Digital Signal Processing 1

Krzysztof Malczewski

www.kstio.com/dsp

Section 1 Introduction

- **Signal**: time-varying measurable quantity
  whose variation normally conveys information.

- **Quantity** often a voltage obtained from some transducer
  e.g. a microphone.

- Define two types of signal:
  - Continuous time (analog)
  - Discrete time

analog signals

- Continuous variations of voltage.
- Exist for all values of t in range -∞ to +∞.

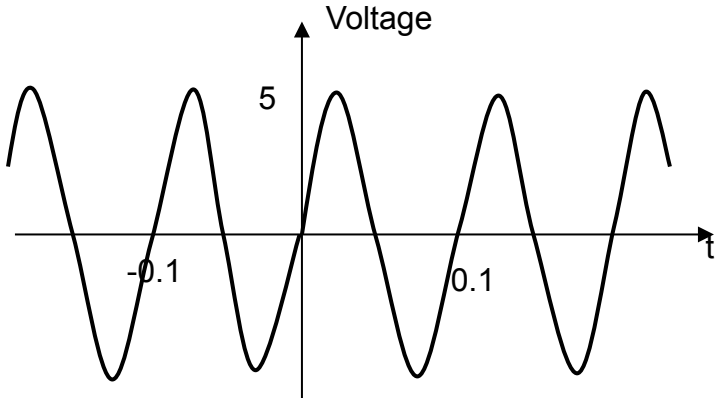
Examples:

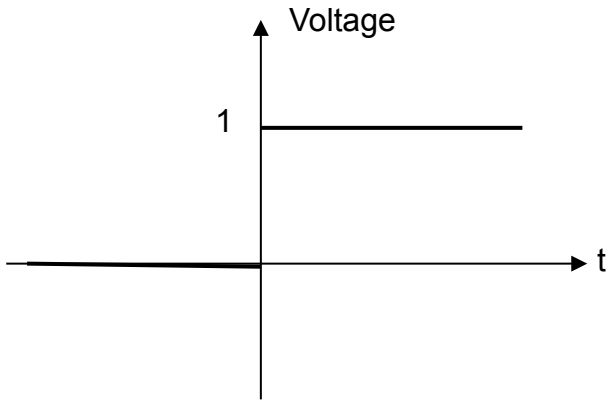
(i) 5sin(62.82t) :

sine-wave of frequency 62.82 radians/second (10 Hz)

Nie można wyświetlić obrazu. Na komputerze może brakować pamięci do otwarcia obrazu lub obraz może być uszkodzony. Uruchom ponownie komputer, a następnie otwórz plik ponownie. Jeśli czerwony znak x nadal będzie wyświetlany, konieczne może być usunięcie obrazu, a następnie ponowne wstawienie go.

Graph of voltage against time gives continuous 'waveform' :

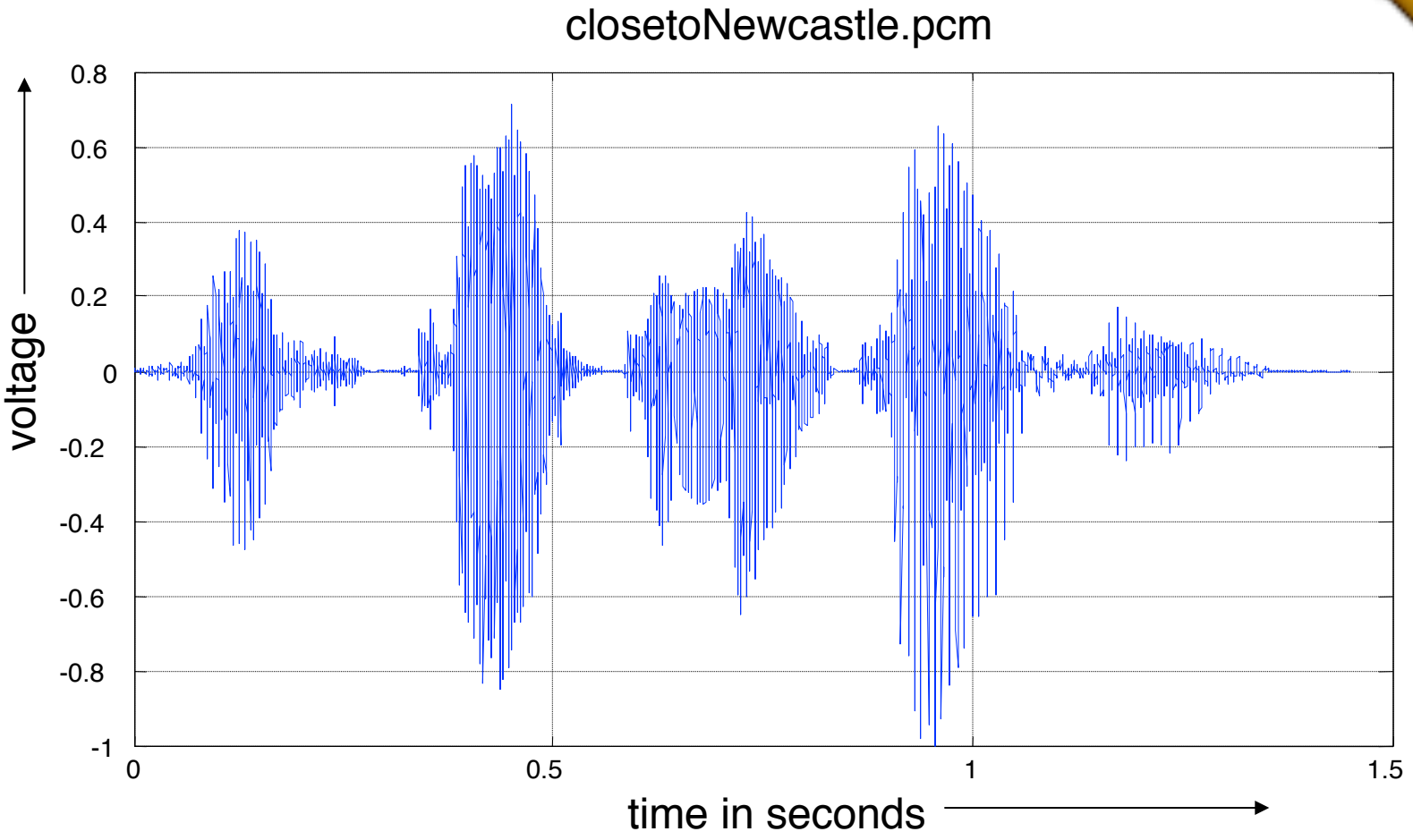




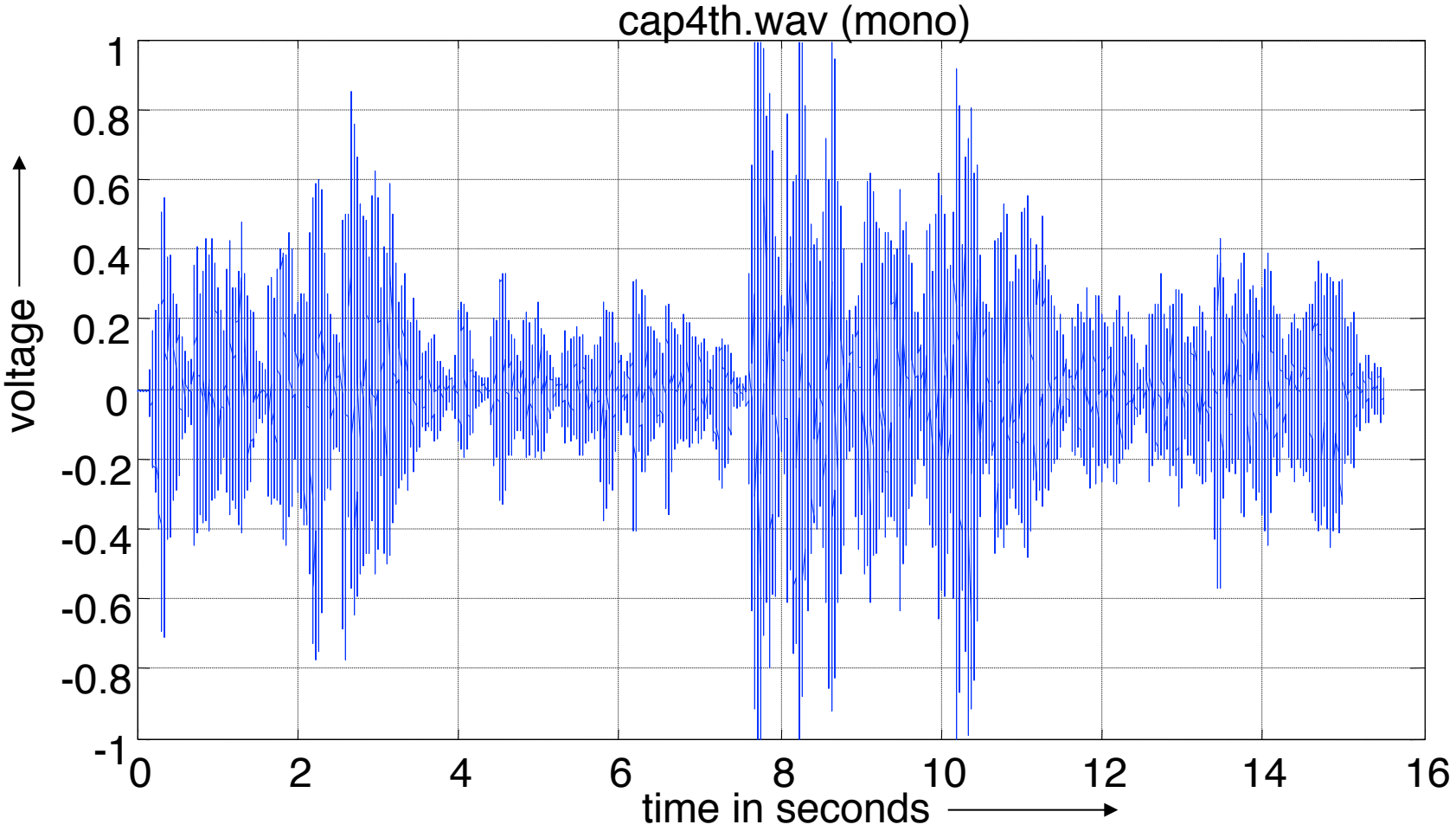
‘Sound’

- Variation in air pressure
- May be generated by musical instruments or a human voice.
- E.g. by a piano string vibrating or your vocal cords vibrating
- Local pressure variation travels thro’ the air to your ear or a microphone.
- There it causes a ‘diaphragm’ to vibrate.
- Microphone produces a continuous voltage proportional to the variation in air pressure.
- Graph of voltage against time is ‘sound waveform’.

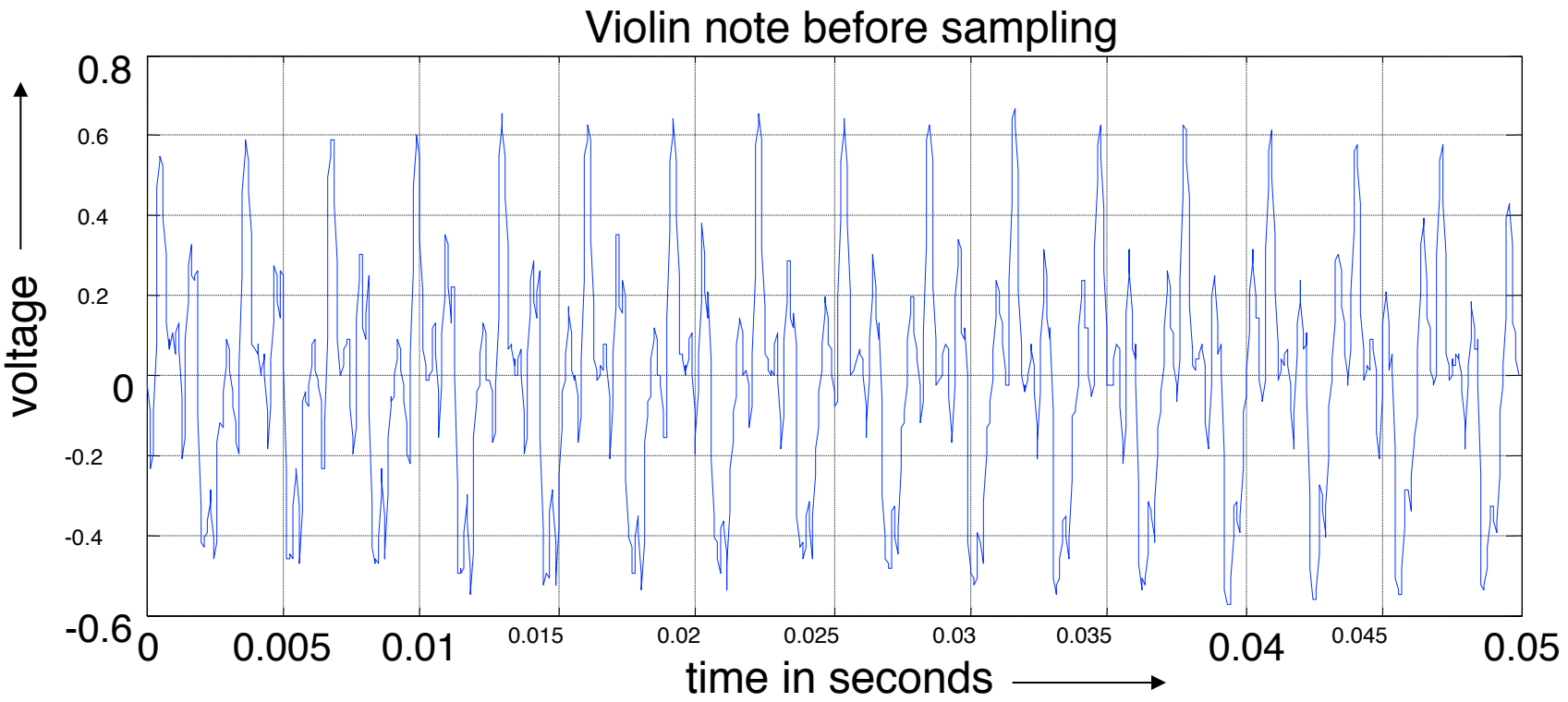
Segment of telephone speech: ‘Its close to Newcastle’



16 s segment of violin music



50ms segment of music: violin note



What is frequency of note being played? Ans ≈16 cycles in 0.05 s ≈ 320 Hz

Discrete-time signal

- Exists only at discrete points in time.
- Often obtained by **sampling** an analog signal,

  i.e. measuring its value at discrete points in time.
- Sampling points separated by equal intervals of T seconds.
- Given analog signal **x(t)**, **x[n]** = value of x(t) when t = nT.
- Sampling process produces a sequence of numbers:

  { ..., x[-2], x[-1], x[0], x[1], x[2], ... }
- Referred to as {x[n]} or ' the sequence x[n] '.
- Sequence exists for all integer n in the range -∞ to ∞.

Examples of discrete time signals:

(i) **{..., -4, -2, 0, 2, 4, 6, ...}**

sequence whose nth element, x[n], is defined by : x[n] = 2n.

Underline sample corresponding to n = 0.

(ii) **{ ..., -4.75, -2.94, 0, 2.94, 4.75, 4.76, ...}**

sequence with x[n] = 5 sin(62.82t) with t=nT and T=0.01.

