Signal processing wrap-up session.

Mathematical Tools for ITS (11MAI)

Mathematical tools, 2021

Jan Přikryl

11MAI, lecture 5
Monday, October 18, 2021

version: 2021-10-19 09:33

Department of Applied Mathematics, CTU FTS

Lectue Contents

Nonlocality of DFT

We said that DFT assumes stationarity. It cannot detect local frequency or phase changes.

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

Q1: What is a time window?
Q2: Which two basic properties are of interest for time window functions?

Spectrogram

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

Spectrogram

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

Spectrogram

ČVUT ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

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

Spectrogram

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

Spectrogram

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

Spectrogram

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

Spectrogram

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

Spectrogram

In MATLAB the command

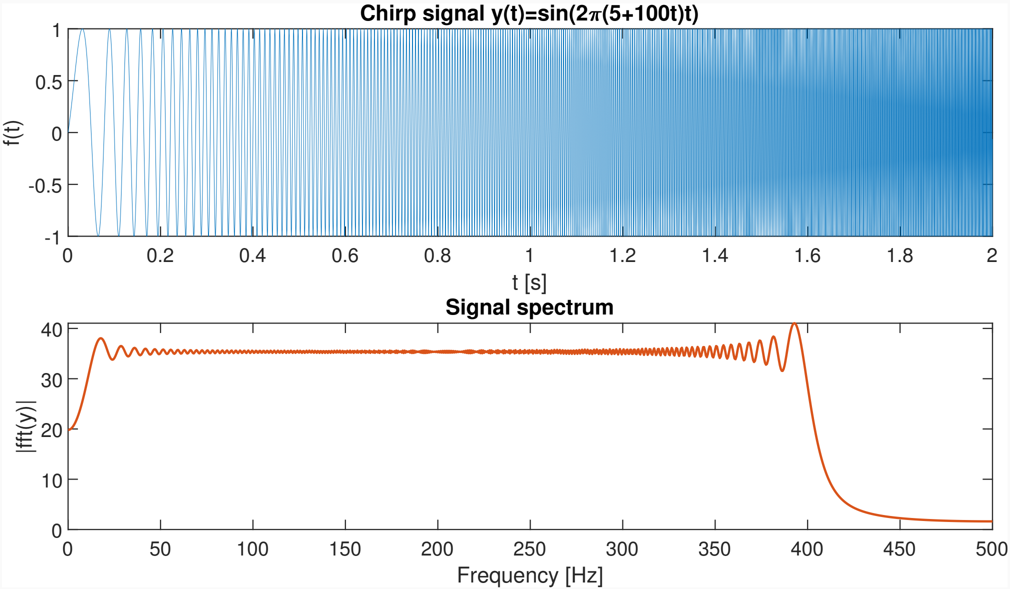
```
spectrogram(x,window,noverlap,nfft,fs,'yaxis')
```

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

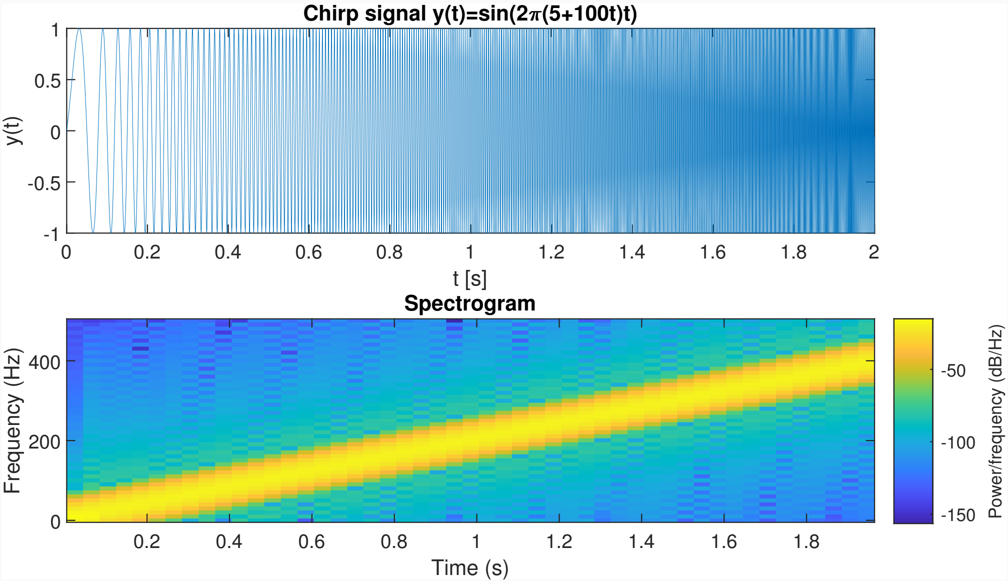
In current Matlab versions, the `colorbar` command is automatically issued to append a color scale to the current axes.

