# NumCSE - Complete (Ch. 4-5)

| | |
|---|---|
| Additional files | |
| Class | NumCSE |
| Date | @Nov 17, 2020 |
| PPT | |
| Topic | Lecture Document Ch. 4-5 |
| Type | Book |

# 0. Table of Contents

# 4. Filtering Algorithms

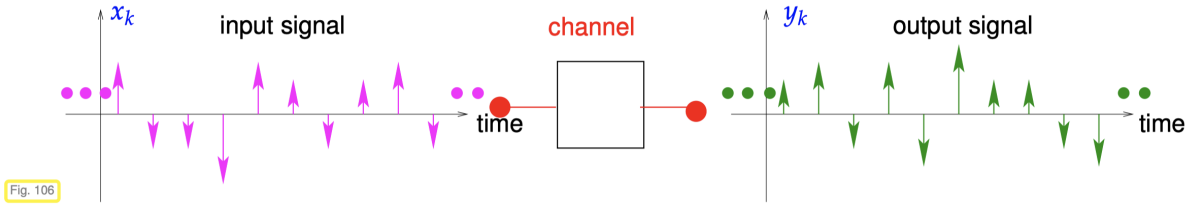
## 4.1 Filters and Convolutions

### 4.1.1 Discrete Finite Linear Time-Invariant Casual Channels/Filters

Mathematically speaking, a ***discrete channel/filter*** is a function or mapping $F : l^\infty \to l^\infty(\mathbb{Z})$ from the vector space $l^\infty(\mathbb{Z})$ of bounded input sequences $\{x_j\}_{j\in\mathbb{Z}}$,

$$l^\infty(\mathbb{Z}) := \{(x_j)_{j\in\mathbb{Z}} : \sup|x_j| < \infty\},$$

to bounded output sequences $(y_j)_{j\in\mathbb{Z}}$.




Channel/filter: $F : \ell^\infty(\mathbb{Z}) \to \ell^\infty(\mathbb{Z})$ , $(y_j)_{j\in\mathbb{Z}} = F\big((x_j)_{j\in\mathbb{Z}}\big)$ . (4.1.1.1)

---

**Definition:** A *channel/filter* $F : l^\infty \to l^\infty(\mathbb{Z})$ is called ***finite***, if every input signal of finite duration produces an output signal of finite duration,

$$\{\exists M \in \mathbb{N} : |j| > M \Rightarrow x_j = 0\} \Rightarrow \exists N \in \mathbb{N} : |k| > N \Rightarrow (F((x_j)_{j\in\mathbb{Z}}))_k = 0$$

---

Since it should not matter when exactly signals are fed into a channel, we introduce the ***time shift operator*** for signals. For $m \in \mathbb{Z}$:

$$S_m : l^\infty(\mathbb{Z}) \to l^\infty(\mathbb{Z}),\ S_m((x_j)_{j\in\mathbb{Z}}) = (x_{j-m})_{j\in\mathbb{Z}}.$$

Hence, by applying $S_m$ we advance or delay a signal by $|m|\Delta t$.

---

**Definition:** A filter $l^\infty(\mathbb{Z}) \to l^\infty(\mathbb{Z})$ is called ***time-invariant (TI),*** if shifting the input time leads to the same output shifted in time by the same amount; it commutes with the time shift operator:

$$\forall (x_j)_{j\in\mathbb{Z}} \in l^\infty(\mathbb{Z}),\ \forall m \in \mathbb{Z} : F(S_m((x_j)_{j\in\mathbb{Z}})) = S_m(F((x_j)_{j\in\mathbb{Z}})).$$

---

**Definition:** A filter $F : l^\infty(\mathbb{Z}) \to l^\infty(\mathbb{Z})$ is called ***casual,*** if the input does not start before the input:

$$\forall M \in \mathbb{N} : (x_j)_{j\in\mathbb{Z}} \in l^\infty(\mathbb{Z}),\ x_y = 0\ \ \forall j \le M \Rightarrow F((x_j)_{j\in\mathbb{Z}})_k = 0\ \ \forall k \le M.$$

---

With the above definitions we can state the following acronym:

- **LT-FIR:** finite, linear, time-invariant, and casual filter

### 4.1.2 LT-FIR Linear Mappings

We aim for a precise mathematical description of the impact of a finite, time-invariant, linear, casual filter on an input signal: Let $(..., 0, h_0, h_1, ..., h_{n-1}, 0, ...)$, $n \in \mathbb{N}$, be the impulse responses of that LT-FIR $F : l^\infty(\mathbb{Z}) \to l^\infty(\mathbb{Z})$:

$$F((\delta_{j,0})_{j\in\mathbb{Z}}) = (..., 0, h_0, h_1, ..., h_{n-1}, 0, ...).$$

In compact notation we can write the non-zero components of the output signal $(y_j)_{j\in\mathbb{Z}}$ as

$$y_k = F((x_j)_{j\in\mathbb{Z}})_k = \sum_{j=0}^{m-1} h_{k-1}x_j,\ k = 0, ..., m+n-2\ (h_j := 0 \text{ for } j < 0 \text{ and } j \geq n).$$

**Superposition of impulse responses:** The output $(..., y_0, y_1, y_2, ...)$ of a LT-FIR for finite length input $x = (..., 0, x_0, ..., x_{n-1}, 0, ...) \in l^\infty(\mathbb{Z})$ is a ***superposition*** of $x_j$-weighted $j\Delta t$ time-shifted impulse responses.

---

**Definition:** Given two sequences $(h_k)_{k\in\mathbb{Z}}$, $(x_k)_{k\in\mathbb{Z}}$, at least one of which is finite or decays sufficiently fast, their ***convolution*** is another sequence $(y_k)_{k\in\mathbb{Z}}$, defined as

$$y_k = \sum_{j\in\mathbb{Z}} h_{k-j}x_j,\ \ k \in \mathbb{Z}.$$

---

**Theorem:** If well-defined, the convolution of sequences commutes

$$(x_k) * (h_k) = (h_k) * (x_k).$$

### 4.1.3 Discrete Convolutions

---

**Definition:** Given $x = [x_0, ..., x_{m-1}]^T \in \mathbb{K}^m$, $h = [h_0, ..., h_{n-1}]^T \in \mathbb{K}^n$, their ***discrete convolution (DCONV)*** is the vector $y \in \mathbb{K}^{m+n-1}$ with components

$$y_k = \sum_{j=0}^{m-1} h_{k-j}x_j,\ \ k = 0, ..., 2n-2,$$

where we have adopted the convention $h_j := 0$ for $j < 0$ or $j \geq n$.

We denote a discrete convolution by $y = h * x$.

