5 Filtering Algorithms

As throughout the whole course materials let $n, m \in \mathbb{N}$ be natural numbers if not otherwise specified.

Perspective of signal processing:

vector $\mathbf{x} \in \mathbb{R}^n$ $\leftrightarrow$ finite discrete (= sampled) signal.

$X = (X(t))_{t\in[0,T]} \hat{=}$ *time-continuous* signal

"sampling": $x_j = X(j\Delta t)$, $j = 0, \dots, n-1$,
$(n-1)\Delta t \leq T$.

$\Delta t > 0 \hat{=}$ time between samples.

Sampled values arranged in a vector $\mathbf{x} = (x_0, x_1, \dots, x_{n-1}) \in \mathbb{R}^n$.

Note: vector indices often denoted by $0, 1, \dots, n-1$!

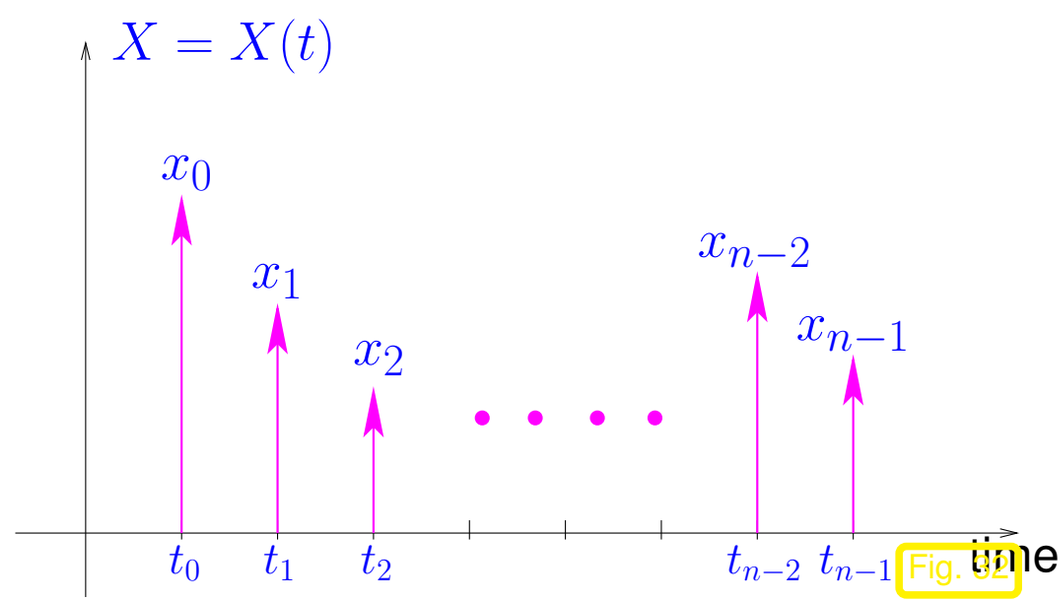
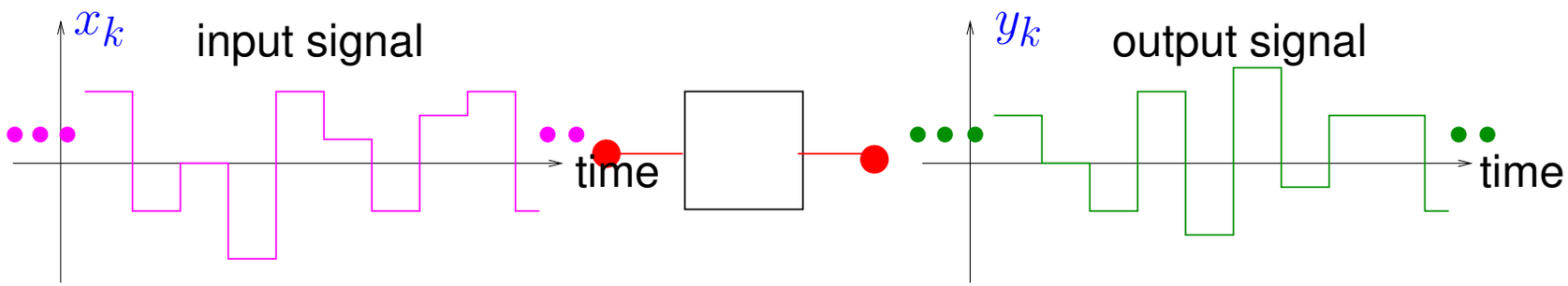


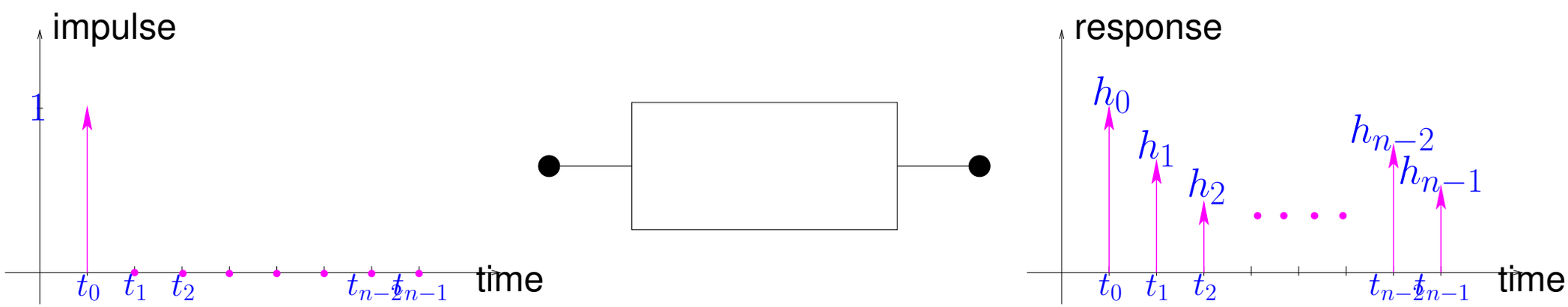
Fig. 62

5.1 Discrete convolutions

Example 5.1.1 (Discrete finite linear time-invariant causal channel (filter)).



Impulse response of channel (filter): $\mathbf{h} = (h_0, \dots, h_{n-1})$



Impulse response = output when filter is fed with a single impulse of strength one, corresponding to input $\mathbf{e}_1$ (first unit vector).

We study a *finite linear time-invariant causal channel (filter)*:
(widely used model for digital communication channels, e.g. in wireless communication theory)

finite: impulse response of finite duration ➢ it can be described by a vector $\mathbf{h}$ of finite length n.

time-invariant: when input is shifted in time, output is shifted by the same amount of time.

linear: input $\mapsto$ output-map is linear

$$\text{output}(\mu \cdot \text{signal 1} + \lambda \cdot \text{signal 2}) = \mu \cdot \text{output}(\text{signal 1}) + \lambda \cdot \text{output}(\text{signal 2}) \ .$$

causal (or physical, or nonanticipative): output depends only on past and present inputs, not on the future.

The assumptions of time-invariance and linearity bring us to the next definition.

$\diamond$

Definition 5.1.1 (Discrete convolution)**.**
For two complex valued sequences $\mathbf{x} = (x_k)_{k\in\mathbb{N}_0}$ *and* $\mathbf{h} = (h_k)_{k\in\mathbb{N}_0}$ *we call the complex valued sequence* $\mathbf{h} * \mathbf{x} = \big((h * x)_k\big)_{k\in\mathbb{N}_0}$ *defined by*

$$(h * y)_k := \sum_{\substack{j_1,j_2\in\mathbb{N}_0\\ j_1+j_2=k}} h_{j_1}\, x_{j_2} = \sum_{j=0}^{k} h_{k-j}\, x_j$$

for all $k \in \mathbb{N}_0$ *the discrete convolution* (ger.: *diskrete Faltung*) *of* $\mathbf{h}$ *and* $\mathbf{x}$*.*
In addition, for every $\mathbf{x} = (x_0, \dots, x_{n-1}) \in \mathbb{C}^n$ *and* $\mathbf{h} = (h_0, \dots, h_{n-1}) \in \mathbb{C}^n$ *we define the discrete convolution* $\mathbf{h} * \mathbf{x} = \big((h * x)_0, (h * x)_1, \dots, (h * x)_{2n-2}\big) \in \mathbb{C}^{2n-1}$ *of* $\mathbf{h}$ *and* $\mathbf{x}$ *by*

$$(h * y)_k := \sum_{\substack{j_1,j_2\in\{0,1,\dots,n-1\}\\ j_1+j_2=k}} h_{j_1}\, x_{j_2} = \sum_{\substack{0\le j\le n-1\\ 0\le k-j\le n-1}} h_{k-j}\, x_j = \sum_{j=\max(0,k+1-n)}^{\min(n-1,k)} h_{k-j}\, x_j$$

for all $k \in \{0, 1, \dots, 2n-2\}$*.*

Note that the discrete convolution is *bilinear*.
In addition, observe that $\mathbf{h} * \mathbf{x} = \mathbf{x} * \mathbf{h}$ for all $\mathbf{h}, \mathbf{x} \in \mathbb{C}^n$ (commutativity).

Matrix notation of $\mathbb{C}^n \ni \mathbf{x} = (x_0, \dots, x_{n-1}) \mapsto \mathbf{y} = (y_0, \dots, y_{2n-2}) = \mathbf{h} * \mathbf{x} \in \mathbb{C}^{2n-1}$:

$$\begin{pmatrix} y_0 \\ \vdots \\ \\ \\ \\ \vdots \\ y_{2n-2} \end{pmatrix} = \begin{pmatrix} h_0 & 0 & \cdots & 0 \\ h_1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ h_{n-1} & \cdots & h_1 & h_0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \\ 0 & \cdots & 0 & h_{n-1} \end{pmatrix} \begin{pmatrix} x_0 \\ \vdots \\ \\ \\ \\ \vdots \\ x_{n-1} \end{pmatrix} . \tag{5.1.1}$$

Beside their appearence in filters for discrete signals (see Ex. 5.1.1), discrete convolutions also appear in the multiplication of polynomials. This is the subject of the next example.

Example 5.1.2 (Multiplication of polynomials).

$$p(z) = \sum_{k=0}^{n-1} a_k z^k \,, \quad q(z) = \sum_{k=0}^{n-1} b_k z^k \quad \blacktriangleright \quad (pq)(z) = \sum_{k=0}^{2n-2} \underbrace{\left(\sum_{j=0}^{k} a_j b_{k-j}\right)}_{=:c_k} z^k \tag{5.1.2}$$

➢ coefficients of product polynomial by discrete convolution of coefficients of polynomial factors! ◇

Definition 5.1.2 (n-periodic sequence/n-periodic signal)**.** *Let $\mathbb{I} = \mathbb{N}_0$ or $\mathbb{Z}$. Then a sequence (signal) $x_j \in \mathbb{C}$, $j \in \mathbb{I}$, is called n-period if $x_{j+n} = x_j$ for all $j \in \mathbb{I}$.*

n-periodic signal $(x_j)_{j\in\mathbb{Z}}$ fixed by $x_0, \dots, x_{n-1} \leftrightarrow$ vector $(x_0, \dots, x_{n-1}) \in \mathbb{C}^n$.

Definition 5.1.3 (Discrete periodic convolution)**.**
For two n-periodic sequences $\mathbf{h} = (h_k)_{k\in\mathbb{Z}}$ and $\mathbf{x} = (x_k)_{k\in\mathbb{Z}}$ we call the sequence $\mathbf{h} *_n \mathbf{x} = \big((h *_n x)_k\big)_{k\in\mathbb{Z}}$ defined by

$$(h *_n x)_k := \sum_{j=0}^{n-1} h_{k-j}\, x_j = \sum_{j=0}^{n-1} h_j\, x_{k-j}$$

for all $k \in \mathbb{Z}$ the discrete n-periodic convolution of $\mathbf{h}$ and $\mathbf{x}$.
In addition, for every $\mathbf{h} = (h_0, \dots, h_{n-1}) \in \mathbb{C}^n$ and $\mathbf{x} = (x_0, \dots, x_{n-1}) \in \mathbb{C}^n$ we define the discrete n-periodic convolution $\mathbf{h} *_n \mathbf{x} = \big((h *_n x)_0, (h *_n x)_1, \dots, (h *_n x)_{n-1}\big) \in \mathbb{C}^n$ of $\mathbf{h}$ and $\mathbf{x}$ by

$$(h * x)_k = \sum_{j=0}^{k} h_{k-j}\, x_j + \sum_{j=k+1}^{n} h_{n+k-j}\, x_j$$

for all $k \in \{0, 1, \dots, n-1\}$.

