Matrix unverändert: $A = QR \;\Rightarrow\; \mathrm{cond}_2(R) = \mathrm{cond}_2(A)$.

c) Realisierung des Gauß–Algorithmus in Gleitpunktarithmetik:
Fehlerschranke hängt linear ab von $\max\limits_{i,k} |l_{ik}|$.

Spaltenpivotisierung: $|l_{ik}| \le 1 \;\rightsquigarrow$ kleine Fehlerschranke

Numerische Stabilität: numerische Lösung $\tilde{x}$ erfüllt

$$(A + \delta_A)\tilde{x} = b$$

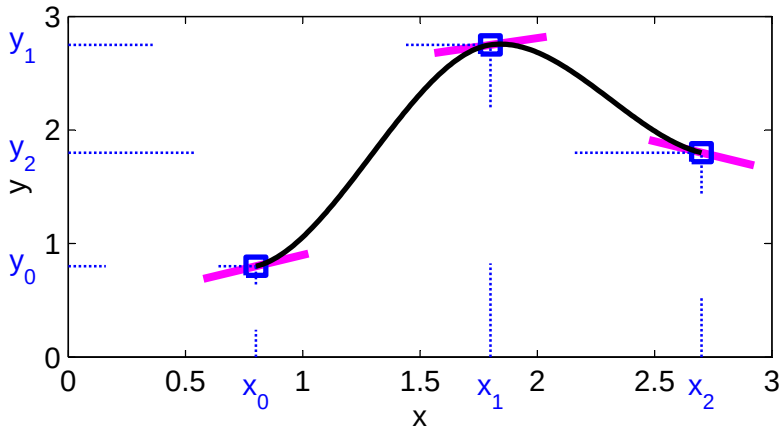
mit

$$\frac{\|\delta_A\|_\infty}{\|A\|_\infty} \lesssim 8n^3 \cdot \frac{\max_{i,j,k} |a_{ij}^{(k)}|}{\max_{i,j} |a_{ij}|}\, \varepsilon\,.$$

4 Interpolation (II)

Bemerkung 4.1 (Stückweise Hermite–Interpolation)

geg.: $r+1$ Stützstellen $x_0, x_1, \ldots, x_r$
Stützwerte (y_k, y_k'), $(k = 0, 1, \ldots, r)$



Definiert man die interpolierende Funktion Φ stückweise durch Hermite–Interpolationspolynome $\Phi\big|_{[x_{i-1},x_i]}$, $(i = 1, \ldots, r)$ mit Interpolationsbedingungen

$$\Phi(x_{i-1}) = y_{i-1}\,,\quad \Phi'(x_{i-1}) = y'_{i-1}\,,\quad \Phi(x_i) = y_i\,,\quad \Phi'(x_i) = y'_i\,,\quad (i = 1, \ldots, r)\,,$$

so ist $\Phi \in C^1[a,b]$, aber $\deg \Phi\big|_{[x_{i-1},x_i]} \le 3$.

4.1 Spline–Interpolation

Bemerkung 4.2 (Kubische Spline–Interpolation)
Kubische Splines erreichen ähnlich wie zusammengesetzte Hermite–Interpolierende eine hohe globale Glattheit, jedoch mit deutlich niedrigerem Polynomgrad:

$$s \in C^2[a,b]\,,\quad s\big|_{[x_i,x_{i+1}]} \in \Pi_3\,.$$

Splines der *Ordnung* k: $s \in C^{k-2}[a,b]\,,\ \ s\big|_{[x_i,x_{i+1}]} \in \Pi_{k-1}\,.$

Splinegitter $a = x_0 < x_1 < \ \ldots\ < x_n = b\,.$

Zum kubischen Spline s ist

$$s\big|_{[x_i,x_{i+1}]} = a_i + b_i(x-x_i) + \frac{c_i}{2}(x-x_i)^2 + \frac{d_i}{6}(x-x_i)^3\,,\ \ (\,i = 0,1,\ldots,n-1\,)$$

$\Rightarrow$ insgesamt $4n$ Parameter (a_i, b_i, c_i, d_i), $(\,i = 0,1,\ldots,n-1\,)$.

$n+1$ Interpolationsbedingungen $\ s(x_i) = y_i \ \ \Rightarrow\ \ a_i = y_i\,,\ (\,i = 0,1,\ldots,n-1\,)\,,$

$s(x_n) = y_n \ \ \Rightarrow\ \ a_{n-1} + b_{n-1}(x_n - x_{n-1}) + \frac{c_{n-1}}{2}(x_n - x_{n-1})^2 + \frac{d_{n-1}}{6}(x_n - x_{n-1})^3 = y_n =: a_n\,.$

$3(n-1)$ Stetigkeitsbedingungen

$$\begin{aligned}
s(x_{i+1}-0) = s(x_{i+1}+0)\ :\qquad & a_i + b_i h_i + \frac{c_i}{2}h_i^2 + \frac{d_i}{6}h_i^3 &&= a_{i+1}\,,\ (\,i = 0,1,\ldots,n-2\,)\\
s'(x_{i+1}-0) = s'(x_{i+1}+0)\ :\qquad & b_i + c_i h_i + \frac{d_i}{2}h_i^2 &&= b_{i+1}\,,\ (\,i = 0,1,\ldots,n-2\,)\\
s''(x_{i+1}-0) = s''(x_{i+1}+0)\,:\qquad & c_i + d_i h_i &&= c_{i+1}\,,\ (\,i = 0,1,\ldots,n-2\,)
\end{aligned}$$

mit *Schrittweiten* $h_i := x_{i+1} - x_i\,,\ (\,i = 0,1,\ldots,n-1\,)\,.$

$\Rightarrow$ insgesamt $4n-2$ lineare Bedingungen an $4n$ Parameter

Zusatzbedingungen

(i) $\ s''(x_0) = s''(x_n) = 0\ \ \ldots$ natürlicher kubischer Spline

oder

(ii) $\ s'(x_0) = y_0'\,,\ s'(x_n) = y_n\ \ \ldots$ vollständiger kubischer Spline

oder

(iii) $\ s'(x_0) = s'(x_n)\,,\ s''(x_0) = s''(x_n)\ \ \ldots$ periodischer kubischer Spline, für periodische Daten $(\,y_0 = y_n\,) \ \Rightarrow\ s(x_0) = s(x_n)\,.$

In jedem der drei Fälle ist die Splinefunktion eindeutig bestimmt.

