From Fourier Series to Analysis of Non-stationary Signals – V

Miroslav Vlček, Jan Přikryl

November 4, 2019

Department of Applied Mathematics, CTU FTS

Contents

Properties of Fourier Transform

Fourier Transform

The Fourier Transform can be defined for signals that are

- Discrete or continuous in time
- Finite or infinite duration
- Provided we denote the variable in time domain as $x(t)$, or $x(n)$, the transformed variables in frequency domain are correspondingly $X(j\omega)$ or $X(k)$.

This unification results in four cases.

An overview of Fourier transforms

	continuous in time	discrete in time periodic in frequency
continuous in frequency	$x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(j\omega)e^{j\omega t}d\omega$ $X(j\omega) = \int_{-\infty}^{\infty} e^{-j\omega t}x(t)dt$ Fourier transform	$x(n) = \frac{T}{2\pi}\int_{-\pi/T}^{+\pi/T} X(e^{j\omega T})e^{jk\omega T}d\omega$ $X(e^{j\omega T}) = \sum_{n=-\infty}^{\infty} x(n)e^{-jk\omega T}$ Fourier transform $t = nT$ (DTFT)
discrete in frequency periodic in time	$x(t) = \sum_{k=-\infty}^{\infty} X(k)e^{jk\omega_0 t}$ $X(k) = \frac{\omega_0}{2\pi}\int_{-\pi/\omega_0}^{\pi/\omega_0} x(t)e^{-jn\omega_0 t}dt$ Fourier series	$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)e^{(j2\pi/N)kn}$ $X(k) = \sum_{n=0}^{N-1} x(n)e^{-(j2\pi/N)kn}$ Discrete Fourier transform (DFT)

The DFT consists of inner products of the input sequence $x[n]$ with sampled complex sinusoidal sections

$$w_N^{kn} = \mathrm{e}^{\mathrm{j}2\pi nk/N}$$

yielding

$$X(k) = \langle x, w_k \rangle = \sum_{n=0}^{N-1} x[n]\mathrm{e}^{-\mathrm{j}2\pi nk/N}, \quad k = 0, 1, 2, \ldots, N-1.$$

By collecting the DFT output samples into a column vector, we have

$$\underbrace{\begin{bmatrix} X[0] \\ X[1] \\ X[2] \\ \vdots \\ X[N-1] \end{bmatrix}}_{\mathbf{X}} = \underbrace{\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \overline{w_N^1} & \overline{w_N^2} & \cdots & \overline{w_N^{N-1}} \\ 1 & \overline{w_N^2} & \overline{w_N^4} & \cdots & \overline{w_N^{2(N-1)}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \overline{w_N^{N-1}} & \overline{w_N^{2(N-1)}} & \cdots & \overline{w_N^{(N-1)(N-1)}} \end{bmatrix}}_{\mathbf{W}_N^*} \underbrace{\begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ \vdots \\ x[N-1] \end{bmatrix}}_{\mathbf{x}}$$

Finally we can write matrix representation as

$$\mathbf{X} = \mathbf{W}_N^* \mathbf{x}. \tag{1}$$

The matrix $\mathbf{W}_N^* = \overline{\mathbf{W}_N^T}$ denotes the Hermitian transpose of the complex matrix $\mathbf{W}_N$. It can be shown that

$$\mathbf{W}_N^* \times \mathbf{W}_N = \begin{bmatrix} N & 0 & 0 & \cdots & 0 \\ 0 & N & 0 & \cdots & 0 \\ 0 & 0 & N & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & N \end{bmatrix} = N \cdot \mathbf{1}$$

and consequently the inversion of the Eq. (1) is

$$\mathbf{x} = \frac{1}{N}\mathbf{W}_N\mathbf{X}. \tag{2}$$

Some practical comments

If the number of digital samples in each time slice is a power of 2, one can use a faster version of the DFT known as the fast Fourier transform (FFT)

The FFT assumes that the samples being analyzed comprise one cycle of a periodic wave. In most cases it is not the case and analysis will contain many spurious frequencies not actually present in the signal.

Sample fast enough and long enough!

To recognize details in frequency domain use spectral interpolation.

Aliasing

What is aliasing?

