

# Algorithmen und Datenstrukturen (für ET/IT)

Wintersemester 2012/13

Dr. Tobias Lasser

Computer Aided Medical Procedures  
Technische Universität München



# Weiterer Ablauf Vorlesung

- Klausurstoff: Kapitel 1 bis Kapitel 10 (einschließlich)
- nicht klausur-relevant:
  - Kapitel 11: Datenkompression (Huffmann-Codierung, JPEG)
  - Kapitel 12: Kryptographie (symm. Verfahren, RSA)
- Weiterer Ablauf:
  - heute: Ende Kapitel 10
  - 28. Januar: Zentralübung, letztes Übungsblatt
  - 30./31. Januar: Vorlesung Kapitel 11
  - 4. Februar: Probeklausur (über 2. Teil der Vorlesung)
  - 6. Februar: Vorlesung Kapitel 12
  - 7. Februar: Vorlesung Wiederholung und Fragestunde
  - 8. Februar: letzte Tutorfragestunde
  - 21. Februar: Klausur

# Wiederholung letzte Vorlesung

## Die LU und LUP Zerlegung

### LU Zerlegung:

- Sei  $A \in \mathbb{R}^{n \times n}$  invertierbar und sei  $A^T$  diagonal dominant (d.h.  $|a_{ii}| \geq \sum_{i \neq j} |a_{ij}|$  für alle  $i = 1, \dots, n$ ).
- Dann gibt es eine untere Einheitsdreiecksmatrix  $L \in \mathbb{R}^{n \times n}$  und eine obere Dreiecksmatrix  $U \in \mathbb{R}^{n \times n}$  mit

$$A = LU.$$

### LUP Zerlegung:

- Sei  $A \in \mathbb{R}^{n \times n}$  invertierbar.
- Dann gibt es eine untere Einheitsdreiecksmatrix  $L \in \mathbb{R}^{n \times n}$ , eine obere Dreiecksmatrix  $U \in \mathbb{R}^{n \times n}$  sowie eine Permutationsmatrix  $P \in \mathbb{R}^{n \times n}$  mit

$$PA = LU.$$

# Wiederholung letzte Vorlesung

## Algorithmus LU Zerlegung

**Input:** Matrix  $A \in \mathbb{R}^{n \times n}$

**Output:** Matrizen  $L, U \in \mathbb{R}^{n \times n}$

**LU-Zerlegung(A):**

$U = 0; L = I_n;$

**for**  $k = 1$  **to**  $n$  {

$u_{kk} = a_{kk};$

**for**  $i = k + 1$  **to**  $n$  {

$l_{ik} = a_{ik} / u_{kk};$  // entspricht  $v_k$

$u_{ki} = a_{ki};$  // entspricht  $w_i^T$

}

**for**  $i = k + 1$  **to**  $n$  {

**for**  $j = k + 1$  **to**  $n$  {

$a_{ij} = a_{ij} - l_{ik} u_{kj};$

}

}

}

- Rekursion wurde durch Iteration ersetzt

- Aufwand:  $\Theta(n^3)$

## Die LU und

### LU Zerleg

- Sei  $A$   
 $|a_{ii}| \geq$
- Dann  
eine c

### LUP Zerle

- Sei  $A$
- Dann

eine obere Dreiecksmatrix  $U \in \mathbb{R}^{n \times n}$  sowie eine  
Permutationsmatrix  $P \in \mathbb{R}^{n \times n}$  mit

$$PA = LU.$$

$$L = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{pmatrix}$$
$$U = \begin{pmatrix} & & & \\ & & & \\ & & & \\ & & & \end{pmatrix}$$

# Wiederholung letzte Vorlesung

## Beispiel-Ablauf: LUP Zerlegung

Algorithmus:

**Input:** Matrix  $A$

**Output:** Matrix

**LU-Zerlegung:**

$U = 0; L =$

**for**  $k = 1$  **to**

$u_{kk} = a_{kk};$

**for**  $i = k +$

$l_{ik} = a_{ik}$

$u_{ki} = a_{ki}$

**}**

**for**  $j = k +$

**for**  $j =$

$a_{ij} =$

**}**

**}**

Die LU und

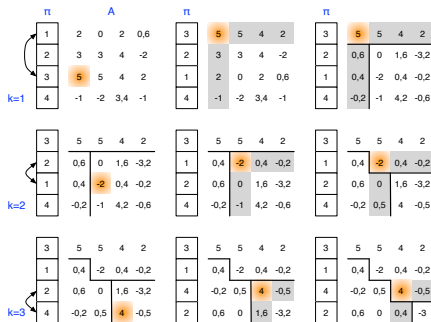
LU Zerleg

- Sei  $A$
- $|a_{ii}| \geq$
- Dann
- eine c

LUP Zerle

- Sei  $A$
- Dann
- eine obere Dreiecksmatrix  $U \in \mathbb{R}^{n \times n}$  sowie eine
- Permutationsmatrix  $P \in \mathbb{R}^{n \times n}$  mit

$$PA = LU.$$



20

11

# Wiederholung letzte Vorlesung

## Beispiel-Ablauf: LUP Zerlegung



Algorithmus:

**Input:** Matrix  $A$

**Output:** Matrix  $U$

**LU-Zerlegung:**

$U = 0; L = 0;$

**for**  $k = 1$  **to**  $n$

$u_{kk} = a_{kk};$

**for**  $i = k + 1$  **to**  $n$

$l_{ik} = a_{ik} / u_{kk};$

$u_{ki} = a_{ki} - l_{ik} u_{kk};$

**end for**

**for**  $j = k + 1$  **to**  $n$

$a_{ij} = a_{ij} - l_{ik} u_{kj};$

**end for**

**end for**

Problemstellung Least Squares

- **Gegeben:** Datenreihe mit  $m$  Datenpunkten

$$(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)$$

mit  $x_j, y_j \in \mathbb{R}$  für  $j = 1, \dots, m$

- **Erwartung:**  $y_j$  enthalten Meßfehler ( $j = 1, \dots, m$ )

- **Gesucht:** Funktion  $F : \mathbb{R} \rightarrow \mathbb{R}$  so daß **Approximationsfehler**

$$\eta_j = F(x_j) - y_j$$

für alle  $j = 1, \dots, m$  möglichst gering

- **Annahme:**  $F$  lässt sich darstellen als Summe von Basisfunktionen  $f_i$ ,

$$F(x) = \sum_{i=1}^n c_i f_i(x)$$

Die LU und

LU Zerleg

- Sei  $A$

$$|a_{ij}| \geq$$

- Dann

eine c

LUP Zerle

- Sei  $A$

- Dann

eine obere Dreiecksmatrix  $U \in \mathbb{R}^{n \times n}$  sowie eine

Permutationsmatrix  $P \in \mathbb{R}^{n \times n}$  mit

$$PA = LU.$$

# Wiederholung letzte Vorlesung

## Beispiel-Ablauf: LUP Zerlegung



Algorithmus:

Problemstellung Least Squares

- Gegeben: Datenreihe mit  $m$  Datenpunkten

Normalengleichung

Normalengleichung

- Sei  $A \in \mathbb{R}^{m \times n}$ ,  $y \in \mathbb{R}^m$  mit  $m \geq n$ .  $c \in \mathbb{R}^n$  ist eine Least-squares Lösung von  $\min_c \|Ac - y\|$  genau dann, wenn die Normalengleichung gilt,

$$A^T A c = A^T y.$$

- Insbesondere ist die Normalengleichung lösbar, falls  $\text{rank}(A) = n$ .

- die Matrix  $A^T A$  ist immer **symmetrisch**
- falls  $\text{rank}(A) = n$  ist  $A^T A$  auf **positiv definit** und damit invertierbar
- die Lösung der Normalengleichung ist dann

$$c = ((A^T A)^{-1} A^T) y$$

Die LU und

LU Zerleg

- Sei  $A$   
 $|a_{ii}| \geq$
- Dann  
eine  $c$

**Input:** Matrix  $A$

**Output:** Matrix

**LU-Zerlegung:**

$U = 0$ ;  $L =$

**for**  $k = 1$  **to**

$u_{kk} = a_{kk}$ ;

**for**  $i = k +$

$l_{ik} = a_{ik}$

$u_{ki} = a_{ki}$

**}**

**for**  $i = k +$

**for**  $j = i$

$a_{ij} =$

**}**

**}**

LUP Zerle

- Sei  $A$
- Dann  
eine obere Dreiecksmatrix  $U \in \mathbb{R}^{n \times n}$  sowie eine  
Permutationsmatrix  $P \in \mathbb{R}^{n \times n}$  mit

$$PA = LU.$$

# Programm heute

7 Fortgeschrittene Datenstrukturen

8 Such-Algorithmen

9 Graph-Algorithmen

10 Numerische Algorithmen

Matrizen

Lineare Gleichungen

Least squares Probleme

Fast Fourier Transformation



# Fourier Transformation

- Signale  $f : \mathbb{R} \rightarrow \mathbb{R}$  sind im **Zeitbereich**
  - jedem Zeitpunkt wird ein Wert (z.B. Amplitude) zugeordnet
  - Beispiele: Audio, Video, Bilder
- **Fourier-Transformation** führt Signal in **Frequenzbereich** über
  - betrachte Signal als Überlagerung von Sinuskurven
- **Anwendungen:**
  - Filterung von bestimmten Frequenzen
  - Datenkompression, z.B.
    - Audio: MP3
    - Video: MPEG
    - Bilder: JPEG
  - Bild- und Signalverarbeitung
- überaus nützlich, und effizient dank **FFT** (Fast Fourier Transformation)

# Polynome

## Polynome

Sei  $n \in \mathbb{N}$ . Eine Funktion  $A : \mathbb{R} \rightarrow \mathbb{R}$  der Form

$$A(x) = a_0 + a_1x + \dots + a_{n-1}x^{n-1}$$

heißt **Polynom** in  $x$  vom **Grad**  $n - 1$ . Hierbei heißen die  $a_i \in \mathbb{R}$ ,  $i = 1, \dots, n - 1$ , **Koeffizienten** von  $A$ ,  $a_{n-1} \neq 0$ .