Matlab Session 5.1

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



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



Consider the chirp signal

$$y(t) = \sin\left(2\pi(f_0 + \alpha t)t\right)$$

for $f_0 = 5\,\mathrm{Hz}$ and $\alpha = 100$ on the interval $t \in [0, 2)$. Sample this signal with $f_s = 1000\,\mathrm{Hz}$ and obtain sample vector $\mathbf{y} = (y_0, y_1, y_2, \ldots, y_{1999})$.

a) Create a subplot which plots an "almost continuous" version of $y(t)$ and its spectrogram

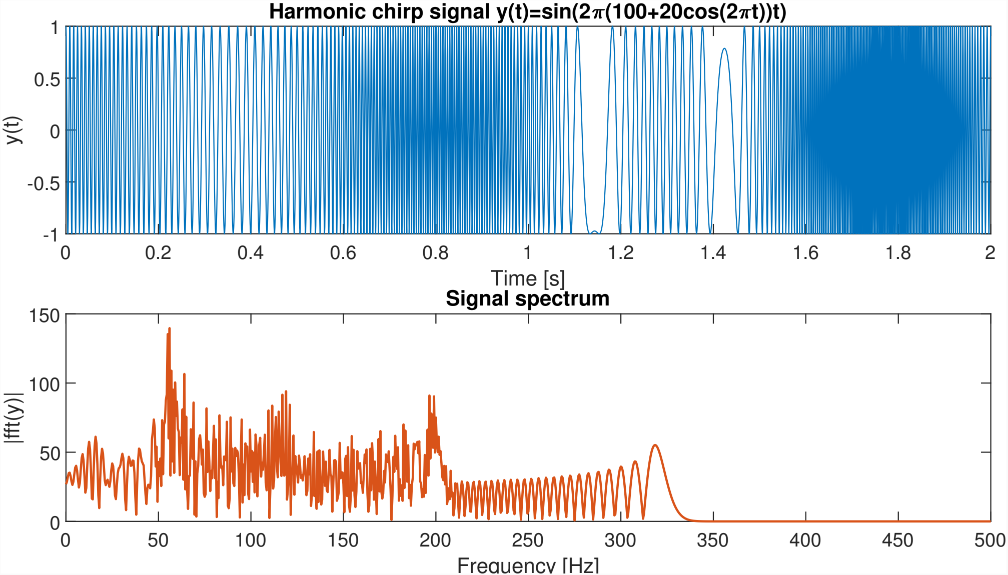
b) To create the spectrogram, use the Blackman window of length 50 samples, 100 DFT coefficients and overlap of 10 samples.

```
clear
fs = 1000; % Sampling frequency
tmax = 2; % End of the time interval
N = tmax*fs; % Number of samples
f0 = 5; % Lowest chirp frequency
alpha = 100; % Chirp rate
% The "continuous" original signal
tc = linspace(0,tmax,40*N+1);
tc(end) = []; % now we have 40*N time samples
yc = sin(2*pi*(f0 + alpha*tc).*tc); % `tc` is a vector, hence `.*`
% The sampled signal
ts = linspace(0,tmax,N+1); % the last sample is at t=2
ts(end) = []; % now we have N time samples
ys = sin(2*pi*(f0 + alpha*ts).*ts); % `ts` is a vector, hence `.*`
```

