

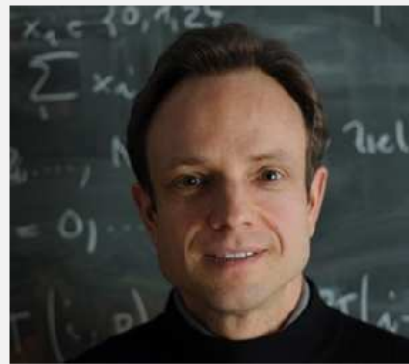
Course Video

Section 4.2.3: Frequency filtering via DFT

Prof. R. Hiptmair, SAM, ETH Zurich

Date: October 17, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



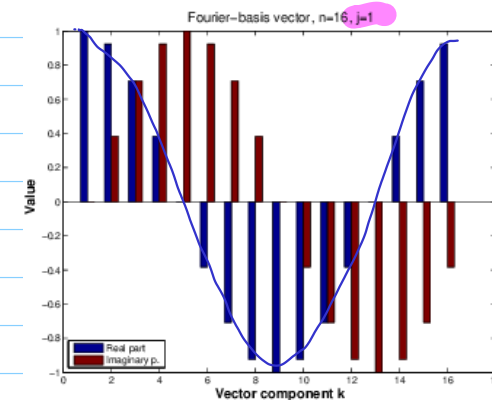
Dependencies. Section 4.2.1, Section 4.2.2

Duration: 30 minutes

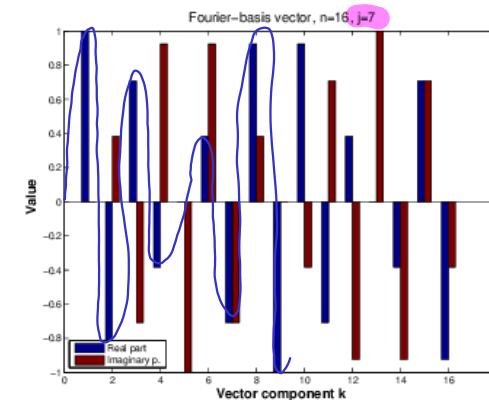
IV. Filtering Algorithms

4.2. Discrete Fourier Transform (DFT)

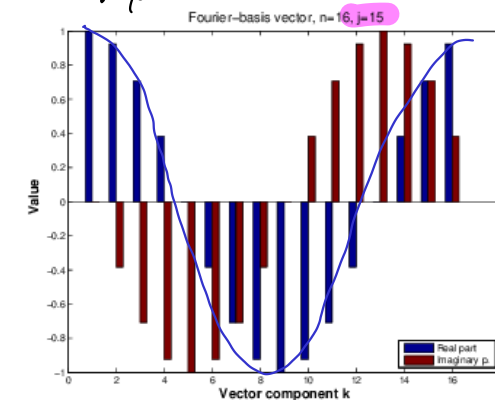
4.2.3. Frequency Filtering via DFT

DFT: $\underline{x} = F_n \underline{y} \Leftrightarrow \underline{y} = \frac{1}{n} \overline{F_n} \underline{x} \stackrel{!}{=} \text{a basis trf.}$ New basis vector: cols. of Fourier matrix F_n
 $\hookrightarrow$ represent oscillations with different frequencies

"slow oscillation/low frequency"



"fast oscillation/high frequency"



"slow oscillation/low frequency"

$$\begin{aligned}
 (F_n)_{:,k} &= [\omega_n^{kj}]_{j=0}^{n-1} = [e^{-\frac{2\pi i}{n}kj}]_j \\
 &= [\cos(2\pi k \frac{j}{n})]_j - i [\sin(2\pi k \frac{j}{n})]_j \\
 \Rightarrow k\text{-th column} &\hat{=} \text{sampled trigonometric functions of frequency } k
 \end{aligned}$$

② DFT discrete Fourier coefficient

$$c_k = \sum_{j=0}^{n-1} y_j \omega_n^{kj} \Leftrightarrow y_j = \frac{1}{n} \sum_{k=0}^{n-1} c_k \omega_n^{-kj}$$

