Numerical Methods for Computational Science and Engineering

Fall Semester 2017 (HS17)

Prof. Rima Alaifari, SAM, ETH Zurich

FFT continued:

Divide-and-conquer approach

Recall $c_k := (F_n y)_k = \sum_{j=0}^{n-1} y_j \, \omega_n^{kj} = \sum_{j=0}^{n-1} y_j \, e^{-2\pi i kj/n}$

note: $c_{k+n\ell} = c_k \qquad \ell \in \mathbb{Z}$

Example: $n = 2^\alpha \qquad \alpha \in \mathbb{N}$

$n = 2m$

Split the DFT:

$$c_k = \sum_{j=0}^{m-1} y_{2j} \, \omega_n^{2kj} + \sum_{j=0}^{m-1} y_{2j+1} \, \underline{\omega_n^{k(2j+1)}}$$

$$= \sum_{j=0}^{m-1} y_{2j} \, \omega_n^{2kj} + \omega_n^{k} \sum_{j=0}^{m-1} y_{2j+1} \, \omega_n^{2kj}$$

$$= \sum_{j=0}^{m-1} \underbrace{y_{2j}}_{} \underbrace{e^{-2\pi i (kj)/m}}_{\omega_m^{kj}} + \omega_n^{k} \sum_{j=0}^{m-1} y_{2j+1} \underbrace{e^{-2\pi i kj/m}}_{\omega_m^{kj}}$$

$$= \sum_{j=0}^{m-1} y_j^1 \, \omega_m^{kj} + \omega_n^{k} \sum_{j=0}^{m-1} y_j^2 \, \omega_m^{kj}$$

$$\left.\begin{aligned} y^1 &= [y_0, y_2, \dots, y_{n-2}]^T \\ y^2 &= [y_1, y_3, \dots, y_{n-1}]^T \end{aligned}\right\} \text{signals of length } m = \frac{n}{2}$$

$= (c^1)_k + \omega_n^k (c^2)_k$

$\uparrow$

c^1 m-DFT of y^1, c^2 m-DFT of y^2

Note: $(c^1)_{k+m\ell} = (c^1)_k$

$(c^2)_{k+m\ell} = (c^2)_k$

$(F_n y)_k = c_k = (c^1)_k + \underbrace{\omega_n^k (c^2)_k}$ $\quad k = 0, \dots, m-1$

$c_{k+m} = (c^1)_k + \underbrace{\omega_n^{m+k}}_{= -\omega_n^k} (c^2)_k$

Complexity: $m = \frac{n}{2}$ multiplications, $2m = n$ additions

$\Rightarrow$ $\frac{3n}{2}$ number of basic operations

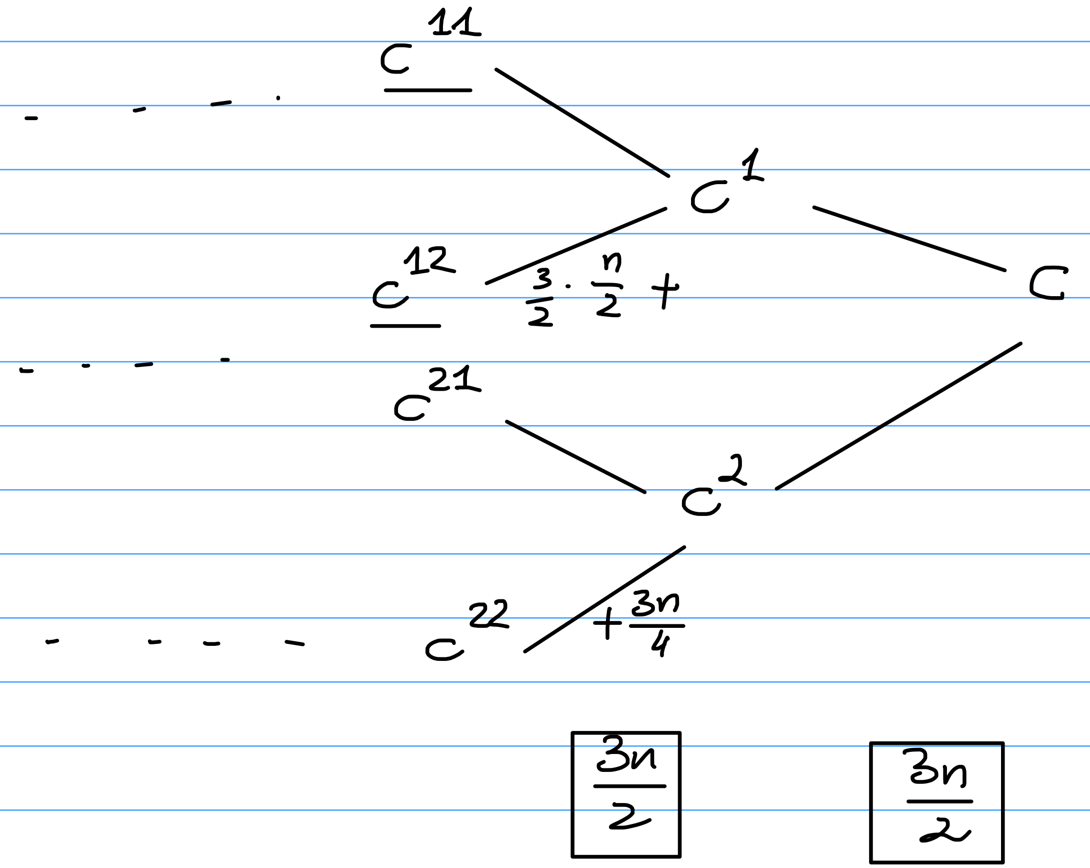
to combine $(c^1), (c^2)$ to c

Proceed: break down y^1, y^2 into shorter signals

How many steps possible?

$(n = 2^\alpha)$ $\quad \log_2 n$ steps

1-point DFTs



At j-th step: $\left(\frac{3}{2} \cdot \frac{n}{2^j}\right) \cdot 2^j = \frac{3n}{2}$ operations

Overall: $\frac{3}{2} \cdot n \cdot \log_2 n = \Theta(n \log_2 n)$ complexity for FFT

4.2.2. Frequency filtering via DFT

Given a signal $\underline{x} = [x_0, \dots, x_{n-1}]^T$

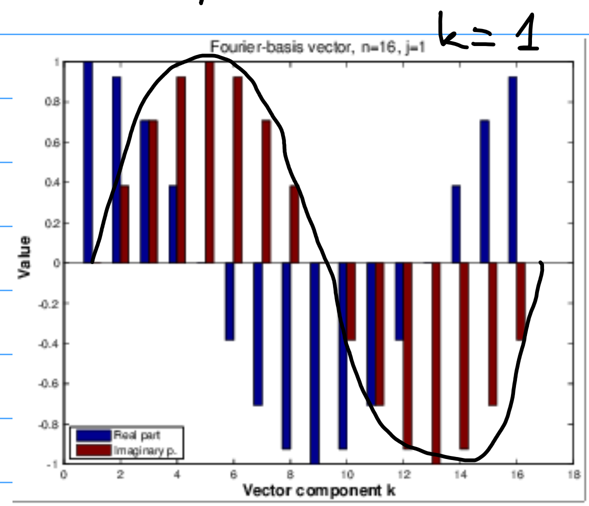
What is the information on $\underline{x}$ carried in $F_n \underline{x}$?