Berechnung der Koeffizienten

$$d_i \ = \ \frac{c_{i+1} - c_i}{h_i}\,,$$

$$b_i \ = \ \frac{a_{i+1} - a_i}{h_i} - \frac{c_i}{2}h_i - \frac{d_i}{6}h_i^2 \ = \ \frac{y_{i+1} - y_i}{h_i} - \frac{2c_i + c_{i+1}}{6}h_i$$

Stetigkeitsbedingung für $s'(x)\ \ \Rightarrow\ (\,i = 0,1,\ldots,n-2\,)$

$$\frac{y_{i+1} - y_i}{h_i} - \frac{2c_i + c_{i+1}}{6}h_i + c_i h_i + \frac{c_{i+1} - c_i}{2}h_i \ = \ \frac{y_{i+2} - y_{i+1}}{h_{i+1}} - \frac{2c_{i+1} + c_{i+2}}{6}h_{i+1}$$

$$\frac{h_i}{6}c_i + \frac{h_i + h_{i+1}}{3}c_{i+1} + \frac{h_{i+1}}{6}c_{i+2} \ = \ \frac{y_{i+2} - y_{i+1}}{h_{i+1}} - \frac{y_{i+1} - y_i}{h_i}$$

Zusammen mit $c_0 = c_n = 0$ ergibt sich für den natürlichen kubischen Spline ein tridiagonales lineares Gleichungssystem der Dimension $n-1$ zur Bestimmung von $c_1, \ldots, c_{n-1}$
$\Rightarrow$ Gaußscher Algorithmus erfordert $\mathcal{O}(n)$ Rechenoperationen. Analoges Vorgehen für vollständigen und periodischen Spline.

Algorithmus 1 Bestimmung der Splinekoeffizienten.

1. $a_i := y_i\,,\ (\,i = 0, 1, \ldots, n-1\,)$
2. Berechne $c_0, c_1, \ldots, c_n$ als Lösung eines tridiagonalen Gleichungssystems.
3. Bestimme $b_i, d_i\,,\ (\,i = 0, 1, \ldots, n-1\,)\,.$

Algorithmus 2 Auswertung der Splinefunktion.

1. Bestimme Teilintervall (binäre Suche)

$\underline{i} := 0\,,\ \overline{i} := n$
`repeat`
$\quad i_* := \left[\ \dfrac{\underline{i} + \overline{i}}{2}\ \right]$
$\quad$ `if` $x \geq x_{i_*}$ `then` $\underline{i} := i_*$ `else` $\overline{i} := i_*$
`until` $\overline{i} - \underline{i} \leq 1$
$i := \underline{i}$

Einfachster Spezialfall: äquidistantes Gitter $x_i = a + ih$ mit $h = \dfrac{b-a}{n}$

$\Rightarrow \quad i := \left[\ \dfrac{x-a}{h}\ \right]$

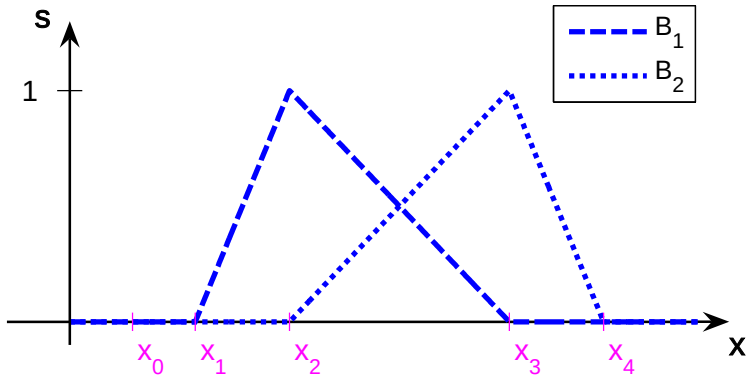
2. Splineauswertung $s(x) = a_i + (x - x_i)\big(b_i + (x - x_i)\big(\,\dfrac{1}{2}c_i + \dfrac{1}{6}d_i(x - x_i)\,\big)\big)$

Bemerkung 4.3 (B–Splines)

Idee Darstellung der Splinefunktion als Linearkombination „einfacher“ Basisfunktionen des Vektorraums der Splinefunktionen $\rightsquigarrow$ B–Splines.

