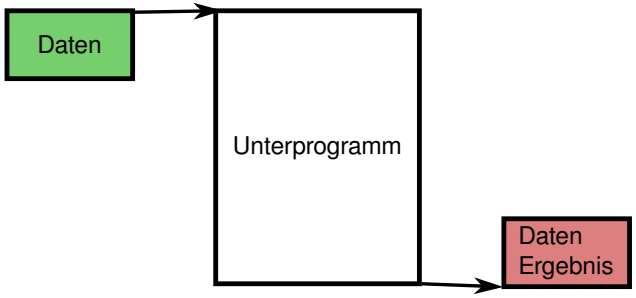
# Inhaltsverzeichnis

# 1 Lösen von Problemen mit dem Computer

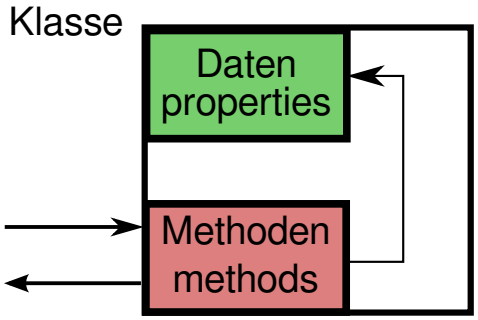
## 1.1 Wahl der Programmiersprache

Je nach Anforderungen Wahl der Programmiersprache.

Auswahlkriterien:




- Prozedural:
  - Meist weniger Planungsaufwand
  - Weniger Flexibilität und Erweiterbarkeit
  - Meist kürzere Laufzeit




- Objektorientiert:
  - Für größere Projekte
  - Teamarbeit
  - Mehr Planungsaufwand, wird später meist eingespart
  - Programme sind umfangreicher und haben längere Laufzeit

- Compiler: C, C++, FORTRAN
  - Kompakte und effiziente Binärdatei
  - Quellcode muss beim Ablauf nicht vorhanden sein
  - Plattform- oder Systemwechsel:
    Oft Anpassung mit neucompilation notwendig
  - Hauptanwendung: Numerik
- Scriptsprachen: Perl, python, haskel, ruby, ...
  - Programme laufen meist langsamer
  - Programm liegt meist im Quellcode vor
  - Plattform- oder Systemwechsel:
    Anpassung meist einfacher, oft platformunabhängig
  - Anwendung: Manipulation von Daten, Text, Datenbanken, ...
    Sehr umfangreiche Bibliotheken verfügbar

## 1.2 Problemstellung und Lösung

1. Verstehen des Problems und seiner Lösungsmöglichkeiten
2. Planung
   - Suchen nach bereits vorhandenen Programmen, Bibliotheken, ...
   - Auswahl der Programmiersprache
   - Entwicklung der eigenen Daten- und Programmstruktur
     Möglichst modular: Zerlegen in kleinere Teilprobleme und -aufgaben
     z.B.: Berechnung von der grafischen Ausgabe trennen
   - Skizzieren des Lösungswegs (Flussdiagramm, ...)

   Längere Planungszeit $\longrightarrow$ verkürzt meist die folgenden Schritte
3. Programmieren
4. Testen
5. Verwendung
6. Wartung und Pflege

# 2 Speichern und archivieren von Daten

## Als Datei

Unzählige Dateiformate

- Text und Zahlen direkt im ASCII-Code

  Beständiges Format, auf allen Plattformen lesbar  mehr Speicherplatz

- Binärformat

  Kompakt  von Plattform und Implementation abhängig

- ASCII-Code als Containment

  z.B.: XML: Bildet Baumstruktur durch “tags”

  Daten erhalten Ordnung  Lesen aufwändig (Bibliotheksfunktionen)  mehr Speicherplatz

- Komprimiert

  Jedes der vorigen Formate möglich, spart Speicherplatz  Rechenzeit

- Veschlüsselt

  Jedes der vorigen Formate möglich  Rechenzeit

# In einer Datenbank

Typ der Datenbank z.B.: SQL, ldap, ...

Daten an vielen Stellen gleichzeitig verfügbar  Aufwand

# 3 Darstellung von Zahlen

## 3.1 Ganze Zahlen (integer numbers)

Ganze Zahlen werden in Binärdarstellung verarbeitet.

z.B. 8 Bit $\rightarrow$ 2 Halbbytes (nibbles) zu je 4 Bit

| Binär | Hexadezimal | Dezimal |
|---|---|---|
| 1000 0101<br>MSB LSB | 0x85 | 133 |

MSB: most significant bit LSB: least significant bit

Die Reihenfolge der Bytes im Speicher ist maschinenabhängig.

- Big-Endian

  Byte mit den höchstwertigen Bits (MSB) wird an der kleineren Speicheradresse abgelegt

- Little-Endian

  Byte mit den niederstwertigen Bits (LSB) wird an der kleineren Speicheradresse abgelegt

Details siehe: z.B.: Wikipedia: Byte-Reihenfolge

Die Speicherlänge (8, 16, 32, 64, . . . Bit) und die Interpretation (Vorzeichen: signed/unsigned) bestimmen den Wertebereich.

z.B. 4 Bit

| Binär | Hex | unsigned | signed |
|---|---|---|---|
| 0000 | 0x0 | 0 | 0 |
| 0001 | 0x1 | 1 | 1 |
| 0010 | 0x2 | 2 | 2 |
| 0011 | 0x3 | 3 | 3 |
| 0100 | 0x4 | 4 | 4 |
| 0101 | 0x5 | 5 | 5 |
| 0110 | 0x6 | 6 | 6 |
| 0111 | 0x7 | 7 | 7 |
| 1000 | 0x8 | 8 | -8 |
| 1001 | 0x9 | 9 | -7 |
| 1010 | 0xA | 10 | -6 |
| 1011 | 0xB | 11 | -5 |
| 1100 | 0xC | 12 | -4 |
| 1101 | 0xD | 13 | -3 |
| 1110 | 0xE | 14 | -2 |
| 1111 | 0xF | 15 | -1 |

MSB: Vorzeichen-Bit

8 Bit $\rightarrow 2^8 = 256$ Möglichkeiten

unsigned: 0 . . . 255

signed: -128 . . . 127

$2^n$ Bit $\rightarrow 2^n$ Möglichkeiten

unsigned: $0 \ldots 2^n - 1$

signed: $-2^{n-1} \ldots 2^{n-1} - 1$

Umrechnen von Dezimal in Binär: z.B.:

$18_{10}$

18 / 2 = 9, 0 Rest LSB

9 / 2 = 4, 1 Rest

4 / 2 = 2, 0 Rest

2 / 2 = 1, 0 Rest

1 / 2 = 0, 1 Rest MSB

$18_{10} = 10010_2$

## 3.2 Reelle Zahlen (real numbers)

Auch in einem endlichen Intervall überabzählbar unendlich viele Zahlen

### Gleitkommazahl (floating point number)

Binärdarstellung mit endlicher Speicherlänge:

Nicht jede reelle Zahl kann dargestellt werden

$\longrightarrow$ nur angenäherte Darstellung möglich

$\longrightarrow$ Rundungsfehler

Umrechnen einer reellen Zahl in binäre Darstellung:

$0.1_{10}$

0.1 * 2 = 0.2 - 0 MSB
0.2 * 2 = 0.4 - 0
0.4 * 2 = 0.8 - 0
0.8 * 2 = 1.6 - 1
0.6 * 2 = 1.2 - 1
0.2 * 2 = 0.4 - 0
...

$0.1_{10} = 0.0001\dot{1}\dot{0}\dot{0}\dot{1}\dot{1}_2$

$0.5_{10}$

0.5 * 2 = 1.0 - 1 MSB
0.0 * 2 = 0.0 - 0

$0.5_{10} = 0.1_2$

## Speicherformat

- Grundrechnungsarten $+ - * \ \hat{}$, mathematische Funktionen (Wurzel, Logarithmus) sollen implementiert sein.
- Auf jedem Computer soll bei einer bestimmten Speicherlänge (Genauigkeit) bitgenau das selbe Ergebnis erhalten werden.
- Grenzfälle ($\frac{2}{0},\ \log(0),\ \ldots$) sollen die “üblichen“ Ergebnisse liefern. d.h. Darstellungen von $-\infty,\ -0,\ +0,\ +\infty$ sollen implementiert sein.
- Fehlerbehandlung soll möglich sein $\sqrt{-1}, \log(-1)$ oder unbestimmte Ausdruck $\frac{0}{0},\ \infty - \infty$ sollen dargestellt werden können.
  $\longrightarrow$NaN (Not a Number)
- Rundungen müssen exakt definiert sein.

Der Standard IEEE 754 erfüllt solche Forderungen.

## Darstellung einer Gleitkommazahl

$$x = s \cdot m \cdot b^e \tag{1}$$

s. . . Vorzeichen ($\pm 1 \rightarrow$1 Bit)

b. . . Basis (für normalisierte Gleitkommazahlen nach IEEE 754 ist b=2)

e. . . Exponent (r Bits)

m. . . Mantisse (p Bits), (Signifikant)

Die Schreibweise einer Zahl gemäß Gleichung 1 ist:

$$\underbrace{\pm d_0 \,.\, d_1 \; d_2 \ldots d_{p-1}}_{m} \times b^e \tag{2}$$

$d_i$. . . Ziffern der Mantisse, $\times$ . . . Teil der Notation, $\cdot$ . . . Multiplikation, . . . . Kommapunkt

sie stellt die Zahl:

$$\pm \left(d_0 + d_1 \cdot b^{-1} + \ldots + d_{p-1}\right) \cdot b^{-(p-1)} \times b^e \quad (0 < d_i < b) \tag{3}$$

dar.

## Normierung

Z.B.: $0.01 \times 10^{1}$ und $1.00 \times 10^{-1} \longrightarrow 0.1$

Ist die erste (führende) Ziffer ($d_0$ in Gleichung 3) ungleich Null $\longrightarrow$ **normierte Darstellung**

$1.00 \times 10^{-1}$: normiert

$0.01 \times 10^{1}$: nicht normiert

Forderung einer normierten Darstellung $\longrightarrow$ eindeutige Darstellung

$\longrightarrow$ Darstellung der Zahl Null unmöglich $\longrightarrow$

Für Sonderfälle ($-\infty,\ -0,\ +0,\ +\infty$) werden spezielle Bitmuster verwendet

## Rundungen

1. Binäre Rundungen:

   Zur nächstgelegenen darstellbaren Zahl

2. Genau in der Mitte zwischen zwei darstellbaren Zahlen:

   Niederwertigste Bit der Mantisse wird 0 $\longrightarrow$ passiert statistisch in 50% der Fälle

   $\longrightarrow$ statistische Drift in langen Rechnungen wird vermieden (D. Knuth)

3. Zusätzliche Rundungen:

| $\rightarrow +\infty$ | $\rightarrow -\infty$ | $\rightarrow 0$ |
|---|---|---|
| immer aufrunden | immer abrunden | immer betragsmäßig verkleinern |

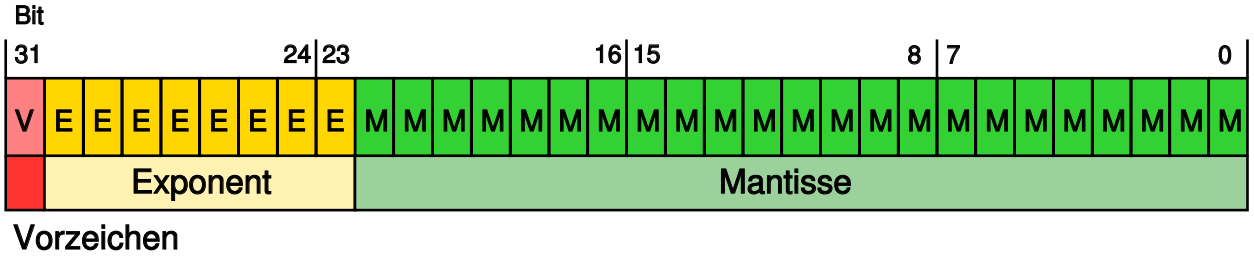
# Zahlenformate

Neben anderen Formaten sind standardmäßig

einfach genau (single precision) und doppelt genau (double precision)

implementiert.

| Typ | Größe (1+r+p) | Exponent (r) | Mantisse (p) | Werte des Exponenten (e) |
|---|---|---|---|---|
| single | 32 bit | 8 bit | 23 bit | $-126 \leq e \leq 127$ |
| double | 64 bit | 11 bit | 52 bit | $-1022 \leq e \leq 1023$ |




