*F;
g=real(ifft2(double(G)));

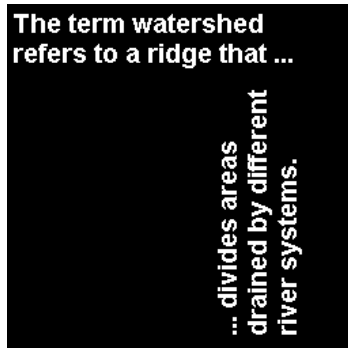
% Show Result
figure(4);
imshow(g);
```



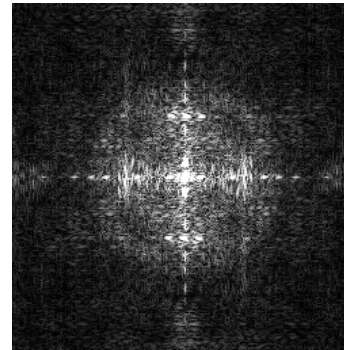
Back

Close

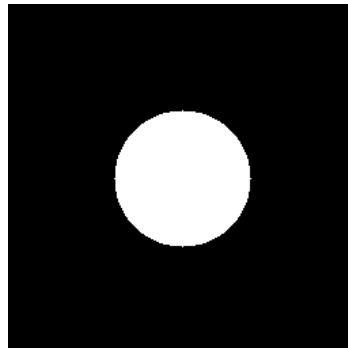
## Ideal Low-Pass Filter Example 2



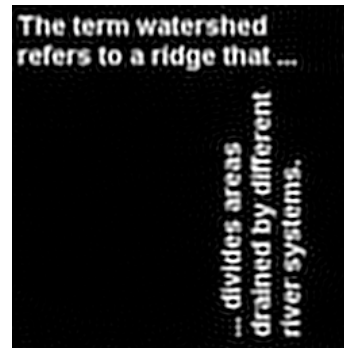
(a) Input Image



(b) Image Spectra



(c) Ideal Low-Pass Filter



(d) Filtered Image



# Ideal Low-Pass Filter Example 2 MATLAB Code

## lowpass2.m:

```
% read in MATLAB demo text image
image = imread('text.png');
[M N] = size(image)

% Show Image

figure(1);
imshow(image);

% compute fft and display its spectra

F=fft2(double(image));
figure(2);
imshow(abs(fftshift(F))/256);
```



Back

Close

## Ideal Low-Pass Filter Example 2 MATLAB Code (Cont.)

```
%compute Ideal Low Pass Filter
u0 = 50; % set cut off frequency

u=0:(M-1);
v=0:(N-1);
idx=find(u>M/2);
u(idx)=u(idx)-M;
idy=find(v>N/2);
v(idy)=v(idy)-N;
[V,U]=meshgrid(v,u);
D=sqrt(U.^2+V.^2);
H=double(D<=u0);

% display
figure(3);
imshow(fftshift(H));

% Apply filter and do inverse FFT
G=H.*F;
g=real(ifft2(double(G)));