k-th row of F_n equal to column $\underline{v}_k$

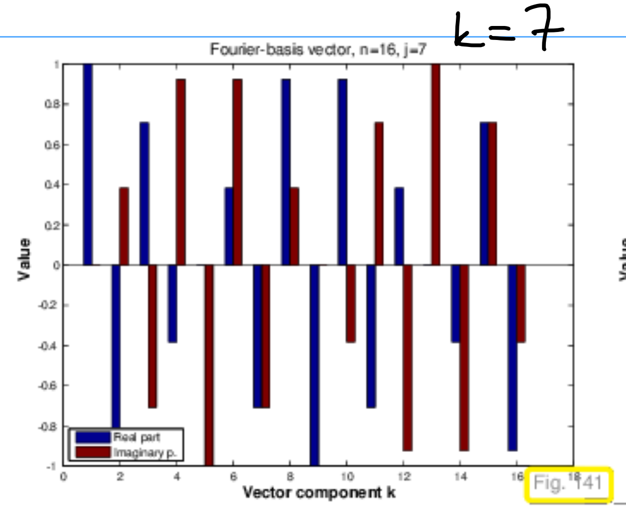
$$\Rightarrow \quad (F_n \underline{x})_k = \underline{v}_k^T \underline{x}$$

trigonometric basis $\{v_0, \dots, v_{n-1}\}$ harmonic oscillations

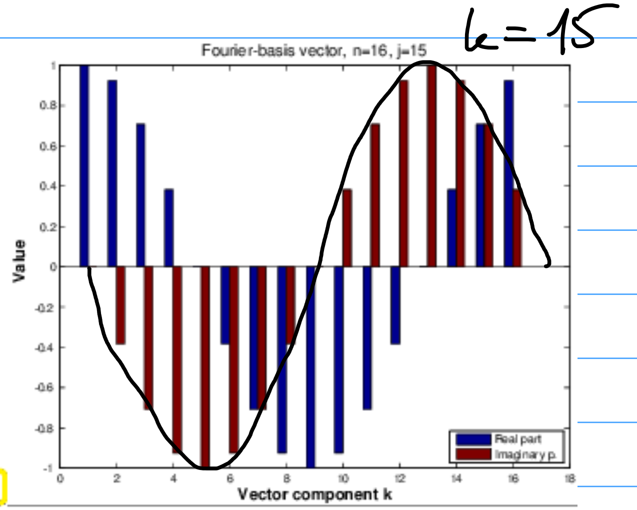
Example $n=16$:



"slow oscillation/low frequency"



"fast oscillation/high frequency"



"slow oscillation/low frequency"

blue: $\mathrm{Re}(\underline{v}_k)$

red: $\mathrm{Im}(\underline{v}_k)$

$$c_k = (F_n \underline{x})_k = \sum_{j=0}^{n-1} x_j \, \omega_n^{kj}$$

Inverse DFT: $x_j = \frac{1}{n} \sum_{k=0}^{n-1} c_k \, \underline{\omega_n^{-kj}}$

for $n = 2m+1$:

$$n x_j = \sum_{k=0}^{m} c_k \, \omega_n^{-kj} + \sum_{k=m+1}^{2m} c_k \, \omega_n^{-kj}$$

$$= c_0 + \sum_{k=1}^{m} c_k \omega_n^{-kj} + \sum_{k=1}^{m} c_{n-k}\, \omega_n^{-(n-k)j}$$

$$= c_0 + \sum_{k=1}^{m} \left(c_k \omega_n^{-kj} + c_{n-k}\, \omega_n^{-(n-k)j} \right)$$

Note: $c_{n-k} = \sum_{j=0}^{n-1} x_j \omega_n^{(n-k)j} = \sum_{j=0}^{n-1} x_j \underbrace{\omega_n^{-kj}}_{\overline{\omega_n}^{kj}} = \overline{c_k}$

and $\omega_n^{-(n-k)j} = \omega_n^{kj} = \overline{\omega_n}^{-kj}$

$$\Rightarrow n x_j = c_0 + \sum_{k=1}^{m} \left(c_k \omega_n^{-kj} + \overline{c_k}\, \overline{\omega_n}^{-kj} \right)$$

$$= c_0 + 2 \sum_{k=1}^{m} \mathrm{Re}\left(c_k \omega_n^{-kj} \right).$$

$$= c_0 + 2 \sum_{k=1}^{m} \Big[\mathrm{Re}(c_k) \cos(2\pi kj/n)$$

$-$ 2.

$$+ \mathrm{Im}(c_k) \sin(2\pi kj/n) \Big]$$

$|c_k|$, $|c_{n-k}|$ measures how much oscillation with frequency k is present in signal $\underline{x}$.

$k \in \{0, \dots, \lfloor \frac{n}{2} \rfloor\}$

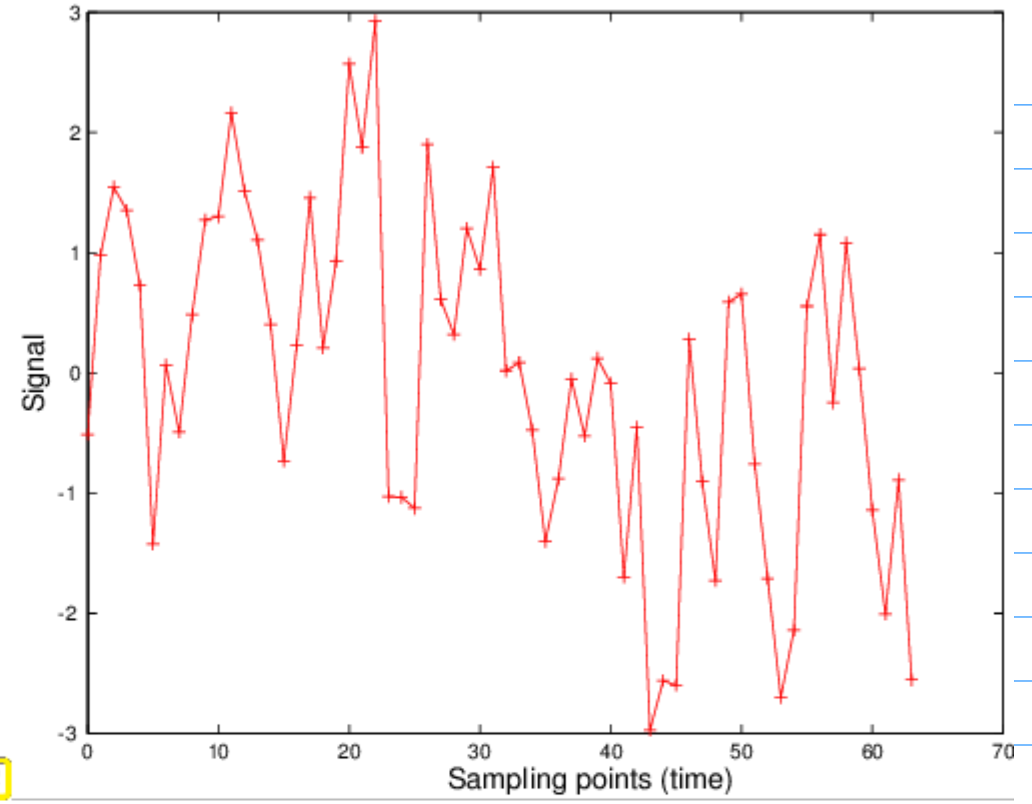


Fig. 142

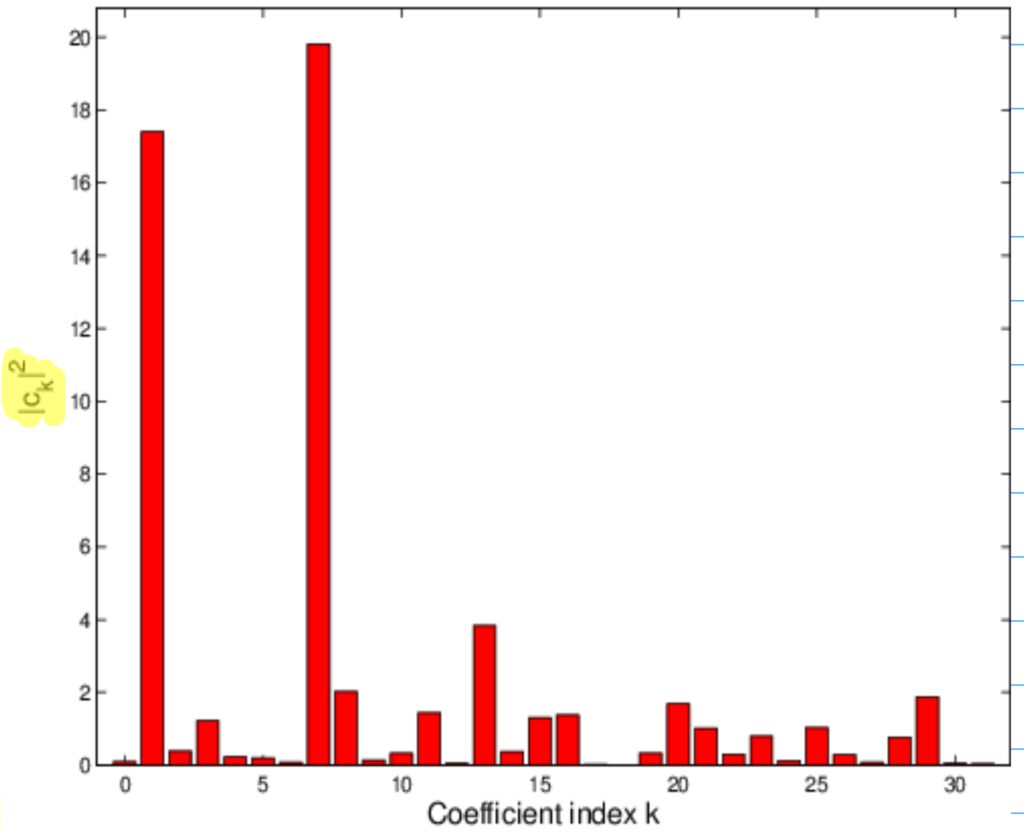


Fig. 143

magnitude squared of the signal's DFT
locates which frequencies are present in signal & how much they're present