Darstellung einer einfach genauen Gleitkommazahl nach IEEE-754

# Zahlenbereich

| Typ | $\epsilon$ | Stellen | betragsmäßig kleinste Zahl | Größte Zahl |
|---|---|---|---|---|
| single | $2^{-(23+1)} \approx 5.96 \cdot 10^{-8}$ | $7-8$ | $2^{-23}2^{-126} = 2^{-149} \approx 1 \cdot 10^{-45}$ | $(1-2^{-24})2^{128} \approx 3.403 \cdot 10^{38}$ |
| double | $2^{-(52+1)} \approx 1.11 \cdot 10^{-16}$ | $15-16$ | $2^{-52}2^{-1022} = 2^{-1074} \approx 5 \cdot 10^{-324}$ | $(1-2^{-53})2^{1024} \approx 1.798 \cdot 10^{308}$ |

Die betragsmäßig kleinsten Zahlen sind nicht normalisiert.

$\epsilon$. . . relativer Abstand zweier Gleitkommazahlen

Stellen. . . Anzahl der Dezimalstellen die ohne Genauigkeitsverlust gespeichert werden können

siehe z.B. `http://de.wikipedia.org/wiki/IEEE_754`

# Rundungsfehler

David Goldberg: What Every Computer Scientist Should Know About Floating Point Arithmetic, 1991

`http://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html#674`

`http://docs.oracle.com/cd/E19957-01/806-3568/ncgTOC.html`

Eine reelle Zahl kann nicht exakt als Gleitkommazahl dargestellt werden:

1. Der häufigste Fall: Siehe Konversion der Dezimalzahl 0.1
   $0.1_{10}$ hat eine endliche Dezimaldarstellung, jedoch eine periodische Binärdarstellung. D.h. für $b = 2$ liegt $0.1$ genau zwischen zwei Gleitkommazahlen und wird durch keine exakt dargestellt.
2. Weniger häufig: Die reelle Zahl liegt außerhalb des darstellbaren Bereichs:

$$1.0 \times b^{e_{min}} > x > b \times b^{e_{max}}$$

Beispiel: 10.1 - 9.93 ($b = 10,\ p = 3$, siehe Gleichung 1)

$x = 1.01 \times 10^1$ Richtiges Ergebnis: $.17$

$y = 0.99 \times 10^1$ Differenz: 3 Einheiten der letzten Stelle

$x - y = 0.02 \times 10^1$ **Maximal möglicher Fehler?**

Gleitkommadarstellung mit dem Parameter $b$, Berechnung der Differenz auf $p$ Stellen:

Maximaler relativer Fehler: $b - 1$

Abhilfe:

guard digit: Beim Berechnen wird um eine Stelle mehr verwendet.

Beispiel: ($b = 10,\ p = 3$)

$x = 1.01|0 \times 10^1$

$y = 0.99|3 \times 10^1$

$x - y = 0.017 \times 10^1$

Richtiges Ergebnis: $0.17$

relativer Fehler: $\frac{1 \times 10^{-2}}{1 \times 10^{-3}} = 10$

## Auslöschungsfehler

Subtraktion zweier fast gleichgroßer Zahlen:

Die höchsten Stellen der Operanden stimmen überein und ”löschen“ einander aus.

1. Der günstigere Fall (benign cancelation)

   Subtraktion von exakt bekannten Größen

2. Der ungünstigste Fall (catastrophic cancelation):

   Die Operanden (Subtraktion) sind mit Rundungsfehlern behaftet.

   Beispiel: Wurzel der quadratischen Gleichung

$$a \cdot x^2 + b \cdot x + c = 0 \quad r_{12} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

   Fehler: Ausdruck $b^2 - 4ac \longrightarrow b^2$ und $4ac$ meist mit Rundungsfehlern behaftet.

   Subtraktion $\longrightarrow$Auslöschung $\longrightarrow$genaue Stellen verschwinden

   $\longrightarrow$Stellen mit Rundungsfehler bleiben übrig

   (Beispiel: genau1.py b=3.34, a=1.22, c=2.28)

Weitere Beispiele für Auslöschungsfehler:

- $(x^2 - y^2)$ die Umformung in $(x + y) \cdot (x - y)$ wird i. A. genauer berechnet.

  (Beispiel: genau3.py und genau3.c)

- Die Fläche eines Dreiecks mit den Seiten a, b, c:

  $A = \sqrt{s(s-a)(s-b)(s-c)}, \quad s = (a+b+c)/2$ für $a \approx b + c$

  $\longrightarrow$ Auslöschungsfehler

  Besser: $\frac{\sqrt{(a+(b+c))\cdot(c-(a-b))\cdot(c+(a-b))\cdot(a+(b-c))}}{4} \quad a \geq b \geq c$ (Beispiel: heron.py)

- Zinseszinsrechnung: $(1 + x)^n \quad x << 1$

  z.B. €100 mit 6% Zinsen p. A.; täglich abgerechnet, $K_0 = 100, \quad n = 365, \quad z = 0.06$

  $K_e = K_0 \frac{(1+z/n)^n - 1}{z/n}$

  Rundungsfehler durch 1+(kleine Zahl) wird durch die n-te Potenz verstärkt. Die Umformung $(1 + x)^n = e^{n \ln (1+x/n)}$ löst das Problem nur bedingt. (Beispiel: zins.py)

Weitere Beispiele für Ungenauigkeiten:

- Quotient komplexer Zahlen: $\frac{a+ib}{c+id} = \frac{ac+bd}{c^2+d^2} + i\frac{bc-ad}{c^2+d^2}$

  Sehr kleine Werte von c und d $\longrightarrow$ Überlauf (overflow) (Beispiel: cmplx.py)

$$\frac{a+ib}{c+id} = \begin{cases} \frac{a+b(d/c)}{c+d(d/c)} + i\frac{b-a(d/c)}{c+d(d/c)} & |d| < |c| \\ \frac{b+a(c/d)}{d+c(c/d)} + i\frac{-a+b(c/d)}{d+c(c/d)} & |d| \geq |c| \end{cases}$$

- Assoziativ- und Distributivgesetz für ungünstige Werte nicht mehr erfüllt:

$$u + (v + w) \neq (u + v) + w \qquad (u \cdot v) + (u \cdot w) \neq u \cdot (v + w)$$

(Beispiel: assoc.py, 1_10.py, 3_7.py)

# 4 Datentypen

## 4.1 Einfache Datentypen

### Zeichen (character):

Fast immer 1 Byte = 8 bit lang

**ASCII-Code:** 7-Bit Code

Ordnet einem Zeichen eine Zahl zu. Auf max 128 Zeichen beschränkt.