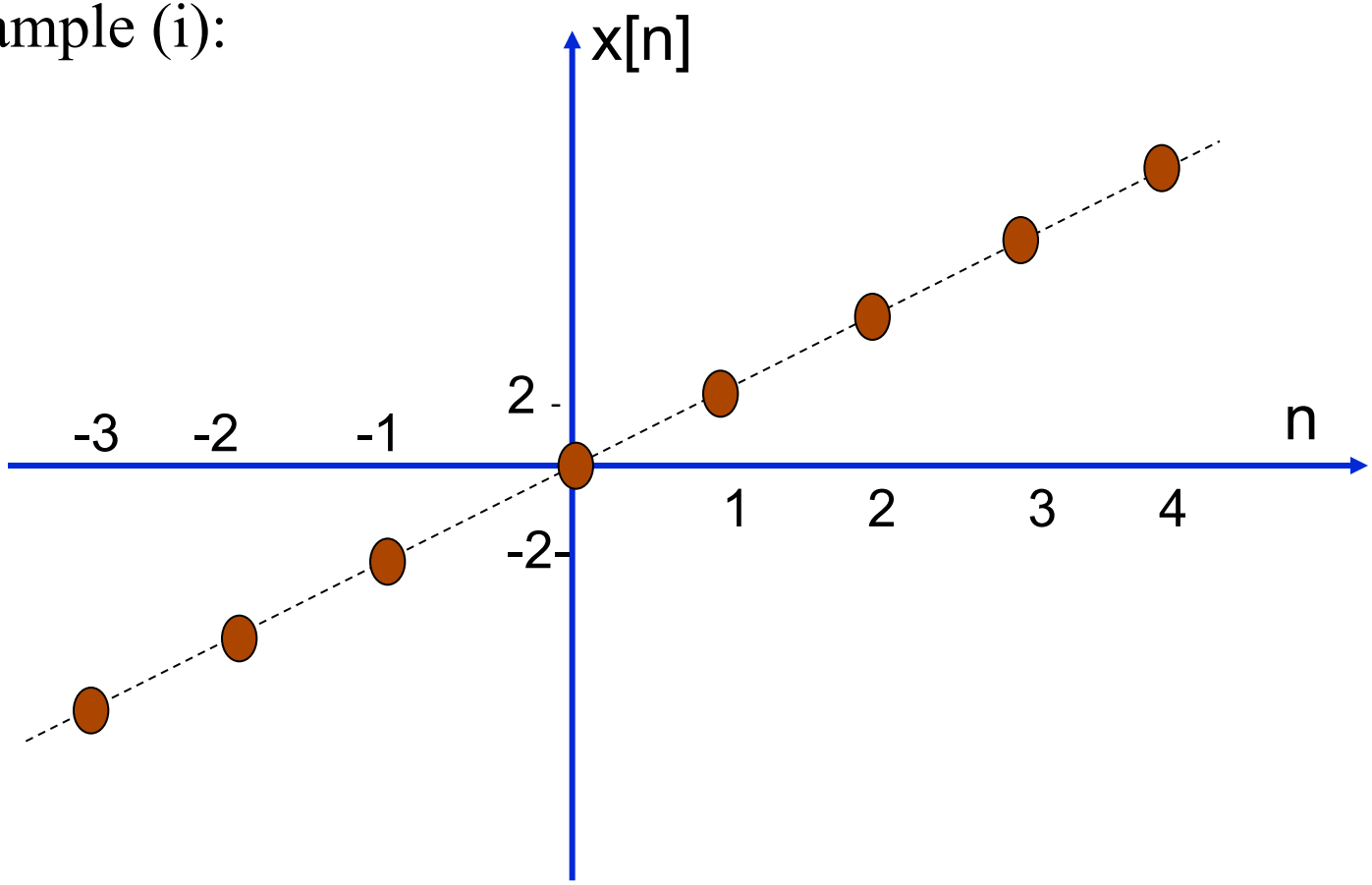
(iii) **{ ..., 0, ..., 0. 0, 1, 1, 1, ..., 1, ...}**

" unit step " sequence whose nth element is:

$$u[n] = \begin{cases} 0 : n < 0 \\ 1 : n \geq 0 \end{cases}$$

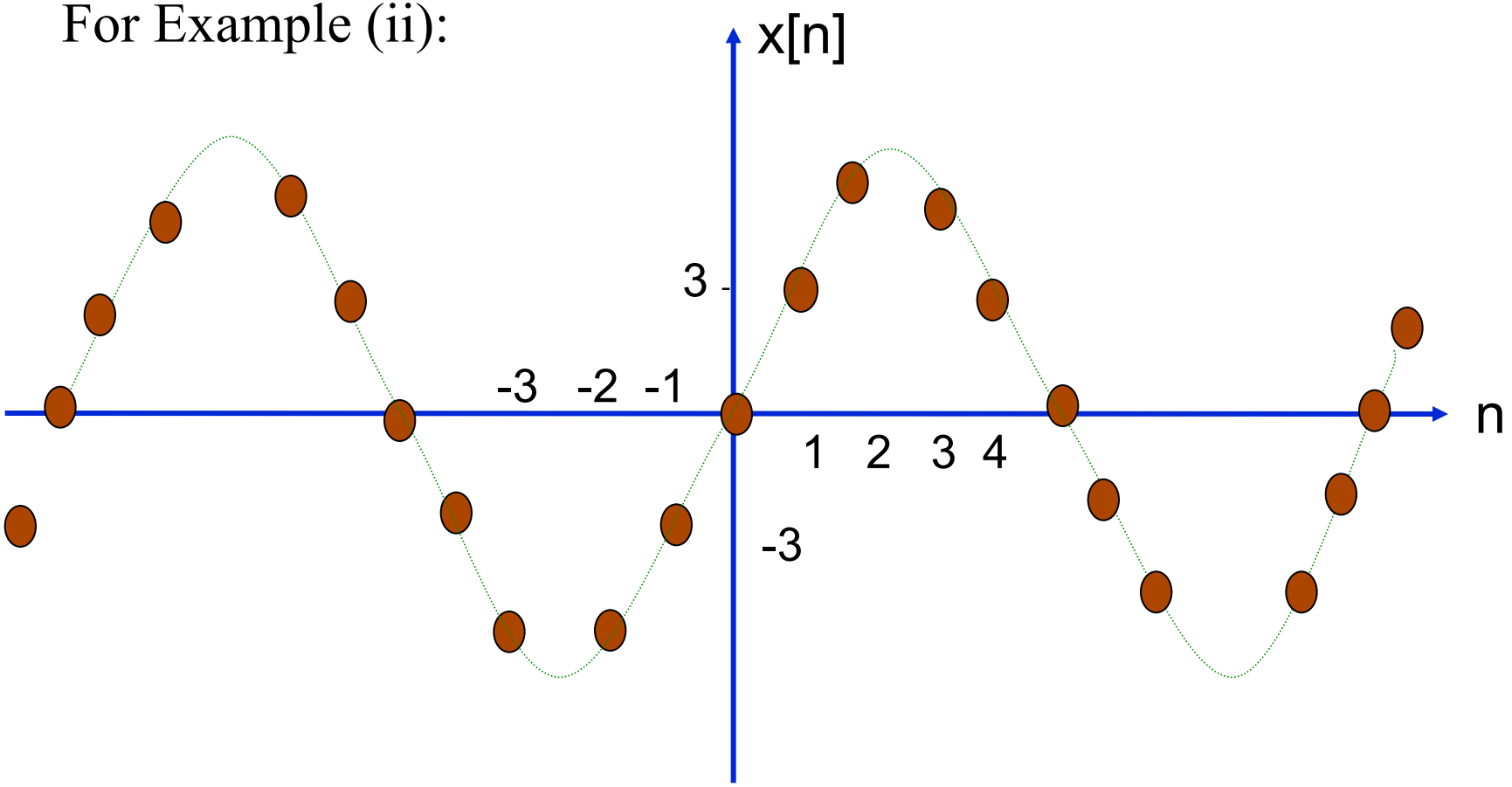
Graphs of discrete time signals

For Example (i):



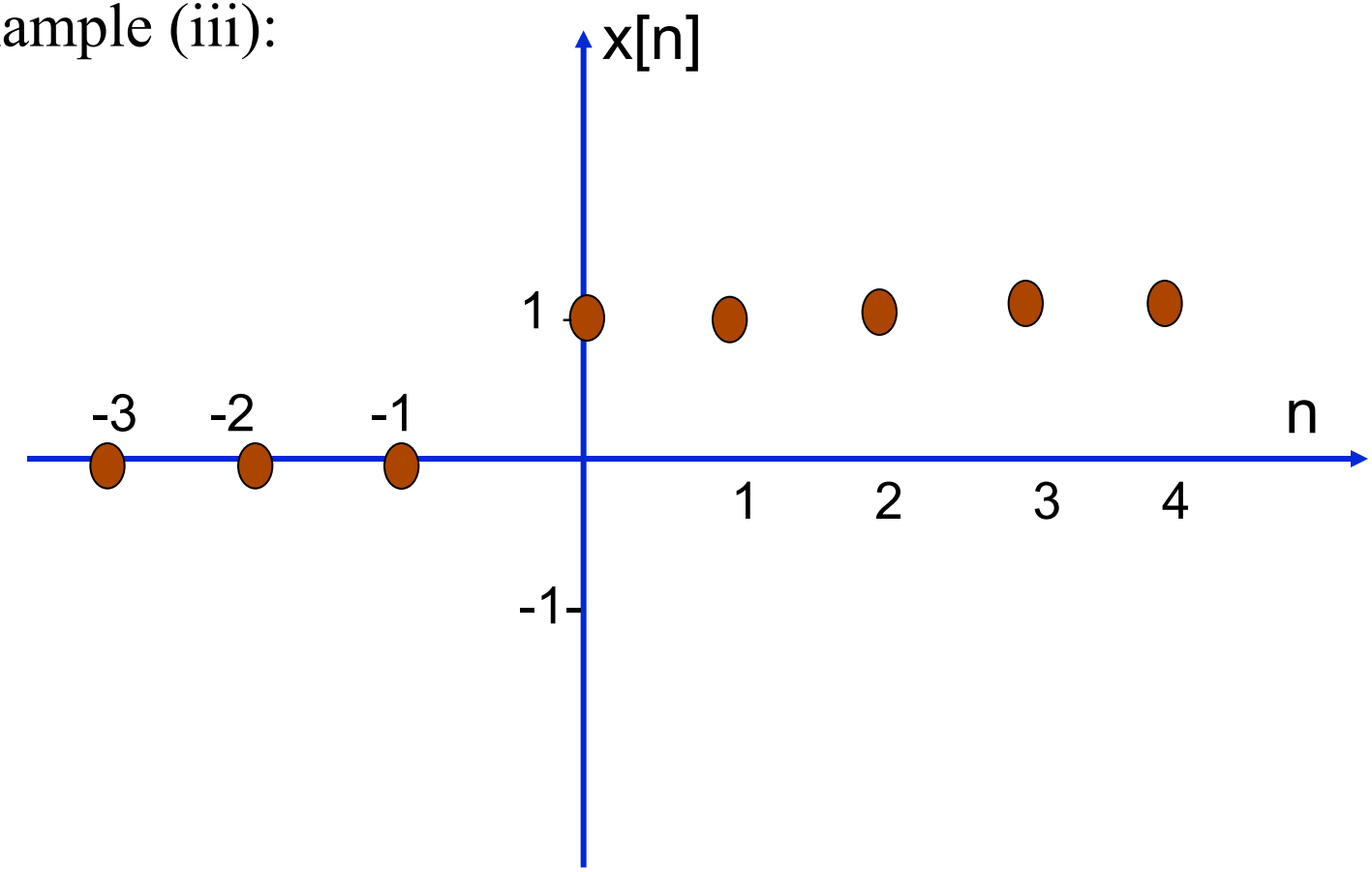
Graphs of discrete time signals

For Example (ii):