Beispiel $k = 2:$ stetige, stückweise lineare Funktion

$$B_j(x) = \begin{cases} \dfrac{x - x_j}{x_{j+1} - x_j}\,, & (\,x \in [x_j, x_{j+1}]\,), \\ \dfrac{x - x_{j+2}}{x_{j+1} - x_{j+2}}\,, & (\,x \in [x_{j+1}, x_{j+2}]\,), \\ 0 & \text{sonst.} \end{cases}$$

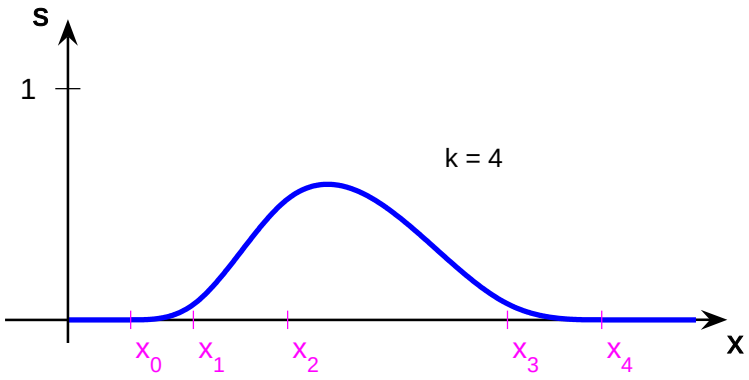


Interpolierender linearer Spline $s(x) = \sum_{j=0}^{n} y_j B_{j-1}(x)$.

allgemein

- $B_j\big|_{[x_i,x_{i+1}]} \in \Pi_{k-1}$, $(i = 0, 1, \ldots, n-1)$
- $B_j \in C^{k-2}[a, b]$
- $\sum_j B_j(x) = 1$, $(x \in [a, b])$
- $\operatorname{supp} B_j = [x_j, x_{j+k}]$, d. h. $B_j(x) = 0$, $(x \le x_j$ oder $x \ge x_{j+k})$

Beispiel Kubischer B–Spline



Bestimmung der Koeffizienten α_j des interpolierenden Splines $\sum_j \alpha_j B_j(x)$ als Lösung eines linearen Gleichungssystems der Bandbreite $k - 1$.

Satz 4.4 (Approximationseigenschaften kubischer Splines)
Gegeben sei eine Funktion $f \in C^4[a, b]$ *mit* $\max_{a \le x \le b} |f^{(4)}(x)| \le M$ *sowie ein Gitter*

$$\Delta = \{\, a = x_0 < x_1 < \ldots < x_n = b \,\}$$

mit Schrittweiten $h_i := x_{i+1} - x_i$, $(i = 0, 1, \ldots, n-1)$ *und einer Konstanten*

$$K \ge \max_{0 \le i \le n-1} h_i \;/\; \min_{0 \le i \le n-1} h_i \,.$$

Dann gibt es zum vollständigen interpolierenden kubischen Spline s_Δ Konstanten C_0, C_1, C_2 und C_3, die von Δ und K unabhängig sind und für die gilt

$$|f^{(k)}(x) - s_\Delta^{(k)}(x)| \le C_k M K \big(\max_{0 \le i \le n-1} h_i \big)^{4-k}, \quad (x \in [a,b],\ k = 0,1,2,3)$$

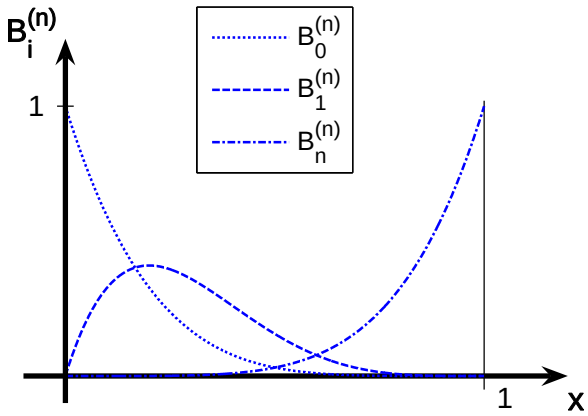
in jedem Punkt x, in dem $s_\Delta^{(k)}(x)$ definiert ist.

Beweisidee (i) $f(x_i) = s_\Delta(x_i)$ für Stützstellen x_i
(ii) Abschätzung von $|f''(x_i) - s''_\Delta(x_i)|$ durch Einsetzen von f in das Gleichungssystem aus Bemerkung 4.2
(iii) Hieraus Abschätzungen für $x \neq x_i$. ∎

Bemerkung 4.5 (Bernstein–Polynome und Bezier–Kurven)

a) Bernstein–Polynome

$$B_i^{(n)}(x) = \binom{n}{i} (1-x)^{n-i} x^i, \quad (i = 0,1,\ldots,n)$$



Eigenschaften

a) i-fache Nullstelle $x = 0$, $(n-i)$-fache Nullstelle $x = 1$

b) $B_i^{(n)}\Big|_{[0,1]} \ge 0$

c) $\displaystyle\sum_{i=0}^{n} B_i^{(n)}(x) = \sum_{i=0}^{n} \binom{n}{i} (1-x)^{n-i} x^i = \big((1-x)+x\big)^n = 1$

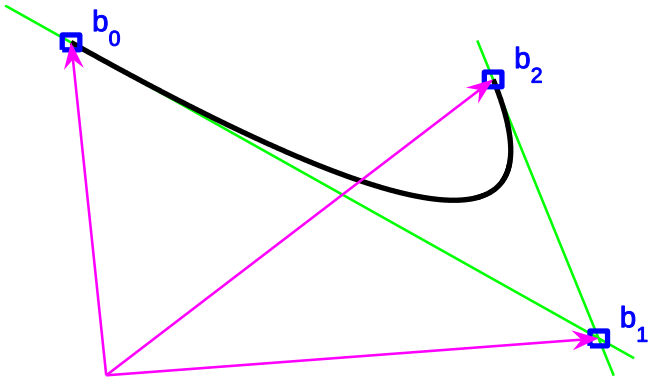
d) $B_i^{(n)}(x) = x \cdot B_{i-1}^{(n-1)}(x) - B_i^{(n-1)}(x)$

b) Bezier–Kurve

geg.: *Kontrollpunkte* $b_0, b_1, \ldots, b_n \in \mathbb{R}^k$

Bezierkurve im $\mathbb{R}^k$: $\sum_{i=0}^{n} b_i B_i^{(n)}(x)\,,\ (\,x \in [0,1]\,)\,.$