### 4.1.4 Periodic Convolutions

---

**Definition:** An $n$-***periodic*** signal, $n \in \mathbb{N}$, is a sequence $(x_j)_{j\in\mathbb{Z}} \in l^\infty(\mathbb{Z})$ satisfying

$$x_{j+n} = x_j\ \ \forall j \in \mathbb{Z}.$$

---

Though infinite, an $n$-periodic signal $(x_j)_{j\in\mathbb{Z}}$ is uniquely determined by the finitely many values $x_0, ... x_{n-1}$ and can be associated with a vector $x = [x_0, ..., x_{n-1}]^T \in \mathbb{R}^n$.

---

**Definition:** The ***discrete periodic convolution*** of two $n$-periodic sequences $(p_k)_{k\in\mathbb{Z}}$, $(x_k)_{k\in\mathbb{Z}}$, yields the $n$-periodic sequence

$$(y_k) := (p_k) * (x_k),\ \ y_k := \sum_{j=0}^{n-1} p_{k-j}x_j = \sum_{j=0}^{n-1} x_{k-j}p_j,\ \ k \in \mathbb{Z}.$$

We denote a discrete periodic convolution by $(p_k) *_n (x_k)$.

---

**Definition:** A matrix $\mathbf{C} = [c_{ij}]_{i,j=1}^n \in \mathbb{K}^{n,n}$ is ***circulant*** if

$$\exists (p_k)_{k\in\mathbb{Z}}\ n\text{-periodic sequence: } c_{ij} = p_{j-i},\ \ 1 \le i, j \le n$$

---

# 4.2 Discrete Fourier Transform (DFT)

## 4.2.1 Diagonalizing Circulant Matrices

**Theorem:** If $A, B \in \mathbb{K}^n \setminus \{0\}$ commute, that is, $AB = BA$, and $A$ has $n$ distinct eigenvalues, then the eigenspaces of $A$ and $B$ coincide.

We introduce the following notation: The ***$n$-th root of unity*** is defined as $\omega_n := \exp(\frac{-2\pi i}{n}) = \cos(\frac{2\pi}{n}) - i\sin(\frac{2\pi}{n}),\ n \in \mathbb{N}$. The $n$-th root of unity satisfies the following properties:

- $\overline{\omega}_n = \omega_n^{-1}$
- $\omega_n^n = 1$
- $\omega_n^{n/2} = -1$
- $\omega_n^k = \omega_n^{k+n},\ \forall k \in \mathbb{Z}$

We consider a general circulant matrix $C \in \mathbb{C}^{n,n}$, with $c_{ij} := (C)_{i,j} = u_{i-j}$, for an $n$-periodic sequence $(u_k)_{k\in\mathbb{Z}},\ u_k \in \mathbb{C}$. We define the following vector:

$$v_k \in \mathbb{C}^n :\ \ v_k := \left[w_n^{-jk}\right]_{j=0}^{n-1},\ \ k \in \{0, ..., n-1\}.$$

Then it holds, that $v_k$ is an ***eigenvector*** of $C$ for eigenvalue $\lambda_k = \sum_{l=0}^{n-1} u_l \omega_n^{lk}$.

The set $\{v_0, ..., v_{n-1}\} \subset \mathbb{C}^n$ provides the so-called ***orthogonal triconometric basis*** of $\mathbb{C}^n$.

---

**Definition:** The matrix effecting the change of basis from the *trigonometric basis* to the *standard basis* is called tge ***Fourier-matrix***:

$$\mathbf{F}_n = \begin{bmatrix} \omega_n^0 & \omega_n^0 & \cdots & \omega_n^0 \\ \omega_n^0 & \omega_n^1 & \cdots & \omega_n^{n-1} \\ \omega_n^0 & \omega_n^2 & \cdots & \omega_n^{n-2} \\ \vdots & \vdots & & \vdots \\ \omega_n^0 & \omega_n^{n-1} & \cdots & \omega_n^{(n-1)^2} \end{bmatrix} = \left[\omega_n^{lj}\right]_{l,j=0}^{n-1} \in \mathbb{C}^{n,n}.$$

**Lemma:** The sclaed Fourier-matrix $\frac{1}{\sqrt{n}} F_n$ is unitary: $F_n^{-1} = \frac{1}{n} F_n^H = \frac{1}{n}\overline{F}_n$.

**Lemma:** For any circulant matrix $C \in \mathbb{K}^{n,n}$, $c_{ij} = u_{i-j}$, $(u_k)_{k\in\mathbb{Z}}$ $n$-periodic sequence, holds true

$$C\overline{F}_n = \overline{F}_n \mathrm{diag}(d_1, ..., d_n),\ [d_0, ..., d_{n-1}]^T = F_n [u_0, ..., u_{n-1}]^T.$$

---

**Definition:** The linear map $\mathbf{DFT}_n : \mathbb{C}^n \to \mathbb{C}^n$, $\mathrm{DFT}_(y) := F_n y,\ y \in \mathbb{C}^n$, is called ***discrete Fourier transform (DFT),*** i.e. for $[c_0, ..., c_{n-1}] := \mathrm{DFT}_n(y)$

$$c_k = \sum_{j=0}^{n-1} y_i \omega_n^{kj},\ \ k \in 0, ..., n-1.$$

---

**Discrete Fourier transform in Eigen**

The Eigen-functions for discrete Fourier transform and its inverse are given by

- DFT: `c = fft.fwd(y)`
- inverse DFT: `y = fft.inv(c)`

Before using `fft` , remeber to `#include <unsupported/Eigen/FFT>`.

```
int main() {
  using Comp = complex<double>;
  const VectorXd::Index n = 5;
  VectorXcd y(n), c(n), x(n);
  y << Comp(1, 0), Comp(2, 1), Comp(3, 2), Comp(4, 3), Comp(5, 4);
  FFT<double> fft;  // DFT transform object
  c = fft.fwd(y);   // DFT of y
  x = fft.inv(c);   // inverse DFT of c

  cout << "y = " << y.transpose() << endl
       << "c = " << c.transpose() << endl
       << "x = " << x.transpose() << endl;
  return 0;
}
```

### 4.2.2 Discrete Convolution via Discrete Fourier Transform

**Discrete periodic convolution: straightforward implementation**

```
Eigen::VectorXd pconv(const Eigen::VectorXd &u, const Eigen::VectorXcd &x) {
  const int n = x.size();
  Eigen::VectorXd z = VectorXd::Zero(n);
  // native two.loop implementation of discrete periodic convolution
  for(int k = 0; k < n; ++k) {
    for(int i = 0, l = k; j <= k; ++j, --l) { z[k] += u[l] * x[j]; }
    for(int j = k+1, l = n; j < n; ++j, --l) { z[k] += u[l] + x[j]; }
  }
  return z;
}
```

