WINDOWING AND SHORT TIME FOURIER TRANSFORM

Michele Buccoli

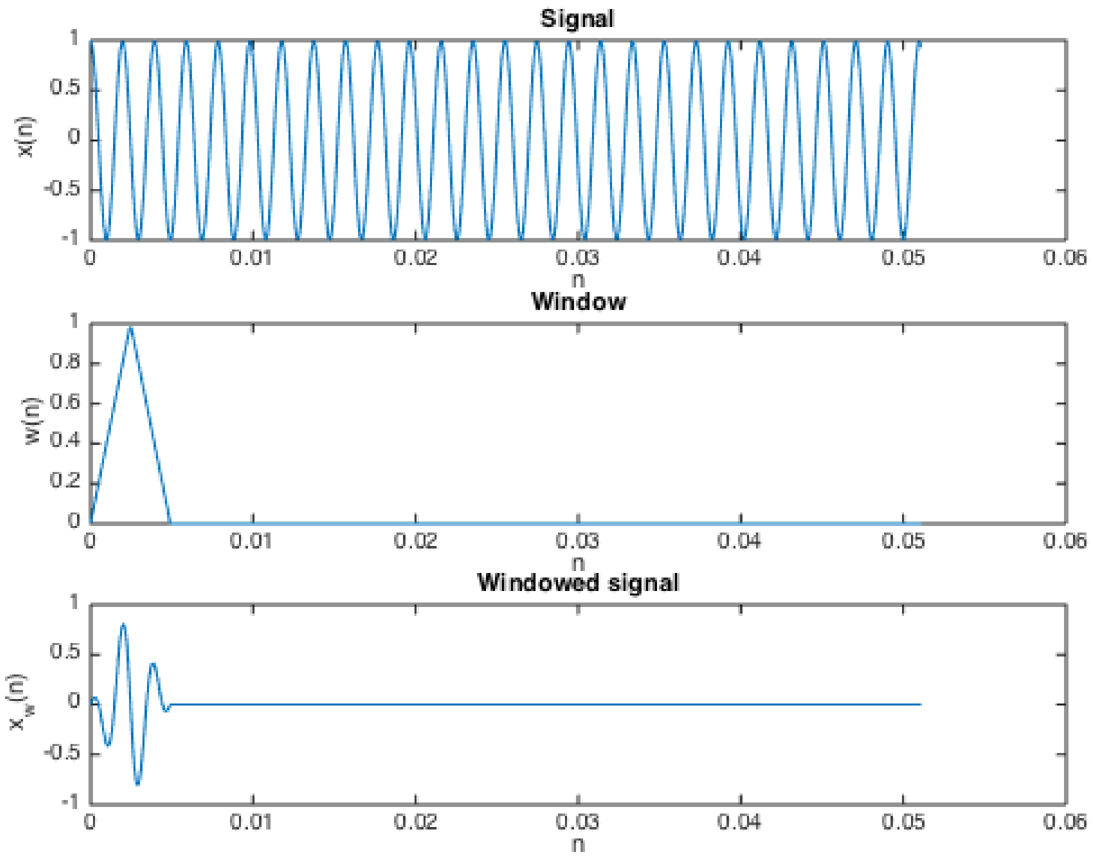
Windowing

- DFT is computed on finite-duration signal
- We may need to operate with infinite-duration signals:
  - Real time operation
  - Long time duration
- Windows are used to convert infinite duration signal to a set of finite duration signals
  - Dividing signals into blocks and multiplying it by a window signal
  - `window(@WNAME,N)` returns an N-point window of type `@WNAME`
    - column vector
- Let's window a signal and compute their DFT

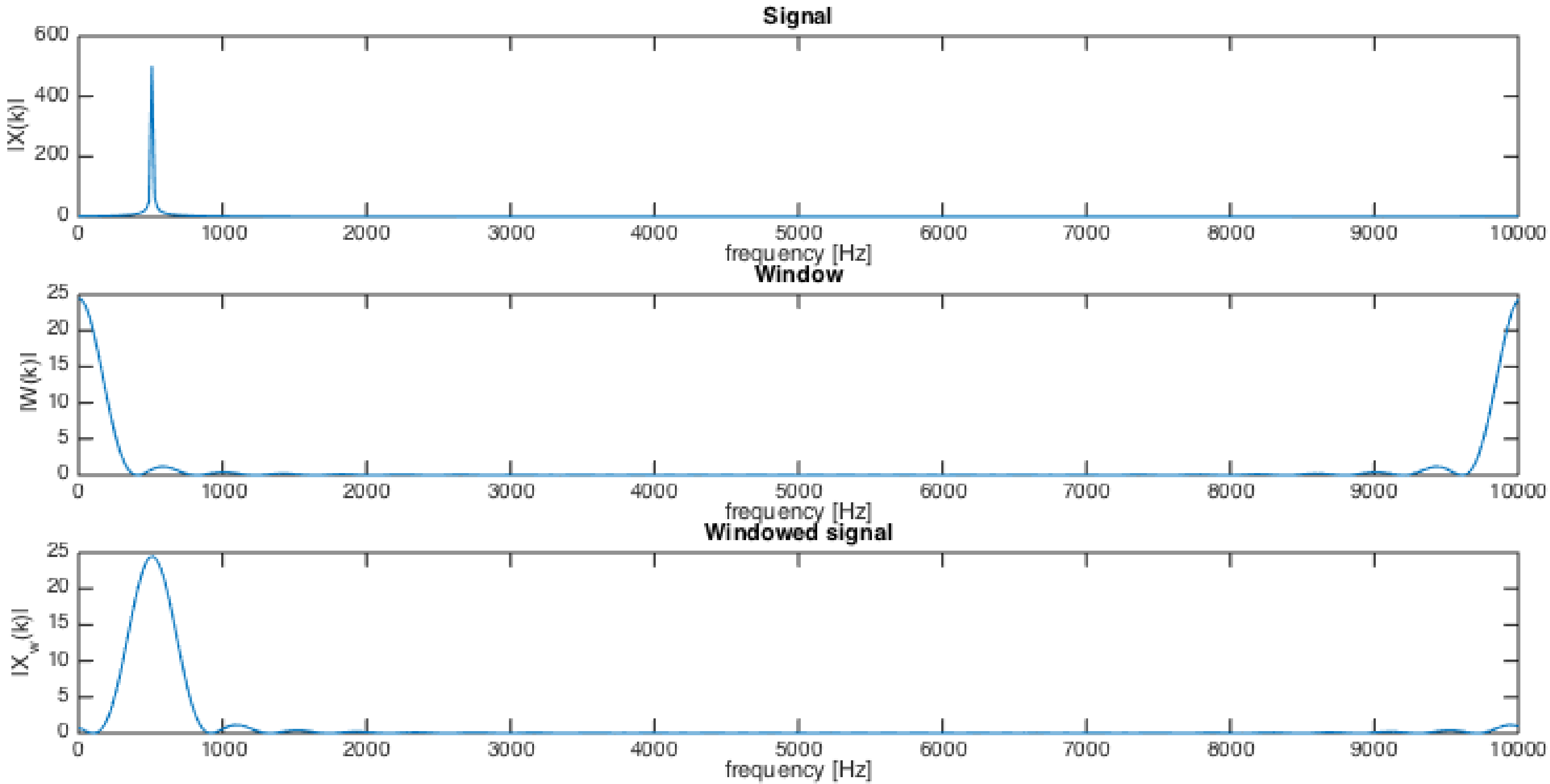
Windowing


```
Fs=10000; f=510; N=512; n=(0:N-1)/Fs;
x=exp(1i*2*pi*f*n); x=x.';
w=window(@bartlett,50);
w(N)=0; % zero padding

xw=x.*w;

X=fft(x,N); % = delta(f-510)
W=fft(w,N);
XW=fft(xw,N); %= conv(W, delta(f-510))
F=linspace(0,Fs,N);
```