0-31: Steuerzeichen (nicht druckend) 32-127: Buchstaben und Zeichen (**`man ascii`**)

Erweiterung auf 8 Bit $\longrightarrow$ zusätzliche 128 Zeichen

$\longrightarrow$ verschiedene nationale Zeichen und Sonderzeichen

**UTF-8:** ASCII-kompatibler Multibyte-Code

Jedem Unicode-Zeichen wird eine speziell kodierte Bytekette variabler Länge zugeordnet

Theoretisch $2^{42}$, durch Unicode ”nur” 1 114 112 Möglichkeiten verfügbar.

## Ganze Zahlen (integer):

Je nach Implementation (Programmiersprache): Mit (signed) und ohne Vorzeichen (unsigned)
Speicherlänge: 2 - 8 Byte

Objektorientierte Programmiersprachen: Beliebig lange integer-Zahlen möglich.

## Gleitkommazahlen (floating point):

Meist nach IEEE754-Standard
Oft wird nach einfach genau (**`float, real`**) und doppelt genau (**`double, double precision`**) unterschieden.
Speicherlänge: 4 oder 8 Byte

FORTRAN95, python: Gleitkommazahlen mit vorgegebener (beliebiger) Genauigkeit

## Zeiger (pointer):

Einige Programmiersprachen unterstützen Zeiger (=Speicheradresse eines Objekts) als Datentyp.

Speicherlänge: 32 oder 64 Bit je nach Architektur

## 4.2 Erweiterte Datentypen

### Feld (array):

Aneinanderreihung von $n$ Elementen des selben Datentyps im Speicher

Zugriff durch den Index: Nummer des Elements (ganze Zahl)

Zählung: $1 \ldots n$ oder $0 \ldots n-1$, FORTRAN: auch negative Indizes

```
int A[20];
A[1]=5;
```

```
INTEGER A(-20,20)
A(1)=0.5
```

### Zeichenkette (string):

Meist ein Array von Zeichen

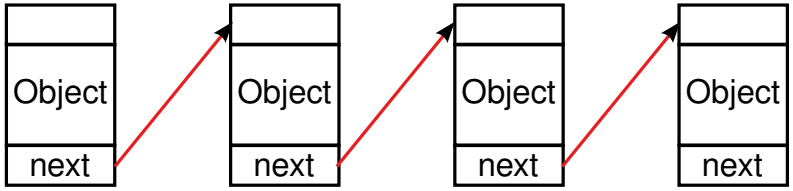
C, C++: Null-Zeichen ($\backslash 0$) markiert das Ende der Zeichenkette

## Liste (list):

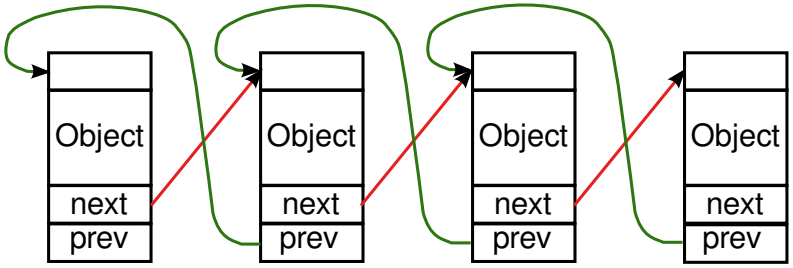
Aneinanderreihung von Objekten beliebigen Datentyps

Zugriff durch den Index: Nummer des Elements (ganze Zahl)

Implementation in C, C++:




einfach verkettete Liste




doppelt verkettete Liste

Vorteil:

Elemente können einfach ohne umkopieren hinzugefügt und entfernt werden

Nachteil:

Mehr Speicher und Zugriffszeit erforderlich

## Dictionary (assoziative Liste):

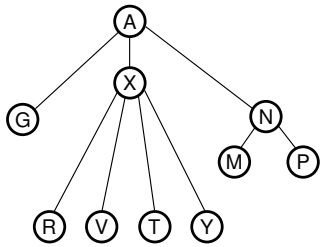
Aneinanderreihung von Objekten beliebigen Datentyps

Index: beliebige unveränderbare Datentypen (z.B. strings)

| Key (Index) | → | Wert |
|---|---|---|
| `'eins'` | → | `'one'` |
| `'gerade'` | → | `[2,4,6,8,10]` |
| `'geheim'` | → | `4711` |
| `'Vorname'` | → | `'Georg'` |

# 4.3 Andere Einteilung

- Array
- Liste
- Baum




- Stapel (Stack): LiFo
- Queue: FiFo

Siehe z.B.: Robert Sedgewick “Algorithmen in C“, Addison-Wesley (1992)

# 5 Kondition

## Numerische Mathematik:

Kondition: Abhängigkeit der Lösung eines Problems von der Störung der Eingangsdaten

Konditionszahl: Maß für diese Abhängigkeit

Faktor für die Verstärkung der Eingangsfehler im ungünstigsten Fall.

Unabhängig von konkreten Lösungsverfahren, Abhängig vom mathematischen Problem

Man unterscheidet zwischen: Kondition, Stabilität und Konsistenz

Mathematisches Problem: $f : \mathbb{K}^n \to \mathbb{K}^m$

$x$: Eingabedaten, $\tilde{f}$: numerische Algorithmus, $\tilde{x}$: gestörte Eingabedaten

Fehler: $\|f(x) - \tilde{f}(\tilde{x})\|$ (Norm)

Dreiecksungleichung:

$$\|f(x) - \tilde{f}(\tilde{x})\| = \|f(x) \underbrace{-f(\tilde{x}) + f(\tilde{x})}_{0} - \tilde{f}(\tilde{x})\| \leq \|f(x) - f(\tilde{x})\| + \|f(\tilde{x}) - \tilde{f}(\tilde{x})\|$$

$\|f(x) - f(\tilde{x})\|$: Kondition des Problems (Eigenschaft der Problems)

$\|f(\tilde{x}) - \tilde{f}(\tilde{x})\|$: Stabilität des Problems (Eigenschaft des Algorithmus)

