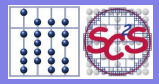


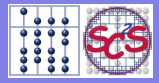
2. Interpolation

Die Lücken der Diskretisierung schließen ...

2.1. Vorbemerkungen

Die Approximationsaufgabe

- **Aufgabe**: Eine (ganz oder partiell unbekannte oder einfach nur komplizierte) Funktion $f(x)$ ist durch ein einfach zu konstruierendes und zu bearbeitendes (auswerten, differenzieren, integrieren) $p(x)$ **anzunähern**. $p(x)$ heißt **Approximant**.
- Dabei soll $f - p$ entweder punktweise oder in einer mittelnden Norm (Fehlerquadratsumme oder Euklidische Norm, z.B.) eine bestimmte Toleranz nicht übersteigen oder bestimmte Designkriterien erfüllen.
- **Beispiele**:
  - Funktionen wie die Exponentialfunktion oder die trigonometrischen Funktionen werden in Rechnern durch extrem schnell konvergierende Reihen aus geeigneten Polynomen bis auf Rechengenauigkeit approximiert und damit berechnet. Die bekannten Taylor-Reihen eignen sich hierfür i.A. nicht: Sie konvergieren nicht schnell genug, d.h., es werden zu viele Terme benötigt, bis die erforderliche Genauigkeit erreicht ist. Die Konstruktion solcher Reihen (bzw. Familien von Funktionen, oftmals orthogonale Polynome) ist eine Aufgabe der **Approximationstheorie**.



Chebyshev-Polynome:

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ T_{k+1}(x) &= 2x \cdot T_k(x) - T_{k-1}(x) \end{aligned}$$

$$a_0 = \frac{1}{\pi}\int_{-1}^{1} \frac{f(x)}{\sqrt{1-x^2}}\mathrm{d}x, \qquad a_k = \frac{2}{\pi}\int_{-1}^{1} \frac{f(x)T_k(x)}{\sqrt{1-x^2}}\mathrm{d}x, \qquad k = 1, 2, \ldots$$

Chebyshev-Approximation:

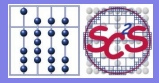
$$f(x) := \sin(\pi x) \approx \sum_{k=0}^{n} a_k \cdot T_k(x)$$

Tabelle:

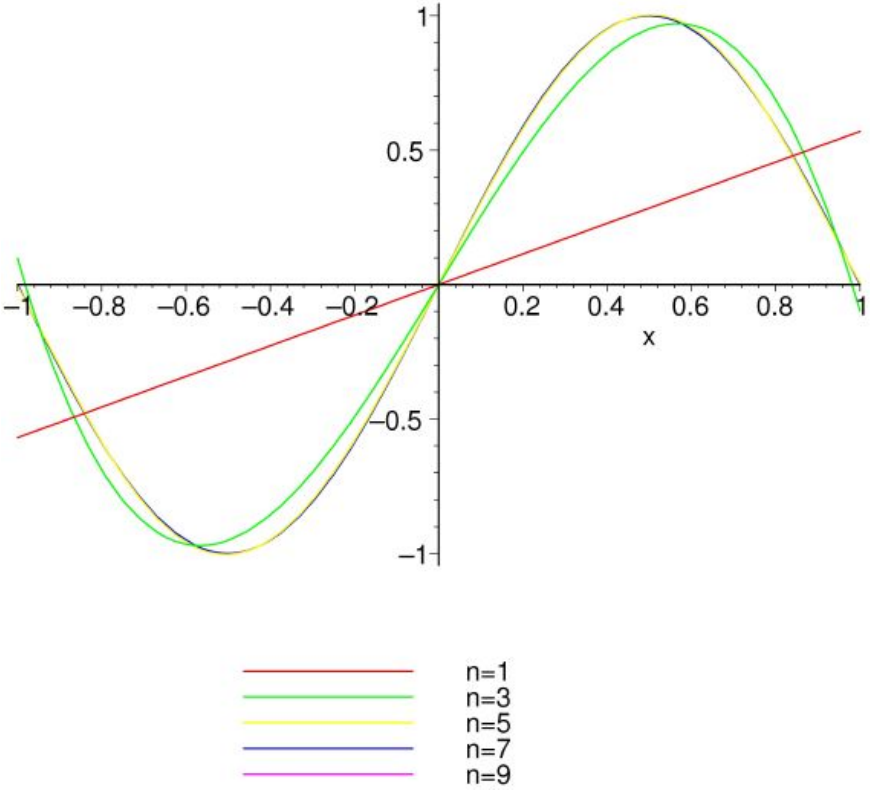
$n = 0 : 0$
$n = 1, 2 : \ 0.592306858x$
$n = 3, 4 : \ 2.569980700x - 2.667666686x^3$
$n = 5, 6 : \ 3.091392515x - 4.753313948x^3 + 1.668517810x^5$
$n = 7, 8 : \ 3.139276852x - 5.136388641x^3 + 2.434667195x^5 - 0.4377996486x^7$
$n = 9 : \ 3.141527662x - 5.166399408x^3 + 2.5427059x^5 - 0.5818513224x^7 + 0.6402296612x^9$

Taylor-Entwicklung:

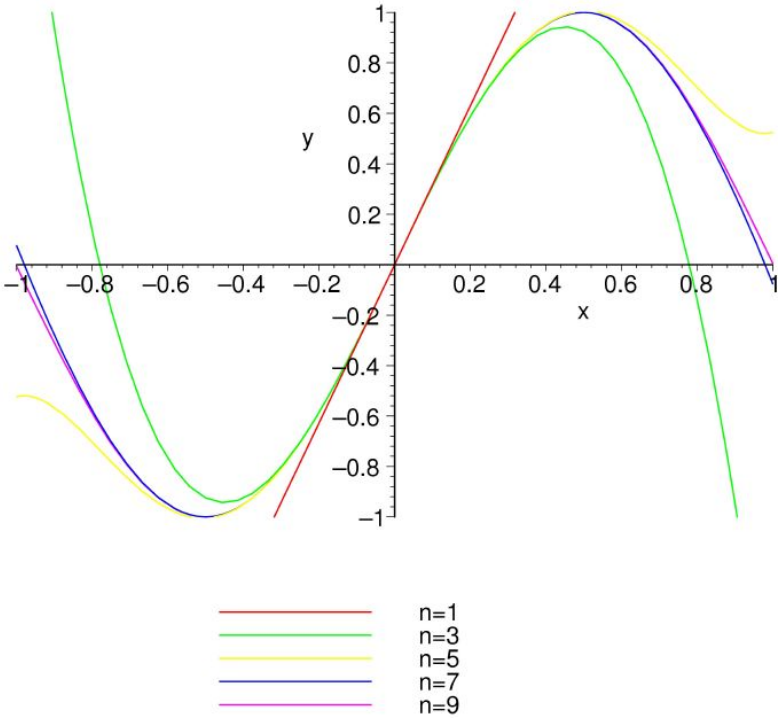
$$f(x) \approx \pi x - \frac{\pi^3 x^3}{6} + \frac{\pi^5 x^5}{120} - \frac{\pi^7 x^7}{5040} + \frac{\pi^9 x^9}{362880} - \ldots$$



Chebyshev-Polynome



Taylor-Polynome

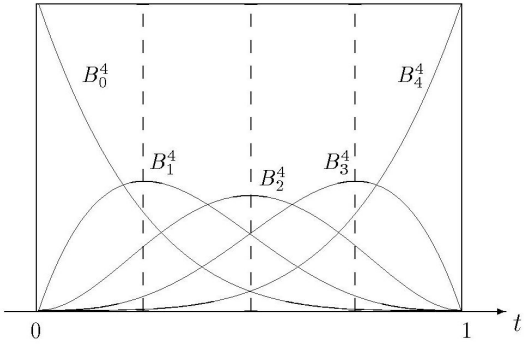


– Im CAGD und in der Computergrafik ist man an einer leicht und intuitiv steuerbaren Darstellung gekrümmter Kurven und Flächen interessiert, deren Verlauf grob durch so genannte Kontrollpunkte vorgegeben wird. Eine berühmte Klasse solcher **Freiformkurven** bzw. **Freiformflächen** sind die **Bézierkurven** bzw. **Bézierflächen**. Erstere sind definiert durch

$$X(t) := \sum_{i=0}^{n} b_i \cdot B_i^n(t), \qquad b_i \in \mathbb{R}^d, \qquad B_i^n(t) := \binom{n}{i}(1-t)^{n-i}t^i,$$

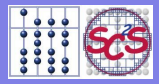
$$i = 0, ..., n,$$

mit d- dimensionalen Kontrollpunkten b_i und **Bernsteinpolynomen** $B_i^n(t)$.