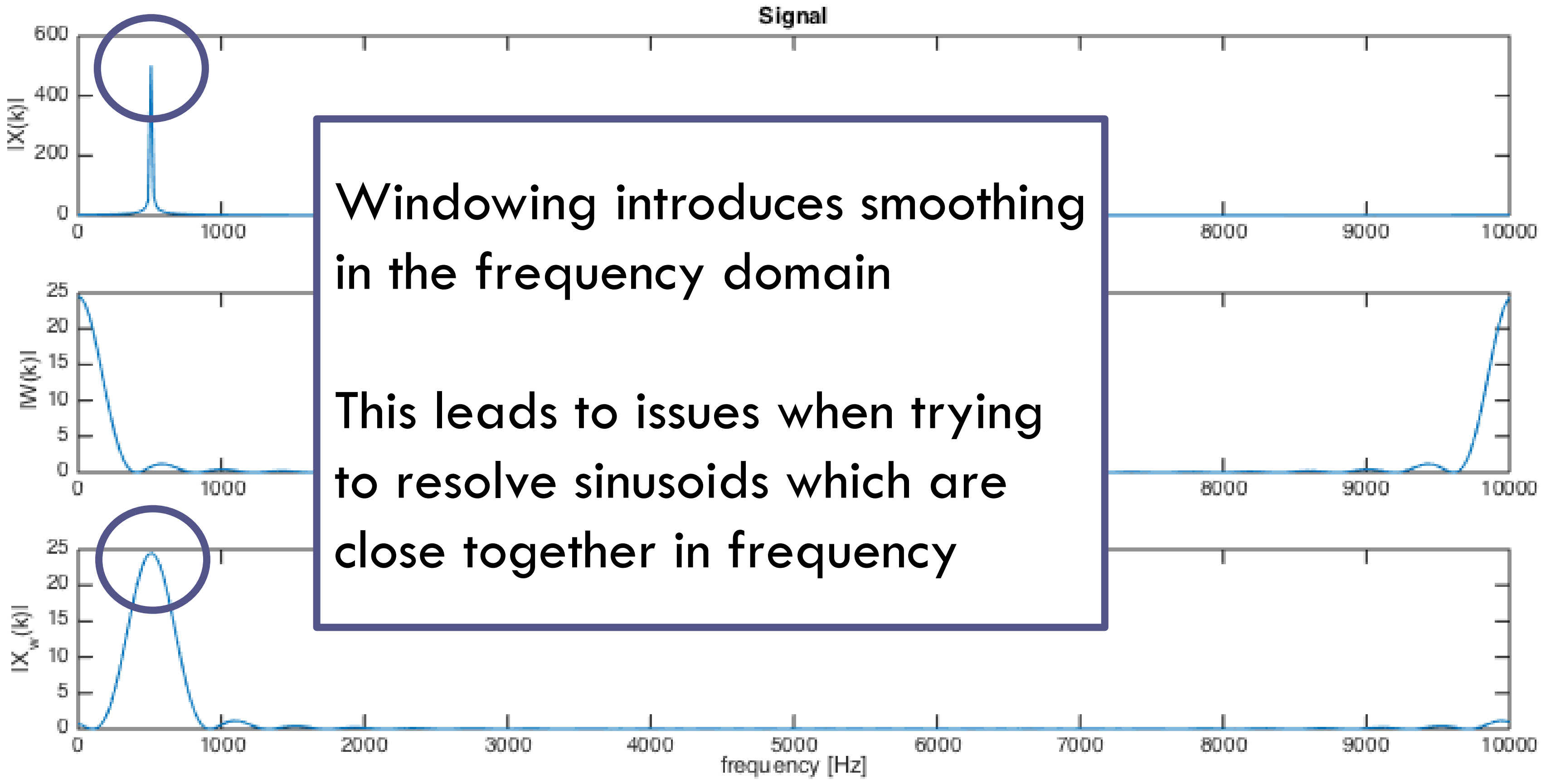
Windowing


Signal
x(n)
1
0.5
0
-0.5
-1
0
0.01
0.02
0.03
0.04
0.05
0.06
n
Window
w(n)
1
0.8
0.6
0.4
0.2
0
Windowed signal
$x_w(n)$

Windowing


Signal
600
400
200
0
|X(k)|
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
frequency [Hz]
Window
25
20
15
10
5
0
|W(k)|
frequency [Hz]
Windowed signal
|X_w(k)|
frequency [Hz]