It is easiest to describe in terms of a visual sampling:

We all know and love movies. If you have ever watched a western and seen the wheel of a rolling wagon appear to be going backwards, you have witnessed aliasing. The movie's frame rate is not adequate to describe the rotational frequency of the wheel, and our eyes are deceived by the misinformation.

The Nyquist Theorem tells us that we can successfully sample and play back frequency components up to one-half the sampling frequency.

Aliasing is the term used to describe what happens when we try to record and play back frequencies higher than one-half the sampling rate.

What is aliasing?

Consider a digital audio system with a sample rate of 48 KHz, recording a steadily rising sine wave tone. At lower frequency, the tone is sampled with many points per cycle. As the tone rises in frequency, the cycles get shorter and fewer and fewer points are available to describe it. At a frequency of 24 KHz, only two sample points are available per cycle, and we are at the limit of what Nyquist says we can do.

Still, those two frequency points are adequate, in a theoretical world, to recreate the tone after conversion back to analog and low-pass filtering.

What is aliasing?

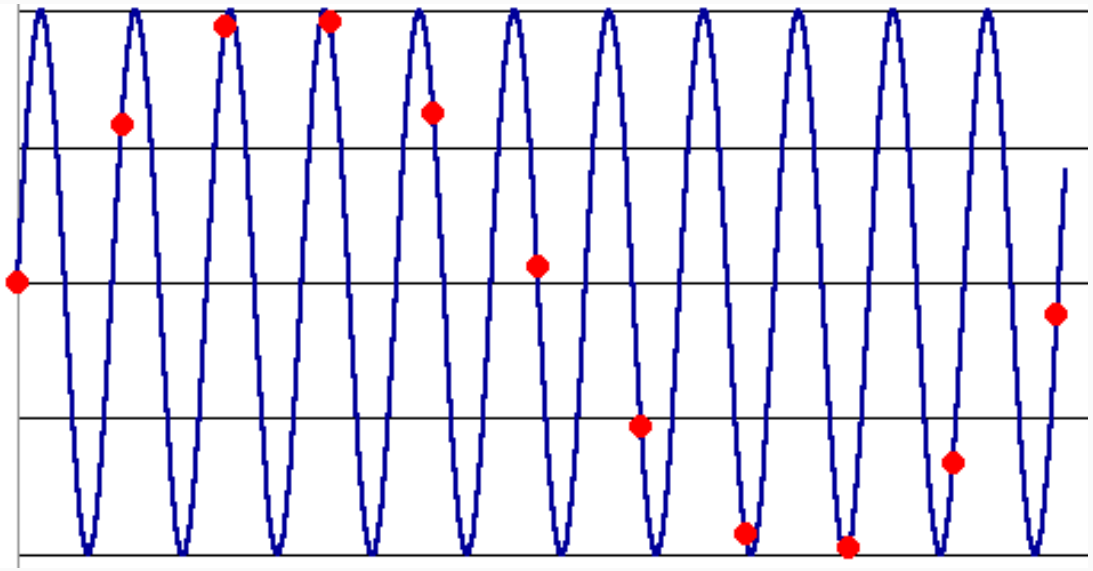
But, if the tone continues to rise, the number of samples per cycle is not adequate to describe the waveform, and the inadequate description is equivalent to one describing a lower frequency tone – this is aliasing.

In fact, the tone seems to reflect around the 24 KHz point:

- A 25 KHz tone becomes indistinguishable from a 23 KHz tone.
- A 30 KHz tone becomes an 18 KHz tone.

Aliasing due to a slow sampling

The following figure illustrates what happens if a signal is sampled at regular time intervals that are slightly less often than once per period of the original signal.