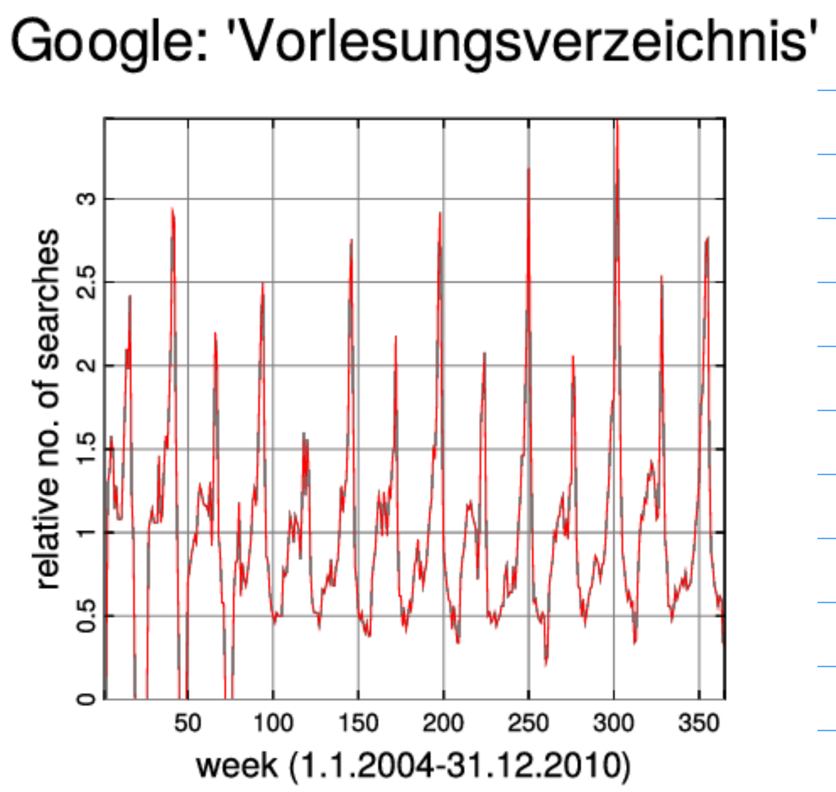


Fig. 144

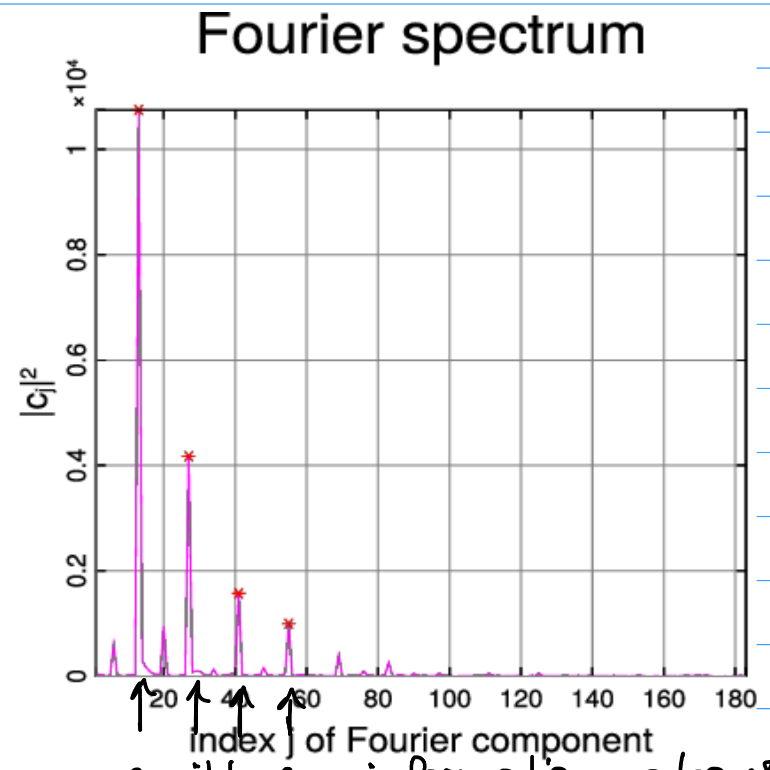


Fig. 145

pronounced peaks: structure of data is periodic

positions: information about length of periods

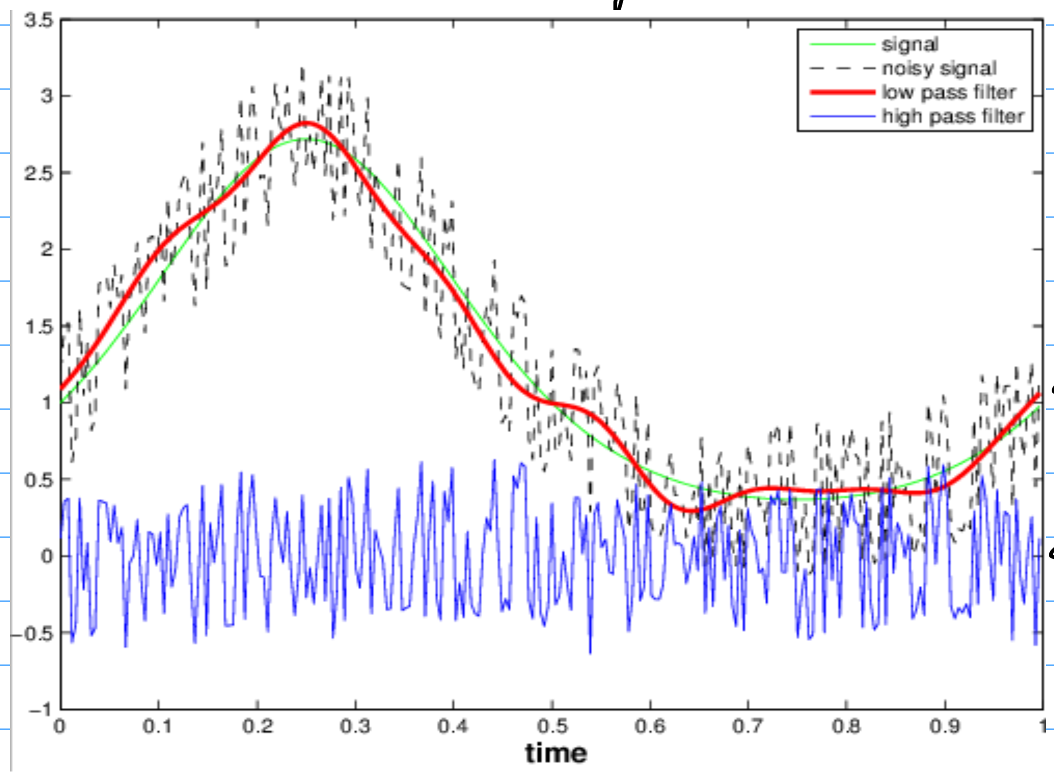
Low vs. high frequencies

typical model for noise: high-frequency

idea for denoising a signal (audio / image)