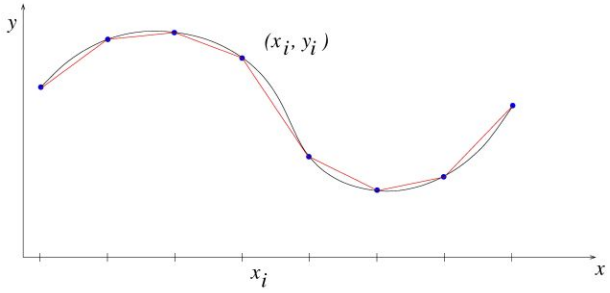
Die Interpolationsaufgabe

- Die **Interpolation** oder **Zwischenwertberechnung** ist eine zentrale Aufgabenstellung der Numerik.
- **Aufgabe**: Spezialfall der Approximation, bei dem an bestimmten Stellen die Werte von f und p übereinstimmen müssen. Der Approximant wird dabei zum **Interpolanten**.
- Oft gibt es f gar nicht explizit, sondern es werden nur diskrete Punkte (x_i, y_i) vorgegeben, wobei dann $p(x_i) = y_i$ erfüllt werden muss. Hierbei können die y_i auch Vektoren sein – dann werden beispielsweise Punkte im Raum interpoliert.
- **Beispiele**:
  - Vorgegebene Mess- oder Kontrollpunkte sind durch eine gekrümmte Kurve (in 2D) bzw. Fläche (in 3D) zu verbinden, z.B. beim Entwurf einer Automobilkarosserie.
  - Die **lineare Interpolation** wird oft intuitiv verwendet – ob gerechtfertigt oder nicht: Ein Programm benötigt bei einer Eingabe der Größe n zwei Minuten, bei der Eingabe $2n$ drei Minuten. Wie lange müssen wir bei einer Eingabe von $1.5n$ auf das Resultat warten?

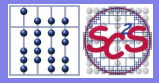


Notation zur Interpolationsaufgabe

- Aus Gründen der Einfachheit beschränken wir uns im Folgenden auf das Eindimensionale. Alles Vorgestellte kann auf den höherdimensionalen Fall verallgemeinert werden.
- Die Abszissen x_i, $i = 0, 1, ..., n$, an denen der Interpolant „festgemacht“ (d.h. durch vorgegebene Werte definiert) werden soll, heißen **Stützstellen**.
- Die Abstände der Stützstellen, $h_i := x_{i+1} - x_i$, nennt man **Maschenweiten**. Bei konstanten $h_i = h$ spricht man von **äquidistanten** Stützstellen.
- Die vorgegebenen Werte y_i heißen **Stützwerte**. Diese können entweder direkt vorgegeben werden oder als Funktionswerte $y_i = f(x_i)$ von einer reellwertigen Funktion f stammen.

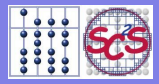


- Die Paare (x_i, y_i) werden **Stützpunkte** genannt.
- Besonders beliebt als Interpolanten sind wegen ihrer einfachen Struktur **Polynome**:
  - Mit $\mathbb{P}_n$ bezeichnen wir den Vektorraum aller Polynome mit reellen Koeffizienten vom Grad kleiner oder gleich n in einer Variablen x.
  - Es gilt $\dim(\mathbb{P}_n) = n + 1$, eine Basis bilden z.B. die **Monome** x^i, $i = 0, ..., n$:
$$p(x) := \sum_{i=0}^{n} a_i \cdot x^i.$$
  - Mit dem Differentialoperator D^k für die k-te Ableitung nach x gilt bekanntlich
$$D^{n+1} p = 0 \qquad \forall p \in \mathbb{P}_n .$$
- Es gibt allerdings keinesfalls nur die Polynom-Interpolation:
  - Man kann stückweise Polynome aneinanderkleben und erhält Polynom-Splines, die etliche wesentliche Vorteile aufweisen (siehe Abschnitt 2.3).
  - Man kann auch mit rationalen Funktionen (besonders beliebt im CAGD), mit trigonometrischen Funktionen (siehe Abschnitt 2.4) oder mit Exponentialfunktionen interpolieren.



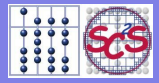
Varianten der Interpolationsaufgabe

- Es gibt verschiedene Möglichkeiten, wie wir in konkreten Anwendungen mit der Aufgabenstellung der Interpolation konfrontiert werden können. Die zwei häufigsten unter ihnen seien explizit erwähnt.
- **einfache Stützstellen**:
  - Dieser Fall wird in diesem Kapitel ausschließlich betrachtet.
  - Zu paarweise verschiedenen x_i ist jeweils ein Wert y_i vorgegeben.
  - Diese Interpolationsaufgabe wird **Lagrange-Interpolation** genannt.
  - Beispiel: Bestimme das quadratische Polynom p mit $p(-1) = p(1) = 1$ und $p(0) = 0$; Lösung: $p(x) = x^2$.
- **mehrfache Stützstellen**:
  - In den Stützstellen x_i werden jeweils Funktionswert y_i und Ableitung(en) y_i' etc. vorgegeben.
  - Eine solche Interpolationsaufgabe wird **Hermite-Interpolation** genannt.
  - Beispiel: Bestimme das quadratische Polynom q mit $q(0) = q'(0) = 0$ und $q''(0) = 2$; Lösung: $q(x) = x^2$.
  - Die Vorgabe von Ableitungen kann z.B. dazu dienen, Polynomstücke glatt (d.h. stetig und ohne Knicke etc.) aneinanderzukleben.



2.2. Interpolation mit Polynomen

Der Polynom-Interpolant und sein Fehler



- Die Interpolationsaufgabe im Falle von Polynomen:
  - $p \in \mathbb{P}_n$ heißt **Polynom-Interpolant** von f zu den Stützstellen $a = x_0 < x_1 < x_2 < ... < x_n = b$, wenn

$$p(x_i) \;=\; f(x_i) \;=:\; y_i \qquad \forall i \in \{0, 1, ..., n\}\,.$$

  - Die Definition erfolgt analog, falls anstelle einer zu interpolierenden Funktion f lediglich ein diskreter Datensatz $y_0, ..., y_n$ vorgegeben ist.
  - Die Anzahl der Stützstellen bestimmt somit direkt den Grad des Interpolationspolynoms: p ist vom Grad n, falls die Stützpunkte (x_i, y_i) nicht auf einem Polynom niedrigeren Grades liegen.
  - Bei einer großen Zahl von Stützstellen erhalten wir also i.A. hochgradige Polynome, was – wie wir gleich sehen werden – mit numerischen Problemen verbunden ist.
  - Existenz und Eindeutigkeit der Lösung dieser Interpolationsaufgabe sind aus der Analysis bekannt. Es gibt also immer genau ein Polynom $p(x)$ minimalen Grades, das die vorgegebenen $n+1$ Punkte interpoliert.

