Short Time Fourier Transform. Spectrograms.

Mathematical Tools for ITS (11MAI)

Mathematical tools, 2020

Jan Přikryl

11MAI, lecture 4

Monday, October 26, 2020

version: 2020-10-27 14:12

Department of Applied Mathematics, CTU FTS

Lectue Contents

Application of DFT

Computer session 1

DFT of Non-stationary Signals

Windowing and Localization

Short-time Fourier Transform

Spectrograms

Matlab Session 4.1

Consider the analog signal

$$x(t) = 2.0\cos(2\pi\, 5t) + 0.8\sin(2\pi\, 12t) + 0.3\cos(2\pi\, 47t)$$

on the interval $t \in [0, 1)$. Sample this signal with period $T = 1/128\,\mathrm{s}$ and obtain sample vector $\mathbf{x} = (x_0, x_1, x_2, \ldots, x_{127})$.

a) Make MATLAB m-file which plots signals $x(t)$ and $\mathbf{x}$

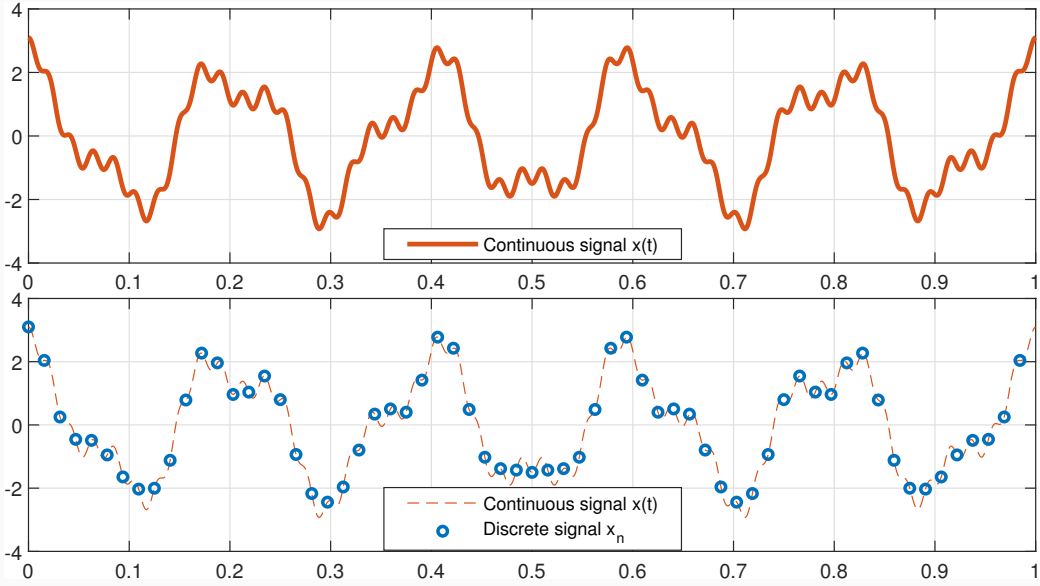
b) Using definition of the DFT find $\mathbf{X}$.

c) Use MATLAB function `fft(x)` to compute DFT of $\mathbf{X}$.

d) Make MATLAB m-file which computes DFT of $\mathbf{x}$ and plots signal and its spectrum.

e) Compute IDFT of the $\mathbf{X}$ and compare it with the original signal $x(t)$.

Matlab Session 4.1 — Input signal plots



Matlab Session 4.1 — Solution (1/3)


```
clear
% The "continuous" original signal
t = linspace(0,1,1001);
x = 2.0*cos(2*pi*5*t) + 0.8*sin(2*pi*12*t) + 0.3*cos(2*pi*47*t);
% The sampled signal
N = 128; % number of samples
ts = linspace(0,1,N+1); % the last sample is at t=1
ts(end) = []; % now we have N time samples
xs = 2.0*cos(2*pi*5*ts) + 0.8*sin(2*pi*12*ts) + 0.3*cos(2*pi*47*ts);
```

Matlab Session 4.1 — Solution (2/3)


```
figure(1);
subplot(2,1,1);
plot(t,x,'LineWidth',2.5,'Color',[1 0 0]);
grid on;
subplot(2,1,2);
plot(ts, xs,'o','LineWidth',2.0,'Color',[0 0 1]);
hold on;
plot(t,x,'--','Color',[1 0 0]);
grid on;
legend('Discrete␣signal␣x(n)','Continuous␣signal␣x(t)');
hold off;
pause
```

Computing **X** using definition of the DFT is not so complicated:

```
X = zeros(N,1);
for k=0:(N-1)
  % Sum of N basis function samples forms X_k
  xk = 0;
  for m=0:(N-1)
    % Note: -1j denotes the imaginary unit
    xk = xk + xs(m+1)*exp(-1j*2*pi*k*m/N); % Matlab indexes start at 1
  end
  X(k+1) = xk;
end
```

Q: How many times will the `xk` update code be executed?

Lectue Contents

Application of DFT

DFT of Non-stationary Signals

Windowing and Localization

Short-time Fourier Transform

Spectrograms

In its original sense, (non)stationarity is a property of stochastic processes:

Definition

A stochastic process is said to be stationary if its unconditional joint probability distribution does not change when shifted in time. Consequently, its parameters such as mean and variance do not change over time.

A signal is an observation of events that correspond to a result of some process. If the properties of the process that generates the events do not change in time, then the process is stationary.

In such a case we (not quite correctly) say that the signal is stationary.

Non-stationary systems $\Leftrightarrow$ differential/difference equations with time-varying coefficients

$$\frac{\mathrm{d}^2}{\mathrm{d}t^2} y(t) - t\, y(t) = 0$$

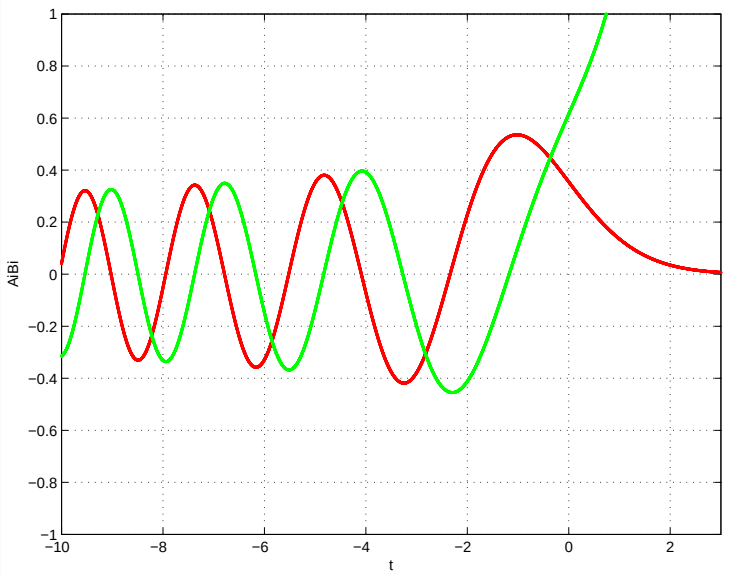
Solution for this particular case is Airy's functions

$$\mathrm{Ai}(t) = \frac{1}{\pi} \int_0^\infty \cos\left(\frac{\tau^3}{3} + t\tau\right) \mathrm{d}\tau$$

$$\mathrm{Bi}(t) = \frac{1}{\pi} \int_0^\infty \left[\exp\left(-\frac{\tau^3}{3} + t\tau\right) + \sin\left(\frac{\tau^3}{3} + t\tau\right)\right] \mathrm{d}\tau$$

Signal $y(t)$ is an output of a non-stationary process/system $\Rightarrow$ we say that the signal is non-stationary.