- Anfangspunkt b_0, Endpunkt b_n
- Tangente in b_0 verläuft durch b_1, denn
$$\frac{\mathrm{d}}{\mathrm{d}x} B_0^{(n)}(0) = -n\,,\quad \frac{\mathrm{d}}{\mathrm{d}x} B_1^{(n)}(0) = n\,,\quad \frac{\mathrm{d}}{\mathrm{d}x} B_i^{(n)}(0) = 0\,,\ (\,i > 1\,)$$
- $n = 2$: Tangenten in b_0 und b_2 schneiden sich in b_1
- Kurve verläuft in der konvexen Hülle des von den b_i gebildeten Polygons
  $\Rightarrow$ keine unerwünschten Oszillationen



4.2 Trigonometrische Interpolation – Schnelle Fouriertransformation

Bemerkung 4.6 (Problemstellung)
Interpolation periodischer Daten durch trigonometrische Polynome.

***N* = 2*n* + 1 ungerade**

$$T_R^N := \{\,\phi_{2n+1}(t) := \frac{a_0}{2} + \sum_{j=1}^{n}(a_j \cos jt + b_j \sin jt) \ \text{ mit } \ a_j, b_j \in \mathbb{R}\,,\ (\,j = 0, \ldots, n\,)\,\}$$

***N* = 2*n* gerade**

$$T_R^N := \{\,\phi_{2n}(t) := \frac{a_0}{2} + \sum_{j=1}^{n-1}(a_j \cos jt + b_j \sin jt) + \frac{a_n}{2}\cos nt \ \text{ mit } \ a_j, b_j \in \mathbb{R}\,,\ (\,j = 0, \ldots, n\,)\,\}$$

Komplexe Darstellung

$$T_C^N := \{\, \phi_N(t) := \sum_{j=0}^{N-1} c_j \mathrm{e}^{ijt} \,:\, c_j \in \mathbb{C}\,,\ (\, j = 0, \ldots, N-1\,)\,\}$$

Ansatzfunktionen $\{\, 1, \cos jt, \sin jt\,\}$ bzw. $\{\, \mathrm{e}^{ijt} \,:\, 0 \le j \le N-1\,\}$ sind linear unabhängig $\rightsquigarrow$ Interpolationsaufgabe zu N Stützpunkten (t_k, f_k), $(\, k = 0, 1, \ldots, N-1\,)$ eindeutig lösbar.

Typische Aufgabenstellung Digitale Signalverarbeitung, Datenerfassung im festen Takt $\Rightarrow$ äquidistante Knoten, normiert auf $t_k = k \cdot 2\pi/N$.

Bemerkung 4.7 (Trigonometrische Interpolation auf äquidistantem Gitter)
Zu $t_k = k \cdot 2\pi/N$, $(\, k = 0, 1, \ldots, N-1\,)$ sind $\omega_k := \mathrm{e}^{it_k} = \mathrm{e}^{ik\cdot 2\pi/N}$ die N-ten Einheitswurzeln.

Interpolationsbedingungen $\phi_N(t_k) = f_k$, $(\, k = 0, 1, \ldots, N-1\,)$ bestimmen ϕ_N mit

$$\begin{pmatrix} 1 & \omega_0 & \omega_0^2 & \cdots & \omega_0^{N-1} \\ 1 & \omega_1 & \omega_1^2 & \cdots & \omega_1^{N-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega_{N-1} & \omega_{N-1}^2 & \cdots & \omega_{N-1}^{N-1} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{N-1} \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_{N-1} \end{pmatrix}.$$

Bemerkung 4.8 (Komplexe und reelle trigonometrische Polynome)
Gilt für das (komplexe) trigonometrische Polynom

$$\phi_N(t) = \sum_{j=0}^{N-1} c_j \mathrm{e}^{ijt}$$

$\phi_N(t) \in \mathbb{R}$, $(\, t \in \mathbb{R}\,)$, so gilt $\phi_N \in T_R^N$ mit

$$a_j = 2\,\mathrm{Re}\, c_j = c_j + c_{N-j}\,, \quad b_j = -2\,\mathrm{Im}\, c_j = i\,(c_j - c_{N-j})\,.$$

Beweis Für die N äquidistanten Knoten $t_k = k \cdot 2\pi/N$, $(\, k = 0, 1, \ldots, N-1\,)$ gilt

$$\mathrm{e}^{-ijt_k} = \underbrace{\mathrm{e}^{iNk\cdot 2\pi/N}}_{=1} \cdot \mathrm{e}^{-ijt_k} = \mathrm{e}^{i(N-j)t_k}$$

und

$$\sum_{j=0}^{N-1} c_j \mathrm{e}^{ijt_k} = \phi_N(t_k) = \overline{\phi_N(t_k)} = \sum_{j=0}^{N-1} \overline{c_j}\, \mathrm{e}^{-ijt_k} = \sum_{j=0}^{N-1} \overline{c_j}\, \mathrm{e}^{i(N-j)t_k} = \sum_{l=1}^{N} \overline{c_{N-l}}\, \mathrm{e}^{-ilt_k}$$

$\Rightarrow\ c_j = \overline{c_{N-j}}$, da Interpolationsaufgabe eindeutig lösbar.

Wegen $\mathrm{e}^{i\cdot 0\cdot t_k} = \mathrm{e}^{i\cdot N\cdot t_k} = 1$ ist insbesondere $c_0 = \overline{c_0}$, also $c_0 \in \mathbb{R}$.

Ebenso folgt $c_n \in \mathbb{R}$ für gerade $N = 2n$.