For $y_j \in \mathbb{R}^n$, $n = 2m+1$

$\triangleright (y_j)$ as superposition of oscillations

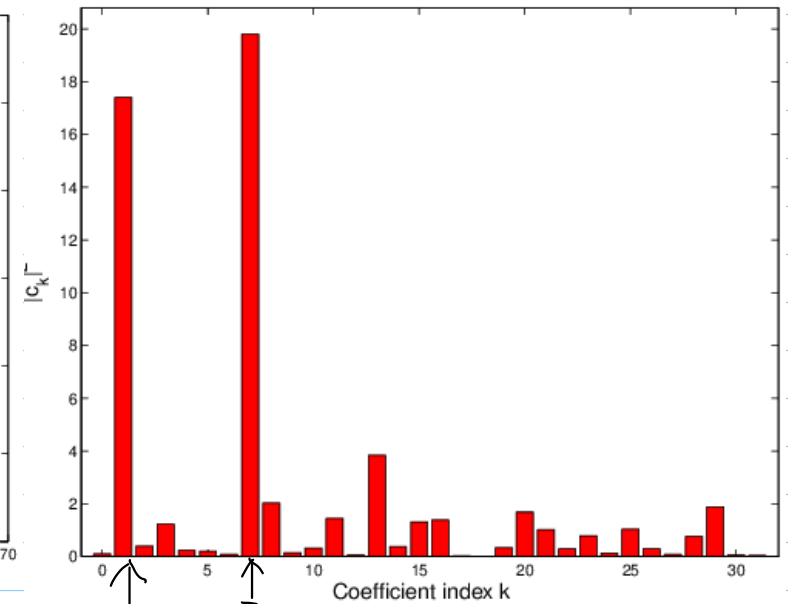
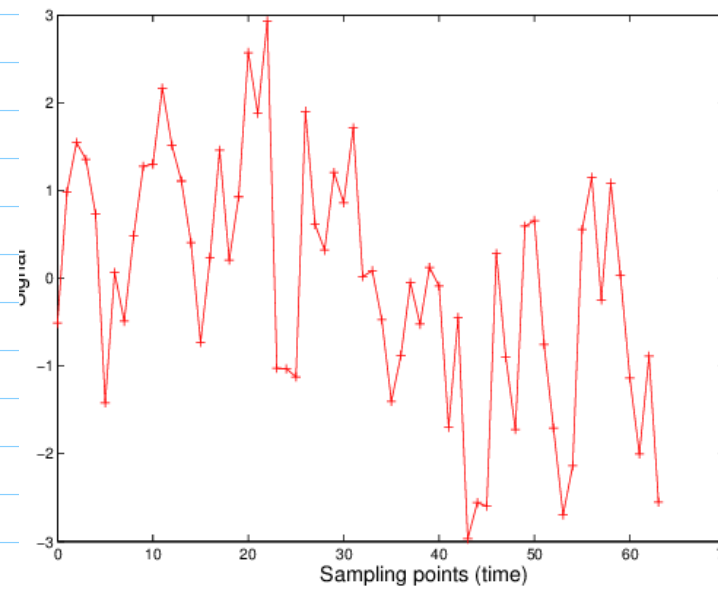
$$\begin{aligned} ny_j &= c_0 + \sum_{k=1}^m c_k \omega_n^{-kj} + \sum_{k=m+1}^{2m} c_k \omega_n^{-kj} = c_0 + \sum_{k=1}^m c_k \omega_n^{-kj} + c_{n-k} \omega_n^{(k-n)j} \\ &= c_0 + 2 \sum_{k=1}^m \operatorname{Re}(c_k) \cos(2\pi kj/n) + \operatorname{Im}(c_k) \sin(2\pi kj/n), \quad j = 0, \dots, n-1, \\ |c_k| &\leftrightarrow \text{strength of frequency } k \text{ in signal (modes)} \end{aligned}$$

Example 4.2.3.3: Frequency identification with DFT

Signal = two modes + white noise

C++ code 4.2.3.4: Generation of noisy sinusoidal signal [→ GITLAB](#)