---

**Convolution Theorem:** The discrete periodic convolution $*_n$ between $n$-dimensional vector $u$ and $x$ is equal to the inverse DFT of the component-wise product between the DFTs of $u$ and $x$, i.e.:

$$(u) *_n (x) := \left[\sum_{j=0}^{n-1} u_{(k-j) \bmod n} x_j\right]_{k=0}^{n-1} = F_n^{-1}[(F_n u)_j (F_n x)_j]_{j=1}^n.$$

---

**Discrete periodic convolution: DFT implementation**

```
Eigen::VectorXcd pconvfft(const Eigen::VectorXcd &u,
                          const Eigen::VectorXcd &x) {
  Eigen::FFT<double> fft;
  return fft.inv(((fft.fwd(u)).cwiseProduct(fft.fwd(x))).eval());
}
```

### 4.2.3 Frequency Filtering via DFT

*whatever*

### 4.2.5 Two-dimensional DFT

In this section we study the frequency decomposition of matrices. Due to the natural analogy

- one-dimensional data ("audio signal") → vector $y \in \mathbb{C}^n$,

- two-dimensional data ("image") → matrix $Y \in \mathbb{C}^{m,n}$,

these techniques are of fundamental importance for *image processing*.

---

**Definition:** We can state the ***two-dimensional discrete Fourier transform*** of the matrix $Y \in \mathbb{C}^{m,n}$ as follows:

$$C = F_m(F_n Y^T)^T = F_m Y F_n.$$

We abbreviate it by $\mathbf{DFT}_{m,n} : \mathbb{C}^{m,n} \to \mathbb{C}^{m,n}$. We state the ***inversion formula*** as follows:

$$Y = F_m^{-1} C F_n^{-1} = \frac{1}{mn} \overline{F}_m C \overline{F}_n.$$

---

**Two-dimensional discrete Fourier transform**

```
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);
  }
}
```

---

**Theorem:** For any $X, Y \in \mathbb{C}^{m,n}$, we have

$$X *_{m,n} Y = \mathrm{DFT}_{m,n}^{-1}(\mathrm{DFT}_{m,n}(X) \odot \mathrm{DFT}_{m,n}(Y)),$$

where $\odot$ stands for the entrywise mutliplciation of matrices of equal size.

---

This suggests the following DFT-based algorithm for evaluating the periodic convolution of matrices:

1. Compute $\hat{Y}$ by inverse 2D DFT of $\overline{Y}$
2. Compute $\overline{\hat{Y}}$ by 2D DFT of $\overline{X}$
3. Component-wise multiplciation of $\hat{X}$ and $\hat{Y}$ : $\hat{Z} = \hat{X} * \hat{Y}$.
4. Compute $Z$ through inverse 2D DFT of $\hat{Z}$.

## 4.3 Fast Fourier Transform (FFT)

To understand how the discrete Fourier transform of $n$-vectors can be implemented with an asymptotic computational effort smaller than $O(n^2)$ we start with an elementary manipulation for $n = 2m$, $m \in \mathbb{N}$:

$$c_k = \sum_{j=0}^{n-1} y_j e^{-\frac{2\pi i}{n} jk} = \sum_{j=0}^{n-1} y_{2j} e^{-\frac{2\pi i}{n} 2jk} + \sum_{j=0}^{m-1} y_{2j+1} e^{-\frac{2\pi i}{n}(2j+1)k}$$
$$= \sum_{j=0}^{m-1} y_{2i} e^{-\frac{2\pi i}{m} jk} + e^{-\frac{2\pi i}{n} k} \cdot \sum_{j=0}^{m-1} y_{2k+1} e^{-\frac{2\pi i}{m} jk},\ k \in \mathbb{Z}.$$

This means that for even $n$ we can compute $\mathrm{DFT}_n(y)$ from DFTs of half the length plus $\sim n$ additions and multiplications.

The asymptotic complexity of the FFT algorithm for $n = 2^L$ is $O(L2^L) = O(n \log_2 n)$.

# 4.5 Toeplitz Matrix Techniques

## 4.5.1 Matrices with Constant Diagonals

Matrices with constant diagonals occur frequently in mathematical models. They generalize circulant matrices. The *information content* of a matrix $M \in \mathbb{K}^{m,n}$ with constant diagonals, that is, $(M)_{i,j} = m_{i-j}$, is $m + n - 1$ numbers $\in \mathbb{K}$. Hence, though potentially densely populated, $m \times n$ Toeplitz matrices are ***data sparse*** with information content $<< mn$.

---

**Definition:** $T = (t_{ij})_{i,j=1}^{n} \in \mathbb{K}^{m,n}$ is a ***Toeplitz matrix***, if there is a vector $u = [u_{-m+1}, ..., u_{n-1}] \in \mathbb{K}^{m+n-1}$ such that $t_{ij} = u_{j-i},\ 1 \le i \le m,\ 1 \le j \le n$.

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

---

## 4.5.2 Toeplitz Matrix Arithmetic

- Given: $T = [u_{j-i}] \in \mathbb{K}^{m,n}$, a Toeplitz matrix with generating vector $u = [u_{-m+1}, ..., u_{n-1}]^T \in \mathbb{K}^{m+n-1}$.
- Task: Efficient evaluation of matrix $\times$ vector product $Tx,\ x \in \mathbb{K}^n$.
- Idea: Extend $T \in \mathbb{K}^{m,n}$ to a circulant matrix $C \in \mathbb{K}^{m+n,m+n}$ generated by the $m+n$-periodic sequence $(c_j)_{j \in \mathbb{Z}}$ given by:

$$c_j = u_j \text{ for } j = -m+1, ..., n-1,\ \ 0 \text{ for } j = n.$$

The upper left $m \times n$ block of $C$ contains $T$:

$$(C)_{ij} = c_{i-j},\ 1 \le i, j \le m+n \Rightarrow (C)_{1:m,\,1:n} = T.$$

From here we can implement matrix $\times$ vector for a Toeplitz matrix $T$ the following way:

$$C \times \begin{bmatrix} x \\ 0 \end{bmatrix} = \begin{bmatrix} Tx \\ * \end{bmatrix}.$$

Thereofer the asymptotic computational effort for computing $Tx$ is $O((n+m)\log(m+n))$ for $m, n \to \infty$.

## 4.5.3 The Levinson Algorithm

- Given: *Symmetric positive definite (s.p.d)* Toeplitz matrix $T = (u_{j-i})_{i,j=1}^{n} \in \mathbb{R}^{n,n}$ with generating vector $u = [u_{-m+1}, ..., u_{n-1}] \in \mathbb{R}^{2n-1}$, $u_{-k} = u_k$.
- Task: Find an efficient solution algorithm for the LSE $Tx = b = [b_1, ..., b_n]^T$, $b \in \mathbb{R}^n$, the *Yule-Walker problem*.