Matrix notation:

$$\begin{pmatrix} y_0 \\ \vdots \\ \\ \\ \vdots \\ y_{n-1} \end{pmatrix} = \underbrace{\begin{pmatrix} h_0 & h_{n-1} & h_{n-2} & \cdots & & & \cdots & h_1 \\ h_1 & h_0 & h_{n-1} & & & & & \vdots \\ h_2 & h_1 & h_0 & \ddots & & & & \\ \vdots & & \ddots & \ddots & \ddots & & & \\ & & & \ddots & \ddots & & \ddots & \\ \vdots & & & & & \ddots & \ddots & h_{n-1} \\ h_{n-1} & \cdots & & & & & h_1 & h_0 \end{pmatrix}}_{=:\mathbf{H}} \begin{pmatrix} x_0 \\ \vdots \\ \\ \\ \vdots \\ x_{n-1} \end{pmatrix} . \tag{5.1.3}$$

Definition 5.1.4 (Circulant matrix)**.**

A matrix $\mathbf{C} = (c_{i,j})_{i,j\in\{1,\dots,n\}} \in \mathbb{C}^{n,n}$ *is circulant* (ger.: *zirkulant*) *if*

$$\exists\ n\text{-periodic sequence } (u_k)_{k\in\mathbb{Z}} : \quad \forall\, i,j \in \{1,\dots,n\}: \quad c_{i,j} = u_{i-j}.$$

Structure of circulant matrix ▷

$$\begin{pmatrix} u_0 & u_{n-1} & u_{n-2} & \cdots & & \cdots & u_1 \\ u_1 & u_0 & & & & & u_2 \\ u_2 & & & & & & \vdots \\ \vdots & & & & & & \\ \vdots & & & & & & \vdots \\ u_{n-2} & & & & & & u_{n-1} \\ u_{n-1} & u_{n-2} & \cdots & & \cdots & u_1 & u_0 \end{pmatrix}$$

Remark 5.1.5 (Reformulation of discrete convolution as discrete periodic convolution). Discrete convolution $\mathbf{h} * \mathbf{x} = \big((h*x)_0, \dots, (h*x)_{2n-2}\big) \in \mathbb{C}^{2n-1}$ (→ Def. 5.1.1) of $\mathbf{h} = (h_0, \dots, h_{n-1}) \in \mathbb{C}^n$ and $\mathbf{x} = (x_0, \dots, x_{n-1}) \in \mathbb{C}^n$:

$$\forall\, k \in \{0, 1, \dots, 2n-2\}: \qquad (h*x)_k = \sum_{\substack{0 \le j \le n-1 \\ 0 \le k-j \le n-1}} h_j\, x_{k-j}\,.$$

Expand $\mathbf{h} = (h_0, \dots, h_{n-1})$, $\mathbf{x} = (x_0, \dots, x_{n-1})$ by zero padding, i.e., define $\widetilde{\mathbf{h}} = (h_0, \dots, h_{2n-2}) \in \mathbb{C}^{2n-1}$ and $\widetilde{\mathbf{x}} = (x_0, \dots, x_{2n-2}) \in \mathbb{C}^{2n-1}$ by

$$\widetilde{h}_k := \begin{cases} h_k & : 0 \le k < n \\ 0 & : n \le k < 2n-1 \end{cases}, \qquad \widetilde{x}_k := \begin{cases} x_k & : 0 \le k < n \\ 0 & : n \le k < 2n-1 \end{cases} \tag{5.1.6}$$

for all $k \in \{0, 1, \dots, 2n-2\}$.

$$\blacktriangleright \qquad \mathbf{h} * \mathbf{x} = \widetilde{\mathbf{h}} *_{2n-1} \widetilde{\mathbf{x}}\,. \tag{5.1.7}$$

Matrix view of reformulation as discrete $(2n-1)$-periodic convolution (*cf.* (5.1.1)):

$$\begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ y_{2n-3} \\ y_{2n-2} \end{pmatrix} = \underbrace{\begin{pmatrix} h_0 & 0 & \cdots & 0 & 0 & h_{n-1} & h_{n-2} & \cdots & h_1 \\ h_1 & h_0 & \ddots & \vdots & 0 & 0 & h_{n-1} & \ddots & h_2 \\ \vdots & \ddots & \ddots & 0 & \vdots & \vdots & \ddots & \ddots & \vdots \\ h_{n-2} & \cdots & h_1 & h_0 & 0 & 0 & \cdots & 0 & h_{n-1} \\ h_{n-1} & h_{n-2} & \cdots & h_1 & h_0 & 0 & 0 & \cdots & 0 \\ 0 & h_{n-1} & h_{n-2} & \cdots & h_1 & h_0 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & h_{n-1} & h_{n-2} & \cdots & h_1 & h_0 & 0 \\ 0 & 0 & \cdots & 0 & h_{n-1} & h_{n-2} & \cdots & h_1 & h_0 \end{pmatrix}}_{\text{a } (2n-1)\times(2n-1) \text{ circulant matrix!}} \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-2} \\ x_{n-1} \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} .$$

$\triangle$

5.2 Discrete Fourier Transform (DFT)

Definition 5.2.1 (nth root of the unity)**.**
By $\omega_n \in \mathbb{C}$ *we denote the complex number* $\omega_n := \exp(-\frac{2\pi i}{n}) = \cos(\frac{2\pi}{n}) - i\sin(\frac{2\pi}{n})$.

Lemma 5.2.2 (Properties of ω_n)**.** *It holds that*

$$\overline{\omega}_n = \frac{1}{\omega_n} \quad , \quad \boxed{\omega_n^n = 1} \quad , \quad \omega_n^{n/2} = -1 \quad , \quad \omega_n^k = \omega_n^{k+n} \quad , \quad \sum_{l=0}^{n-1} \omega_n^{kl} = \begin{cases} n & : k = 0 \\ 0 & : k \neq 0 \end{cases}$$

for all $k \in \mathbb{Z}$.

Lemma 5.2.3 (Eigenvalues and eigenvectors of circulant matrices)**.**
Let $\mathbf{C} = (c_{i,j})_{i,j\in\{1,\dots,n\}} \in \mathbb{C}^{n,n}$ *be a circulant matrix (→ Def. 5.1.4), let* $k \in \mathbb{Z}$ *and let* $\mathbf{v} = \left(\omega_n^{0k}, \omega_n^{1k}, \dots, \omega_n^{(n-1)k}\right) \in \mathbb{C}^n$*. Then*

$$\mathbf{C}v = \left(\sum_{l=0}^{n-1} c_{l+1,1}\,\omega_n^{-lk}\right) v\,.$$

Proof of Lemma 5.2.3. Let $u_k \in \mathbb{C}$, $k \in \mathbb{Z}$, be an n-periodic sequence with $c_{i,j} = u_{i-j}$ and let $(y_0, \dots, y_{n-1}) := \mathbf{C}\mathbf{v} \in \mathbb{C}^n$. Then

$$y_j = \sum_{l=1}^{n} c_{j+1,l}\,\omega_n^{(l-1)k} = \sum_{l=0}^{n-1} u_{j-l}\,\omega_n^{lk} = \omega_n^{jk}\left[\sum_{l=0}^{n-1} u_{j-l}\,\omega_n^{(l-j)k}\right] = \omega_n^{jk}\left[\sum_{l=j+1-n}^{j} u_l\,\omega_n^{-lk}\right]$$

$$= \omega_n^{jk}\left[\sum_{l=j+1-n}^{-1} u_l\,\omega_n^{-lk} + \sum_{l=0}^{j} u_l\,\omega_n^{-lk}\right] = \omega_n^{jk}\left[\sum_{l=j+1}^{n-1} u_l\,\omega_n^{-lk} + \sum_{l=0}^{j} u_l\,\omega_n^{-lk}\right] = \omega_n^{jk}\left[\sum_{l=0}^{n-1} \frac{u_l}{\omega_n^{lk}}\right]$$

for all $j \in \{0, \dots, n-1\}$.
This completes the proof of Lemma 5.2.3. □

Definition 5.2.4 (Fourier matrix and discrete Fourier transform (DFT)). *The matrix* $\mathbf{F}_n \in \mathbb{C}^{n,n}$ *defined by*

$$\mathbf{F}_n = \begin{pmatrix} \omega_n^0 & \omega_n^0 & \omega_n^0 & \cdots & \omega_n^0 \\ \omega_n^0 & \omega_n^1 & \omega_n^2 & \cdots & \omega_n^{n-1} \\ \omega_n^0 & \omega_n^2 & \omega_n^4 & \cdots & \omega_n^{2(n-1)} \\ \omega_n^0 & \omega_n^3 & \omega_n^6 & \cdots & \omega_n^{3(n-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ \omega_n^0 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \cdots & \omega_n^{(n-1)(n-1)} \end{pmatrix} = \left(\omega_n^{(l-1)(k-1)}\right)_{(l,k)\in\{1,\dots,n\}^2} \in \mathbb{C}^{n,n} \tag{5.2.3}$$

is called $(n \times n)$*-Fourier matrix. In addition, the linear map* $\mathbb{C}^n \ni \mathbf{y} \mapsto \mathbf{F}_n(y) = \mathbf{F}_n\,\mathbf{y} \in \mathbb{C}^n$ *is called discrete Fourier transform (DFT).*

Note that if $\mathbf{y} = (y_0, \dots, y_{n-1}) \in \mathbb{C}^n$ and if $\mathbf{c} = (c_0, \dots, c_{n-1}) := \mathbf{F}_n\mathbf{y}$, then

$$\forall\, k \in \{0, 1, \dots, n-1\}: \qquad c_k = \sum_{l=0}^{n-1} y_l\, \omega_n^{kl} = \sum_{l=0}^{n-1} y_l\, e^{\frac{-i2\pi kl}{n}} \quad . \tag{5.2.4}$$

Terminology: $\mathbf{c} = \mathbf{F}_n\mathbf{y}$ is also called the (discrete) Fourier transform of $\mathbf{y}$

MATLAB-functions for discrete Fourier transform (and its inverse):

Lemma 5.2.5 (Properties of Fourier matrix)**.** *The scaled Fourier-matrix* $\frac{1}{\sqrt{n}}\mathbf{F}_n$ *is unitary and it holds that* $\mathbf{F}_n^{-1} = \frac{1}{n}\mathbf{F}_n^H = \frac{1}{n}\overline{\mathbf{F}}_n$.

Remark 5.2.2 (Multi dimensional discrete fourier transform)*.* The MATLAB command `fft2` can be used to compute the two dimensional discrete fourier transform, that is, the discrete fourier transform of a matrix. It is based on iterated applications of the "one-dimensional" discrete fourier transform through the MATLAB command `fft`. More details can, e.g., be obtain through the MATLAB command `help fft2`.

5.2.1 Discrete convolution via DFT

Lemma 5.2.6 (Diagonalization of circulant matrices ($\rightarrow$ Def. 5.1.4)).
For any circulant matrix $\mathbf{C} = (c_{i,j})_{i,j\in\{1,\dots,n\}} \in \mathbb{C}^{n,n}$ *it holds that*

$$\boxed{\mathbf{C} = \mathbf{F}_n^{-1}\operatorname{diag}\big(\mathbf{F}_n(c_{1,1},\dots,c_{n,1})\big)\,\mathbf{F}_n} \;.$$

In particular, it holds that

$$\forall\,\mathbf{x},\mathbf{y}\in\mathbb{C}^n: \qquad \mathbf{x} *_n \mathbf{y} = \mathbf{F}_n^{-1}\Big(\operatorname{diag}\big(\mathbf{F}_n(\mathbf{x})\big)\,\mathbf{F}_n(y)\Big) \;.$$

Proof of Lemma 5.2.6. Lemma 5.2.3 implies

$$\begin{aligned}\mathbf{C} &= \overline{\mathbf{F}}_n\operatorname{diag}\big(\mathbf{F}_n(c_{1,1},\dots,c_{n,1})\big)\,\big(\overline{\mathbf{F}}_n\big)^{-1} = n\mathbf{F}_n^{-1}\operatorname{diag}\big(\mathbf{F}_n(c_{1,1},\dots,c_{n,1})\big)\,\big(n\mathbf{F}_n^{-1}\big)^{-1}\\ &= \mathbf{F}_n^{-1}\operatorname{diag}\big(\mathbf{F}_n(c_{1,1},\dots,c_{n,1})\big)\,\mathbf{F}_n \;.\end{aligned}$$

This completes the proof of Lemma 5.2.6. $\square$

Code 5.1: discrete periodic convolution: straightforward implementation

```
function z=pconv(u,x)
n = length(x); z = zeros(n,1);
for i=1:n, z(i)=dot(conj(u),
  x([i:-1:1,n:-1:i+1]));
end
```

Code 5.2: discrete periodic convolution: DFT implementation

```
function z=pconvfft(u,x)
z = ifft(fft(u).*fft(x));
```

Implementation of discrete convolution (→ Def. 5.1.1) based on periodic discrete convolution ▷

Built-in MATLAB-function:

```
y = conv(h,x);
```

Code 5.3: discrete convolution: DFT implementation

```
function y = myconv(h,x)
n = length(h);
%Zero padding
h = [h;zeros(n-1,1)]; x =
  [x;zeros(n-1,1)];
%Periodic discrete convolution of length 2n-1
y = pconvfft(h,x);
```

5.2.2 Fast Fourier Transform (FFT)

At first glance (at (5.3.2)): DFT in $\mathbb{C}^n$ seems to require asymptotic computational effort of $O(n^2)$ (matrix$\times$vector multiplication with dense matrix).

Example 5.2.8 (Efficiency of `fft`).