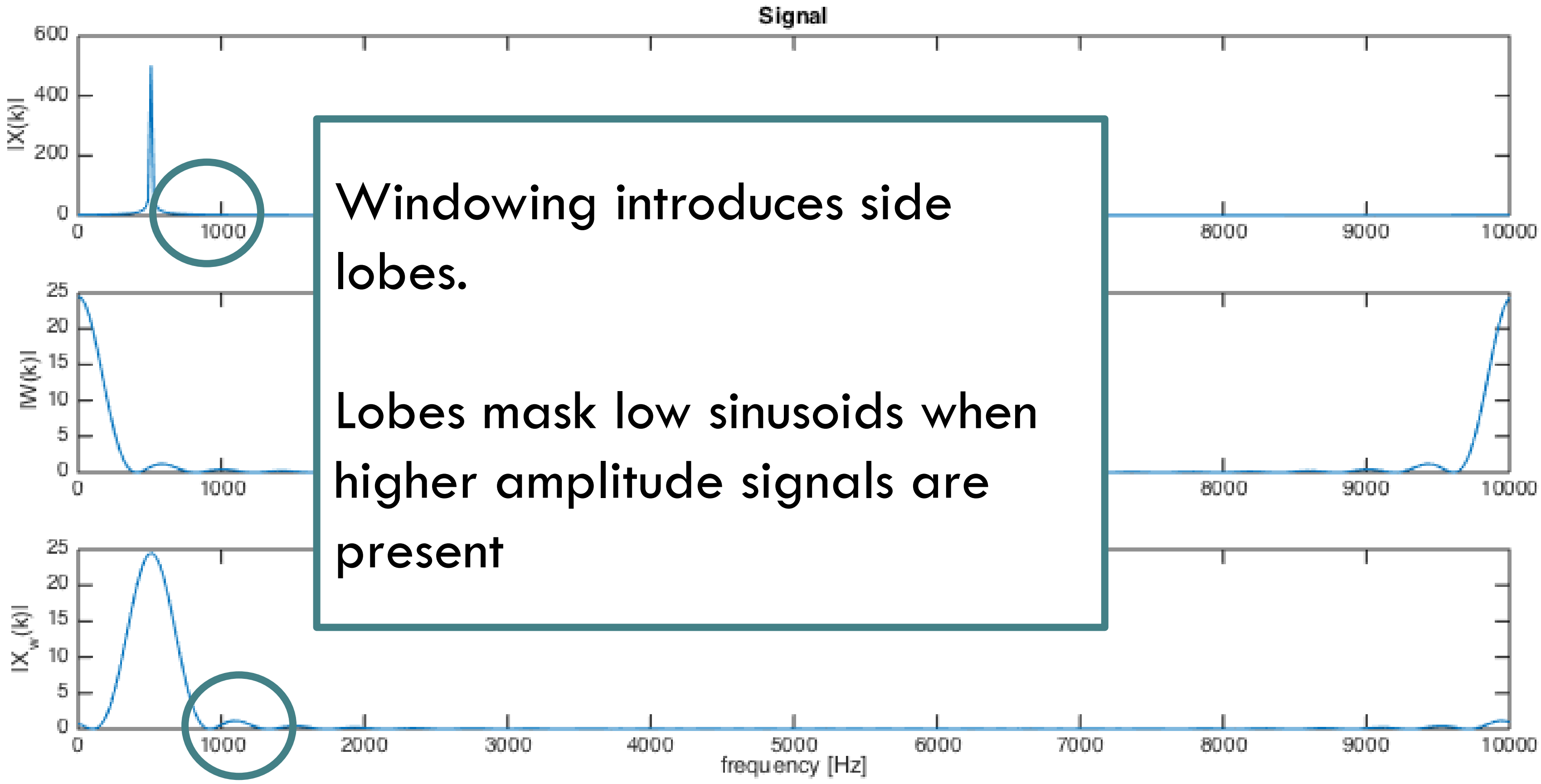
Windowing



Windowing introduces smoothing in the frequency domain

This leads to issues when trying to resolve sinusoids which are close together in frequency

Windowing



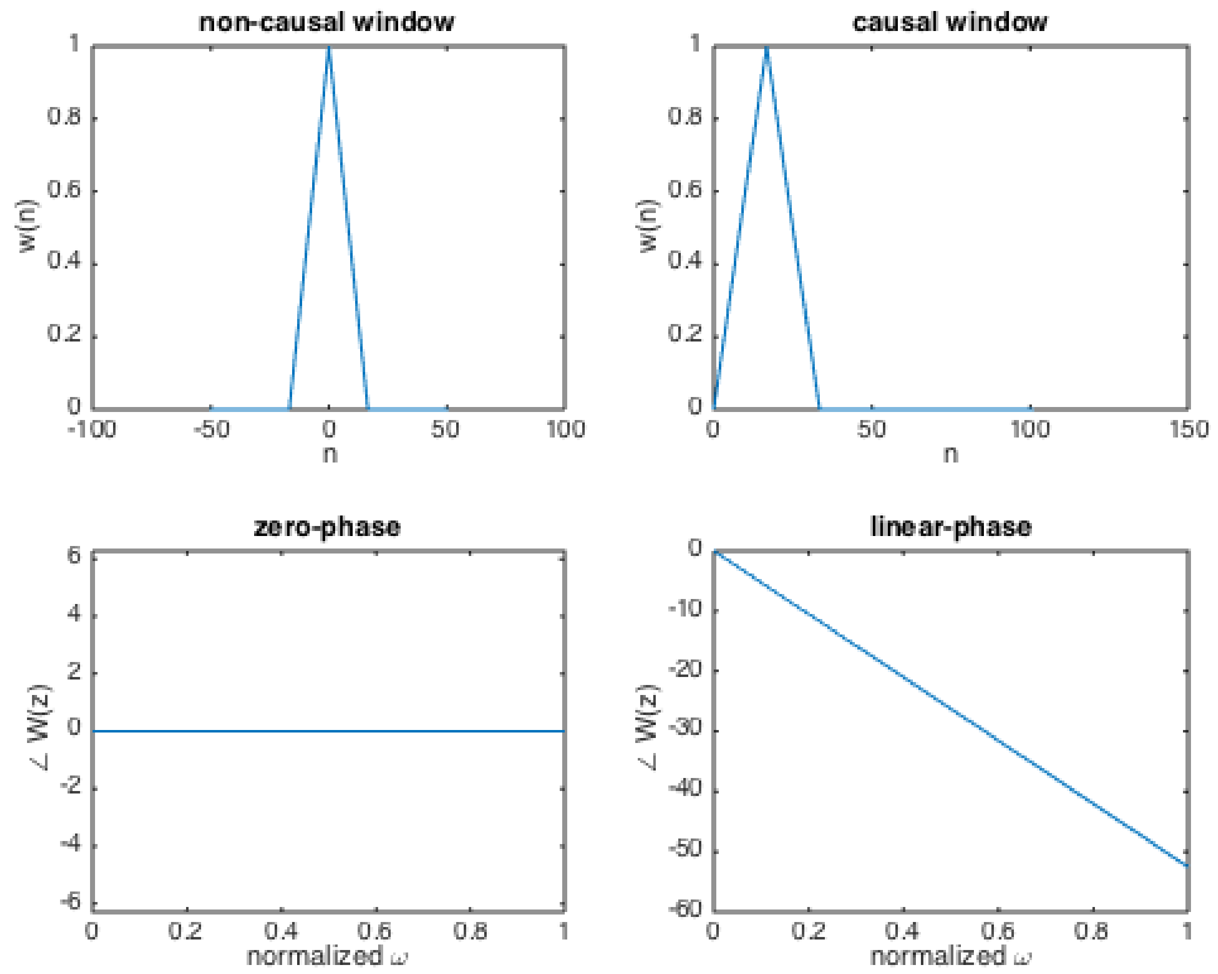
Windowing introduces side lobes.