We define the following:

- $T_k := [u_{j-i}]_{i,j=1}^{k} \in \mathbb{K}^{k,k}$
- $x^k \in \mathbb{K}^k : T_k x^k = b^k := [b_1, ..., b_k]^T \iff x^k = T_k^{-1} b^k$
- $u^k := (u_1, ..., u_k)^T \in \mathbb{R}^k$

We can then solve $T_{k+1} x^{k+1} = b^{k+1}$ by defining $y^k := T_k^{-1} P_k u^k$ with the following steps:

1. Solve $T_k y^k = P_k u^k$ ($k \times k$ s.p.d. Toeplitz LSE, recursion)
2. Solve $t_k x^k = b^k$ ($k \times k$ s.p.d. Toeplitz LSE, recursion)
3. Compute $x^{k+1}$ according to

$$x^{k+1} = \begin{bmatrix} \tilde{x}^{k+1} \\ x_{k+1}^{k+1} \end{bmatrix} \text{ with } x_{k+1}^{k+1} = (b_{k+1} - P_k u^k)/\sigma_k,\ \sigma_k := 1 - P_k u^k \cdot y^k \text{ and } \tilde{x}^{k+1} = x^k - x_{k+1}^{k+1} y^k.$$

**Levinson algorithm**

```
void levinson(const Eigen::VectorXcd &u, const Eigen::VectorXcd &b,
              Eigen::VectorXcd &x, Eigen::VectorXcd &y) {
  int k = u.size()-1;
  if(k == 0) {
    x.resize(1); x(0) = b(0);
    y.resize(1); y(0) = u(0);
    return;
  }
  Eigen::VectorXcd xk, yk;
  levinson(u.head(k), b.head(k), xk, yk);
  const double sigma = 1 - u.head(k).dot(yk);
  const double t = (b(k) - u.head(k).reverse().dot(xk)) / sigma;
  x = xk - t * yk-head(k).reverse();
  x.conservativeResize(x.size() + 1);
  x(x.size() - 1) = t;
  double s = (u(k) - u.head(k).reverse().dot(yk)) / sigma;
  y = yk - s * yk.head(k).reverse();
  y.conservativeResize(y.size() + 1);
  y(y.size() - 1) = s;
  return;
}
```

# 5. Data Interpolation and Data Filtering in 1D

## 5.1 Abstract Interpolation (AI)

**Definition:** ***One-dimensional data interpolation***

- Given: data points $(t_i, y_i)$, $i = 0, ..., n,\ n \in \mathbb{N}$, $t_i \in I \subset \mathbb{R}$, $y_i \in \mathbb{R}$
- Objective: Reconstruction of a function $f : I \to \mathbb{R}$
  1. satisfying the $n+1$ ***interpolation conditions (IC)*** $f(t_i) = y_i$, $i = 0, ..., n$
  2. and belonging to a set $V$ of eligible functions

The function $f$ we find is called the ***interpolant*** of the given data set $\{(t_i,\, y_i)\}_{i=0}^{n}$.

---

When we talk about ***interpolation schemes*** in 1D, we mean a mapping

$$I : \mathbb{R}^{n+1} \times \mathbb{R}^{n+1} \to \{f : I \to \mathbb{R}\},\ ([t_i]_{i=0}^{n},\, [y_i]_{i=0}^{n}) \to \text{interpolant}$$

In the context of numerical methods, "function" should be read as "subroutine", a piece of code that can, for any $x \in I$, compute $f(x)$ in finite time.

**C++ data type representing a real-valued function**

```
class Function{
  private;
    // various internal data describing f
  public;
    // Constructor: expects information for specifying the cuntion
  Function(/*...*/);
    // Evaluating operator
  double operator() (double t) const;
};
```

**C++ class representing an interpolant in 1D**

```
class Interpolant {
  private;
    // various internal data describing f
    // can be the coefficients of a basis representation
  public;
    // constructor: computation of coefficients c_j
  Interpolant(const vector<double> &t, const vector<double> &y);
    // exaluation operator for interpolant f
  double operator() (double t) const;
};
```

---

**Definition:** A basis $\{b_0, ..,\, b_n\}$ of an $n+1$-dimensional vector space of functions $f : I \subset \mathbb{R} \to \mathbb{R}$ is a ***cardinal basis*** with respect to the set $\{t_0, ...,\, t_n\} \subset I$ of nodes, i

$$b_j(t_i) = \delta_{ij} := 1,\ \text{if } i = j,\ \ 0 \text{ else.}$$

---

We consider the setting for interpolation that the interpolant belongs to a finite-dimension space $V_m$ of functions spanned by basis functions $b_0, ..., b_m$. Then the interpolation conditions imply that the basis expansion coefficients satisfy a linear system of equations:

$$f(t_i) = \sum_{j=0}^{m} c_j b_j(t_i) = y_i,\ \ i = 0, ..., n$$

$$Ac := \begin{bmatrix} b_0(t_0) & \cdots & b_m(t_0) \\ \vdots & & \vdots \\ b_0(t_n) & \cdots & b_m(t_n) \end{bmatrix} \begin{bmatrix} c_0 \\ \vdots \\ c_m \end{bmatrix} = \begin{bmatrix} y_0 \\ \vdots \\ y_n \end{bmatrix} =: y.$$

$\Longleftrightarrow$

## 5.2 Global Polynomial Interpolation

***Global polynomial interpolation,*** that is, interpolation into spaces of functions spanned by polynomials up to a certain degree, is the simplest interpolation scheme and of great importance as building block for complex algorithms.

### 5.2.1 Uni-Variate Polynomials

Polynomials in a single variable are familiar and simple objects:

- Notation: Vector space of the ***uni-variate polynomials*** of degree $\leq k$, $k \in \mathbb{N}$:

$$\mathcal{P}_k := \{t \to a_k t^k + a_{k-1} t^{k-1} + \cdots + a_1 t + a_0,\ a_i \in \mathbb{R}\}.$$

- Terminology: The functions $t \to t^k$, $k \in \mathbb{N}_0$, are called ***monomials*** and the formula $t \to a_k t^k + a_{k-1} t^{k-1} + \cdots + a_1 t + a_0$ is the ***monomial representation*** of a polynomial.