`tic-toc`-timing in MATLAB: compare `fft`, loop based implementation, and direct matrix multiplication

(MATLAB V6.5, Linux, Mobile Intel Pentium 4 - M CPU 2.40GHz, minimum over 5 runs)

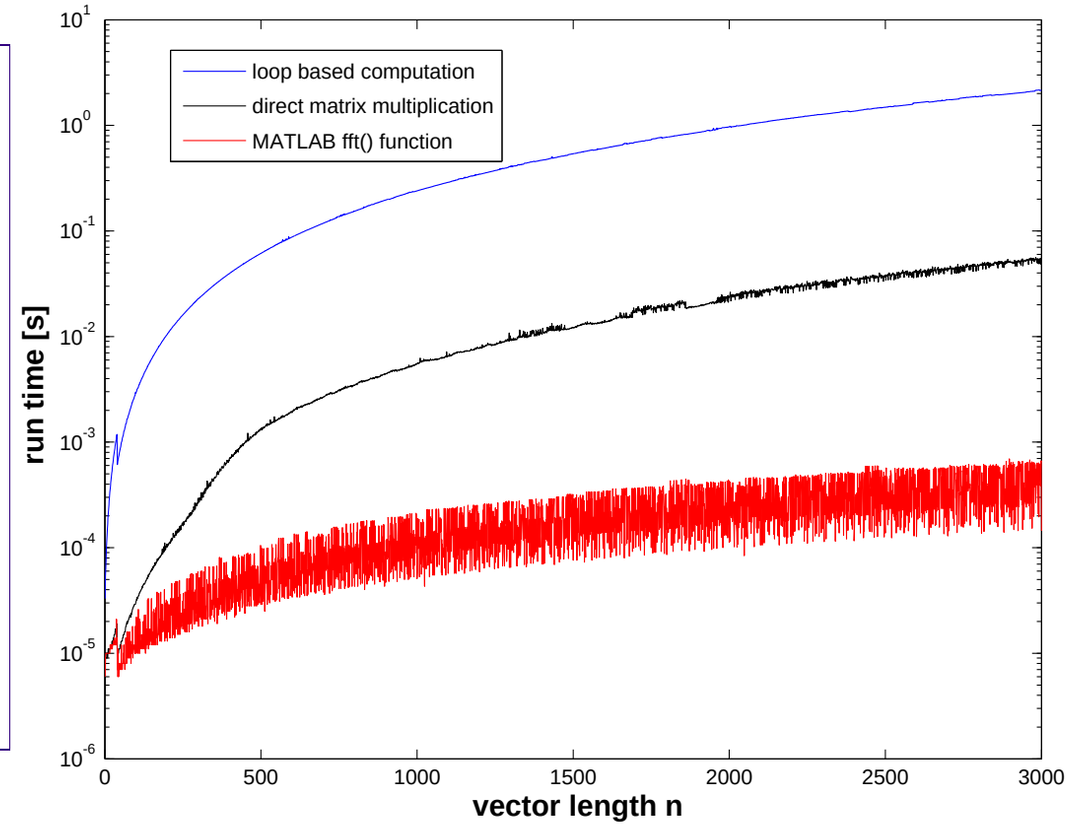
Code 5.4: timing of different implementations of DFT

```
res = [];
for n=1:1:3000,  y = rand(n,1); c = zeros(n,1);
  t1 = realmax; for k=1:5, tic;
    omega = exp(-2*pi*i/n); c(1) = sum(y); s = omega;
    for j=2:n, c(j) = y(n);
      for k=n-1:-1:1, c(j) = c(j)*s+y(k);  end
      s = s*omega;
    end
    t1 = min(t1,toc);
  end
  [I,J] = meshgrid(0:n-1,0:n-1); F = exp(-2*pi*i*I.*J/n);
```

```
  t2 = realmax; for k=1:5, tic; c = F*y; t2 = min(t2,toc); end
  t3 = realmax; for k=1:5, tic; d = fft(y); t3 = min(t3,toc); end
  res = [res; n t1 t2 t3];
end

figure('name','FFT timing');
semilogy(res(:,1),res(:,2),'b-',res(:,1),res(:,3),'k-',
  res(:,1),res(:,4),'r-');
ylabel('{\bf run time [s]}','Fontsize',14);
xlabel('{\bf vector length n}','Fontsize',14);
legend('loop based computation','direct matrix multiplication','MATLAB
  fft() function',1);
print -deps2c '../PICTURES/ffttime.eps'
```

MATLAB-CODE naive DFT-implementation

```
c = zeros(n,1);
omega = exp(-2*pi*i/n);
c(1) = sum(y); s = omega;
for j=2:n
  c(j) = y(n);
  for k=n-1:-1:1
    c(j) = c(j)*s+y(k);
  end
  s = s*omega;
end
```



Incredible! The MATLAB `fft()`-function clearly beats the $O(n^2)$ asymptotic complexity of the other implementations. Note the logarithmic scale!

◇

The secret of MATLAB's `fft()`:

the Fast Fourier Transform algorithm [15]

(discovered by C.F. Gauss in 1805, rediscovered by Cooley & Tuckey in 1965, one of the "top ten algorithms of the century").

An elementary manipulation of (5.3.2) for $n = 2m$, $m \in \mathbb{N}$:

$$\forall\, k \in \{0,\dots,n-1\}: \quad c_k = \sum_{j=0}^{n-1} y_j\, e^{-\frac{2\pi i}{n}jk} = \sum_{j=0}^{m-1} y_{2j}\, e^{-\frac{2\pi i}{n}2jk} + \sum_{j=0}^{m-1} y_{2j+1}\, e^{-\frac{2\pi i}{n}(2j+1)k}$$

$$= \sum_{j=0}^{m-1} y_{2j}\, \underbrace{e^{-\frac{2\pi i}{m}jk}}_{=\omega_m^{jk}} + e^{-\frac{2\pi i}{n}k} \left[\sum_{j=0}^{m-1} y_{2j+1}\, \underbrace{e^{-\frac{2\pi i}{m}jk}}_{=\omega_m^{jk}} \right]$$

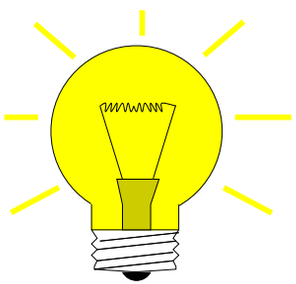
$$= \underbrace{\left[\sum_{j=0}^{m-1} y_{2j}\, \omega_m^{jk} \right]}_{=:\widetilde{c}_k^{\mathtt{even}}} + e^{-\frac{2\pi i}{n}k} \underbrace{\left[\sum_{j=0}^{m-1} y_{2j+1}\, \omega_m^{jk} \right]}_{=:\widetilde{c}_k^{\mathtt{odd}}} . \tag{5.2.5}$$

Note m-periodicity: $\forall\, k \in \{0,\dots,m-1\}: \quad \widetilde{c}_k^{\mathtt{even}} = \widetilde{c}_{k+m}^{\mathtt{even}}$ and $\widetilde{c}_k^{\mathtt{odd}} = \widetilde{c}_{k+m}^{\mathtt{odd}}$.

Note: $(\widetilde{c}_0^{\mathtt{even}}, \dots, \widetilde{c}_{m-1}^{\mathtt{even}})$ and $(\widetilde{c}_0^{\mathtt{odd}}, \dots, \widetilde{c}_{m-1}^{\mathtt{odd}})$ from DFTs of length m, i.e.,

$$(\widetilde{c}_0^{\mathtt{even}}, \dots, \widetilde{c}_{m-1}^{\mathtt{even}}) = \mathbf{F}_m \underbrace{(y_0, y_2, \dots, y_{n-2})}_{=:\mathbf{y}_{\mathtt{even}} \in \mathbb{C}^m}, \quad (\widetilde{c}_0^{\mathtt{odd}}, \dots, \widetilde{c}_{m-1}^{\mathtt{odd}}) = \mathbf{F}_m \underbrace{(y_1, y_3, \dots, y_{n-1})}_{=:\mathbf{y}_{\mathtt{odd}} \in \mathbb{C}^m} !$$

(5.2.5): DFT of length $2m$ = 2× DFT of length m + 2m additions & multiplications



Idea:

divide & conquer recursion

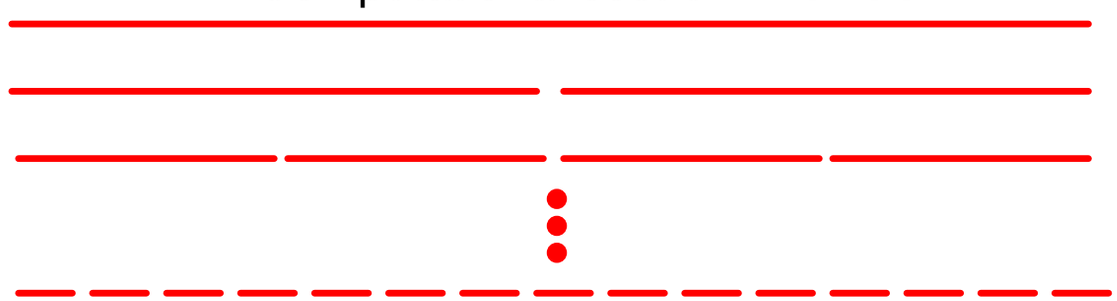
(for DFT of length $n = 2^L$)

FFT-algorithm

Code 5.5: Recursive FFT

```
function c = fftrec(y)
n = length(y);
if (n == 1), c = y; return;
else
  c1 = fftrec(y(1:2:n));
  c2 = fftrec(y(2:2:n));
  c = [c1;c1] +
    (exp(-2*pi*i/n).^((0:n-1)'))
    .*[c2;c2];
end
```

Computational cost of `fftrec`:



$1\times$ DFT of length 2^L

$2\times$ DFT of length 2^{L-1}

$4\times$ DFT of length 2^{L-2}

$2^L\times$ DFT of length 1

Code 5.5: each level of the recursion requires $O(2^L)$ elementary operations.

Asymptotic complexity of FFT algorithm, $n = 2^L$: $O(L2^L) = O(n\log_2(n))$

(MATLAB `fft`-function: cost $\approx 5n\log_2(n)$).

➢

What if $n \neq 2^L$? Quoted from MATLAB manual:

To compute an n-point DFT when n is composite (that is, when $n = pq$), the FFTW library decomposes the problem using the Cooley-Tukey algorithm, which first computes p transforms of size q, and then computes q transforms of size p. The decomposition is applied recursively to both the p- and q-point DFTs until the problem can be solved using one of several machine-generated fixed-size "codelets." The codelets in turn use several algorithms in combination, including a

variation of Cooley-Tukey, a prime factor algorithm, and a split-radix algorithm. The particular factorization of n is chosen heuristically.

The execution time for fft depends on the length of the transform. It is fastest for powers of two. It is almost as fast for lengths that have only small prime factors. It is typically several times slower for lengths that are prime or which have large prime factors $\rightarrow$ Ex. 5.2.8.

Remark 5.2.12 (FFT based on general factorization). Fast Fourier transform algorithm for DFT of length $n = pq$ where $p, q \in \mathbb{N}$ (Cooley-Tuckey-Algorithm) $\forall\, k \in \{0, \dots, n-1\}$:

$$\begin{aligned} c_k &= \sum_{j=0}^{n-1} y_j\, \omega_n^{jk} = \sum_{l=0}^{q-1} \sum_{m=0}^{p-1} y_{lp+m}\, \omega_n^{(lp+m)k} = \sum_{m=0}^{p-1} \omega_n^{mk} \left[\sum_{l=0}^{q-1} y_{lp+m}\, \omega_n^{lpk} \right] \\ &= \sum_{m=0}^{p-1} \omega_n^{mk} \left[\sum_{l=0}^{q-1} y_{lp+m}\, \omega_q^{lk} \right] = \sum_{m=0}^{p-1} \omega_n^{mk} \left[\sum_{l=0}^{q-1} y_{lp+m}\, \omega_q^{l(k \mod q)} \right] \end{aligned} \tag{5.2.6}$$

Step I: perform p DFTs of length q : $\quad z_{m,k} := \sum_{l=0}^{q-1} y_{lp+m}\, \omega_q^{lk}, \quad 0 \le m < p, \quad 0 \le k < q.$

Step II: for $k =: rq + s, \quad 0 \le r < p, \quad 0 \le s < q$:

$$c_k = c_{rq+s} = \sum_{m=0}^{p-1} \omega_n^{mk}\, z_{m,(k \mod q)} = \sum_{m=0}^{p-1} \omega_n^{m(rq+s)}\, z_{m,s} = \sum_{m=0}^{p-1} \left(\omega_n^{ms} z_{m,s} \right) \omega_p^{mr}$$

and thus q DFTs of length p give $(c_0, c_1, \dots, c_{n-1})$.

△

Asymptotic complexity of `c=fft(y)` for $\mathbf{y} \in \mathbb{C}^n$ = $O(n \log n)$.

← Sect. 5.2.1

Asymptotic complexity of discrete periodic convolution/multiplication with circulant matrix, see Code 5.2:

Cost(`z = pconvfft(u,x)`, $\mathbf{u}, \mathbf{x} \in \mathbb{C}^n$) = $O(n \log n)$.