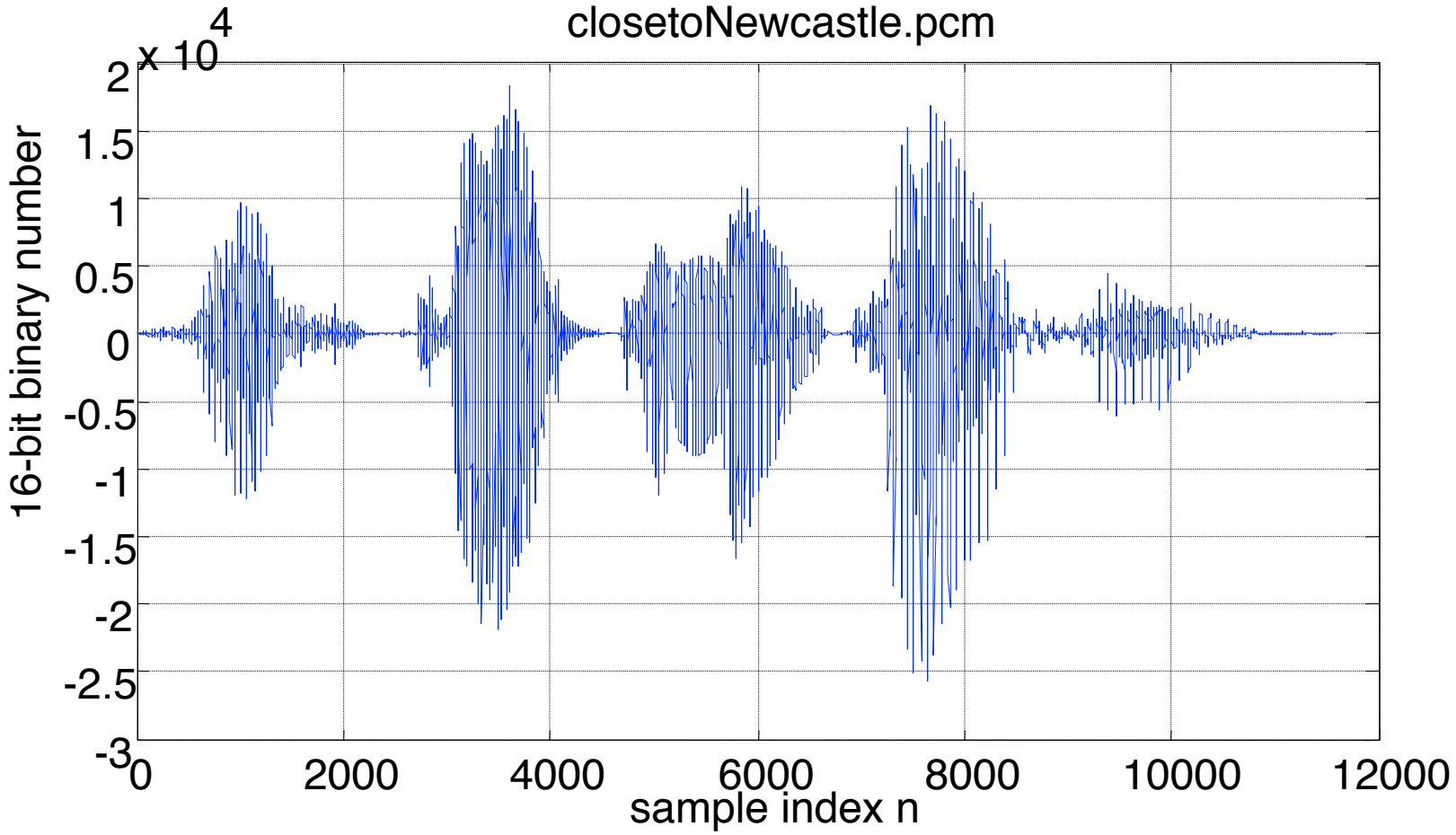


Graphs of discrete time signals

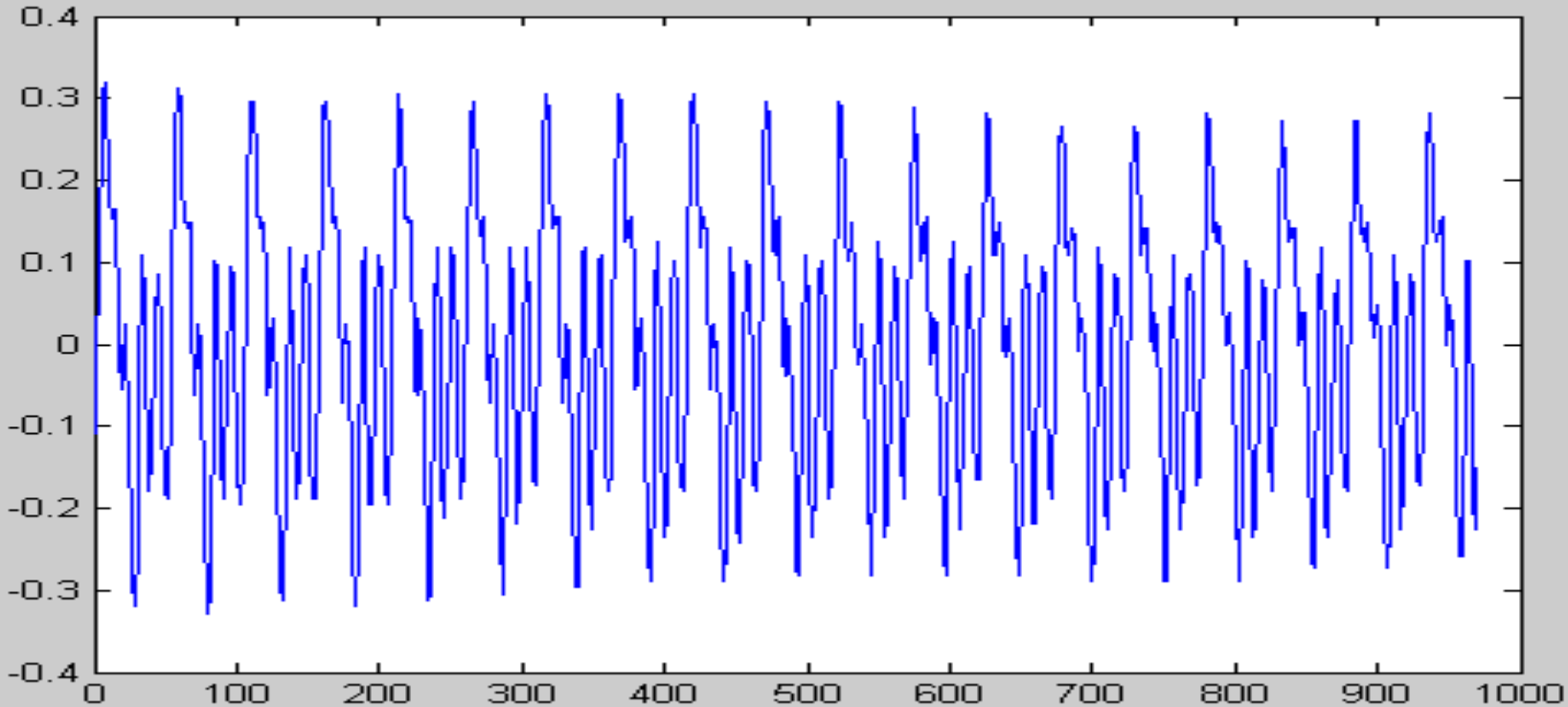
For Example (iii):



Speech sampled at 8kHz with 16 bits per sample



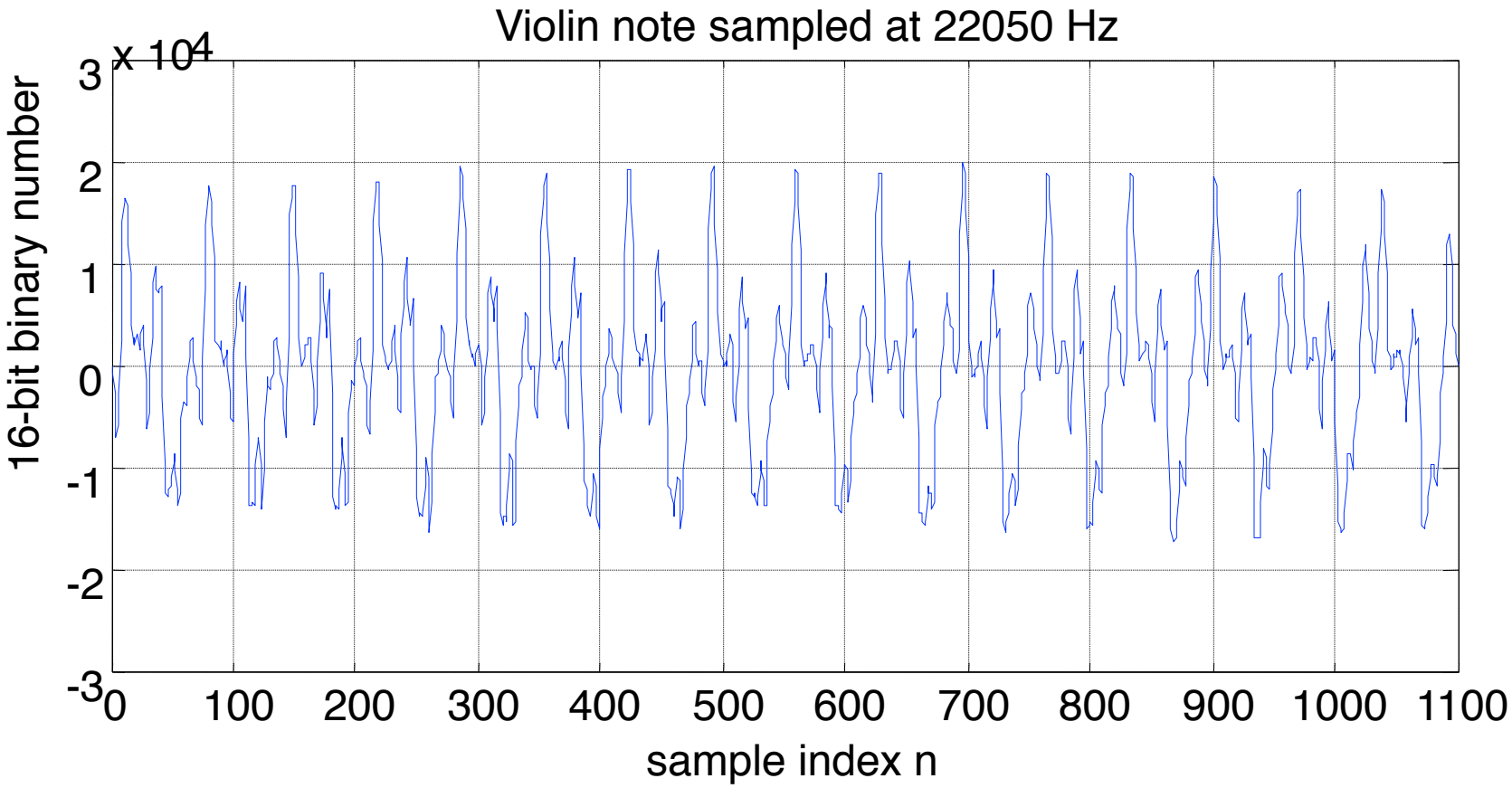
45 ms segment of music sampled at 22.05 kHz



Frequency of note ≈19/1000 = 0.019 cycles/sample

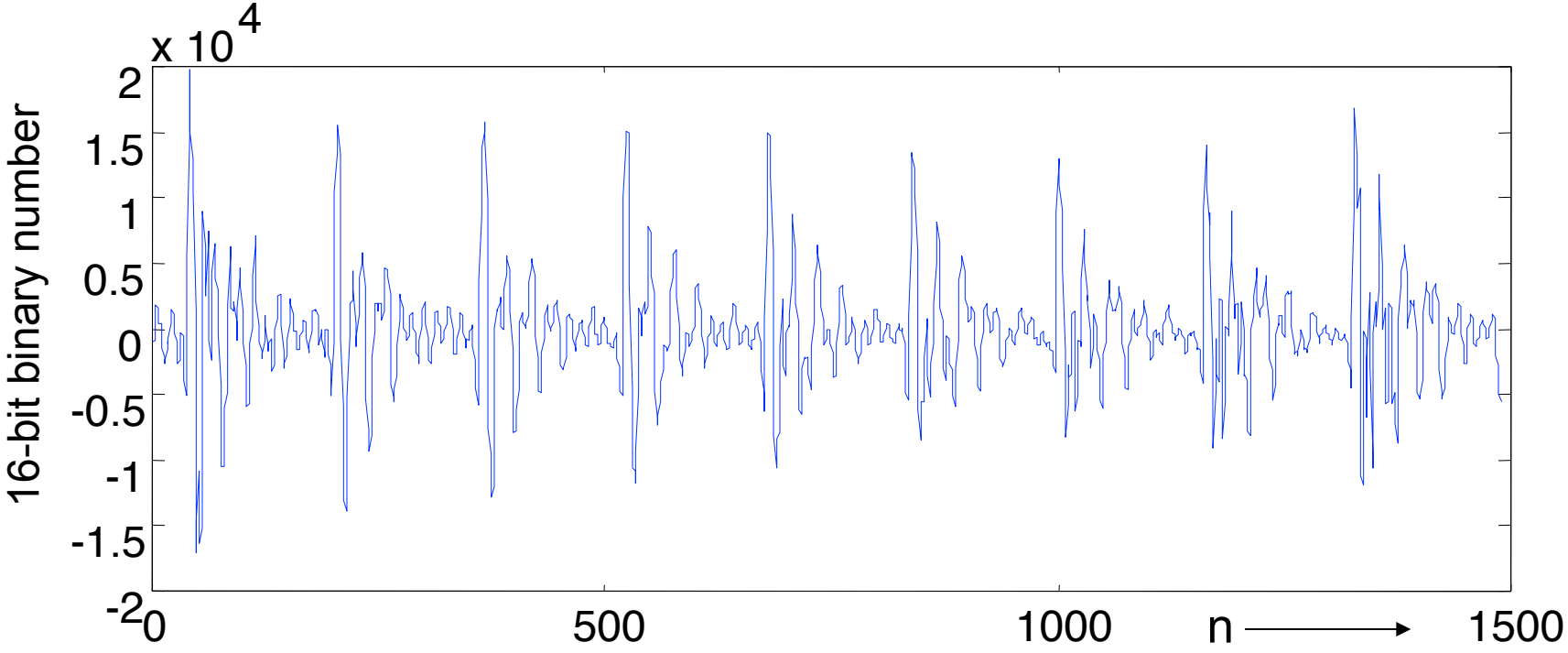
= 0.019 *22050 = 419 Hz

50 ms segment of music sampled at 22.05 kHz



What's the frequency now, folks?

Voiced telephone speech sampled at 16000 Hz



Short (≈ 100ms) segment of a vowel. [1ms = 1/1000 second]

Vowels are approximately periodic.

What is the pitch of the voice here?

Ans: ≈ 9 cycles in 1/10 s i.e. ≈ 90 Hz - probably male speech

Digital signal

- Discrete time signals often generated by **ADC** devices.
- Produce binary numbers from sampled voltages or currents.
- Accuracy determined by 'word-length ' of ADC device,
  i.e. number of bits available for each binary number.
- **Quantisation:** Truncating or rounding sampled value
  to nearest available binary number
- Resulting sequence of quantised numbers is a **digital signal**.
- i.e. discrete time signal with each sample digitised for arithmetic processing.

Signal Processing

- analog signals "processed" by circuits consisting of resistors, capacitors, transistors etc.
- Digital signals "processed" using programmed computers, microcomputers or special purpose digital hardware.
- Examples of the type of processing that may be carried out are:
  - (i) amplification or attenuation.
  - (ii) filtering out some unwanted part of the signal.
  - (iii) rectification : making waveform purely positive.
  - (iv) multiplying by another signal.

‘Real time’ & ‘non-real time’ processing

’Real time' processing:

A mobile phone contains 'DSP' processor fast & powerful enough to perform the mathematical operations required to process digitised speech as it is being received.

‘Non-real time’ processing:

- A PC can perform DSP processing on a stored recording of music.
- It can take as much time as it needs to complete this processing.
- Useful in its own right; e.g for MP3 compression.
- Also used to 'simulate' software for real time DSP systems before they are built into special purpose hardware.
- Simulated DSP systems tested with stored segments of speech/music.

Introduction to 'MATLAB'

- High level computing language for matrix operations.
- Widely used for studying signal processing & comms.
- No need to declare variables in advance.
- Each variable may be array with real or complex elements.
- No distinction between reals & integers

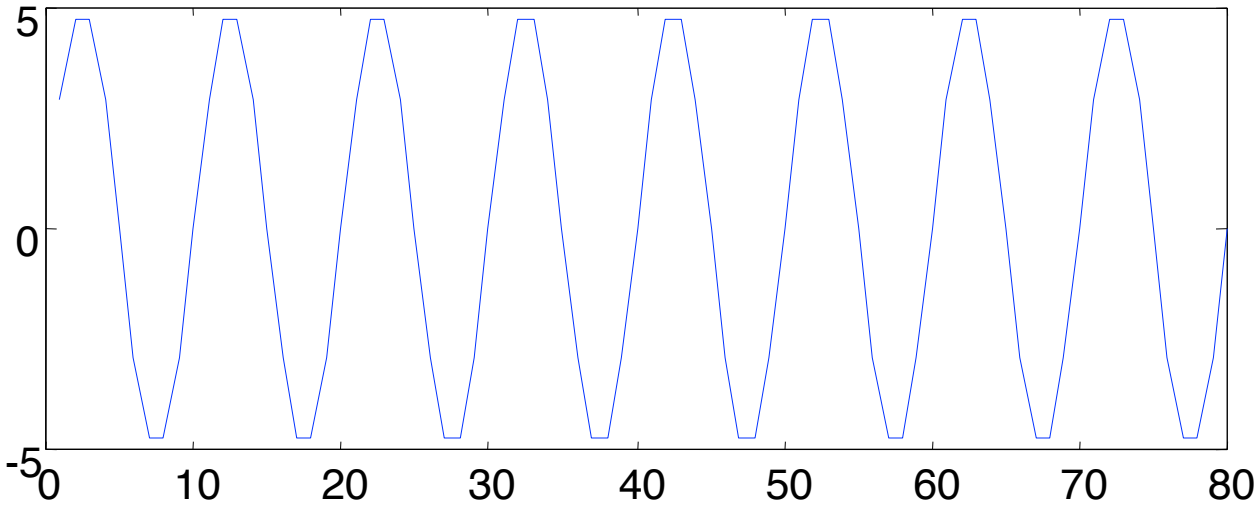
Examples of MATLAB statements

```
A = [ 1   2.8   3.2   4
      5    6    7.7   8  ] ;

X = [ 1  2  3  4 ] ;

Y = [ 1
      2
      3
      4 ]  ;

Z = 3 ;

W = 4 + 5.5j ;                    % Comment
```

Generating, storing & plotting signals

Generate & plot 80 samples of a 10 Hz sine-wave of amplitude 5, sampled at 100 Hz:

```
clear all;
T = 0.01;    % sampling interval (seconds)
for n=1:80
   s(n) = 5*sin(2*pi*10*n*T);
end;
plot (s);
```



Signals in MATLAB

Segments in row or column 'vector' arrays: n x 1 or 1 x n matrices.

Previous example produces row vector s

To produce column vector s containing sine-wave segment

```
for n = [1:100]'
        s(n) = sin(n*pi/10)
end;
```

Another (preferred) way of defining column vector s:

```
n=[1:100]' ;          % 100x1 col vector with entries 1,2,...,100
s(n) = sin(n*pi/10)   % For each entry of n, calculate value.
```

Row-vectors & column-vectors

x = [1.1, 2, 3.3, 4, 5] defines row vector

x = [1.1; 2; 3.3; 4; 5] defines column vector.

Easier to write than:

```
x =   [ 1.1
        2
        3.3
        4
        5 ]
```

Transpose

- **x.'** is transpose of x. (Note the **.'** symbols).
- If **x** is col vector, **x.'** will be row vector.
- Note dot apostrophe (**.'**) rather than simply apostrophe (').
- Omitting dot will give "complex conjugate transpose".
- Index of first element of a row or column vector is always one.
- Sometimes inconvenient since a signal starts at time zero.
- Frequency spectrum normally starts at frequency zero.
- Consider adopting following statements:

```
for n = 0 : 1023
        x(1+n) = sin(n*pi/10)
end;
```

- Preserves natural definition of n as time-index.

Command line mode

- Run MATLAB by clicking on icon & type in statements.
- If in doubt about any command, e.g. ‘plot’ type: **help plot**
  <return

>> 4 + 5

>> X=4;Y=5; X+Y

>> X*Y

>>X/Y

>> bench **(Tells you how fast your computer is)**

>> for n = 0 : 99

>> x(1+n) = sin(n*pi/10)

>> end;

>> plot(x)

Scripting ‘m-files’

- More convenient to have statements in script file e.g. **myprog.m**.
- Just a text file often referred to as an "M-file".
- To execute the statements in "**myprog.m**" just type
- "**myprog"**
- MATLAB has a text editor.
- Paths must be set to a "work" directory
  where the "M-files" will be stored.

