CM0340: Multimedia
Tutorial 2
Digital Signal Processing Recap
Fourier Transforms

Prof. David Marshall

School of Computer Science & Informatics

February 6, 2013

Fourier Transform

The Frequency Domain

The **Frequency domain** can be obtained through the transformation, via **Fourier Transform (FT)**, from

- one (**Temporal (Time)** or **Spatial**) domain

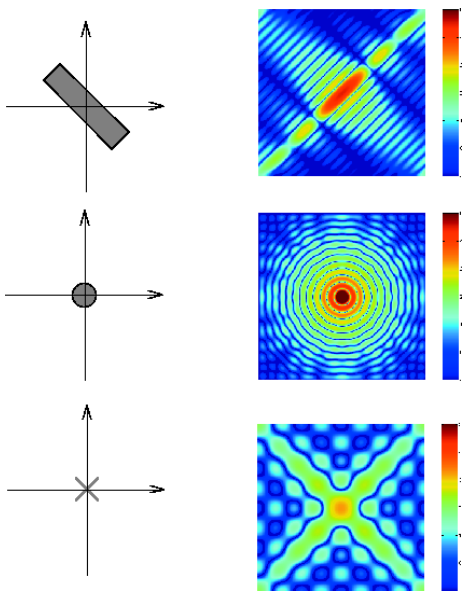
  to the other

- **Frequency** Domain
  - We do not think in terms of signal or pixel intensities but rather underlying sinusoidal waveforms of varying frequency, amplitude and phase.

Applications of Fourier Transform

Numerous Applications including:

- Essential tool for Engineers, Physicists, Mathematicians and Computer Scientists
- Fundamental tool for Digital Signal Processing and Image Processing
- Many types of Frequency Analysis:
  - **Filtering**
  - **Noise Removal**
  - Signal/Image Analysis
  - Simple implementation of **Convolution**
  - **Audio** and Image **Effects Processing**.
  - Signal/Image Restoration — *e.g.* **Deblurring**
  - Signal/Image Compression — **MPEG** (Audio and Video), **JPEG** use related techniques.
  - Many more .....



Introducing Frequency Space

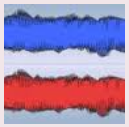
1D Audio Example

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

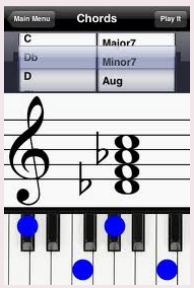
Consider a **complicated sound** such as the a **chord** played on a **piano** or a **guitar**.

We can describe this sound in two related ways:

Temporal Domain : Sample the **amplitude** of the sound many times a second, which gives an approximation to the sound as a **function** of **time**.



Frequency Domain : **Analyse** the sound in terms of the **pitches** of the notes, or **frequencies**, which make the sound up, recording the **amplitude** of **each frequency**.



Fundamental Frequencies

Db : 554.40Hz

F : 698.48Hz

Ab : 830.64Hz

C: 1046.56Hz

plus harmonics/partial frequencies ....

Back to Basics

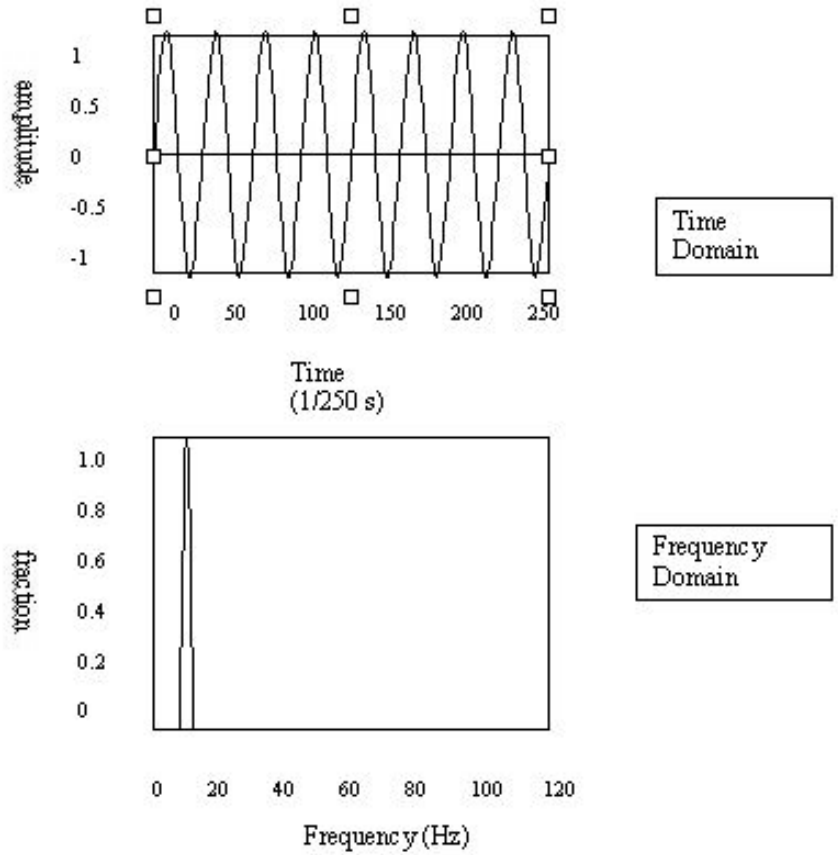
An 8 Hz Sine Wave

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 is 8 Hz.

From the **frequency domain** we can see that the composition of our signal is

- **one peak** occurring with a frequency of 8 Hz — there is only one sine wave here.
  - with a **magnitude/fraction** of **1.0** i.e. it is the **whole signal**.



2D Image Example

What do Frequencies in an Image Mean?

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.

Frequency components of an image

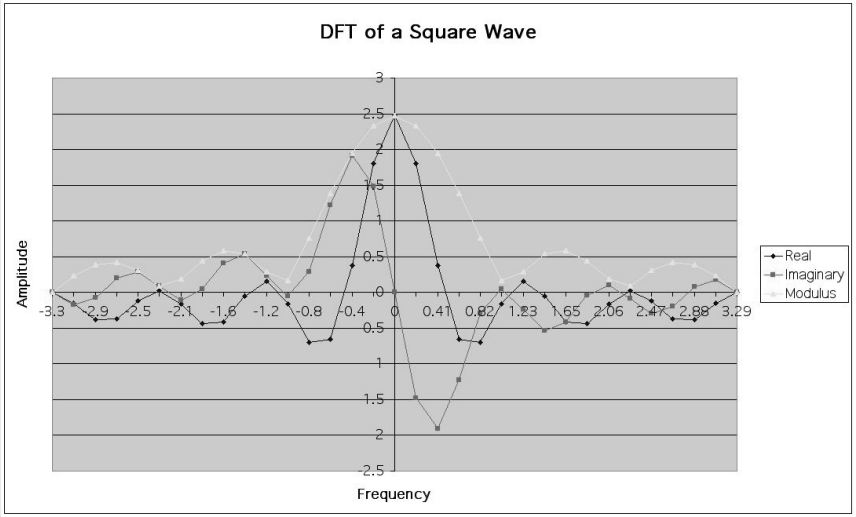
What do Frequencies in an Image Mean? (Cont.)

- Large values at **high** frequency components then the data is changing rapidly on a short distance scale.
  - *e.g.* a **page of text**
  - **However**, **Noise** contributes (very) **High Frequencies** also
- 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.