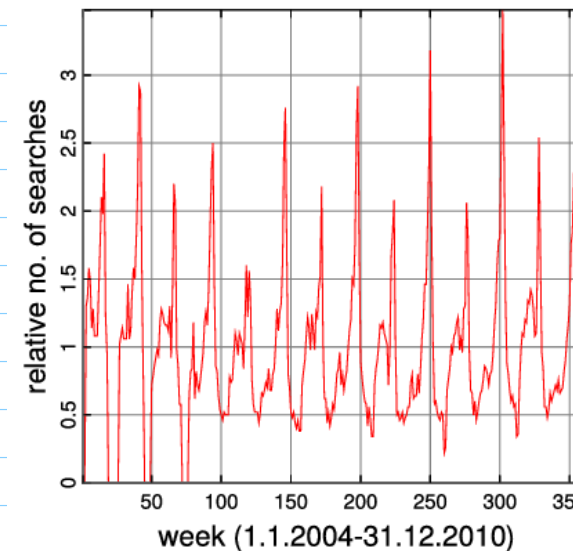
```
2 VectorXd signalgen() {
3     const int N = 64;
4     const ArrayXd t = ArrayXd::LinSpaced(N, 0, N);
5     const VectorXd x = ((2*M_PI/N*t).sin() + (14*M_PI/N*t).sin()).matrix();
6     return x + VectorXd::Random(N);
7 }
```



time-domain signal $\xleftrightarrow{\text{DFT}}$ Frequency-domain signal

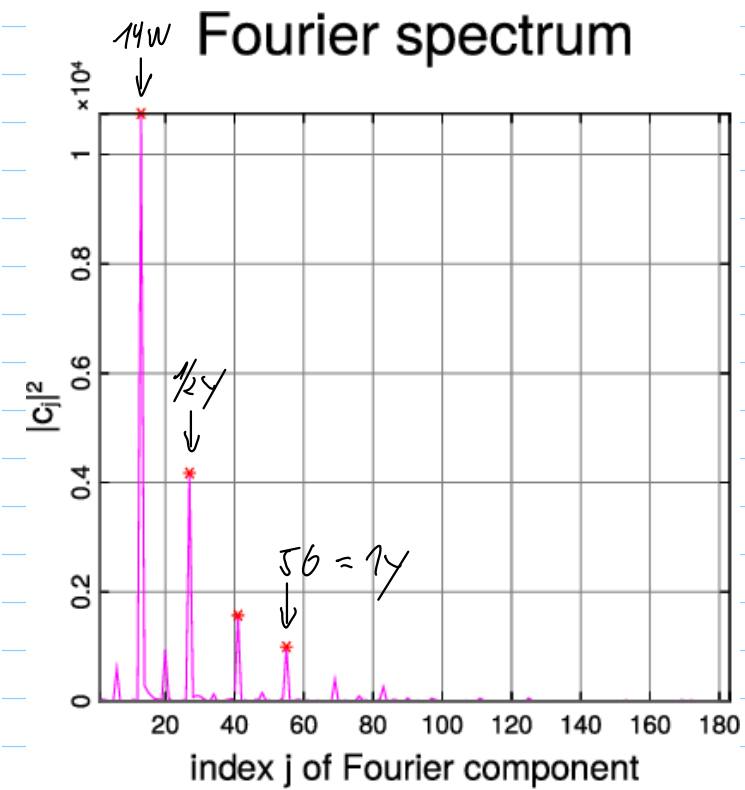
Example 4.2.3.5 (Detecting periodicity in data)

Google: 'Vorlesungsverzeichnis'



$\triangle$ # weekly searches
[320 weeks]

③



Power spectrum of data

$$f \mapsto |c_f|^2$$

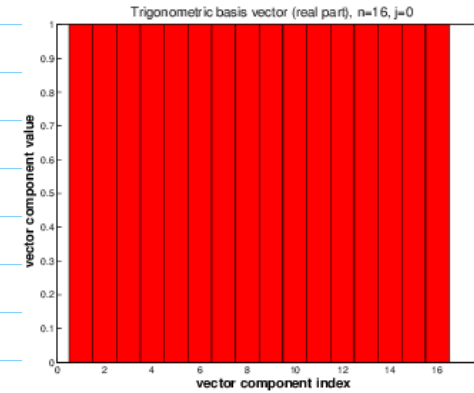
$$\mathbb{c} = \text{DFT}_n \mathbf{x}$$

→ up to $\lceil n/2 \rceil$

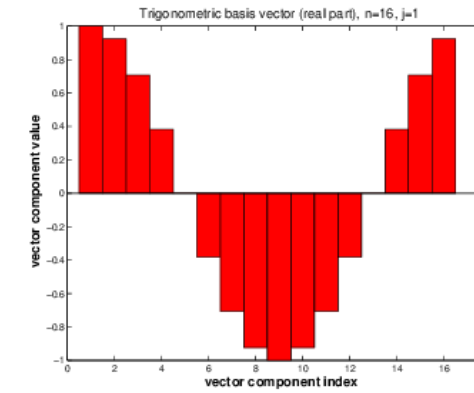
DFT is a computer's eye for periodic patterns in data

