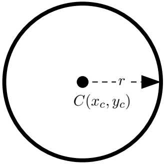
CM2202: Scientific Computing and Multimedia Applications
Lab Class Week 8

School of Computer Science & Informatics

Circles



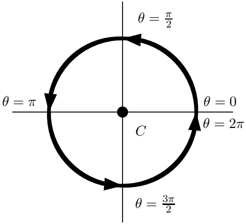
The implicit equation of a circle is the standard formula:

$$(x - x_c)^2 + (y - y_c)^2 - r^2 = 0$$

where the centre of the circle is $C(x_c, y_c)$ and r is the radius of the circle.

This form is commonly used for whole circles.

Circle (parametric form)



The parametric equation of a circle is given by:

$$\begin{aligned} x &= x_c + r\cos(\theta) \\ y &= y_c + r\sin(\theta) \end{aligned}$$

- Parameterisation in terms of angle subtended at the circle centre, C.

MATLAB Circle code

To create *n* points, p, equally space on cirlce of centre and radius, r:

Implicit form, circle_imp_points_2d:

```
for i = 1 : n
    theta = ( 2.0 * pi * ( i - 1 ) ) / n;
    p(1,i) = center(1) + r * cos ( theta );
    p(2,i) = center(2) + r * sin ( theta );
  end
```

Parametric form, is similar.

Fourier Transform 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.

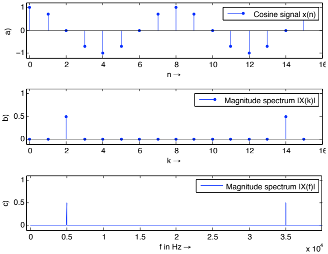
See also: **MALTAB Docs Image Processing → User's Guide → Transforms → Fourier Transform**

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**:

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

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

MATLAB Fourier Frequency Spectra Example

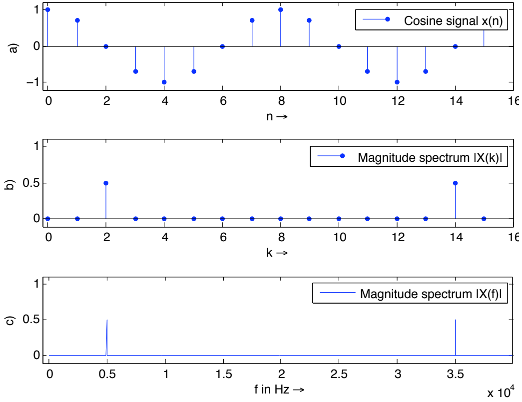
fourierspectraeg.m

```
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')
```

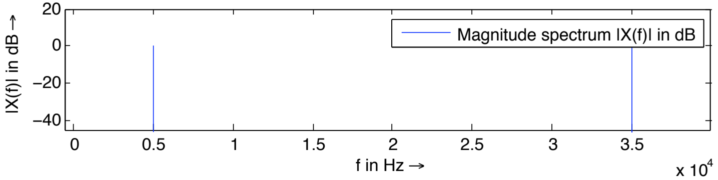
MATLAB Fourier Frequency Spectra Example Output

fourierspectraeg.m 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 fourierspectraeg.m)



Time-Frequency Representation: Spectrogram

Spectrogram

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

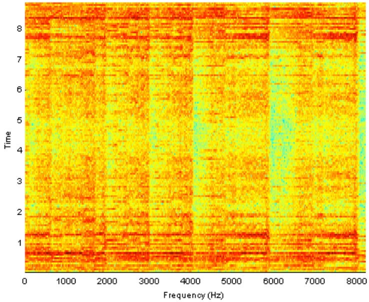
- Split signal into N segments
- Do a **windowed Fourier Transform** — **Short-Time Fourier Transform** (**STFT**)
  - Window needed to reduce *leakage* effect of doing a shorter sample SFFT.
  - Apply a **Blackman**, **Hamming** or **Hanning** Window
- MATLAB function does the job: `Spectrogram` — see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

MATLAB spectrogram Example

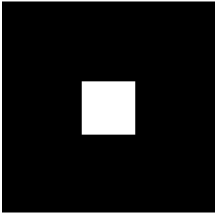
spectrogrameg.m

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

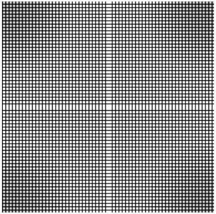
Produces the following:



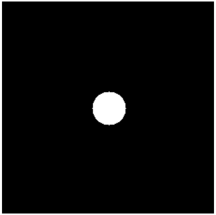
Ideal Low Pass Filter Example 1



(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);
```

Butterworth Low-Pass Filter Example 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
```

MATLAB Convolution Reverb (1)

Let's develop a fast convolution routine: fconv.m

```
function [y]=fconv(x, h)
%   FCONV Fast Convolution
%   [y] = FCONV(x, h) convolves x and h,
%         and normalizes the output   to +-1.
%       x = input vector
%       h = input vector
%

Ly=length(x)+length(h)-1;  %
Ly2=pow2(nextpow2(Ly));    % Find smallest power of 2 that is > Ly
X=fft(x, Ly2);             % Fast Fourier transform
H=fft(h, Ly2);             % Fast Fourier transform
Y=X.*H;                    % DO CONVOLUTION
y=real(ifft(Y, Ly2));      % Inverse fast Fourier transform
y=y(1:1:Ly);               % Take just the first N elements
y=y/max(abs(y));           % Normalize the output
```

See also: MATLAB built in function `conv()`

MATLAB Convolution Reverb (2)

reverb_convolution_eg.m

```
% reverb_convolution_eg.m
% Script to call implement Convolution Reverb

% read the sample waveform
filename='../acoustic.wav';
[x,Fs,bits] = wavread(filename);

% read the impulse response waveform
filename='impulse_room.wav';
[imp,Fsimp,bitsimp] = wavread(filename);

% Do convolution with FFT
y = fconv(x,imp);

% write output
wavwrite(y,Fs,bits,'out_IRreverb.wav');
```