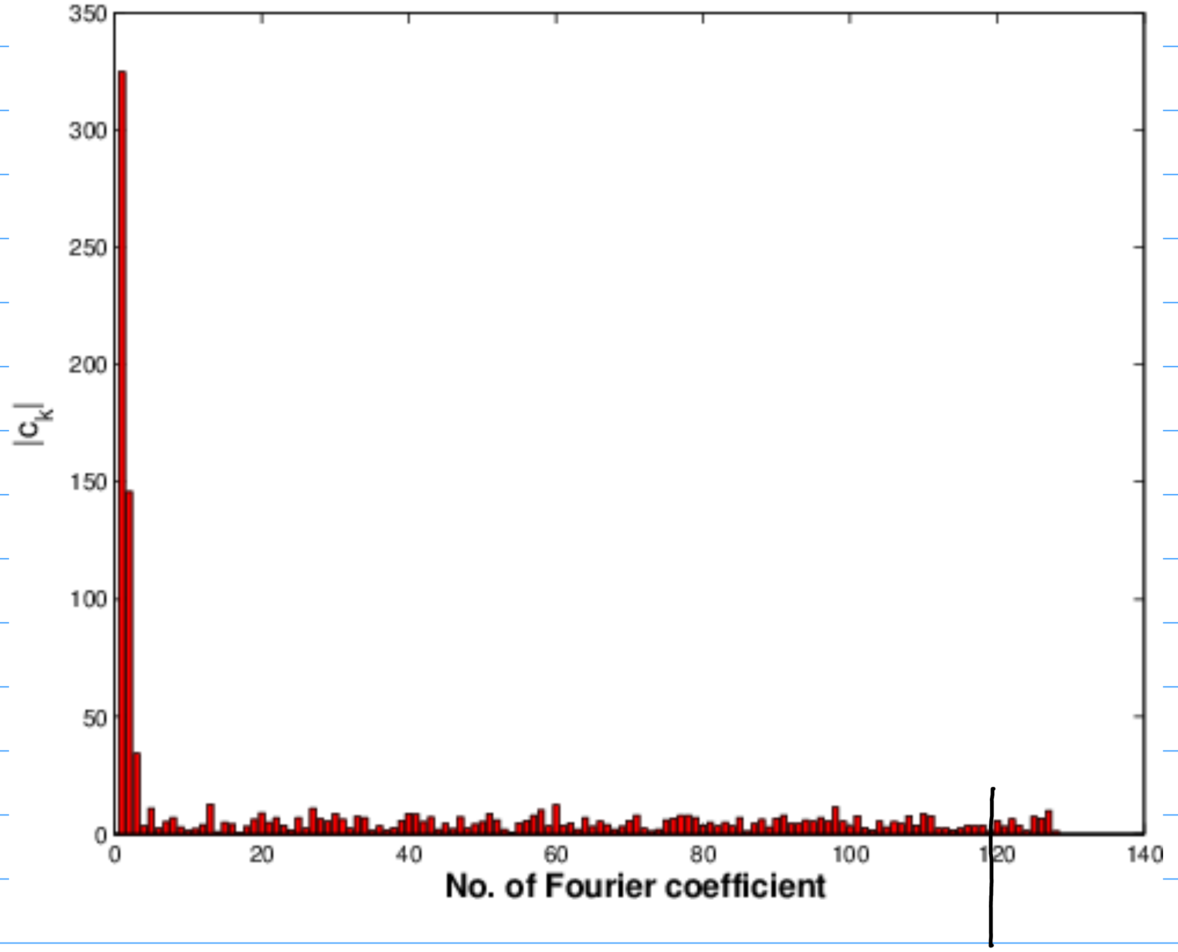
- transform to frequency domain
- "low-pass filter" (i.e. cut-off high frequencies)
- transform back to time/space domained to obtained a denoised signal.

Example:



← throwing away high freq.

← keeping only high freq.



cut-off at frequency $k=120$

4.2.4. 2D DFT

Given a matrix $Y \in \mathbb{C}^{m,n}$, its 2D DFT is defined as 2 nested 1D DFTs:

$$\underset{\text{2D DFT}(Y)}{(\mathbf{C})_{k_1,k_2}} = \sum_{j_1=0}^{m-1}\sum_{j_2=0}^{n-1} y_{j_1,j_2}\,\omega_m^{j_1k_1}\omega_n^{j_2k_2} = \underbrace{\sum_{j_1=0}^{m-1}\omega_m^{j_1k_1}\underbrace{\left(\sum_{j_2=0}^{n-1}\omega_n^{j_2k_2}y_{j_1,j_2}\right)}}\,, \quad 0 \le k_1 < m\,, 0 \le k_2 < n\,.$$

$$(\mathbf{C})_{k_1,k_2} = \sum_{j_1=0}^{m-1}(\mathbf{F}_n(\mathbf{Y})_{j_1,:})_{k_2}\,\omega_m^{j_1k_1} \quad \blacktriangleright \quad \boxed{\overset{\text{2D DFT of } Y}{\mathbf{C}} = \mathbf{F}_m(\mathbf{F}_n\mathbf{Y}^\top)^\top = \mathbf{F}_m\mathbf{Y}\mathbf{F}_n}\,. \tag{4.2.46}$$

[Recall: 1D: $c = F_n y$ $\quad y \in \mathbb{R}^n$]

and 2D inverse DFTs

$$\mathbf{C} = \sum_{j_1=0}^{m-1}\sum_{j_2=0}^{n-1} y_{j_1,j_2}(\mathbf{F}_m)_{:,j_1}(\mathbf{F}_n)^\top_{:,j_2} \quad \Rightarrow \quad \boxed{\mathbf{Y} = \mathbf{F}_m^{-1}\mathbf{C}\mathbf{F}_n^{-1} = \tfrac{1}{mn}\overline{\mathbf{F}}_m\mathbf{C}\overline{\mathbf{F}}_n}\,. \tag{4.2.47}$$

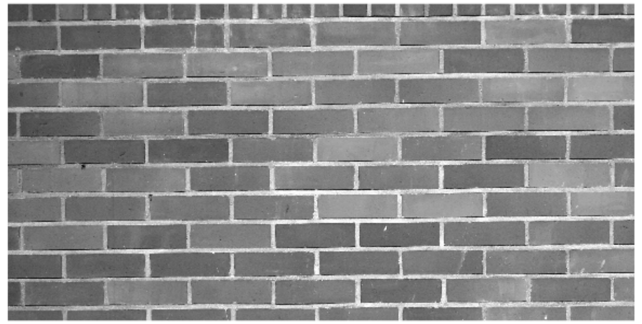
$$F_m^{-1} = \frac{1}{m}\overline{F_m}$$
$$F_n^{-1} = \frac{1}{n}\overline{F_n}$$

C++11 code 4.2.48: Two-dimensional discrete Fourier transform ➺ GITLAB

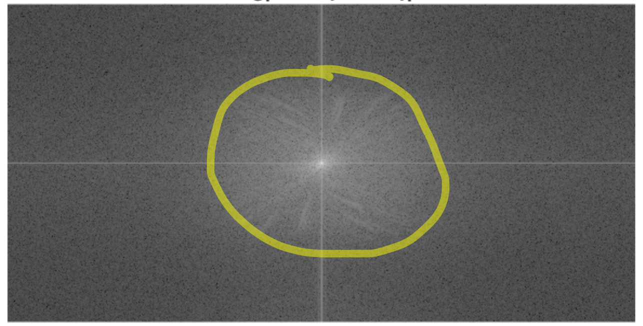
```cpp
template <typename Scalar>
void fft2(Eigen::MatrixXcd &C, const Eigen::MatrixBase<Scalar> &Y) {
  using idx_t = Eigen::MatrixXcd::Index;
  const idx_t m = Y.rows(),n=Y.cols();
  C.resize(m,n);
  Eigen::MatrixXcd tmp(m,n);

  Eigen::FFT<double> fft; // Helper class for DFT
  // Transform rows of matrix Y
  for (idx_t k=0;k<m;k++) {
    Eigen::VectorXcd tv(Y.row(k));
    tmp.row(k) = fft.fwd(tv).transpose();
  }

  // Transform columns of temporary matrix
  for (idx_t k=0;k<n;k++) {
    Eigen::VectorXcd tv(tmp.col(k));
    C.col(k) = fft.fwd(tv);
  }
}
```

C++11 code 4.2.49: Inverse two-dimensional discrete Fourier transform ➺ GITLAB

```cpp
template <typename Scalar>
void ifft2(Eigen::MatrixXcd &C, const Eigen::MatrixBase<Scalar> &Y) {
  using idx_t = Eigen::MatrixXcd::Index;
  const idx_t m = Y.rows(),n=Y.cols();
  fft2(C,Y.conjugate()); C = C.conjugate()/(m*n);
}
```