Relative Konditionszahl:

$$\kappa_{rel} = \frac{\|\frac{d}{dx} f(x)\| \|x\|}{\|f(x)\|}$$

Ist $f : \mathbb{R} \to \mathbb{R}$ differenzierbar $\longrightarrow$Taylorreihe ohne Terme höhere Ordnung:

$$\kappa_{rel} = \left| \frac{f'(x)}{f(x)} x \right|$$

$\kappa_{rel} >> 1$: schlecht konditioniertes Problem

sonst: gut konditioniertes Problem

$\kappa_{rel} \to \infty$: schlecht gestelltes Problem

Computerprogramm:

Bereits ”verfälschte” Eingangsdaten: Umwandlung relle Zahlen $\longrightarrow$Gleitkommazahlen

Schlecht konditioniertes Problem: Algorithmus liefert keine brauchbaren Ergebnisse

Gegebenes Problem:

schlechte Kondition $\longrightarrow$ umformulieren

Äquivalente Umformulierung eines Problems zur Konditionsverbesserung:Vorkonditionierung

Verbesserung der Kondition: eingehende Zahlenwerte auf gut verarbeitbare Zahlenwerte normieren (skalieren)

## Beispiele:

Multiplikation: $x_1 \cdot x_2$: Abbildung $f : \mathbb{R}^2 \to \mathbb{R}$ mit $f(a,b) = a \cdot b$

$\frac{df}{d(a,b)} = \left(\frac{df}{da}, \frac{df}{db}\right) = (b,a)$ $\quad \|\cdot\|_2 \ldots$ 2er Norm

$\kappa_{rel} = \frac{\|\frac{d}{d(a,b)} f(a,b)\|_2 \cdot \|(a,b)\|_2}{|f(a,b)|} = \frac{\|f'(a,b)\|_2 \cdot \|(a,b)\|_2}{|f(a,b)|} = \frac{\sqrt{b^2+a^2} \cdot \sqrt{a^2+b^2}}{|ab|} = \frac{a^2+b^2}{|ab|}$

$a \approx b$: gut konditioniert

$a >> b$: schlecht konditioniert z.B.: $a = 10^{10}$ $b = 10^{-10}$ $\kappa_{rel} \doteq 10^{20}$

Addition: $x_1 + x_2$: Abbildung $f : \mathbb{R}^2 \to \mathbb{R}$ mit $f(a,b) = a + b$

$\frac{df}{d(a,b)} = \left(\frac{df}{da}, \frac{df}{db}\right) = (1,1)$

$\kappa_{rel} = \frac{|a|+|b|}{|a+b|}$

$|a+b| \approx 0$ oder $a \approx -b \longrightarrow$ schlecht konditioniert

Siehe:

`http://de.wikipedia.org/wiki/Kondition_%28Mathematik%29`

# 6 Verschlüsselung

Es gibt viele Gründe warum Daten “verschlüsselt” werden:

1. Datenkomprimierung

2. Prüfsummen (Hashes)

3. Geheimniskrämerei (Kryptologie)

## 6.1 Entropie

Die Entropie in der Informationstheorie ist ein Maß für den mittleren Informationsgehalt (Informationsdichte) einer Nachricht

Definition (nach Shannon):

Die Entropie $H$ einer diskreten, gedächtnislosen Quelle (diskreten Zufallsvariable) $X$ über einem endlichen Alphabet $Z = \{z_1, z_2, \ldots, z_m\}$

(kurz: eine Botschaft aus $z_m$ verschiedenen Zeichen):

Man ordnet jeder Wahrscheinlichkeit $p$ eines Ereignisses $z_i$ seinen Informationsgehalt $I(p) = -\log_2 p$ zu

Die Entropie eines Zeichens ist definiert als der Erwartungswert des Informationsgehalts:

$$H_1 = \sum_{z \in Z} p_z \cdot I(p_z) = -\sum_{z \in Z} p_z \cdot \log_2 p_z$$

(Beispiel: `./analyze_n.py -V entropie.txt`)

Eigenschaften der Entropie:

- Die Entropie wird maximal, wenn alle Zeichen mit der gleichen Wahrscheinlichkeit auftreten
- Treten bestimmte Zeichen oft auf, sinkt die Entropie
- Die Entropie gibt an wieviele Bit je Zeichen mindestens für eine Nachricht notwendig wären
- Botschaft aus lauter gleichen Zeichen: $H = 0$
- Zufällige Datenströme: $H \to H_{\max}$
- Die Einheit 1 Shannon:
  Der Informationsgehalt eines Ereignisses mit der Wahrscheinlichkeit $p = 0{,}5$. (z.B.: Ergebnis Kopf eines Münzwurfs)
- Die Basis 2 für den Logarithmus ist willkürlich.
  Bits (Binärziffern) lassen sich technisch einfach handhaben
  Andere Basis ($n$): Ziffernsystem mit der Basis $n$

## 6.2 Datenkomprimierung

$\longrightarrow$ Platz einsparen ohne viel Zeit zu verbrauchen

Verwendeten Daten haben oft:

- hohe Redundanz
- geringer Informationsgehalt

Beispiele:

- Einfacher Text

  Das Alphabet hat 26 Zeichen + Leerzeichen und einige Satzzeichen

  $\longrightarrow$ 32 Zeichen $\longrightarrow$ 5 Bit

  $\longrightarrow$ es wird jedoch der 8 Bit lange ASCII-Code verwendet

  $\longrightarrow$ 37 % Einsparung möglich

- Geringer Informationsgehalt

  Bei der Kodierung von Bildern oder Audiodaten treten häufige Wiederholungen von gewisse Mustern auf: Blauer Himmel, Hintergrundgeräusch, ...

Diese Eigenheiten verwenden bestimmte Algorithmen um Speicherplatz einzusparen

- Textdateien: Einsparungen 20-50%
- Binäre Dateien: Einsparungen 50-90%
- Manche Dateien: Keine Einsparung
  Dateien aus zufälligen Bits, verschlüsselte Dateien
- Jedes Komprimierungs-Verfahren muss gewisse Dateien verlängern,
  sonst könnte man beliebig kleine Dateien erstellen.