- $A$  hat **Gradschranke**  $n$ , falls der Grad von  $A$  zwischen  $0$  und  $n - 1$  liegt.
- **Summen-Notation** für  $A$ :

$$A(x) = \sum_{j=0}^{n-1} a_j x^j$$

- Die **Koeffizientendarstellung** von  $A$  ist der Vektor

$$a = (a_0, a_1, \dots, a_{n-1})^T \in \mathbb{R}^n$$

# Operationen auf Polynomen

Seien  $A, B$  Polynome.

- **Auswertung von Polynom:** berechne für  $x_0 \in \mathbb{R}$

$$A(x_0)$$

- **Summe von Polynomen:** berechne Polynom  $C$  für  $x \in \mathbb{R}$  mit

$$C(x) = A(x) + B(x)$$

- **Multiplikation von Polynomen:** berechne Polynom  $C$  für  $x \in \mathbb{R}$  mit

$$C(x) = A(x)B(x)$$

# Auswertung von Polynomen

Sei  $A$  Polynom mit Koeffizientendarstellung  $a = (a_0, \dots, a_{n-1})$ .

- **Auswertung** von  $A$  an  $x_0 \in \mathbb{R}$  mit **Horner-Schema**:

$$A(x_0) = a_0 + x_0(a_1 + x_0(a_2 + \dots + x_0(a_{n-2} + x_0(a_{n-1}))) \dots))$$

- Aufwand:  $\Theta(n)$
- **Beispiel**:  $A(x) = 5x^2 - 3x + 4$  an Stelle  $x_0 = 2$ :

$$A(x_0) = 4 + x_0(-3 + x_0(5)) = 4 + 2(-3 + 2(5)) = 18$$

# Addition von Polynomen

Seien  $A, B$  Polynome mit Gradschranke  $n \in \mathbb{N}$ .

- Koeffizientendarstellungen  $a = (a_0, \dots, a_{n-1})$ ,  
 $b = (b_0, \dots, b_{n-1})$
- **Summe**  $A(x) + B(x)$  ist Polynom  $C$  mit Gradschranke  $n$ ,

$$C(x) = \sum_{j=0}^{n-1} c_j x^j$$

mit  $c_j = a_j + b_j$  für  $j = 0, \dots, n-1$ .

- Aufwand:  $\Theta(n)$
- **Beispiel:**  $A(x) = 5x^2 - 3x + 4$ ,  $B(x) = 4x - 2$ , dann ist

$$C(x) = 5x^2 + x + 2$$

# Multiplikation von Polynomen

Seien  $A, B$  Polynome mit Gradschranke  $n \in \mathbb{N}$ .

- Koeffizientendarstellungen  $a = (a_0, \dots, a_{n-1})$ ,  
 $b = (b_0, \dots, b_{n-1})$
- **Produkt**  $A(x)B(x)$  ist Polynom  $C$  mit Gradschranke  $2n - 1$ ,

$$C(x) = \sum_{j=0}^{2n-2} c_j x^j$$

mit

$$c_j = \sum_{k=0}^j a_k b_{j-k} \quad \text{für } j = 0, \dots, 2n - 2$$

- Aufwand:  $\Theta(n^2)$
- **Beispiel:**  $A(x) = 5x^2 - 3x + 4$ ,  $B(x) = 4x - 2$ , dann sind

$$c_0 = -8, \quad c_1 = 22, \quad c_2 = -22, \quad c_3 = 20, \quad c_4 = 0$$

# Faltung

Seien  $A, B$  Polynome mit Gradschranke  $n \in \mathbb{N}$ .

- Koeffizientendarstellung  $a = (a_0, \dots, a_{n-1})$ ,  
 $b = (b_0, \dots, b_{n-1})$
- Sei  $c = (c_0, \dots, c_{2n-2})$  die Koeffizientendarstellung des Produktpolynoms  $C(x) = A(x)B(x)$  mit

$$c_j = \sum_{k=0}^j a_k b_{j-k}$$

- $c$  wird auch **Faltung** der Vektoren  $a, b$  genannt,

$$c = a * b$$

- Faltung ist sehr häufige Operation auf Signalen (z.B. Anwendung von Filtern)

# Stützstellendarstellung

## Stützstellendarstellung von Polynom

Sei  $A$  Polynom mit Gradschranke  $n \in \mathbb{N}$ . Die **Stützstellendarstellung** von  $A$  ist ein Menge von  $n$  Stützstellenpaaren

$$\{(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1})\}$$

mit  $x_k$  paarweise verschieden für  $k = 0, \dots, n - 1$  sowie

$$y_k = A(x_k).$$

- Die Stützstellendarstellung ist nicht eindeutig
  - jede Menge von verschiedenen Punkten  $x_0, \dots, x_{n-1}$  kann verwendet werden



# Auswertung

Sei  $A$  Polynom mit Gradschranke  $n \in \mathbb{N}$  mit Koeffizienten  $a = (a_0, \dots, a_{n-1})$ .