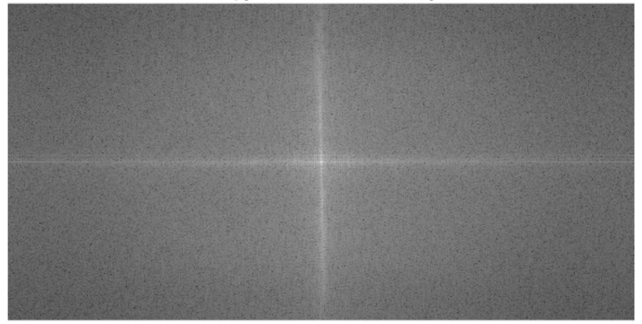


log|FFT2(rabbit)|



more spread out

log|FFT2(brick wall)|



Filtering with 2D DFT:

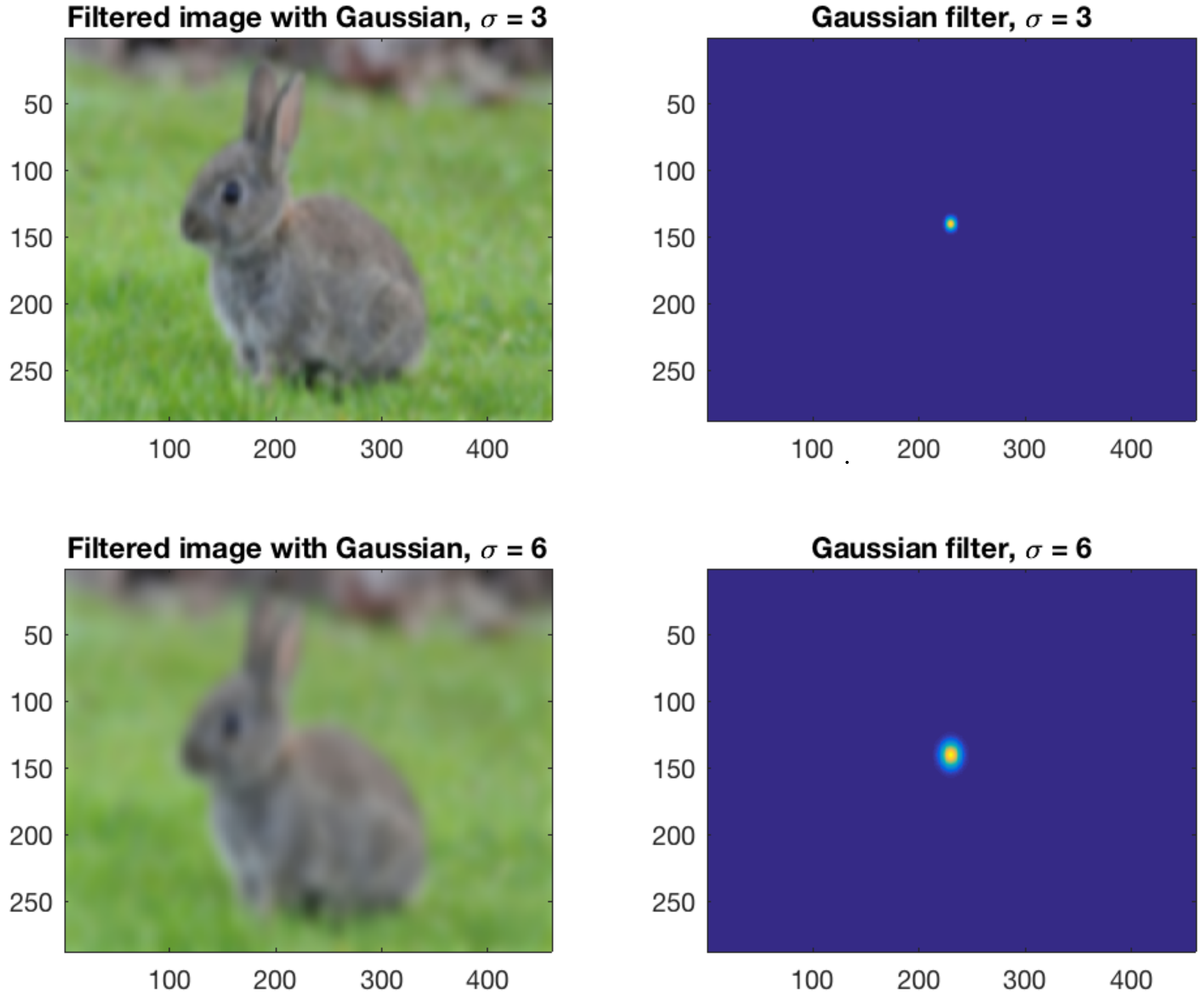
As in 1D: describe filtering via 2D discrete convolution & with zero-padding

2D discrete conv. is reducable to

2D discrete periodic (circular) conv.

Example: Smoothing with a Gaussian filter

Original Image



2D convolution theorem

Let $U, X \in \mathbb{C}^{m,n}$ and let the 2D discrete per. convolution $U *_{m,n} X$ be defined by

$$(U *_{m,n} X)_{k,l} := \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (U)_{i,j} \, (X)_{\substack{k-i \bmod m \\ l-j \bmod n}}$$

Then,

$$\boxed{U *_{m,n} X = \frac{1}{mn} \overline{F_m} \left[(F_m U F_n)_{i,j} \overset{\text{comp. wise}}{\cdot} (F_m X F_n)_{i,j} \right]_{\substack{i=0,\dots,m-1 \\ j=0,\dots,n-1}} \overline{F_n}}$$

$$U *_{m,n} X = \mathrm{IDFT2} \left\{ [\mathrm{DFT2}(U)]_{i,j} \cdot [\mathrm{DFT2}(X)]_{i,j} \right\}_{\substack{i=0,\dots,m-1 \\ j=0,\dots,n-1}}$$

C++11 code 4.2.55: DFT-based 2D discrete periodic convolution ➺ GITLAB

```
// DFT based implementation of 2D periodic convolution
template <typename Scalar1, typename Scalar2, class EigenMatrix>
void pmconv(const Eigen::MatrixBase<Scalar1> &X, const
   Eigen::MatrixBase<Scalar2> &Y,
            EigenMatrix &Z) {
  using Comp = std::complex<double>;
  using idx_t = typename EigenMatrix::Index;
  using val_t = typename EigenMatrix::Scalar;
  const idx_t n=X.cols(),m=X.rows();
  if ((m!=Y.rows()) || (n!=Y.cols())) throw
     std::runtime_error("pmconv: size mismatch");
  Z.resize(m,n); Eigen::MatrixXcd Xh(m,n),Yh(m,n);
  // Step ❶: 2D DFT of Y
  fft2(Yh,(Y.template cast<Comp>()));
  // Step ❷: 2D DFT of X
  fft2(Xh,(X.template cast<Comp>()));
  // Steps ❸, ❹: inverse DFT of component-wise product
  ifft2(Z,Xh.cwiseProduct(Yh));
}
```

5. Data Interpolation in 1D

Given a set of data points $(t_i, y_i) \in \mathbb{R}^2$

nodes (t_i), data values (y_i)