Lobes mask low sinusoids when higher amplitude signals are present

Causal windows

- A window is usually a real and even signal in the time domain
- Therefore, it is real and even in the frequency domain
  - zero-phase signal
- For real time processing, we need that the window is causal,
  - $w(n)=0$ if $n<0$
  - the window is shifted by half of its length
  - a linear phase term is consequently introduced

Causal windows



Types of windows

- The type and parameters of windows must be properly chosen to fulfill our requirements
- Matlab provides
  - `w=window(@wname, N, opt)`
    - Computes the N-points *wname* window
    - *opt* are the options that the window may need
  - `window`
    - Window Design and Analysis Tool
  - `wtool(w1,w2,w3)`
    - Graphical User Interface Tool to analyze and compare windows

Types of windows

- From requirements to choice of window's type and parameters
- Properties
  - **Leakage factor**: ratio of power in the side lobes to the total window power
  - **Sidelobe attenuation**: difference between the value of the main lobe peak and of the highest side lobe peak
  - **Mainlobe width (-3dB)**: bandwidth where amplitude is -3dB with respect to the mainlobe peak
  - **Roll-off**: how much fast the sidelobes' energy decreases
- Write a function `show_win(win, N, name)` that given a window win, it shows it in the time and N-point frequency domain (log magnitude)

Windowing


```
function [   ] = show_win(win, N, name)
  M=length(win); win(N)=0; n=0:N-1;
  figure; subplot(1,2,1); plot(n, win);
  xlabel('n'); ylabel(['w_{', name,'}']);
  title([name ' in the time domain']);
  WIN= 20*log10(abs(fft(win)));
  subplot(1,2,2); plot(n,WIN); xlabel('k');
  ylabel(['|W_{', name,'}|  [dB]']);
  title([name ' in the frequency domain']);
  xlim([0, N/2+1]);
end
% note that [] stacks strings in matlab
```

Types of window - Rectangular

- $w_R(n) = \left\{ \frac{1}{M} : n \in [0, M-1]; 0 \text{ otherwise} \right\}$
- $W_R(k) = C \ \mathrm{sinc}_M(\omega\, T)$
  - zero crossing at multiple of N/M
- Main lobe width: 2 Ω_M
  - $\Omega_M = 2\pi/M$
- sidelobe attenutation: -13 dB

```
rect_win=window(@rectwin,N_win);
show_win(rect_win, N, 'rect');
```

Types of window – Bartlett (triangular)

- $$w_T(n) = \begin{cases} \dfrac{2n}{M-1} & 0 \le n \le \dfrac{M-1}{2} \\ 2 - \dfrac{2n}{M-1} & \dfrac{M-1}{2} \le n \le M-1 \\ 0 & otherwise \end{cases}$$
- Main lobe width: 4 Ω_M;
- side lobe attenuation: -27 dB

```
bart_win=window(@bartlett,N_win);
show_win(bart_win, N, 'bart');
```

Types of window – Hann

- $$w_{HANN}(n) = w_R(n)\left[\frac{1}{2} + \frac{1}{2}\cos(\Omega_M n)\right] = w_R(n)\cos^2\left(\frac{\Omega_M}{2} n\right)$$
- Main Lobe width: $4\Omega_M$

```
hann_win=window(@hann,N_win);
show_win(hann_win, N, 'Hann');
```

Types of window – Hamming

- $$w_H(n) = w_R(n)\left[0.54 + 0.46\cos(\Omega_M n)\right]$$

- Hide sidelobe attenuation!

```
hamming_win=window(@hamming, N_win);
show_win(hamming_win, N, 'Hamming');
```

Types of window – Blackman-Harris

$$w_{BH}(n) = w_R(n)\sum_{l=0}^{L-1}\alpha_l \cos(l\Omega_M n)$$

- L=1: rectangular
- L=2: generalized Hamming
- L=3: Blackman (α_0 =0.42; α_1=0.5; α_2=0.08)

```
black_win=window(@blackman, N_win);
show_win(black_win, N, 'Blackman');
```

Types of window – Kaiser

$$w_K(n) = \begin{cases} \dfrac{I_0\left(\beta\sqrt{1-\left(\dfrac{n}{M/2}\right)}\right)}{I_0(\beta)} & -\dfrac{M-1}{2} \le n \le \dfrac{M-1}{2} \\ 0 \quad elsewhere & \end{cases}$$

- I_0 is a Bessel function of the first kind
- Maximizes the energy in the main lobe

```
kaiser_win=window(@kaiser, N_win);
show_win(kaiser_win, N, 'Kaiser');
```

Types of window – Gaussian

- $$w_G(n) = e^{\frac{-t^2}{2\sigma^2}} \qquad W_G(\omega) = \sqrt{2\pi\sigma^2}\, e^{\frac{-\omega^2\sigma^2}{2}}$$

  - Infinite duration window truncated at M

```
gauss_win=window(@gausswin, N_win);
show_win(gauss_win, N, 'Gaussian');
```

Types of windows – sum up

Properties of the most frequently used windows (M=64)

Window	Main lobe width	Side lobe level	Roll Off (dB/decade)
Rectangular	$2\ \Omega_M$	-13.3 dB	-6
Hann	$4\ \Omega_M$	-31.5 dB	-18
Hamming	$4\ \Omega_M$	- 42.7 dB	-6
Blackman	$6\ \Omega_M$	- 58.1 dB	-18