- **Konversion** von Koeffizientendarstellung in Stützstellendarstellung erfolgt mittels **Auswertung**:
  - wähle Stützstellen  $x_0, \dots, x_{n-1}$
  - werte  $A(x_k)$  aus für  $k = 0, \dots, n - 1$
- Auswertung von  $n$  Stützstellen mit Horner-Schema:  $\Theta(n^2)$
- später: Auswertung mittels FFT:  $\Theta(n \log n)$

# Interpolation

## Eindeutigkeit interpolierendes Polynom

Für jede Menge von  $n$  Stützstellenpaaren

$$\{(x_0, y_0), \dots, (x_{n-1}, y_{n-1})\}$$

mit paarweise verschiedenen  $x_k$  existiert ein **eindeutiges Polynom**  $A(x)$  mit Gradschranke  $n$ , so daß gilt

$$y_k = A(x_k) \quad \text{für } k = 0, \dots, n - 1.$$

- **Konversion** von Stützstellendarstellung in Koeffizientendarstellung heißt **Interpolation**.

## Interpolation: Beweisidee

- Ähnlich zu Least squares Problemen kann Matrix-Gleichung aufgestellt werden:

$$\underbrace{\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{pmatrix}}_{=:V(x_0, \dots, x_{n-1})} \underbrace{\begin{pmatrix} a_0 \\ \vdots \\ a_{n-1} \end{pmatrix}}_{=:a} = \underbrace{\begin{pmatrix} y_0 \\ \vdots \\ y_{n-1} \end{pmatrix}}_{=:y}$$

- $V(x_0, \dots, x_{n-1})$  heißt **Vandermonde-Matrix**
- da  $x_k$  paarweise verschieden ist

$$\det V(x_0, \dots, x_{n-1}) = \prod_{0 \leq j < k \leq n-1} (x_k - x_j) \neq 0$$

- also ist  $V(x_0, \dots, x_{n-1})$  invertierbar und

$$a = V(x_0, \dots, x_{n-1})^{-1}y$$

ist die Koeffizientendarstellung von  $A$ .

# Interpolation: Algorithmen

Sei  $A$  Polynom mit  $n$  Stützstellenpaaren  $\{(x_0, y_0), \dots, (x_{n-1}, y_{n-1})\}$

- Berechnung mit **LUP-Zerlegung**:
  - $a = V(x_0, \dots, x_{n-1})^{-1}y$
  - Aufwand:  $\Theta(n^3)$
- Berechnung mit **Lagrange-Interpolationsformel**:

$$A(x) = \sum_{k=0}^{n-1} y_k \frac{\prod_{j \neq k} (x - x_j)}{\prod_{j \neq k} (x_k - x_j)}$$

- Aufwand:  $\Theta(n^2)$
- später: Berechnung mit IFFT:  $\Theta(n \log n)$

# Addition von Polynomen

Seien  $A, B$  Polynome mit Gradschranke  $n \in \mathbb{N}$ .

- Stützstellendarstellung für  $A$ :

$$\{(x_0, y_0), \dots, (x_{n-1}, y_{n-1})\}$$

- Stützstellendarstellung für  $B$  mit **selben**  $x_k$ :

$$\{(x_0, y'_0), \dots, (x_{n-1}, y'_{n-1})\}$$

- Stützstellendarstellung für  $C(x) = A(x) + B(x)$ :

$$\{(x_0, y_0 + y'_0), \dots, (x_{n-1}, y_{n-1} + y'_{n-1})\}$$

- Aufwand:  $\Theta(n)$

# Multiplikation von Polynomen

Seien  $A, B$  Polynome mit Gradschranke  $n \in \mathbb{N}$ .