Asymptotic complexity of discrete convolution, see Code 5.3:

Cost(`z = myconv(h,x)`, $\mathbf{h}, \mathbf{x} \in \mathbb{C}^n$) = $O(n \log n)$.

5.2.3 Frequency filtering via DFT

Definition 5.2.7 ((Complex) Fourier basis vectors)**.** *The vectors* $\mathbf{f}_0^{(n)}, \mathbf{f}_1^{(n)}, \dots, \mathbf{f}_{n-1}^{(n)} \in \mathbb{C}^n$ *defined through*

$$\mathbf{f}_k^{(n)} := \left(\omega_n^{-0k}, \omega_n^{-k}, \dots, \omega_n^{-(n-1)k}\right) = \left(1, \exp\left(\tfrac{k2\pi i}{n}\right), \exp\left(\tfrac{2k2\pi i}{n}\right), \dots, \exp\left(\tfrac{(n-1)k2\pi i}{n}\right)\right)$$

$$= \left(1, \cos\left(\tfrac{k2\pi}{n}\right) + i\,\sin\left(\tfrac{k2\pi}{n}\right), \cos\left(\tfrac{2k2\pi}{n}\right) + i\,\sin\left(\tfrac{2k2\pi}{n}\right), \dots, \cos\left(\tfrac{(n-1)k2\pi}{n}\right) + i\,\sin\left(\tfrac{(n-1)k2\pi}{n}\right)\right)$$

$$= \Big(\cos\big(k2\pi x\big) + i\,\sin\big(k2\pi x\big)\Big)_{x=0,\frac{1}{n},\dots,\frac{(n-1)}{n}}$$

for all $k \in \{0, 1, \dots, n-1\}$ *are called (complex) Fourier basis vectors.*

Observe that the vectors $\overline{\mathbf{f}_0^{(n)}}, \overline{\mathbf{f}_1^{(n)}}, \dots, \overline{\mathbf{f}_{n-1}^{(n)}} \in \mathbb{C}^n$ are the column vectors of the matrix $\mathbf{F}_n$. Lemma 5.2.5 hence shows that the vectors $\mathbf{f}_0^{(n)}, \mathbf{f}_1^{(n)}, \dots, \mathbf{f}_n^{(n)} \in \mathbb{C}^n$ are an orthogonal basis of the $\mathbb{C}^n$. Moreover, note that $|\mathbf{f}_k^{(n)}| = \sqrt{n}$ for all $k \in \{0, 1, \dots, n-1\}$.

In addition, if $\mathbf{y} = (y_0, y_1, \dots, y_{n-1}) \in \mathbb{C}^n$ and if $\mathbf{c} = (c_0, c_1, \dots c_{n-1}) = \mathbf{F}_n\mathbf{y}$, then

$$\forall\, k \in \{0, 1, \dots, n-1\}: \qquad \langle \mathbf{f}_k^{(n)}, \mathbf{y}\rangle = \sum_{l=0}^{n-1} \overline{\omega_n^{-lk}}\, y_l = \sum_{l=0}^{n-1} y_l\, \omega_n^{lk} = c_k$$

and $$\mathbf{y} = \sum_{k=0}^{n-1} c_k \left[\frac{1}{n} \mathbf{f}_k^{(n)} \right] . \tag{5.2.7}$$

The Fourier basis vectors, when interpreted as time-periodic signals, represent harmonic oscillations. This is illustrated when plotting some vectors of the Fourier basis ($n = 16$):

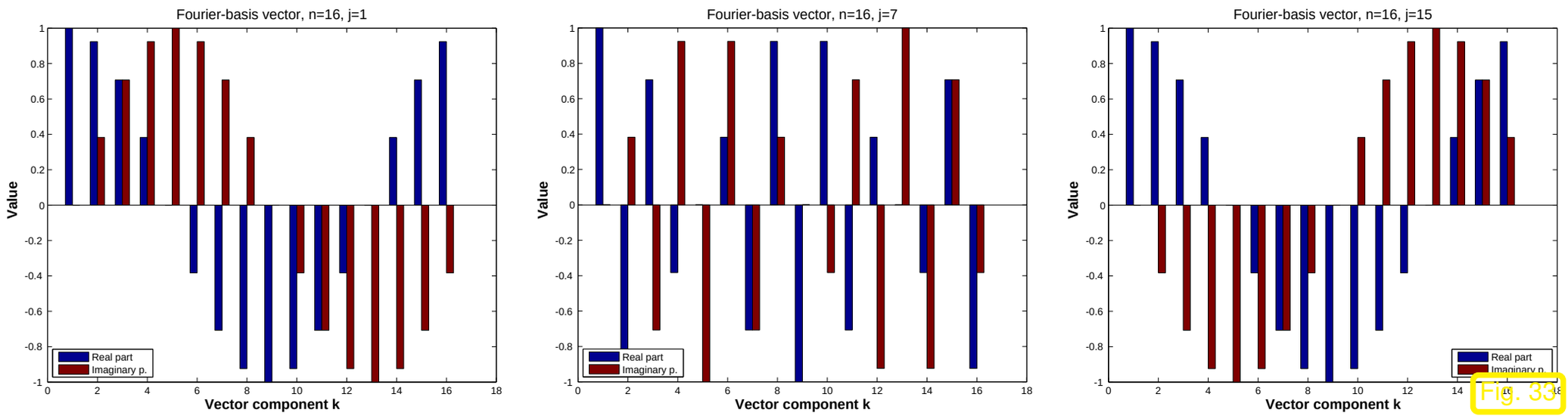


Fig. 33

"slow oscillation/low frequency" "fast oscillation/high frequency" "slow oscillation/low frequency"

- Dominant coefficients of a signal after transformation to trigonometric basis indicate dominant frequency components.

Terminology: coefficients of a signal w.r.t. $\frac{1}{n}$-scaled Fourier basis = signal in frequency domain (*ger.*: Frequenzbereich) (cf. (5.2.7)), original signal = time domain (*ger.*: Zeitbereich).

If $\mathbf{y} \in \mathbb{C}^n$ is a signal and if $\mathbf{c} = (c_0, \dots, c_{n-1}) = \mathbf{F}_n \mathbf{y}$ is the discrete Fourier transform of $\mathbf{y}$, then $\mathbf{c} = (c_0, \dots, c_{n-1})$ represents the signal $\mathbf{y}$ in frequency domain (spectrum) (cf. (5.2.7)) and the vector $\left(|c_0|^2, \dots, |c_{n-1}|^2\right) \in \mathbb{R}^n$ is called power spectrum of the signal $\mathbf{y} \in \mathbb{C}^n$.

Example 5.2.14 (Frequency identification with DFT). Extraction of characteristical frequencies from a distorted discrete periodical signal:

```
t = 0:63; x = sin(2*pi*t/64)+sin(7*2*pi*t/64);
y = x + randn(size(t)); %distortion
```

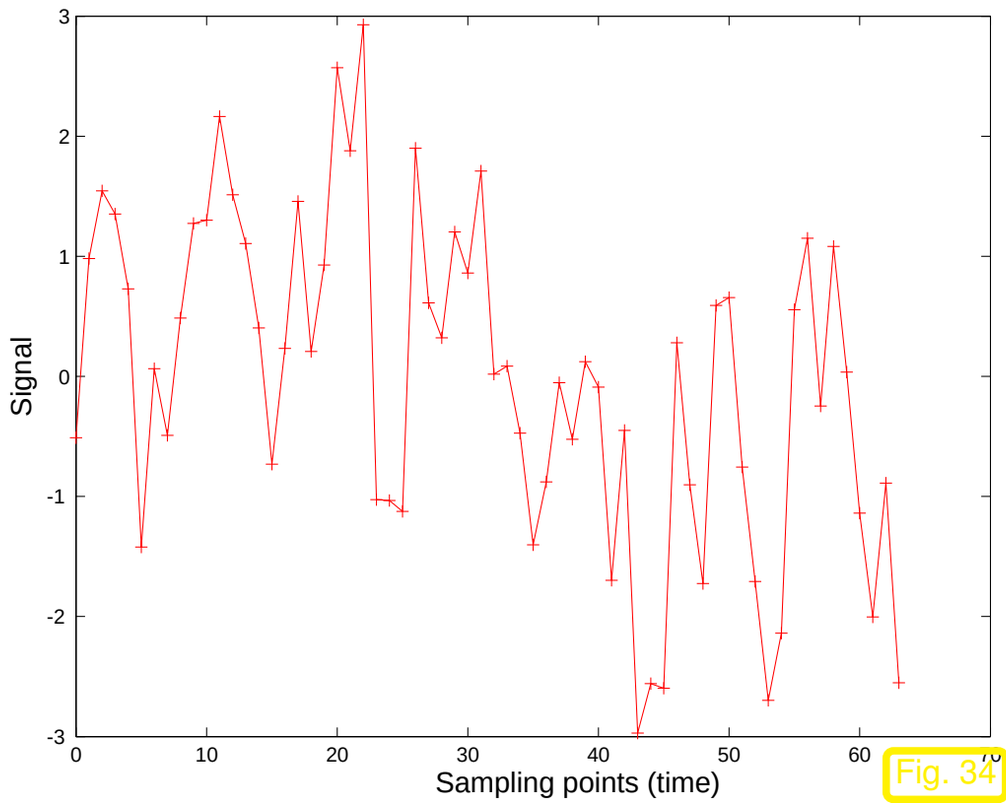


Fig. 34

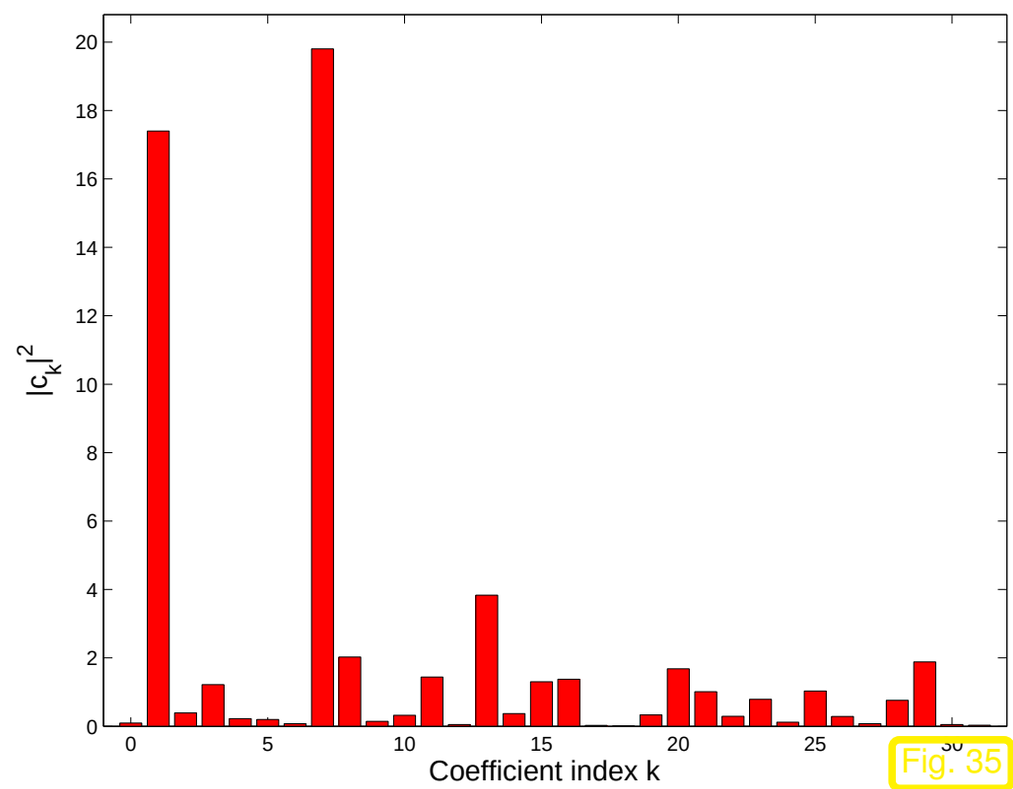
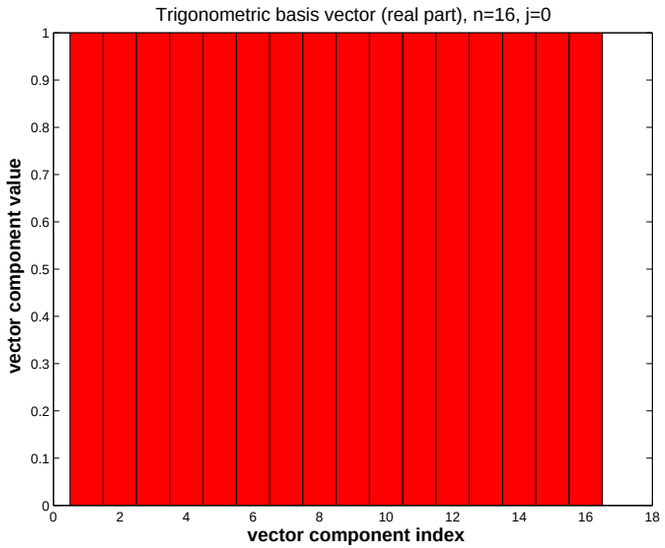