Sei nun $N = 2n+1$, so ist

$$\begin{aligned}\phi_N(t_k) &= c_0 + \sum_{j=1}^{2n} c_j \mathrm{e}^{ijt_k} = c_0 + \sum_{j=1}^{n} c_j \mathrm{e}^{ijt_k} + \sum_{l=1}^{n} c_{N-l}\mathrm{e}^{i(N-l)t_k} \\ &= c_0 + \sum_{j=1}^{n}(c_j \mathrm{e}^{ijt_k} + \overline{c_j}\mathrm{e}^{-ijt_k}) \\ &= c_0 + 2\sum_{j=1}^{n} \mathrm{Re}\,(c_j \mathrm{e}^{ijt_k}) = c_0 + 2\sum_{j=1}^{n}(\mathrm{Re}\, c_j \cdot \cos jt_k - \mathrm{Im}\, c_j \cdot \sin jt_k)\,.\end{aligned}$$

Wegen der Eindeutigkeit des trigonometrischen Interpolationspolynoms folgt die Behauptung durch Koeffizientenvergleich. Analog für gerade $N = 2n$. ∎

Satz 4.9 (Lösung der trigonometrischen Interpolationsaufgabe)
Für äquidistante Stützstellen $t_k = k \cdot 2\pi/N$, $(k = 0, 1, \ldots, N-1)$ ist die Lösung der trigonometrischen Interpolationsaufgabe

$$\phi_N(t_k) = f_k\,, \quad (k = 0, 1, \ldots, N-1)$$

gegeben durch

$$\phi_N(t) = \sum_{j=0}^{N-1} c_j \mathrm{e}^{ijt} \quad \textit{mit} \quad c_j := \frac{1}{N}\sum_{k=0}^{N-1} f_k \omega_k^{-j}\,, \quad (j = 0, 1, \ldots, N-1)\,.$$

Beweis Wegen der Eindeutigkeit des trigonometrischen Interpolationspolynoms reicht es aus, für $l = 0, 1, \ldots, N-1$ zu zeigen

$$f_l \overset{!}{=} \sum_{j=0}^{N-1} \underbrace{\left(\frac{1}{N}\sum_{k=0}^{N-1} f_k \omega_k^{-j}\right)}_{=\,c_j} \mathrm{e}^{ijt_l} = \sum_{k=0}^{N-1} f_k \cdot \frac{1}{N}\sum_{j=0}^{N-1} \omega_k^{-j}\omega_l^j\,.$$

Nun ist

$$\sum_{j=0}^{N-1} \omega_k^{-j}\omega_l^j = \sum_{j=0}^{N-1} \omega_{l-k}^j = \begin{cases} N & \text{falls } k = l\,, \\ 0 & \text{sonst}\,, \end{cases}$$

denn

$$\frac{\omega_{l-k}-1}{\omega_{l-k}-1}\cdot \sum_{j=0}^{N-1}\omega_{l-k}^j = \frac{1}{\omega_{l-k}-1}\sum_{j=0}^{N-1}(\omega_{l-k}^{j+1} - \omega_{l-k}^j) = \frac{\omega_{l-k}^N - 1}{\omega_{l-k}-1} = 0\,,$$

falls $l \neq k$, denn ω_{l-k} ist N-te Einheitswurzel. Hieraus folgt die Behauptung. ∎

Bemerkung 4.10 (Diskrete Fourier–Transformation)
Für 2π–periodische Funktionen $f \in L^2(\mathbb{R})$ erhält man mit den Fourierkoeffizienten

$$\hat{f}(j) := \frac{1}{2\pi} \int_0^{2\pi} f(t) \mathrm{e}^{-ijt} \, \mathrm{d}t \,, \quad (\, j \in \mathbb{Z}\,)$$

die abgebrochenen Fourier–Reihen

$$f_n(t) := \sum_{j=-n}^{n} \hat{f}(j) \mathrm{e}^{ijt} \,.$$

Setzt man für $N = 2n+1$

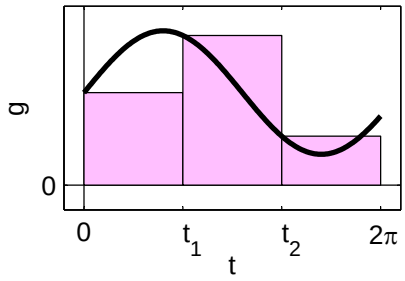
$$c_{-j} := c_{N-j} \,, \quad (\, j = 1, \ldots, n\,) \,,$$

so ist

$$\begin{aligned} \phi_N(t_k) &= \sum_{j=0}^{N-1} c_j \mathrm{e}^{ijt_k} = \sum_{j=0}^{n} c_j \mathrm{e}^{ijt_k} + \sum_{j=n+1}^{N-1} c_j \mathrm{e}^{ijt_k} \\ &= \sum_{j=0}^{n} c_j \mathrm{e}^{ijt_k} + \sum_{l=1}^{n} c_{N-l} \mathrm{e}^{i(N-l)t_k} = \sum_{j=0}^{n} c_j \mathrm{e}^{ijt_k} + \sum_{l=1}^{n} c_{-l} \mathrm{e}^{i\cdot(-l)\cdot t_k} = \sum_{j=-n}^{n} c_j \mathrm{e}^{ijt_k} \,. \end{aligned}$$

Approximiert man

$$\int_0^{2\pi} g(t) \, \mathrm{d}t \approx \frac{2\pi}{N} \sum_{k=0}^{N-1} g(t_k) \,,$$



so ergibt sich

$$\hat{f}(j) \approx \frac{1}{N} \sum_{k=0}^{N-1} f(t_k) \mathrm{e}^{-ijt_k} = \frac{1}{N} \sum_{k=0}^{N-1} f_k \omega_k^{-j} = c_j \,.$$