Digital media file formats

- Confusing number of different ways of storing speech, music, images & video in digital form. Consider just six:
- '***.pcm**' or '***.raw**': binary file without header often used for uncompressed 'narrowband' speech sampled at 8 or 16 kHz.
- '***.wav**': binary file with header often used uncompressed high-fidelity stereo sound (music or speech) sampled at 44.1 kHz.
- '***.mp3**' for mp3 compressed music or hi-fi speech.
- '***.tif**' files for images with lossless compression.
- '***jpg**' files for images with lossy compression
- '***.avi**' audio/video interleaved files.

MATLAB functions for reading/writing *.pcm

```
% Open *.pcm file of 8 kHz sampled speech,16 bits/sample
IFid=fopen('operamp.pcm','rb');
inspeech = fread(IFid, 'int16');
L = length(inspeech);            % number of samples
scale = max(abs(inspeech)) + 1;
sound(outspeech/scale,8000,16);   % listen to the speech
outspeech = inspeech / 2;     % reduce by 6 dB
OFid = fopen( ‘opout.raw’ , ‘wb’ );
fwrite(OFid,outspeech, ‘int16’ );
fclose('all');
```

MATLAB functions for reading/writing *.wav

```
clear all;  close all;          % closes all graphs etc.
[inmusic, Fs, Nbits] = wavread('cap4th.wav');
S=size(inmusic);  % inmusic may be mono or stereo
L=S(1);                 % number of mono or stereo samples
if S(2) ==2 disp( 'music is stereo' ); end;
sound(inmusic,Fs,Nbits);
outmusic = inmusic /sqrt(2); % reduce amplitude by 3 dB
wavwrite(outmusic, Fs, Nbits,' outfile.wav' );
fclose ( 'all' );   % closes all files
```

Another way of saving/loading arrays

- Does not produce files that can be read by wave-editors, though it can produce text files.
- To save an array called "b" in a disk file called "bits.dat":

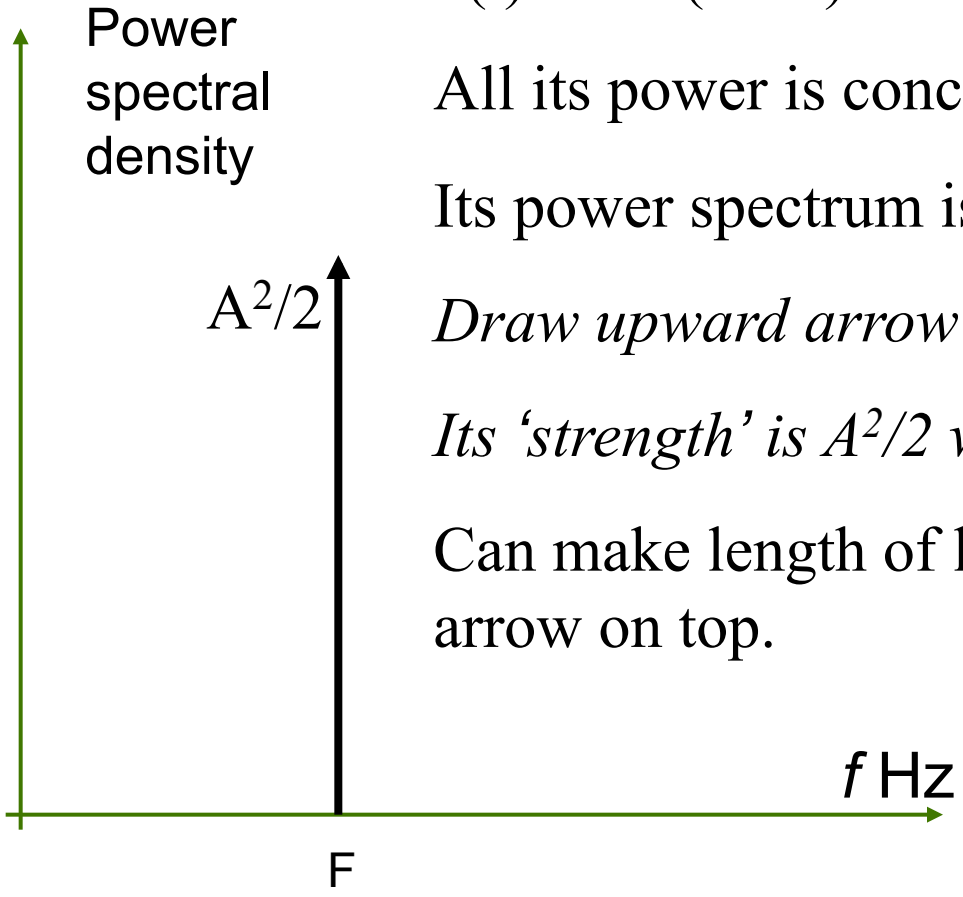
  save bits.dat b -ascii
- Omitting ‘-ascii’ creates a more compact ‘non-ascii’ binary file.
- To read contents of ascii file ‘bits.dat’ into array called ‘bits’:

  load bits.dat -ascii

Frequency spectrum & sampling

- A sine-wave has a ‘frequency’ (f) in Hz or $2\pi f$ radians/second.
- Sine-waves do not sound very nice.
- Speech & music need of lots of sine-waves added together.
- Harmonically related, i.e. *f, 2f, 3f, 4f, 5f, ...*
- Other sounds approximated as sum of lots of very small sine-waves.
- Any type of signal has a ‘frequency spectrum’.
- Most energy in **speech** is within 300 Hz to 3400 Hz (3.4 kHz).
- Energy in **music** that we can hear is in range 50 Hz to 20 kHz.

Spectral analysis of a sine-wave



x(t)=Asin(2πFt) is sine-wave of frequency F Hz

All its power is concentrated at F Hz.

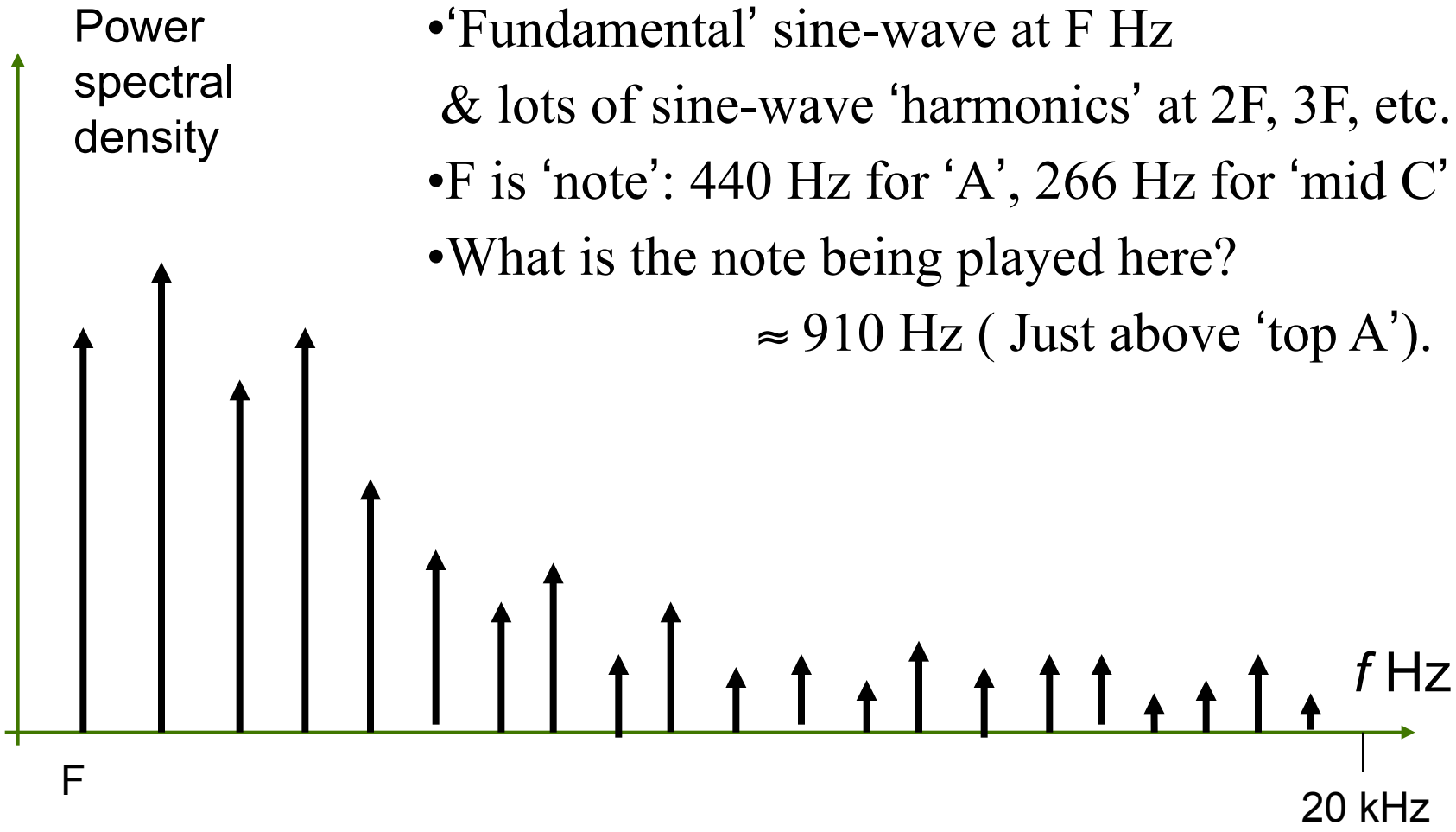
Its power spectrum is infinitely high line a*t f=F.*

Draw upward arrow & call it an 'impulse'.

Its 'strength' is $A^2/2$ watt/Hz for Asin(2πFt)

Can make length of line ∝ strength but must put arrow on top.

Spectral analysis of musical note



- 'Fundamental' sine-wave at F Hz
  & lots of sine-wave 'harmonics' at 2F, 3F, etc.
- F is 'note': 440 Hz for 'A', 266 Hz for 'mid C'
- What is the note being played here?
  ≈ 910 Hz (Just above 'top A').

Spectral analysis of analog signals

- Spectral analysis of periodic sound before it is sampled is tricky because of the upward arrows.
- They are infinitely high!!
- We plotted their ‘strengths’, not their amplitudes.
- Things become easier when we spectrally analyse digitised sound (later)!

'Sampling Theorem'

- If a signal has all its spectral energy below **B** Hz, & is sampled at **Fs** Hz where **Fs** $\geq$ **2B**, it can be reconstructed exactly from the samples.
- Speech below **3400 Hz** can be sampled at **6800 Hz** (in practice **8 kHz**).
- Music can be sampled at **40 kHz** (in practice **44.1 kHz**).
- When we process a sampled signal in MATLAB & sampling rate was **Fs** Hz, we can only observe frequencies in range **0** to **Fs/2**.

‘Aliasing’

- What happens if you sampled a speech or music signal at **Fs** & it has some spectral energy above **Fs/2** ?
- Leads to a form of distortion known as ‘aliasing’ which sounds very bad.
- Must filter off all spectral energy at & above **Fs/2** Hz before sampling at **Fs**.

-> ex. ppt

MATLAB Signal Processing toolbox

Set of functions for:

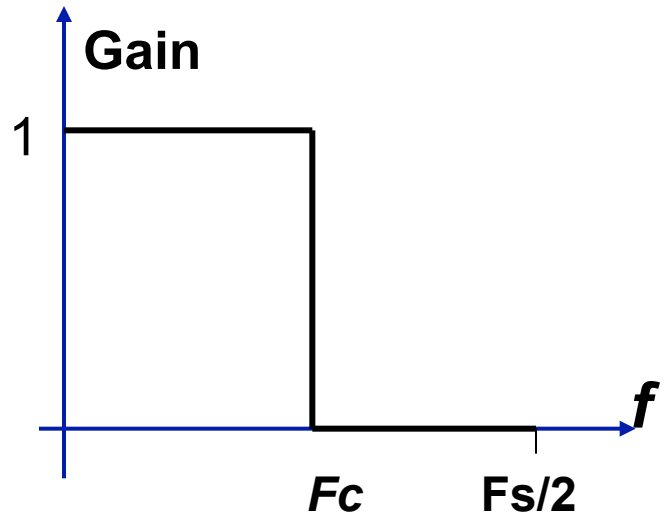
(i) carrying out operations on signal segments stored in vectors
(ii) evaluating the effect of these operations, and
(iii) computing parameters of digital filters etc.

Two commonly used operations are:

- Digital filtering
- Spectral analysis

Digital filtering

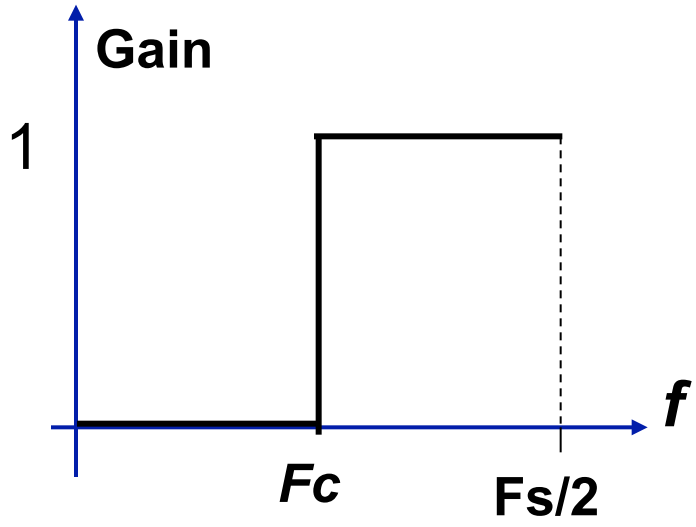
- Low-pass digital filters, high-pass and other categories
- Remove or enhance frequency components of a sampled signal.
- Low-pass filter aims to 'filter out' spectral energy above cut-off frequency Fc, keeping all spectral energy below Fc unchanged.
- Only consider frequencies up to Fs/2 .



'Gain' frequency-response graph for ideal low-pass digital filter.

Ideal high-pass digital filter

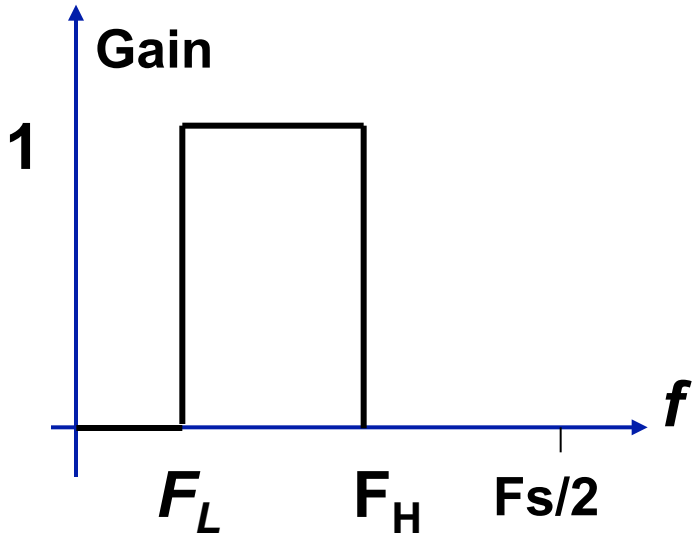
- Aims to 'filter out' spectral energy below cut-off frequency Fc, keeping all spectral energy above Fc unchanged.
- Only consider frequencies up to Fs/2 .



'Gain' frequency-response graph for ideal high-pass digital filter:

Ideal band-pass digital filter

- Aims to 'filter out' spectral energy below frequency F_L, & above F_H keeping all spectral energy between F_L & F_H unchanged.
- Only consider frequencies up to Fs/2 .