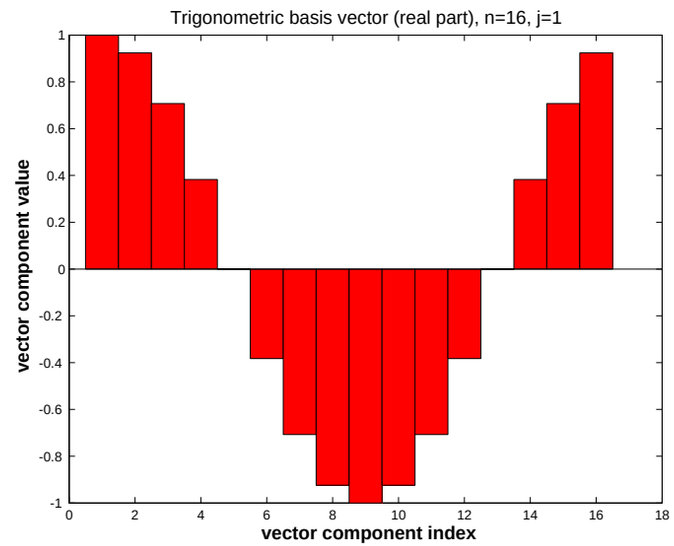
Fig. 35

Frequencies present in unperturbed signal become evident in frequency domain. ◇

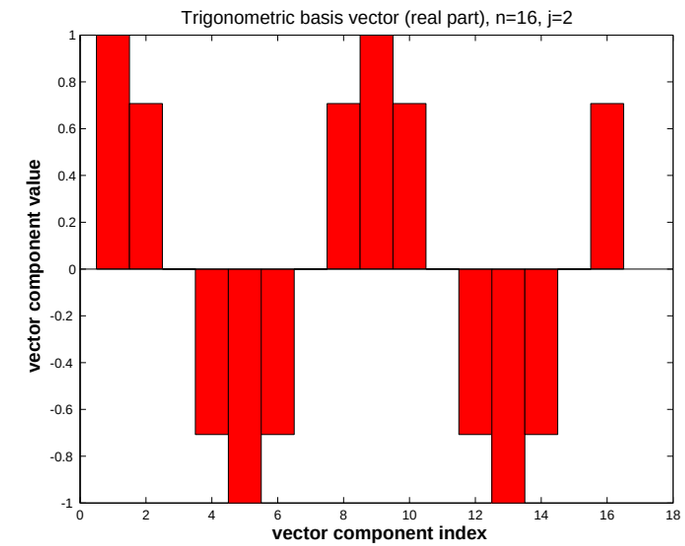
Remark 5.2.15 ("Low" and "high" frequencies). Plots of real parts of Fourier basis vectors $\mathbf{f}_j^{(n)}$ (= complex conjugate of columns of Fourier matrix), $j \in \{0, 1, \dots, 8\}$, $n = 16$.



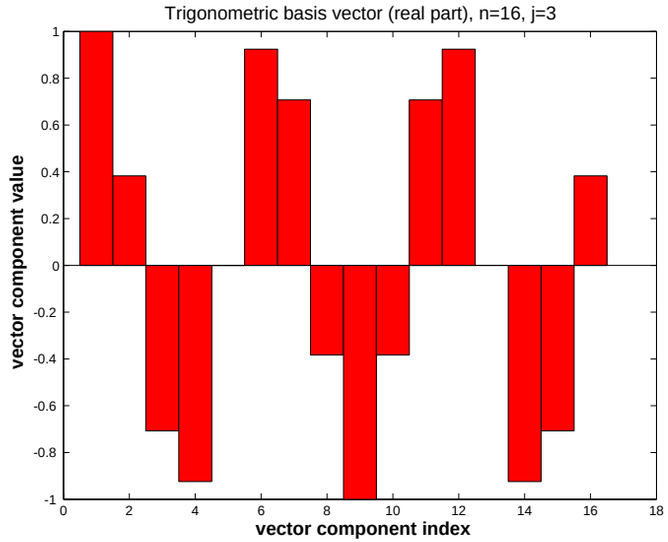
$\mathrm{Re}\left(\mathbf{f}_0^{(16)}\right)$



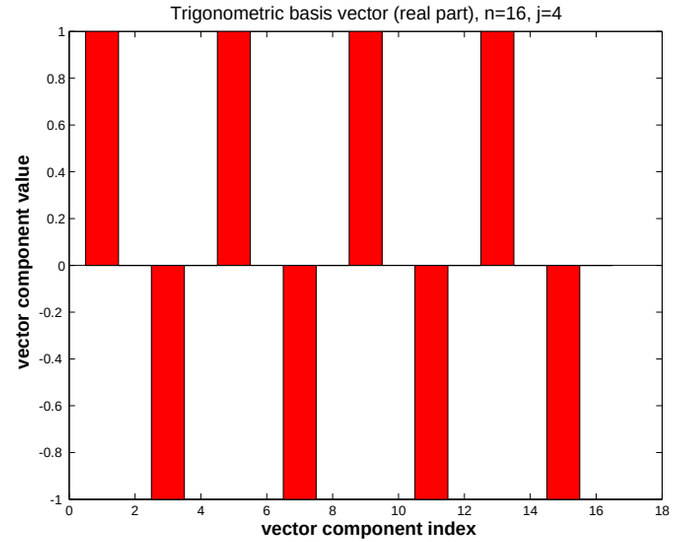
$\mathrm{Re}\left(\mathbf{f}_1^{(16)}\right)$



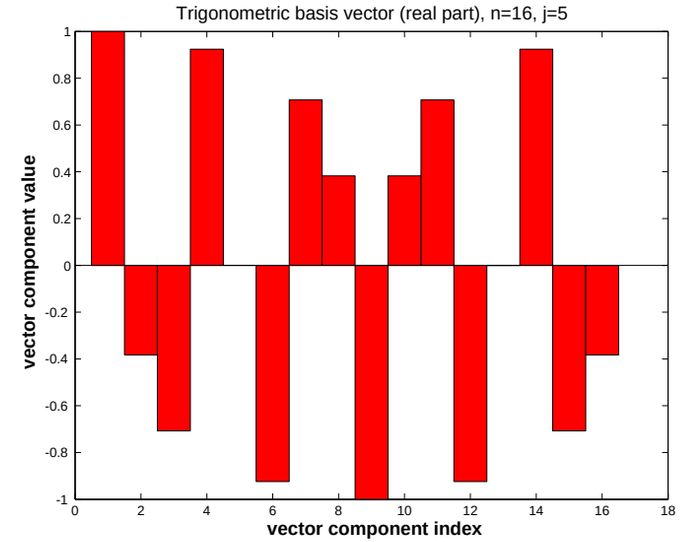
$\mathrm{Re}\left(\mathbf{f}_2^{(16)}\right)$



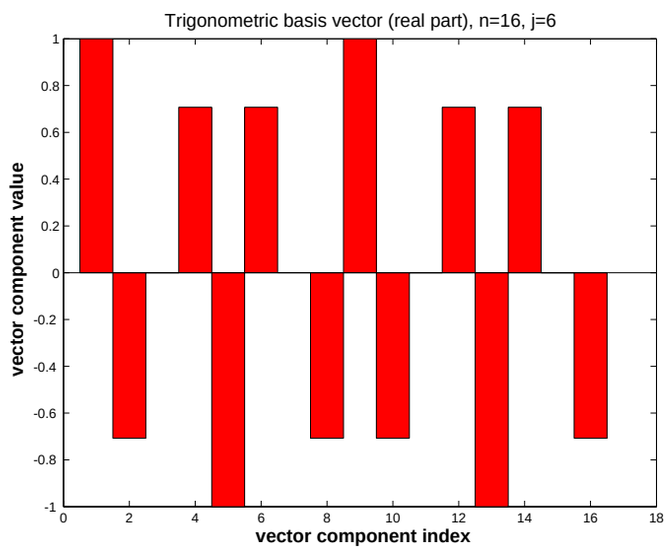
$$\mathrm{Re}\left(\mathbf{f}_3^{(16)}\right)$$



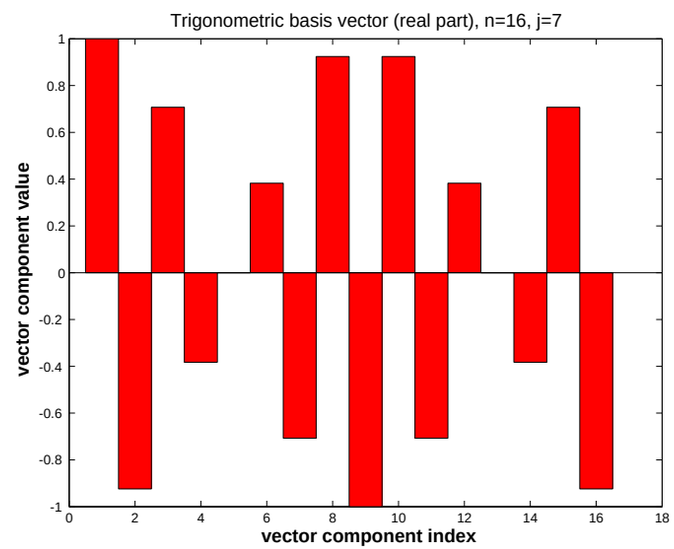
$$\mathrm{Re}\left(\mathbf{f}_4^{(16)}\right)$$



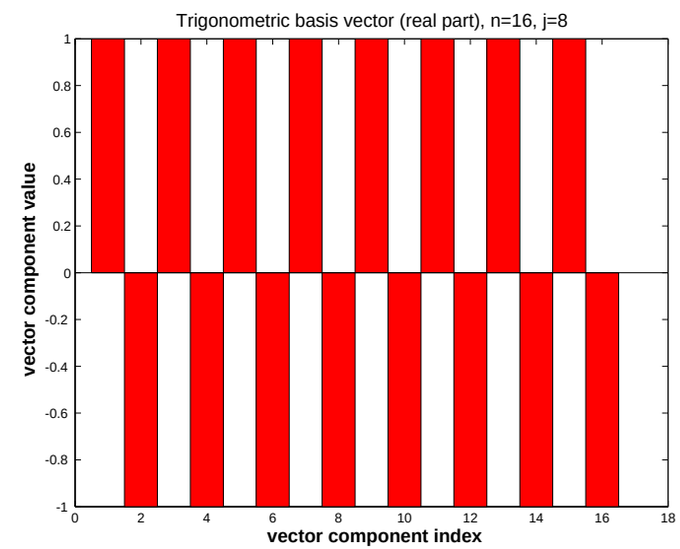
$$\mathrm{Re}\left(\mathbf{f}_5^{(16)}\right)$$



$$\mathrm{Re}\left(\mathbf{f}_6^{(16)}\right)$$



$$\mathrm{Re}\left(\mathbf{f}_7^{(16)}\right)$$



$$\mathrm{Re}\left(\mathbf{f}_8^{(16)}\right)$$

△

Note for every $j \in \{0, 1, \dots, n-1\}$ that

$$\mathrm{Re}\Big(\mathbf{f}_j^{(16)}\Big) = \big(\cos(2\pi j x)\big)_{x=0,\frac{1}{n},\dots,\frac{n-1}{n}} = \big(\cos(-2\pi j x)\big)_{x=0,\frac{1}{n},\dots,\frac{n-1}{n}}$$
$$= \Big(\mathrm{Re}\big(\omega_n^{jk}\big)\Big)_{k=0,1,\dots,n-1} = \Big(\mathrm{Re}\big(\omega_n^{-(n-j)k}\big)\Big)_{k=0,1,\dots,n-1} = \mathrm{Re}\Big(\mathbf{f}_{n-j}^{(16)}\Big) \ .$$

Slow oscillations/low frequencies $\leftrightarrow$ $j \approx 1$ and $j \approx n$.

Fast oscillations/high frequencies $\leftrightarrow$ $j \approx n/2$.

▶ Frequency filtering of real discrete periodic signals by suppressing certain "Fourier coefficients".