Visualising Frequency Domain Transforms

Sinusoidal Decomposition

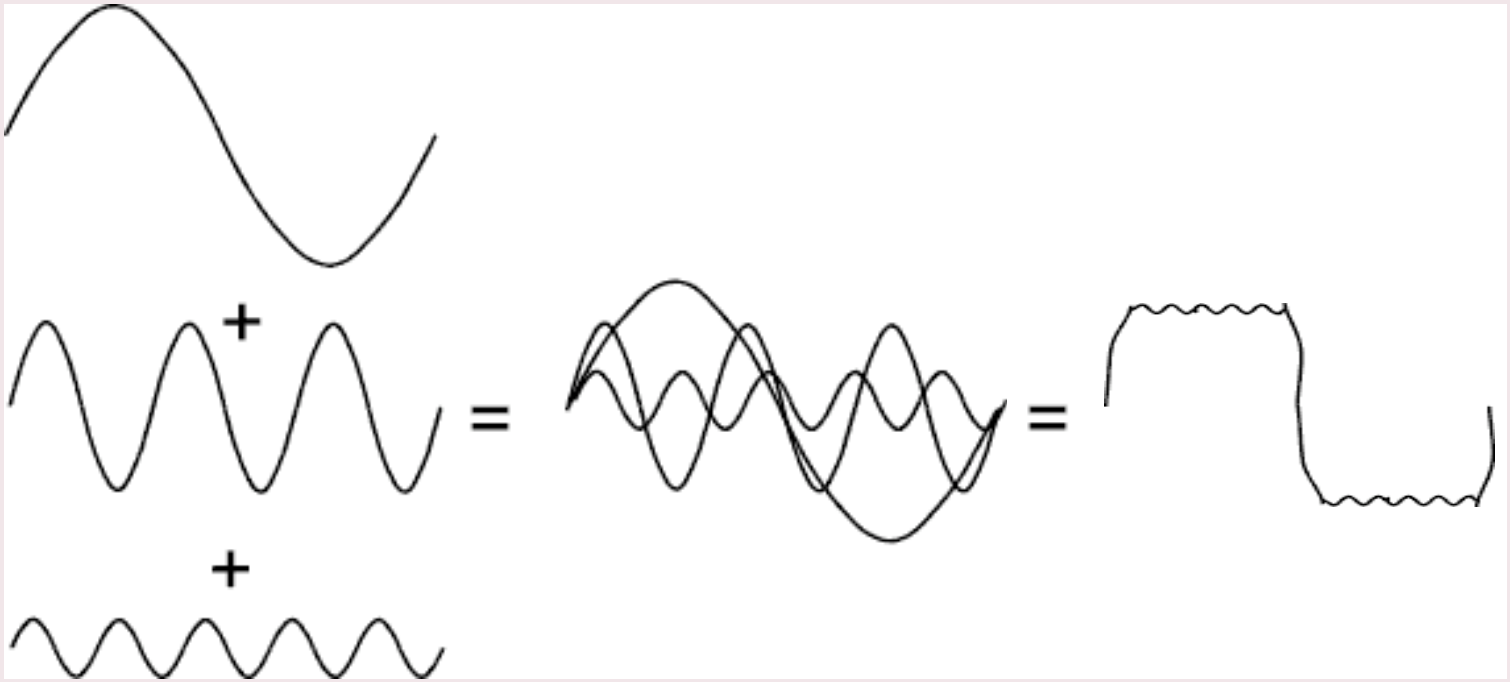
- **Any digital signal** (function) can be **decomposed** into purely **sinusoidal components**
  - Sine waves of different size/shape — varying **amplitude**, **frequency** and **phase**.
- When **added** back **together** they **reconstitute** the **original signal**.
- The **Fourier transform** is the tool that performs such an operation.



Summing Sine Waves. Example: to give a Square(ish) Wave (**E.g. Additive Synthesis**)

Digital signals are composite signals made up of many sinusoidal frequencies

- A 200Hz digital signal (**square(ish) wave**) may be a composed of 200, 600, 1000, *etc.* sinusoidal signals which sum to give:



Summary so far

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.
  - *Etc.*
- More **complex** signals will give more complex decompositions but the idea is exactly the same.

How is this Useful then?

Basic Idea of Filtering in Frequency Space

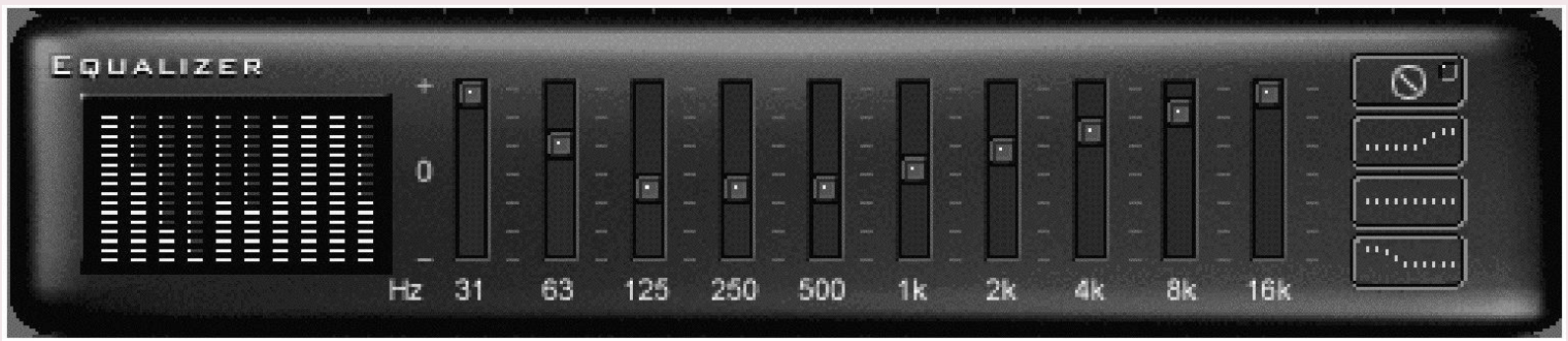
Filtering now involves **attenuating** or **removing** certain frequencies — **easily performed**:

- **Low pass filter** —
  - **Ignore high frequency** noise components — make **zero** or a **very low value**.
  - Only store lower frequency components
- **High Pass Filter** — **opposite of above**
- **Bandpass Filter** — only **allow** frequencies in a **certain range**.

Visualising the Frequency Domain

Think Graphic Equaliser

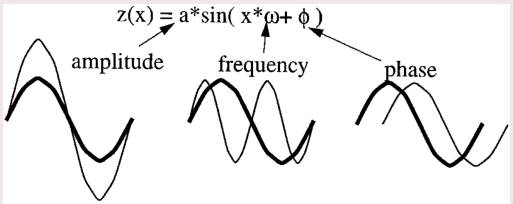
An easy way to visualise what is happening is to think of a graphic equaliser on a stereo system (or some software audio players, *e.g. iTunes*).



So are we ready for the Fourier Transform?

We have all the Tools....

- This lecture, so far, (hopefully) set the context for Frequency decomposition.
- Past **CM2202 Lectures**:
  - **Odd/Even Functions**: $\sin(-x) = -\sin(x)$, $\cos(-x) = \cos(x)$
  - **Complex Numbers**: **Phasor Form** $re^{i\phi} = r(\cos\phi + i\sin\phi)$
  - Calculus **Integration**: $\int e^{kx}\,dx = \frac{e^{kx}}{k}$
- Digital Signal Processing:
  - Basic Waveform Theory. Sine Wave $y = A.sin(2\pi.n.F_w/F_s)$ where: A = amplitude, F_w = wave frequency, F_s = sample frequency, n is the sample index.
  - Relationship between Amplitude, Frequency and Phase:



    - Cosine is a Sine wave 90° out of phase
  - Impulse Responses
- DSP + Image Proc.: Filters and other processing, Convolution

Fourier Theory

Introducing The Fourier Transform

The tool which **converts** a **spatial** or **temporal** (real space) **description** of **audio**/**image** data, for example, 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** (or **invert**) 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**.

1D Fourier Transform

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

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

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

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

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**.
- Recall **Phasors** from **Complex Number Lectures**.
  - $e^{-2\pi ixu}$ above is a **Phasor**.

Inverse Fourier Transform

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 to the (forward) Fourier transform

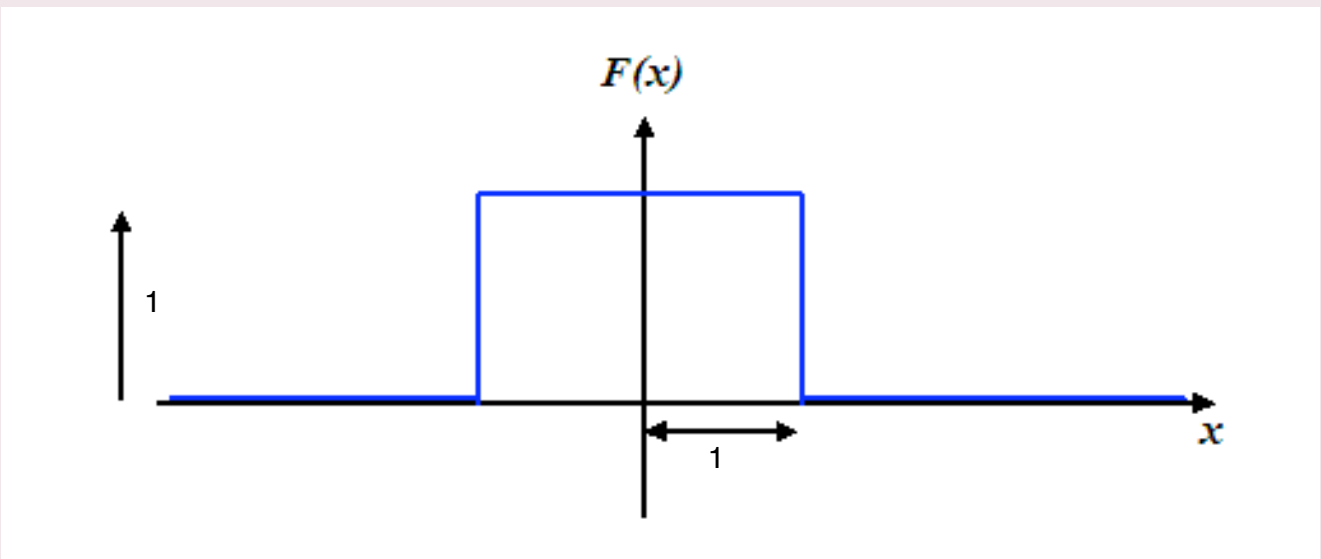
- except that the **exponential term has the opposite sign.**
- It is **not negative**