§ 4.2.3.6 ("low" and "high" frequencies)

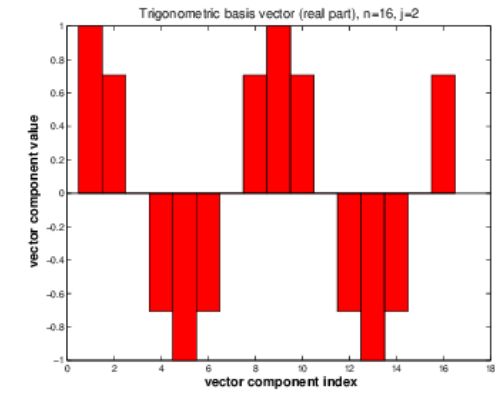
Real parts of cols. of F_n , $n=16$



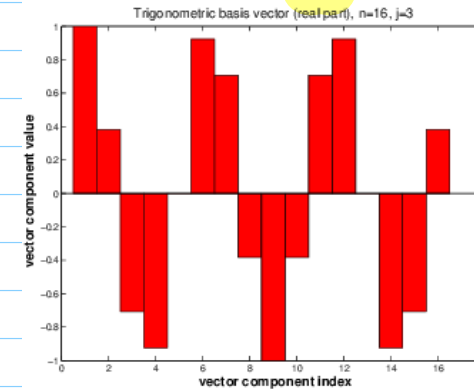
$\text{Re}(F_{16})_{:,0}$



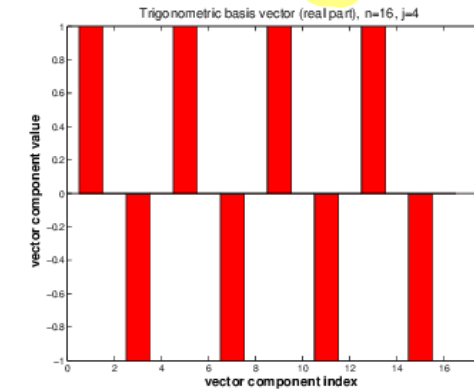
$\text{Re}(F_{16})_{:,1}$



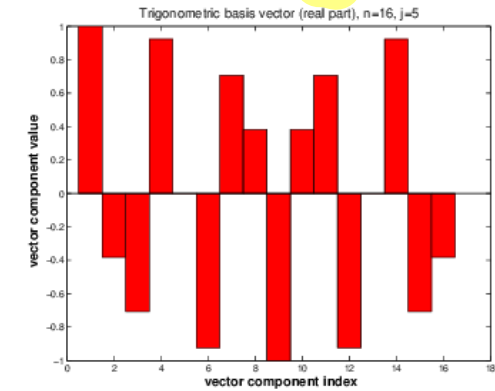
$\text{Re}(F_{16})_{:,2}$



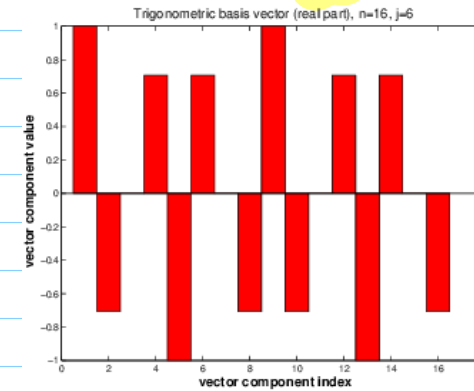
$\text{Re}(F_{16})_{:,3}$