Example: Non-stationary signal



Stationary systems $\Leftrightarrow$ differential/difference equations with constant coefficients

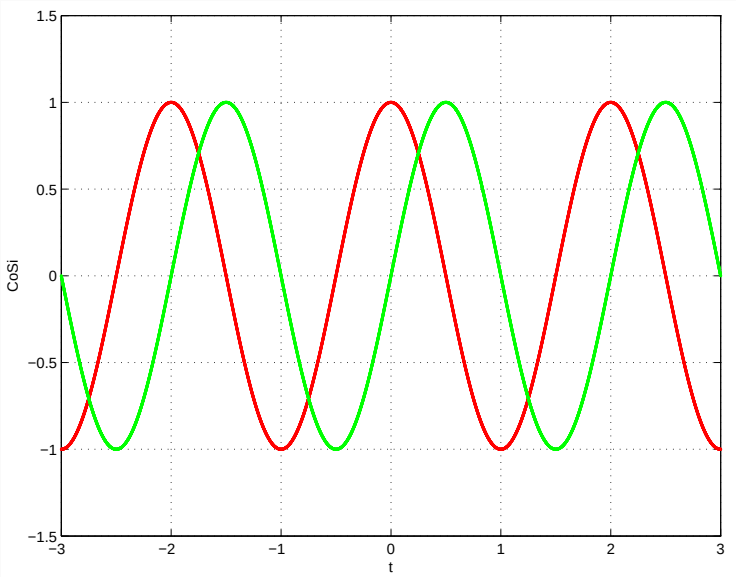
$$\frac{\mathrm{d}^2}{\mathrm{d}t^2} y(t) + \omega_0^2\, y(t) = 0$$

For this particular case the solution is a harmonic wave

$$\cos \omega_0 t,\ \sin \omega_0 t$$

Signal $y(t)$ is an output of a stationary process/system $\Rightarrow$ we say that the signal is stationary.

Example: Stationary signal



A deterministic signal is said to be stationary if it can be written as a discrete sum of cosine waves or exponentials:

$$x(t) = A_0 + \sum_{k=1}^{N} A_k \cos(\omega_k t + \Phi_k)$$

$$x(t) = \sum_{k=-N}^{N} C_k e^{j\omega_k t + \Phi_k}$$

i.e. as a sum of elements which have constant instantaneous amplitude and instantaneous frequency.

In the discrete case, a sequence $(x_n)_n$ assumed to be sampled from an output of a random process is said to be wide-sense stationary (or stationary up to the second order) if its variance is independent of time:

$$\forall m : \mathrm{var}(X_{(m:m+M)}) = E[(x-\mu)^2] = \frac{1}{M-1}\sum_{k=m}^{M}(x_k-\mu)^2 = \sigma^2$$

Here, the population mean μ is always computed over the corresponding slice $m : m + M$.

Stationary and Non-stationary signals

A signal is said to be non-stationary if one of these fundamental assumptions is no longer valid, e.g. either variance σ^2, or autocorrelation $\varrho_{xx}(n, n, m)$, or both are time-varying.

For example:

- a finite duration signal, and in particular a transient signal (for which the length is short compared to the observation duration), is non-stationary.
- speech and EEG are non-stationary signals.
- however, in specific situations may short time sections of EEG be considered stationary.

Nonlocality of DFT

DFT assumes the signal is stationary. It cannot detect local frequency or phase changes.

Example (Frequency hop)

Consider two different periodic signals $f(t)$ and $g(t)$ defined on $0 \leq t < 1$ with frequencies $f_1 = 96\,\text{Hz}$ and $f_2 = 235\,\text{Hz}$ as follows:

- $f(t) = 0.5\,\sin(2\pi f_1 t) + 0.5\,\sin(2\pi f_2 t)$
- $g(t) = \begin{cases} \sin(2\pi f_1 t) & \text{for } 0 \leq t < 0.5, \\ \sin(2\pi f_2 t) & \text{for } 0.5 \leq t < 1.0. \end{cases}$

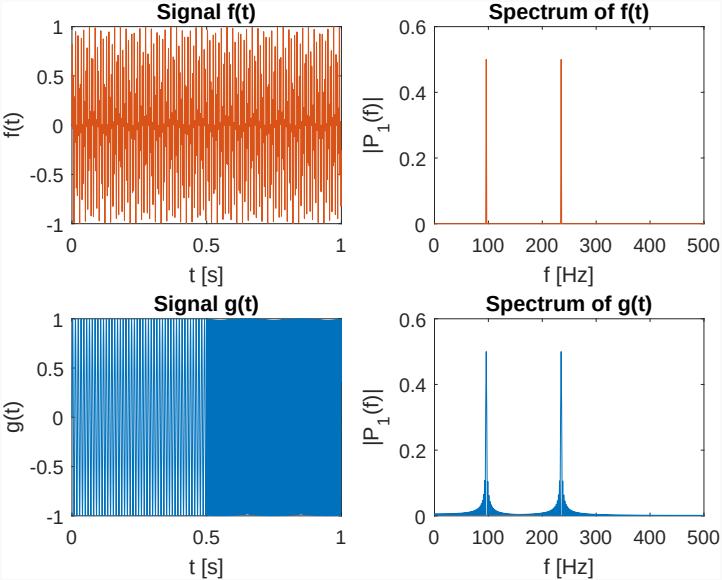
Use the sampling frequency $f_s = 1000\,\text{Hz}$ to produce sample vectors $\mathbf{f}$ and $\mathbf{g}$. Compute the DFT of each sampled signal.