Fourier Transform Example

Fourier Transform of a Top Hat Function

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

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



The Sinc Function (1)

We derive the Sinc function

So its Fourier transform is:

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

$$\begin{aligned} \sin\theta &= \frac{\mathbf{e}^{\mathbf{i}\theta} - \mathbf{e}^{-\mathbf{i}\theta}}{\mathbf{2i}}, \text{ So:} \\ \mathbf{F(u)} &= \frac{\sin \mathbf{2\pi u}}{\pi \mathbf{u}}. \end{aligned}$$

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

- $f(x)$ is an **even** function.

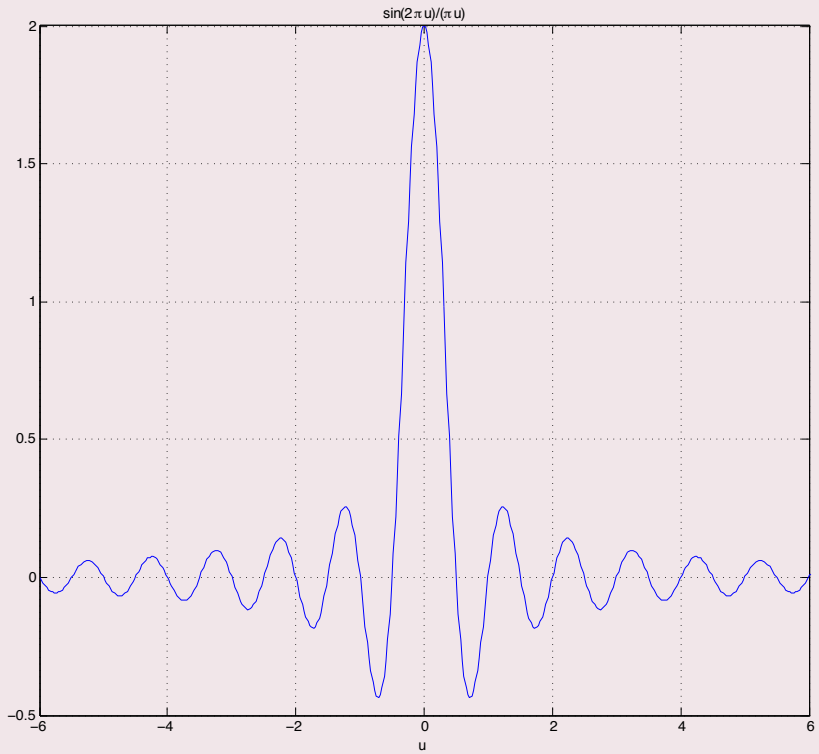
A graph of $F(u)$ is shown overleaf.

This function is often referred to as the **Sinc function**.

The Sinc Function Graph

The Sinc Function

The Fourier transform of a top hat function, the **Sinc function**:



The 2D Fourier Transform

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

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

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

and the **inverse transform**, as might be expected, is

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

The Discrete Fourier Transform

But All Our Audio and Image data are Digitised!!

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

- **Assumes 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.

1D Discrete Fourier transform

1D Case:

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 $N-1$.

So the DFT takes the form

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

while the inverse DFT is

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

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

2D Case

The **2D DFT** works is similar.

So for an $N \times M$ grid in x and y we have

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

and

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

Balancing the 2D DFT

Most Images are Square

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

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

and

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

Fourier Transforms in MATLAB

`fft()` and `fft2()`

MATLAB provides functions for 1D and 2D **Discrete Fourier Transforms** (**DFT**):

fft(X)	is the 1D discrete Fourier transform (DFT) of **vector** X. For **matrices**, the FFT operation is applied to **each column** — **NOT** a 2D DFT transform.
fft2(X)	returns the 2D Fourier transform of matrix X. If X is a vector, the result will have the same orientation.
fftn(X)	returns the N-D discrete Fourier transform of the **N-D array X**.
Inverse DFT	**ifft()**, **ifft2()**, **ifftn()** perform the **inverse** DFT.

See appropriate MATLAB **help/doc** pages for **full details**.

Plenty of examples to Follow.

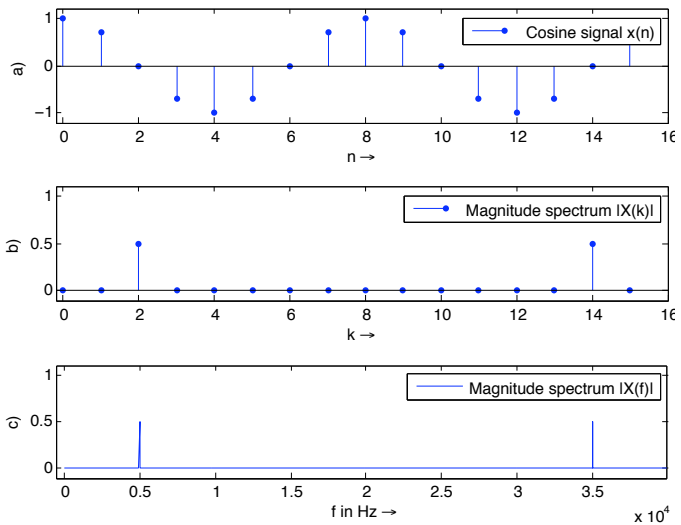
See also: **MALTAB Docs Image Processing → User's Guide → Transforms → Fourier Transform**

Visualising the Fourier Transform

Visualising the Fourier Transform

Having computed a DFT it might be useful to visualise its result:

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



The Magnitude Spectrum of Fourier Transform

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

- **Phasors**: This is how we encode the **phase** of the underlying signal's **Fourier Components**.

How can we visualise a complex data array?

Back to Complex Numbers:

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

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

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

Recall MATLAB: `Sp = abs(fft(X,N))/N;`
(**Normalised form**)

The Phase Spectrum of Fourier Transform

The Phase Spectrum

Phase Spectrum

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

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

Recall MATLAB: `phi = angle(fft(X,N))`

Relating a Sample Point to a Frequency Point

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

There is a **simple relation** between the two:

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

Time-Frequency Representation: Spectrogram

Spectrogram

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

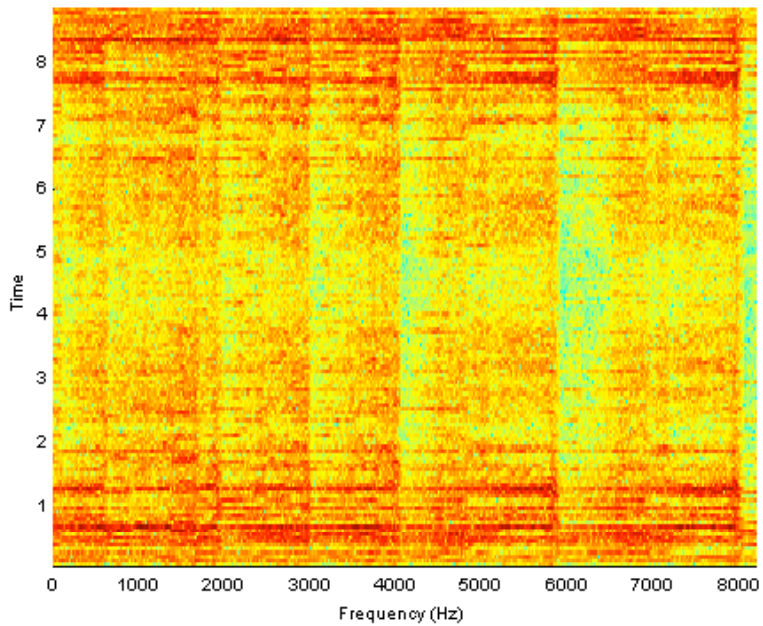
- Split signal into N segments
- Do a **windowed Fourier Transform** — **Short-Time Fourier Transform** (**STFT**)
  - Window needed to reduce *leakage* effect of doing a shorter sample SFFT.
  - Apply a **Blackman**, **Hamming** or **Hanning** Window
- MATLAB function does the job: `Spectrogram` — see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