% Show Result
figure(4);
imshow(g);
```



Back

Close

# Low-Pass Butterworth Filter

Another filter sometimes used is the *Butterworth low pass filter*.

In the 2D case,  $H(u, v)$  takes the form

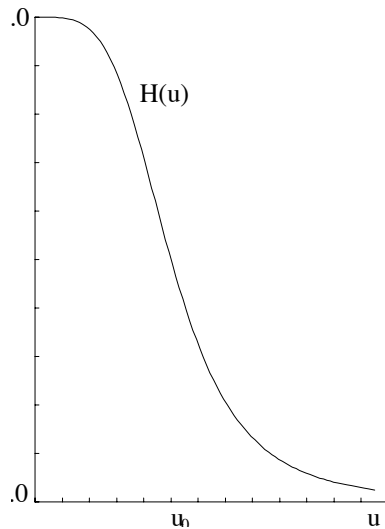
$$H(u, v) = \frac{1}{1 + [(u^2 + v^2)/w_0^2]^n},$$

where  $n$  is called the **order** of the filter.



## Low-Pass Butterworth Filter (Cont.)

This keeps some of the high frequency information, as illustrated by the second order one dimensional Butterworth filter:

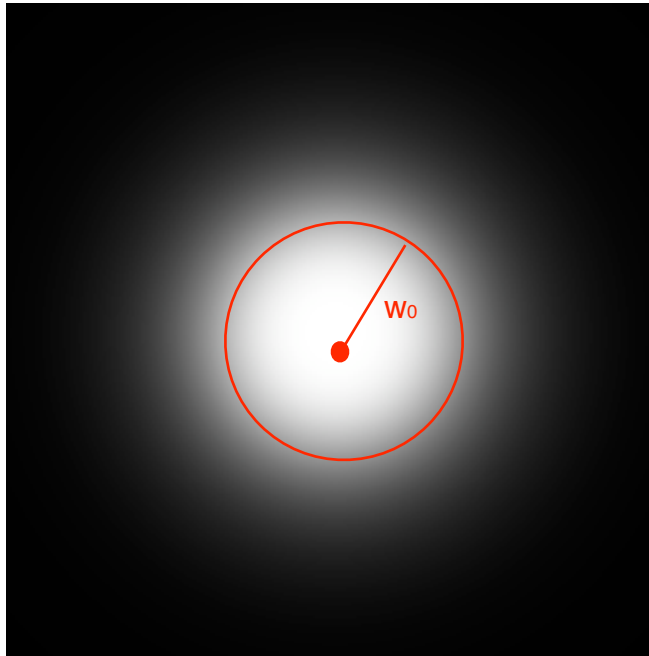


Consequently reduces the blurring.



## Low-Pass Butterworth Filter (Cont.)

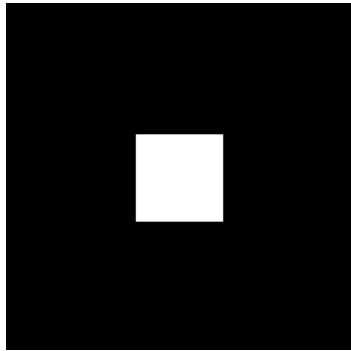
