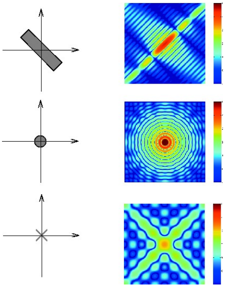
Lecture 3 : Fourier Transform

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**
  - **NoiseRemoval**
  - Signal/ImageAnalysis
  - Simple implementation of **Convolution**
  - **Audio** and Image **Effects Processing**.
  - Signal/Image Restoration—*e.g.* ***Deblurring***
  - Signal/Image Compression—**MPEG** (Audio and Video), **JPEG** user related techniques.
  - Many more ......



Introducing Frequency Space

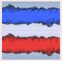
1D Audio Example

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

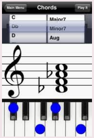
Consider a **complicated sound** such as a **chord** played on a **piano** or a **guitar**.

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**.



FrequencyDomain: **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

D	:	554.40Hz
F	:	698.48Hz
A	:	830.64Hz
C	:	1046.56Hz

plus harmonics/partial frequencies....

Back to Basics

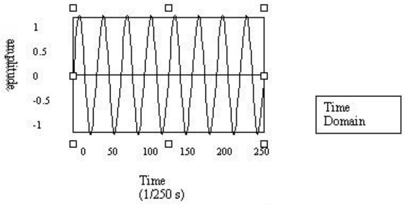
An 8Hz Sine Wave

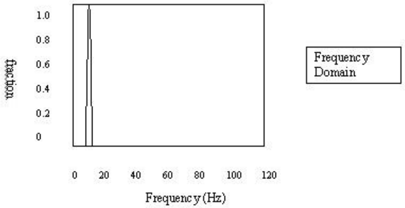
A signal that consists of a **sinusoidal** wave at **8Hz**.

- 8 Hz means that wave is completing 8 cycles in 1 second
- The **frequency** of that wave is 8Hz.

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

- **One peak** occurring with a frequency of 8Hz—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

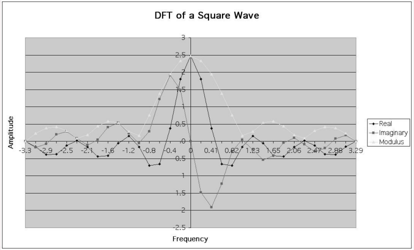
What do Frequencies in an Image Mean?

- 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

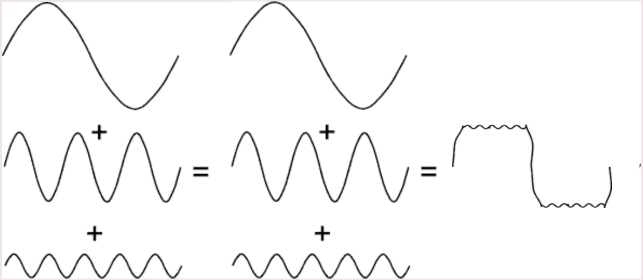
- **Any digital signal** (function) can be **decomposed** into purely **sinusoidal components**
    - Sine waves of different size/shape — varying **amplitude**, **frequency and phase**.
- When **added** back **together** they **reconstitute** the **original signal**
- The **Fourier transform** is the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



Summing Sine Waves.
Example: to give a Square(ish)Wave

Digital signals are composite signals made up of many sinusoidal frequencies

- A 200 Hz 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 50Hz and
  - Second part sinusoidal wave at 200Hz.
  - *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

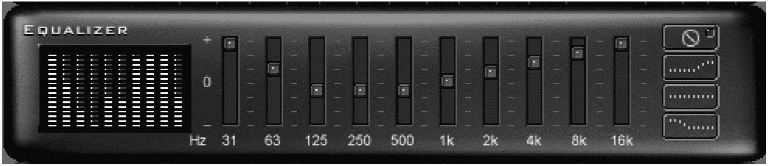
Filtering now involves *attenuating* or *removing* certain frequencies — *easily performed*:

- *Low-pass-filter* —
  - *Ignore high frequency* noise components—make *zero* or a *very low value*.
  - Onlystorelowerfrequencycomponents
- *High-pass filter—opposite of above*
- *Band-pass filter* — only *allow* frequencies in a *certain range*.

Visualising the Frequency Domain

Think Graphic Equaliser

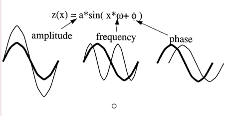
An easy way to visualise what is happening is to think of a graphic equaliser on a stereo system (or some software audio players, *e.g. iTunes*).



So are we ready for the FourierTransform?

We have all theTools....