Nonlocality of DFT

Two different signals $f(t)$ and $g(t)$ are constructed with Matlab commands

```
Fs = 1000; % sampling frequency
f1 = 96;
f2 = 235;
t1 = (0:499)/Fs; % time samples for 'g1'
t2 = (500:999)/Fs; % time samples for 'g2'
t = [t1 t2]; % time samples for 'f'
f = 0.5*sin(2*pi*f1*t)+0.5*sin(2*pi*f2*t);
g1 = [sin(2*pi*f1*t1) zeros(1,500)];
g2 = [zeros(1,500) sin(2*pi*f2*t2)];
g = g1+g2;
```

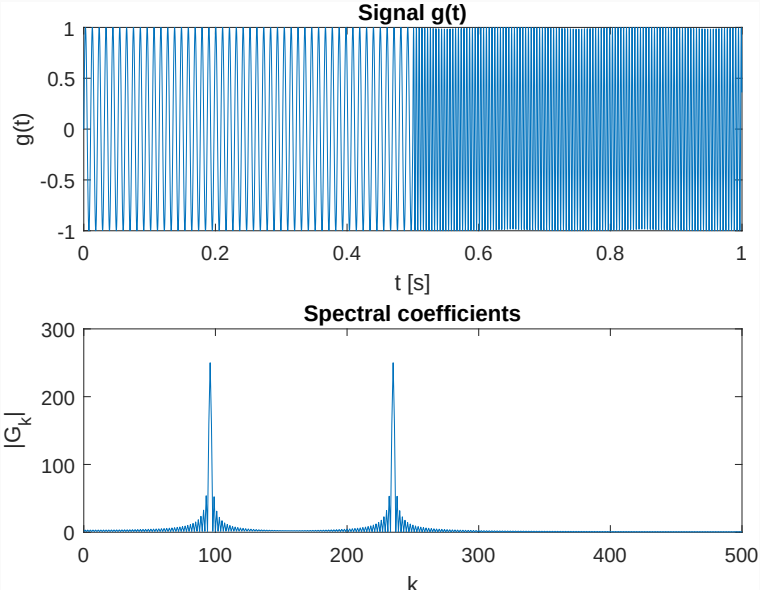
Magnitude of DFT for $f(t)$ and $g(t)$



Nonlocality of DFT

- It is obvious that each signal contains dominant frequencies close to 96 Hz and 235 Hz and the magnitudes are fairly similar.
- But: The signals $f(t)$ and $g(t)$ are quite different in the time domain!
- The example illustrates one of the shortcomings of traditional Fourier transform: nonlocality or global nature of the basis vectors $\mathbf{w}_N$ or its constituting analog waveforms $e^{j2\pi kt/T}$.

Detail of signal $g(t)$



Summary:

- Discontinuities are particularly troublesome.
- The signal $g(t)$ consists of two sinusoids only, but the excitation of several G_ks in frequency domain around the dominant frequencies gives the impression that the entire signal is more oscillatory.
- We would like to have possibility to make the frequency analysis more local — e.g. by analysing smaller portions of the signal.

What is wrong with Fourier Transform?

Consider basis functions $\sin \omega_k t$, $\cos \omega_k t$ and $\delta(t)$ and their support:

support region	in time	in frequency
$\sin \omega_k t$	$(-\infty, \infty)$	0
$\cos \omega_k t$	$(-\infty, \infty)$	0
$\delta(t)$	0	$(-\infty, \infty)$

- The basis functions $\sin \omega_k t$ and $\cos \omega_k t$ cannot localize time!
- The $\delta(t)$ cannot localize frequency!

To localize changes in the signal in time domain by FFT we need to look at shorter parts of the signal — time windows.

DFT of long signals

Computing DFT of long signals is unfeasible:

- When sampling an audio signal at a sampling rate 44.1 kHz, 1 hour of stereophonic music would be $44\,100 \times 2 \times 60 \times 60 = 317\,520\,000$ samples!
- If we want to compute DFT, the closest power-of-two FFT is $2^{28} = 268\,435\,456$ per channel.
- A better approach is to break the long signal into small segments and analyze each one with FFT

These requirements led to development of windowed version of Fourier transform — the short-time Fourier transform, STFT.

Lectue Contents

Application of DFT

DFT of Non-stationary Signals

Windowing and Localization

Computer session 2

Short-time Fourier Transform

Spectrograms

Windowing

Consider a sampled signal $\mathbf{x} \in \mathbb{C}^N$, indexed from 0 to $N-1$. We wish to analyse the frequencies present in $\mathbf{x}$, but only within a certain time range. We choose integers $m \geq 0$ and M such that $m + M \leq N$ and define a vector $\mathbf{w} \in \mathbb{C}^N$ as

$$w_k = \begin{cases} 1 & \text{for } m \leq k \leq m + M - 1 \\ 0 & \text{otherwise} \end{cases}$$

We use $\mathbf{w}$ to define a new vector $\mathbf{y}$ with components

$$y_k = w_k x_k \quad \text{for } 0 \leq k \leq N - 1.$$

We use notation $\mathbf{y} = \mathbf{wx}$ and refer to the vector $\mathbf{w}$ as the (rectangular) window.

Proposition

Let $\mathbf{x}$ *and* $\mathbf{w}$ *be vectors in* $\mathbb{C}^N$ *with discrete Fourier transforms* $\mathbf{X}$ *and* $\mathbf{W}$*, respectively. Let* $\mathbf{y} = \mathbf{wx}$ *have DFT* $\mathbf{Y}$*. Then*

$$\mathbf{Y} = \frac{1}{N}\mathbf{X} * \mathbf{W},$$

where $*$ *is circular convolution in* $\mathbb{C}^N$*.*

Definition (Circular convolution)

The n-th element of an N-point circular convolution of N-periodic vectors $\mathbf{X}$ and $\mathbf{W}$ is

$$Y_n = \frac{1}{N}\sum_{m=0}^{N-1} X_m W_{(n-m) \bmod N}.$$

Definition

When processing a non-stationary signal we assume that the signal is short-time stationary and we perform a Fourier transform on these small blocks — we multiple the signal by a window function that is zero outside the defined "short-time" range.

Main problem: spectral leakage

- high resolution — ability to distinguish close frequencies
- high dynamic range — ability to distinguish frequencies with different amplitudes

Definition (Rectangular window)

The rectangular window is defined as:

$$w_n = \begin{cases} 1 & \text{for } 0 \leq n < N \\ 0 & \text{otherwise} \end{cases}$$

The Matlab command `rectwin(N)` produces the N-point rectangular window.

High resolution $\times$ low dynamic range: good separation of similar amplitudes for similar frequencies, poor at distinguishing far away frequencies of different amplitudes.

Definition (Hamming window)

The most common windowing function in speech analysis is the Hamming window:

$$w_n = \begin{cases} 0.54 + 0.46 \cos\left(\frac{2\pi n}{N-1}\right) & \text{for } 0 \leq n < N \\ 0 & \text{otherwise} \end{cases}$$

Matlab command `hamming(N)` produces the N-point Hamming window.