Code 5.6: DFT-based frequency filtering

```
function [low,high] =
  freqfilter(y,k)
m = length(y)/2; c = fft(y);
clow = c; clow(m+1-k:m+1+k) = 0;
chigh = c-clow;
low = real(ifft(clow));
high = real(ifft(chigh));
```

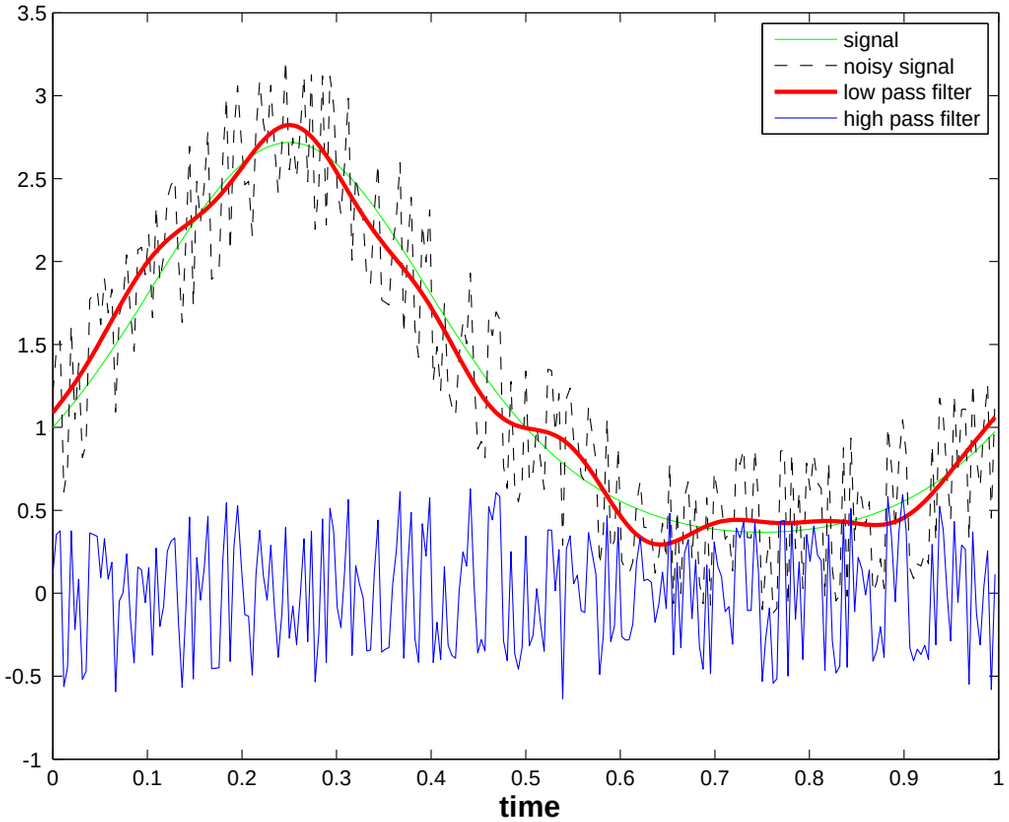
Map $\mathtt{y} \mapsto \mathtt{low}$ (in Code 5.6) $\hat{=}$ low pass filter (*ger.:* Tiefpass).

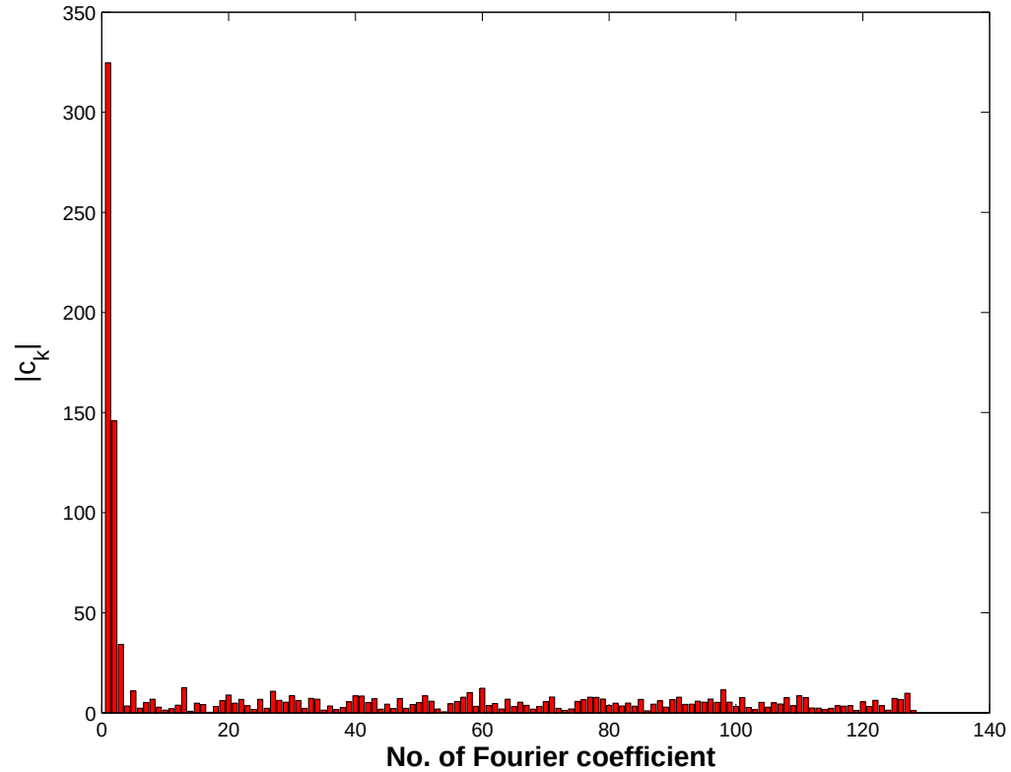
Map $\mathtt{y} \mapsto \mathtt{high}$ (in Code 5.6) $\hat{=}$ high pass filter (*ger.:* Hochpass).

Example 5.2.17 (Frequency filtering by DFT). Noisy signal:

```
n = 256; y = exp(sin(2*pi*((0:n-1)')/n)) + 0.5*sin(exp(1:n)');
```

Frequency filtering by Code 5.6 with $k = 120$.





Low pass filtering can be used for *denoising*, that is, the removal of high frequency perturbations of a signal.

◇

Example 5.2.18 (Sound filtering by DFT).

Code 5.7: DFT based sound compression

```
%Sound compression by DFT
%Example: ex:soundfilter

%Read sound data
[y,freq,nbits] = wavread('hello.wav');

n = length(y);
fprintf('Read wav File: %d samples, freq = %d, nbits = %d\n',
  n,freq,nbits);
k = 1; s{k} = y; leg{k} = 'Sampled signal';

c = fft(y);

figure('name','sound signal');
plot((22000:44000)/freq,s{1}(22000:44000),'r-');
title('samples sound signal','fontsize',14);
xlabel('{\bf time[s]}','fontsize',14);
ylabel('{\bf sound pressure}','fontsize',14);
grid on;

print -depsc2 '../PICTURES/soundsignal.eps';

figure('name','sound frequencies');
```

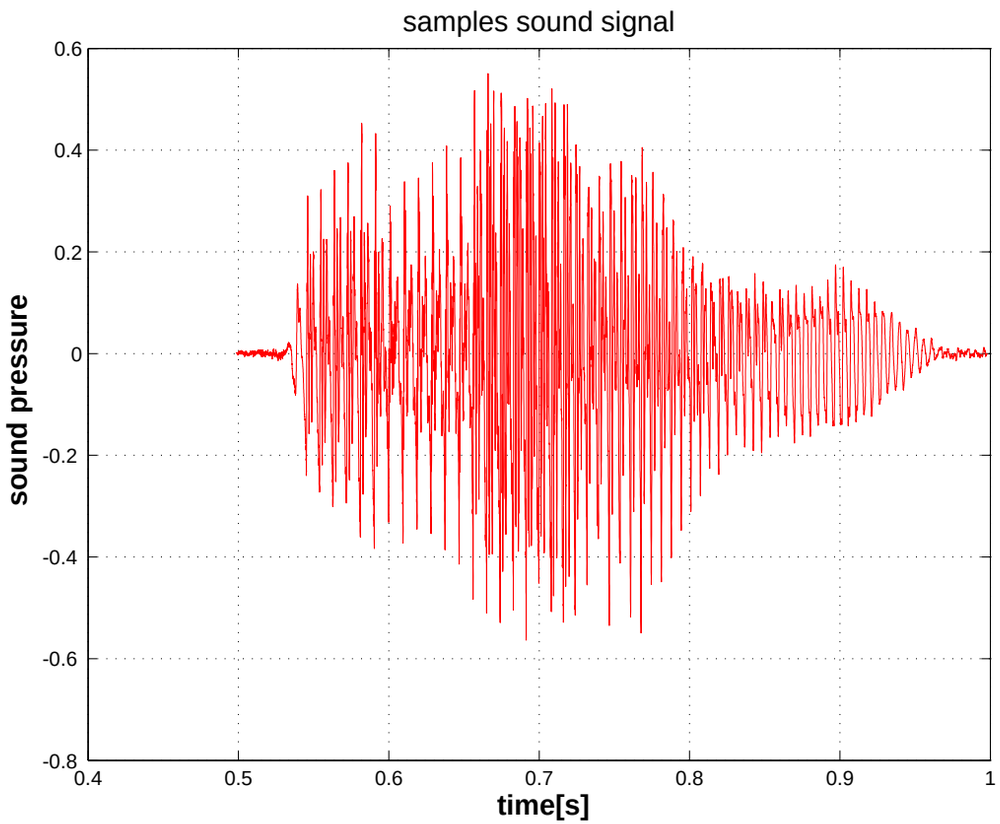
```
plot(1:n,abs(c).^2,'m-');
title('power spectrum of sound signal','fontsize',14);
xlabel('{\bf index k of Fourier coefficient}','fontsize',14);
ylabel('{\bf |c_k|^2}','fontsize',14);
grid on;

print -depsc2 '../PICTURES/soundpower.eps';

figure('name','sound frequencies');
plot(1:3000,abs(c(1:3000)).^2,'b-');
title('low frequency power spectrum','fontsize',14);
xlabel('{\bf index k of Fourier coefficient}','fontsize',14);
ylabel('{\bf |c_k|^2}','fontsize',14);
grid on;

print -depsc2 '../PICTURES/soundlowpower.eps';

for m=[1000,3000,5000]

  %Low pass filtering
  cf = zeros(1,n);
  cf(1:m) = c(1:m); cf(n-m+1:end) = c(n-m+1:end);

  %Reconstruct filtered signal
```

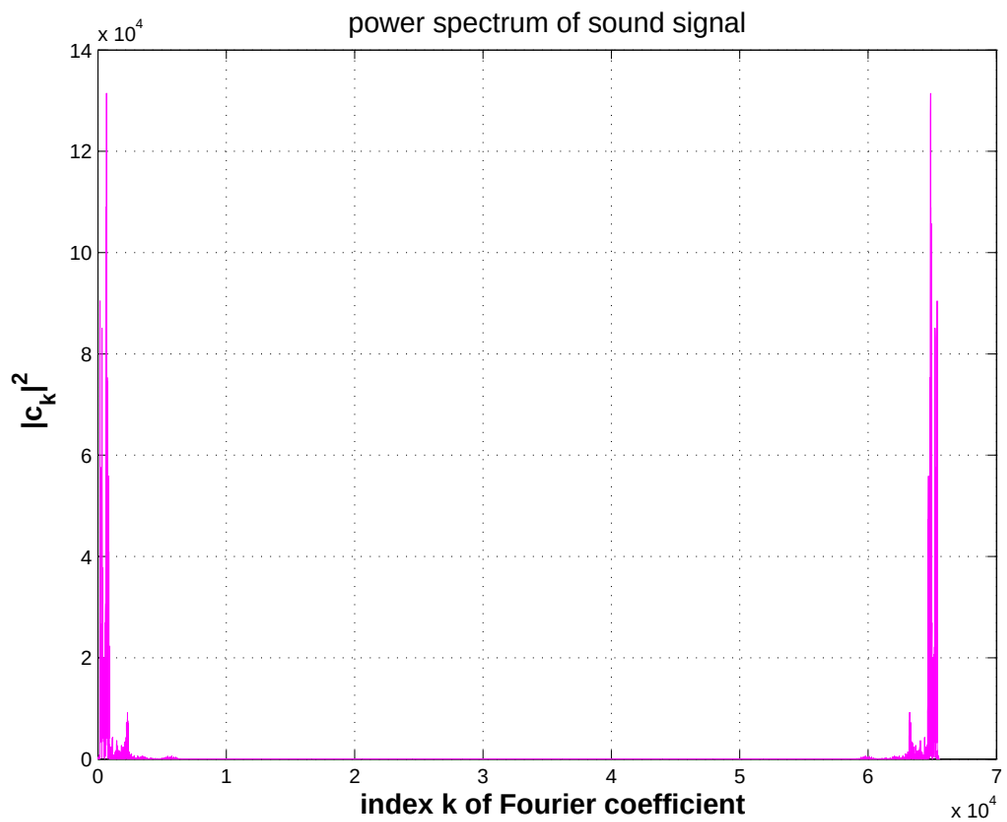
```
  yf = ifft(cf);
  wavwrite(yf,freq,nbits,sprintf('hellof%d.wav',m));

  k = k+1;
  s{k} = real(yf);
  leg{k} = sprintf('cutt-off = %d',m');
end

% Plot original signal and filtered signals
figure('name','sound filtering');
plot((30000:32000)/freq,s{1}(30000:32000),'r-',...
     (30000:32000)/freq,s{2}(30000:32000),'b--',...
     (30000:32000)/freq,s{3}(30000:32000),'m--',...
     (30000:32000)/freq,s{2}(30000:32000),'k--');
xlabel('{\bf time[s]}','fontsize',14);
ylabel('{\bf sound pressure}','fontsize',14);
legend(leg,'location','southeast');

print -depsc2 '../PICTURES/soundfiltered.eps';
```