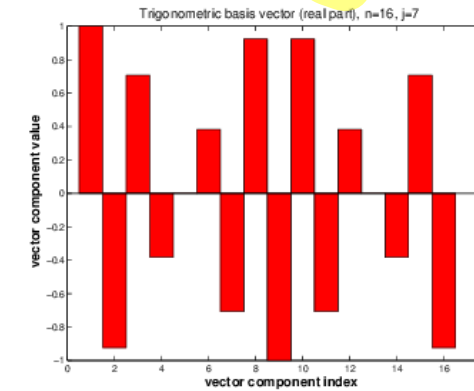
$\text{Re}(F_{16})_{:,4}$



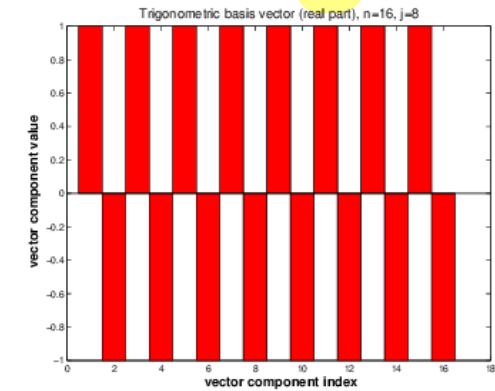
$\text{Re}(F_{16})_{:,5}$



$\text{Re}(F_{16})_{:,6}$

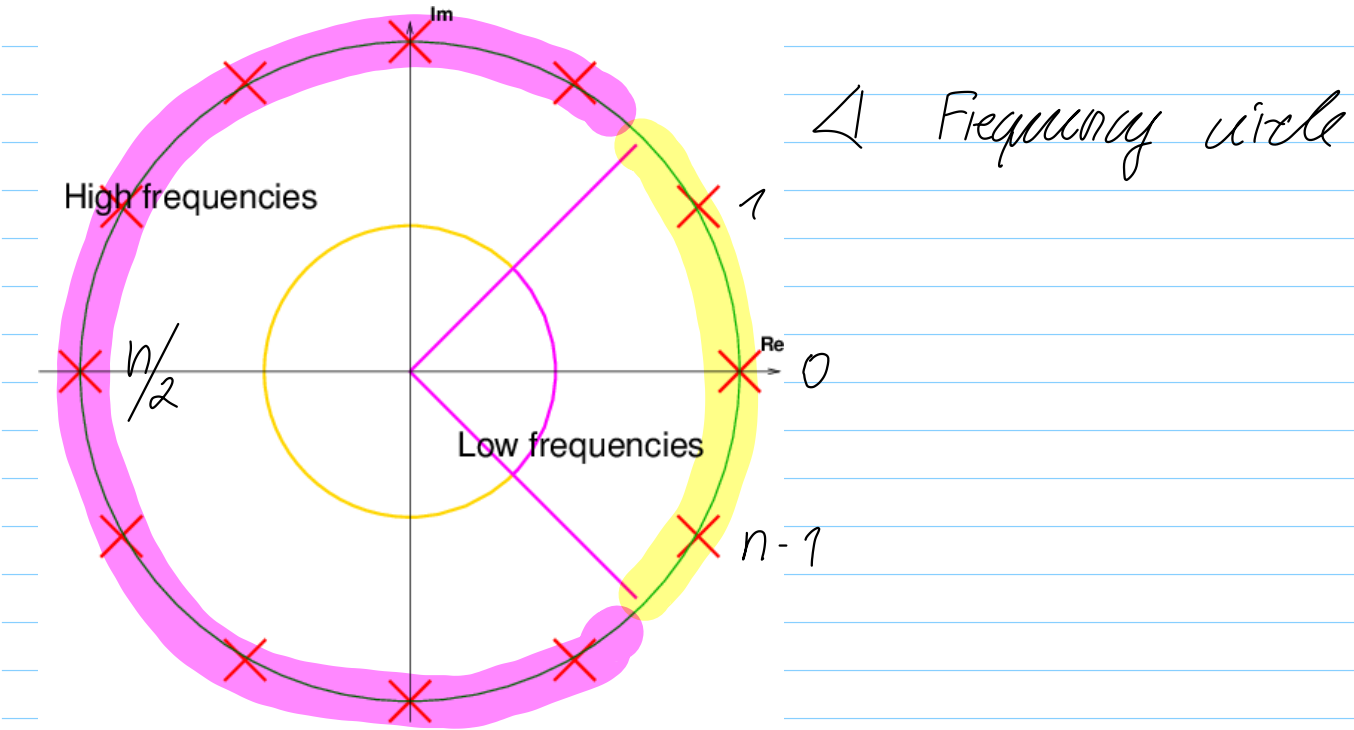


$\text{Re}(F_{16})_{:,7}$



$\text{Re}(F_{16})_{:,8}$

④ $(F_n)_{:,n-k} = [\omega_n^{j(n-k)}]_{j=0}^{n-1} \xrightarrow{\omega_n^n = 1} [\omega_n^{-jk}] = (\bar{F}_n)_{:,k}$ ["wrapped around"]