Zero Padding in discrete Fourier Transform

Zero padding

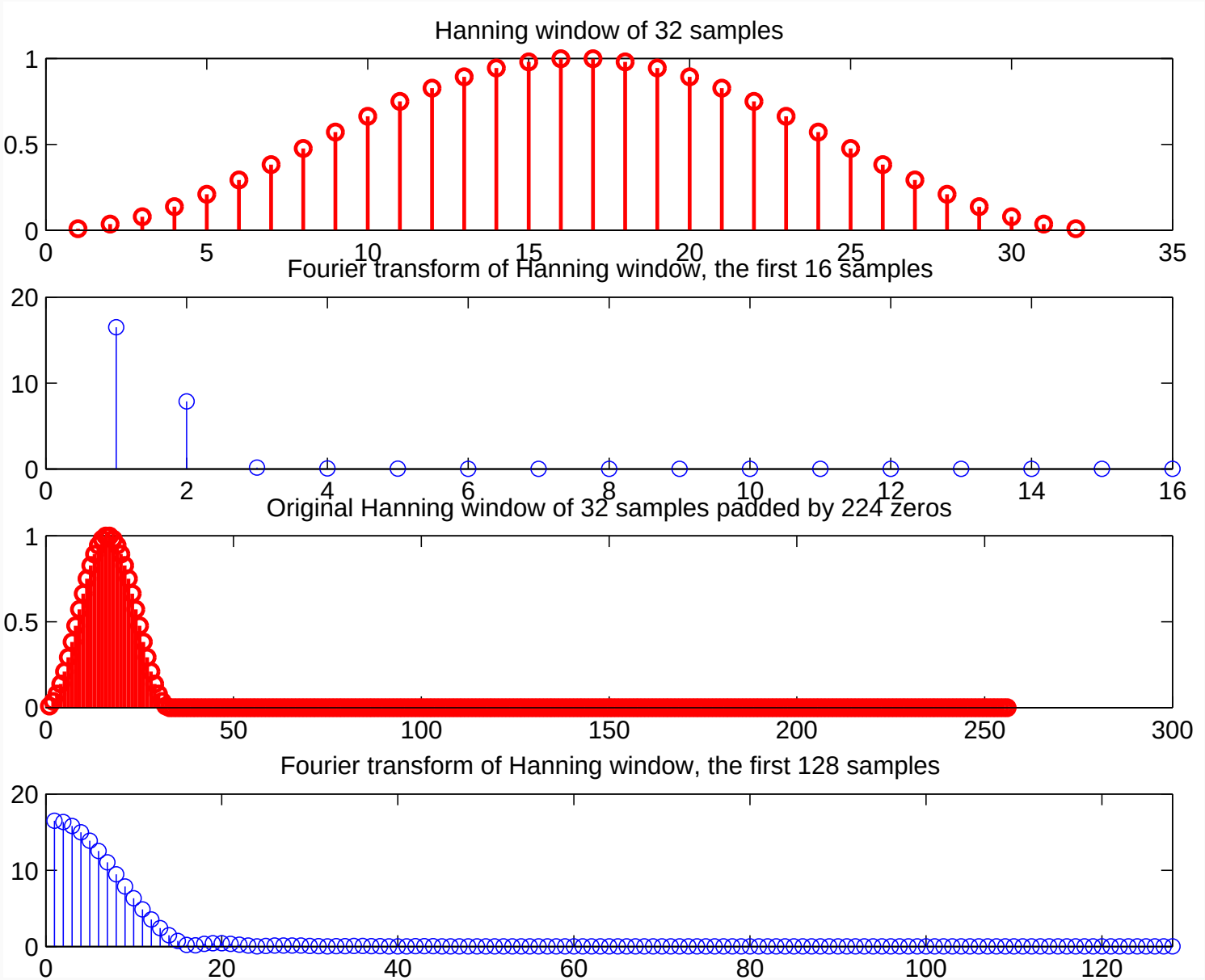
Zero padding consists of appending zeros to a signal. It maps a length N signal to a length $M > N$ signal. M does not need to be an integer multiple of N.

Zero padding in the time domain gives spectral interpolation in the frequency domain.

Similarly, zero padding in the frequency domain gives bandlimited interpolation in the time domain. This is how ideal sampling rate conversion is accomplished.

Usually we use FFT which requires signals of length $M = 2^m$ which means we chose the number of zeros equal to $2^m - N$.

Zero padding: How does it work?



Project

Project 1: Zero padding

Using reasonable resolution in frequency domain with zero padding in the time domain, determine the frequency of the periodic signal defined as

$$x_s = \sin(32.044245t) + \sin(37.070793t).$$

The discrete signal has only 32 samples x_n produced by sampling frequency $f_0 = 1/32$.