- Produkt  $C(x) = A(x)B(x)$  hat Gradschranke  $2n$ 
  - benötigen  $2n$  Stützstellen
  - erweitere Darstellung von  $A, B$  auf  $2n$  Stützstellen vor Multiplikation
- Stützstellendarstellungen für  $A, B$ :

$$\{(x_0, y_0), \dots, (x_{2n-1}, y_{2n-1})\}$$

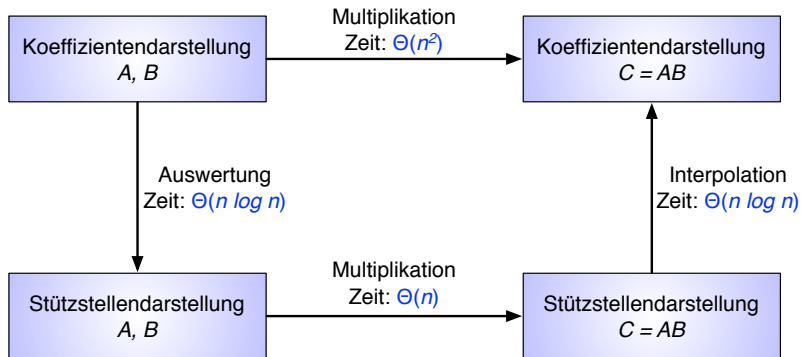
$$\{(x_0, y'_0), \dots, (x_{2n-1}, y'_{2n-1})\}$$

- Stützstellendarstellung für  $C(x)=A(x)B(x)$ :

$$\{(x_0, y_0 y'_0), \dots, (x_{2n-1}, y_{2n-1} y'_{2n-1})\}$$

- Aufwand:  $\Theta(n)$

# Effiziente Multiplikation von Polynomen



- Multiplikation von Polynomen in Koeffizientendarstellung via Stützstellendarstellung:  $\Theta(n \log n)$
- Auswertung und Interpolation mit FFT/IFFT

# Komplexe Einheitswurzeln

## Komplexe Einheitswurzeln

Sei  $n \in \mathbb{N}$ . Die komplexe  $n$ -te Einheitswurzel ist eine Zahl  $\omega \in \mathbb{C}$  mit

$$\omega^n = 1.$$

- Es gibt  $n$  komplexe  $n$ -te Einheitswurzeln:  $e^{2\pi ik/n}$  für  $k = 0, \dots, n - 1$ .



# Komplexe Einheitswurzeln

## Komplexe Einheitswurzeln

Sei  $n \in \mathbb{N}$ . Die **komplexe  $n$ -te Einheitswurzel** ist eine Zahl  $\omega \in \mathbb{C}$  mit

$$\omega^n = 1.$$

- Es gibt  $n$  komplexe  $n$ -te Einheitswurzeln:  $e^{2\pi ik/n}$  für  $k = 0, \dots, n - 1$ .

## $n$ -te Haupteinheitswurzel

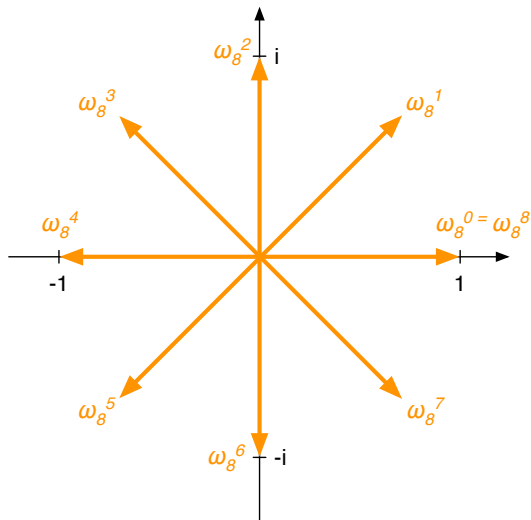
Die Zahl  $\omega_n \in \mathbb{C}$  mit

$$\omega_n = e^{2\pi i/n}$$

heißt  **$n$ -te Haupteinheitswurzel**.

- Die anderen komplexen  $n$ -ten Einheitswurzeln sind Potenzen von  $\omega_n$ .

## Beispiel: die komplexen 8-ten Einheitswurzeln



$$\omega_8 = e^{2\pi i/8}$$

# Diskrete Fourier Transformation

## Diskrete Fourier Transform

Sei  $A$  Polynom mit Gradschranke  $n$  mit Koeffizienten

$$a = (a_0, \dots, a_{n-1}),$$

$$A(x) = \sum_{j=0}^{n-1} a_j x^j.$$

Die Auswertung von  $A$  an den Stellen  $\omega_n^0, \omega_n^1, \dots, \omega_n^{n-1}$ , d.h.

$$y_k = A(\omega_n^k) = \sum_{j=0}^{n-1} a_j \omega_n^{kj}$$

für  $k = 0, \dots, n-1$ , heißt **Diskrete Fourier Transformation** des Koeffizientenvektors  $a$ . Kurz notiert auch als

$$y = DFT_n(a)$$

mit  $y = (y_0, \dots, y_{n-1})$ .