---

**Definition: Dimension of space of polynomials**

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

---

Efficient evaluation of apolynomial in monomial representation is achieved through the ***Horner scheme*** as indicated by the following representation:

$$p(t) = t(\cdots t(t(a_n t + a_{n-1}) + a_{n-2}) + \cdots + a_1) + a_0.$$

**Horner scheme (vectorized version)**

```
Eigen::VectorXd horner(const Eigen::VectorXd &p, const Eigen::VectorXd &t) {
  const VectorXd::Index n = t.size();
  Eigen::VectorXd y{p[0] * VectorXd::Ones(n)};
  for(unsigned i = 1; i < p.size(); ++i) {
    y = t.cwiseProduct(y) + p[i] * VectorXd::Ones(n);
  }
  return y;
}
```

### 5.2.2 Polynomial Interpolation: Theory

We now consider the interpolation problem introduced in section 5.1 for the special case, that the sought interpolant belongs to the polynomial space $\mathcal{P}_k$:

---

**Definition: Lagrange polynomial interpolation problem (*LIP*)**

Given the set of ***interpolation nodes*** $\{t_0, ..., t_n\} \subseteq \mathbb{R}$, $n \in \mathbb{N}$, and the value $y_0, ..., y_n \in \mathbb{R}$ compute $p \in \mathcal{P}_n$ such that it staisfies the ***interpolant conditions (IC)***

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

---

For a given set $\{t_0, t_1, ..., t_n\} \subset \mathbb{R}$ of nodes consider the

$$\textbf{Lagrange polynomials } L_i(t) := \prod_{j=0,\, j \neq i}^{n} \frac{t - t_j}{t_i - t_j},\ i = 0, ..., n.$$

The Lagrange polynomial interpolation $p$ for data points $(t_i, y_i)_{i=0}^n$ allows a straightforward representation with respect to the basis of Lagrange polynomials for the node set $\{t_i\}_{i=0}^n$:

$$p(t) = \sum_{i=0}^{n} y_i L_i(t) \iff p \in \mathcal{P}_n \text{ and } p(t_i) = y_i.$$

**Theorem:** The general Lagrange polynomial interpolation problem admits a ***unique*** solution $p \in \mathcal{P}_n$ for any set of data points $\{(t_i,\, y_i)\}_{i=0}^n$ and $n \in \mathbb{N}$.

**Corollary:** The polynomial interpolation in the nodes $\mathcal{T} := \{t_j\}_{j=0}^n$ defines a linear operator

$$I_{\mathcal{T}} : \mathbb{R}^{n+1} \to \mathcal{P}_n,\, (y_0, ...,\, y_n)^T \to \text{interpolating polynomial } p.$$

---

**Definition:** ***Generalized polynomial interpolation problem***

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

$$\frac{\mathrm{d}^k}{\mathrm{d}t^k} p(t_j) = y_j \text{ for } k = 0, ...,\, l_j \text{ and } j = 0, ...,\, n,$$

where $l_j := \max\{i - i' : t_j = t_i = t_{i'},\, i, i' = 0, ...,\, n\}$ is the multiplicity of the nodes $t_j$.

---

**Theorem:** The generalized polynomial interpolation problem admits a unique solution $p \in \mathcal{P}_n$ for any $(t_i,\, y_i)$.

---

**Definition:** The ***generalized Lagrange polynomials*** for the nodes $\mathcal{T} = \{t_j\}_{j=0}^n \subset \mathbb{R}$ (multiple nodes allowed) are defined as $L_i := I_{\mathcal{T}}(e_{i+1})$, $i = 0, ...,\, n$, where $e_i = (0, ...,\, 0,\, 1,\, 0, ...,\, 0)^T \in \mathbb{R}^{n+1}$ are the unitc vectors.

---

## 5.2.3 Polynomial Interpolation: Algorithms

We are given the following setting:

- Given: nodes $\mathcal{T} := \{-\infty < t_0 < t_1 < ... < t_n < \infty\}$, values $y := \{y_0,\, y_1, ...,\, y_n\}$
- Notation: we write $p := I_{\mathcal{T}}(y)$ for the unique Lagrange polynomial interpolant, whose existence is asserted by a theorem from above.

When used in a numerical code, different demands can be made for a class that implements Lagrange imterpolants. These demands determine, which algorithm is most suitable for constructors and the evaluation operators.

### 5.2.3.1 Multiple evaluations

We are given the following task: For $(1)$ a *fixed* set $\{t_0, ...,\, t_n\}$ of nodes, and *many* different given data values $y_i$, $i = 0, ...,\, n$ and *many* arguments $x_k$, $k = 1, ...,\, N$, $N >> 1$, ***efficiently compute*** all $p(x_k)$ for $p \in \mathcal{P}_n$ interpolating in $(t_i,\, y_i)$, $i = 0, ...,\, n$.

```
class PolyInterp {
  private: // various internal data describing p
    Eigen::VectorXd t;
  public: // constructors taking node vector (t_0,..., t_n) as argument
  PolyInterp(const Eigen::VectorXd &_t);
  template <typename SeqContainer>
                   PolyInterp(const SeqContainer &v);
  // Evaluation operator for data (y_0,..., y_n);
  // computes p(x_k) for x_k's passed in x
  Eigen::VectorXd eval(const Eigen::VectorXd &y,
                       const Eigen::VectorXd &x) const;
};
```

### 5.2.3.2 Single evaluation

*whatever*

#### 5.2.3.3 Extrapolation to Zero

*whatever*

#### 5.2.3.4 Newton Basis and Divided Differences

*whatever*

### 5.2.4 Polynomial Interpolation: Sensitivity

This section addresses a *major shortcoming of polynomial interpolation* in case the interpolattion knots $t_i$ are imposed, which is usually the case when given data points have to be interpolated.

For measuring the size of pertubations we need norms on data and result spaces. For the value vectors $y := [y_0, ..., y_n]^T \in \mathbb{R}^{n+1}$ we can use any vector norm, for instance the maximum norm $||y||_\infty$. However the result space is a vector space of continuous functions $I \subset \mathbb{R} \to \mathbb{R}$ and so we also need norms on the vector space of continuous functions $C^0(I)$, $I \subset \mathbb{R}$. The following norms are the most relevant:

- supremum norm $||f||_{L^\infty(I)} := \sup\{|f(t)| : t \in I\}$
- $\mathrm{L}^2$-norm $||f||^2_{L^2(I)} := \int_I |f(t)|^2 \, \mathrm{d}t$
- $\mathrm{L}^1$-norm $||f||_{L^1(I)} := \int_I |f(t)| \, \mathrm{d}t$

Now let $L : X \to Y$ be a linear problem map between two normed spaces, the data space $X$ (with norm $|| \cdot ||_X$) and the result space $Y$ (with norm $|| \cdot ||_Y$). Thanks to linearity, pertubations of the result $y := L(x)$ for the input $x \in X$ can be expressed as follows:

$$L(x + \delta x) = L(x) + L(\delta x) = y + L(\delta x).$$

Hence, the sensitivity can be measured by the ***operator norm***

$$||L||_{X \to Y} := \sup_{\delta x \in X \setminus \{0\}} \frac{||L(\delta x)||_Y}{||\delta x||_X}.$$