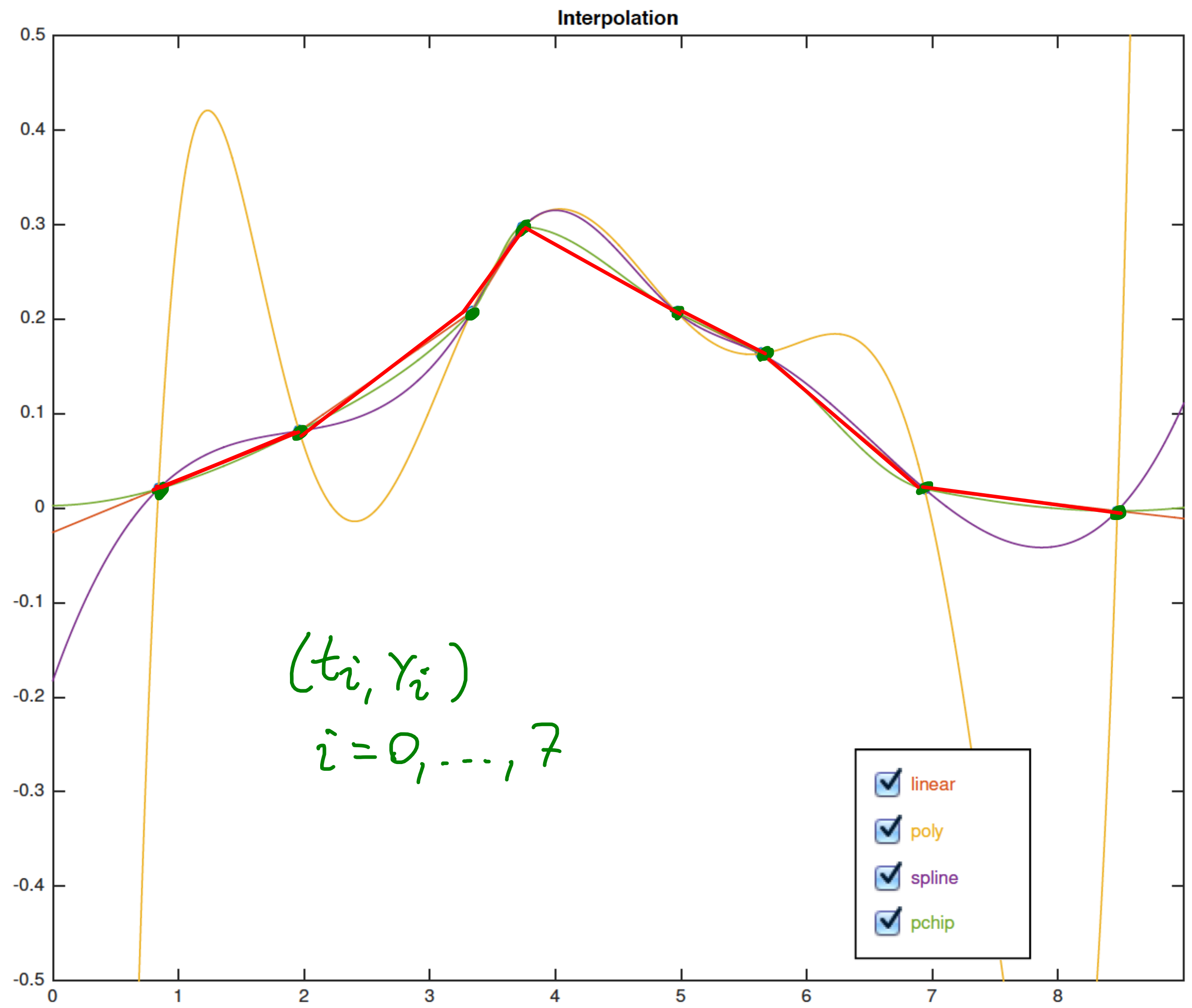
$i = 0, \dots, n$, $t_i \in I \subset \mathbb{R}$

Task: Find interpolant, i.e. a (continuous) function

$$f : I \to \mathbb{R} \quad \text{s.t.} \quad f \in C^0(I) \quad \text{and}$$

$$f(t_i) = y_i \quad \forall\, i = 0, \dots, n$$

interpolation conditions (I.C.)



Many (inf.) and very different options to interpolate!

Need additional assumptions on f, such as smoothness properties.

Typically: search for $f \in S \subset C^0(I)$

S ← $(m+1)$-dim. subspace

i.e. $S = \text{span}\{b_0, \dots, b_m\}$, $b_j \in C^0(I)$ form a basis of S.

Applications of interpolation:

- Reconstruction of constitutive relations from measurements

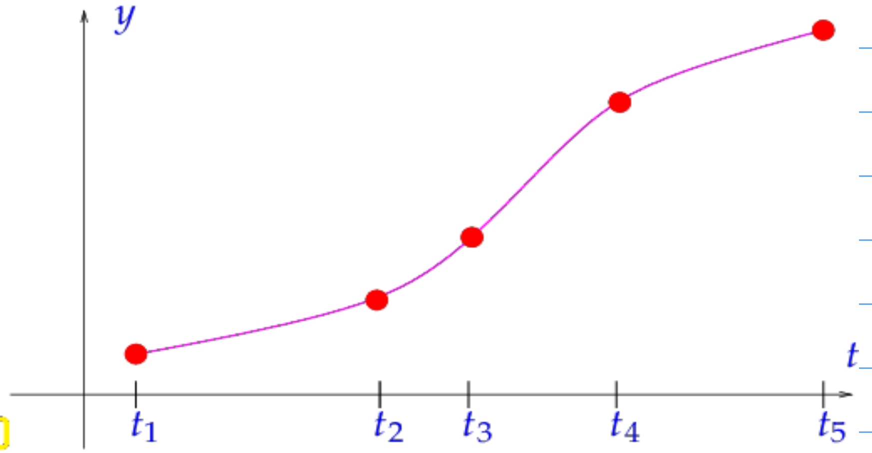
Examples: t and y could be

t	y
voltage U	current I
pressure p	density ρ
magnetic field H	magnetic flux B
...	...

Known: several *accurate* $(*)$ measurements

(t_i, y_i), $i = 1, \dots, m$

Fig. 167



E.g. a task contains knowing a relation changes in voltage U vs. changes in current I.

→ need as a model a differentiable function

$f(t) = y$.

Note: interpolation is used when measurements are assumed to be sufficiently accurate (otherwise: data fitting)

- Given some function, one looks for a "simple" approximation by taking a set of data points and interpolating in some S ($m+1$-dim)

Note: working on a computer, what does it mean to find $f: I \to \mathbb{R}$?

infinite amount of information

What is meant: subroutine that given any $t \in I \cap M$, can compute $f(t)$.

Typically: finite-dim basis of S
$\uparrow$ m-dim.

$\{b_0, \dots, b_{m-1}\}$

and $f(t) = \sum_{j=0}^{m-1} \alpha_j b_j(t)$

coefficients $\{\alpha_0, \dots, \alpha_{m-1}\}$ fully characterize f

Example: Piecewise linear interpolation

Simplest way to continuously connect data points

Piecewise linear interpolation

= connect data points (t_i, y_i), $i = 0, \dots, n$, $t_{i-1} < t_i$, by line segments

➢ interpolating polygon

Piecewise linear interpolant of data ▷

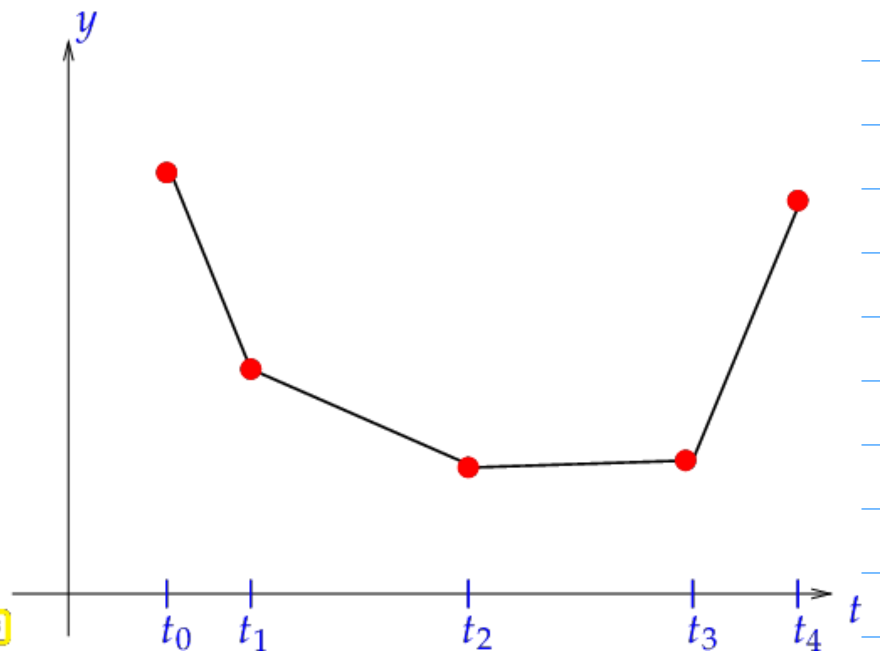


Fig. 168

Here: $S = \{ f \in C^0(I),\ \text{s.t. } f(t) = \beta_i t + \gamma_i \text{ on } [t_{i-1}, t_i],$
for $i = 0, \dots, n;\ \beta_i, \gamma_i \in \mathbb{R} \}$