'Gain' frequency-response graph for ideal band-pass digital filter:

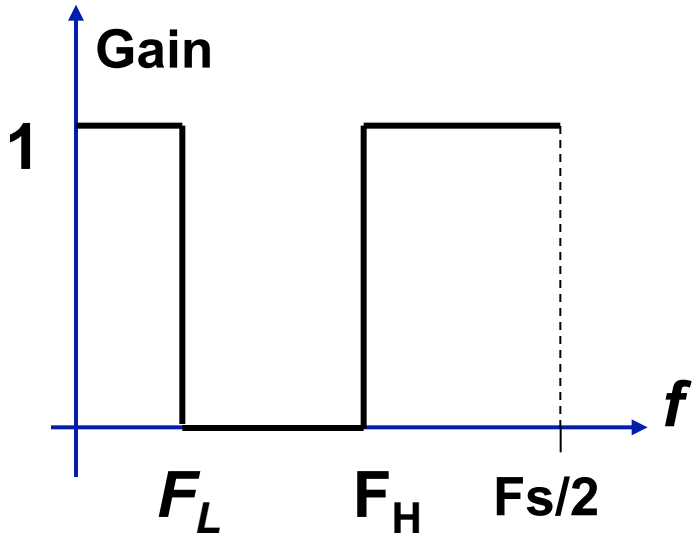
Ideal band-stop digital filter

- Aims to 'filter out' spectral energy between F_L, & F_H keeping all spectral energy below F_L & above F_H unchanged.
- Only consider frequencies up to Fs/2 .

'Gain' frequency-response graph for ideal band-stop digital filter:

Ideal band-stop digital filter

- Aims to 'filter out' spectral energy between F_L, & F_H keeping all spectral energy below F_L & above F_H unchanged.
- Only consider frequencies up to Fs/2 .



'Gain' frequency-response graph for ideal band-stop digital filter:

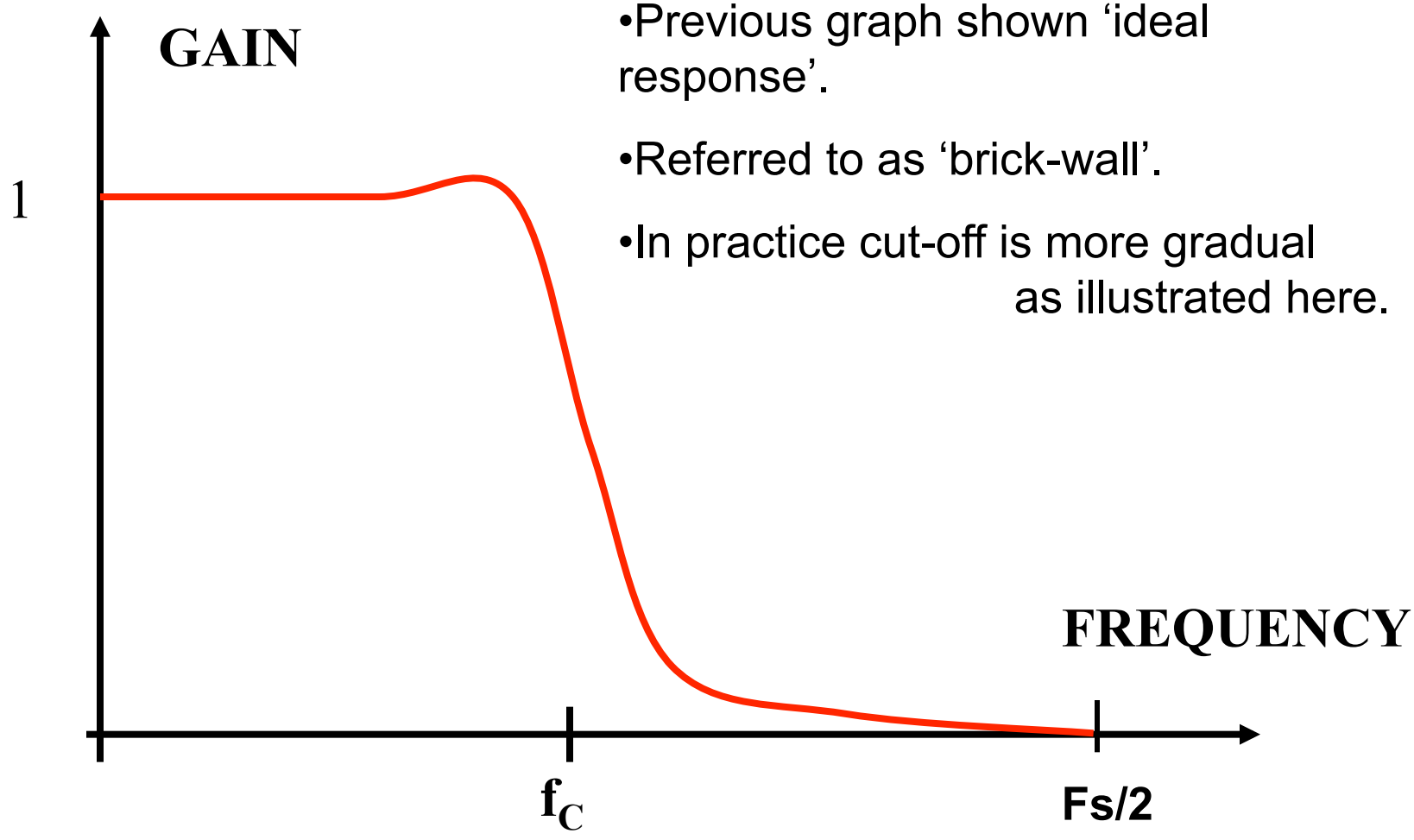
IIR & FIR digital filters

- There are 2 types of digital filter:

  Infinite impulse response (**IIR**) type.

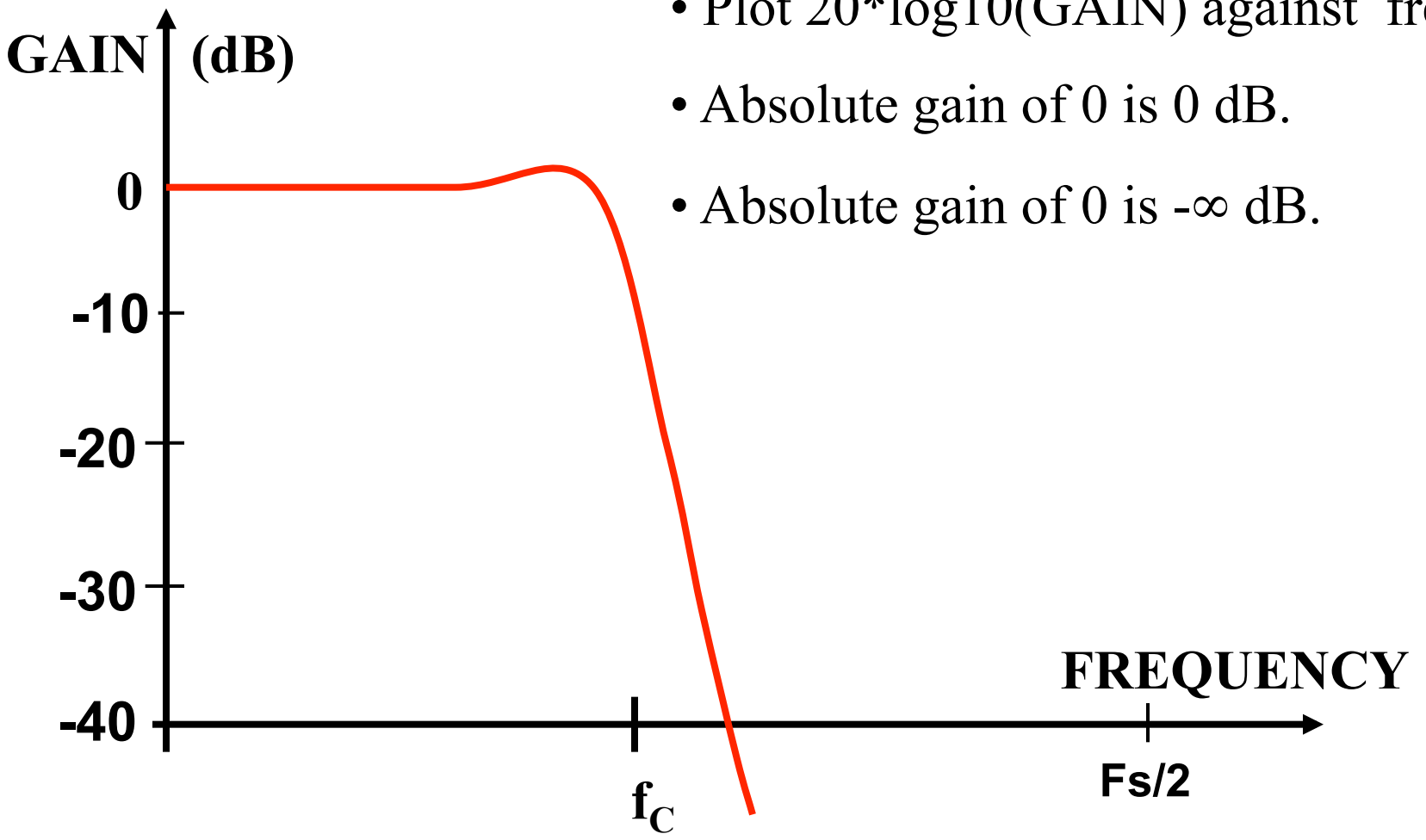
  Finite impulse response (**FIR**) type.
- Either can be low-pass, high-pass, band-pass etc.
- MATLAB can design both types by many different methods.

Gain-response of typical IIR low-pass filter in practice



- Previous graph shown 'ideal response'.
- Referred to as 'brick-wall'.
- In practice cut-off is more gradual as illustrated here.

Same gain-response now in deciBels



- Plot 20*log10(GAIN) against freq
- Absolute gain of 0 is 0 dB.
- Absolute gain of 0 is -∞ dB.

Decibel scale

- Gain is often converted to deciBels (dB)
- Gain in dB = 20 log 10 (Absolute Gain)

dB	Absolute
0	1
6	2
-6	1/2
3	√2
20	10
40	100

Some of the signal processing functions provided

y = filter(a, b, x) ; % Pass x thro' digital filter to produce y.
Coefficients in vectors a and b determine the filter's effect.
For an FIR filter, b=1.

The filter has a 'transfer function' which is:

$$H(z) = \frac{a(1) + a(2)z^{-1} + a(3)z^{-2} + \ldots}{1 + b(2)z^{-1} + b(3)z^{-2} + \ldots}$$

freqz(a,b); % Display gain & phase responses of digital filters:-
% Note the 'zoom' function.

[a,b] = butter(n, f_C/(Fs/2)); % Get a & b for nth order low-pass filter.

More functions for designing digital filters

[a,b] = butter(n, fc/(Fs/2), 'high'); % High-pass Butt IIR.
[a,b] = butter(n, [f_L f_U]/(Fs/2)); % Band-pass Butt IIR.
[a,b] = butter(n, [f_L f_U]/(Fs/2),'stop'); % Band-stop Butt IIR

[a,b] = cheby1(n, Rp,fc/(Fs/2)); % Low-pass Chebychev type 1.
[a,b] = cheby2(n, Rs,..); % Low-pass Chebychev type 2.

a = fir1(n, fc/(Fs/2)); % Get 'a' coeffs of FIR low-pass filter order n.
% Cut-off is fc.

a = fir1(n, (f_L, f_U]/(Fs/2)); % FIR band-pass filter design.
Many more exist.

Specifying the cut-off frequencies

- This is a bit cumbersome.
- If the cut-off is Fc, must enter:

$$Fc / (Fs/2)$$

i.e. divide by half the sampling frequency!

To illustrate for IIR :

- Design 4th order "Butterworth type" IIR low-pass filter with Fc = 1000, Fs=8kHz.

```
[a,b] = butter(4, 1000/4000 ) ;

freqz(a,b); % plot gain & phase responses
```

- Implement it by applying it to signal in array x :

```
y = filter(a,b,x);
```

To illustrate for FIR:

- Design & implement 40^{th} order FIR low-pass digital filter with cut-off frequency 1 kHz
- Apply it to signal in array x.

```
%To design:
  a = fir1(40, 1 / 4 ) ;
  freqz ( a , 1) ; % plot gain & phase

% To implement:
  y = filter(a, 1, x );
```

MATLAB Demo1: Filtering speech

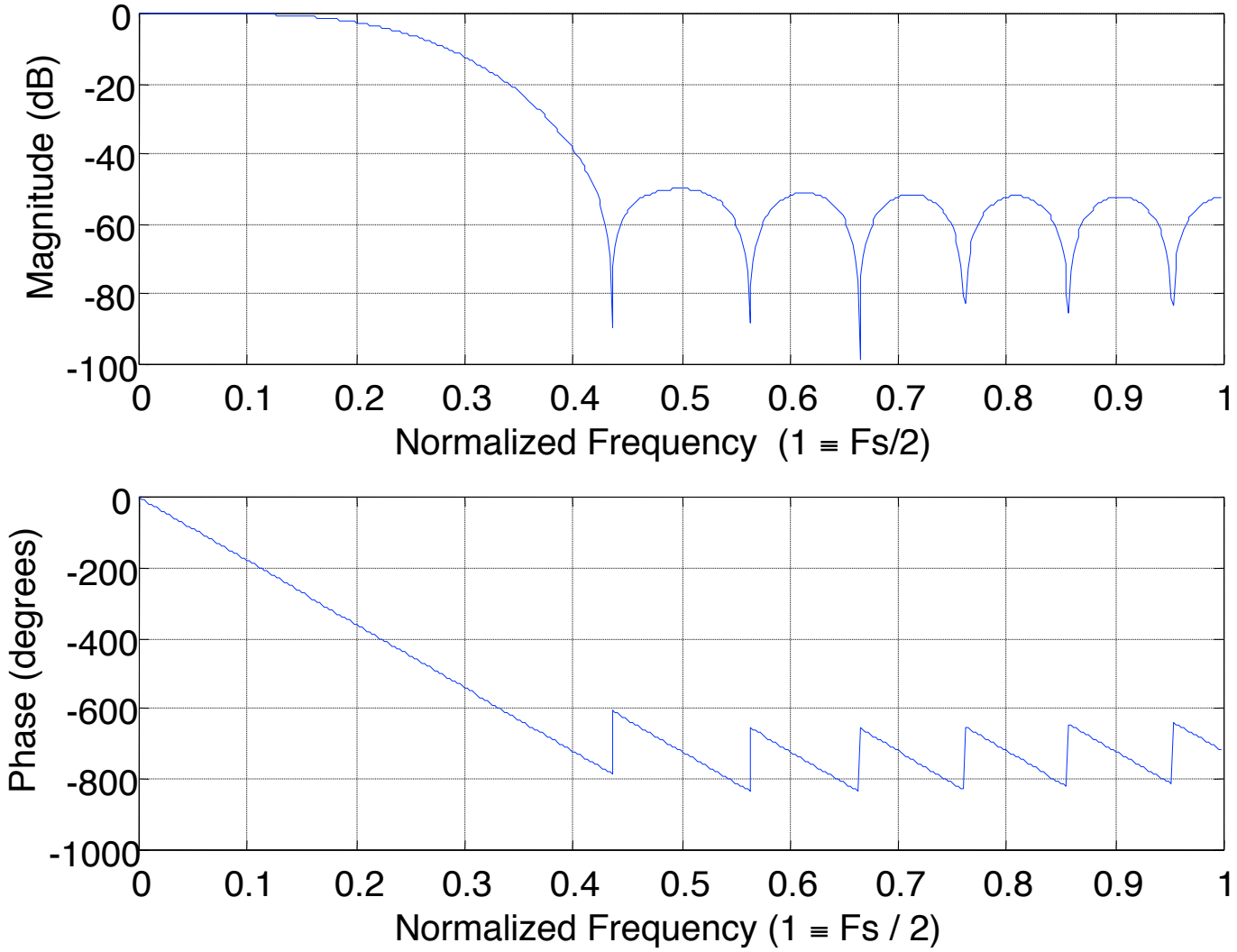
```
clear all;
%Input speech from a file:-
fs = 8000;  % sampling rate in Hz
IFid=fopen('operamp.pcm','rb');
Inspeech = fread(IFid, 'int16');

%Design FIR digital filter:-
fc = 1000;   % cut-off frequency in Hz
[a b] = fir1(20, fc/(0.5*fs) );
freqz(a,b);

%Process speech by filtering:-
Outspeech = filter(a, b, Inspeech);

% Output speech to a file
OFid=fopen('newop.pcm','wb');
fwrite(OFid, Outspeech, 'int16');  fclose('all');
```