- This lecture, so far, (hopefully) set the context for frequency decomposition. Also, remember
  - **Odd/EvenFunctions**: $\sin(-x)=-\sin(x)$, $\cos(-x)=\cos(x)$
  - **ComplexNumbers**: **Phasor Form** $re^{i\varphi} = r(\cos\varphi + i\sin\varphi)$
  - Calculus **Integration**: $e^{kx}dx = e^{kx}/k$
- Digital Signal Processing:
  - Basic Wave formTheory. 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** (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:*

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

Note: In general $F(z)$ 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 Theory**.
  - $e^{-2\pi ixz}$ *above is a **Phasor**.*

Inverse Fourier Transform

Inverse 1D Fourier Transform

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

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

which is rather similar to the (forward) Fourier transform

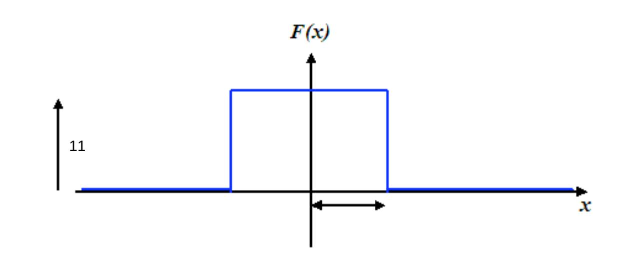
- except that the **exponential term has the opposite sign.**
- It is **not negative**

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

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



The Sinc Function (1)

We derive the Sinc function

So its Fourier transform is:

$$
\begin{aligned}
F(z) &= \int_{-\infty}^{\infty} f(x) e^{-2\pi ixz} \cdot dx \\
&= \int_{-1}^{1} 1 \times e^{-2\pi ixz} \cdot dx \\
&= \frac{-1}{2\pi iz} \left(e^{\pi iz} - e^{-\pi iz} \right)
\end{aligned}
$$

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

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

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

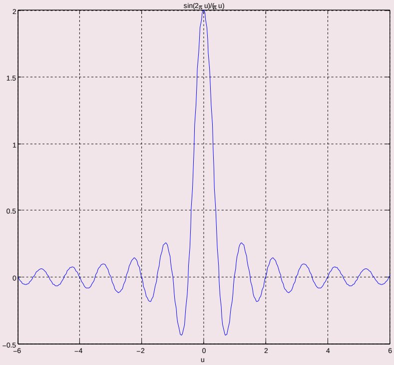
A graph of *F*(*z*) is shown overleaf.

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

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

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

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

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

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

while the inverse DFT is

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

NOTE: Minor changes from the continuous case area 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*

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

and

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

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**:

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

and

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

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.**

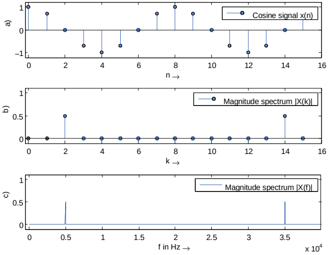
InverseDFT **ifft()**, **ifft2()**, **ifftn()** perform the **inverse** DFT.

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



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:**

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

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

The Phase Spectrum of Fourier Transform

The Phase Spectrum

Phase Spectrum

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

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

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 *fs* is the *sampling frequency* and N the **number** of samples.
- Thus we have **equidistant frequency steps** of f_s/N ranging from 0 Hz to $(N-1)f_s/N$ Hz

Time-Frequency Representation: Spectrogram

Spectrogram

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

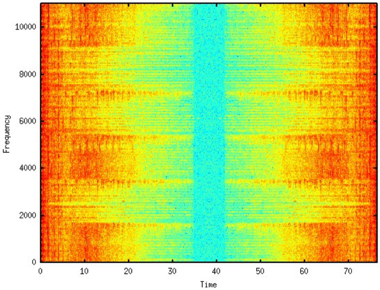
- Split signal into *N* segments
- Do a **windowed Fourier Transform** — **Short-Time FourierTransform** (**STFT**)
  - Window needed to reduce *leakage* effect of doing a shorter sample SFFT.
  - Apply a **Blackman**, **Hamming** or **Hanning** Window
- MATLAB function does the job: Spectrogram — see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

OCTAVE specgram Example

spectrogrameg.m

```
y = wavread('echoes.wav')
[N M]=size(y);
figure(1)
x = fft(y, N);
Fs=22050;
specgram(x,1024,Fs,1024,20);
```

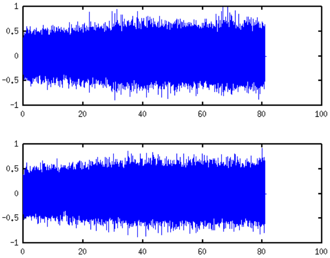
Produces the following:

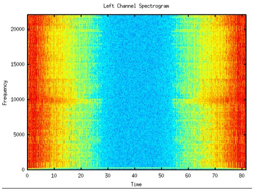


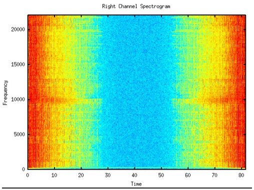
Another specgram Example

spectrogrameg2.m

```
[y, fs] = wavread('starWars.wav');
left = y(:,1);
[N1 M1]=size(left);
xl = fft(left, N1);
right = y(:,2);
[Nr Mr] = size(right);
xr = fft(right, Nr);
figure(1)
subplot(2,1,1), plot((1:length(left))/fs, left);
subplot(2,1,2), plot((1:length(right))/fs, right);
figure(2)
specgram(xl,1024,fs,1024,20);
title('Left Channel Spectrogram');
figure(3)
specgram(xr,1024,fs,1024,20);
title('Right Channel Spectrogram');
```