△ Frequency circle

▷ Frequency filtering:

time-domain signal $\xrightarrow{\text{DFT}}$ $\subseteq$ "do something" $\xrightarrow{\text{DFT}^{-1}}$ $\hat{\sim}$

"do something": low-pass: set $c_j := 0$ for $j: |j - n/2| < k$

high-pass: set $c_j := 0$ for $j: |j - n/2| \geq k$

C++11-code 4.2.3.7: DFT-based frequency filtering → GITLAB

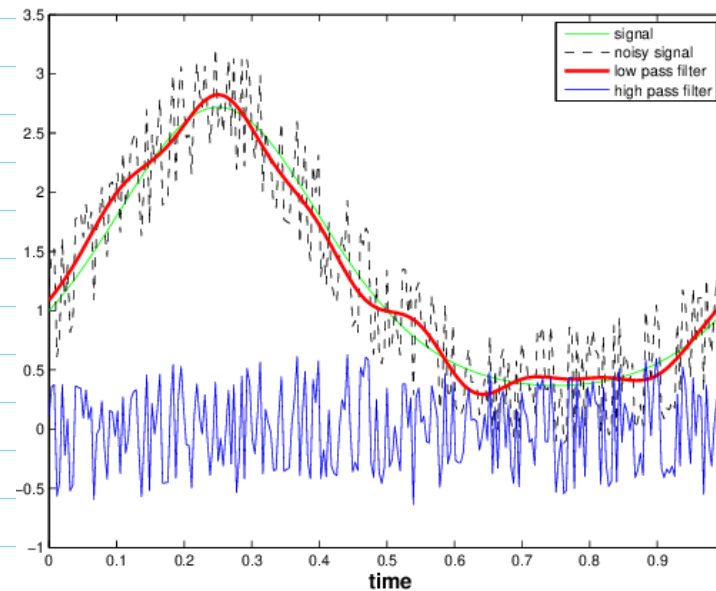
```
void freqfilter(const VectorXd &y, int k, VectorXd &low, VectorXd &high) {
    const VectorXd::Index n = y.size(); // threshold
    if (n % 2 != 0)
        throw std::runtime_error("Even vector length required!");
    const VectorXd::Index m = y.size() / 2;

    Eigen::FFT<double> fft; // DFT helper object
    VectorXd c = fft.fwd(y); // Perform DFT of input vector

    VectorXd clow = c;
    // Set high frequency coefficients to zero, Fig. 125
    for (int j = -k; j <= +k; ++j) clow(m + j) = 0; // low-pass
    // (Complementary) vector of high frequency coefficients
    VectorXd chigh = c - clow; // high-pass

    // Recover filtered time-domain signals
    low = fft.inv(clow).real();
    high = fft.inv(chigh).real();
}
```

Example 4.2.3.8 (Denoising by frequency filtering)

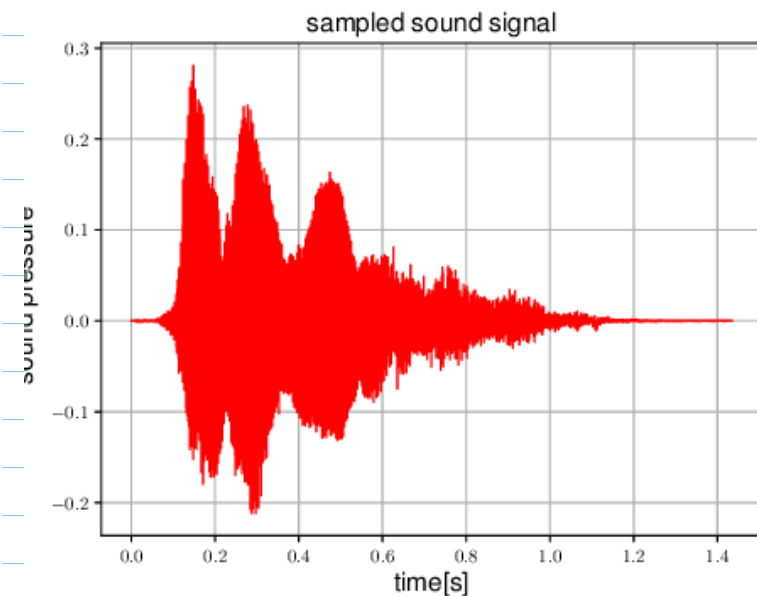


△ Signal + high-frequency noise
[n = 64k]