A frequently used form of Hann window, better dynamic range at the cost of some resolution.

Definition (Blackman window)

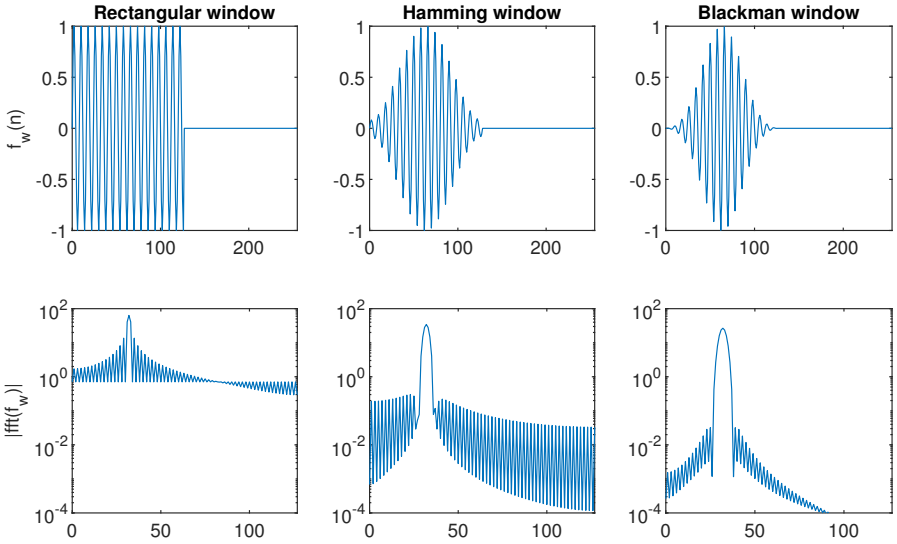
Another common type of window is the Blackman window:

$$w_n = \begin{cases} 0.42 + 0.5\cos\left(\frac{2\pi n}{N-1}\right) + 0.08\cos\left(\frac{4\pi n}{N-1}\right) & \text{for } 0 \leq n < N \\ 0 & \text{otherwise} \end{cases}$$

Use `blackman(N)` to produce the N-point Blackman window.

Better dynamic range than Hamming at the cost of some resolution.

Windowing result



Matlab Session 4.2

Matlab Session 4.2 — Windowing (1/3)

Consider signal $f(t) = \sin(2\pi f_1 t) + 0.4\,\sin(2\pi f_2 t)$ defined on $0 \leq t \leq 1$ with frequencies $f_1 = 137\,\text{Hz}$ and $f_2 = 147\,\text{Hz}$:

a) Use Matlab to sample $f(t)$ at $N = 1000$ points $t_k = \{k/f_s\}_{k=0}^{N}$ with sampling frequency $f_s = 1000\,\text{Hz}$

```
N = 1000; % number of samples
Fs = 1000; % sampling frequency
f1 = 137; % 1. frequency
f2 = 147; % 2. frequency
tk = (0:(N-1))/Fs; % sampling times
f = sin(2*pi*f1*tk) + 0.4*sin(2*pi*f2*tk); % sampled signal
```

b) Compute the DFT of the signal with `F=fft(f)` resp. `F=fft(f,N)`.
Consult the Matlab documentation and explain the difference!

c) Display the magnitude of the Fourier transform with `plot(abs(F(0:501))`

d) Construct a rectangular windowed version of $f(n)$ for window length 200 with

```
fwa = f;
fwa(201:1000) = 0.0;
```

e) Compute the DFT of `fwa` and display the magnitude of the first 501 components.

f) Can you distinguish the two constituent frequencies?
Be careful: is it really obvious that the second frequency is not a side lobe leakage?

g) Construct a windowed version of $f(n)$ of length 200 with

```
fwb = f(1:200);
```

h) Compute the DFT and display the magnitude of the first 101 components.

i) Can you distinguish the two constituent frequencies? Compare the plot of `fwb` with the DFT of `fwa`.

j) Repeat the parts d)–h) using other window lengths such as 300, 100 or 50. How short can the time window be and still allow resolution of the two constituent frequencies?

k) Does it matter whether we treat the windowed signal as a vector of length 1000 as in part 4 or shorter vector as in part 7? Does the side lobe energy confuse the results?

Lectue Contents

Application of DFT

DFT of Non-stationary Signals

Windowing and Localization

Short-time Fourier Transform

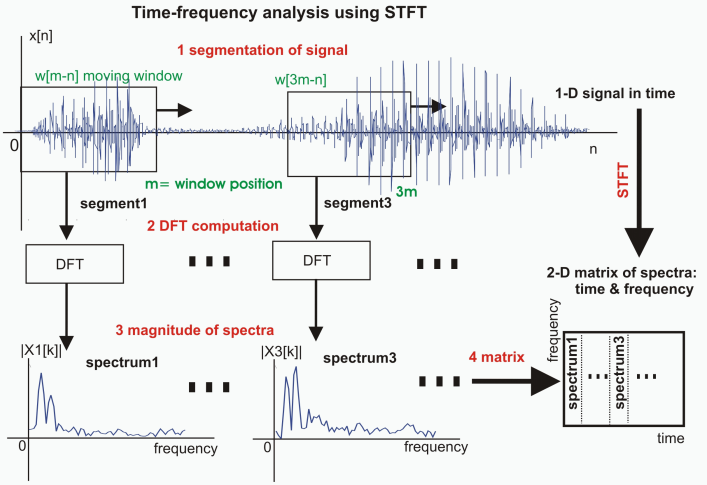
Spectrograms

Short Time Fourier Transform

- Assume that $(x_n)_{n=0}^{\infty}$ is an infinitely long sequence
- In order to localize energy in both time and frequency we segment the signal into short-time pieces and calculate DFT of each segment
- Sampled STFT for a window $(w_m)_{m=0}^{M-1}$ defined in the region $0 \leq m \leq M-1$ is given by

$$X_{k,\ell} = \sum_{m=0}^{M-1} w_m \cdot x_{\ell-m}\, \mathrm{e}^{-\mathrm{j}2\pi \frac{km}{N}}$$