**Lemma:** Given a mesh $\mathcal{T} \subset \mathbb{R}$ with generalized Lagrange polynomials $L_i$, $i = 0, ..., n$, and fixed $I \subset \mathbb{R}$, the norm of the interpolation operator satisfies

$$||I_\mathcal{T}||_{\infty \to \infty} := \sup_{y \in \mathbb{R}^{n+1} \setminus \{0\}} \frac{||I_\mathcal{T}(y)||_{L^\infty(I)}}{||y||_\infty} = \Big|\Big| \sum_{i=0}^n |L_i| \Big|\Big|_{L^\infty(I)},$$
$$||I_\mathcal{T}||_{2 \to 2} := \sup_{y \in \mathbb{R}^{n+1} \setminus \{0\}} \frac{||I_\mathcal{T}(y)||_{L^2(I)}}{||y||_2} \leq \Big( \sum_{i=0}^n ||L_i||^2_{L^2(I)} \Big)^{\frac{1}{2}}.$$

---

**Definition:** We define the **Lebesgue constant** of $\mathcal{T}$ as follows:

$$\lambda_\mathcal{T} := \Big|\Big| \sum_{i=0}^n |L_i| \Big|\Big|_{L^\infty(I)} = ||I_\mathcal{T}||_{\infty \to \infty}.$$

---

## 5.3 Shape-Preserving Interpolation

When reconstructing a quantitative dependence of quantities from measurements, first principles from physics often stipulate qualitative constraints, which translate into *shape properties* of the function $f$, e.g. when modelling the material law for a gas:

→ $t_i$ pressure values, $y_i$ densities $\Rightarrow$ $f$ positive and monotonic

Notation: given data is $(t_i,\, y_i) \in \mathbb{R}^2,\, i = 0, ..., n,\, n \in \mathbb{N},\, t_0 < t_1 < ... < t_n$.

### 5.3.1 Shape Properties of Functions and Data

**Definition:** The data $(t_i,\, y_i)$ are called ***monotonic*** when $y_i \geq y_{i-1}$ or $y_i \leq y_{i-1}$ for $i = 0, ..., n$.

**Definition:** The data $\{(t_i,\, y_i)\}_{i=0}^n$ are called ***convex (concave)*** *if*

$$\Delta_j \leq_{(\geq)} \Delta_{j+1},\, j = 1, ..., n-1,\;\; \Delta_j = \frac{y_j - y_{j-1}}{t_j - t_{j-1}},\, j = 1, ..., n.$$

### 5.3.2 Piecewise Linear Interpolation

There is a very simple method of achieving perfect shape preservation by means of a linear interpolation operator into the space of continuous functions. Given the following data: $(t_i,\, y_i) \in \mathbb{R}^2,\, i = 0, ..., n,\, n \in \mathbb{N},\, t_0 < t_1 < ... < t_n$.

Then the ***piecewise linear interpolant*** $s : [t_0,\, t_n] \to \mathbb{R}$ is defined as

$$s(t) = \frac{(t_{i+1} - t)y_i + (t - t_i)y_{i+1}}{t_{i+1} - t_i} \text{ for } t \in [t_i,\, t_{i+1}].$$

The piecewise linear interpolant is also called a *polygonal curve*. It is continuous and consists of $n$ line segments.

**Theorem:** Let $s \in C([t_0,\, t_n])$ be the piecewise linear interpolant of $(t_i,\, y_i) \in \mathbb{R}^2,\, i = 0, ..., n$, for every subinterval $I = [t_j,\, t_k] \subset [t_0,\, t_n]$:

- if $(t_i,\, y_i)|_I$ are positive/negative $\Rightarrow s|_I$ is positive/negative
- if $(t_i,\, y_i)|_I$ are monotonic $\Rightarrow s|_I$ is monotonic
- if $(t_i,\, y_i)|_I$ are convex/concave $\Rightarrow s|_I$ is convex/concave

### 5.3.3 Cubic Hermite Interpolation

Aim: local shape-preserving (linear) interpolation operator that fixes short-coming of piecwise linearinterpolation by ensuring $C^1$-smoothness of the interpolant.

notation: $C^1([a,\, b])$ = space of ***continuously differentiable*** functions $[a,\, b] \to \mathbb{R}$.

#### 5.3.3.1 Definition and Algorithms

Given: mesh points $(t_i,\, y_i) \in \mathbb{R}^2,\, i = 0, ..., n,\, t_0 < t_1 < \cdots < t_n$.

Goal: build function $f \in C^1([t_0,\, t_n])$ satisfying the interpolant conditions $f(t_i) = y_i,\, i = 0, ..., n$.

**Definition:** Given data points $(t_i,\, y_i) \in \mathbb{R} \times \mathbb{R},\, j = 0, ..., n$ with pairwise distinct ordered notes $t_j$, and ***slopes*** $c_j \in \mathbb{R}$, the piecwise ***cubic Hermite interpolants*** $s : [t_0,\, t_n] \to \mathbb{R}$ is defined by the requirements

$$s_{|[t_{i-1},\, t_i]} \in \mathcal{P}_3,\, i = 1, ..., n,\, s(t_i) = y_i,\, s'(t_i) = c_i,\, i = 0, ..., n.$$

#### 5.3.3.2 Local Monotonicity-Preserving Hermite Interpolation

*whatever*

**Theorem:** If, for fixed node set $\{t_j\}_{j=0}^n,\, n \geq 2$, an interpolation scheme $I : \mathbb{R}^{n+1} \to C^1(I)$ is *linear* as a mapping from data values to continuous functions on the interval covered by the nodes, and *monotonicity*

*preserving,* then $I(y)'(t_j) = 0$ for all $y \in \mathbb{R}^{n+1}$ and $j = 1, ..., n-1$.

## 5.4 Splines

**Definition:** Given an interval $I := [a, b] \subset \mathbb{R}$ and a ***knot sequence*** $\mathcal{M} := \{a = t_0 < t_1 < ... < t_n = b\}$, $n \in \mathbb{N}$, the vector space $\mathcal{S}_{d,\mathcal{M}}$ of the ***spline functions*** of degree $d$ (or order $d+1$) is defined by

$$\mathcal{S}_{d,\mathcal{M}} := \{s \in C^{d-1}(I) : s_j := s_{|[t_{j-1}, t_j]} \in \mathcal{P}_d \, \forall j = 1, ..., n\}.$$

The ***dimension of a splice space*** can be found by a *counting argument*: We count the number of "degrees of freedom" possessed by a $\mathcal{M}$-piecewise polynomial of degree $d$, and subtract the number of *linear constraints:*

$$\dim \mathcal{S}_{d,\mathcal{M}} = n \cdot \dim \mathcal{P}_d - \#\{C^{d-1} \text{ continuity constraints}\} = n \cdot (d+1) - (n-1) \cdot d = n + d.$$