# Diskrete Fourier Transformation

- Aufwand der DFT:  $\Theta(n^2)$
- zur Multiplikation von Polynomen benötigen wir tatsächlich  $2n$  Koeffizienten statt  $n$
- Zur effizienten Berechnung der DFT verwenden wir einen Divide & Conquer Ansatz  $\rightarrow$  FFT

# Eigenschaften komplexer Einheitswurzeln

## Halbierungslemma

Sei  $n \in \mathbb{N}$  gerade Zahl. Dann gilt für  $k = 0, \dots, n-1$

$$(\omega_n^k)^2 = \omega_{n/2}^k = (\omega_n^{k+n/2})^2.$$

# Eigenschaften komplexer Einheitswurzeln

## Halbierungslemma

Sei  $n \in \mathbb{N}$  gerade Zahl. Dann gilt für  $k = 0, \dots, n-1$

$$(\omega_n^k)^2 = \omega_{n/2}^k = (\omega_n^{k+n/2})^2.$$

Beweis-Idee:

- Es ist

$$(\omega_n^k)^2 = (e^{2\pi i k/n})^2 = (e^{2\pi i \cdot 2/n})^k = \omega_{n/2}^k$$

# Eigenschaften komplexer Einheitswurzeln

## Halbierungslemma

Sei  $n \in \mathbb{N}$  gerade Zahl. Dann gilt für  $k = 0, \dots, n-1$

$$(\omega_n^k)^2 = \omega_{n/2}^k = (\omega_n^{k+n/2})^2.$$

Beweis-Idee:

- Es ist

$$(\omega_n^k)^2 = (e^{2\pi i k/n})^2 = (e^{2\pi i \cdot 2/n})^k = \omega_{n/2}^k$$

- Ausserdem:

$$(\omega_n^{k+n/2})^2 = \omega_n^{2k+n} = \omega_n^{2k} \omega_n^n = \omega_n^{2k} = (\omega_n^k)^2$$

$\approx \eta$

# DFT mit Divide and Conquer 1

Sei  $A$  Polynom mit Koeffizienten  $a = (a_0, \dots, a_{n-1})$ ,  $n \in \mathbb{N}$ .

- **Annahme:**  $n$  sei eine Zweier-Potenz ( $n = 2^m$ )
- **Divide-Schritt:** teile  $A$  auf entlang Koeffizienten mit geradem und ungeradem Index in  $A^{[0]}$  und  $A^{[1]}$ :

$$A^{[0]}(x) = a_0 + a_2x + a_4x^2 + \dots + a_{n-2}x^{n/2-1}$$

$$A^{[1]}(x) = a_1 + a_3x + a_5x^2 + \dots + a_{n-1}x^{n/2-1}$$

- $A^{[0]}$  und  $A^{[1]}$  haben Gradschranke  $n/2$
- **Conquer-Schritt:** es gilt

$$A(x) = A^{[0]}(x^2) + xA^{[1]}(x^2)$$



## DFT mit Divide and Conquer 2

Auswertung von  $A(x)$  für  $x = \omega_n^0, \omega_n^1, \dots, \omega_n^{n-1}$  also berechnet als:

- **Divide:** werte  $A^{[0]}$  und  $A^{[1]}$  mit Gradschranke  $n/2$  aus an

$$(\omega_n^0)^2, (\omega_n^1)^2, \dots, (\omega_n^{n-1})^2$$

Laut Halbierungslemma sind das nur  $n/2$  verschiedene Einheitswurzeln!

- **Rekursion:** weitere Aufteilung der Polynome in kleinere Probleme
- **Conquer:** führe Ergebnisse zusammen

$$A(x) = A^{[0]}(x^2) + xA^{[1]}(x^2)$$

Der Divide Schritt halbiert also (gemäß Halbierungslemma) die Problemgröße, und eine  $DFT_n$  wird mittels 2  $DFT_{n/2}$  berechnet.

## Algorithmus: FFT (rekursiv)

**Input:** Vektor  $a = (a_0, \dots, a_{n-1})^T \in \mathbb{R}^n$ ,  $n \in \mathbb{N}$  Zweier-Potenz

**Output:**  $DFT_n(a)$

**FFTrekursiv(a):**

```
1  if (n == 1) return a;
2   $\omega_n = e^{2\pi i/n}$ ;
3   $\omega = 1$ ;
4   $a^{[0]} = (a_0, a_2, a_4, \dots, a_{n-2})$ ;
5   $a^{[1]} = (a_1, a_3, a_5, \dots, a_{n-1})$ ;
6   $y^{[0]} = \text{FFTrekursiv}(a^{[0]})$ ; // divide
7   $y^{[1]} = \text{FFTrekursiv}(a^{[1]})$ ;
8  for k = 0 to n/2 - 1 { // conquer
9       $y_k = y_k^{[0]} + \omega y_k^{[1]}$ ;
10      $y_{k+n/2} = y_k^{[0]} - \omega y_k^{[1]}$ ;
11      $\omega = \omega \omega_n$ ; //  $\omega = \omega_n^k$ 
12 }
13 return y;
```