Wegen der Analogie zur klassischen kontinuierlichen Fourier–Transformation heißt die Abbildung

$$\mathcal{F}_N \,:\, \mathbb{C}^N \to \mathbb{C}^N \,, \quad \left(f_k\right)_{k=0}^{N-1} \mapsto \left(c_j\right)_{j=0}^{N-1}$$

aus Satz 4.9 *Diskrete Fourier–Transformation* (DFT) und die Umkehrabbildung

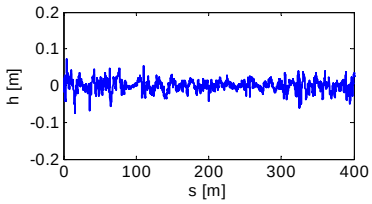
$$\mathcal{F}_N^{-1} \,:\, (c_j)_j \mapsto (f_k)_k \,, \quad f_k := \sum_{j=0}^{N-1} c_j \omega_j^k \,, \; (\, k = 0, 1, \ldots, N-1\,)$$

(Fourier–)Synthese oder *Inverse diskrete Fourier–Transformation* (IDFT).

Beispiel 4.11: Tiefpassfilter

Beispiel (Prof. Dr.-Ing. P. Pickel, U Halle, Landwirtschaftliche Fakultät)

Fahrweg eines landwirtschaftlichen Nutzfahrzeugs (vertikale Auslenkung)



Messdaten im Abstand $\Delta_s = 0.05\,\mathrm{m}$

Geschwindigkeit $v = 2.0\,\mathrm{m/s}$

$\Rightarrow \quad \Delta_t = 0.025\,\mathrm{s} \quad \Rightarrow \quad \bar{f}_{\mathrm{input}} = 40\,\mathrm{Hz}$

Programmieraufgabe
Übungsblatt 5

Abbildung 4.1: Anwendung der DFT in der Signalverarbeitung: Ausgangsdaten.

Beispiel 4.11: Tiefpassfilter (II)

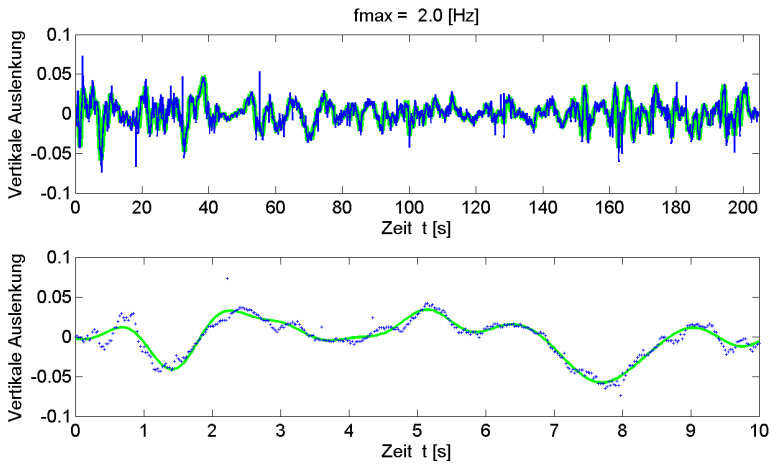


Abbildung 4.2: Anwendung der DFT in der Signalverarbeitung: $f_{\max} = 2.0\,\mathrm{Hz}$.

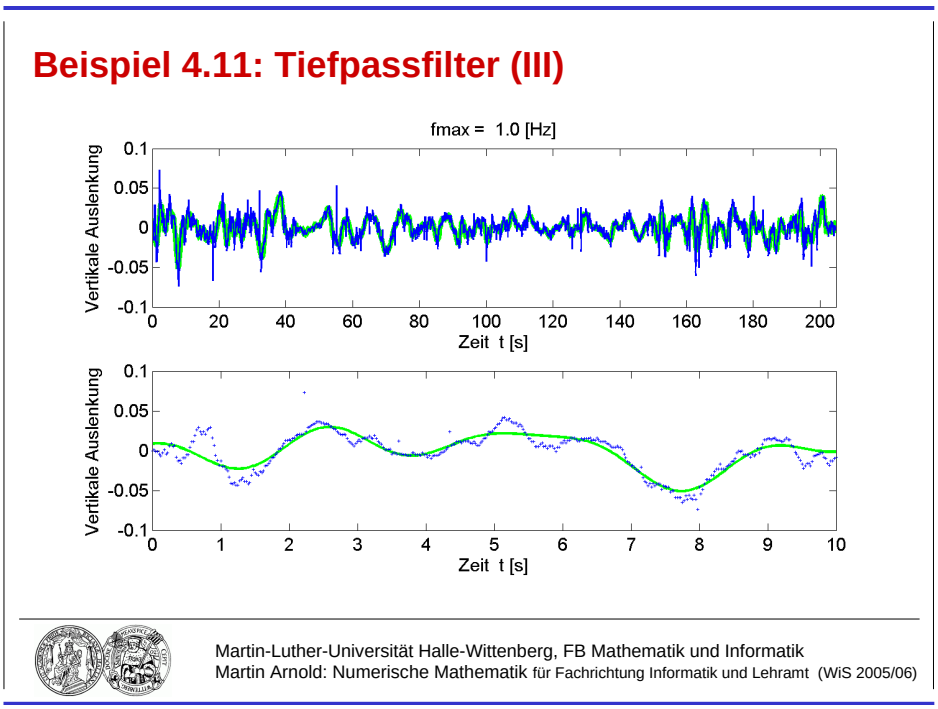


Abbildung 4.3: Anwendung der DFT in der Signalverarbeitung: $f_{\max} = 1.0\,\mathrm{Hz}$.