Speech file: ‘operamp.pcm’

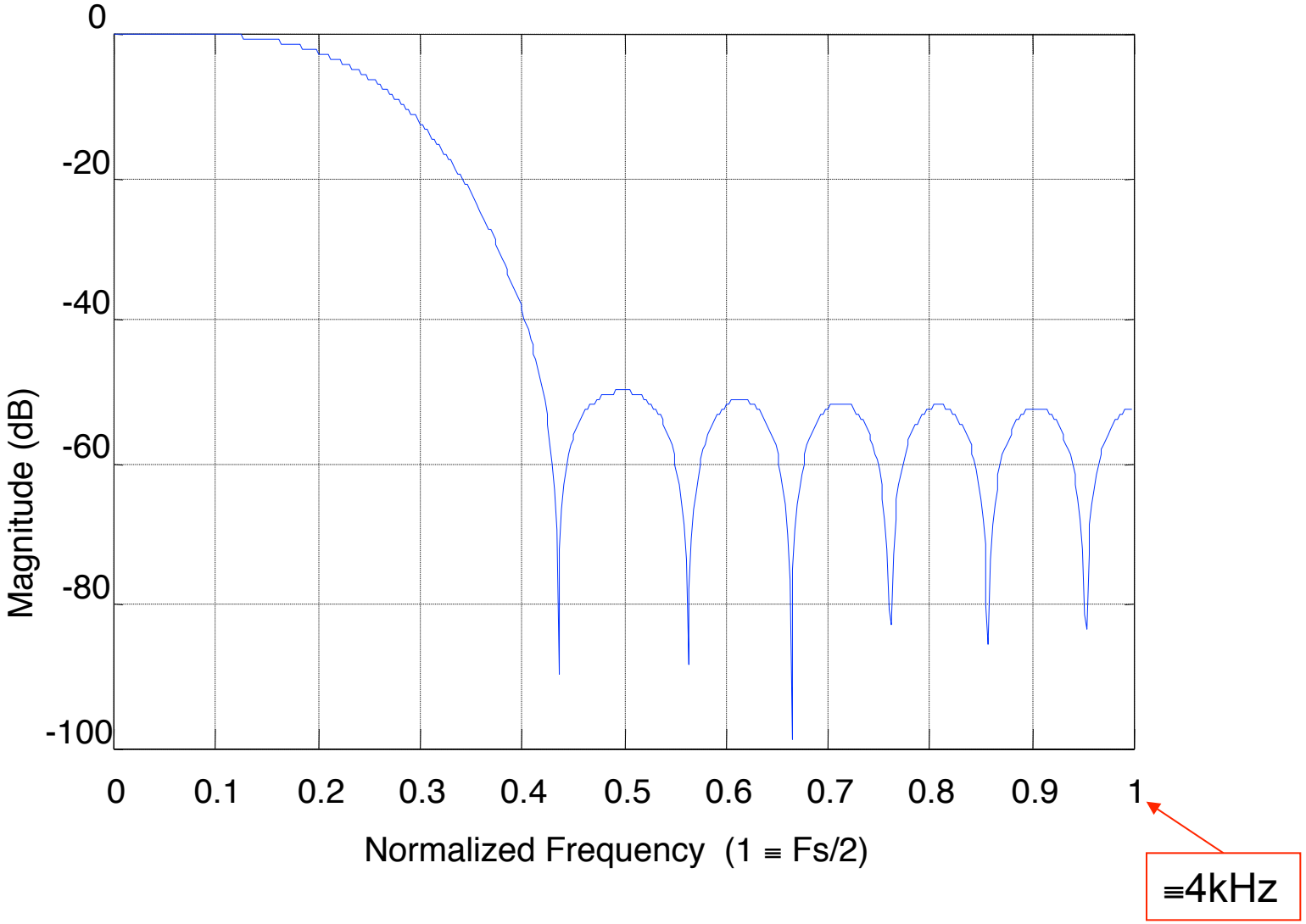
- This may be downloaded from

http://www.kstio.com/dsp/

Graphs produced by ‘freqz’ in previous slide



Gain-response from previous slide



MATLAB Demo2: Listen to speech in a file

- To listen to the filtered speech as produced by MATLAB Demo1, run the following program.

```
IFid=fopen('newop.pcm','rb');
speech = fread(IFid,'int16');
scale = max(abs(speech))+1;
SOUND(speech/scale,8000,16);
fclose('all');
```

- Effect of the lowpass filtering should be to make speech sound 'muffled'
- Higher frequencies have been filtered out.

Spectral analysis

- Fast Fourier Transform (FFT) is famous algorithm for determining frequency content of sampled signals.

- Given a signal segment of ***N*** samples stored in array **x**:

X = fft(x) ;

- gives complex vector **X** of ***N*** samples of frequency spectrum of **x**.

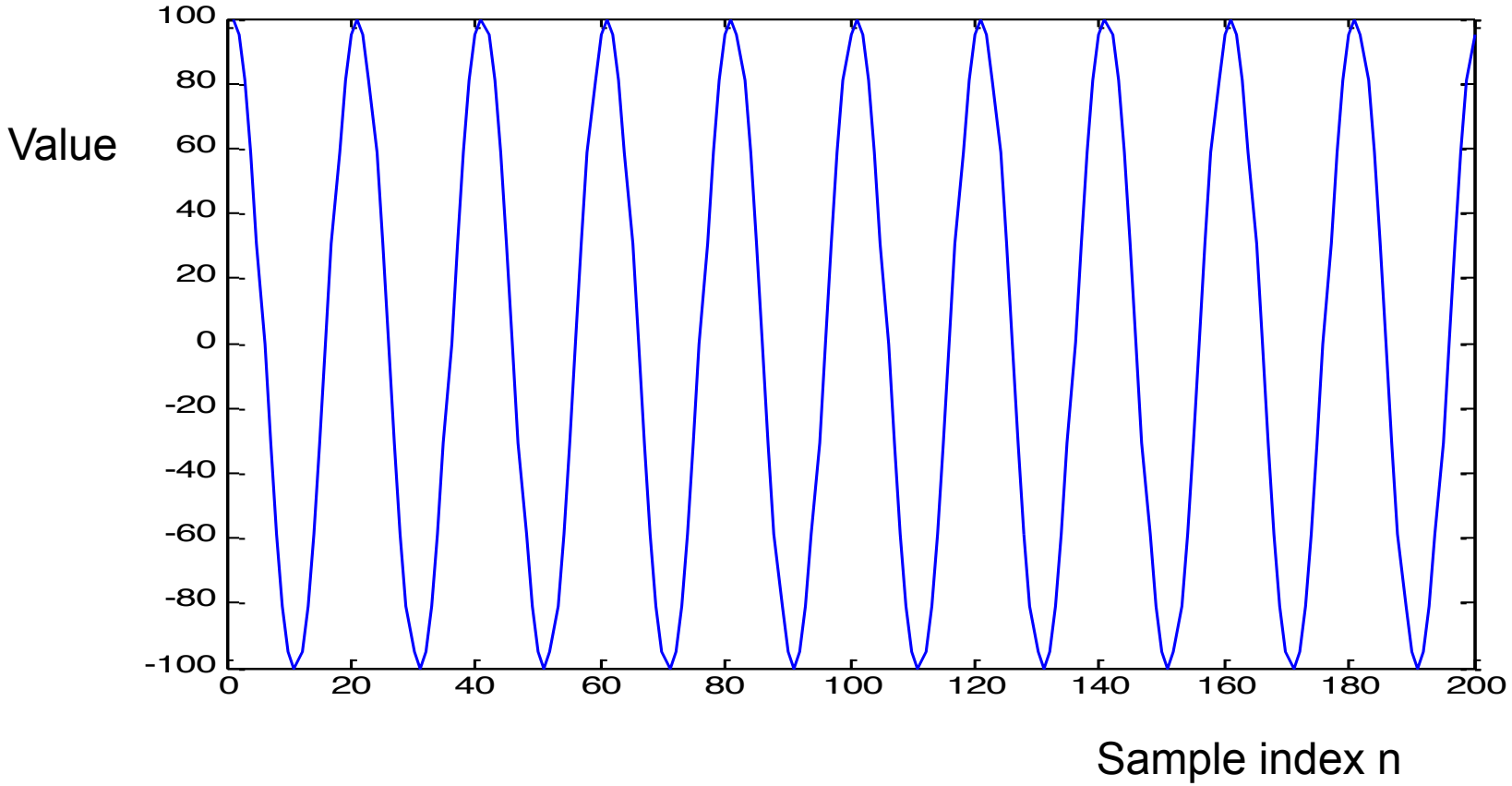
plot (abs(X[1:*N* / 2]) ;

- plots 'magnitude spectrum' of x from **0** Hz to **Fs/2**.

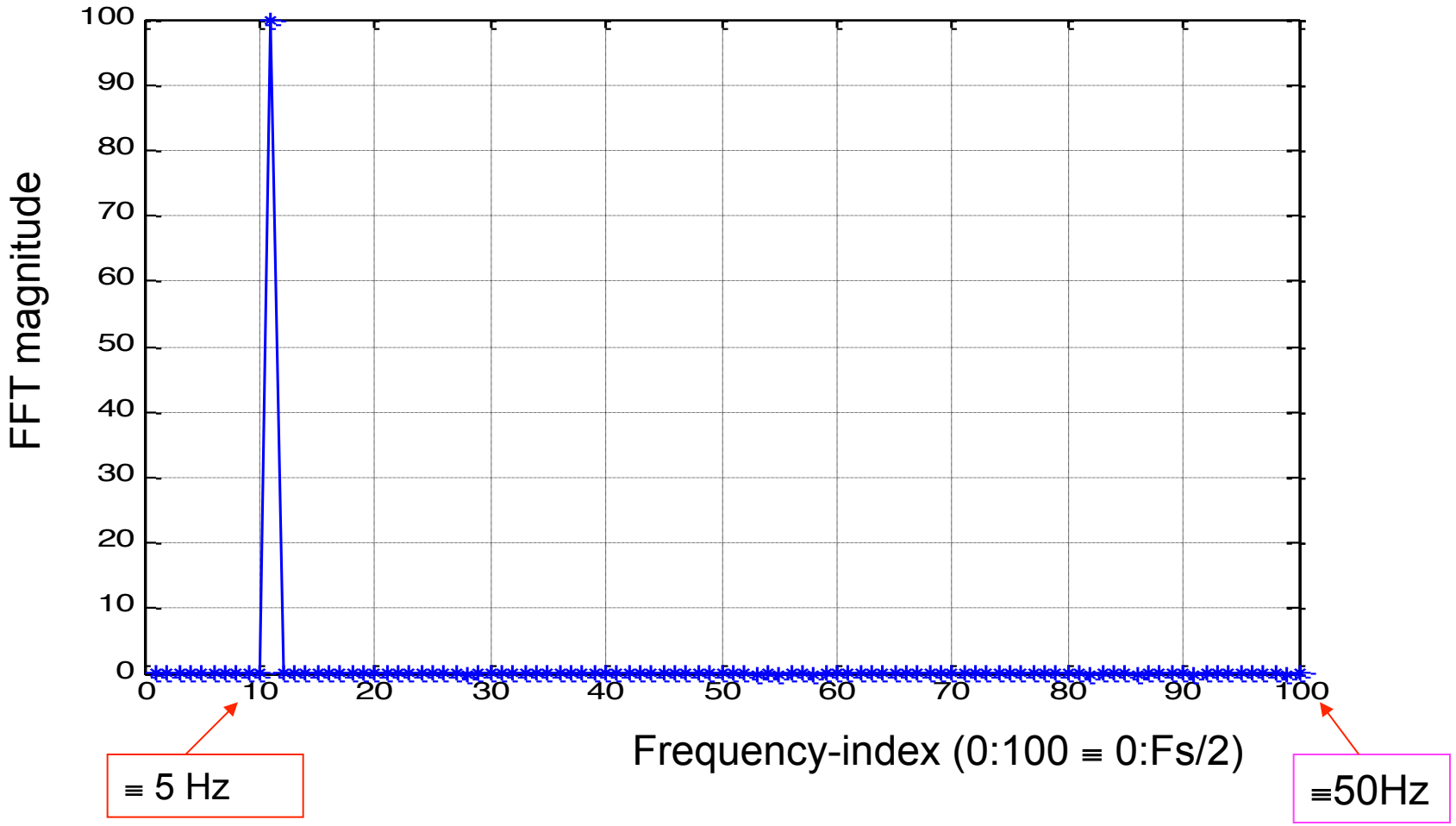
FFT analyse 200 samples of 5Hz sine-wave

```
clear all;  close all;
N=200;
Fs = 100;  % sampling frequency (Hz)
for n=0:N-1
   x(1+n) = 100*cos(2*pi*5*n/Fs) ;
end;
figure(1); plot(x);
X=fft(x)/(N/2);
% Dividing by N/2 gives correct scaling for graph.
figure(2); plot(abs(X(1:N/2)),'*-');
grid on;
```