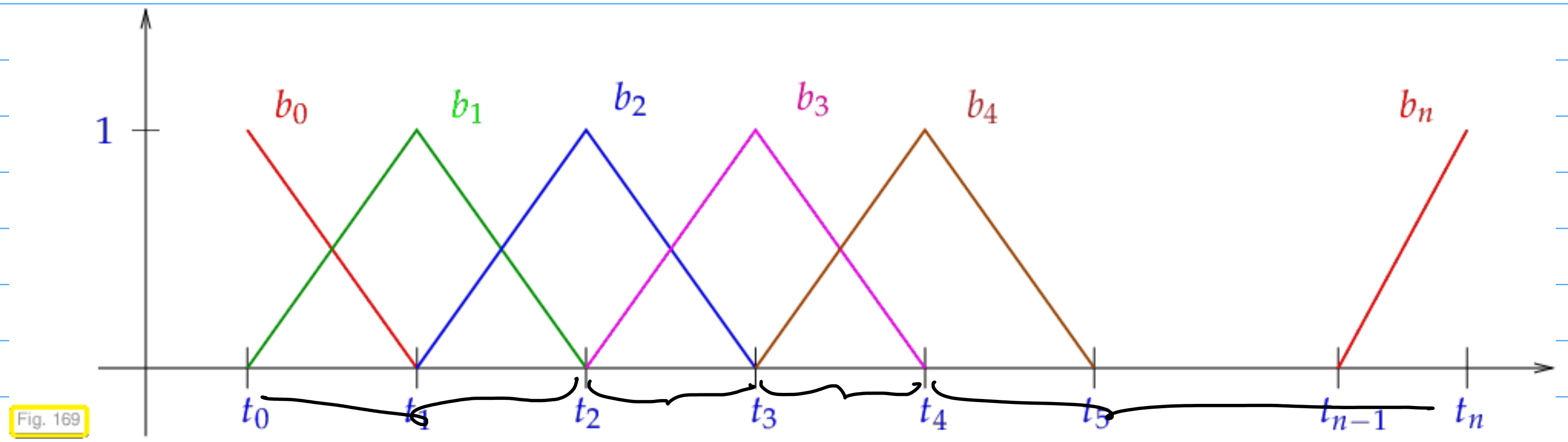
cont.

piecewise linear

points t_i are fixed

$\dim S = n+1$ [can choose $y_0, \dots, y_n$
$n+1$ degrees of freedom]

Basis for S ?



"hat functions"

Equations for b_i:

$$b_0(t) = \begin{cases} 1 - \frac{t-t_0}{t_1-t_0} & \text{for } t_0 \le t < t_1 , \\ 0 & \text{for } t \ge t_1 . \end{cases}$$

$$b_j(t) = \begin{cases} 1 - \frac{t_j-t}{t_j-t_{j-1}} & \text{for } t_{j-1} \le t < t_j , \\ 1 - \frac{t-t_j}{t_{j+1}-t_j} & \text{for } t_j \le t < t_{j+1} , \\ 0 & \text{elsewhere in } [t_0, t_n] . \end{cases} \quad , \quad j = 1, \dots, n-1 , \tag{5.1.11}$$

$$b_n(t) = \begin{cases} 1 - \frac{t_n-t}{t_n-t_{n-1}} & \text{for } t_{n-1} \le t < t_n , \\ 0 & \text{for } t < t_{n-1} . \end{cases}$$

$$b_j(t_i) = \begin{cases} 1 , & i = j \\ 0 , & i \neq j \end{cases}$$

$$= \delta_{i,j} \qquad \text{(Kronecker delta)}$$

Interpolant to $\{(t_i, y_i)\}_{i=0}^{n}$ in S ?

$$\boxed{f(t) = \sum_{j=0}^{n} y_j \cdot b_j(t)}$$

$$f(t_i) = \sum_{j=0}^{n} y_j\, b_j(t_i) = y_i \underbrace{b_i(t_i)}_{=1} = y_i$$

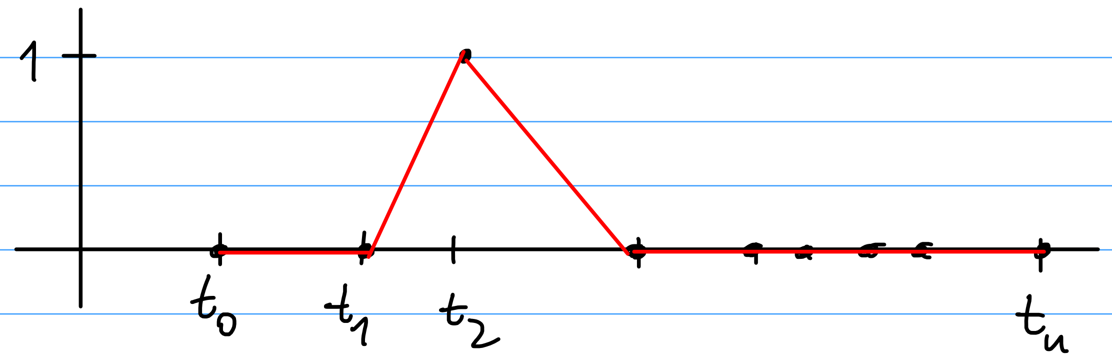
→ interpolant conditions are fulfilled

A basis $\{b_0, \dots, b_n\}$ s.t. $b_j(t_i) = \delta_{ij}$ is called a cardinal basis.

Note: • both S and basis $\{b_j\}_{j=0}^{n}$ depend on points t_i

- infinitely many choices for a basis of S.
- for S as in our example: cardinal basis is unique:

because b_j cont. pw. linear



only way to construct b_2!

<u>More general setting</u>:

- interpolating conditions: $f(t_i) = y_i \quad i = 0, \dots, n$
- basis representation: $f(t) = \sum_{j=0}^{m} \alpha_j b_j(t)$

$\{b_j\}_{j=0}^{m}$ basis of S, $f \in S$

$$f(t_i) = \sum_{j=0}^{m} \alpha_j b_j(t_i) = y_i \qquad i = 0, \dots, n$$

I.C. basis repr.

& (5.1.9) $\Rightarrow \quad f(t_i) = \sum_{j=0}^{m} \alpha_j b_j(t_i) = y_i\,, \quad i = 0, \dots, n\,,$ (5.1.14)

$$\Updownarrow$$

$$\boxed{\mathbf{Ac} := \begin{bmatrix} b_0(t_0) & \dots & b_m(t_0) \\ \vdots & & \vdots \\ b_0(t_n) & \dots & b_m(t_n) \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \vdots \\ \alpha_m \end{bmatrix} = \begin{bmatrix} y_0 \\ \vdots \\ y_n \end{bmatrix} =: \mathbf{y}}\,. \qquad (5.1.15)$$

This is an $(m+1) \times (n+1)$ linear system of equations!

Solving for $[\alpha_0, \dots, \alpha_m]^T$ would determine f!

Existence & uniqueness of interpolant for all $\mathbf{y} \in \mathbb{R}^{n+1}$

$\Leftrightarrow$ A is regular

Necessary condition: $m = n$

The map $\mathcal{I}: \begin{cases} \mathbb{R}^{n+1} \to S \subset C^0(I) \\ \mathbf{y} \mapsto f = \sum_{j=0}^{n} \underbrace{(A^{-1}\mathbf{y})_j}_{\alpha_j} b_j \end{cases}$

is a linear map

When is A invertible?

depends on:
- nodes t_i
- space S

but: is independent of choice of basis $\{b_j\}_{j=0}^n$

Why? Take 2 bases of S: $\{b_0, \dots, b_n\}$
$\{b_0', \dots, b_n'\}$

and suppose we want to solve for

$$\sum_{j=0}^{n} \beta_j \, b_j'(t_i) = y_i \qquad i = 0, \dots, n \qquad (*)$$

$b_j' \in \operatorname{span}\{b_0, \dots, b_n\} \Rightarrow \exists \{\gamma_{k,j}\}_{k=0,\dots,n}$ s.t.

$$b_j'(t_i) = \sum_{k=0}^{n} \gamma_{k,j} \, b_k(t_i)$$

$$(*) \iff \sum_{j=0}^{n} \beta_j \left(\sum_{k=0}^{n} \gamma_{k,j} b_k(t_i) \right) = y_i$$

$$\iff \sum_{k=0}^{n} \underbrace{\left(\sum_{j=0}^{n} \beta_j \gamma_{k,j} \right)}_{\alpha_k} b_k(t_i) = y_i$$