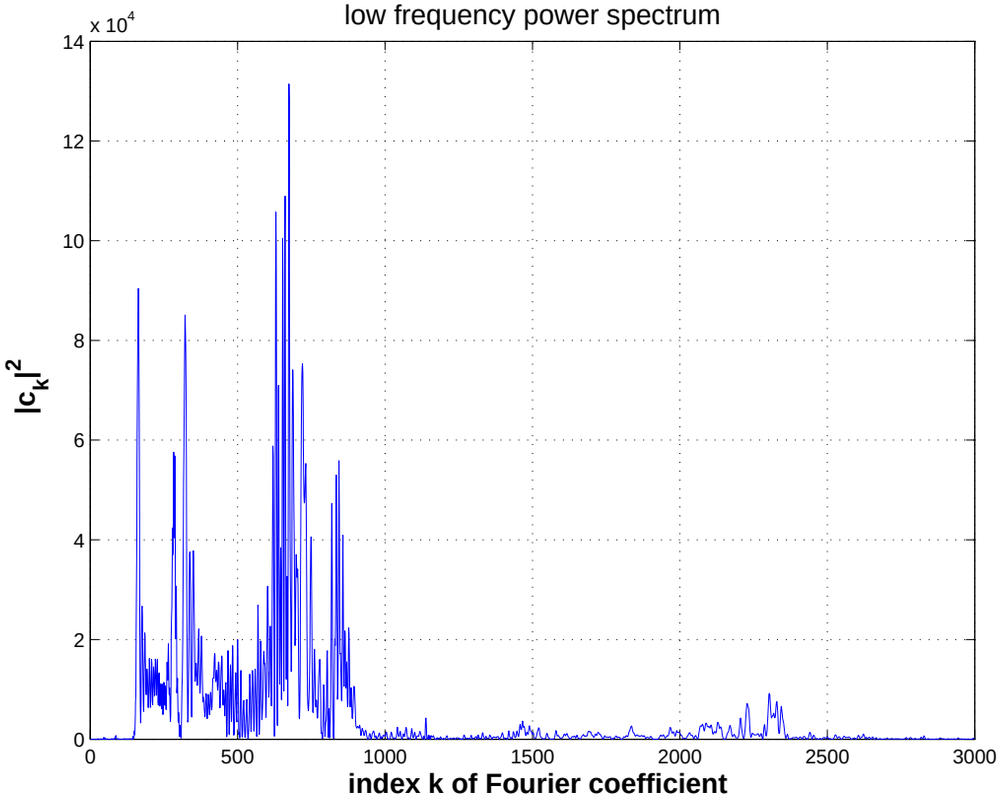
DFT based low pass frequency filtering of sound

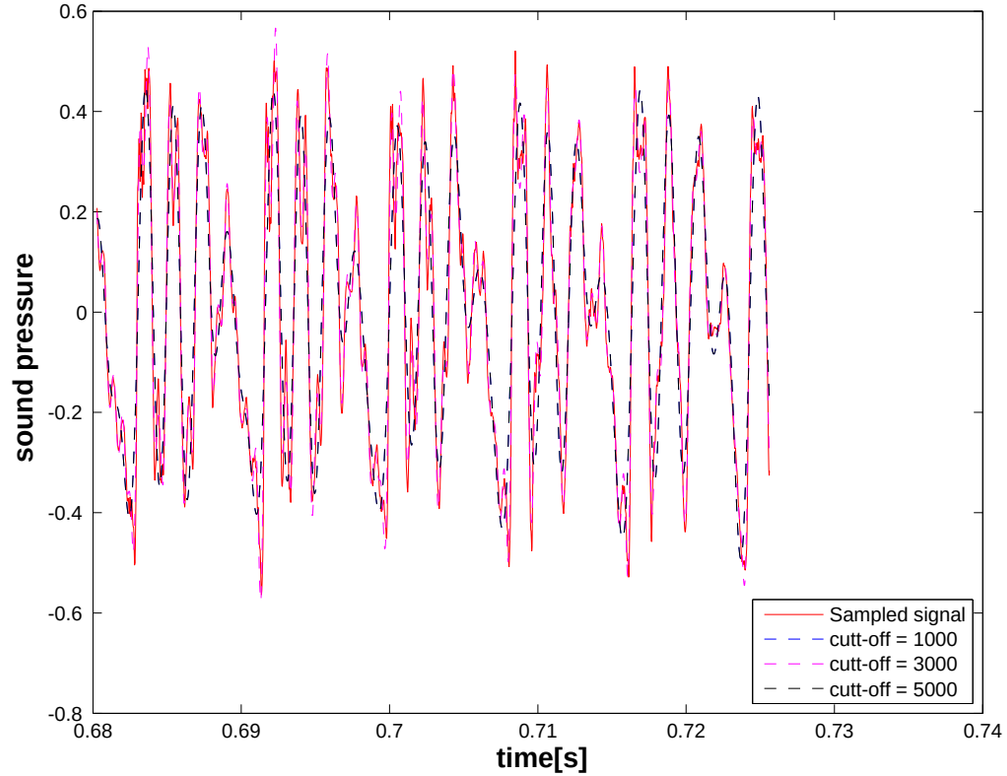
```
[y,sf,nb] = wavread('hello.wav');
c = fft(y); c(m+1:end-m) = 0;
```

```
wavwrite(ifft(c),sf,nb,'filtered.wav');
```






low frequency power spectrum
x 10^4
14
12
10
8
6
4
2
0
0
500
1000
1500
2000
2500
3000
|c_k|^2
index k of Fourier coefficient


0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0.68
0.69
0.7
0.71
0.72
0.73
0.74
sound pressure
time[s]
Sampled signal
cutt-off = 1000
cutt-off = 3000
cutt-off = 5000

◇

5.3 Trigonometric transformations

5.3.1 Discrete sine transform

Definition 5.3.1 (Discrete sine transformation matrix and discrete sine transform (DST)). *The matrix* $\mathbf{S}_n \in \mathbb{R}^{n,n}$ *defined by*

$$\mathbf{S}_n = \begin{pmatrix} \sin\left(\frac{\pi}{n+1}\right) & \sin\left(\frac{2\pi}{n+1}\right) & \cdots & \sin\left(\frac{n\pi}{n+1}\right) \\ \sin\left(\frac{2\pi}{n+1}\right) & \sin\left(\frac{4\pi}{n+1}\right) & \cdots & \sin\left(\frac{2n\pi}{n+1}\right) \\ \vdots & & & \vdots \\ \sin\left(\frac{n\pi}{n+1}\right) & \sin\left(\frac{2n\pi}{n+1}\right) & \cdots & \sin\left(\frac{n^2\pi}{n+1}\right) \end{pmatrix} = \left(\sin\left(\frac{kl\pi}{(n+1)}\right)\right)_{(k,l)\in\{1,\dots,n\}^2} \in \mathbb{R}^{n,n}$$

is called $(n \times n)$*-discrete sine transformation matrix. In addition, the linear map* $\mathbb{R}^n \ni \mathbf{y} \mapsto \mathbf{S}_n(y) = \mathbf{S}_n\,\mathbf{y} \in \mathbb{R}^n$ *is called discrete sine transform (DST).*

Note that if $\mathbf{y} = (y_0, \dots, y_{n-1}) \in \mathbb{R}^n$ and if $\mathbf{c} = (c_0, \dots, c_{n-1}) := \mathbf{S}_n\mathbf{y}$, then

$$\forall\, k \in \{0, 1, \dots, n-1\}: \qquad c_k = \sum_{l=0}^{n-1} y_l \, \sin\left(\frac{(k+1)(l+1)\pi}{(n+1)}\right) \quad . \tag{5.3.1}$$

Terminology: $\mathbf{c} = \mathbf{S}_n\mathbf{y}$ is also called the (discrete) sine transform of $\mathbf{y}$

MATLAB-functions for discrete sine transform (and its inverse):

DST: `c=dst(y)` $\leftrightarrow$ inverse DST: `y=idst(c)`

5.3.2 Discrete cosine transform

Definition 5.3.2 (Discrete cosine transformation matrix and discrete cosine transform (DCT)). *The matrix* $\mathbf{C}_n \in \mathbb{R}^{n,n}$ *defined by*

$$\mathbf{C}_n = \begin{pmatrix} \frac{1}{\sqrt{n}} & \frac{1}{\sqrt{n}} & \cdots & \frac{1}{\sqrt{n}} \\ \frac{\sqrt{2}}{\sqrt{n}}\cos\left(\frac{\frac{1}{2}\pi}{n}\right) & \frac{\sqrt{2}}{\sqrt{n}}\cos\left(\frac{\frac{3}{2}\pi}{n}\right) & \cdots & \frac{\sqrt{2}}{\sqrt{n}}\cos\left(\frac{(n-\frac{1}{2})\pi}{n}\right) \\ \frac{\sqrt{2}}{\sqrt{n}}\cos\left(\frac{\pi}{n}\right) & \frac{\sqrt{2}}{\sqrt{n}}\cos\left(\frac{3\pi}{n}\right) & \cdots & \frac{\sqrt{2}}{\sqrt{n}}\cos\left(\frac{2(n-\frac{1}{2})\pi}{n}\right) \\ \vdots & & & \vdots \\ \frac{\sqrt{2}}{\sqrt{n}}\cos\left(\frac{(n-1)\frac{1}{2}\pi}{n}\right) & \frac{\sqrt{2}}{\sqrt{n}}\cos\left(\frac{(n-1)\frac{3}{2}\pi}{n}\right) & \cdots & \frac{\sqrt{2}}{\sqrt{n}}\cos\left(\frac{(n-1)(n-\frac{1}{2})\pi}{n}\right) \end{pmatrix}$$

$$= \left(\frac{\sqrt{2}^{\min(k-1,1)}}{\sqrt{n}}\cos\left(\frac{(k-1)(l-\frac{1}{2})\pi}{n}\right)\right)_{(k,l)\in\{1,\dots,n\}^2} \in \mathbb{R}^{n,n}$$

is called $(n \times n)$*-discrete cosine transformation matrix. In addition, the linear map* $\mathbb{R}^n \ni \mathbf{y} \mapsto \mathbf{C}_n(y) = \mathbf{C}_n\,\mathbf{y} \in \mathbb{R}^n$ *is called discrete cosine transform (DCT).*

Note that if $\mathbf{y} = (y_0, \dots, y_{n-1}) \in \mathbb{R}^n$ and if $\mathbf{c} = (c_0, \dots, c_{n-1}) := \mathbf{C}_n\mathbf{y}$, then

$$\forall\, k \in \{0, 1, \dots, n-1\}: \qquad c_k = \begin{cases} \frac{1}{\sqrt{n}} \sum\limits_{l=0}^{n-1} y_l & : k = 0 \\ \frac{\sqrt{2}}{\sqrt{n}} \sum\limits_{l=0}^{n-1} y_l \, \cos\left(\frac{k(l+\frac{1}{2})\pi}{n}\right) & : k \geq 1 \end{cases} . \tag{5.3.2}$$

Terminology: $\mathbf{c} = \mathbf{C}_n\mathbf{y}$ is also called the (discrete) cosine transform of $\mathbf{y}$

MATLAB-functions for discrete cosine transform (and its inverse):

DCT: `c=dct(y)` $\leftrightarrow$ inverse DCT: `y=idct(c)`

5.4 Toeplitz matrix techniques

Definition 5.4.1 (Toeplitz matrices (diagonal-constant matrices)).
A matrix $\mathbf{T} = (t_{i,j})_{i\in\{1,\dots,m\},j\in\{1,\dots,n\}} \in \mathbb{C}^{m,n}$ *is called Toeplitz matrix if there is a vector* $\mathbf{u} = (u_{1-n},\dots,u_{m-1}) \in \mathbb{C}^{m+n-1}$ *such that* $t_{i,j} = u_{i-j}$ *for all* $i \in \{1,\dots,m\}$ *and all* $j \in \{1,\dots,n\}$*, i.e., if*

$$\mathbf{T} = \begin{pmatrix} u_0 & u_{-1} & \cdots & & \cdots & u_{1-n} \\ u_1 & u_0 & u_{-1} & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & \ddots & u_{-1} \\ u_{m-1} & \cdots & & \cdots & u_1 & u_0 \end{pmatrix} .$$

In that case $\mathbf{u} = (u_{1-n},\dots,u_{m-1}) \in \mathbb{C}^{m+n-1}$ *is called generating vector of* $\mathbf{T}$*.*

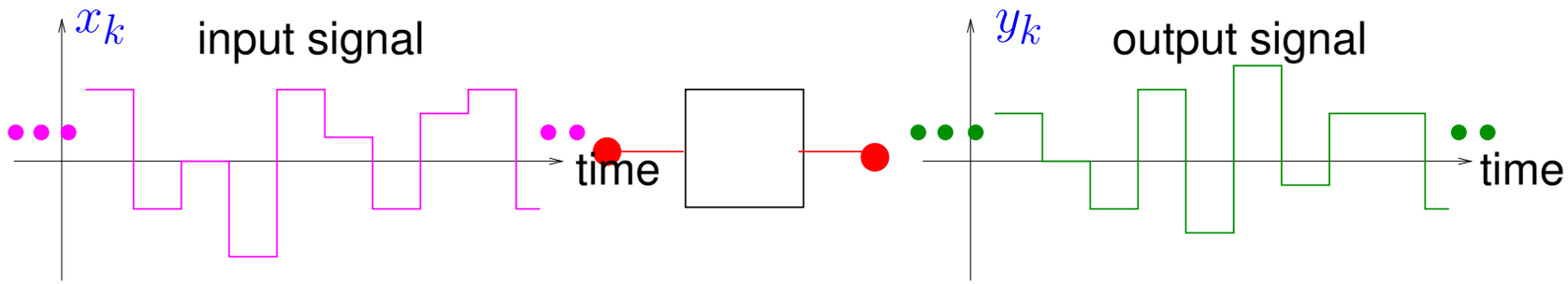
<u>Remark:</u> MATLAB command `toeplitz(...)` returns Toepliz matrix.
See `help toeplitz` for details.