MATLAB spectrogram Example

spectrogrameg.m

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

Produces the following:



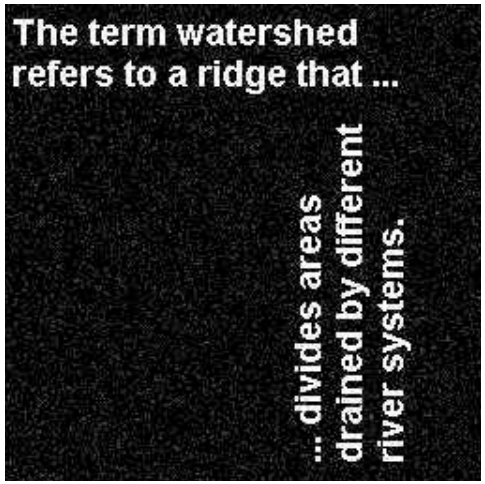
Filtering in the Frequency Domain

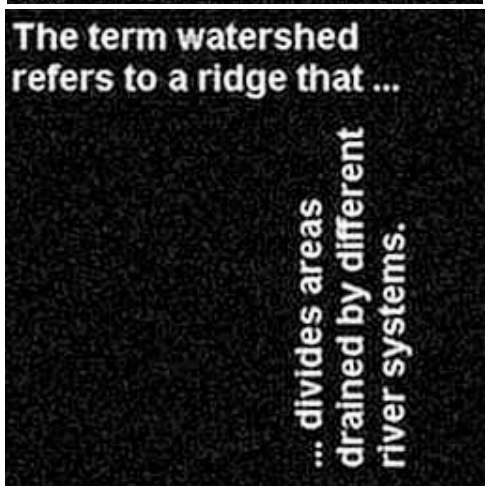
Low Pass Filter

Example: *Audio Hiss, 'Salt and Pepper' noise in images,*

Noise:

- The idea with **noise Filtering** 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.





Frequency Space Filtering Methods

Low Pass Filtering — Remove Noise

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 signal when it is **analysed** in **Fourier space**.

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

(Low-pass) Filtering in the Fourier Space

Low Pass Filtering with the Fourier Transform

We **filter** in Fourier space by computing

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

where:

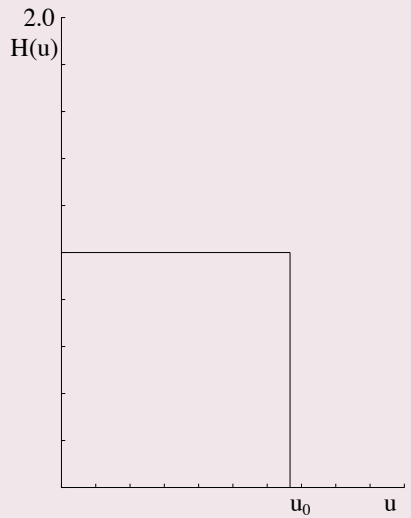
- $F(u, v)$ is the **Fourier transform** of the **original** image,
- $H(u, v)$ is a filter function, designed to reduce high frequencies, and
- $G(u, v)$ is the **Fourier transform of the improved image**.
- **Inverse Fourier transform** $G(u, v)$ to get $g(x, y)$ our **improved image**

Ideal Low-Pass Filter

We need to design or compute $H(u, v)$

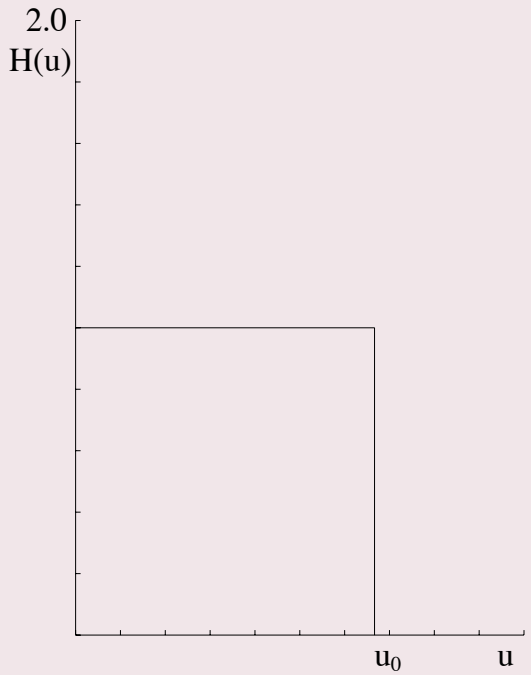
- If we know $h(x, y)$ or have a discrete sample of $h(x, y)$ can compute its Fourier Transform
- Can simply design simple filters in Frequency Space

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



Ideal Low-Pass Filter (2)

How the Low Pass Filter Works with Frequencies



This is a $h(x, y)$ 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 **discarded**, and all information **below** u_0 is **kept**.
- A **very simple** computational process.

Ideal 2D Low-Pass Filter

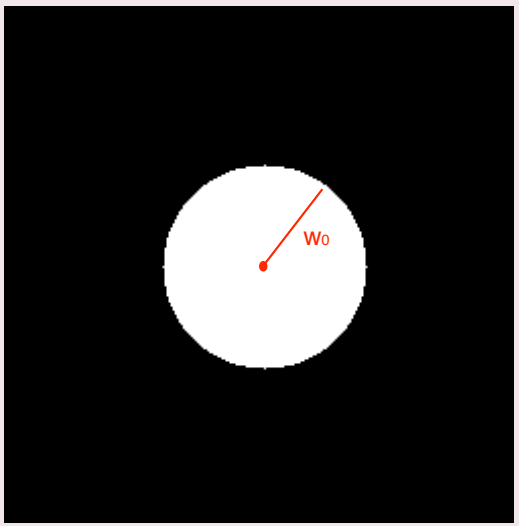
Ideal 2D Low-Pass Filter

The two dimensional version of this is the Low-Pass Filter:

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

where w_0 is now the **cut-off frequency** for **both** dimensions.

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



Not So Ideal Low-Pass Filter? (1)

In practice, the ideal Low-Pass Filter is no so ideal

The **problem** with this filter is that as well as noise there may be **useful** high frequency content:

- In **audio**: plenty of other high frequency content: high pitches, rustles, scrapes, wind, mechanical noises, cymbal crashes etc.
- In **images**: **edges** (places of rapid transition from light to dark) also significantly contribute to the high frequency components.

Choosing the **most appropriate** cut-off frequency is not so easy

- Similar problem to choosing a threshold in **image thresholding**.

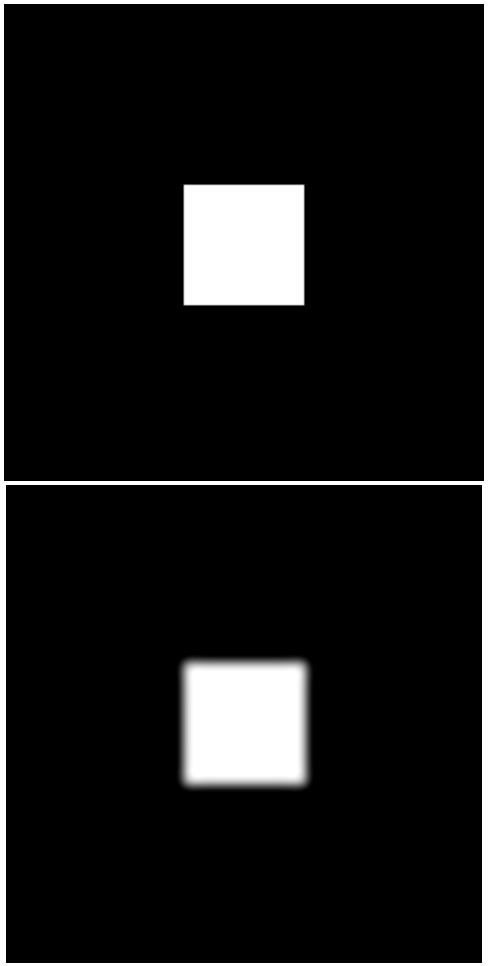
Not So Ideal Low-Pass Filter? (2)

What if you set the wrong value for the cut-off frequency?