Plot of 5Hz sine-wave segment sampled at 100 Hz



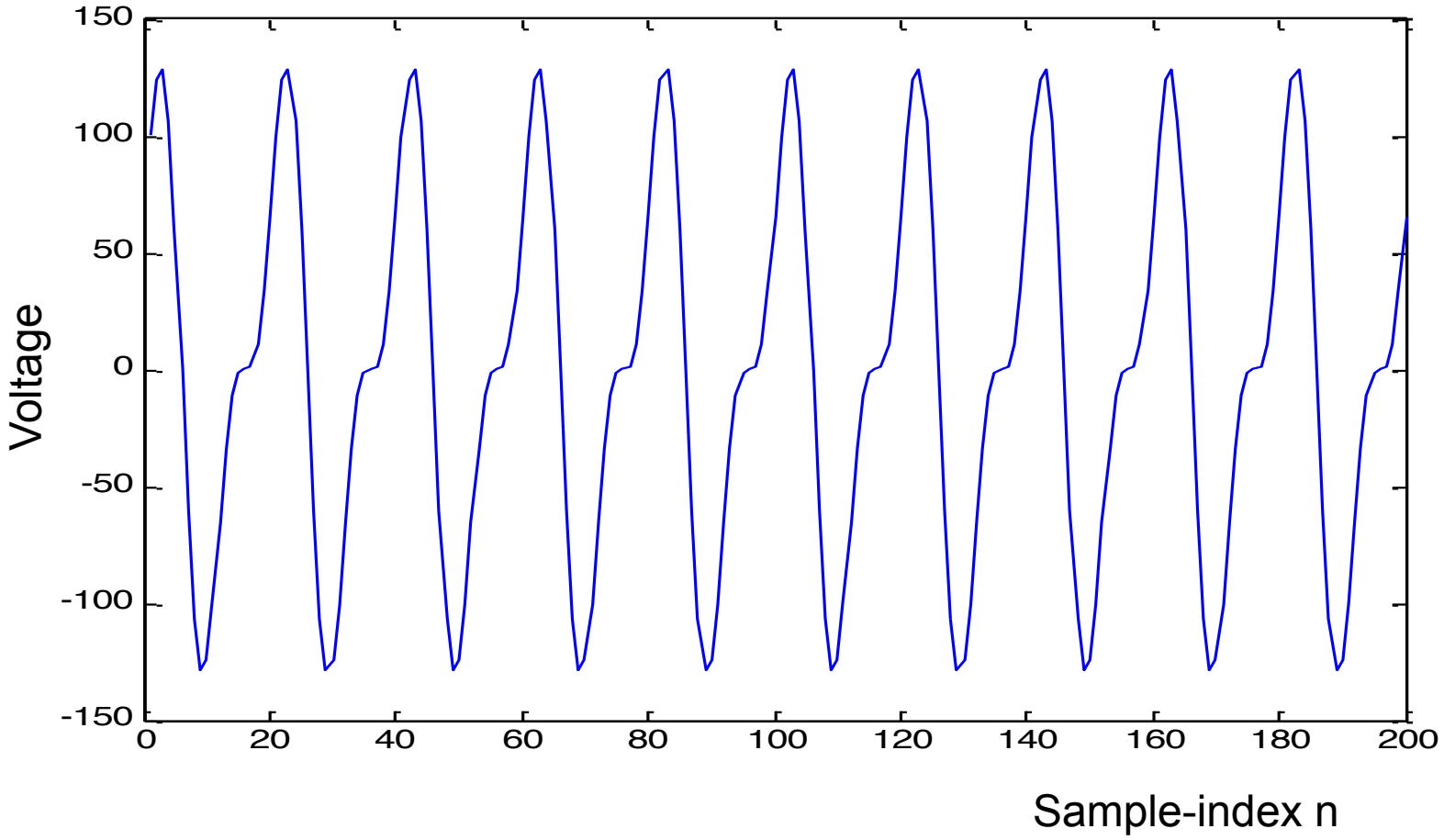
Spectral graph from fft



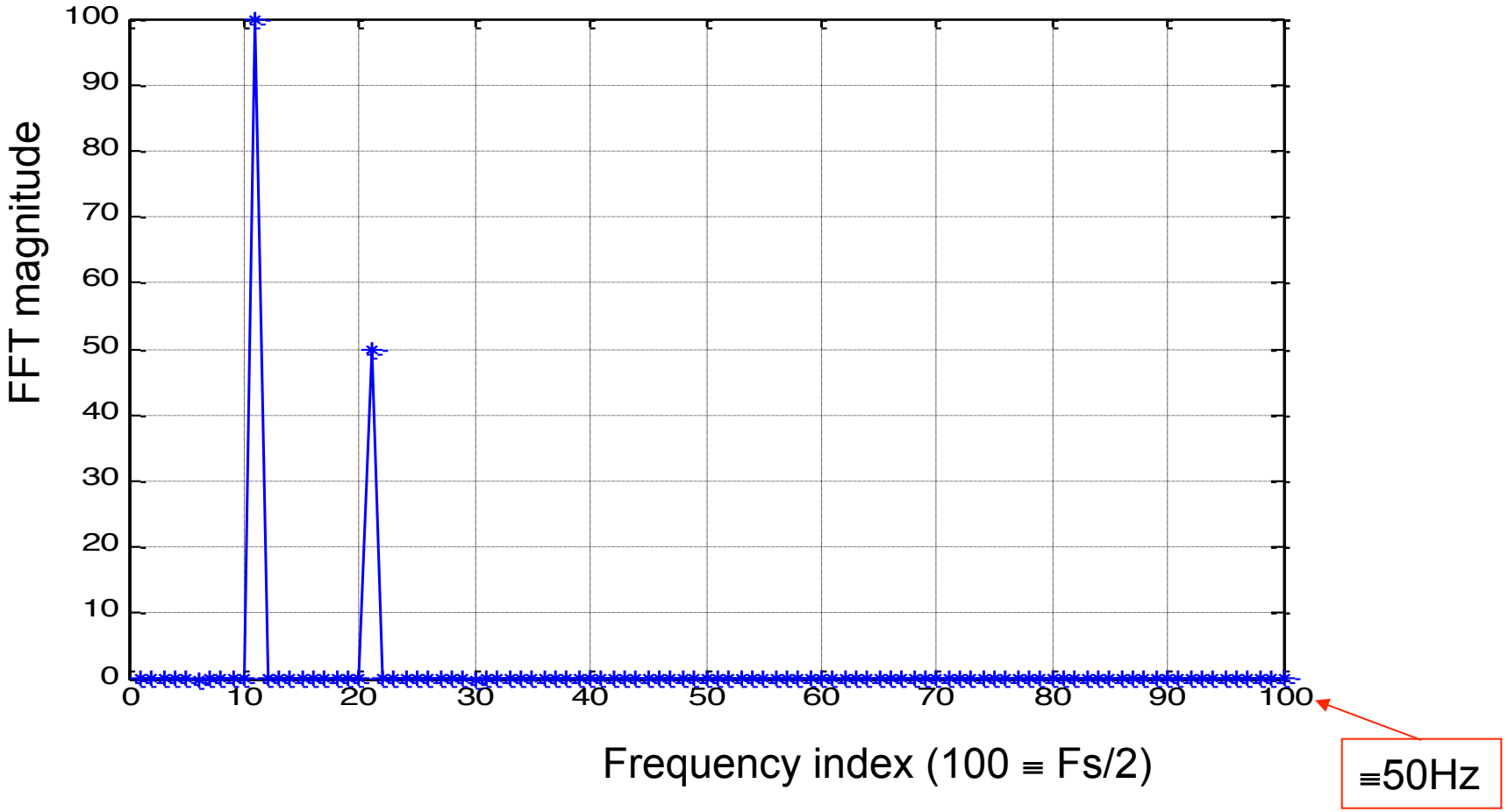
Analyse sum of 2 sine-waves

```
clear all;
N=200;  Fs = 100;
for n=0:N-1
   x(1+n) = 100*cos(2*pi*5*n/Fs) + 50*sin(2*pi*10*n/Fs);
end;
figure(1); plot(x);
X=fft(x)/(N/2);
figure(2); plot(abs(X(1:N/2)),'*-');  grid on;
```

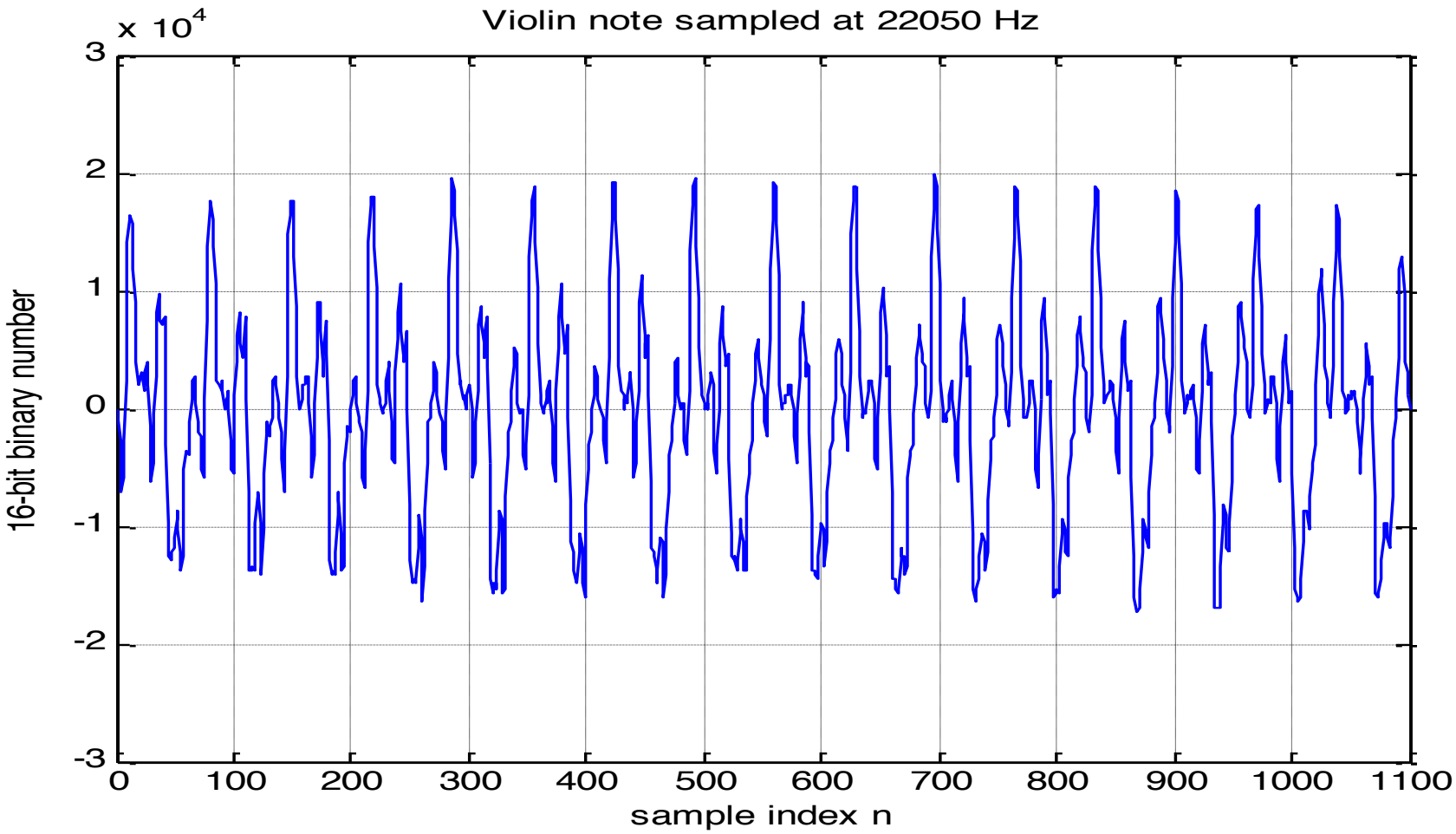
Plot of sum of 2 sine-waves



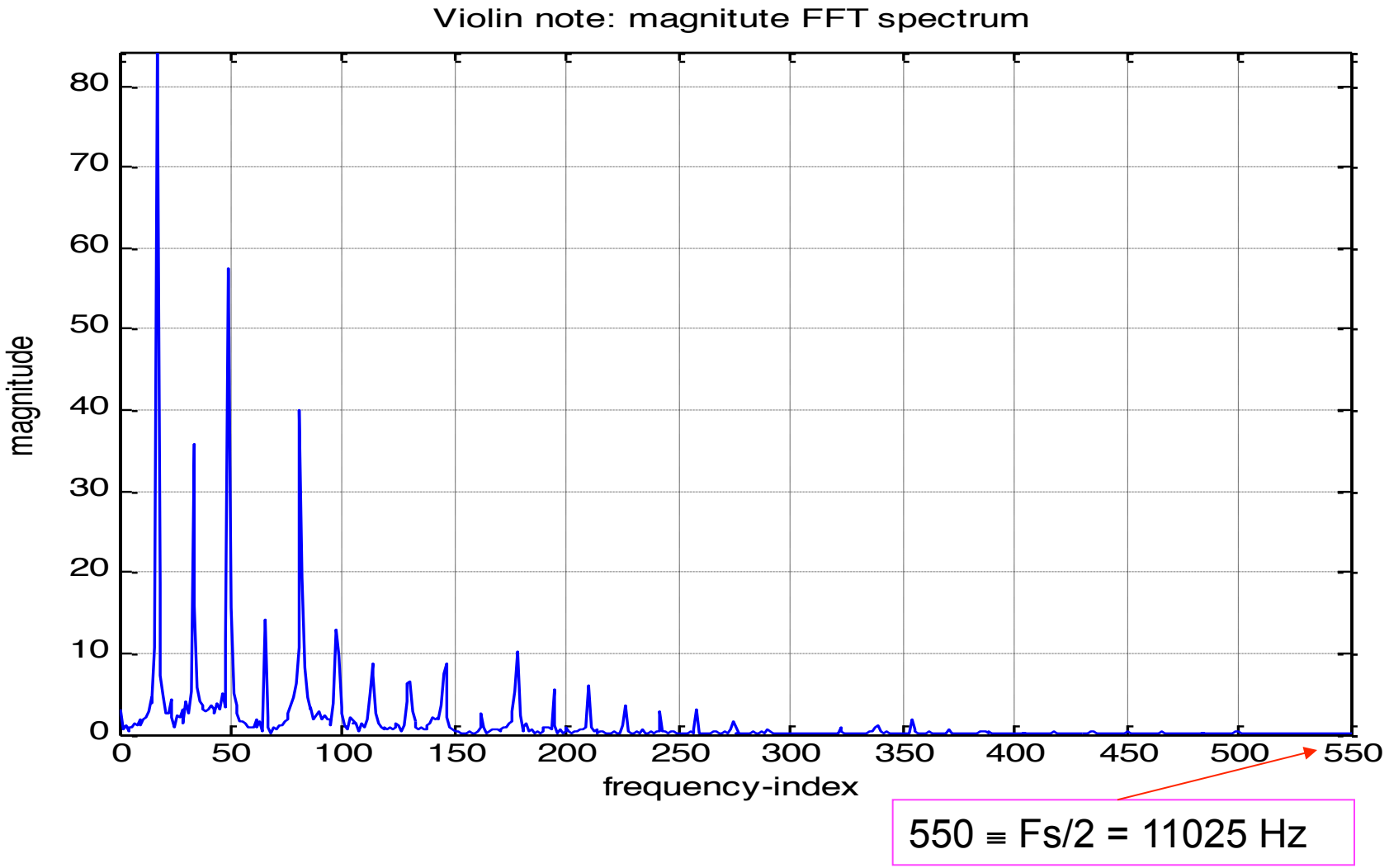
Spectral graph from FFT for 2 sine-waves



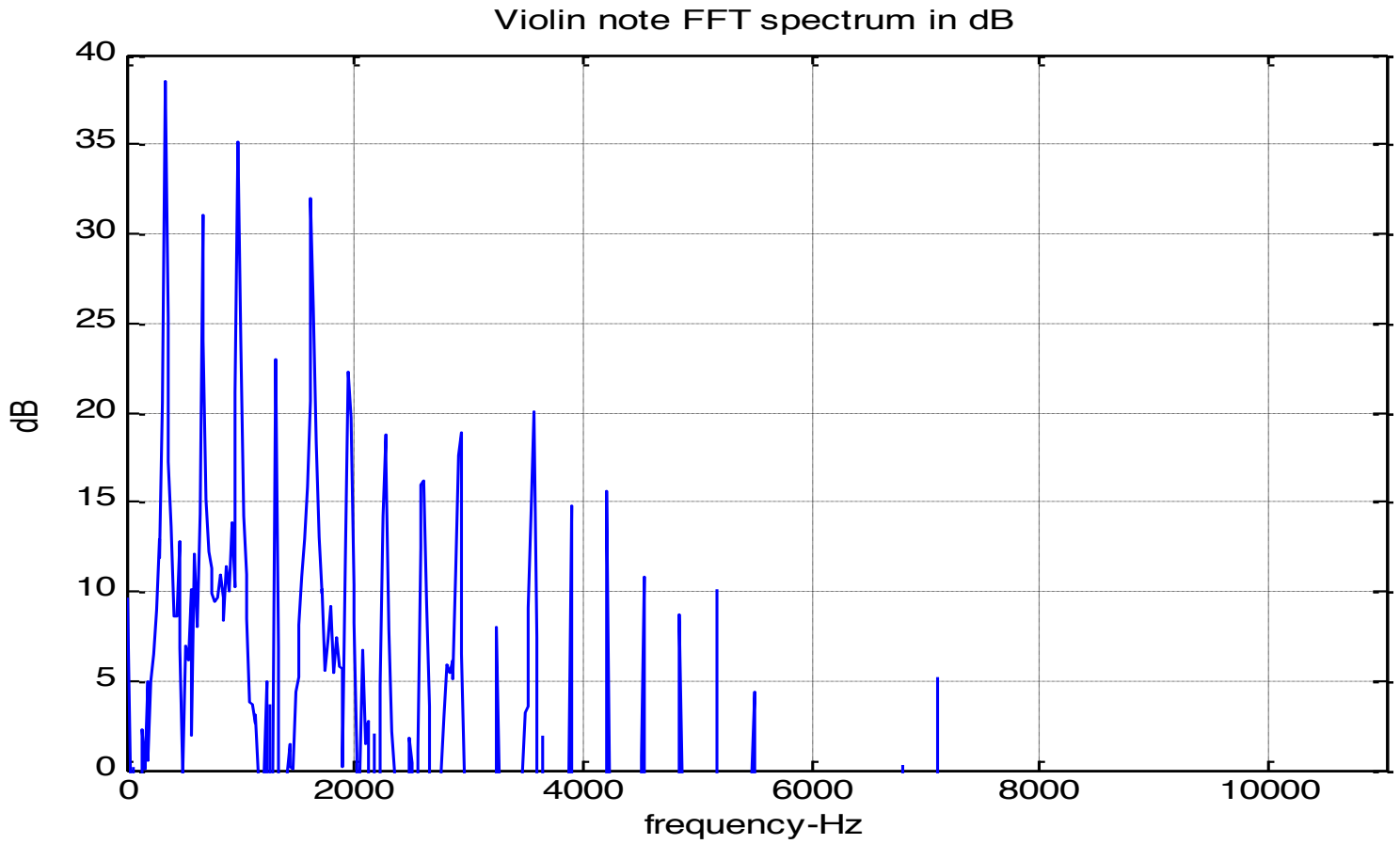
50 ms segment of music sampled at 22.05 kHz



FFT magnitude spectrum of music segment



Same FFT spectrum in dB against frequency - also frequency axis scaled to show Hz



Decimation (down sampling)

- If a signal is sampled at 44.1 kHz, FFT gives spectrum with frequency range 0 to 22.05 kHz.
- You may only want to see spectrum from 0 to 2kHz.
- Down-sampling by a factor 10 makes Fs = 4.41 kHz.
- FFT then gives spectrum in range 0 to 2.205Hz.
- To down-sample **x** by factor 10:

  xd = decimate(**x**, 10);

- Creates new array **xd** of length 1/10 th of that of **x**.
- FFT analyse **xd** rather than **x**.
- 'decimate' works by low-pass filtering then re-sampling.