The 2D second order Butterworth filter looks like this:



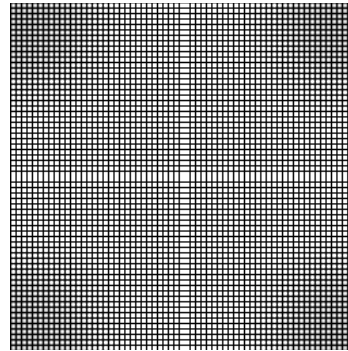
Back

Close

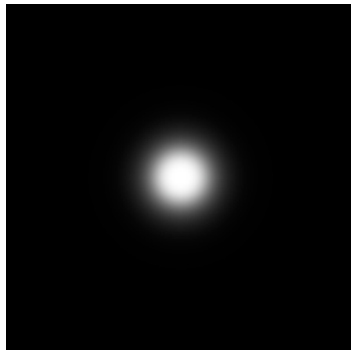
# Butterworth Low Pass Filter Example 1



(a) Input Image



(b) Image Spectra



(c) Butterworth Low-Pass Filter

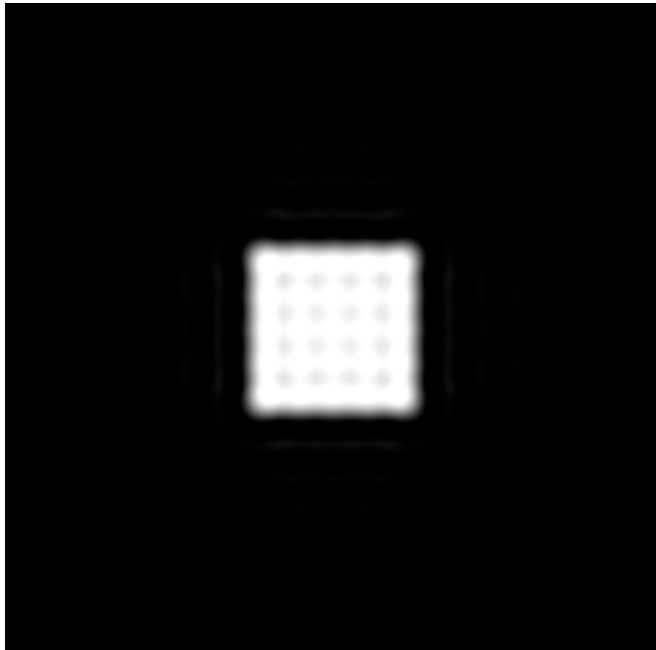


(d) Filtered Image

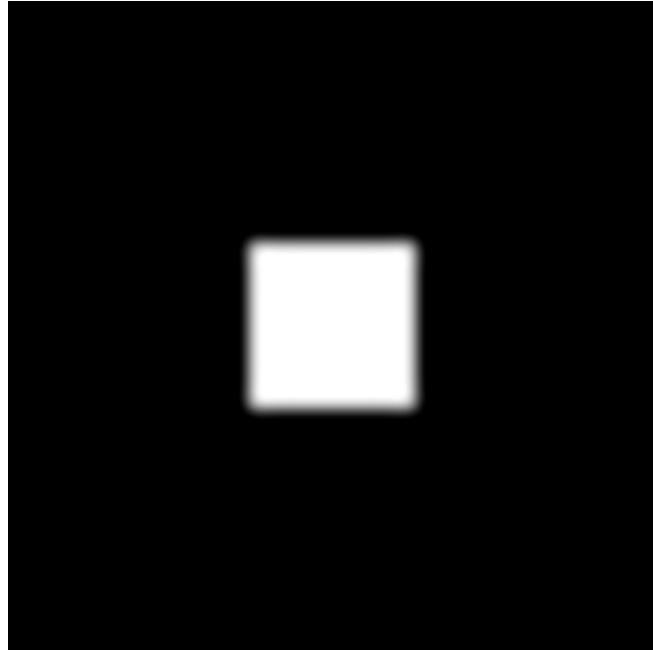


# Butterworth Low-Pass Filter Example 1 (Cont.)

Comparison of Ideal and Butterworth Low Pass Filter:



Ideal Low-Pass



Butterworth Low Pass



# Butterworth Low-Pass Filter Example 1 MATLAB Code

## butterworth.m:

```
% Load Image and Compute FFT as in Ideal Low Pass Filter
% Example 1
.....
% Compute Butterworth Low Pass Filter
u0 = 20; % set cut off frequency

u=0:(M-1);
v=0:(N-1);
idx=find(u>M/2);
u(idx)=u(idx)-M;
idy=find(v>N/2);
v(idy)=v(idy)-N;
[V,U]=meshgrid(v,u);

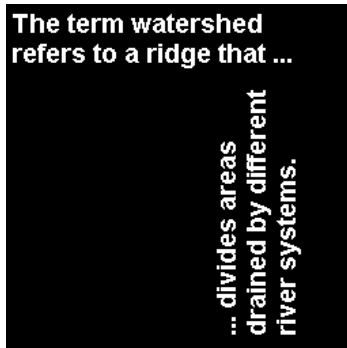
for i = 1: M
    for j = 1:N
        %Apply a 2nd order Butterworth
        UVw = double((U(i,j)*U(i,j) + V(i,j)*V(i,j))/(u0*u0));
        H(i,j) = 1/(1 + UVw*UVw);
    end
end
% Display Filter and Filtered Image as before
```



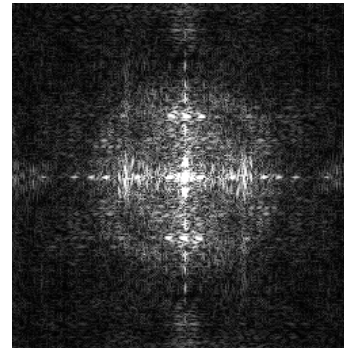
Back

Close

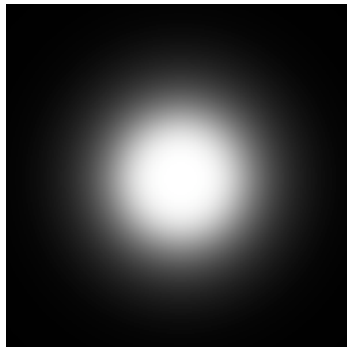
# Butterworth Low-Pass Butterworth Filter Example 2



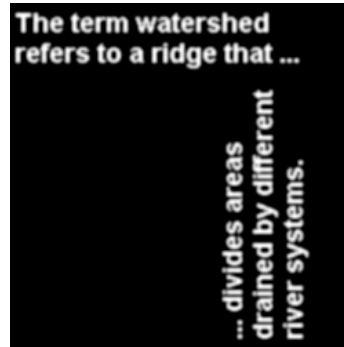
(a) Input Image



(b) Image Spectra



(c) Butterworth Low-Pass Filter

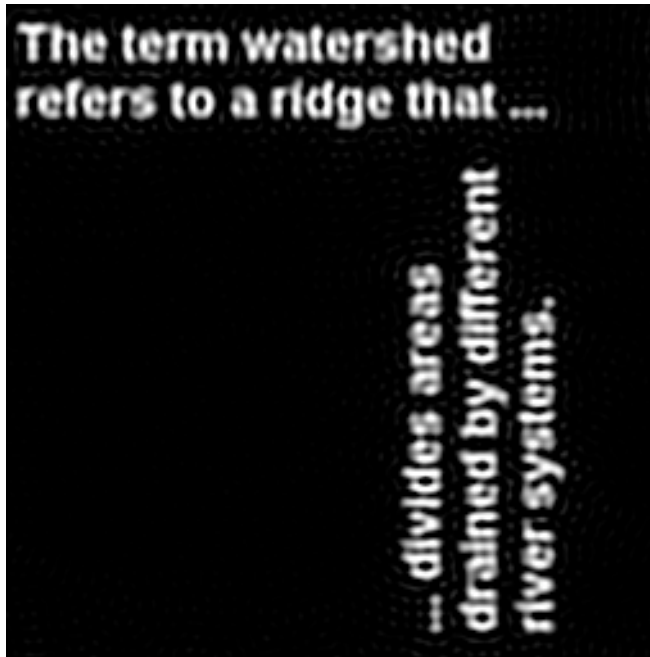


(d) Filtered Image

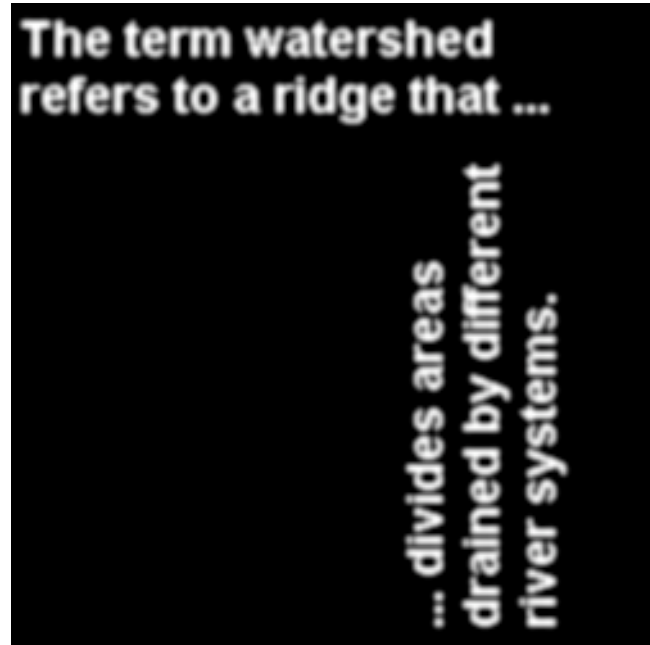


# Butterworth Low-Pass Filter Example 2 (Cont.)

Comparison of Ideal and Butterworth Low-Pass Filter:



Ideal Low Pass



Butterworth Low Pass



# Butterworth Low Pass Filter Example 2 MATLAB Code

## butterworth2.m:

```
% Load Image and Compute FFT as in Ideal Low Pass Filter
% Example 2
.....
% Compute Butterworth Low Pass Filter
u0 = 50; % set cut off frequency

u=0:(M-1);
v=0:(N-1);
idx=find(u>M/2);
u(idx)=u(idx)-M;
idy=find(v>N/2);
v(idy)=v(idy)-N;
[V,U]=meshgrid(v,u);