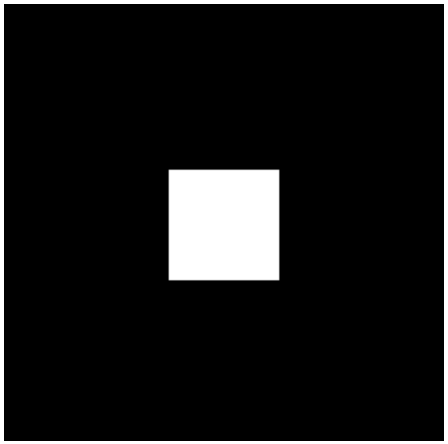
If you **choose the wrong cut-off frequency** an ideal low-pass filter will tend to *blur* the data:

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

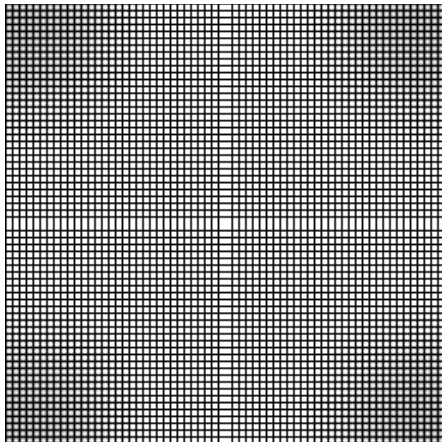
The lower the cut-off frequency is made, the more pronounced this effect becomes in *useful data content*



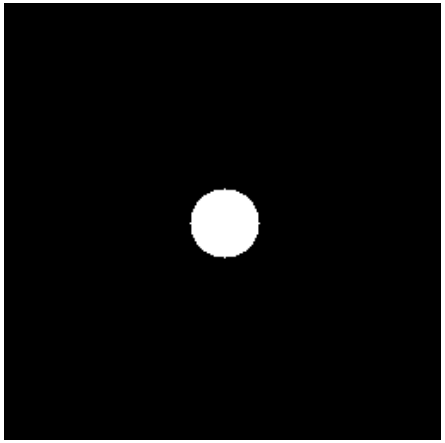
Ideal Low Pass Filter Example 1



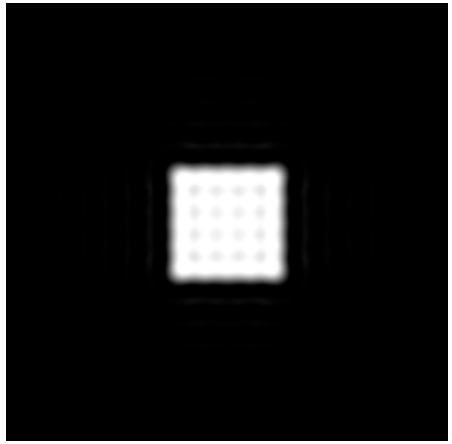
(a) Input Image



(b) Image Spectra



(c) Ideal Low Pass Filter



(d) Filtered Image

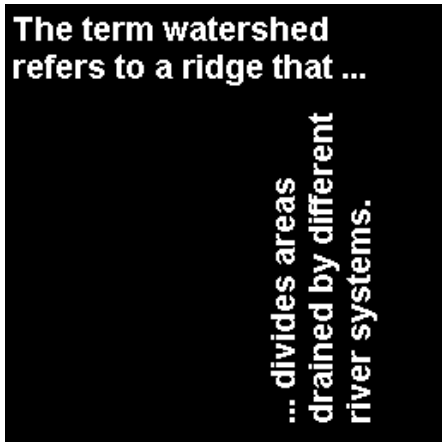
Ideal Low-Pass Filter Example 1 MATLAB Code

low pass.m:

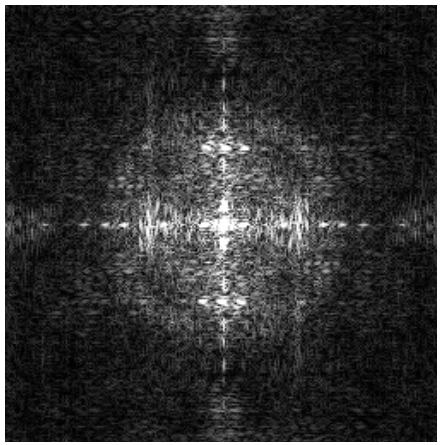
```
% Create a white box on a
% black background image
M = 256; N = 256;
image = zeros(M,N)
box = ones(64,64);
%box at centre
image(97:160,97:160) = box;

% Show Image

figure(1);
imshow(image);

% compute fft and display its spectra

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

```
% 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);
```

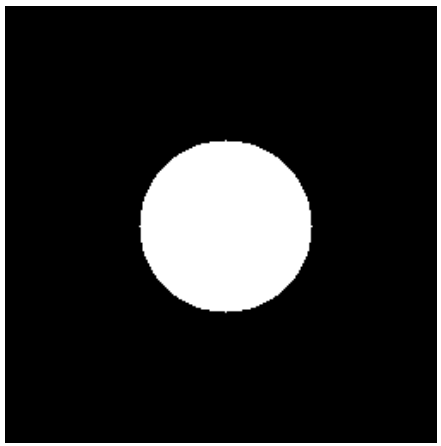
Ideal Low Pass Filter Example 2



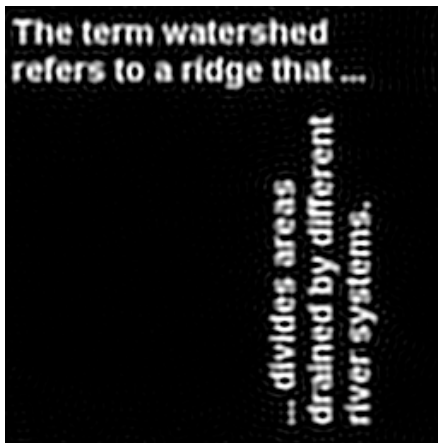
(a) Input Image



(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);
```

```
% 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);
```

Low-Pass Butterworth Filter (1)

We introduced the **Butterworth Filter** with **IIR/FIR Filters** (**Temporal Domain Filtering**). Let's now study it in more detail.

- Much easier to visualise in Frequency space

2D Low-Pass Butterworth Filter

Another popular (and general) filter is the **Butterworth low pass filter**.

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

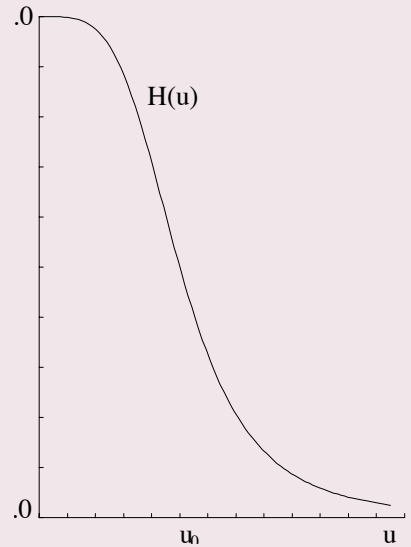
$$H(u, v) = \frac{1}{1 + \left[(u^2 + v^2)/w_0^2\right]^n},$$

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

Low-Pass Butterworth Filter (2)

Visualising the 1D Low-Pass Butterworth Filter

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



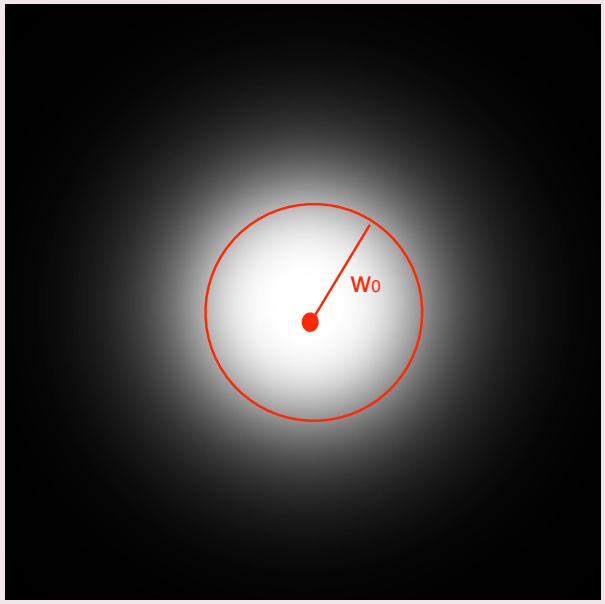
Consequently **reduces** the **blurring**.

- **Blurring** the **filter** — Butterworth is essentially a **smoothed** top hat functions — **reduces blurring by** the filter.