Understanding decimation

We can easily write our own decimation procedure:

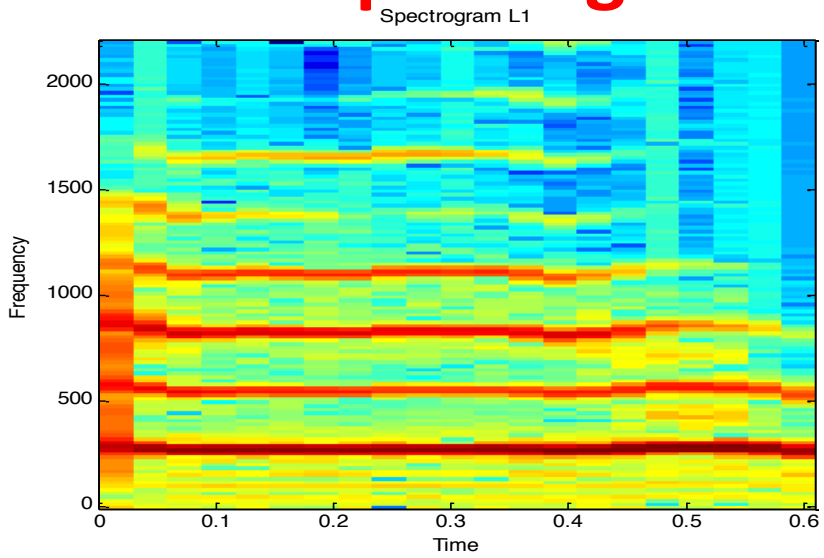
```
[a b] = butter(8, (Fs/24)/(Fs/2) );
xf = filter(a,b,x);  %filter off all components above Fs/20 Hz
% Now re-sample at Fs/10 Hz:
for n=1:length(xf)/10
  xd(n) = xf(n*10);      % Omit 9 out of 10 samples
end;
```

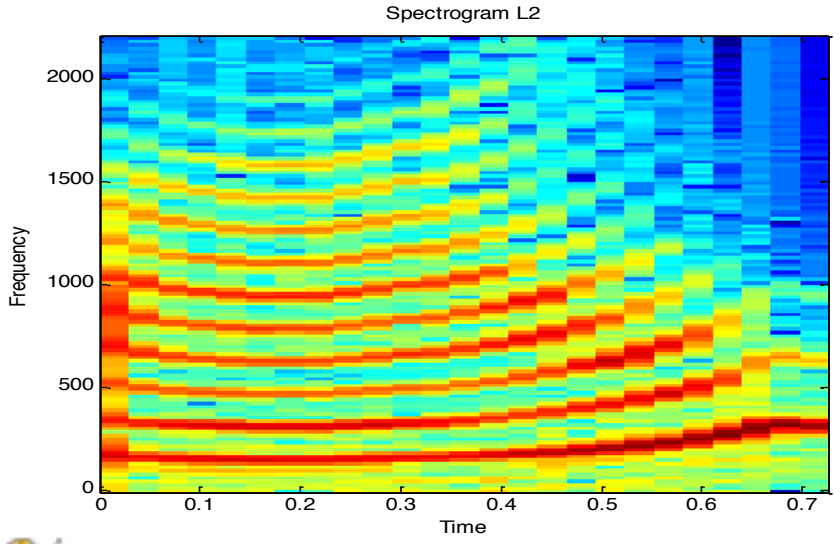
But it's quicker to use the MATLAB function 'decimate'

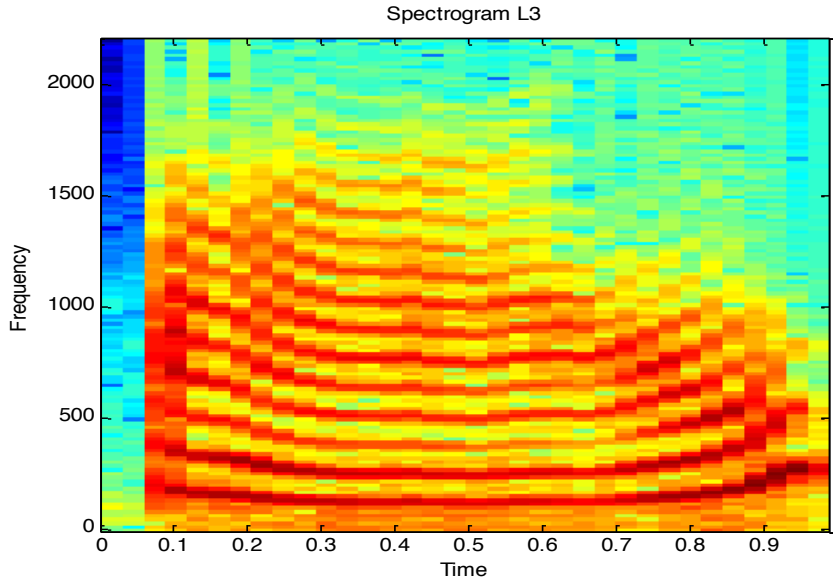
Spectrograms

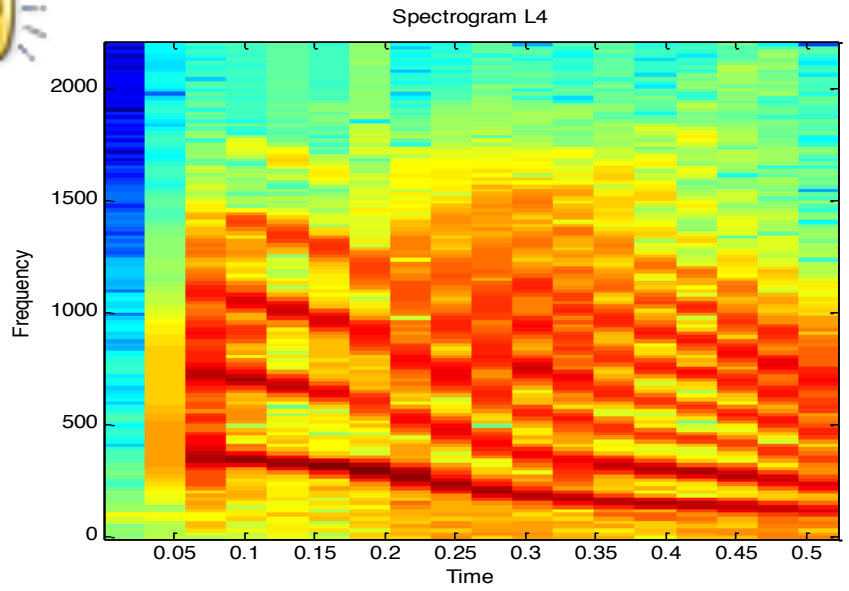
- FFT can be used to analyse ≈20 to 30 ms of speech or music.
- Over longer periods of time, spectral distribution will be changing.
- We often want to analyse this change.
- A related function is '**specgram**'
- Plots a whole succession of FFT spectral graphs showing how the distribution of frequencies is changing over time.
- Result is a 'spectrogram' which plots spectral distribution (using colour) against time.
- Demonstrated on next slide.

Spectrograms for 4 Chinese words









MATLAB demo: Stores sine-wave segment in pcm file

```
clear all;
fs = 8000;  %sampling rate in Hz
T = 1/fs;   % sampling interval (seconds)
% Generate 10000 samples of 500 Hz sine-wave:-
for n=1:10000
   s(n) = 4000 * sin(2 * pi * 500 * n * T);
end;
% Store in a non-formatted (binary) file:-
OFid=fopen('newsin.pcm','wb');
fwrite(OFid, s, 'int16');
fclose('all');
```

Using MATLAB more efficiently

```
for n=1:10000
 Outsin(n) = 2*Insin(n);
end;
```

→ `Outsin = 2*Insin;`

Slow Faster

```
for n=1:10000
 Outsin(n)=abs(Insin(n));
end;
```

→ `Outsin = abs(Insin);`

MATLAB Exercise 1: Modulation

Generate 320 samples of a 50 Hz sine wave sampled at 8kHz & multiply this by a 1kHz sine-wave sampled at 8kHz. Plot resulting waveform

MATLAB Exercise 2: Simulate POTS speech quality

A ‘plain old fashioned telephone’ transmits speech between 300Hz & 3 kHz. Design & implement a digital filter to allow effect of this band-width restriction to be assessed by listening.

MATLAB Exercise 3

- File ‘noisyviolin.wav’ contains music sampled at 22.05 kHz.
- It is corrupted by:

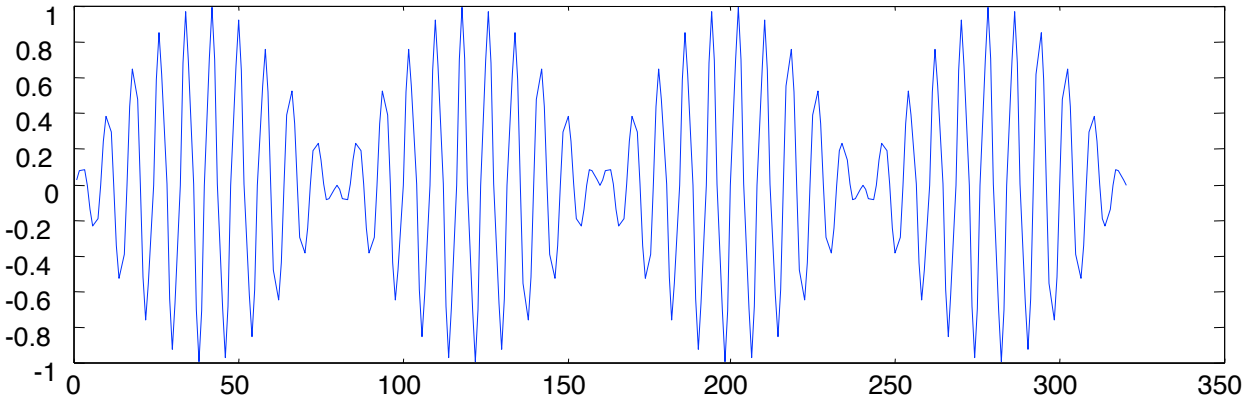
  ‘high-pass’ random noise above 6 kHz

  and a sine-wave around 4 kHz.

- Design & implement 2 digital filters to remove the random noise and the tone.
- Listen to the ‘cleaned up music’

Solution to MATLAB Exercise 1

```
clear all;  fs = 8000;
T=1/fs;
for n=1:320
   s(n) = sin(2*pi*50*n*T);
end;
figure(1); plot (s);
for n=1:320
   smod(n) = s(n)*sin(2*pi*1000*n*T);
end;
figure(2); plot (smod);
```



```
% A solution for MATLAB Exercise 2
clear all;
%Input speech from a file:-
fs = 8000;  % sampling rate in Hz
IFid=fopen('operamp.pcm','rb');
Inspeech = fread(IFid, 'int16');

%Design FIR band-pass digital filter:-
fU = 3000;   % upper cut-off freq Hz
fL  =  300;  % lower cut-off freq
[a b] = fir1(100,[fL/(fs/2) fU/fs/2]);
freqz(a,b);

%Process speech by filtering:-
Outspeech = filter(a, b, Inspeech);

% Output speech to a file
OFid=fopen('newop.pcm','wb');
fwrite(OFid, Outspeech, 'int16');  fclose('all');
```

Solution to MATLAB exercise 3

```
clear all;  close all;
[noisymusic,Fs,bits] = wavread('noisyviolin.wav');
S=size(noisymusic);  disp(sprintf('Size of array = %d, %d', S(1),S(2)));
L=S(1); % length of vector
wavplay(noisymusic,Fs); % Listen to noisy violin
figure(1); plot(noisymusic); title('noisyviolin.wav (mono)');
[a1 b1] = butter(10,(Fs/4)/(Fs/2));  figure(2); freqz(a1,b1);
cleanmusic=filter(a1,b1,noisymusic);   % Low-pass filter
[a2 b2] = butter(4,[3800 4200]/(Fs/2),'stop');  figure(3); freqz(a2,b2);
cleanmusic = filter (a2,b2,cleanmusic);  %Band-stop filter
scale=max(abs(cleanmusic));
wavplay(cleanmusic/scale,Fs);  figure(4); plot(cleanmusic/scale);
```

Fixed point arithmetic

- Real time DSP often uses '**fixed point**' microprocessors since they consume less power & are less expensive than '**floating point**' devices.
- A fixed point processor deals with all numbers as **integers**.
- Numbers often restricted to a word-length of only 16 bits.
- Overflow can occur with disastrous consequences to sound quality.
- If we avoid overflow by scaling numbers to keep them small in amplitude, we lose accuracy as quantisation error is larger proportion of the value.
- Fixed point DSP programming can be quite a difficult task.

Floating point arithmetic

- When we use a PC for non real time DSP, we have **floating point** operations available with word-lengths that are larger than 16 bits.
- This makes the task much easier.
- When simulating fixed point real time DSP on a PC, we can restrict programs to integer arithmetic, & this is useful for testing.
- Can do this with MATLAB

Advantages of digital signal processing

- Signals are generally transmitted & stored in digital form so it makes sense to process them in digital form also.
- DSP systems can be designed & tested in simulation using PCs.
- Accuracy pre-determined by word-length & sampling rate.
- Reproducible: every copy of system will perform identically.
- Characteristics will not drift with temperature or ageing.
- Availability of advanced VLSI technology.
- Systems can be reprogrammed without changing hardware.
- Products can be updated via Internet.
- DSP systems can perform highly complex functions such as
  adaptive filtering
  speech recognition.

Disadvantages of DSP

- Can be expensive especially for high bandwidth signals where fast analog/digital conversion is required.
- Design of DSP systems can be extremely time-consuming & a highly complex and specialised activity.
- Power requirements for digital processing can be high in battery powered portable devices.
- Fixed point processing devices are less power consuming but ability to program such devices is valued & difficult skill.
- There is a shortage of computer science & electrical engineering graduates with the knowledge & skill required.

Summary of Course Objectives

1. Significance of DSP for multi-media, storage and comms.
   Introduction to MATLAB
2. Fundamental concepts: linearity etc
3. Digital filters & their application to sound & images
   Low-pass digital filters etc.
   Butterworth low-pass gain response approximation.
4. FIR & IIR type digital filter design & implementation.
5. MATLAB for design, simulation & implementation of DSP.
6. Real time implementation of DSP (fixed point arith).
7. A/D conversion for DSP
8. FFT and its applications
9. Speech and music compression (CELP, MP3, etc)