- Verwendet man zwischen den Stützstellen den Interpolanten p anstelle der Funktion f, so begeht man dort einen **Interpolationsfehler**:
  - Die Differenz $f(x)-p(x)$ heißt **Fehlerterm** oder **Restglied**. Wie wir später in diesem Abschnitt zeigen werden, gilt

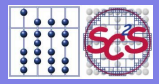
$$f(x) - p(x) \;=\; \frac{D^{n+1} f(\xi)}{(n+1)!} \cdot \prod_{i=0}^{n} (x - x_i)$$

    für eine Zwischenstelle ξ,

$$\xi \;\in\; [\min(x_0, \ldots, x_n, x), \max(x_0, \ldots, x_n, x)] \,.$$

    Im Falle hinreichend glatter (d.h. hinreichend oft differenzierbarer) Funktionen f gestattet uns diese Beziehung die Abschätzung des Interpolationsfehlers.

- Zunächst wenden wir uns allerdings der Frage zu, wie man Polynom-Interpolanten konstruieren kann.



Darstellung 1: Lagrange-Polynome

- Der einfachste Konstruktionsansatz ist die bekannte **Punkt-** oder **Inzidenz-probe**:
  - Setze $p(x)$ mit allgemeinen Koeffizienten an als

$$p(x) := \sum_{i=0}^{n} a_i \cdot x^i \,.$$

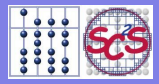
  - Setze dann für jeden Stützpunkt (x_i, y_i) das jeweilige Wertepaar ein und löse das entstehende System aus $n+1$ linearen Gleichungen in den $n+1$ Unbekannten $a_0, ..., a_n$.

- Ein alternativer Ansatz verwendet die **Lagrange-Polynome** $L_k(x)$ vom Grad n,

$$L_k(x) := \prod_{i:i\neq k} \frac{x - x_i}{x_k - x_i} \,,$$

  um den Interpolanten zu bestimmen:

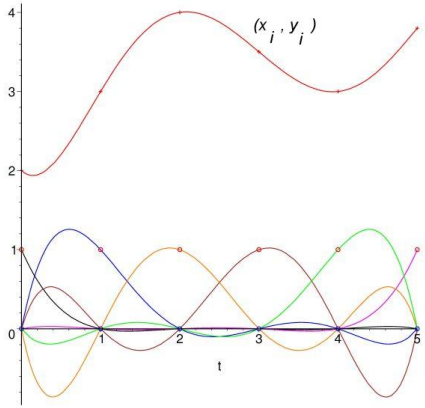
$$p(x) := \sum_{k=0}^{n} y_k \cdot L_k(x) \,.$$



- Eigenschaften der Lagrange-Polynome:
  - L_k verschwindet an allen Stützstellen außer x_k:

$$L_k(x_i) \;=\; \delta_{ik} \;=\; \begin{cases} 1 & \text{für } i = k \\ 0 & \text{sonst.} \end{cases}$$

  - Die L_k sind linear unabhängig und bilden eine Basis des $\mathbb{P}_n$.
  - Das oben definierte Polynom $p(x)$ ist tatsächlich der gesuchte Interpolant.
- Beachte, dass obige Formel nicht nur $p(x)$ für ein festes x liefert, sondern zudem eine geschlossene Darstellung des Polynoms.



Darstellung 2: Das Schema von Aitken und Neville

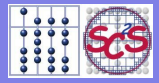
- Oft ist man nicht an einer expliziten oder geschlossenen Darstellung des Polynoms interessiert, sondern will $p(x)$ nur an einer oder mehreren Zwischenstelle(n) x auswerten. Eine solche Auswertung erfordert es interessanterweise nicht, das interpolierende Polynom p vorher explizit anzugeben. Eine elegante *rekursive* Auswertemöglichkeit liefert das **Schema von Aitken und Neville**:
  - Definiere Hilfspolynome $p_{i,k}$ vom Grad k, die die $k+1$ Stützpunkte (x_l, y_l) für $l = i, ..., i+k$ interpolieren $(p_{i,0} = y_i \ \forall_i = 0, \ldots, n)$.
  - Die $p_{i,k}(x)$ gehorchen dabei folgender Rekursionsformel:

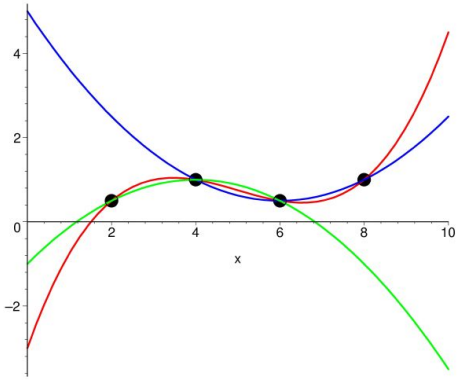
$$p_{i,k}(x) \ = \ \frac{x_{i+k} - x}{x_{i+k} - x_i} \cdot p_{i,k-1}(x) \ + \ \frac{x - x_i}{x_{i+k} - x_i} \cdot p_{i+1,k-1}(x) \, .$$

    Die Formel ist leicht anhand der Interpolationseigenschaften sowie der Eindeutigkeit der Interpolationsaufgabe zu verifizieren.

- Dieses Schema kann für verschiedene Zwecke eingesetzt werden:
  - Für einen konkreten Wert x liefert es den Polynomwert $p(x)$ als $p_{0,n}(x)$.
  - Mit Maple etwa kann x als Variable behandelt werden, wodurch man als Ergebnis das interpolierende Polynom p als $p_{0,n}$ explizit, d.h. in geschlossener Darstellung bekommt.

- Aufgrund der Eindeutigkeit der Interpolationsaufgabe ist das hier ermittelte p natürlich mit der Summe der Lagrange-Polynome identisch – lediglich der Weg, wie $p(x)$ berechnet wird, ist unterschiedlich.



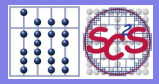


Das Schema von Aitken und Neville (2)

- Wir halten das Schema von Aitken-Neville im Algorithmus fest:

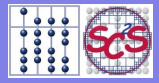
```
for i=0 to n do p[i,0] := y[i] od;
for k=1 to n do
    for i=0 to n-k do
        p[i,k] := (x[i+k]-x)/(x[i+k]-x[i])*p[i,k-1] +
                  (x-x[i])/((x[i+k]-x[i])*p[i+1,k-1];
    od;
od;
```

- Ob sich die Angabe einer geschlossenen Darstellung für p rechnet, hängt i.W. von der konkreten Aufgabenstellung ab:
  - Sind nur zu einer oder zu wenigen Stützstellen die Polynomwerte anzugeben, dann ist die direkte Berechnung der Werte $p(x)$ die bessere Wahl.
  - Werden dagegen viele Auswertungen benötigt, dann kann sich die explizite Berechnung des Polynoms (d.h. aller seiner Koeffizienten) lohnen.



Darstellung 3: Die Newtonsche Interpolationsformel

- Für eine weitere Darstellungsmöglichkeit für den Polynom-Interpolanten benötigen wir die so genannten **dividierten Differenzen**:
  - Der höchste Koeffizient des zuvor eingeführten Polynoms $p_{i,k}$ wird mit
    $$[x_i, ..., x_{i+k}]f$$
    bezeichnet und **dividierte Differenz von f der Ordnung k zu $x_i, ..., x_{i+k}$** genannt, d.h.
    $$p_{i,k}(x) \; - \; [x_i, ..., x_{i+k}]f \cdot x^k \; \in \; \mathbb{P}_{k-1} \, .$$
  - Für die dividierten Differenzen kann man aus dem Schema von Aitken und Neville ebenfalls eine Rekursionsformel ableiten:
    $$[x_i, ..., x_{i+k}]f \; = \; \frac{[x_{i+1}, ..., x_{i+k}]f - [x_i, ..., x_{i+k-1}]f}{x_{i+k} - x_i} \, .$$
  - Auch diese Rekursionsformel führt zu einem Neville-artigen Dreiecksschema.
  - Es gilt
    $$[x_i]f \; = \; f(x_i) \; = \; y_i$$
    sowie
    $$[x_i, x_{i+1}]f \; = \; \frac{y_{i+1} - y_i}{x_{i+1} - x_i} \, .$$
  - Weder für das Aitken-Neville-Schema noch für die dividierten Differenzen spielt die Reihenfolge der Stützstellen eine Rolle.