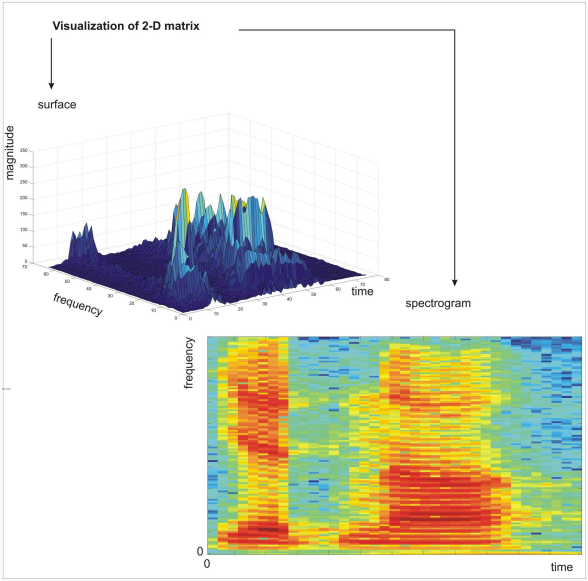
Short Time Fourier Transform



STFT = signal segmentation using window & repeatedly used FT (or DFT)

visualisation of matrix: surface or spectrogram (next slide)

Short Time Fourier Transform



Lectue Contents

Application of DFT

DFT of Non-stationary Signals

Windowing and Localization

Short-time Fourier Transform

Spectrograms

Computer session 3

In MATLAB the command

```
spectrogram(x,window,noverlap,nfft,fs,'yaxis')
```

performs short-time Fourier transform and plots a 2D frequency-time diagram, where

- `x` is the signal specified by vector x.
- if `window` is an integer, `x` is divided into segments of length equal to that integer value
- otherwise, `window` is a Hamming window of length `nfft`
- `noverlap` is the number of samples each segment of `x` overlaps

In MATLAB the command

```
spectrogram(x,window,noverlap,nfft,fs,'yaxis')
```

performs short-time Fourier transform and plots a 2D frequency-time diagram, where

- `x` is the signal specified by vector x.
- if `window` is an integer, `x` is divided into segments of length equal to that integer value
- otherwise, `window` is a Hamming window of length `nfft`
- `noverlap` is the number of samples each segment of `x` overlaps

In MATLAB the command

```
spectrogram(x,window,noverlap,nfft,fs,'yaxis')
```

performs short-time Fourier transform and plots a 2D frequency-time diagram, where

- `x` is the signal specified by vector x.
- if `window` is an integer, `x` is divided into segments of length equal to that integer value
- otherwise, `window` is a Hamming window of length `nfft`
- `noverlap` is the number of samples each segment of `x` overlaps

In MATLAB the command

```
spectrogram(x,window,noverlap,nfft,fs,'yaxis')
```

performs short-time Fourier transform and plots a 2D frequency-time diagram, where

- `x` is the signal specified by vector x.
- if `window` is an integer, `x` is divided into segments of length equal to that integer value
- otherwise, `window` is a Hamming window of length `nfft`
- `noverlap` is the number of samples each segment of `x` overlaps

In MATLAB the command

```
spectrogram(x,window,noverlap,nfft,fs,'yaxis')
```

performs short-time Fourier transform and plots a 2D frequency-time diagram, where

- `nfft` is the FFT length and is the maximum of 256 or the next power of 2 greater than the length of each segment of `x`
- `fs` is the sampling frequency, which defaults to normalized frequency
- using `'yaxis'` displays frequency on the y-axis and time on the x-axis

In MATLAB the command

```
spectrogram(x,window,noverlap,nfft,fs,'yaxis')
```

performs short-time Fourier transform and plots a 2D frequency-time diagram, where

- `nfft` is the FFT length and is the maximum of 256 or the next power of 2 greater than the length of each segment of `x`
- `fs` is the sampling frequency, which defaults to normalized frequency
- using `'yaxis'` displays frequency on the y-axis and time on the x-axis

In MATLAB the command

```
spectrogram(x,window,noverlap,nfft,fs,'yaxis')
```

performs short-time Fourier transform and plots a 2D frequency-time diagram, where

- `nfft` is the FFT length and is the maximum of 256 or the next power of 2 greater than the length of each segment of `x`
- `fs` is the sampling frequency, which defaults to normalized frequency
- using `'yaxis'` displays frequency on the y-axis and time on the x-axis

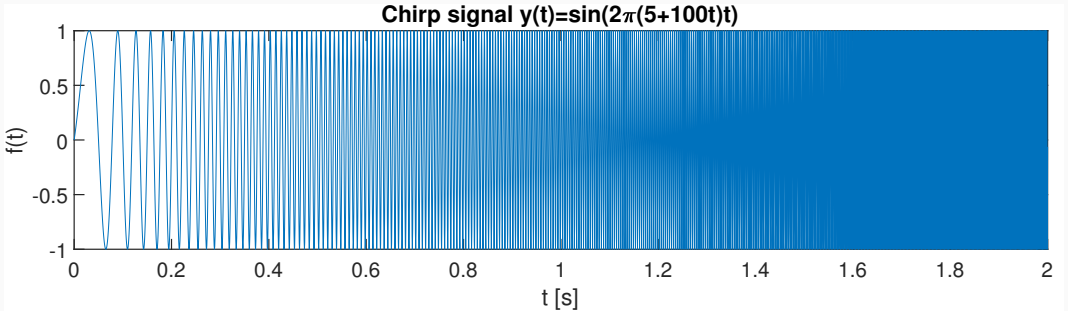
In MATLAB the command

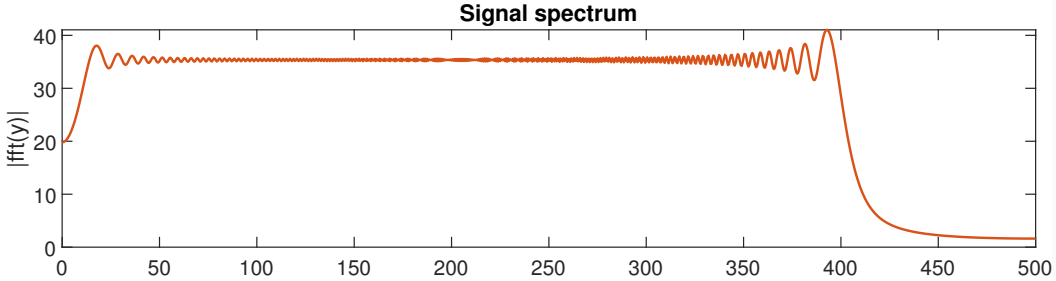
```
spectrogram(x,window,noverlap,nfft,fs,'yaxis')
```

performs short-time Fourier transform and plots a 2D frequency-time diagram.