# FFT (rekursiv): Korrektheit und Laufzeit 1

- Zeile 1: DFT von einem Element ist das Element selbst,

$$y_0 = a_0 \omega_1^0 = a_0 \cdot 1 = a_0$$

# FFT (rekursiv): Korrektheit und Laufzeit 1

- Zeile 1: DFT von einem Element ist das Element selbst,

$$y_0 = a_0 \omega_1^0 = a_0 \cdot 1 = a_0$$

- Zeilen 6/7: wegen Halbierungslemma ist

$$y_k^{[0]} = A^{[0]}(\omega_{n/2}^k) = A^{[0]}(\omega_n^{2k})$$

$$y_k^{[1]} = A^{[1]}(\omega_{n/2}^k) = A^{[1]}(\omega_n^{2k})$$

# FFT (rekursiv): Korrektheit und Laufzeit 1

- Zeile 1: DFT von einem Element ist das Element selbst,

$$y_0 = a_0 \omega_1^0 = a_0 \cdot 1 = a_0$$

- Zeilen 6/7: wegen Halbierungslemma ist

$$y_k^{[0]} = A^{[0]}(\omega_{n/2}^k) = A^{[0]}(\omega_n^{2k})$$

$$y_k^{[1]} = A^{[1]}(\omega_{n/2}^k) = A^{[1]}(\omega_n^{2k})$$

- Zeile 9: berechnet  $y_0, y_1, \dots, y_{n/2-1}$  da

$$y_k = y_k^{[0]} + \omega_n^k y_k^{[1]} = A^{[0]}(\omega_n^{2k}) + \omega_n^k A^{[1]}(\omega_n^{2k}) = A(\omega_n^k)$$

## FFT (rekursiv): Korrektheit und Laufzeit 2

- Zeile 10: berechnet  $y_{n/2}, y_{n/2+1}, \dots, y_{n-1}$  da

$$y_{k+n/2} = y_k^{[0]} - \omega_n^k y_k^{[1]} = y_k^{[0]} + \omega_n^{k+n/2} y_k^{[1]}$$

da  $\omega_n^{n/2} = -1$  und damit  $\omega_n^{k+n/2} = -\omega_n^k$ , und somit weiter

$$\begin{aligned} y_{k+n/2} &= A^{[0]}(\omega_n^{2k}) + \omega_n^{k+n/2} A^{[1]}(\omega_n^{2k}) \\ &= A^{[0]}(\omega_n^{2k+n}) + \omega_n^{k+n/2} A^{[1]}(\omega_n^{2k+n}) = A(\omega_n^{k+n/2}) \end{aligned}$$

da  $\omega_n^{2k+n} = \omega_n^{2k}$ .

## FFT (rekursiv): Korrektheit und Laufzeit 2

- Zeile 10: berechnet  $y_{n/2}, y_{n/2+1}, \dots, y_{n-1}$  da

$$y_{k+n/2} = y_k^{[0]} - \omega_n^k y_k^{[1]} = y_k^{[0]} + \omega_n^{k+n/2} y_k^{[1]}$$

da  $\omega_n^{n/2} = -1$  und damit  $\omega_n^{k+n/2} = -\omega_n^k$ , und somit weiter

$$\begin{aligned} y_{k+n/2} &= A^{[0]}(\omega_n^{2k}) + \omega_n^{k+n/2} A^{[1]}(\omega_n^{2k}) \\ &= A^{[0]}(\omega_n^{2k+n}) + \omega_n^{k+n/2} A^{[1]}(\omega_n^{2k+n}) = A(\omega_n^{k+n/2}) \end{aligned}$$

da  $\omega_n^{2k+n} = \omega_n^{2k}$ .

- also tatsächlich  $y = DFT_n(a)$

## FFT (rekursiv): Korrektheit und Laufzeit 2

- **Zeile 10:** berechnet  $y_{n/2}, y_{n/2+1}, \dots, y_{n-1}$  da

$$y_{k+n/2} = y_k^{[0]} - \omega_n^k y_k^{[1]} = y_k^{[0]} + \omega_n^{k+n/2} y_k^{[1]}$$

da  $\omega_n^{n/2} = -1$  und damit  $\omega_n^{k+n/2} = -\omega_n^k$ , und somit weiter

$$\begin{aligned} y_{k+n/2} &= A^{[0]}(\omega_n^{2k}) + \omega_n^{k+n/2} A^{[1]}(\omega_n^{2k}) \\ &= A^{[0]}(\omega_n^{2k+n}) + \omega_n^{k+n/2} A^{[1]}(\omega_n^{2k+n}) = A(\omega_n^{k+n/2}) \end{aligned}$$

da  $\omega_n^{2k+n} = \omega_n^{2k}$ .