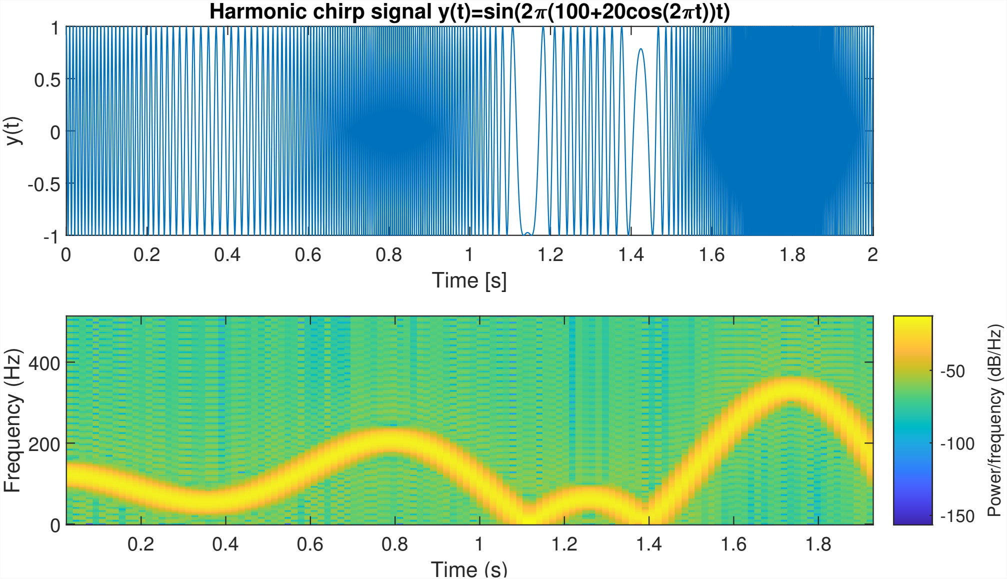
```
figure(1);
cmap = colormap('lines');
ax1 = subplot(2,1,1);
plot(tc, yc, 'LineWidth', 0.1, 'Color', cmap(1,:));
xlabel('t␣[s]');
ylabel('y(t)');
title(chirp_title);
ax2 = subplot(2,1,2);
spectrogram(y,blackman(50),10,100,1000,'yaxis');
colormap('parula');
title('Spectrogram');
```

Matlab Session 5.2

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



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



Lectue Contents

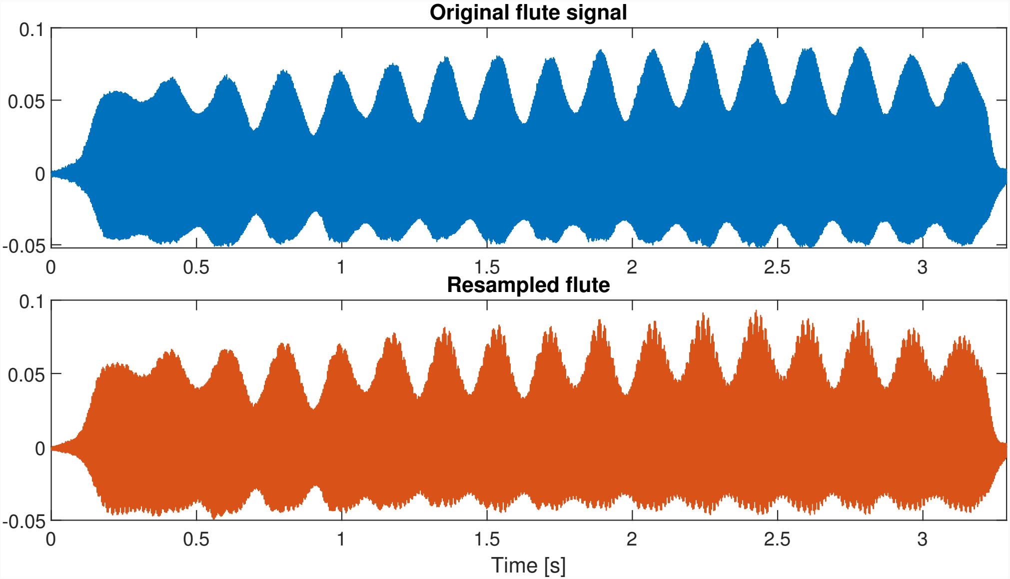
Matlab Session 5.3

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.

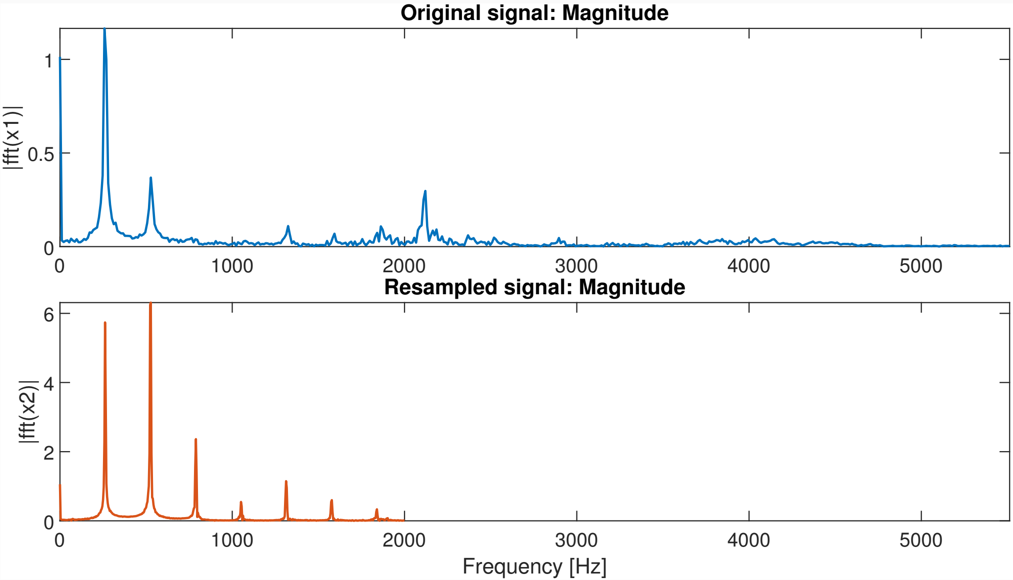
l) You can use a systematic way to find the frequency of the peaks in spectrum `abs(X2)` using following commands:

```
% 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)
```