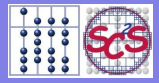


Die Newtonsche Interpolationsformel (2)

- Mit den dividierten Differenzen erhalten wir eine zweite geschlossene Darstellung für $p(x)$, die **Newtonsche Interpolationsformel**:

$$\begin{aligned} p(x) \quad := \quad & [x_0]f \; + \\ & [x_0, x_1]f \cdot (x - x_0) \; + \\ & [x_0, x_1, x_2]f \cdot (x - x_0) \cdot (x - x_1) \; + \\ & [x_0, \ldots, x_3]f \cdot (x - x_0) \cdot (x - x_1) \cdot (x - x_2) \; + \\ & \ldots \\ & [x_0, \ldots, x_n]f \cdot \prod_{i=0}^{n-1} (x - x_i) \end{aligned}$$

- Der Reiz dieser Darstellung ist ihr **inkrementeller** Charakter:
  - Fügt man eine weitere Stützstelle x_{n+1} hinzu, so ist der aktuelle Interpolant zu $x_0, ..., x_n$ in seiner Newton-Gestalt lediglich um den weiteren Summanden
$$[x_0, \ldots, x_{n+1}]f \cdot \prod_{i=0}^{n} (x - x_i)$$
    zu ergänzen.
  - Das bisher Berechnete geht also bei nachträglicher Erhöhung des Polynomgrads nicht verloren.
  - Man veranschaulicht sich leicht, dass diese schöne Eigenschaft weder für die Lagrange- noch für die Aitken-Neville-Darstellung gilt.



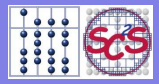
Zur Abschätzung des Interpolationsfehlers

- Jetzt kommen wir zum eingangs erwähnten Interpolationsfehler $f(x) - p(x)$ an einer beliebigen Zwischenstelle x. Bei seiner Abschätzung helfen die dividierten Differenzen:
  - Falls f n-mal stetig differenzierbar ist auf $[x_0, x_n]$, dann gibt es eine Zwischenstelle ξ in diesem Intervall mit

$$[x_0, ..., x_n]f \;=\; \frac{D^n f(\xi)}{n!}\,.$$

    Der Beweis hierfür erfolgt analog zum aus der Analysis bekannten Mittelwertsatz.
  - Betrachte jetzt ein $\bar{x} \in [x_0, x_n]$, welches nicht Stützstelle ist. Bezeichne ferner $P(x)$ das Polynom vom Grad $n+1$, das zusätzlich zu den Interpolationseigenschaften von $p(x)$ unsere Funktion f (jetzt als $n+1$-mal stetig differenzierbar angenommen) auch noch in $\bar{x}$ interpoliert. Dann gilt:

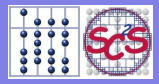
$$\begin{aligned} f(\bar{x}) - p(\bar{x}) \;=\; P(\bar{x}) - p(\bar{x}) \;&=\; [x_0, \ldots, x_n, \bar{x}]f \cdot \prod_{i=0}^{n}(\bar{x} - x_i) \\ &=\; \frac{D^{n+1} f(\xi)}{(n+1)!} \cdot \prod_{i=0}^{n}(\bar{x} - x_i)\,; \end{aligned}$$



– Für äquidistante Stützstellen mit Maschenweite $h := x_{i+1} - x_i$ beispielsweise kann man den Fehler abschätzen:

$$| f(\bar{x}) - p(\bar{x}) | \leq \frac{\max_{[a,b]} | D^{n+1} f(x) |}{n+1} \cdot h^{n+1} = O(h^{n+1}) .$$

– Die Fehlerabschätzung zeigt uns aber auch, dass die äqudistante Wahl der Stützstellen nicht optimal ist. Eine $n+1$-te Stützstelle $\bar{x}$ sollte vielmehr so gewählt werden, dass der maximale Wert des Produkts auf der rechten Seite minimiert wird.



Die Kondition der Polynom-Interpolation

- Wie gut oder schlecht ist die Interpolationsaufgabe konditioniert?
  - Eingabedaten: die Stützstellen x_i, die Stützwerte y_i sowie die Zwischenstelle x, an der der Funktionswert durch Interpolation näherungsweise rekonstruiert werden soll
  - Ergebnis: der Wert $y := p(x)$
- Somit gibt es verschiedene „Konditionen“ – je nachdem, bzgl. welchem Eingabedatum die Sensitivität betrachtet werden soll.
- Am einfachsten ist die Empfindlichkeit von $p(x)$ gegenüber Schwankungen in der Auswertestelle x – sie wird gerade durch $p'(x)$ beschrieben und ist bei Polynomen zwar in jedem Fall beschränkt auf $[a, b]$, kann aber dennoch (gerade bei höherem Grad) sehr groß werden.
- Wichtiger für die Praxis ist die Empfindlichkeit gegenüber Schwankungen in den Stützpunkten und insbesondere den Stützwerten. Aus

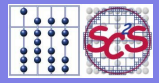
$$y \;=\; p(x) \;=\; \sum_{k=0}^{n} y_k \cdot L_k(x)$$

folgt unmittelbar

$$\frac{\partial y}{\partial y_k} \;=\; L_k(x)\,.$$

Die Größe der Lagrange-Polynome bestimmt also diese Kondition.

- Um ein Gefühl für die Größenordnungen zu bekommen, die die Werte der Lagrange-Polynome erreichen, betrachten wir hierzu ein Beispiel.



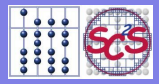
Ein Beispiel zur Kondition der Polynom-Interpolation

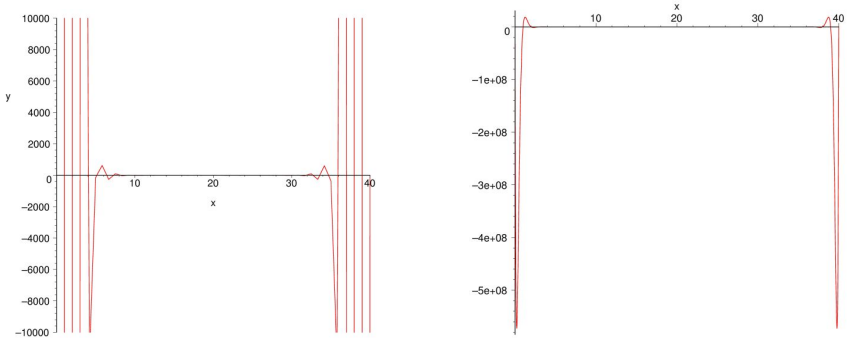
- Gegeben sei die folgende Situation:
$$x_i := i,\ i = 0, 1, ..., 40 =: n; \qquad k := 20; \qquad x \in]x_0, x_1[=]0, 1[$$

- Daraus ergit sich folgende Abschätzung für den Wert von L_{20} in x:

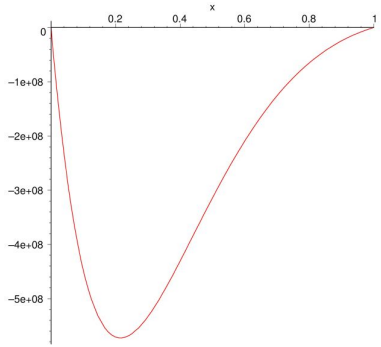
$$\begin{aligned} |\, L_{20}(x)\,| &= \Big|\, \prod_{i\neq 20} \frac{x - x_i}{x_{20} - x_i}\,\Big| = \prod_{i\neq 20} \frac{|\, x - x_i\,|}{|\, 20 - x_i\,|} \\ &= \frac{x}{20} \cdot \frac{1-x}{19} \cdot \prod_{i=2}^{19} \frac{i-x}{20-i} \cdot \prod_{i=21}^{40} \frac{i-x}{i-20} \\ &\geq \frac{x - x^2}{380} \cdot \prod_{i=2}^{19} \frac{i-1}{20-i} \cdot \prod_{i=21}^{40} \frac{i-1}{i-20} \\ &= \frac{x - x^2}{380} \cdot \frac{18!}{18!} \cdot \frac{39!}{19! \cdot 20!} \\ &\geq 1.8 \cdot 10^8 \cdot (x - x^2) \end{aligned}$$

- Insbesondere gilt $|\, L_{20}(0.5)\,| \ \geq\ 4.5 \cdot 10^7$.
- Dies ist ein grundlegendes Resultat: Kleine Fehler in den zentralen Stützwerten werden durch die Polynominterpolation am Rand des betrachteten Intervalls drastisch verstärkt.
- Für große n (ab 7 oder 8) ist die Polynom-Interpolation somit extrem schlecht konditioniert und deshalb faktisch unbrauchbar.
- Aus diesem Grund muss man sich um diesbezüglich bessere Verfahren kümmern.





Lagrange-Polynom $L_{20}(x)$ auf $[0, 40]$

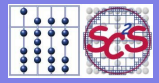


Lagrange-Polynom $L_{20}(x)$ auf $[0, 1]$

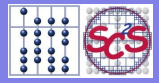
2.3. Polynom-Splines

Definition der Polynom-Splines

- Die Hauptnachteile der Polynom-Interpolation zur Erinnerung:
  - Anzahl der Stützpunkte und Polynomgrad sind starr aneinander gekettet.
  - Für größeres n (und somit für den in der Praxis oft auftreten Fall einer größeren Zahl von Stützpunkten) ist die Polynominterpolation schon aus Konditionsgründen unbrauchbar.
- Idee zur Abhilfe:
  - Klebe Polynomstücke niedrigen Grades aneinander, um so einen globalen Interpolanten auch für eine große Zahl von Stützpunkten zu konstruieren.
  - Genau dies machen **Polynom-Splines** oder kurz **Splines**, die wir jedoch allgemein ohne konkreten Bezug zur Interpolationsaufgabe einführen.
  - Sei wieder $a = x_0 < x_1 < ... < x_n = b$ und $m \in \mathbb{N}$. Die x_i werden **Knoten** genannt. Wir betrachten nur den Spezialfall **einfacher** Knoten, d.h. $x_i \neq x_j$ für $i \neq j$.
  - Eine Funktion $s : [a, b] \to \mathbb{R}$ heißt **Spline der Ordnung** m bzw. **vom Grad** $m - 1$, wenn gilt:
    - $s(x) = p_i(x)$ auf $[x_i, x_{i+1}]$ mit $p_i \in \mathbb{P}_{m-1}$, $i = 0, 1, ..., n - 1$.
    - $s \in \mathcal{C}^{m-2}([a, b])$.

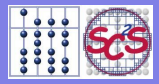


– Zwischen je zwei benachbarten Knoten ist s also ein Polynom vom Grad $m-1$, und global (also insbesondere auch in den Knoten selbst!) ist s $m-2$-mal stetig differenzierbar.
– Man sagt auch, s ist **stückweise** ein Polynom.

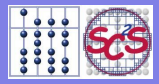


Beispiele, Eigenschaften und Einsatzgebiete

- Wie sieht ein Polynom-Spline aus? Wir betrachten hierzu die einfachsten Fälle:
  - $m = 1$: s ist eine Treppenfunktion (stückweise konstant, nicht einmal Stetigkeit in den Knoten ist gegeben)
  - $m = 2$: s ist ein Polygonzug (stückweise linear, stetig in den Knoten)
  - $m = 3$: quadratischer Spline (stückweise quadratisch, global einmal stetig differenzierbar)
  - $m = 4$: kubischer Spline (stückweise kubisch, global zweimal stetig differenzierbar)
- zum Namen: Das englische Wort *Spline* bezeichnet eine elastische Holzlatte, wie sie beim Schiffsbau eingesetzt wird bzw. wurde.
- Bei festen Knoten und festem Grad bildet die Menge aller Splines offensichtlich einen Vektorraum. Dessen Dimension ist übrigens $n + m - 1$ (ein globales $p_0 \in \mathbb{P}_{m-1}$ legt m Freiheitsgrade fest; dazu kommt für jeden Innenknoten $x_1, ..., x_{n-1}$ ein Freiheitsgrad für einen möglichen Sprung in der $m-1$-ten Ableitung – alle niedrigeren Ableitungen sind ja stetig).
- Haupteinsatzgebiete für Splines:
  - **Interpolation**: Ein Spline soll so konstruiert werden, dass er $n + m - 1$ Interpolationsbedingungen erfüllt – entsprechend der Anzahl seiner Freiheitsgrade. Die Stützstellen der Interpolation können, müssen aber nicht Knoten sein.

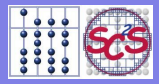


– **CAGD**: Hier soll der Spline so konstruiert werden, dass er einen durch Kontrollpunkte vorgegebenen Kurvenverlauf annimmt (wichtig im Zusammenhang mit **Freiformkurven und -flächen**).



Interpolation mit kubischen Splines

- Jetzt wollen wir Splines konkret zur Interpolation einsetzten. Für Interpolationszwecke weit verbreitet und außerdem sehr schön zu konstruieren und zu analysieren sind **kubische Splines**, also der Fall $m = 4$. Betrachte folgenden Spezialfall:
  - einfache Knoten: $a = x_0 < x_1 < ... < x_n = b$
  - lokal: Spline $s \in \mathbb{P}_3$ auf jedem Teilintervall $[x_i, x_{i+1}]$, $i = 0, 1, ..., n-1$
  - global: Spline $s \in \mathcal{C}^2([a, b])$
  - Stützstellen seien gerade die Knoten, also lauten die Interpolationsbedingungen
$$s(x_i) \;=\; y_i\,, \qquad i = 0, 1, ..., n\,.$$
- Damit gilt:
  - Wir haben $n+m-1 = n+3$ Freiheitsgrade für die Festlegung des konkreten Spline (Dimension des zugrunde liegenden Vektorraums, s.o.).
  - Davon werden $n+1$ durch die obigen Interpolationsbedingungen aufgebraucht.
  - Somit bleiben 2 offene Freiheitsgrade, die wir für weitere Forderungen an s verwenden können. Doch dazu später.
  - Zuerst machen wir uns an daran, unseren kubischen Spline zu „bauen“.