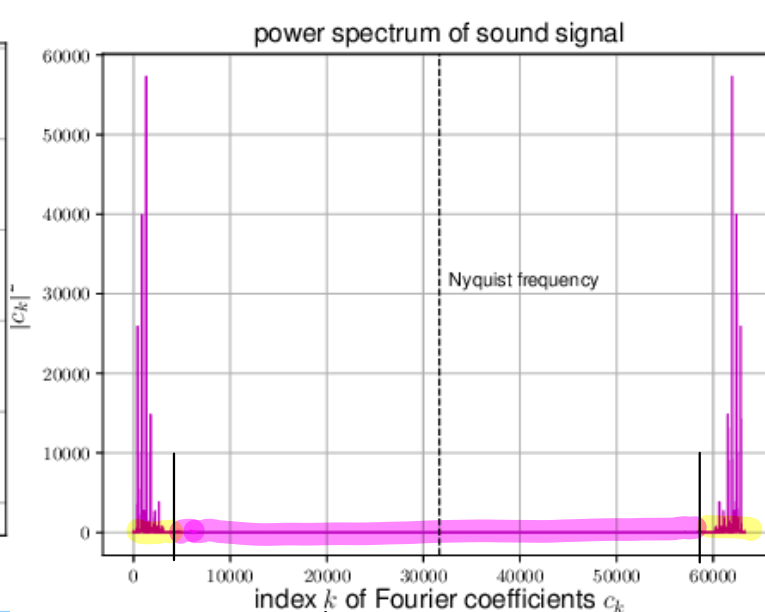
→ Low-pass filtering removes noise

5

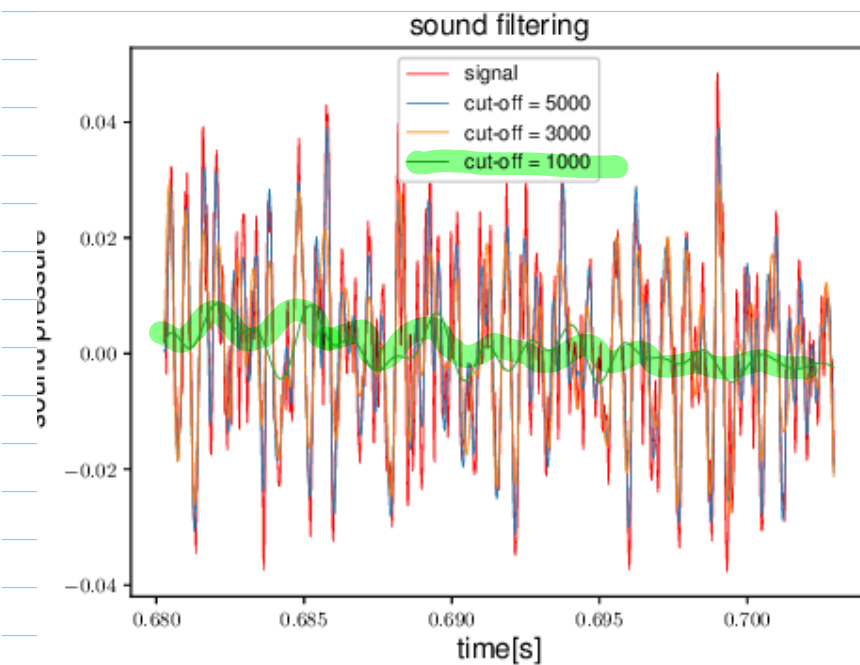
Example 4.2.3.9. (Sound filtering by DFT)



Time domain : human voice



→ signal information mainly in low frequencies



▷ Compression by low-pass filtering

Remark 4.2.3.10 : Linear Filtering

Signal $\not\sim$ (in "pulse bases") $\xrightarrow[\text{h.f.}]{\text{basis}}$ $\subseteq$ $\xrightarrow[\text{operation}]{\text{linear}}$ $\subseteq$ $\xrightarrow[\text{h.f.}]{\text{inv. bas}}$ $\not\sim$

Review questions 4.2.3.11/4.2.4.6

(Tackle the questions without relying on tablet or lecture notes)

7