Toeplitz matrices appear, e.g., in discrete convolutions; cf. (5.1.1).

Example 5.4.1 (Parameter identification for linear time-invariant filters in the periodic setting).

- $(x_k)_{k\in\mathbb{Z}}$ m-periodic discrete signal = *known* input
- $(y_k)_{k\in\mathbb{Z}}$ m-periodic *known* output signal of a linear time-invariant filter in the periodic setting
- • Known: impulse response of filter has maximal duration $n\Delta t$, $n \in \mathbb{N}$, $n \leq m$

$$\blacktriangleright \quad \exists\, \mathbf{h} = (h_0, \dots, h_{n-1}) \in \mathbb{C}^n, \quad n \leq m : \quad y_k = \sum_{j=0}^{n-1} h_j\, x_{k-j} \,. \tag{5.4.1}$$



Parameter identification problem: seek $\mathbf{h} = (h_0, \dots, h_{n-1}) \in \mathbb{C}^n$ such that

$$\|\mathbf{A}\mathbf{h} - \mathbf{y}\|_2 = \left\| \begin{pmatrix} x_0 & x_{-1} & \cdots & & \cdots & x_{1-n} \\ x_1 & x_0 & x_{-1} & & & \vdots \\ \vdots & x_1 & x_0 & \ddots & & \\ & & \ddots & \ddots & & \vdots \\ \vdots & & & \ddots & \ddots & x_{-1} \\ x_{n-1} & & & & x_1 & x_0 \\ x_n & x_{n-1} & & & & x_1 \\ \vdots & & & & & \vdots \\ x_{m-1} & \cdots & & & \cdots & x_{m-n} \end{pmatrix} \begin{pmatrix} h_0 \\ \vdots \\ \\ \\ \vdots \\ h_{n-1} \end{pmatrix} - \begin{pmatrix} y_0 \\ \vdots \\ \\ \\ \vdots \\ y_{m-1} \end{pmatrix} \right\|_2 \to \min \,.$$

➢ Linear least squares problem $\rightarrow$ Ch. 3 with Toeplitz matrix $\mathbf{A} = (x_{i-j})_{(i,j)\in\{1,\dots,m\}\times\{1,\dots,n\}} \in \mathbb{C}^{m,n}$.

$\diamond$

Definition 5.4.2 (Reversing permutation)**.** *By* $\mathbf{P}_n \in \mathbb{R}^{n,n}$ *we denote the matrix*

$$\mathbf{P}_n := \begin{pmatrix} 0 & 0 & \dots & 0 & 1 \\ 0 & 0 & \dots & 1 & 0 \\ \vdots & & \vdots & & \vdots \\ 0 & 1 & \dots & 0 & 0 \\ 1 & 0 & \dots & 0 & 0 \end{pmatrix} . \tag{5.4.2}$$

Lemma 5.4.3 (Properties of $\mathbf{P}_n$ and symmetric Toeplitz matrices)**.** *It holds that*

$$(\mathbf{P}_n)^2 = \mathbf{I} \quad , \quad (\mathbf{P}_n)^T = \mathbf{P}_n \quad , \quad \mathbf{P}_n \mathbf{e}_k^{(n)} = \mathbf{e}_{n-k+1}^{(n)} \quad , \quad \mathbf{P}_n \mathbf{T} \mathbf{P}_n = \mathbf{T}$$

for all $k \in \{1, \dots, n\}$ *and all symmetric Toeplitz matrices* $\mathbf{T} \in \mathbb{C}^{n,n}$.

5.4.1 Toeplitz matrix arithmetic

Consider Toeplitz matrix $\mathbf{T} = (u_{i-j})_{i\in\{1,\dots,m\},j\in\{1,\dots,n\}} \in \mathbb{C}^{m,n}$
with generating vector $\mathbf{u} = (u_{-m+1},\dots,u_{n-1}) \in \mathbb{C}^{m+n-1}$.

<u>Task:</u> efficient evaluation of matrix $\times$ vector product $\mathbf{Tx}$ for $\mathbf{x} \in \mathbb{C}^n$.

<u>Idea:</u> In the case $m = n$ the following extended matrix is circulant ($\rightarrow$ Def. 5.1.4):

$$\mathbf{C} = \begin{pmatrix} \mathbf{T} & \mathbf{S} \\ \mathbf{S} & \mathbf{T} \end{pmatrix} = \left(\begin{array}{cccccc|cccccc}
u_0 & u_{-1} & \cdots & & \cdots & u_{1-n} & 0 & u_{n-1} & \cdots & & \cdots & u_1 \\
u_1 & u_0 & u_{-1} & & & \vdots & u_{1-n} & 0 & \ddots & & & \vdots \\
\vdots & \ddots & \ddots & \ddots & & \vdots & \vdots & \ddots & \ddots & & & \\
\vdots & & \ddots & \ddots & \ddots & \vdots & & & & & \ddots & \\
\vdots & & & \ddots & \ddots & u_{-1} & \vdots & & & \ddots & \ddots & u_{n-1} \\
u_{n-1} & \cdots & & \cdots & u_1 & u_0 & u_{-1} & & & & u_{1-n} & 0 \\
\hline
0 & u_{n-1} & \cdots & & \cdots & u_1 & u_0 & u_{-1} & \cdots & & \cdots & u_{1-n} \\
u_{1-n} & 0 & \ddots & & & \vdots & u_1 & u_0 & u_{-1} & & & \vdots \\
\vdots & \ddots & \ddots & & & & \vdots & \ddots & \ddots & \ddots & & \vdots \\
 & & & & \ddots & & \vdots & & \ddots & \ddots & \ddots & \vdots \\
\vdots & & & \ddots & \ddots & u_{n-1} & \vdots & & & \ddots & \ddots & u_{-1} \\
u_{-1} & & & & u_{1-n} & 0 & u_{n-1} & \cdots & & \cdots & u_1 & u_0
\end{array}\right).$$

Note that there are many possibilities how $\mathbf{T}$ can be extended to a circulant matrix.

In general, $\mathbf{T}$ can, e.g., be extended through

```
T = toeplitz(u(0:m-1),u(0:-1:1-n));
S = toeplitz([0,u(1-n:-1)],[0,u(m-1:-1:1)]);
```

$$\mathbf{C}\begin{pmatrix}\mathbf{x}\\0\end{pmatrix} = \begin{pmatrix}\mathbf{Tx}\\\mathbf{Sx}\end{pmatrix}$$

Computational effort $O\big((n+m)\log(n+m)\big)$ for computing $\mathbf{Tx}$ (FFT based, Sect. 5.2.2).

5.4.2 The Levinson algorithm

Given:

- Symmetric positive definite Toeplitz matrix $\mathbf{T} = (u_{i-j})_{i,j\in\{1,\dots,n\}} \in \mathbb{C}^{n,n}$ with generating vector $\mathbf{u} = (u_{-n+1},\dots,u_{n-1}) \in \mathbb{C}^{2n-1}$
  (Symmetry $\Leftrightarrow \forall\, k \in \{1,2,\dots,n-1\}\colon u_{-k} = u_k$).
  In addition, we assume w.l.o.g. that $u_0 = 1$.
- Vector $\mathbf{b} = (b_1,\dots,b_n) \in \mathbb{C}^n$.

Task: Efficient solution algorithm for LSE $\mathbf{T}\mathbf{x} = \mathbf{b}$ (Yule-Walker problem).

Recursive (inductive) solution strategy:

For every $k \in \{1, \dots, n\}$ define

- $\mathbf{T}_k := (u_{i-j})_{i,j \in \{1,\dots,k\}} \in \mathbb{C}^{k,k}$ (left upper block of $\mathbf{T}$) ➢ $\boxed{\mathbf{T}_k \text{ is s.p.d. Toeplitz matrix}}$, and
- $\mathbf{b}^k := (b_1, \dots, b_k)$,
- $\mathbf{x}^k = (x_1^k, \dots, x_k^k) \in \mathbb{C}^k$: $\mathbf{T}_k \mathbf{x}^k = \mathbf{b}^k \quad (\Leftrightarrow \quad \mathbf{x}^k = \mathbf{T}_k^{-1} \mathbf{b}^k)$,
- reversing permutation $P_k \in \mathbb{R}^{k,k}$ through $P_k \mathbf{e}_j^{(k)} := \mathbf{e}_{k-j+1}^{(k)}$ for all $j \in \{1, \dots, k\}$

for every $k \in \{1, \dots, n-1\}$ define $\mathbf{u}^k := (u_1, \dots, u_k)$.

Let $k \in \{1, \dots, n-1\}$ and consider Block-partitioned LSE

$$\mathbf{T}_{k+1} \mathbf{x}^{k+1} = \left(\begin{array}{ccc|c} & \mathbf{T}_k & & \begin{matrix} u_k \\ \vdots \\ u_1 \end{matrix} \\ \hline u_k & \cdots & u_1 & 1 \end{array} \right) \left(\begin{array}{c} \widetilde{\mathbf{x}}^{k+1} \\ \hline x_{k+1}^{k+1} \end{array} \right) = \left(\begin{array}{c} b_1 \\ \vdots \\ b_k \\ \hline b_{k+1} \end{array} \right) = \left(\begin{array}{c} \mathbf{b}^k \\ b_{k+1} \end{array} \right) \tag{5.4.3}$$

$$\widetilde{\mathbf{x}}^{k+1} = \mathbf{T}_k^{-1}\left(\mathbf{b}^k - x_{k+1}^{k+1}\mathbf{P}_k\mathbf{u}^k\right) = \mathbf{x}^k - x_{k+1}^{k+1}\mathbf{T}_k^{-1}\mathbf{P}_k\mathbf{u}^k = \mathbf{x}^k - x_{k+1}^{k+1}\mathbf{P}_k\underbrace{\mathbf{T}_k^{-1}\mathbf{u}^k}_{=:\mathbf{y}^k} ,$$

$$x_{k+1}^{k+1} = b_{k+1} - \left(\mathbf{P}_k\mathbf{u}^k\right)^T\widetilde{\mathbf{x}}^{k+1} = b_{k+1} - \left(\mathbf{P}_k\mathbf{u}^k\right)^T\mathbf{x}^k + x_{k+1}^{k+1}\left(\mathbf{u}^k\right)^T\left(\mathbf{P}_k\right)^T\mathbf{P}_k\mathbf{y}^k \tag{5.4.4}$$

and hence

$$\mathbf{x}^{k+1} = \begin{pmatrix}\widetilde{\mathbf{x}}^{k+1}\\ x_{k+1}^{k+1}\end{pmatrix} \quad \text{with} \quad \begin{aligned} x_{k+1}^{k+1} &= \frac{1}{\sigma_k}\left(b_{k+1} - (\mathbf{P}_k\mathbf{u}^k)^T\mathbf{x}^k\right)\\ \widetilde{\mathbf{x}}^{k+1} &= \mathbf{x}^k - x_{k+1}^{k+1}\,\mathbf{P}_k\,\mathbf{y}^k \end{aligned} \quad , \quad \sigma_k := 1 - (\mathbf{u}^k)^T\mathbf{y}^k . \tag{5.4.5}$$

Levinson algorithm ▷

(recursive, u_n not used in the calculation of $\mathbf{x}$!)

➢ Asymptotic complexity $O(n^2)$

MATLAB-CODE Levinson algorithm

```
function [x,y] = levinson(u,b)
k = length(u)-1;
if (k == 0)
  x=b(1); y = u(1); return;
end
[xk,yk] = levinson(u(1:k),b(1:k));
sigma = 1-dot(u(1:k),yk);
t= (b(k+1)-dot(u(k:-1:1),xk))/sigma;
x= [xk-t*yk(k:-1:1); t];
s= (u(k+1)-dot(u(k:-1:1),yk))/sigma;
y= [yk-s*yk(k:-1:1); s];
```

Remark 5.4.2 (Fast Toeplitz solvers).

FFT-based algorithms for solving $\mathbf{Tx} = \mathbf{b}$ with asymptotic complexity $O(n \log^3 n)$ [49] **!**

△

Further reading for the material in this chapter:

[10, Sect. 8.5]: Very detailed and elementary presentation. Hardly addresses discrete convolution.

[29, Ch. IX] presents the topic from a mathematical point of few stressing approximation and trigonometric interpolation. Good reference for algorithms for circulant and Toeplitz matrices.

[47, Ch. 10] also discusses the discrete Fourier transform with emphasis on interpolation and (least squares) approximation. The presentation of signal processing differs from that of the course.

There is a vast number of books and survey papers dedicated to discrete Fourier transforms, see, for instance, [15, 6]. Issues and technical details way beyond the scope of the course are treated there.