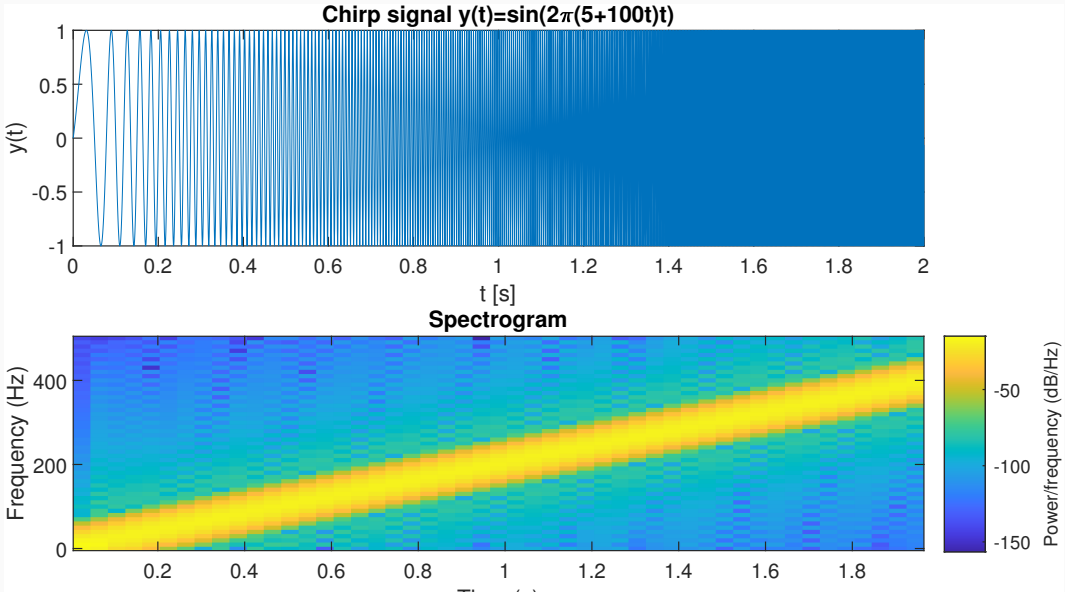
In current Matlab versions, the `colorbar` command is automatically issued to append a color scale to the current axes.

DFT — Chirp signal analysis $\sin(2\pi(f_0 + \alpha t)t)$

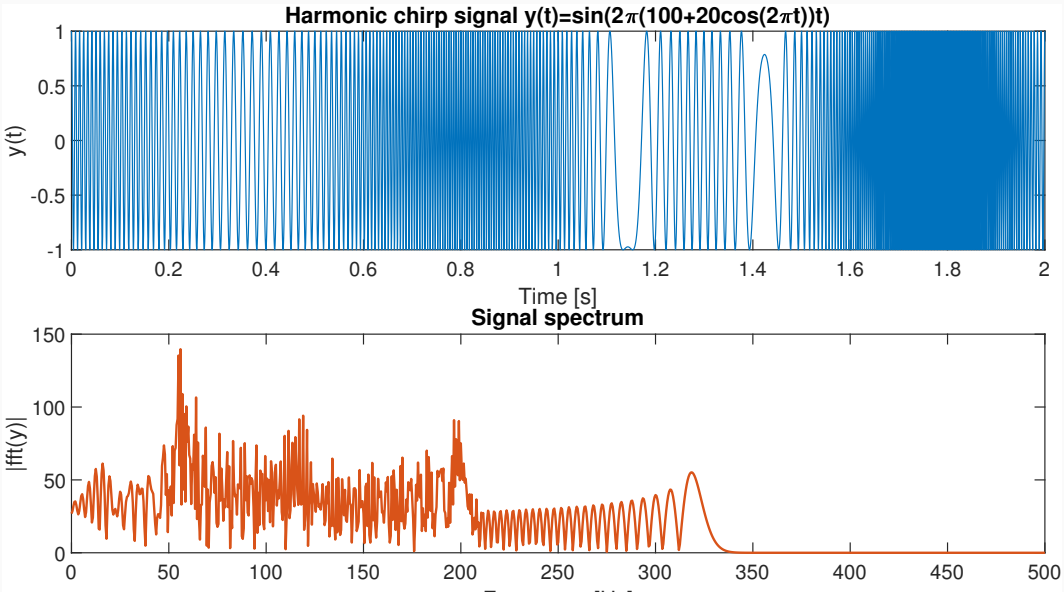




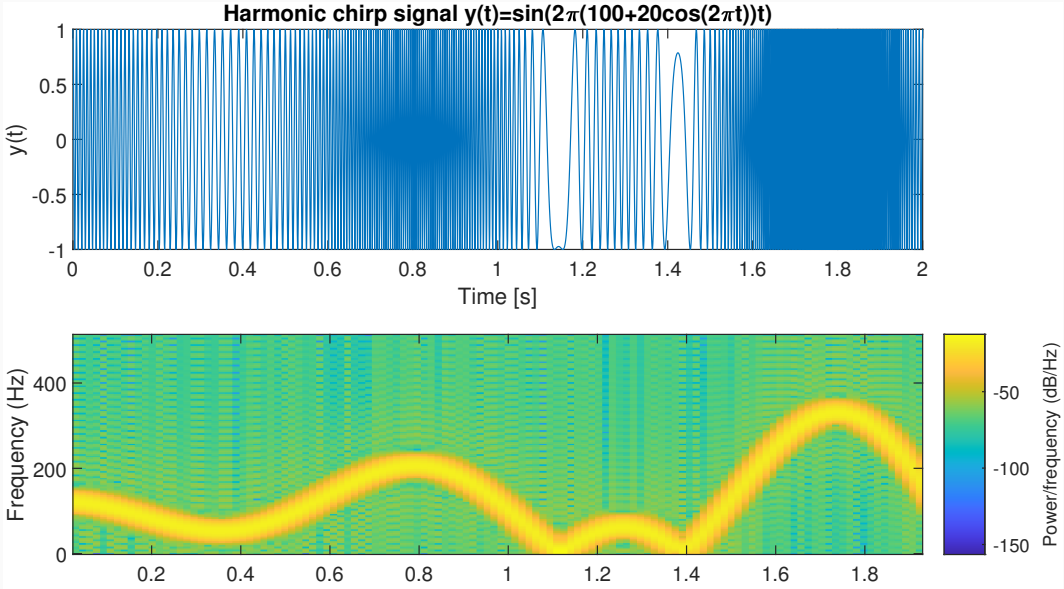
STFT — Chirp signal analysis $\sin(2\pi(f_0 + \alpha t)t)$