### 6.2.1 Lauflängenkodierung (run-length encoding)

Redundanz: Lange Folgen sich wiederholender Zeichen (runs)

z.B.: "CCCCAAABBAAAAAACCCCCCCCCCCCCABCDDDDDDDAAAAABB" Länge: 43

Mögliche Kodierung: "4C3A2B6A12CABC6D5A2B" Länge: 19

$\longrightarrow$ Komprimierung auf 44%

- Es lohnt sich natürlich nicht Runs der Länge $\leq 2$ zu kodieren
- Binäre Dateien:

  es gibt nur Runs, die zwischen 0 und 1 wechseln

  $\longrightarrow$ “0” und ”1“ selbst brauchen nicht gespeichert zu werden

  Einsparung nur wenn, die Länge des Runs größer ist, als die Anzahl der Bits für die benötigte Binärzahl
- Textdateien: Wenig Einsparungspotential; nur das Leerzeichen wiederholt sich öfter.

### 6.2.2 Kodierung mit variabler Länge (Huffman-Codes)

Gute Platzeinsparung bei Textdateien

Text: normalerweise 8 (7) Bit ASCII-Code für jedes Zeichen

$\longrightarrow$ für Zeichen die häufiger auftreten weniger Bits verwenden

Beispiel: `MISSISSIPPI`

Kodierung: Alphabet: 5 Bit (32 Zeichen), A:1, . . . ,Z:26

`M: 13: 01101, I:  9: 01001, S: 19:10011, P: 16: 10000`

`01101010011001110011010011001110011010011000010000001001`

I: 4, M:1, P: 2, S: 4

Das nur einmal vorkommende M benötigt gleichviele Bits wie das S, das viermal auftritt.

$\longrightarrow$ Code mit variabler Länge

$\longrightarrow$ häufig vorkommende Zeichen mit möglichst wenig Bits kodieren

$\longrightarrow$ Gesamtzahl der Bits für die Zeichenfolge soll minimal werden

z.B.: Kode: I:0, S:1, P:01, M:10 $\longrightarrow$ `10 0 1 1 0 1 1 0 01 01 0`

Einsparung: vorher 11*5 = 55 Bits $\longrightarrow$ nachher: 14 Bits

Problem: Leerzeichen zwischen den Zeichen

$\longrightarrow$ einfach weglassen ist nicht sie Lösung, da die Bitfolge nicht eindeutig ist!

`10011011001010` kann unter anderem auch als SIP. . . interpretiert werden

Lösung: Kein Zeichenkode darf mit dem Anfang eine anderen übereinstimmen

Kode: I:11, S:00, P:10, M:010 $\longrightarrow$ `01011000011000011101011`

$\longrightarrow$ es sind zwar 23 Bits, dafür ist die Dekodierung eindeutig

Algorithmus (D. Huffman 1952):

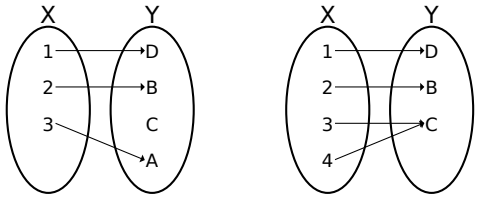
Führt mit Hilfe eines binären Baums für jede beliebige Zeichenkette zu einer Bitfolge mit minimaler Länge, die eindeutig wieder zur ursprünglichen Zeichenkette entschlüsselt werden kann.

## 6.3 Hashes

Eine Hashfunktion ist eine Funktion, die eine Zeichenfolge x **beliebiger Länge** auf eine Zeichenfolge y mit **fester Länge** abbildet

$$h\colon\ X \to Y\ :\quad h(x) = y$$

Hashfunktionen sind nicht injektiv (linkseindeutig) und nicht notwendigerweise surjektiv (rechtstotal)




Anwendungen:

- Integritätsprüfung von Dateien oder Nachrichten
- ”Verschlüsselung“ von Passwörtern
- Digitale Signaturen

Für die praktische Verwendbarkeit muss $h(x)$ bestimmte Bedingungen erfüllen:

- $h(x)$ muss, auch für lange $x$ schnell berechnet werden können
- $x$ darf aus $y$ praktisch nicht berechenbar sein
- $h(x_1) \neq h(x_2)$ für alle $x_1 \neq x_2$ (Kollisionssicherheit)
  Ist zwar theoretisch unmöglich, da die Menge aller Dateien (belibig Lange) sicher größer ist als die $h(x_i)$ mit fixer Länge.
- Ändert sich $x$ nur wenig (z.B. 1 Byte), soll sich $y$ ”stark“ ändern

Praktisch umöglich bedeutet, dass nur durch ausprobieren aller Möglichkeiten eine Lösung gefunden werden kann. Die Anzahl der Möglichkeiten muß dabei so hoch sein, dass es die Berechnungen nicht in einem absehbaren Zeitrahmen möglich sind.

Beispiele:

### 6.3.1 Message-Digest Algorithm (MD5)

- 1993 entwickelt, weit verbreitet
- aus einer beliebigen Nachricht wird ein 128-Bit-Hashwert erzeugt
- 2004 Kollisionen berechnet, gilt als unsicher
  Es ist mit überschaubarem Aufwand möglich unterschiedliche Nachrichten mit gleichem MD5-Wert zu erzeugen

Anwendungen:

- Überprüfung eines Downloads auf Korrektheit
- Passwortspeicherung
- Signieren von Nachrichten, Dokumenten oder Schlüsseln

Beispiel:

```
md5sum Dateiname
echo "Irgend ein Text" | md5sum
```

### 6.3.2 Secure Hash Algorithm (SHA)

Eine Reihe von Algoritmen von NIST und NSA für Digitale Signaturen entwickelt

1. **SHA-1**
   - 160 Bit lang, für beliebige digitale Daten von maximal $2^{64} - 1$ Bit Länge
   - gilt seit 2005 als unsicher, durch Nachfolger ersetzen
2. **SHA-2** Vier weitere Varianten des Algorithmus
   - Es werden längere Hash-Werte erzeugt
   - SHA-224, SHA-256, SHA-384 und SHA-512. Zahl: Länge des Hash-Werts (in Bit)
   - gelten noch als sicher
   - Beispiel: `sha###sum Dateiname`
     `echo "Irgend ein Text" | sha###sum`
   - **SHA-3**
     Keccak wurde 2012 vom NIST als Gewinner des SHA-3-Wettbewerbs bekanntgegeben. Wird dzt. standardisiert.