- also tatsächlich  $y = DFT_n(a)$
- **Laufzeit:** ohne Rekursion  $\Theta(n)$ , Rekursionsgleichung:

$$T(n) = 2T(n/2) + \Theta(n) = \Theta(n \log n)$$



# Interpolation mit komplexen Einheitswurzeln

Sei  $A$  Polynom mit Koeffizienten  $a = (a_0, \dots, a_{n-1})$ .

- DFT lässt sich auch als Matrix-Gleichung mit Vandermonde-Matrix  $V_n$  mit  $\omega_n$ -Potenzen als Einträge:

$$\underbrace{\begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix}}_{=y} = \underbrace{\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \omega_n^2 & \cdots & \omega_n^{n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \cdots & \omega_n^{(n-1)(n-1)} \end{pmatrix}}_{=:V_n} \underbrace{\begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix}}_{=a}$$

- **Inverse Operation**  $a = V_n^{-1}y$  entspricht **Interpolation**, auch geschrieben als

$$a = IDFT_n(y)$$

# Inverse Diskrete Fourier Transformation

- Es ist also  $IDFT_n(y) = V_n^{-1}y$
- $V_n^{-1}$  hat Einträge  $\omega_n^{-kj}/n$  an Stelle  $(j, k)$  für  $j, k = 0, \dots, n-1$
- $IDFT_n(y)$  ist also für  $j = 0, \dots, n-1$

$$a_j = \frac{1}{n} \sum_{k=0}^{n-1} y_k \omega_n^{-kj}$$

- IFFT lässt sich aus FFT gewinnen durch
  - vertauschen von  $a$  und  $y$
  - ersetzen von  $\omega_n$  durch  $\omega_n^{-1}$
  - jedes Element mit  $\frac{1}{n}$  multiplizieren
- IFFT Laufzeit also auch  $\Theta(n \log n)$

# Faltungstheorem

## Faltungstheorem

Seien  $a, b \in \mathbb{R}^n$  mit  $n \in \mathbb{N}$  und  $n = 2^m$ . Dann ist

$$a * b = IDFT_{2n}(DFT_{2n}(a) \cdot DFT_{2n}(b))$$

- $a, b$  müssen mit Nullen zur Länge  $2n$  aufgefüllt werden
- $\cdot$  ist das komponentenweise Produkt zweier Vektoren
- Gesamtlaufzeit:  $\Theta(n \log n)$

## Algorithmus: FFT (rekursiv)

**Input:** Vektor  $a = (a_0, \dots, a_{n-1})^T \in \mathbb{R}^n$ ,  $n \in \mathbb{N}$  Zweier-Potenz

**Output:**  $DFT_n(a)$

**FFTrekursiv(a):**

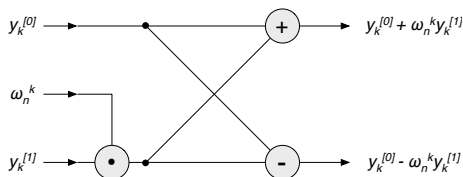
```
1  if (n == 1) return a;
2   $\omega_n = e^{2\pi i/n}$ ;
3   $\omega = 1$ ;
4   $a^{[0]} = (a_0, a_2, a_4, \dots, a_{n-2})$ ;
5   $a^{[1]} = (a_1, a_3, a_5, \dots, a_{n-1})$ ;
6   $y^{[0]} = \text{FFTrekursiv}(a^{[0]})$ ; // divide
7   $y^{[1]} = \text{FFTrekursiv}(a^{[1]})$ ;
8  for k = 0 to n/2 - 1 { // conquer
9       $y_k = y_k^{[0]} + \omega y_k^{[1]}$ ;
10      $y_{k+n/2} = y_k^{[0]} - \omega y_k^{[1]}$ ;
11      $\omega = \omega \omega_n$ ; //  $\omega = \omega_n^k$ 
12 }
13 return y;
```

# FFT effizienter

- **FFTrekursiv** berechnet in **Zeile 9,10** zweimal  $\omega y_k^{[1]}$
- effizienter mit Hilfsvariable  $t$ :

```
for  $k = 0$  to  $n/2 - 1$  {  
     $t = \omega y_k^{[1]}$ ;  
     $y_k = y_k^{[0]} + t$ ;  
     $y_{k+n/2} = y_k^{[0]} - t$ ;  
     $\omega = \omega \omega_n$ ;  
}
```

- dieses Schema ist bekannt als **Butterfly-Operation**:



# FFT: weitere Optimierungen

Zusätzlich zu Butterfly-Operationen:

- Entrekursivierung möglich mittels Bitreversierung
- Komplexität immer noch  $\Theta(n \log n)$ , aber kleinere Konstanten
- Implementierung als parallele Schaltkreise möglich

# Zusammenfassung

7 Fortgeschrittene Datenstrukturen

8 Such-Algorithmen

9 Graph-Algorithmen

10 Numerische Algorithmen

- Matrizen

- Lineare Gleichungen

- Least squares Probleme

- Fast Fourier Transformation