Die lokalen Polynom-Segmente

- Definiere auf jedem Teilintervall $[x_i, x_{i+1}]$ das lokale kubische Polynom als Funktion der vier Parameter y_i und y_{i+1} (Funktionswerte in den beiden tangierten Stützstellen) sowie y'_i und y'_{i+1} (erste Ableitungen ebenda):

$$\begin{aligned} s(x) &= p_i\left(\frac{x - x_i}{x_{i+1} - x_i}\right) =: p_i(t) \\ &=: y_i \cdot \alpha_1(t) + y_{i+1} \cdot \alpha_2(t) + (x_{i+1} - x_i) \cdot \left(y'_i \cdot \alpha_3(t) + y'_{i+1} \cdot \alpha_4(t)\right) \end{aligned}$$

mit $\alpha_1, \alpha_2, \alpha_3, \alpha_4 \in \mathbb{P}_3$, $t \in [0, 1]$ und

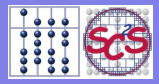
$$\begin{aligned} \alpha_1(0) &= 1\,, \quad \alpha_1(1) = \alpha'_1(0) = \alpha'_1(1) = 0 \quad \Rightarrow \quad \alpha_1(t) := 1 - 3t^2 + 2t^3 \\ \alpha_2(1) &= 1\,, \quad \alpha_2(0) = \alpha'_2(0) = \alpha'_2(1) = 0 \quad \Rightarrow \quad \alpha_2(t) := 3t^2 - 2t^3 \\ \alpha'_3(0) &= 1\,, \quad \alpha_3(0) = \alpha_3(1) = \alpha'_3(1) = 0 \quad \Rightarrow \quad \alpha_3(t) := t - 2t^2 + t^3 \\ \alpha'_4(1) &= 1\,, \quad \alpha_4(0) = \alpha_4(1) = \alpha'_4(0) = 0 \quad \Rightarrow \quad \alpha_4(t) := -t^2 + t^3 \end{aligned}$$

- Man sieht leicht:

$$s(x_i) = p_i(0) = y_i\,, \qquad s(x_{i+1}) = p_i(1) = y_{i+1}\,,$$

$$\frac{ds}{dx}(x_i) = \frac{dp_i}{dt}(0) \cdot \frac{dt}{dx}(x_i) = y'_i\,, \qquad \frac{ds}{dx}(x_{i+1}) = \frac{dp_i}{dt}(1) \cdot \frac{dt}{dx}(x_{i+1}) = y'_{i+1}\,;$$

- Beachte, dass dieser Ansatz also bereits die globale Stetigkeit und stetige Differenzierbarkeit garantiert!



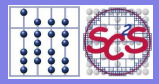
Das globale Zusammenkleben

- In der Darstellung über lokale Polynome haben wir $2n + 2$ Freiheitsgrade ($n + 1$ Knotenwerte und $n + 1$ Knotenableitungen), von denen $n + 1$ durch die $n + 1$ Interpolationsbedingungen bereits festgelegt sind.
- Somit verbleiben $n + 1$ Freiheitsgrade zur Sicherstellung der Stetigkeit auch der zweiten Ableitung an den $n - 1$ Innenknoten. Auch in dieser Sehweise bleiben also 2 freie Parameter übrig.
- Wir leiten alle lokalen Polynome zweimal ab und setzen an den $n - 1$ Klebestellen (Nahtstellen) die Stetigkeit an, indem wir die zweite Ableitung von p_{i-1} in $t = 1$ mit der zweiten Ableitung von p_i in $t = 0$ gleichsetzen:

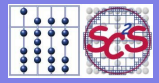
$$y'_{i-1}\frac{1}{h_{i-1}} + y'_i\left(\frac{2}{h_{i-1}} + \frac{2}{h_i}\right) + y'_{i+1}\frac{1}{h_i} = 3\left(\frac{y_i - y_{i-1}}{h_{i-1}^2} + \frac{y_{i+1} - y_i}{h_i^2}\right),$$

$$h_i := x_{i+1} - x_i, i = 1, \ldots, n-1\,.$$

- Dies ist offensichtlich ein tridiagonales lineares Gleichungssystem der Dimension $n - 1$ – eine erste Motivation, uns in den Kapiteln 4 und 5 intensiv mit der numerischen Lösung linearer Gleichungssysteme zu befassen.



- Wie soll man die Bedingungen für die beiden restlichen Freiheitsgrade festlegen? Üblicherweise werden **Randbedingungen** festgelegt. Dabei sind drei Vorgehensweisen verbreitet:
  - Vorgabe der Steigung am Rand, also von y'_0 und y'_n
  - Vorgabe der zweiten Ableitungen am Rand (**natürliche Randbedingungen**)
  - **periodische Randbedingungen**: Die ersten und zweiten Ableitungen stimmen an beiden Randpunkten jeweils überein, also $y'_0 = y'_n$ und $s''(x_0) = s''(x_n)$.

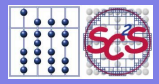


Kosten und Leistung

- Was kostet die kubische Spline-Interpolation?
  - Ein (je nach Wahl der Randbedingungen u.U. leicht gestörtes) tridiagonales Gleichungssystem ist zu lösen.
  - Man macht sich leicht klar, dass dies mit $O(n)$ arithmetischen Operationen zu bewerkstelligen ist (z.B. mittels der aus der Linearen Algebra bzw. Mathematik-Vorlesung bekannten Gauß-Elimination).
  - Dies ist mit den $O(n^3)$ Rechenoperationen (Reminiszenz an die Lineare Algebra oder siehe Kapitel 4) für das Lösen des vollen Gleichungssystems bei der Inzidenzprobe bzw. mit dem quadratischen Rechenaufwand des Aitken-Neville-Schemas zu vergleichen.
- Welche Genauigkeit bringt die Spline-Operation?
  - Da wir lokal Polynome dritten Grades verwenden und die Stützstellen jeweils den Abstand $h := x_{i+1} - x_i$ haben, liefert eine Fehlerabschätzung analog zu Abschnitt 2.2

$$| f(x) - s(x) | \;=\; O(h^4)\,.$$

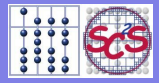
  - Dies ist mit der Fehlerordnung $O(h^{n+1})$ bei der Polynom-Interpolation zu vergleichen. Allerdings wird dort eine extrem hohe Differenzierbarkeit von f vorausgesetzt, und es stellen sich bei größerem n die zuvor festgestellten Probleme mit der Kondition ein.



2.4. Trigonometrische Interpolation

Das Prinzip der trigonometrischen Interpolation

- Bei der **trigonometrischen Interpolation** betrachtet man komplexwertige Funktionen, die auf dem Einheitskreis in der komplexen Zahlenebene definiert sind – man spricht auch von der **Darstellung im Frequenzbereich**.
- Gegeben seien also n Stützstellen auf dem Einheitskreis der komplexen Zahlebene,
$$z_j := e^{\frac{2\pi i}{n} j}, \qquad j = 0, 1, ..., n-1,$$
sowie n Stützwerte v_j. Gesucht ist der Interpolant
$$p(z), \quad z = e^{2\pi i t}, t \in [0, 1],$$
mit
$$p(z_j) = v_j, \quad j = 0, 1, ..., n-1, \qquad p(z) = \sum_{k=0}^{n-1} c_k z^k = \sum_{k=0}^{n-1} c_k e^{2\pi i k t}.$$
- $p(z)$ wird also als Linearkombination von Exponentialfunktionen oder – nach Auftrennung in Realteil und Imaginärteil – von Sinus- und Cosinusfunktionen angesetzt.
- Dieses p finden heißt de facto die Koeffizienten c_k berechnen, und die sind natürlich nichts anderes als die Koeffizienten der (diskreten) Fouriertransformierten.
- Für diese Aufgabe werden wir im Folgenden einen fast schon legendären Algorithmus, die **Schnelle Fourier-Transformation** oder **Fast Fourier Transform (FFT)**, diskutieren.



Die diskrete Fourier-Transformation

- Mit dem zuvor eingeführten p und der Abkürzung

$$\omega := e^{\frac{2\pi i}{n}}$$

stellt sich also folgende Aufgabe: Finde n komplexe Zahlen $c_0, ..., c_{n-1}$, die

$$v_j = p(\omega^j) = \sum_{k=0}^{n-1} c_k \omega^{jk} \qquad \text{für} \qquad j = 0, 1, ..., n-1$$

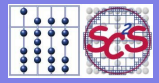
erfüllen.