for i = 1: M
    for j = 1:N
        %Apply a 2nd order Butterworth
        UVw = double((U(i,j)*U(i,j) + V(i,j)*V(i,j))/(u0*u0));
        H(i,j) = 1/(1 + UVw*UVw);
    end
end
% Display Filter and Filtered Image as before
```



Back

Close

# Other Filters

**High-Pass Filters** — opposite of low-pass, select high frequencies, attenuate those **below**  $u_0$

**Band-pass** — allow frequencies in a range  $u_0 \dots u_1$  attenuate those outside this range

**Band-reject** — opposite of band-pass, attenuate frequencies within  $u_0 \dots u_1$  **select** those **outside** this range

**Notch** — attenuate frequencies in a narrow bandwidth around cut-off frequency,  $u_0$

**Resonator** — amplify frequencies in a narrow bandwidth around cut-off frequency,  $u_0$

Other filters exist that are a combination of the above



Back

Close

# Convolution

Several important audio and optical effects can be described in terms of convolutions.

- In fact the above Fourier filtering is applying convolutions of low pass filter where the equations are Fourier Transforms of real space equivalents.
- deblurring — high pass filtering
- reverb — **see CM0268**.



# 1D Convolution

Let us examine the concepts using 1D continuous functions.

The convolution of two functions  $f(x)$  and  $g(x)$ , written  $f(x) * g(x)$ , is defined by the integral

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

[Back](#)[Close](#)

# 1D Convolution Example

For example, let us take two top hat functions of the type described earlier.

Let  $f(\alpha)$  be the top hat function shown:

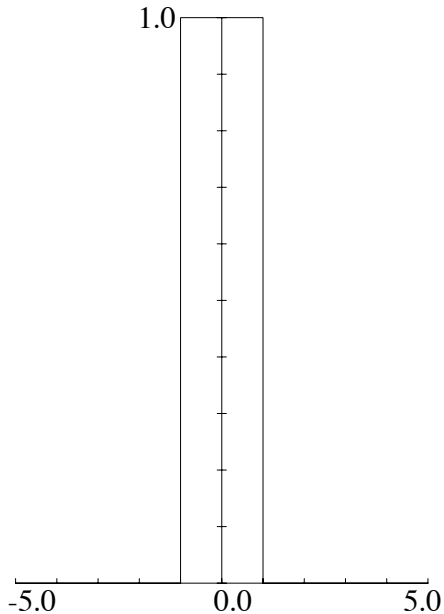
$$f(\alpha) = \begin{cases} 1 & \text{if } |\alpha| \leq 1 \\ 0 & \text{otherwise,} \end{cases}$$

and let  $g(\alpha)$  be as shown in next slide, defined by

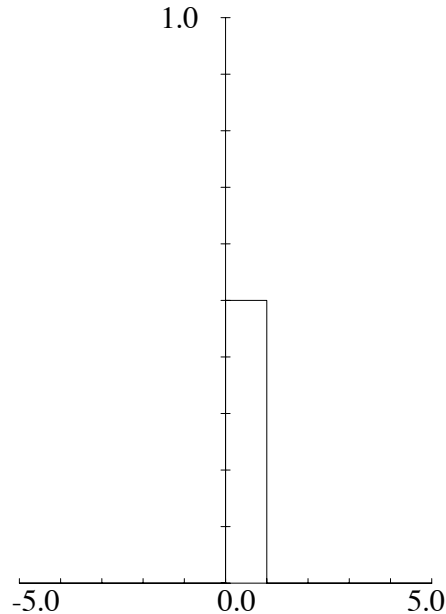
$$g(\alpha) = \begin{cases} 1/2 & \text{if } 0 \leq \alpha \leq 1 \\ 0 & \text{otherwise.} \end{cases}$$



# 1D Convolution Example (Cont.)



$$f(\alpha) = \begin{cases} 1 & \text{if } |\alpha| \leq 1 \\ 0 & \text{otherwise,} \end{cases}$$

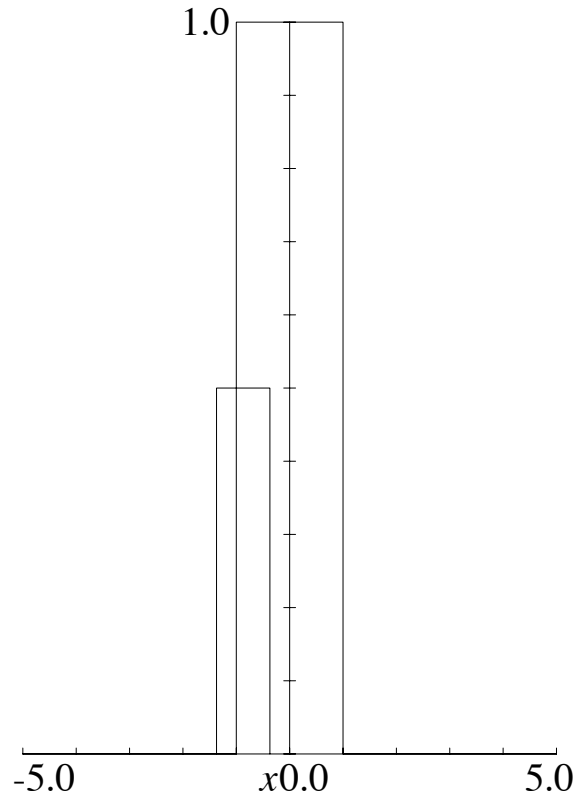


$$g(\alpha) = \begin{cases} 1/2 & \text{if } 0 \leq \alpha \leq 1 \\ 0 & \text{otherwise.} \end{cases}$$



# 1D Convolution Example (Cont.)

- $g(-\alpha)$  is the reflection of this function in the vertical axis,
- $g(x - \alpha)$  is the latter shifted to the right by a distance  $x$ .