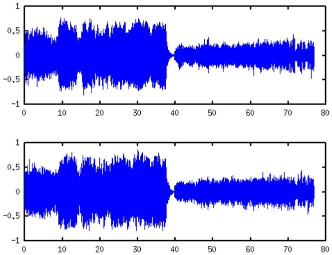


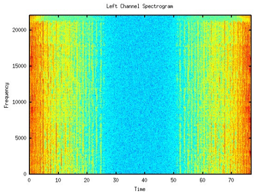


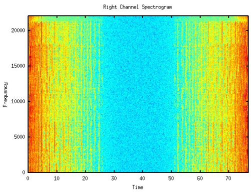
A new specgram Example

spectrogrameg3.m

```
[y, fs] = wavread('fuga.wav');
left = y(:,1);
[N1 M1]=size(left);
xl = fft(left, N1);
right = y(:,2);
[Nr Mr] = size(right);
xr = fft(right, Nr);
figure(1)
subplot(2,1,1), plot((1:length(left))/fs, left);
subplot(2,1,2), plot((1:length(right))/fs, right);
figure(2)
specgram(xl,1024,fs,1024,20);
title('Left Channel Spectrogram');
figure(3)
specgram(xr,1024,fs,1024,20);
title('Right Channel Spectrogram');
```







Filtering in the Frequency Domain

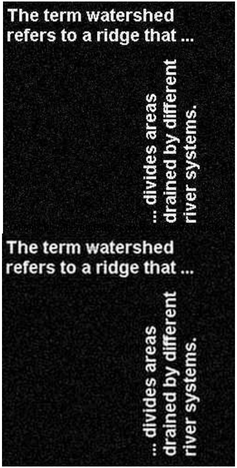
Low Pass Filter

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.

Image with Noise Added



Frequency Space Filtering Methods

Low Pass Filtering — Remove Noise

Noise = High Frequencies:

- In audio data many spurious peaks in over a short time scale.
- In an image means there are many rapid transitions (over a short distance) in intensity from high to low and back again or viceversa, 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:

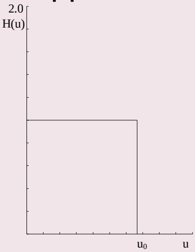
- $F(u,v)$ is the **Fourier transform** of the **original** image,
- $H(u,v)$ is a filter function, designed to reduce high frequencies, and
- $G(u,v)$ is the **Fourier transform of the improved image.**
- **Inverse Fourier transform** $G(u,v)$ to get $g(x,y)$ our **Improved image**

Ideal Low-Pass Filter

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

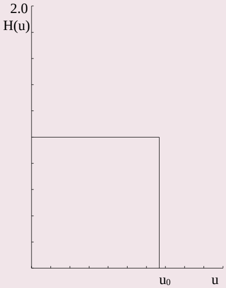
- If we know $h(x,y)$ or have a discrete sample of $h(x,y)$ can compute its FourierTransform
- Can simply design simple filters in Frequency Space

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



Ideal Low-Pass Filter

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-PassFilter

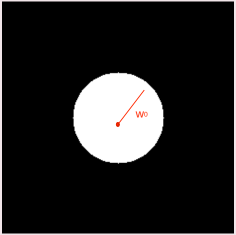
Ideal 2D Low-Pass Filter

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?

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 contents:

- In **audio**: plenty of other high frequency contents: 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**.

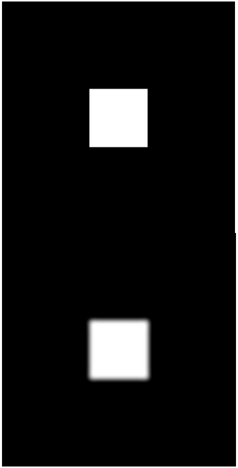
Not so ideal Low-PassFilter?

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

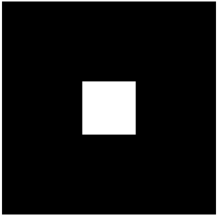
If you **choose the wrong cut-off frequency** an ideal low-pass filter will tend to *blur* the data:

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

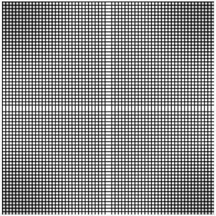
The lower the cut-off frequency is Made, the more pronounced this effect becomes in *useful data content*



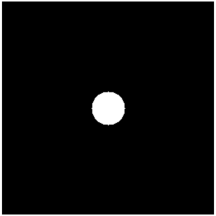
Ideal Low-Pass Filter Example



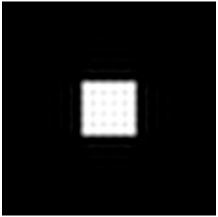
(a) Input Image



(b) Image Spectra



(c) Ideal Low-Pass Filter



(d) Filtered Image

Ideal Low-Pass Filter Example

lowpass.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;

%ShowImage

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 cutoff 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);
```