Low-Pass Butterworth Filter (3)

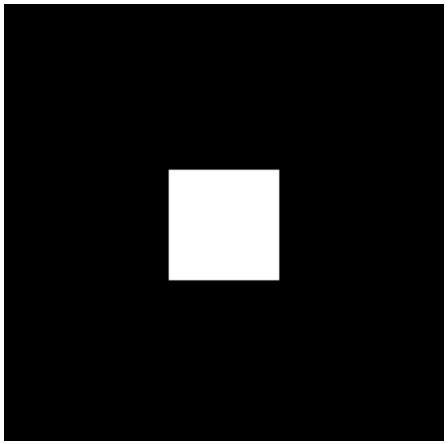
Visualising the 2D Low-Pass Butterworth Filter

The **2D second order** Butterworth filter looks like this:

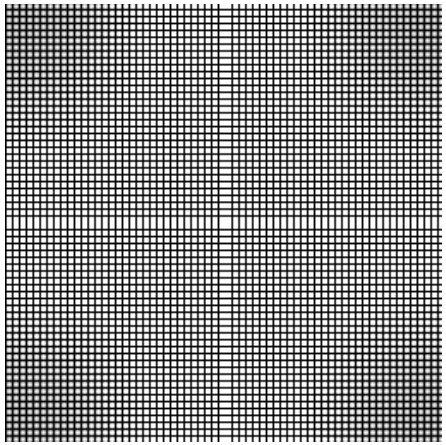


- Note this is **blurred circle** — blurring of the ideal 2D Low-Pass Filter.

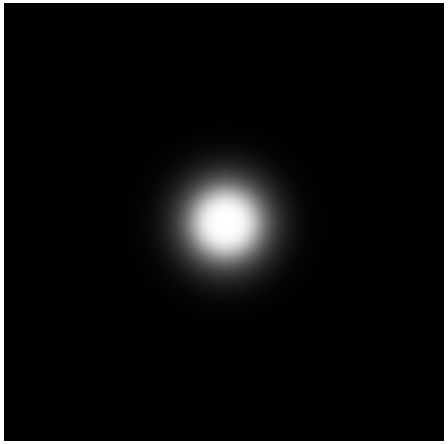
Butterworth Low Pass Filter Example 1 (1)



(a) Input Image



(b) Image Spectra



(c) Butterworth Low-Pass Filter

(d) Filtered Image

Butterworth Low-Pass Filter Example 1 (2)

Comparison of Ideal and Butterworth Low Pass Filter:

Ideal Low-Pass

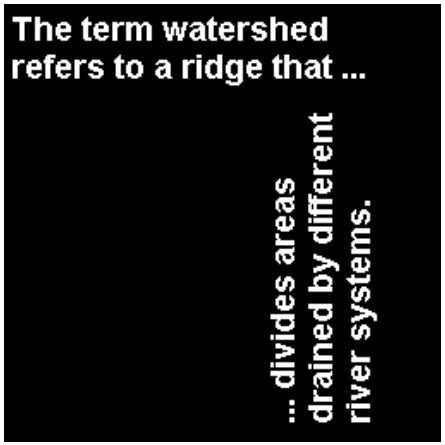
Butterworth Low-Pass

Butterworth Low-Pass Filter Example 1 (3)

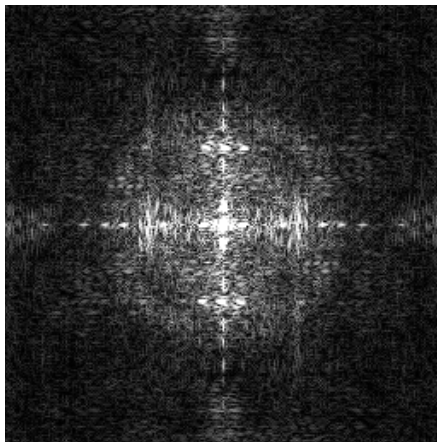
butterworth.m:

```
% 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
```

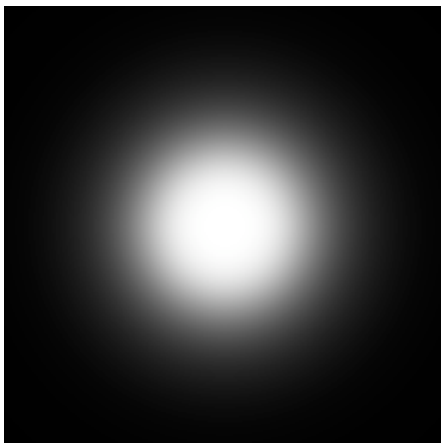
Butterworth Low-Pass Butterworth Filter Example 2 (1)



(a) Input Image



(b) Image Spectra



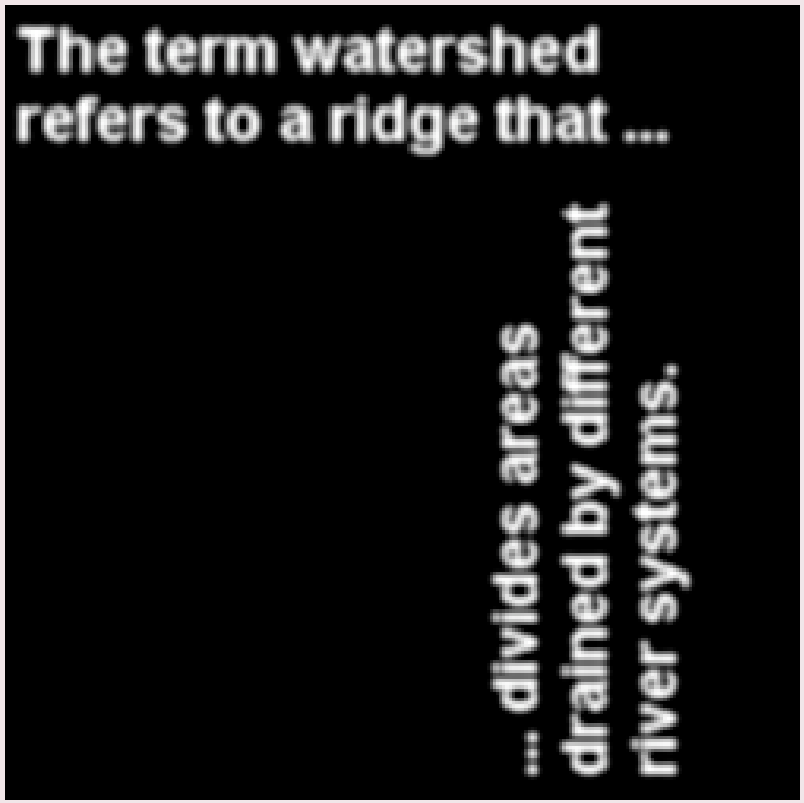
(c) Butterworth Low-Pass Filter

(d) Filtered Image

Butterworth Low-Pass Filter Example 2 (2)

Comparison of Ideal and Butterworth Low-Pass Filter:

Ideal Low Pass



Butterworth Low-Pass

Butterworth Low Pass Filter Example 2 MATLAB (3)

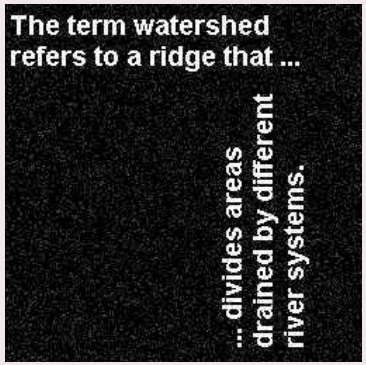
butterworth2.m:

```
% 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
```

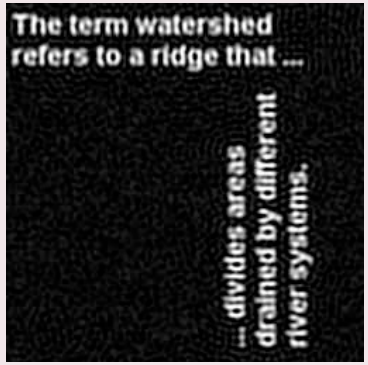
Low Pass Filtering Noisy Images

How to create noise and results of Low Pass Filtering

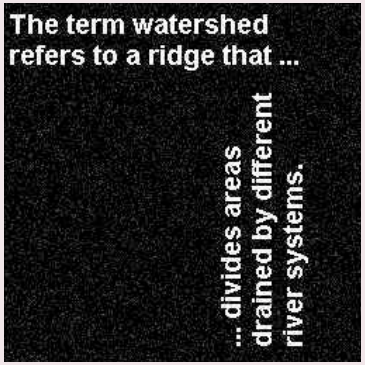
Use Matlab function, `imnoise()` to add noise to image (lowpass.m, lowpass2.m):



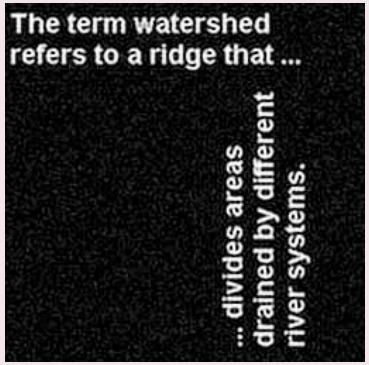
(a) Input Noisy Image



(b) Deconvolved Noisy Image (Low Cut Off)