- Thus for a given value of  $x$ ,  $f(\alpha)g(x - \alpha)$  integrated over all  $\alpha$  is the area of overlap of these two top hats, as  $f(\alpha)$  has unit height.
- An example is shown for  $x$  in the range  $-1 \leq x \leq 0$  opposite



# 1D Convolution Example (cont.)

If we now consider  $x$  moving from  $-\infty$  to  $+\infty$ , we can see that

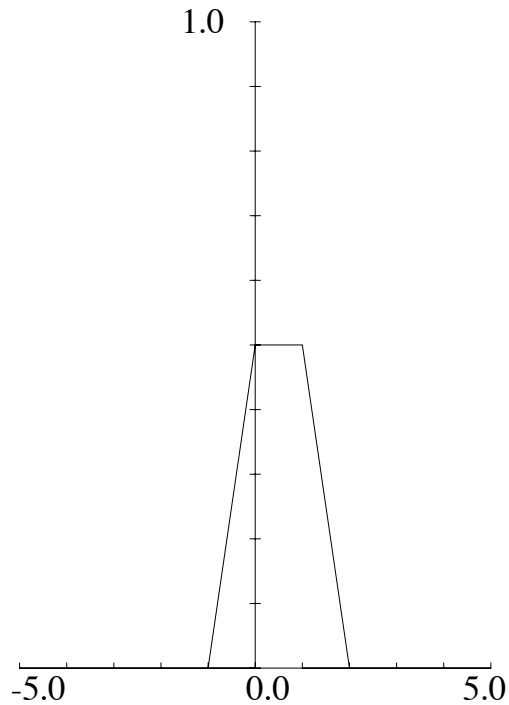
- For  $x \leq -1$  or  $x \geq 2$ , there is no overlap;
- As  $x$  goes from  $-1$  to  $0$  the area of overlap steadily increases from  $0$  to  $1/2$ ;
- As  $x$  increases from  $0$  to  $1$ , the overlap area remains at  $1/2$ ;
- Finally as  $x$  increases from  $1$  to  $2$ , the overlap area steadily decreases again from  $1/2$  to  $0$ .
- Thus the convolution of  $f(x)$  and  $g(x)$ ,  $f(x) * g(x)$ , in this case has the form shown on next slide



Back

Close

# 1D Convolution Example (cont.)



Result of  $f(x) * g(x)$



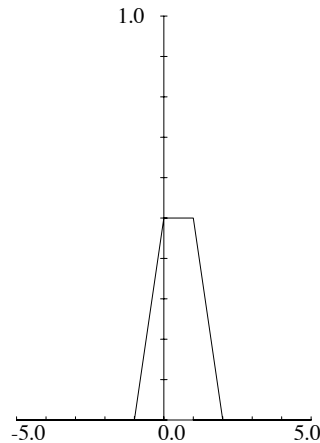
Back

Close

# 1D Convolution Example (cont.)

Mathematically the convolution is expressed by:

$$f(x) * g(x) = \begin{cases} (x+1)/2 & \text{if } -1 \leq x \leq 0 \\ 1/2 & \text{if } 0 \leq x \leq 1 \\ 1-x/2 & \text{if } 1 \leq x \leq 2 \\ 0 & \text{otherwise.} \end{cases}$$



# Fourier Transforms and Convolutions

One major reason that Fourier transforms are so important in image processing is the **convolution theorem** which states that:

*If  $f(x)$  and  $g(x)$  are two functions with Fourier transforms  $F(u)$  and  $G(u)$ , then the Fourier transform of the convolution  $f(x) * g(x)$  is simply the **product** of the **Fourier transforms** of the **two functions**,  $F(u)G(u)$ .*

## Recall our Low Pass Filter Example (MATLAB CODE)

```
% Apply filter  
G=H.*F;
```

Where  $F$  was the Fourier transform of the image,  $H$  the filter



Back

Close

# Computing Convolutions with the Fourier Transform

*E.g.:*

- To apply some reverb to an audio signal, **example later**
- To compensate for a less than ideal image capture system:

To do this **fast convolution** we simply:

- Take the Fourier transform of the audio/imperfect image,
- Take the Fourier transform of the function describing the effect of the system,
- Multiply by the effect to apply effect to audio data
- To remove/compensate for effect: Divide by the effect to obtain the Fourier transform of the ideal image.
- Inverse Fourier transform to recover the new audio/ideal image.

This process is sometimes referred to as **deconvolution**.



Back

Close