Unique solvability of $(*)$ $\iff$
unique solvability of

$$\sum_{k=0}^{n} \alpha_k \, b_k(t_i) = y_i$$

$i = 0, \dots, n$

Note: cardinal basis $\{b_0, \dots, b_n\}$
yields $A = I$

5.2. Global Polynomial Interpolation

Polynomials of degree $\leq k$, $k \in \mathbb{N}$

$$\mathcal{P}_k := \left\{ t \mapsto \sum_{i=0}^{k} \alpha_i t^i,\ \alpha_i \in \mathbb{R} \right\}$$

monomials: $t \mapsto t^i$

monomial representation of a polynomial:
linear combination of basis functions $t \mapsto t^i$

$$t \mapsto \alpha_k t^k + \ldots + \alpha_1 t + \alpha_0$$

$\dim \mathcal{P}_k = k+1$, $\mathcal{P}_k \subset C^\infty(\mathbb{R})$

$\Rightarrow$ polynomial of degree k is determined by $k+1$ points

advantage of polynomials

- differentiation & integration is easy to compute
- efficient evaluation through Horner scheme

$$t(\ldots(t(t(\alpha_k t + \alpha_{k-1}) + \alpha_{k-2}) + \ldots + \alpha_1) + \alpha_0$$

C++-code 5.2.7: Horner scheme (vectorized version)

```
// Efficient evaluation of a polynomial in monomial representation
// using the Horner scheme (5.2.6)
// IN: p = vector of monomial coefficients, length = degree + 1
// (leading coefficient in p(0), MATLAB convention Rem. 5.2.4)
// t = vector of evaluation points t_i
// OUT: y = polynomial evaluated at t_i
void horner(const VectorXd& p, const VectorXd& t, VectorXd& y) {
  const VectorXd::Index n = t.size();
  y.resize(n); y = p(0)*VectorXd::Ones(n);
  for (unsigned i = 1; i < p.size(); ++i)
    y = t.cwiseProduct(y) + p(i)*VectorXd::Ones(n);
}
```

$p = [\alpha_k, \ldots, \alpha_0]$

Computational complexity: $O(k)$

- Approximation property of polynomials

5.2.2. Theory

Lagrange polynomial interpolation problem

Given the simple nodes $t_0, \dots, t_n$, $n \in \mathbb{N}$, $-\infty < t_0 < t_1 < \dots < t_n < \infty$ and the values $y_0, \dots, y_n \in \mathbb{R}$ compute $p \in \mathcal{P}_n$ such that

$$p(t_j) = y_j \quad \text{for} \quad j = 0, \dots, n\,. \tag{5.2.9}$$

$(n+1) \times (n+1)$ LSE

We know: $\dim \mathcal{P}_n = n+1$

and monomials $\{t \mapsto t^k\}_{k=0}^n$ form a basis of $\mathcal{P}_n$

Could build matrix A as before ...

Easier approach: building a cardinal basis

for $\{t_j\}_{j=0}^n$ and $S = \mathcal{P}_n$ (i.e. $A = I$)

Lagrange polynomials:

$$L_i(t) := \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{t - t_j}{t_i - t_j} \qquad i = 0, \dots, n$$

- $L_i \in \mathcal{P}_n$ ✓
- $L_i(t_\ell) = \delta_{i\ell}$?

$$L_i(t_\ell) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{t_\ell - t_j}{t_i - t_j} = \begin{cases} \prod\limits_{j \neq i} \dfrac{t_i - t_j}{t_i - t_j} = \underline{\underline{1}} & \text{if } \ell = i \\ \underline{\underline{0}} & \text{if } \ell \neq i \end{cases}$$

- Linear independence?

Consider:

$$\mu_0 L_0(t) + \mu_1 L_1(t) + \dots + \mu_n L_n(t) = 0$$

Then, choosing $t = t_i$

$$\mu_i \underbrace{L_i(t_i)}_{=1} = 0 \Rightarrow \mu_i = 0$$

plug in all t_i for $i = 0, \dots, n$

$$\Rightarrow \quad \mu_0 = \mu_1 = \dots = \mu_n = 0.$$

$\Rightarrow$ Linear independence.

$n+1$ linearly ind. polynomials $L_0(t), \dots, L_n(t) \in \mathcal{P}_n$

$\Rightarrow$ Lagrange polynomials are a cardinal basis for $\{t_i\}_{i=0}^{n}$ and $\mathcal{P}_n$.

$\Rightarrow$ Existence & uniqueness of interpolant.

Theorem 5.2.14. Existence & uniqueness of Lagrange interpolation polynomial → [?, Thm. 8.1], [?, Satz 8.3]

The general Lagrange polynomial interpolation problem admits a unique solution $p \in \mathcal{P}_n$.

$$p(t) = \sum_{i=0}^{n} y_i L_i(t)$$

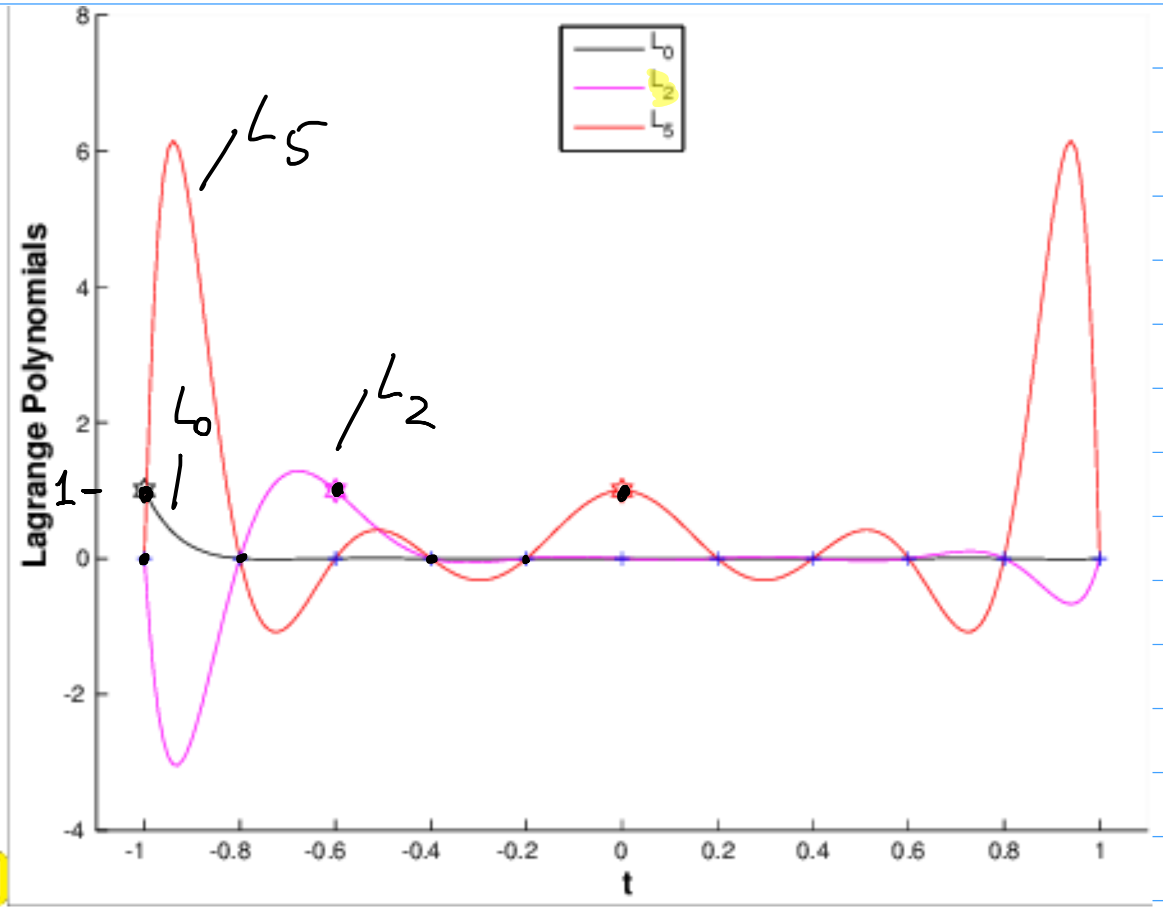


Fig. 170