$$\Omega_M = 2\pi/M \text{ [rad/sample]}$$

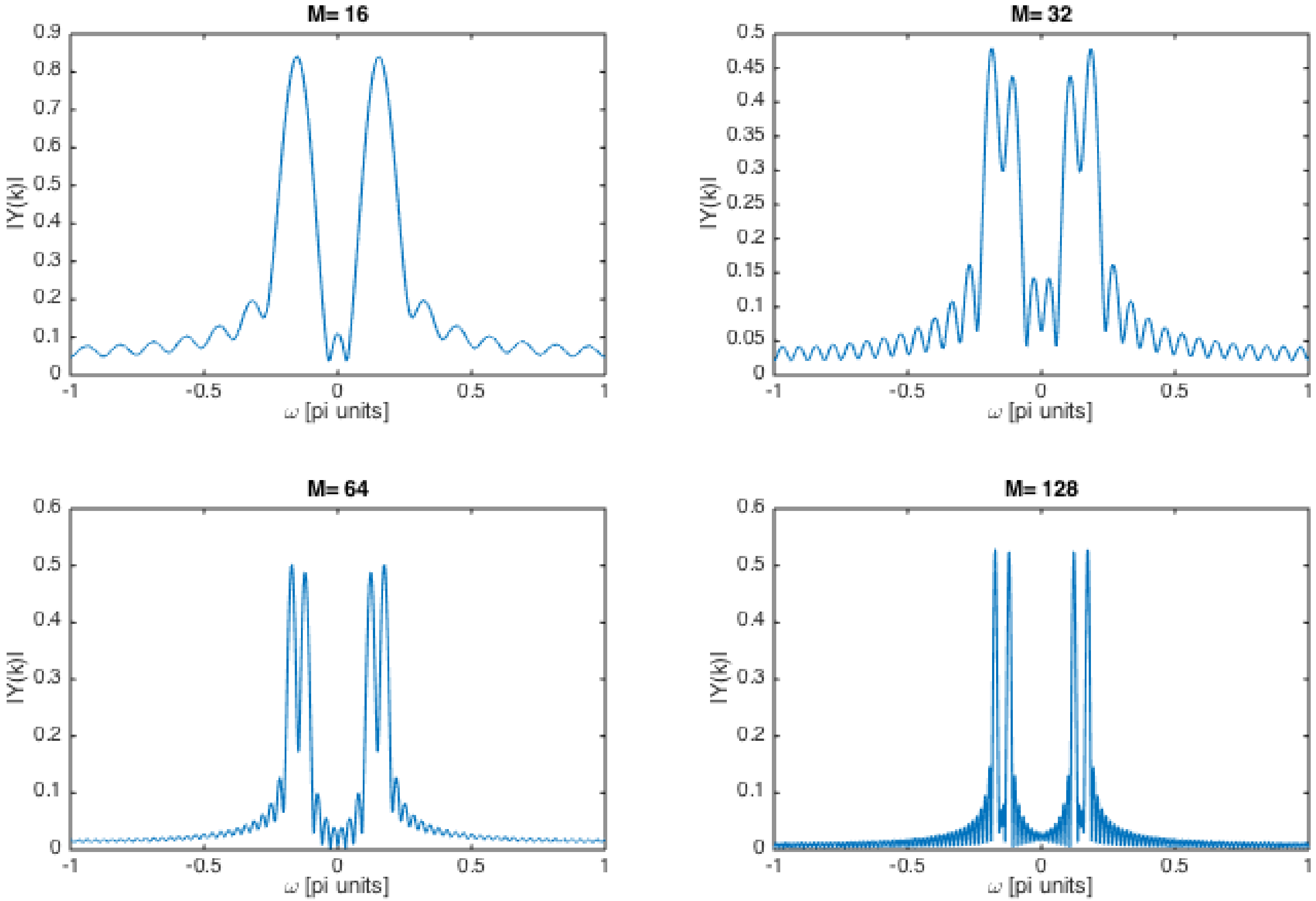
Exercise on windows

- Given a signal $x=\cos(\omega_1 n)+\cos(\omega_2 n)$
  - `omega1=2*pi*1000/N;`
  - `omega2=omega1 + 2*pi/40;`
- Define $\Delta_\omega=|\omega_1-\omega_2|$
- We use a rectangular window with length M
- See how the DFT changes for changing M
  - `M=[16, 32, 64, 128];`

Exercise on windows


```
N=2^14; n=(0:N-1).'; M=[16, 32, 64, 128];
w_norm=linspace(-1,1,N);
omega1=2*pi*1000/N;
omega2=omega1 + 2*pi/40;
x=cos(omega1*n)+cos(omega2*n);
figure;
for m =1:length(M);
    subplot(2,2,m); M_i=M(m);
    w_R=window(@rectwin,M_i)/M_i;
    w_R(N)=0; y=x.*w_R; Y=fft(y,N);
    Y=fftshift(Y); %centered in 0
    plot(w_norm, abs(Y)); % labels...
End
```

Exercise on windows



Choosing M to distinguish sinusoids

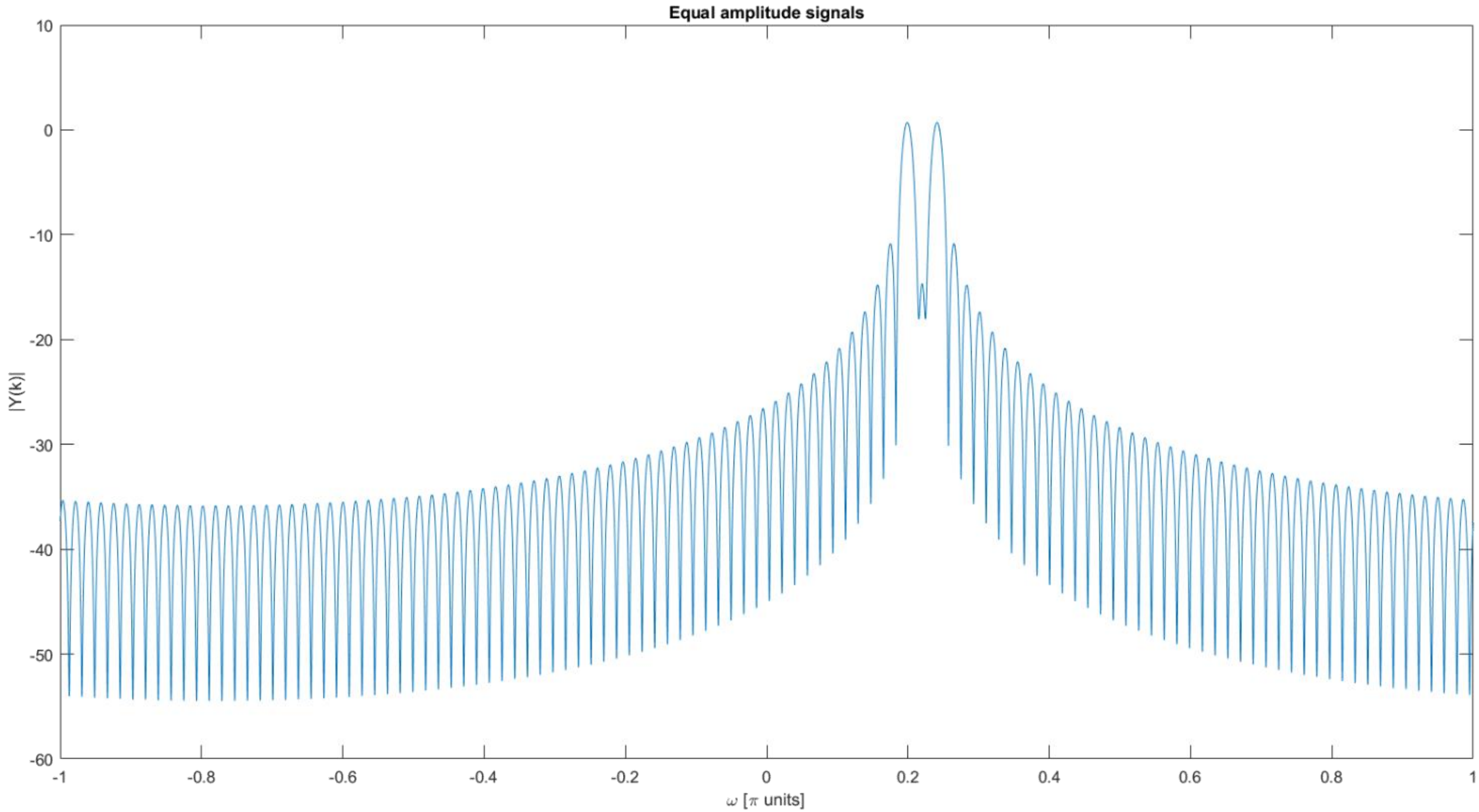
- Main lobe width B_w decreases when M increases
- Which is the lowest M that can be used to distinguish two sinusoids?
- $\Delta_\omega = |\omega_1 - \omega_2| \geq B_w = 2L\ 2\pi/M$
  - L=1 for rectangular windows
  - L=2 for Hann and Hamming windows
  - L=3 for Blackman windows
- $M \geq 2\ L\ 2\ \pi\ /\ \Delta_\omega$