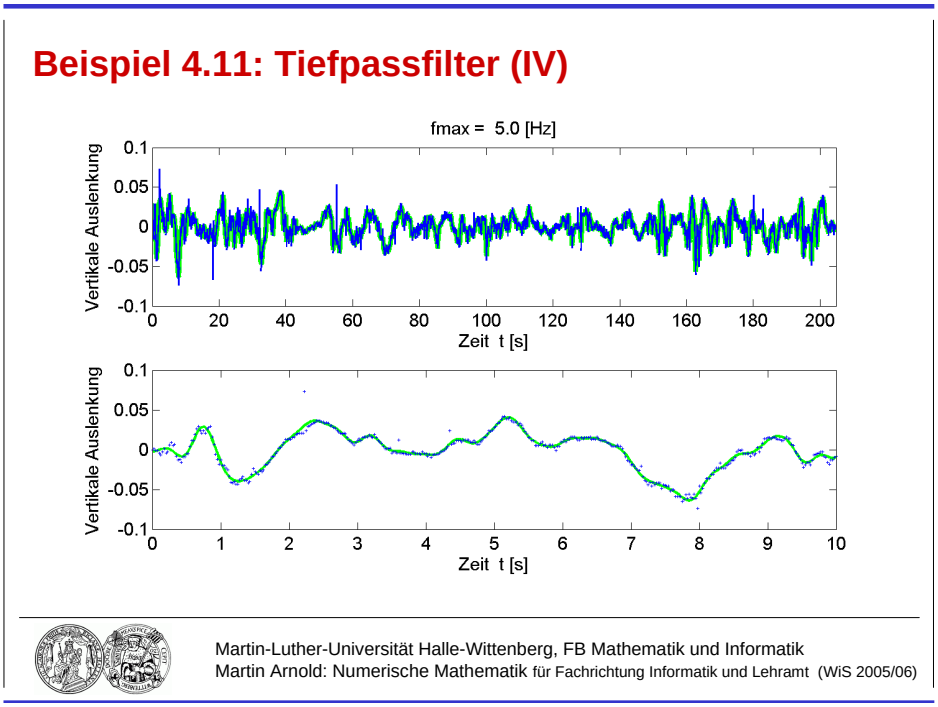


Abbildung 4.4: Anwendung der DFT in der Signalverarbeitung: $f_{\max} = 5.0\,\mathrm{Hz}$.

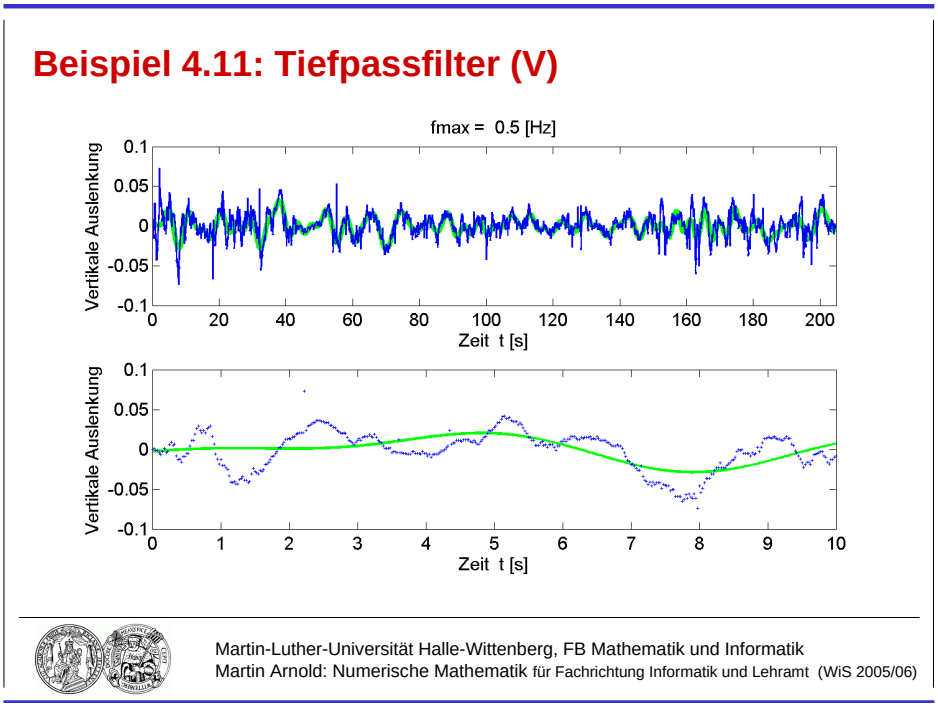


Abbildung 4.5: Anwendung der DFT in der Signalverarbeitung: $f_{\max} = 0.5\,\mathrm{Hz}$.

Beispiel 4.11 (Tiefpassfilter)

Anwendung der DFT in der Signalverarbeitung

Elimination hochfrequenter Anteile im Signal, die häufig durch Messfehler verfälscht („Messrauschen“) und darüberhinaus für die praktische Anwendung oft nicht relevant sind $\Rightarrow$ „*Tiefpassfilter*“.

praktisch Berücksichtige in $\phi_N(t)$ nur die Terme $a_j \cos jt$ und $b_j \sin jt$, die zu den ersten $j_{\max} \ll n$ Frequenzen gehören ($j_{\max}$ ist vom Anwender vorzugeben). Für $N = 2n+1$ erhält man

$$\tilde{f}_k := c_0 + \sum_{j=1}^{j_{\max}} c_j \omega_j^k + \sum_{j=1}^{j_{\max}} c_{N-j} \omega_{N-j}^k = c_0 + \sum_{j=1}^{j_{\max}} (c_j \omega_j^k + c_{N-j} \overline{\omega_j}^k), \quad (k = 0, 1, \ldots, N-1).$$

Beispiel Rauigkeit einer Feldoberfläche

(Originaldaten von Prof. Dr.-Ing. P. Pickel, Institut für Agrartechnik und Landeskultur).