DFT — Analysis of $\cos(2\pi(100 + 20\cos 2\pi t)t)$



STFT — Analysis of $\cos(2\pi(100 + 20\cos 2\pi t)t)$



Matlab Session 4.3

a) Download the soundfile `flute-C4.wav` from 11MAI website

b) Start MATLAB. Load in the downloaded audio signal with commands

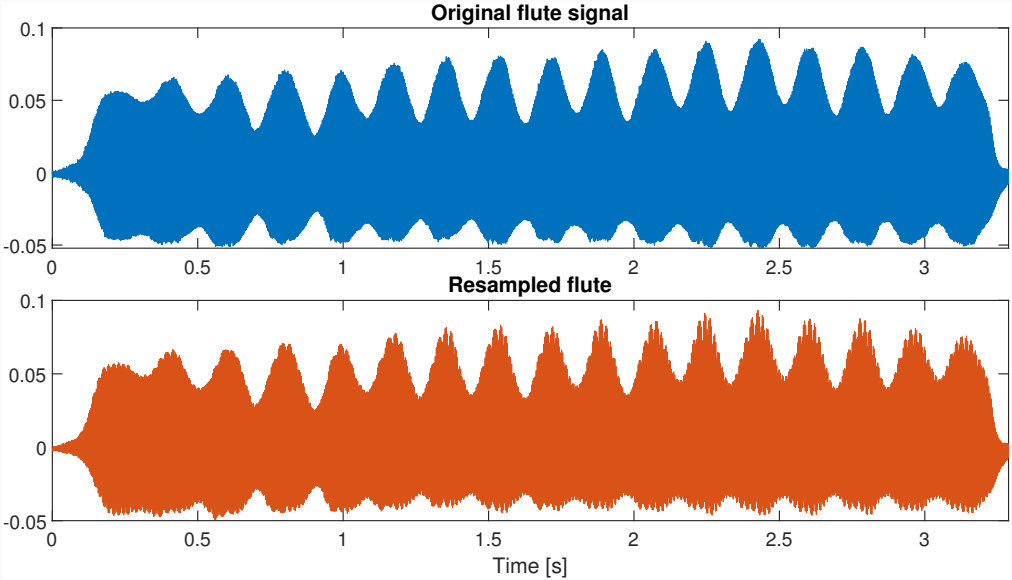
```
filename = 'flute-C4.wav';
[x1 sr1] = audioread(filename);
```

c) The sampling rate is 11 025 Hz, and the signal contains 36 250 samples.
Q: If we consider this signal as sampled on an interval $[0, T)$, what is the time duration of the flute sound ?

d) Use command `soundsc(x1,sr1)` to obtain flute sound click to play

e) Resample the audio signal by $f_r = 4000$ Hz and write the sound file to disk using

```
audiowrite('flute-resampled.wav', x2, sr2);
```


Original flute signal
0.1
0.05
0
-0.05
0
0.5
1
1.5
2
2.5
3
Resampled flute
0.1
0.05
0
-0.05
0
0.5
1
1.5
2
2.5
3
Time [s]

f) Compute the DFT of the signal with

```
X1 = fft(x1(1:1024));
X2 = fft(x2(1:1024));
```

g) DFT of real-valued signals is always symmetric around `fr/2` so we only need to plot the first half. Display the magnitude of the Fourier transform using

```
plot(f1(1:end/2+1), abs(X1(1:end/2+1)));
```

h) **Q**: What is the approximate fundamental frequency of the flute note C4?

i) Compute the DFT of the signal with

```
X1 = fft(x1(1:1024));
X2 = fft(x2(1:1024));
```

j) DFT of real-valued signals is always symmetric around `fr/2` so we only need to plot the first half. Display the magnitude of the Fourier transform using

```
plot(f1(1:end/2+1), abs(X1(1:end/2+1)));
```

k) **Q:** What is the approximate fundamental frequency of the flute note C4?
A: Find the bin corresponding to the first peak in the magnitude spectrum.

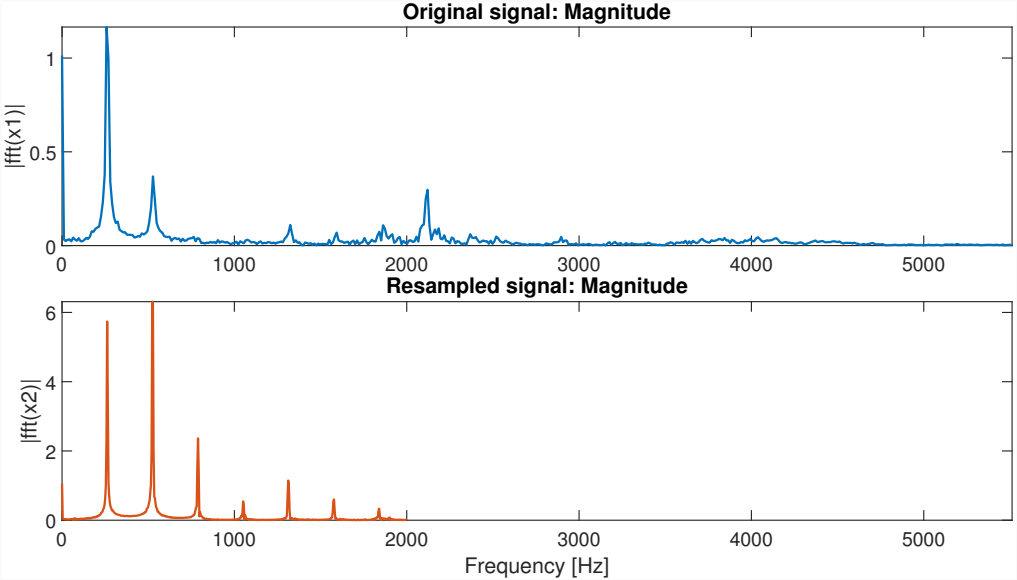
l) You can use a systematic way to find the frequency of the peaks in spectrum `abs(X2)` using following commands:

```
% find local maxima
mag = abs(X2);
mag = mag(1:end/2+1);
peaks = (mag(1:end-2) < mag(2:end-1)) & (mag(2:end-1) > mag(3:end));
```

m) Then evaluate the peaks at corresponding frequencies above a threshold:

```
peaks = peaks & mag(2:end-1) > 0.5;
fmax = f2(peaks)
```