- Let's try with a rectangular window: L=1;
  - `x=exp(1i*omega1*n)+exp(1i*omega2*n);`
  - `omega1=0.2*pi; omega2=0.24*pi;`

Choosing M to distinguish sinusoids


```
N=2^14; n=(0:N-1).';
w_norm=linspace(-1,1,N);
omega1=0.2*pi; omega2=0.24*pi;
x=exp(1i*omega1*n)+exp(1i*omega2*n);
Delta=abs(omega1-omega2);
L=2; M=2*L*2*pi/Delta; % rectWin
M=ceil(1.1*M); % adding a 10% of the value
w_R=window(@rectwin,M)/M; w_R(N)=0;
y=x.*w_R; Y=fftshift(fft(y,N));
figure;
plot(w_norm, 20*log10(abs(Y)));
% labels...
```

Choosing M to distinguish sinusoids

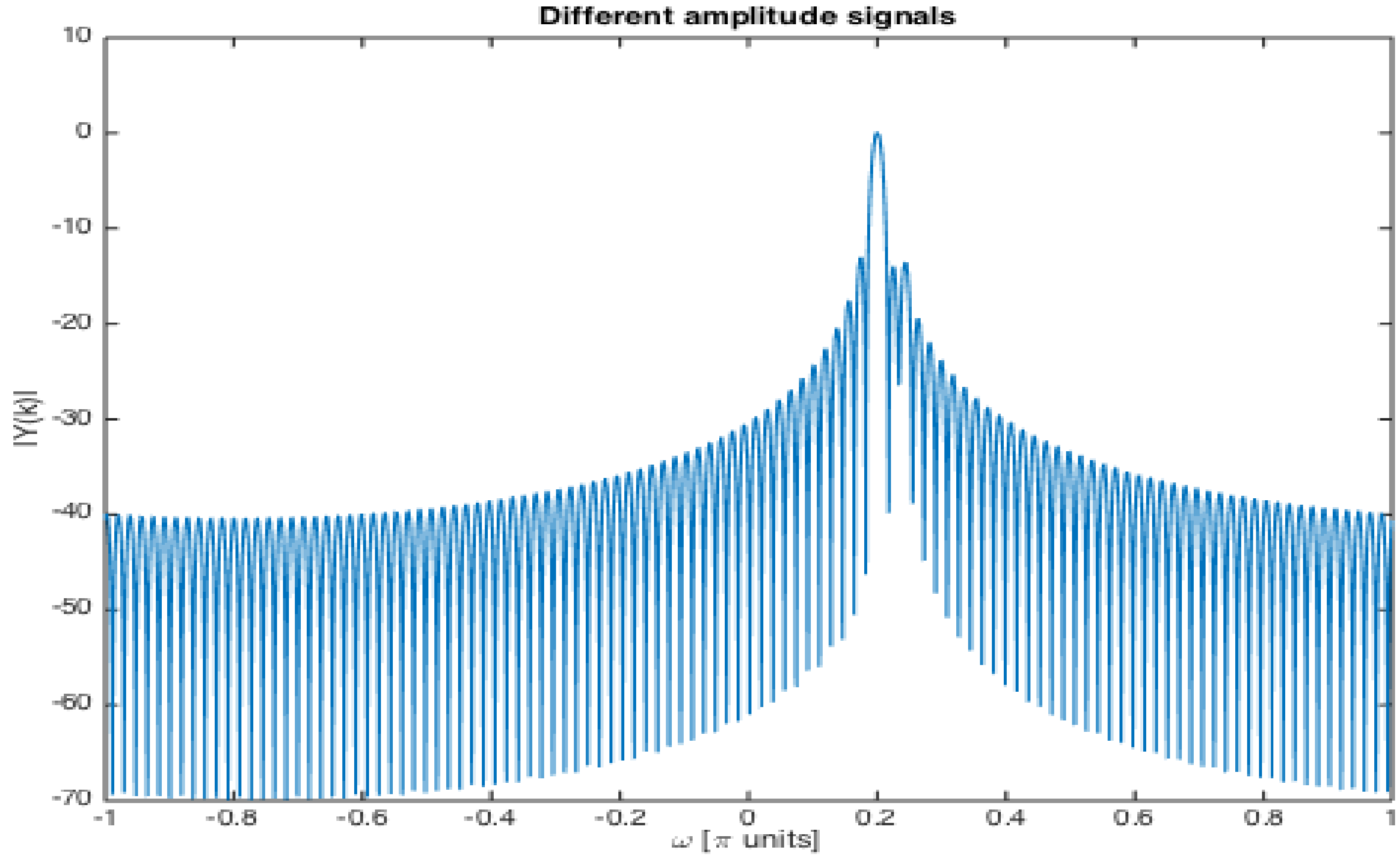


M for different amplitude signals

- Given $x(n) = A\,e^{j\omega_1 n} + 0.1 \cdot Ae^{j\omega_2 n}$
- Is the previously computed M still good?

```
% variables defined before...
x2=exp(1i*omega1*n)+0.1*exp(1i*omega2*n);
y2=x2.*w_R; Y2=fftshift(fft(y2,N));
figure;
plot(w_norm, 20*log10(abs(Y2)));
% labels...
```