- Mit etwas Analysis ($\bar{\omega}$ konjugiert komplex zu ω) kann man zeigen, dass

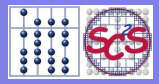
$$c_k = \frac{1}{n} \sum_{j=0}^{n-1} v_j \bar{\omega}^{jk} \qquad \text{für} \qquad k = 0, 1, ..., n-1\,.$$

- Wir bezeichnen mit c und v die n-dimensionalen Vektoren der diskreten Fourier-Koeffizienten bzw. der DFT-Eingabe. Ferner sei die Matrix M gegeben als $M = (\omega^{jk})_{0 \le j,k \le n-1}$. Somit gilt in Matrix-Vektor-Schreibweise

$$v = M \cdot c,\ c = \frac{1}{n} \cdot \overline{M} \cdot v\,.$$



- Die Formel für die Berechnung der Koeffizienten c_k gibt gerade die **diskrete Fourier-Transformation (DFT)** der Ausgangsdaten v_k an.
- Die Formel für die Berechnung der Werte v_j aus den Fourier-Koeffizienten c_k wird **inverse diskrete Fourier-Transformation (IDFT)** genannt.
- Offenkundig ist die Zahl der erforderlichen arithmetischen Operationen für DFT und IDFT von der Ordnung $O(n^2)$. Der FFT-Algorithmus kommt dagegen für geeignete n mit $O(n \log n)$ Operationen aus.



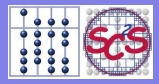
Der FFT-Algorithmus – Grundprinzip (1)

- **Grundidee**: Zerlege Summen der Länge n in zwei Teilsummen halber Länge und fahre rekursiv so lange fort, bis man triviale Teilsummen der Länge 1 erreicht hat.
- Daher beschränken wir uns von vornherein auf den Fall $n = 2^p$, d.h. auf Zweierpotenzen.
- Wir untersuchen im Folgenden die inverse diskrete Fourier-Transformation IDFT. Das Verfahren für die eigentliche DFT ergibt sich daraus offensichtlich durch Übergang zum konjugiert Komplexen und Division durch n:

$$v = IDFT(c),\ c = DFT(v) \ = \ \frac{1}{n} \cdot \overline{IDFT(\overline{v})}\,.$$

- Für $v = IDFT(c)$ gilt mit $n = 2m$ für $j = 0, 1, ..., m-1$

$$\begin{aligned} v_j \ = \ \sum_{k=0}^{n-1} c_k \cdot e^{\frac{2\pi i j k}{n}} \ &= \ \sum_{k=0}^{n/2-1} c_{2k} \cdot e^{\frac{2\pi i j 2k}{n}} + \sum_{k=0}^{n/2-1} c_{2k+1} \cdot e^{\frac{2\pi i j (2k+1)}{n}} \\ &= \ \sum_{k=0}^{m-1} c_{2k} \cdot e^{\frac{2\pi i j k}{m}} + \omega^j \cdot \sum_{k=0}^{m-1} c_{2k+1} \cdot e^{\frac{2\pi i j k}{m}}\,. \end{aligned}$$



- Die noch fehlenden Einträge $v_m, ..., v_{n-1}$ (die obere Indexhälfte) erhält man nach demselben Prinzip gemäß

$$\begin{aligned} v_{m+j} &= \sum_{k=0}^{m-1} c_{2k} \cdot e^{\frac{2\pi i(j+m)2k}{n}} + \sum_{k=0}^{m-1} c_{2k+1} \cdot e^{\frac{2\pi i(j+m)(2k+1)}{n}} \\ &= \sum_{k=0}^{m-1} c_{2k} \cdot e^{\frac{2\pi ijk}{m}} - \omega^j \cdot \sum_{k=0}^{m-1} c_{2k+1} \cdot e^{\frac{2\pi ijk}{m}} \end{aligned}$$

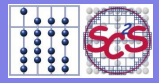
für $j = 0, 1, ..., m-1$, da

$$e^{\frac{2\pi i(j+m)}{n}} = e^{\pi i} \cdot e^{\frac{\pi ij}{m}} = -e^{\frac{\pi ij}{m}} = -\omega^j .$$

- Dies entspricht einer Strukturierung

$$v_j = A + \omega^j \cdot B, \; v_{m+j} = A - \omega^j \cdot B ,$$

wobei die Bausteine A und B ihrerseits Fourier-Summen halber Länge sind.



Der FFT-Algorithmus – Grundprinzip (2)

- Wir können also die Berechnung von $v = IDFT(c)$ zurückführen auf

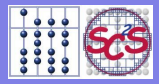
$$IDFT(c_0, c_2, ..., c_{n-2}) \qquad \text{und} \qquad IDFT(c_1, c_3, ..., c_{n-1}),$$

  die dann geeignet linear kombiniert werden müssen, um die erste bzw. zweite Hälfte von v zu erhalten.

- „Geeignet kombiniert“ heißt hier Verknüpfung mithilfe des so genannten **Butterfly-Schemas**.

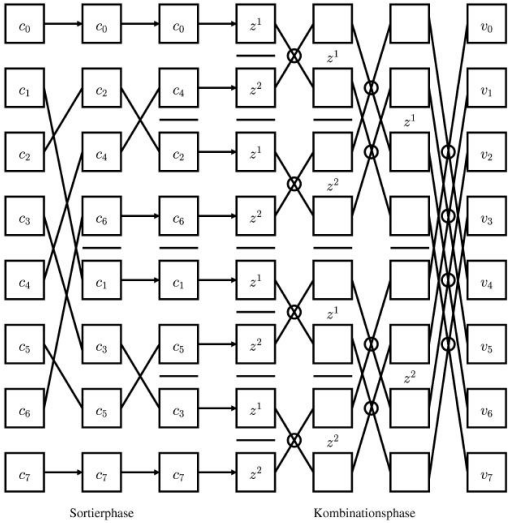
- Diese Vorgehensweise wird nun rekursiv wiederholt. Damit können wir folgendes erstes rekursives Programm in Pseudocode angeben:

```
funct IDFT(c[0],...,c[n-1],n);
if (n=1) then do v[0]:=c[0] od;
else do
  m:=n/2;
  z1:=IDFT(c[0],c[2],...,c[n-2],m);
  z2:=IDFT(c[1],c[3],...,c[n-1],m);
  omega:=exp(2*pi*i/n);
  for j:=0 to m-1 do
    v[j]:=z1[j]+omega^j*z2[j];
    v[m+j]:=z1[j]-omega^j*z2[j];
  od;
  return(v[0],...,v[n-1]);
od;
```

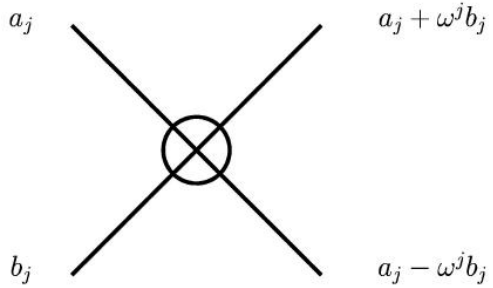


Analyse des FFT-Algorithmus

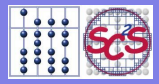
- Obiger Basisalgorithmus ist rekursiv; für ein besseres Laufzeit- und Speicherplatzverhalten wurden iterative Varianten entwickelt.
- Der FFT-Algorithmus zerfällt in natürlicher Weise in eine **Sortierphase** (Umordnen der $c[k]$) und in eine **Kombinationsphase** (Butterfly-Operation). Am Beispiel $n = 8$ kann man sich dies veranschaulichen:



- Dabei steht die Butterfly-Operation für

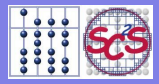


- Aus der Abbildung auf der vorigen Folie entnehmen wir intuitiv Zweierlei:
  - Die dem FFT-Algorithmus zugrunde liegenden Prinzipien findet man an ganz anderen Stellen wieder, z.B. bei dynamischen Netzwerken wie dem **Shuffle-Exchange**.
  - Die Rechenkomplexität des FFT-Algorithmus ist von der Ordnung $O(n \log(n))$.