### 5.4.1 Cubic-Spline Interpolation

Task: Given a mesh $\mathcal{M} := \{t_0 < t_1 < \cdots < t_n\}$, $n \in \mathbb{N}$, "find" a cubic spline $s \in \mathcal{S}_{3,\mathcal{M}}$ that compiles with the interpolation conditions

$$s(t_j) = y_j, \; j = 0, ..., n.$$

*whatever*

To saturate the remainign two degree of freedom the following three approaches are popular:

1. Complete cubic spline interpolation: $s'(t_0) = c_0, \; s'(t_n) = c_n$ prescribed
2. Natural cubic spline interpolation: $s''(t_0) = s''(t_n) = 0$
3. Periodic cubic spline interpolation: $s'(t_0) = s'(t_n), \; s''(t_0) = s''(t_n)$

### 5.4.2 Structural Properties of Cubic Spline Interpolation

For a function $f : [a, b] \to \mathbb{R}, \; f \in C^2([a, b])$, the term

$$E_{\text{bd}}(f) := \frac{1}{2} \int_a^b |f''(t)|^2 \, \mathrm{d}t,$$

models the elastic bending energy of a rod, whose shape is described by the graph of $f$. We will show the cubic spline interpolants have minimal bending energy among all $C^2$-smooth interpolating functions.

Given: mesh $\mathcal{M} := \{a = t_0 < t_1 < ... < t_n = b\}$ of $[a, b]$ with knots $t_j$

Set $s \in \mathcal{S}_{3,\mathcal{M}}$ := natural cubic spline interpolant of data points $(t_i, y_i) \in \mathbb{R}^2$, $i = 0, ..., n$.

**Theorem:** The natural cubic spline interpolant $s$ minimizes the elastic curve energy among all interpolating functions in $C^2([a, b])$, that is

$$E_{\text{bd}}(S) \leq E_{\text{bd}}(f) \, \forall f \in C^2([a, b]), \; f(t_i) = y_i, \; i = 0, .., n.$$

### 5.4.3 Shape Preserving Spline Interpolation

This section presents a non-linear ***quadratic spline*** based interpolation scheme that manages to preserve both monotonicity and curvature of data even in a local sense.

Given: data points $(t_i, y_i) \in \mathbb{R}^2$, $i = 0, ..., n$ assume ordering $t_0 < t_1 < ... < t_n$.

Sought: extended knot set $\mathcal{M} \subset [t_0, t_n]$ and an interpolating ***quadratic spline function*** $s \in \mathcal{S}_{2,\mathcal{M}}, \; s(t_i) = y_i, \; i = 0, ..., n$ that preserves the shape of the data.

We proceed in 4 steps:

1. Shape preserving choice of slopse $c_j$, $j = 0, ..., n$

   Recall: We fix the slopes $c_i$ in the nodes using the harmonci mean of data slopse $\Delta_j$, the final interpolant will be tangent to these segments in the points $(t_i,\, t_j)$. If $(t_i,\, y_i)$ is a local maximum or minimu of the data, $c_j$ is set to zero.

   $$\text{Limiter: } c_i := \frac{2}{\Delta_i^{-1} + \Delta_{i+1}^{-1}}, \text{ if } \operatorname{sign}(\Delta_i) = \operatorname{sign}(\Delta_{i+1}), \;\; 0 \text{ otherwise}$$

   and $c_0 := 2\Delta_1 - c_1$, $c_n := 2\Delta_n - c_{n-1}$, where $\Delta_j = \frac{y_j - y_{j-1}}{t_j - t_{j-1}}$.

2. Choice of extra knots $p_i \in ]t_{i-1},\, t_i[$, $i = 1, ..., n$ :

   Let $T_i$ be the unique straight line through $(t_i,\, y_i)$ with slope $c_i$. If the intersection of $T_{i-1}$ and $T_i$ is non-empty and has a $t$-coordinate $\in ]t_{i-1},\, t_i[$,

   - then $p_i := t$-coordinate of $T_{i-1} \cap T_i$,
   - otheriwse $p_i = \frac{1}{2}(t_{i-1} + t_i)$.

   These points will be used to build the knot set for the final ***quadratic spline:***

   $$\mathcal{M} = \{t_0 < p_1 \leq t_1 < p_2 \leq ... < p_n \leq t_n\}.$$

3. Set $l$ = linear spline (polygon) of the knot set $\mathcal{M}'$ (middle points of $\mathcal{M}$)

   $$\mathcal{M}' = \{t_0 < \frac{1}{2}(t_0 + p_1) < \frac{1}{2}(p_1 + t_1) < \frac{1}{2}(t_1 + p_2) < ... < \frac{1}{2}(p_n + t_n) < t_n\},$$

   with $l(t_i) = y_i$, $l'(t_i) = c_i$.

4. Local quadratic approximation / interpolation of $l$:

   **Lemma:** If $g$ is a linear spline through the three points

   $$(a,\, y_a),\, (\frac{1}{2}(a+b),\, w),\, (b,\, y_b) \text{ with } a < b,\, y_a, y_b, w \in \mathbb{R},$$

   then the parabola

   $$p(t) := (y_a(b-t)^2 + 2w(t-a)(b-t) + y_b(t-a)^2)/(b-a)^2,\, a \leq t \leq b,$$

   satisfies

   1. $p(a) = y_a$, $p(b) = y_b$, $p'(a) = g'(a)$, $p'(b) = g'(b)$,
   2. $g$ monotonic increasing / decreasing $\Rightarrow$ $p$ monotonic increasing / decreasing,
   3. $g$ convex / concave $\Rightarrow$ $p$ convex / concave

## 5.6 Trigonometric Interpolation

We consider time series data $(t_i,\, y_i)$, $i = 0, ..., n$, $t_i, y_i \in \mathbb{R}$, obtained by sampling a time-dependent scalar physical quantity $t \to \phi(t)$. We know that $\phi$ is a ***T-periodic*** function with period $T > 0$, that is $\phi(t) = \phi(t + T)$ for all $t \in \mathbb{R}$. In the spirit of shape preservation an interpolant $f$ of the time series should also be $T$-periodic: $f(t + T) = f(t)$ for all $t \in \mathbb{R}$.

**Assumption:** We assume the period $T > 0$ to be known and $t_i \in [0,\, T[$ for all interpolation nodes $t_i$, $i = 0, ..., n$.

In the sequel, for the case of simplicity, we consider only $T = 1$.

Task: Given $T > 0$ and data points $(t_i,\, y_i)$, $y_i \in \mathbb{K}$, $t_i \in [0,\, T[$, find a ***T-periodic*** function $f : \mathbb{R} \to \mathbb{K}$ (the interpolant), $f(t + T) = f(t)\, \forall t \in \mathbb{R}$, that satisfies the ***interpolation conditions***

$$f(t_i) = y_i,\, i = 0, ..., n.$$

### 5.6.1 Trigonometric Polynomials

The most fundamental periodic functions are derived from the trigonometric functions $\sin$ and $\cos$ and dilations of them. A ***dilation*** of a function $t \to \phi(t)$ is a function of the form $t \to \phi(ct)$ with some $c > 0$.