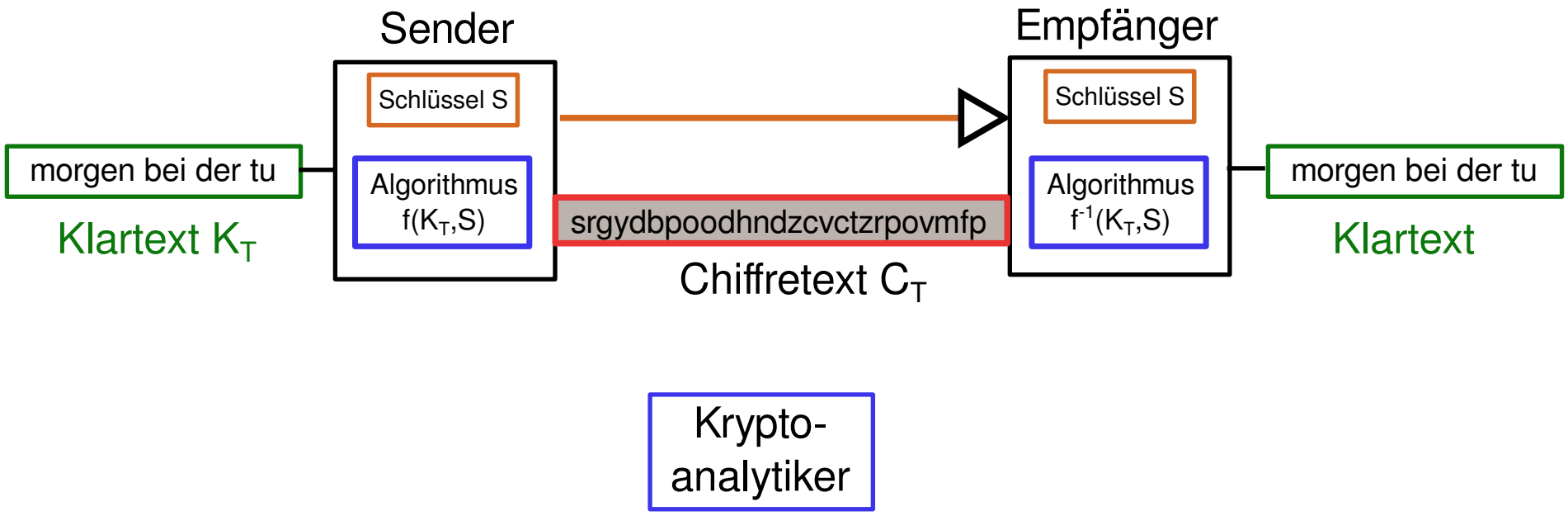
### 6.3.3 Anwendungen

1. **Integritatsprüfung von Dateien oder Nachrichten**
   (a) Der ”Sender“ stellt den Hash (z.B. auf seiner Download-Seite) zur Verfügung
   (b) Der ”Empfänger“ berechnet nach dem Download den Hash-Wert
   (c) Sind die beiden Werte gleich, kann mit einer hohen Wahrscheinlichkeit angenommen werden, dass die beiden Dateien gleich sind.
2. **”Verschlüsselung“ von Passwortern**
   (a) Es wird nicht das Passwort, sondern dessen Hash-Wert gespeichert
   (b) Nach der Passwort-Eingabe wird der Hash-Wert berechnet und verglichen
   (c) Oft werden bei der ersten Passwort-Eingabe noch einige zufällige Bytes (salt) generiert und mit dem Hash-Wert gespeichert.
   Verhindert, dass bei zufällig gleichen Passwörtern auch der Hash-Wert gleich ist und erschwert das Vorausberechnen von Passwörtern (rainbow tables)
3. **Digitale Signaturen** Damit (z.B. zur Überprüfung der Echtheit) nicht der private Schlüssel bekanntgegeben werden muss, wird nur dessen Hash-Wert verwendet.

## 6.4 Geheimniskrämerei (Kryptologie)

In den vorigen Kapiteln wurden bereits Daten verschlüsselt um Platz zu sparen (6.2) bzw. um sie vor allen (auch sich selbst) geheim zu halten (6.3).

Kryptographie: Vertraulichkeit, Integrität und Authentizität einer Nachricht schützen




Sender: $C_T = f(K_T, S)$ $\longrightarrow$ Empfänger: $K_T = f^{-1}(C_T, S)$

Man unterscheidet:

Symmetrische Verschlüsselung: $C_T = f(K_T, S) \quad \rightarrow \quad K_T = f^{-1}(C_T, S)$

Asymmetrische Verschlüsselung: $C_T = f(K_T, S_1) \quad \rightarrow \quad K_T = f^{-1}(C_T, S_2)$

### 6.4.1 Einfache Methoden

#### Cäsar-Chiffre

- Klartext: $n$-ter Buchstabe im Alphabet $\longrightarrow$
  Chiffretext: durch den $(n + k)$-ten Buchstaben im Alphabet ersetzt ($k$: Konstante Zahl)
- Älteste Verschlüsselungsmethode
- Monoalphabetisch: Jeder Klartext-Buchstabe wird immer auf den selben Chiffretext-Buchstaben abgebildet
- Leicht zu entschlüsseln; man braucht nur alle $l - 1$ in Frage kommenden Möglichkeiten ausprobieren ($l$: Länge des Alphabets)

(Beispiel: `./caesar.py -o 3 pangramm1.txt`)

## Ersetzungstabelle

- Für jedes Zeichen im Klartext wird ein Zeichen aus einer Ersetzungstabelle verwendet
  Ersetzungstabelle: Umsortierung des Alphabets
- Wesentlich stärkere Verschlüsselung als Cäsar-Chiffre
- Monoalphabetisch
- Entschlüsselung:
  Zwar $(l+1)!$ Möglichkeiten um alle Tabellen auszuprobieren.
  Mit statistischen Methoden (Häufigkeiten von Buchstaben und deren Kombinationen) leicht zu entschlüsseln

(Beispiel: `./ersetze.py text_faust.txt 1> ch1.txt 2> key1.txt`)

## Vigenere-Chiffre

- Ein meist kurzer sich wiederholender Schlüssel wird benutzt, um den Wert $k$ (aus Cäsar-Chiffre) für jeden Buchstaben neu zu bestimmen $S =$ABC $\rightarrow \quad k = 1, 2, 3$
- Verallgemeinerung der Cäsar-Chiffre
- Ist der Schlüsseltext gleichlang wie der Klartext liegt Vernam-Verschlüsselung (one time pad) vor
- Nicht mehr monoalphabetisch
- Entschlüsselung:
  Mit statistischen Methoden und Perioden bei kurzen Schlüsseln
  Gilt für lange (zufällige) Schlüssel als sicher

(Beispiel:

```
./vignere.py -k xyz text_wiki1.txt 1> chiffre.txt 2> schluessel.txt)
```

## XOR-Verschlüsselung

- Bitweise XOR-Verknüpfung von Klartext und Schlüssel
- Für Daten in binärer Form einfach zu handhaben
- Gilt als sicher, wenn der Schlüssel ein Zufallsmuster und gleichlang wie der Klartext ist
- Ver- und Entschlüsselung durch die selbe Operation
- Die XOR-Verküpfung von Klartext und Chiffre liefert den Schlüssel!!