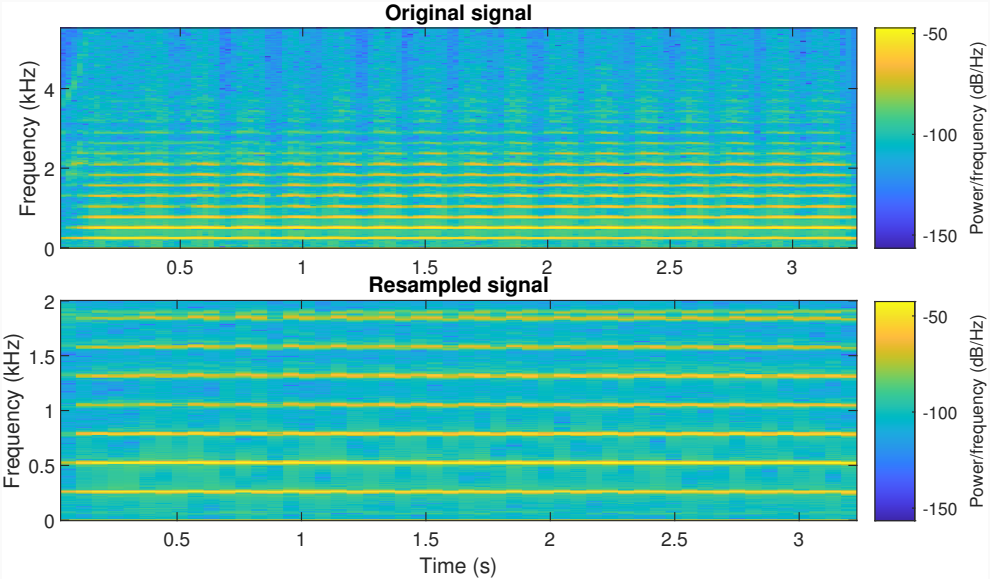
The same can be accomplished using `findpeaks()` function from Signal Processing Toolbox. Check its documentation using `doc findpeaks`.


Original signal: Magnitude
|fft(x1)|
0
0.5
1
0
1000
2000
3000
4000
5000
Resampled signal: Magnitude
|fft(x2)|
0
2
4
6
0
1000
2000
3000
4000
5000
Frequency [Hz]

n) Finally we will use Spectrogram with following specifications:

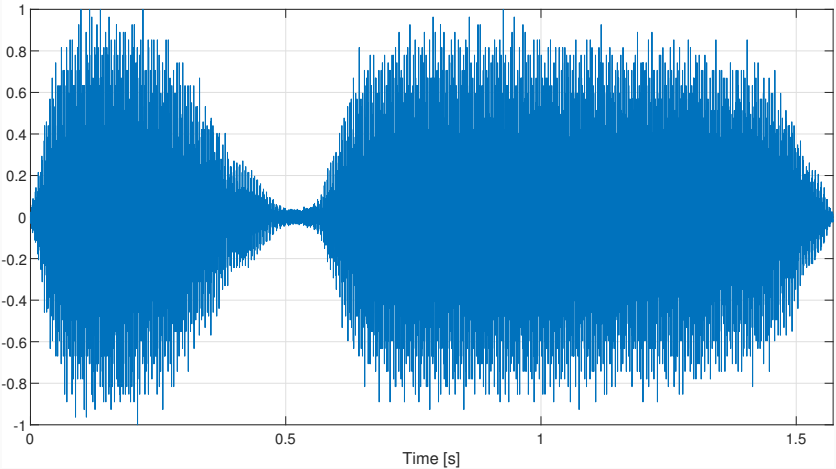
```
nwin = 512; % samples of a window
noverlap = 256; % samples of overlaps
nfft = 512; % samples of fast Fourier transform
f = figure(4);
subplot(211);
spectrogram(x1, nwin, noverlap, nfft, sr1, 'yaxis');
subplot(212);
spectrogram(x2, nwin, noverlap, nfft, sr2, 'yaxis');
```

o) Carefully study the options for the `spectrogram()` using `doc spectrogram`!


Original signal
Frequency (kHz)
0
2
4
0.5
1
1.5
2
2.5
3
-50
-100
-150
Power/frequency (dB/Hz)
Resampled signal
Frequency (kHz)
0
0.5
1
1.5
2
0.5
1
1.5
2
2.5
3
Time (s)
-50
-100
-150
Power/frequency (dB/Hz)

Non-compulsory Matlab Session 4.4

a) Start MATLAB. Load in the "train" signal with command `load('train')`. Note that the audio signal is loaded into a variable `y` and the sampling rate into `Fs`.



b) The sampling rate is 8192 Hz, and the signal contains 12 880 samples. If we consider this signal as sampled on an interval $(0, T)$, then $T = 12880/8192 \approx 1.5723$ seconds.

c) Compute the DFT of the signal with `Y=fft(y)`. Display the magnitude of the Fourier transform with `plot(abs(Y))`

The DFT is of length 12 880 and symmetric about center.

d) Since MATLAB indexes from 1, the DFT coefficient Y_k is actually `Y(k+1)` in MATLAB!

e) You can plot only the first half of the DFT with `plot(abs(Y(1:6441))`.

f) Compute the actual value of each significant frequency in Hertz. Use the data cursor on the plot window to pick out the frequency and amplitude of three largest components.

g) Denote these frequencies f_1, f_2, f_3, and let A_1, A_2, A_3 denote the corresponding amplitudes. Define these variables in MATLAB.

h) Synthesize a new signal using only these frequencies, sampled at 8192 Hz on the interval $[0, 1.5)$ with

```
t=[0:1/8192:1.5];
ys=(A1*sin(2*pi*f1*t)+A2*sin(2*pi*f2*t)+A3*sin(2*pi*f3*t))/(A1+A2+A3);
```

i) Play the original train sound with `sound(y)` and the synthesized version `sound(ys)`. Compare the quality!

j) Can you explore another frequency components? If it is so, follow the steps g)–i) and listen to the result.

We can now extend this observation and study a simple approach to lossy audio signal compression.

Proposition (Simple lossy audio signal compression)

The idea is to transform the audio signal in the frequency domain with DFT and then to eliminate the insignificant frequencies by thresholding, i.e. by zeroing out any Fourier coefficients below a given threshold. This becomes a compressed version of the signal. To recover an approximation to the signal, we use inverse DFT to take the limited spectrum back to the time domain.

k) Thresholding: Compute the maximum value of Y_k with `m=max(abs(Y))`. Choose a thresholding parameter $\in (0, 1)$, for example, `thresh=0.1`

l) Zero out all frequencies in `Y` that fall below a value `thresh*m`. This can be done with logical indexing or e.g. with

```
Ythresh=(abs(Y)>m*thresh).*Y;
```

Plot the thresholded transform with `plot(abs(Ythresh))`.

m) Compute the compression ratio as the fraction of DFT coefficients which survived the cut, `sum(abs(Ythresh)>0)/12880.`

n) Recover the original time domain with inverse transform `yt=real(ifft(Ytresh))` and play the compressed audio with `sound(yt)`. The `real` command truncates imaginary round-off error in the `ifft` procedure.

o) Compute the distortion (as a percentage) of the compressed signal using formula

$$\epsilon = \frac{\|\mathbf{y} - \mathbf{y}_t\|^2}{\|\mathbf{y}\|^2}$$

Note: The `norm(y)` command in MATLAB computes $\|\mathbf{y}\|$, the standard Euclidean norm of the vector $\mathbf{y}$.

p) Repeat the computation for threshold values `thresh=0.5`, `thresh=0.05` and `thresh=0.005`. In each case compute the compression ratio, the distortion, and play the audio signal and rate its quality.