Solution to project 1: Zero padding (1/2)

ČVUT
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

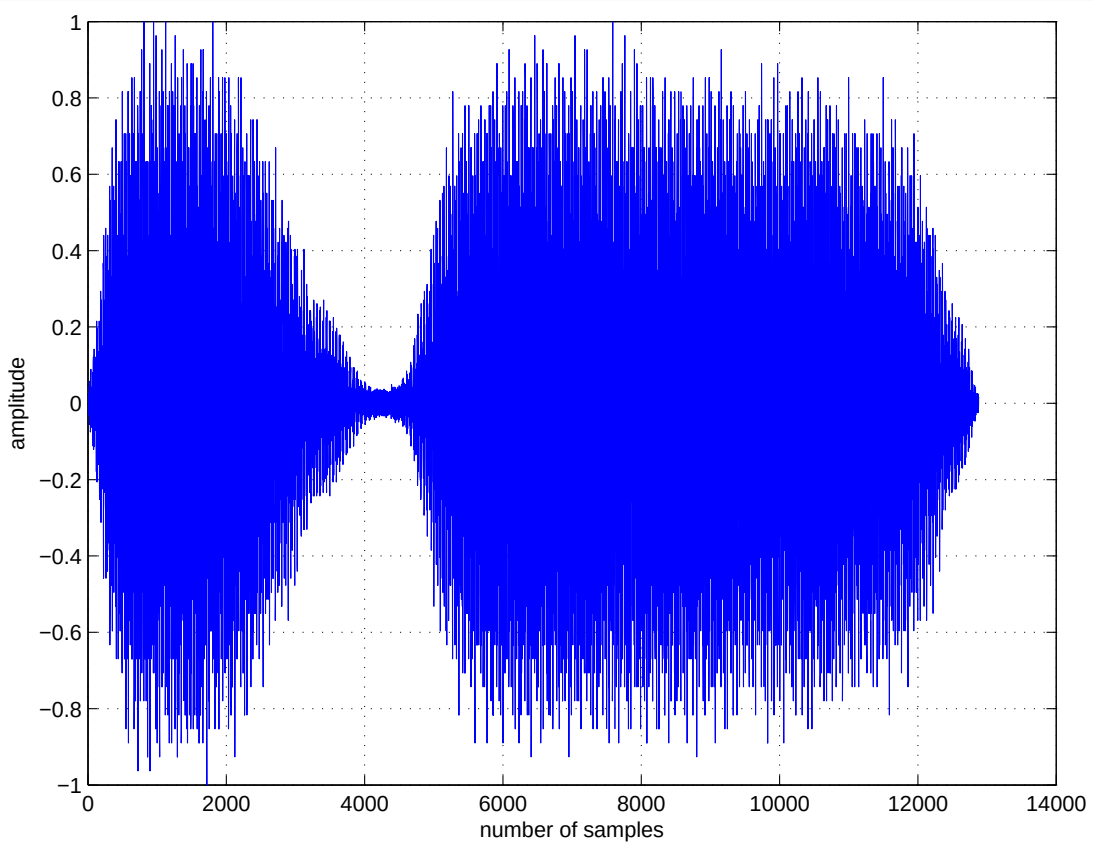
```
clear
t = linspace(0,1,1001);
xs = sin(32.044245*t)+sin(37.070793*t);
N = 32;
f0 = 1/N;
k = 0:1:N-1;
x1 = sin(32.044245*f0*k) + sin(37.070793*f0*k);
figure(1)
subplot(3,1,1)
plot(t,xs,'LineWidth',1.5,'Color',[1 0 0]);
```

Solution to project 1: Zero padding (2/2)


```
% Second row, make the lenghth 64 samples
subplot(3,1,2)
xfa_64 = abs(fft([x1 zeros(1,32)]));
plot(xfa_64); hold on;
stem(xfa_64); hold off;
% Third row, make the length 512 samples
xfa_512 = abs(fft([x1 zeros(1,32)]));
plot(xfa_512); hold on;
stem(xfa_512); hold off;
```

Project 2: Train

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`.



Project 2: Train, steps 2–6

b) The sampling rate is 8192 Herz, 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 Herz. Use the data cursor on the plot window to pick out the frequency and amplitude of three largest components.

Project 2: Train, steps 7–10

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) Synthetize a new signal using only these frequencies, sampled at 8192 Herz 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 hear the result.

Project 2: Train, compression

We can study a simple approach to compressing an audio signal:

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.

Project 2: Train, steps 11–14

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.

Project 2: Train, steps 15-16

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.