M for different amplitude signals

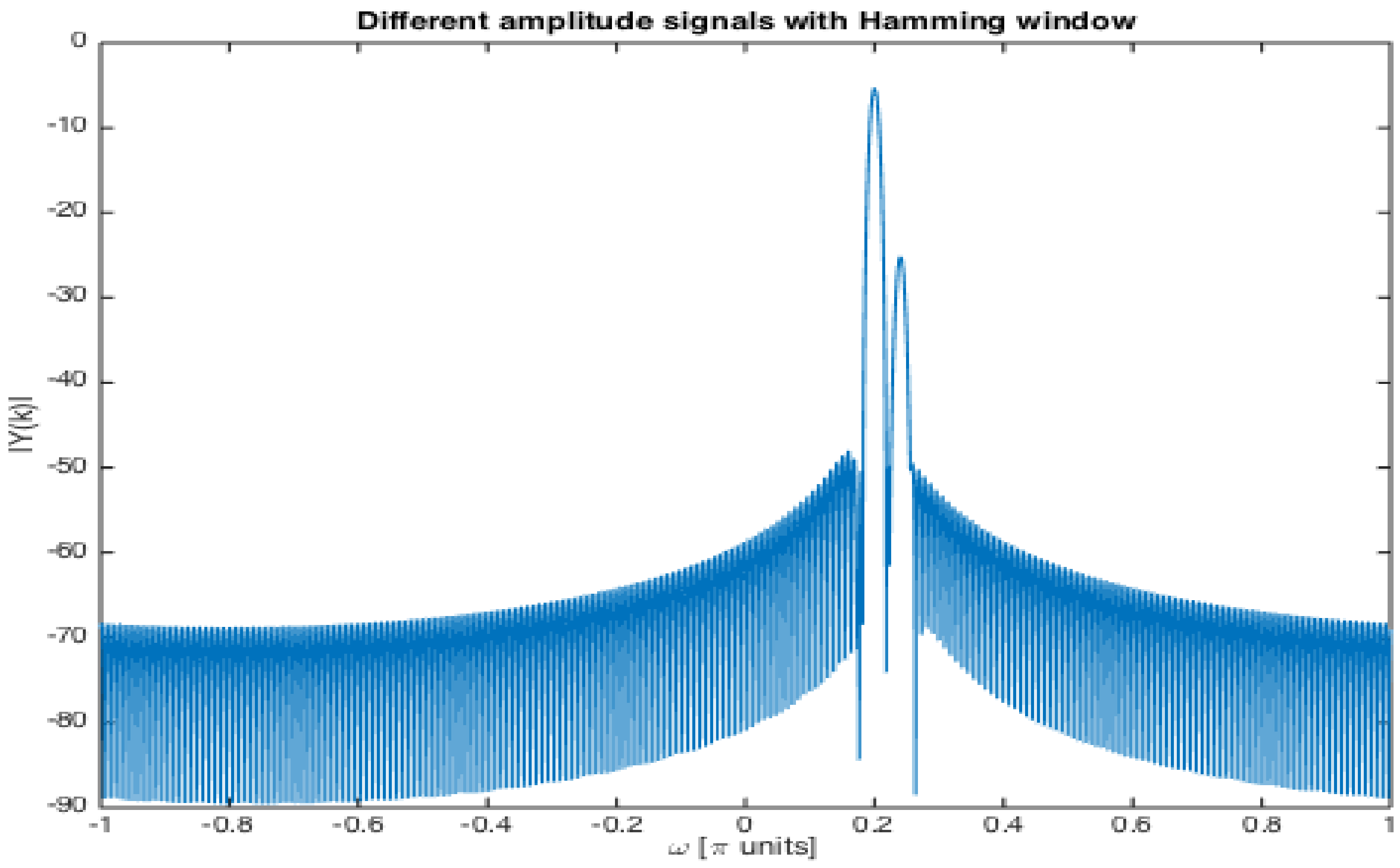


M for different amplitude signals

- Rectangular window provides a roll-off of -6dB/decade
- Let's try a Hamming window

```
L=2; M=2*L*2*pi/Delta; % rectWin
M=ceil(1.1*M); % adding a 10% of the value
w_H=window(@hamming,M)/M; w_H(N)=0;
y3=x2.*w_H; Y3=fftshift(fft(y3,N));
figure;
plot(w_norm, 20*log10(abs(Y3)));
% labels...
```

M for different amplitude signals



Overlap and add

Overlap and add

- Given a N-sample signal x and a K-tap filter h, y=conv(x,y) has $l_y = N + K - 1$ samples
  - For small K (≈12), time-domain convolution is faster
  - For larger K, DFT multiplication is faster
  - For extremely large N or real time operation, we need to use block-level processing
- Windows are used to decompose an infinite-duration signal in a set of finite-duration blocks
  - Similar to the *overlap and save* methods

Overlap and add

- The signal x is segmented into overlapping block x_m
  - $x_m(n)=x(mR:mR+M-1)$ with m =1,2,…,N/R
  - M is the length of the block, N is the length of the signal, R =M-overlap is the hopsize in samples
- Each block is windowed as $x_{m,w}(n)=w(n).*x_m(n)$
- The hopsize value must be chosen so that it satisfies the Constant Overlap and Add condition (COLA): sum of all the overlapping windows must be constant

$$\sum_{m=1}^{N/M} w(n-mR) = C$$

Overlap and add

- Typical windows and their hopsize
  - Rectangular window with 0% overlap (R=M)
  - Rectangular window with 50% overlap (R=M/2)
  - Bartlett window at 50% overlap (R=M/2)
  - Hamming window at 50% overlap (R=M/2)
  - Hamming window at 75% overlap (R=M/4)

Overlap and add

- The N_{fft}-point DFT of $x_{m,w}(n)$ is computed
  - $N_{fft} > M+K-1$
  - K is the length of the filter h(n)
- $Y_m = X_m H \rightarrow y_m(n) = \mathrm{IDFT}\{Y_m\}$
  - H is the N_{fft}-point DFT of h(n)
- $y(mR:mR+M-1) = y(mR:mR+M-1) + y_m$
- See the matlab code for a graphical example

Overlap and add

- Load 'gb.wav' in x
- Define a filter h from $H(z)=1-0.99z^{-1}$ in K=1000 samples
- Use a Bartlett window with length of 50 ms and hopsize of 50%
- Choose Nfft accordingly
- Filter x with h using OLA

Overlap and add


```
[x, Fs]=audioread('gb.wav'); N=length(x);
K=1000; delta=[1;0]; delta(K)=0;
h=filter(1,[1 -0.99],delta); y_=conv(x,h);
M=floor(0.050*Fs); R=floor(M*0.5);
% frame of 50ms with 50% overlap
w=window(@bartlett,M); Nfft=2^ceil(log2(K+M-1));
H=fft(h, Nfft);
x(N+M+R+Nfft)=0; y=zeros(size(x));
for m=1:floor(N/R)+1
  mR=(m-1)*R+1; x_m=x(mR:mR+M-1);
  x_wm=x_m.*w; X_wm=fft(x_wm, Nfft);
  Y_m=X_wm.*H; y_m=ifft(Y_m);
  y(mR:mR+Nfft-1)=y(mR:mR+Nfft-1)+y_m;
end
y=y(1:N+K-1);
```

Comparison between OS and OLA

- Overlap and save
  - the blocks are not windowed (rectangular)
  - The first samples are neglected due to circular-convolution errors
  - The output is the result of a stacking operation
- Overlap and add
  - the blocks are windowed
  - The length of the FFT is properly chosen to create meaningful DFT-multiplication
  - The output is the result of a sum operation
  - **Best for analysis and processing purposes**

The Short Time Fourier Transform

The Short Time Fourier Transfrom

- The STFT of a signal x(n) is a matrix X(k,m) where
  - k is the bin of the DFT
  - m is the index of a frame (a block)
  - Also called the *spectrogram*
- It is used to see the evolution in time of the DFT of x