(c) Input Noisy Image



(d) Deconvolved Noisy Image (Higher Cut Off)

Other Filters

Other Filters

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

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

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

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

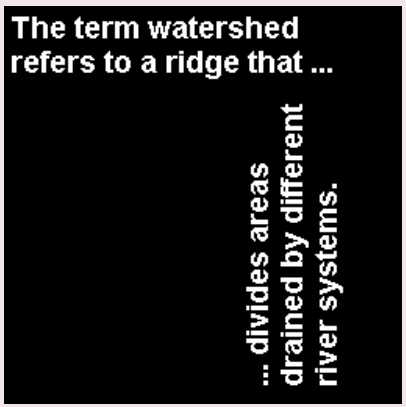
Resonator — amplify frequencies in a narrow bandwidth around cut-off frequency, u_0

Other filters exist that essentially are a combination/variation of the above

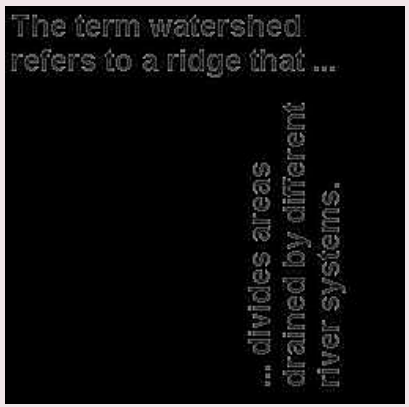
High Pass Filtering

Easy to Implement from the above Low Pass Filter

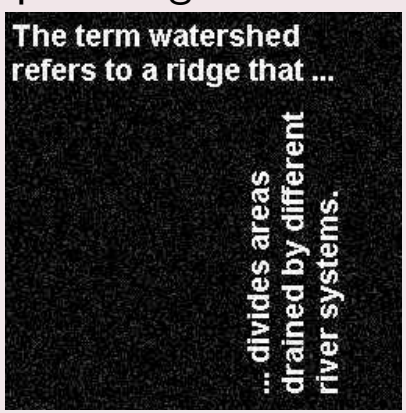
A High Pass Filter is usually defined as **1 - low pass** $= 1 - H$:



(a) Input Image



(b) High Pass Filtered Image



(c) Input Noisy Image

(d) High Pass Filtered Noisy Image

Convolution

Many Useful Applications of Convolution

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

- FIltering — In fact the **above Fourier filtering** is applying **convolutions** of a **low pass filter** where the equations are Fourier Transforms of real space equivalents.
- Deblurring — **high pass** filtering
- Reverb — impulse response convolution (**more soon**).

Note we have seen a discrete **real domain** example of Convolution with **Edge Detection**.

Formal Definition of 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

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

- $*$ is the mathematical **notation** for **convolution**.

No Fourier Transform in sight here — **but wait**!

1D Convolution Real Domain Example (1)

Convolution of Two Top Hat Functions

For example, let us take two **top hat functions**:

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

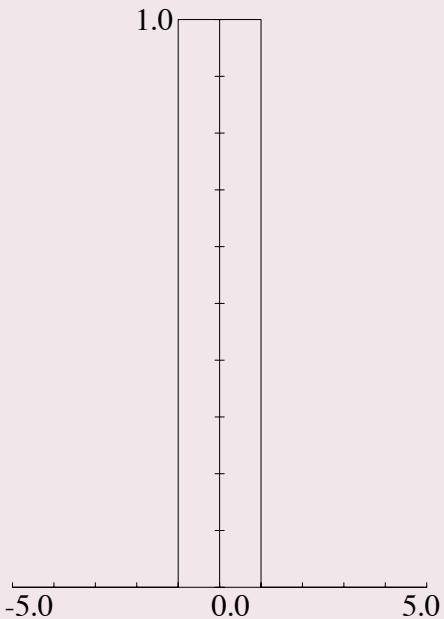
$$f(\alpha) = \begin{cases} 1 & \textbf{if } |\alpha| \leq 1 \\ 0 & \textbf{otherwise,} \end{cases}$$

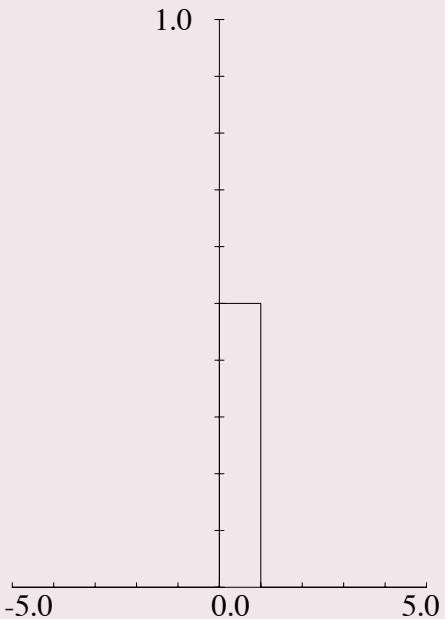
and let $g(\alpha)$ be as shown in next slide, defined by

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

1D Convolution Example (2)

Our Two Top Hat Functions Plots





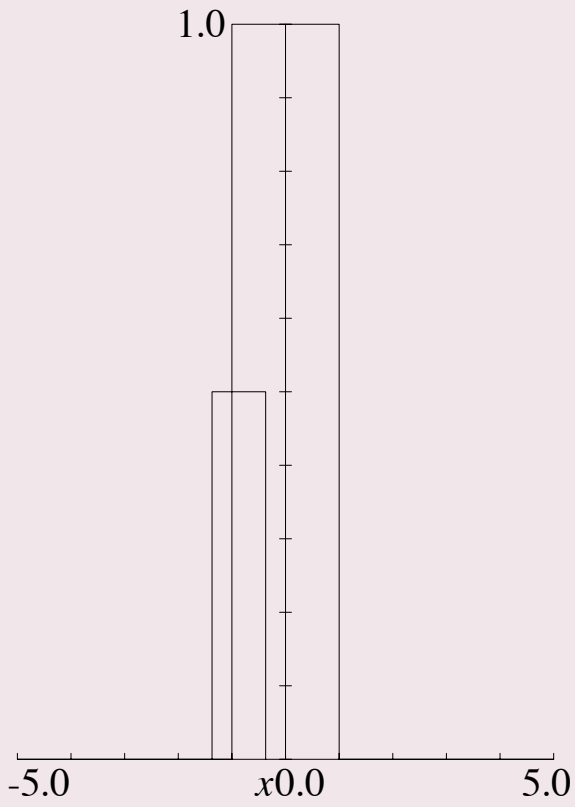
$$f(\alpha) = \begin{cases} 1 & \textbf{if } |\alpha| \leq 1 \\ 0 & \textbf{otherwise,} \end{cases}$$

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

1D Convolution Example (3)

The Convolution Process: Graphical Interpretation

- $g(-\alpha)$ is the **reflection** of this function in the **vertical** y-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 α 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



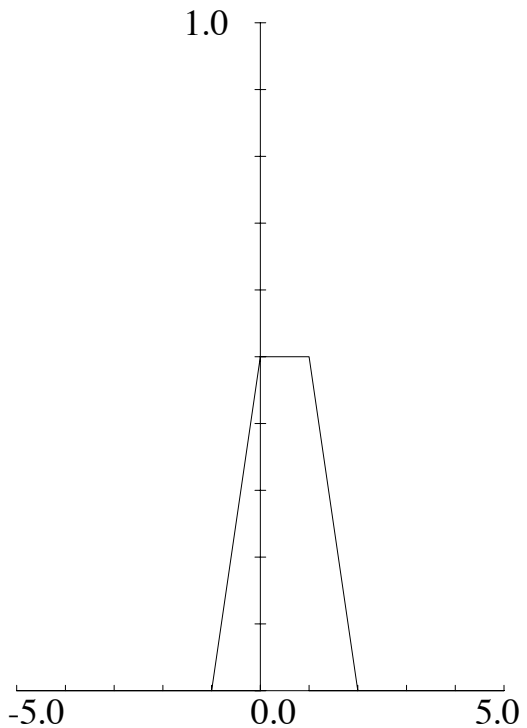
1D Convolution Example (4)

So the solution is:

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** from0 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

1D Convolution Example (5)

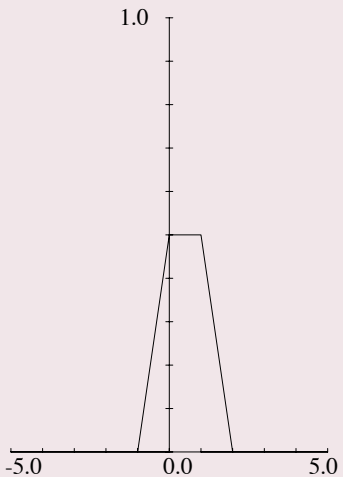


Result of $f(x) * g(x)$

1D Convolution Example (6)

Mathematically the convolution is expressed by:

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



Fourier Transforms and Convolution

Convolution Theorem: Convolution in Frequency Space is Easy

One **major** reason that Fourier transforms are so important in signal/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

Computing Convolutions with the Fourier Transform

Example Applications:

- To apply some reverb to an audio signal.
- To compensate for a less than ideal image capture system.