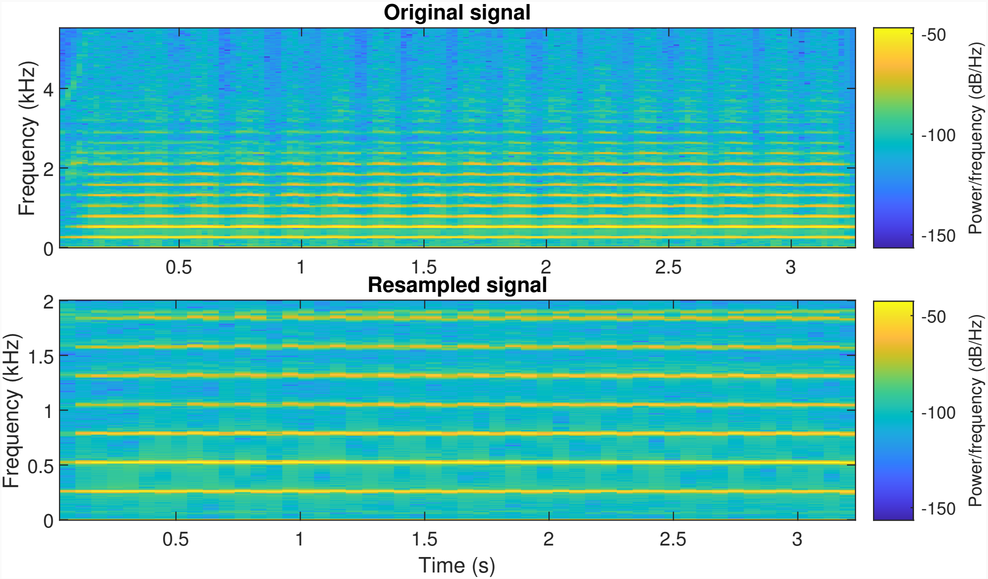
The same can be accomplished using `findpeaks()` function from Signal Processing Toolbox. Check its documentation using `doc findpeaks`.


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

n) Finally we will use Spectrogram with following specifications:

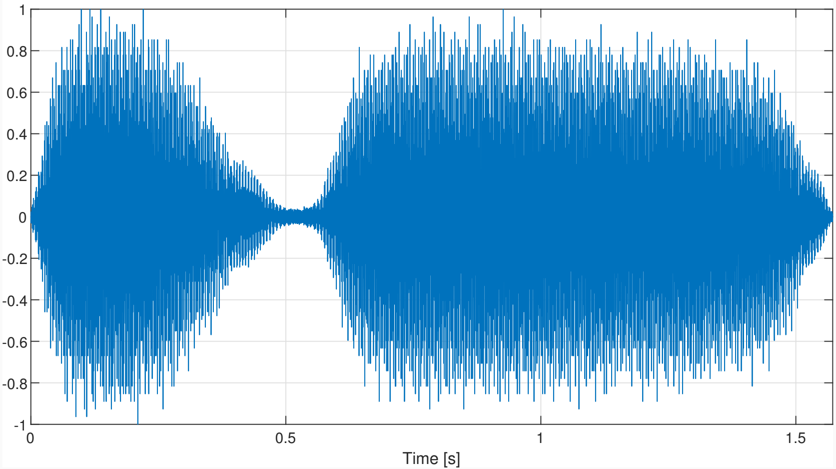
```
nwin = 512; % samples of a window
noverlap = 256; % samples of overlaps
nfft = 512; % samples of fast Fourier transform
f = figure(4);
subplot(211);
spectrogram(x1, nwin, noverlap, nfft, sr1, 'yaxis');
subplot(212);
spectrogram(x2, nwin, noverlap, nfft, sr2, 'yaxis');
```

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)

Matlab Session 5.4

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.
   Q: What is the length of the sound in 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.

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.

ČVUT ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

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.