---

**Definition:** The vector space of $1$-periodic ***trigonometric polynomials*** of degree $2n$, $n \in \mathbb{N}$, is given by

$$\mathcal{P}_{2n}^T := \mathrm{Span}\{t \to \cos(2\pi jt),\, t \to \sin(2\pi jt)\}_{j=0}^n \subset C^\infty(\mathbb{R}).$$

---

We can rewrite $q \in \mathcal{P}_{2n}^T$ given in the form

$$q(t) = \alpha_0 + \sum_{j=1} \alpha_j \cos(2\pi jt) + \beta_j \sin(2\pi jt), \;\; \alpha_j, \beta_j \in \mathbb{R}.$$

Notation: $\mathbb{S}^1 := \{z \in \mathbb{C} : |z| = 1\}$ is the unit circle in the complex plane.

### 5.6.2 Reduction to Lagrange Interpolation

*whatever*

### 5.6.3 Equidistant Trigonometric Interpolation

Often timeseries data for a time-periodic quantity are measured with a constant rhythm over the entire period of duration $T > 0$, that is, $t_j = j\Delta t$, $\Delta t = \frac{T}{n+1}$, $j = 0, ..., n$. In this case, the formulas for computing coefficients of the interpolating trigonometric polynomial become special versions of the ***discrete Fourier transfrom (DFT).*** Efficient implementation can thus harness the speed of *FFT*.

Now we consider trigonometric interpolation in the $1$-periodic setting with $2n + 1$ *uniformly distributed* interpolation nodes $t_k = \frac{k}{2n+1}$, $k = 0, ..., 2n$, and associated data values $y_k$. Existence and uniqueness of an interpolating trigonometric polynomial $q \in \mathcal{P}_{2n}^T$, $q(t_k) = y_k$, was established earlier. We rely on the following relation ship:

$$q \in \mathcal{P}_{2n}^T \Rightarrow q(t) = e^{-2\pi int} \cdot p(e^{2\pi t}) \text{ with } p(z) = \sum_{j=0}^{2n} \gamma_j z^j \in \mathcal{P}_{2n},$$

to arrive at the following $(2n + 1) \times (2n + 1)$ linear system of equations for computing the unknown coefficients $\gamma_j$:

$$\overline{\mathrm{F}}_{2n+1} c = b,\, c = [\gamma_0, ..., \gamma_{2n}]^T \Rightarrow c = \frac{1}{2n+1} \mathrm{F}_{2n+1} b.$$

## 5.7 Least Squares Data Fitting

The most general task of multidimensional, vector-valued ***least squares data fitting*** can be described as follows:

---

**Least square data fitting**

Given: data points $(t_i,\, y_i)$, $i \in \{1, ..., m\}$, $m \in \mathbb{N}$, $t_i \in D \subset \mathbb{R}^k$, $y_i \in \mathbb{R}^d$, $d \in \mathbb{N}$.

Objective: Find a continuous function $f : D \to \mathbb{R}^d$ in some set $S \subset C^0(D)$ of *admissible functions* satisfying

$$f \in \operatorname{argmin}_{g \in S} \sum_{i=1}^{m} ||g(t_i) - y_i||_2^2.$$

Such a function $f$ is called a ***best least squares fit*** for the data in $S$.

---

Consider a special variant of the general least squares data fitting problem: The set $S$ of admissible continuous functions is now chosen as a *finite-dimensional vector space* $V_n \subset C^0(D)$, $\dim V_n = n \in \mathbb{N}$.

Choose a basis of $V_n$, $V_n = \operatorname{Span}\{b_1, ..., b_n\}$, $b_j : D \to \mathbb{R}^d$ continuous.

→ The best least squares fit $f \in V_n$ can be represented by a finite linear combination of the basis functions $b_j$:

$$f(t) = \sum_{j=1}^{n} x_j b_j(t),\; x_j \in \mathbb{R}.$$

It can be furthermore be recast to the following problem:

---

**General linear least squares fitting problem**

Given: data points $(t_i,\, y_i) \in \mathbb{R}^k \times \mathbb{R}^d$, $i = 1, ..., m$ and basis functions $b_j : D \subset \mathbb{R}^k \to \mathbb{R}$, $j = 1, ..., n$, $n < m$.

Sought: coefficients $x_j \in \mathbb{R}$, $j = 1, ..., n$, such that

$$x := [x_1, ..., x_n]^T := \operatorname{argmin}_{z_j \in \mathbb{R}^d} \sum_{i=1}^{m} \left|\left| \sum_{j=1}^{n} z_j b_j(t_i) - y_i \right|\right|_2^2.$$

---

**Theorem:** The solution $[x_1, ..., x_n]^T \in \mathbb{R}^n$ of the linear least squares fitting problem is the least squares solution of the overdetermined linear system of equations

$$\begin{bmatrix} A_1 \\ \vdots \\ A_d \end{bmatrix} x = \begin{bmatrix} b_1 \\ \vdots \\ b_d \end{bmatrix},$$

with

$$A_r := \begin{bmatrix} (b_1(t_1))_r & \cdots & (b_n(t_n))_r \\ \vdots & & \vdots \\ (b_q(t_m))_r & \cdots & (b_n(t_m))_r \end{bmatrix} \in \mathbb{R}^{m,n},\; b_r := \begin{bmatrix} (y_1)_r \\ \vdots \\ (y_m)_r \end{bmatrix} \in \mathbb{R}^m,\; r = 1, ..., d.$$

**Lemma:** The scalar one-dimensional linear least squares fitting problem with $\dim V_n = n$, $V_n$ the vector space of admissible functions, has a unique solution, if and only if there are $t_{i_1}, .., t_{i_n}$ such that

$$\begin{bmatrix} b_1(t_{i_1}) & \cdots & b_n(t_{i_1}) \\ \vdots & & \vdots \\ b_1(t_{i_n}) & \cdots & b_n(t_{i_n}) \end{bmatrix} \in \mathbb{R}^{n,n} \text{ is invertible,}$$

which is *independent of the choice of basis* of $V_n$.

## 5.8 Learning outcomes

- You should understand the use of basis functions for representing functions on a computer.
- You should know the concept of an interpolation operator and what its linearity means.
- You should know the connection between the linear interpolation operators and linear systems of equations.
- You should be familiar with efficient algorithms for polynomial interpolation in different settings.
- You should know the meaning and significance of "sensitivity" in the context of interpolation.
- You should be familiar with the notions of "shape preservation" for an interpolation scheme and its different aspects (monotonicity, curvature).
- You should know the details of cubic Hermite interpolation and how to ensure that it is monotonicity preserving.
- You should know what splines are and how cubic spline interpolation with different endpoint constraints works.