Die Messdaten für die Höhe $u(x)$ einer Feldoberfläche am Ort x liegen im Abstand von $\Delta_x = 0.05\,\mathrm{m}$ vor (Abb. 4.1). Bei konstanter Fahrgeschwindigkeit $v = 2.0\,\mathrm{m/s}$ entspricht dies einer Frequenz von $1\,/\,(\,0.05\,\mathrm{m}\,/\,2.0\,\mathrm{m/s}\,) = 40\,\mathrm{Hz}$. Die Abb. 4.2–4.5 zeigen für verschiedene maximale Frequenzen $f_{\max}$ den Vergleich zwischen gefilterten Daten und Originaldaten. Die Elimination der hochfrequenten Anteile in $u(t)$ ist deutlich erkennbar.

Bemerkung 4.12 (Schnelle Fourier–Transformation)
engl.: Fast Fourier Transform(ation) (FFT)

Problem Standard–Algorithmus zur Auswertung von $\mathcal{F}_N$ oder $\mathcal{F}_N^{-1}$ würde $\mathcal{O}(N^2)$ Rechenoperationen erfordern (Matrix–Vektor–Multiplikation).

Cooley–Tuckey (1965) Sei $N = 2M$ gerade und $\omega = \mathrm{e}^{i\frac{2\pi}{N}}$ oder $\omega = \mathrm{e}^{-i\frac{2\pi}{N}}$.
Dann gilt für

$$\alpha_j = \sum_{k=0}^{N-1} f_k \omega^{kj}\,, \quad (\, j = 0, 1, \ldots, N-1\,)$$

$$\alpha_{2l} = \sum_{k=0}^{M-1} g_k \xi^{kl}\,, \qquad \alpha_{2l+1} = \sum_{k=0}^{M-1} h_k \xi^{kl}\,, \quad (\, l = 0, 1, \ldots, M-1\,)$$

mit $M := N/2$, $\xi := \omega^2$ und

$$g_k := f_k + f_{k+M}\,, \qquad h_k := (f_k - f_{k+M})\,\omega^k\,.$$

Mit $2M$ Additionen und $2M$ Multiplikationen ($\omega^k \to \omega^{k+1} = \omega \cdot \omega^k$, $h_k = (\ldots) \cdot \omega^k$) wird die Berechnung von N Summen der Länge N zurückgeführt auf $2M = N$ Summen der Länge $M = N/2$.

Beweis

$$\alpha_{2l} = \sum_{k=0}^{N-1} f_k \omega^{k\cdot 2l} = \sum_{k=0}^{\frac{N}{2}-1} \left(f_k \omega^{2kl} + f_{k+\frac{N}{2}} \omega^{2(k+\frac{N}{2})l} \right) = \sum_{k=0}^{M-1} (f_k + f_{k+M})\,\omega^{2kl}$$

$$\alpha_{2l+1} = \sum_{k=0}^{N-1} f_k \omega^{(2l+1)k} = \sum_{k=0}^{\frac{N}{2}-1} \left(f_k \omega^{2kl+k} + f_{k+\frac{N}{2}} \omega^{(2l+1)(k+\frac{N}{2})} \right) = \sum_{k=0}^{M-1} (f_k - f_{k+\frac{N}{2}})\,\omega^k \cdot \omega^{2kl}$$

Rekursive Anwendung besonders einfach für $N = 2^p$ $\Rightarrow$ Aufwand zur Berechnung von $\alpha_0, \alpha_1, \ldots, \alpha_{N-1}$ (Analyse oder Synthese) beträgt $2N \log_2 N$ Multiplikationen.

Algorithmus 4.13 (Schnelle Fourier–Transformation)
Sei $N = 2^p$ und $\omega = \mathrm{e}^{i\frac{2\pi}{N}}$ oder $\omega = \mathrm{e}^{-i\frac{2\pi}{N}}$.

Eingabe: $f_0, f_1, \ldots, f_{N-1} \in \mathbb{C}$

Ausgabe: $\alpha_0, \alpha_1, \ldots, \alpha_{N-1} \in \mathbb{C}$ mit $\alpha_j := \sum_{k=0}^{N-1} f_k \omega^{kj}$.

```
N_red := N ;  z := ω
while N_red > 1 do
  | M_red := N_red/2
  | for j = 0 : (N/N_red − 1)
  |   | l := jN_red
  |   | for k = 0 : M_red − 1
  |   |   | a := f_{l+k} + f_{l+k+M_red}
  |   |   | f_{l+k+M_red} := (f_{l+k} − f_{l+k+M_red}) z^k
  |   |   | f_{l+k} := a
  | N_red := M_red ;  z := z^2
for k = 0 : N − 1
  | α_{σ(k)} := f_k
```

Vertauschung der Komponenten von α bestimmt durch Permutation $\sigma(k)$:

$$\sigma\big(\sum_{j=0}^{N-1} a_j 2^j\big) := \sum_{j=0}^{N-1} a_{N-j} 2^j \quad \text{mit} \quad a_0, a_1, \ldots, a_{N-1} \in \{\, 0,\, 1 \,\}\,.$$

$\rightsquigarrow$ „bit reversal“, einfache Implementierung durch Bitmanipulationen

- Typisches Beispiel eines „divide–and–conquer“–Algorithmus,
- gut parallelisierbar,
- in Signalprozessoren hardwaremäßig verfügbar.