Deconvolution: Compensating for undesirable effects

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 **improved** audio image.

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

Image Deblurring Deconvolution Example

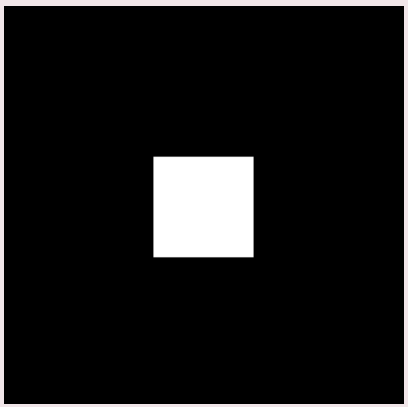
Inverting our Previous Low-Pass Filter

Recall our Low Pass (Butterworth) Filter example of a few slides ago: butterworth.m: deconv.m and deconv2.m reuses this code and adds a deconvolution stage:

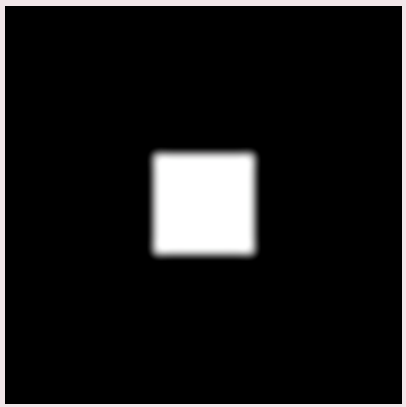
- Our computed butterworth low pass filter, H is our blurring function
- So to simply invert this we can divide (as opposed to multiply) by H with the blurred image G — effectively a **high pass filter**

```
Ghigh = G./H;
ghigh=real(ifft2(double(Ghigh)));
figure(5)
imshow(ghigh)
```

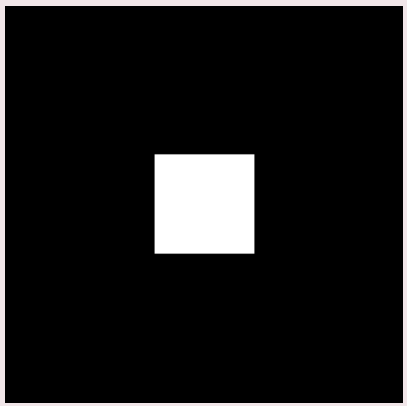
- In this ideal example we clearly get F back and to get the image simply to inverse Fourier Transfer.
- In the real world we don't really know the **exact blurring function** H so things are not so easy.



(a) Input Image

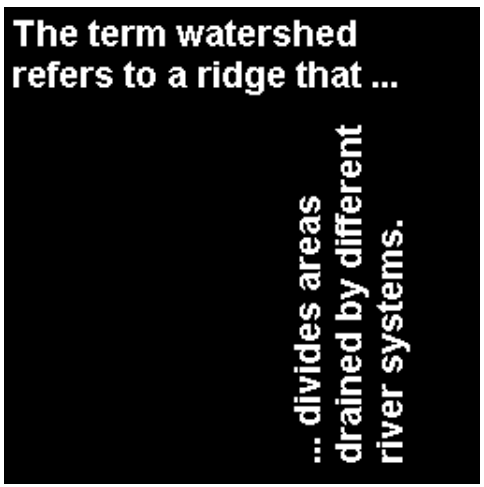


(b) Blurred Low-Pass Filtered Image

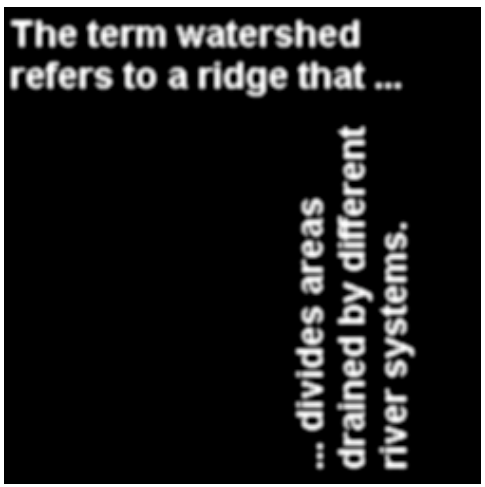


(c) Deconvolved Image

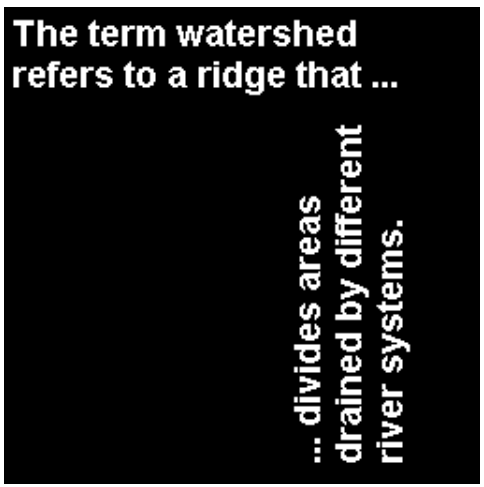
deconv2.m results



(a) Input Image

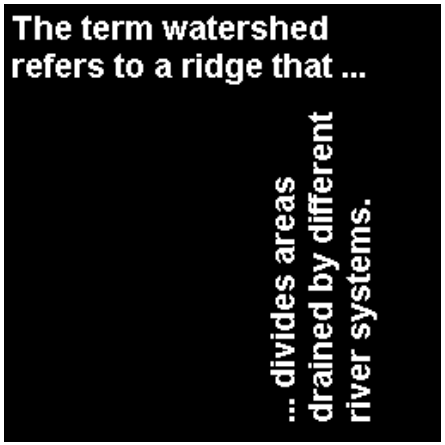


(b) Blurred Low-Pass Filtered Image

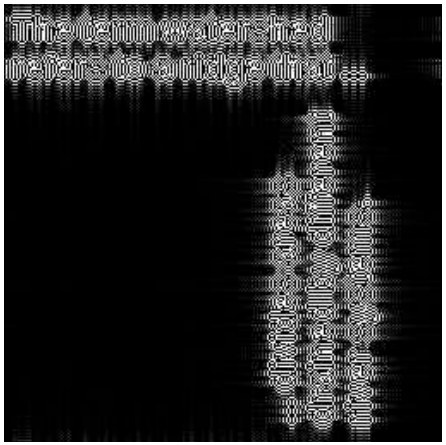


(c) Deconvolved Image

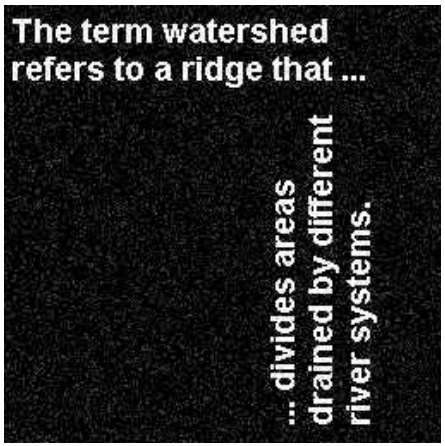
Deconvolution is not always that simple!



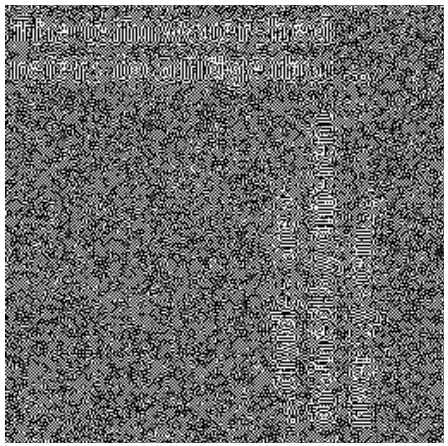
(a) Input Image



(b) Deconvolved Image



(c) Input Noisy Image



(d) Deconvolved Noisy Image