Eine Anwendung: Kompression von Bilddaten

- Ein Graustufenbild werde als Matrix von Grauwerten aufgefasst. Jeder Matrixeintrag $g_{i,j}$ entspricht dann dem Grauwert des Bildpixels in Zeile i und Spalte j des Bildes.
- Um unabhängig von der Anzahl der zur Verfügung stehenden Graustufen zu bleiben, erlauben wir $g_{i,j} \in [0, 1]$, wobei Null der Farbe Schwarz entspricht.
- Das **JPEG-Verfahren** macht sich die besondere Frequenzverteilung „natürlicher“ Bilder zunutze, um diese stark komprimiert zu speichern. Dabei werden Verluste in Kauf genommen – die Kompression ist **verlustbehaftet**. Die Komprimierung erfolgt in folgenden Schritten:
  - Farbbilder werden in das YUV-Farbmodell umgewandelt (Y steht für die Intensität, U und V bezeichnen Farbton und Sättigung).
  - Das Bild wird in Blöcke der Größe 8×8 Pixel eingeteilt. Auf jedem dieser Blöcke wird eine **Schnelle Cosinus-Transformation** (**Fast Cosine Transform FCT**) ausgeführt, die auf der nächsten Folie beschrieben wird.
  - Die Frequenzwerte werden **quantisiert** – nur eine bestimmte Anzahl diskreter Farbwerte wird zugelassen, Zwischenwerte müssen entsprechend gerundet werden. Dieser Schritt ist wesentlich für die auftretenden Qualitätsverluste maßgeblich.
  - Schließlich werden die quantisierten Daten geeignet komprimiert.



Die Cosinus-Transformation

- Bilddaten sind weder periodisch, noch haben sie komplexe Anteile. Die „herkömmliche“ Fourier-Transformation ist daher nicht anwendbar.
- Das JPEG-Verfahren bedient sich folgender Variante, der so genannten
- **Cosinus-Transformation**:
  - Transformation:

$$F_k := \sum_{j=0}^{N-1} f_j \cos\left(\frac{\pi k(j+\frac{1}{2})}{N}\right) \qquad k = 0, 1, ..., N-1$$

  - inverse Transformation:

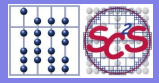
$$f_j := \frac{2}{N} \sum_{k=0}^{N-1}{}' F_k \cos\left(\frac{\pi k(j+\frac{1}{2})}{N}\right) := \frac{2}{N}\left(\frac{F_0}{2} + \sum_{k=1}^{N-1} F_k \cos\left(\frac{\pi k(j+\frac{1}{2})}{N}\right)\right)$$

$$j = 0, ..., N-1$$

- Anstelle der komplexwertigen Exponentialterme tauchen hier nur reellwertige Cosinusterme als Grundbausteine auf.
- In 2 D – dem relevanten Fall für Bilddaten – ergibt sich folgende Darstellung:

$$F_{k_1,k_2} := \sum_{j_1=0}^{N_1-1} \sum_{j_2=0}^{N_2-1} f_{j_1,j_2} \cos\left(\frac{\pi k_1(j_1+\frac{1}{2})}{N_1}\right) \cos\left(\frac{\pi k_2(j_2+\frac{1}{2})}{N_2}\right)$$

$$f_{j_1,j_2} := \frac{4}{N_1 N_2} \sum_{k_1=0}^{N_1-1}{}' \sum_{k_2=0}^{N_2-1}{}' F_{k_1,k_2} \cos\left(\frac{\pi k_1(j_1+\frac{1}{2})}{N_1}\right) \cos\left(\frac{\pi k_2(j_2+\frac{1}{2})}{N_2}\right)$$



Rückführung auf die FFT

- Auf der Suche nach einem schnellen Algorithmus zur Lösung der diskreten Cosinus-Transformation wollen wir unsere FFT-Kenntnisse einfließen lassen.
- Hierzu führen wir obige 2 D Cosinus-Transformation in drei Schritten auf die bekannte FFT zurück:
  - Die 2 D Cosinus-Transformation lässt sich durch Hintereinanderausführung von 1 D Cosinus-Transformationen realisieren.
  - Durch Spiegelung der Werte von F_k bzw. f_j lässt sich die Cosinus-Transformation in eine modifizierte Fourier-Transformation doppelter Länge überführen. Mit

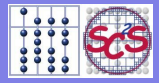
$$\tilde{f}_j := \begin{cases} f_j & j = 0, ..., N-1, \\ f_{2N-j-1} & j = N, ..., 2N-1 \end{cases} \qquad \text{und}$$

$$\tilde{F}_k := \begin{cases} 2F_k & k = 0, ..., N-1, \\ 0 & k = N, \\ -2F_{2N-k} & k = N+1, ..., 2N-1 \end{cases}$$

gilt

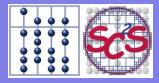
$$\tilde{f}_j = \frac{1}{2N} \sum_{k=0}^{2N-1} \tilde{F}_k e^{\left(\frac{-2\pi i(j+\frac{1}{2})k}{2N}\right)}, \qquad \text{und} \qquad \tilde{F}_k = \sum_{j=0}^{2N-1} \tilde{f}_j e^{\left(\frac{2\pi i(j+\frac{1}{2})k}{2N}\right)}.$$

  - Durch geeignete Skalierung der $\tilde{F}_k$ und $\tilde{f}_j$ ergibt sich die eingeführte Standard-Darstellung der (I)DFT.



2.5. Anwendungsbeispiele der Interpolation

Interpolation in der Informatik



- **Tafelwerke** – früher auch für Funktionen wie den Logarithmus oder die trigonometrischen Funktionen unerlässlich, heute eigentlich nur noch für Verteilungsfunktionen wie die Normalverteilung in Gebrauch – drucken Funktionswerte an diskreten Stützstellen ab. Will man Zwischenwerte, so muss man interpolieren.
- In jedem **Funktionsplot** müssen diskrete Punkte interpoliert werden.
- Beliebig gekrümmte Kurven und Flächen – so genannte **Freiformkurven** bzw. **Freiformflächen** – spielen in der geometrischen Modellierung und in der Computergrafik eine große Rolle.
- Bei der Erzeugung von **Computer-Animationen**, also z.B. von Trickfilm-Sequenzen, ist die explizite Erzeugung jedes Einzelbilds (**Frames**) bei Echtzeitanwendungen i.A. zu rechenaufwändig – schließlich werden für eine optisch ansprechende (d.h. insbesondere ruckfreie) Animation ca. 25 Bilder pro Sekunde benötigt. Aus diesem Grund berechnet man oft nur bestimmte Schlüsselbilder (**Keyframes**) und bestimmt die Zwischenbilder durch Interpolation, in diesem Zusammenhang auch **Inbetweening** genannt.
- Wenn sich Roboter bewegen (z.B. Fußball spielen), werden diskrete Zielpunkte berechnet – der Weg vom momentanen Standpunkt zum nächsten Ziel wird dann mittels Interpolation bestimmt.

- Bei der **Bildrekonstruktion** muss aus komprimierten oder fehlerhaften Bilddaten das Original möglichs genau rekonstruiert werden – auch dies ein Fall für Interpolation.
- Letztendlich tritt bei jeder **DA-Wandlung** eines Signals die Aufgabe der Interpolation auf.