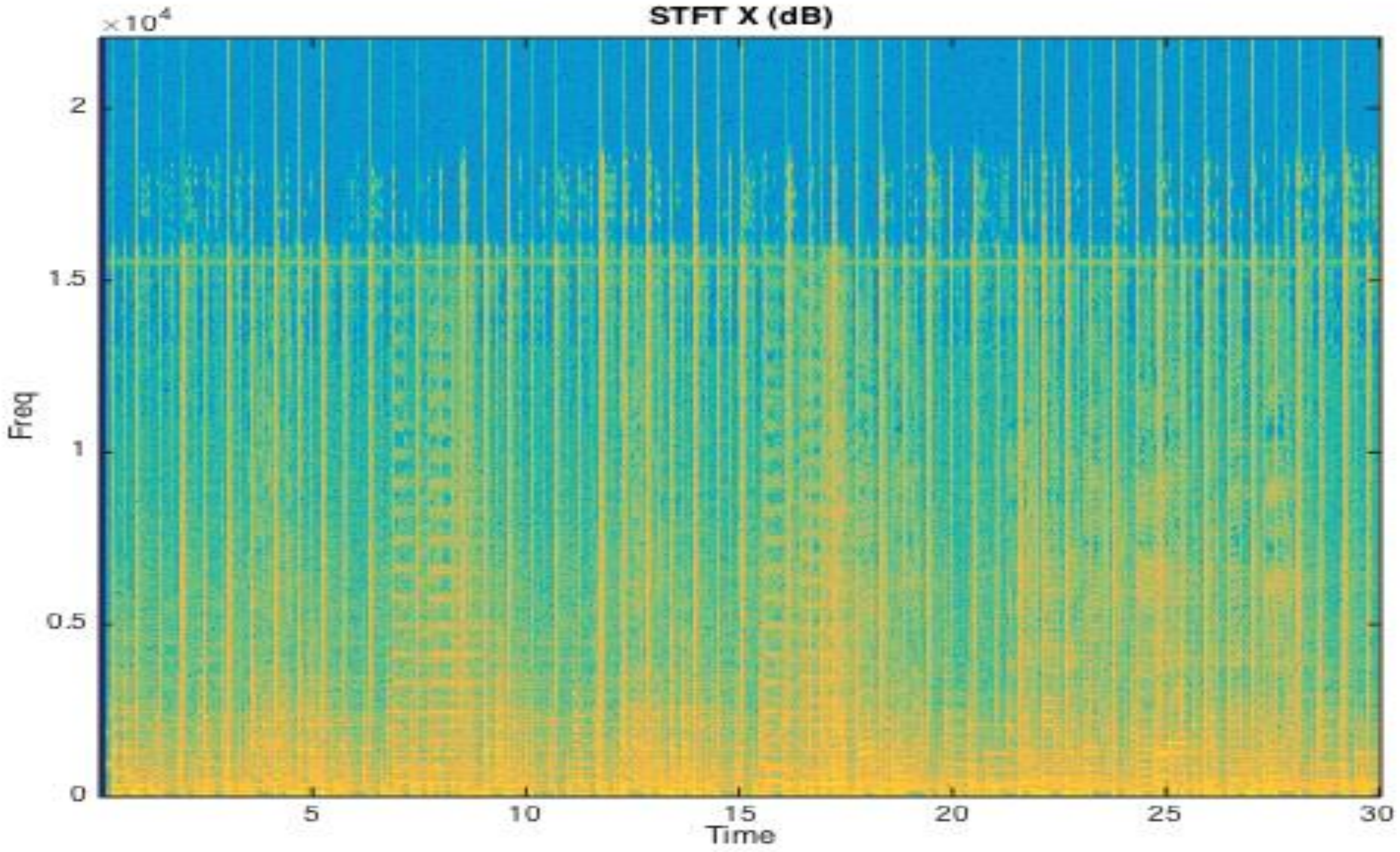
The Short Time Fourier Transfrom

- The signal x(n) is segmented into overlapping frames x_m
  - $x_m(n)=x(mR:mR+M-1)$ with $m=1,2,\dots,N/R$
- Each x_m is multiplied by an analysis window w(n)
  - $x_{m,w}=w(n).*x_m(n)$
- The DFT of $x_{m,w}$ is computed and stored in the spectrogram
  - $X(m,k)=X_{m,w}(k)$
  - the positive frequency components are usually considered
- Matlab also provides the function `spectrogram`

The Short Time Fourier Transform


```
[x, Fs]=wavread('gb.wav');
N=length(x);
M=floor(0.050*Fs);
R=floor(M*0.5);
w=window(@bartlett,M);
Nfft=M;
x(N+M)=0;
X=zeros(floor(Nfft/2+1),floor(N/R)+1);
for m=1:floor(N/R)+1
  mR=(m-1)*R+1; x_m=x(mR:mR+M-1);
  x_wm=x_m.*w; X_wm=fft(x_wm, Nfft);
  X(:,m)=X_wm(1:floor(Nfft/2+1));
end
X=flipud(X);
```

The Short Time Fourier Transform



Switching magnitude and phases STFT

- Load 'fgi.wav' and 'wttj.wav' as x_f and x_w
  - convert them to mono (mean over the channels)
  - Trim them to a common length
- Compute the two spectrograms X_f and X_w
  - using any window and specifying Noverlap
- Switch phases

$$X_{fw} = \left|X_f\right| \exp(i\angle X_w) \qquad X_{wf} = |X_w| \exp\left(i\angle X_f\right)$$

- Use `istft` (defined by the teacher) to compute x_{fw} and x_{wf}
- How to they sound?

Switching magnitude and phases STFT


```
[y_fgi,Fs]=audioread('fgi.wav'); y_fgi=mean(y_fgi.');
[y_wttj,Fs]=audioread('wttj.wav'); y_wttj=mean(y_wttj.');
%mean converts mono to stereo

Nmin=min(length(y_fgi),length(y_wttj));
y_fgi=y_fgi(1:Nmin); y_wttj=y_wttj(1:Nmin);

logN=floor(log2(0.05*Fs)); N=2^logN; win=window(@hamming,N);
Noverlap=2^(logN-1); Nol=Noverlap % 50% overlap

Y_fgi=spectrogram(y_fgi, win,Nol);
Y_wttj=spectrogram(y_wttj, win,Nol);
Y_fm_wa=abs(Y_fgi).*exp(1i*angle(Y_wttj));
Y_wm_fa=abs(Y_wttj).*exp(1i*angle(Y_fgi));
y_wm_fa=istft(Y_wm_fa,Nol); y_fm_wa=istft(Y_fm_wa,Nol);

sound(y_fm_wa,Fs); sound(y_wm_fa,Fs)
```