(Beispiel:

```
./xor.py -k text_wik1.txt text_wiki2.txt 1> chiffre.txt 2> schluessel.txt)
```

### 6.4.2 RSA Verschlüsselung

Problem: Sichere Übertragung (Verteilung) des Schlüssels

Abhilfe: Asymmetrische Private/Public-Key-Verfahren

**RSA**-Verfahren (Rivest, Shamir und Adleman):

- asymmetrisches Verfahren
- Anwendung: Verschlüsselung und digitale Signatur
- verwendet ein Schlüsselpaar:
  privater Schlüssel $S$:
  - zum Entschlüsseln oder Signieren von Daten
  - wird geheim gehalten
  - darf nur mit extrem hohem Aufwand aus dem öffentlichen Schlüssel berechnet werden können

  öffentlicher Schlüssel $P$:
  - zum Verschlüsseln und prüfen von Signaturen
  - wird veröffentlicht

Es gilt: $f(S, f(P, K_T)) = K_T$

Es gibt folgende Möglickeiten:

1. A sendet an B einen Text $K_T$ den nur B lesen können soll:
   - A holt sich den öffentlichen Schlüssel $P_B$ von B
   - Verschlüsselt $K_T$ : $\quad C_T = f(P_B, K_T)$
   - nur B kann aus $C_T,\ K_T$ mit seinem privaten Schlüssel ermitteln $K_T = f(S_B, C_T)$
2. A sendet an beliebige Personen eine Botschaft $K_T$, sie können sicher sein, dass sie von A stammt:
   - A verschlüsselt $K_T$ mit seinem privaten Schlüssel $S_A$: $C_T = f(S_A, K_T)$
   - Nur wenn man mit dem öffentlichen Schlüssel $P_A$ von A, $K_T = f(P_A, C_T)$ entziffern kann, ist man sicher dass, $K_T$ mit $S_A$ chiffriert wurde, also mit hoher Sicherheit von A stammt.

3. A signiert eine Botschaft $K_T$, ohne sie zu verschlüsseln:
   - A berechnet mit einer geeigneten Hashfunktion $H(K_T)$ und verschlüsselt diese mit seinem privaten Schlüssel $S_A$: $U_T = f(S_A, H(K_T))$
     $U_T$ wird an die Nachricht $K_T$ angehängt (Unterschrift)
   - Der Empfänger B verifiziert die Signatur $U_T$ mit dem öffentlichen Schlüssel von A: $H(U_T) = f(P_A, U(T))$. $H$ wird mit dem von B selbst gebildeten Hashwert $H'(K_T)$ verglichen
   - Ist $H = H'$ kann mit hoher Wahrscheinlichkeit davon ausgegangen werden, dass die Nachricht fehlerfrei übertragen und nicht gefälscht wurde.

## Verfahren

Zum Verschlüsseln benötigt man zwei sehr große (mehrere 100 Dezimalstellen, mindestens 1024 Bit) stochastisch unabhängige Primzahlen $p,\ q$

Öffentlicher Schlüssel (public key): $(e, N)$

Privater Schlüssel (private key): $(d, N)$

$N := p \cdot q$ (RSA-Modul)

$e$ (Verschlüsselungsexponent), $d$ (Entschlüsselungsexponent) werden aus $N$ erzeugt

(Siehe z.B.: `https://de.wikipedia.org/wiki/RSA-Kryptosystem`)

Das Verfahren ist vor allem für lange Texte sehr rechenaufwendig

$\longrightarrow$ RSA wird mit symmetrischen Verschlüsselungsverfahren kombiniert

$\longrightarrow$ Nur der Sitzungsschlüssel für das symmetrische Verfahren wird mit RSA ausgetauscht

$\longrightarrow$ Als sehr sicher eingestufte Algorithmen zur symmetrischen Verschlüsselung:
3DES und AES (112, 128, 168 oder maximal 256 Bit)
https://de.wikipedia.org/wiki/Blockverschl

$\longrightarrow$ Sichere Hashfunktion: SHA-256 $\rightarrow$ 256 Bit $\longrightarrow$ Signaturverfahren mittels RSA mit nur

einen Verschlüsselungsschritt

⟶ Die Sicherheit und die Performance des Gesamtsystems:

- Von der Sicherheit des Public-Key-Verfahrens abhängig
- Als sicher eingestufte Algorithmen müssen verwendet werden
- Wahl des Schlüssels muss hinreichend zufällig sein

# 7 Programm, Prozess, Thread

- Computerprogramm:

  Passive Anordnung von Befehlen (Instruktionen)

- Prozess:

  Aktuelle Instanz eines Programms, das gerade ausgeführt wird.

  Benötigt Resourcen: Speicher, CPU, Plattenplatz, ...

  Verschiedene Prozesse laufen vollkommen isoliert voneinder ab

  UNIX: Parent und Child Process (`pstree -p`)

- Thread:

  Ein Prozess kann aus mehreren Threads bestehen, die sich den Speicherplatz teilen.

  Ein Prozess hat mindestens einen Thread.

Aufteilung eines Programms in Threads (einer Aufgabe in mehrere Prozesse)

$\longrightarrow$ Bessere Ausnutzung von Mehrprozessorsystemen

# 8 Kommunikation zwischen Prozessen

## 8.1 Pipes

In UNIX-artigen Betriebssytemen kann die Ausgabe eines Befehls in eine Datei oder einen anderen Befehl umgeleitet werden.

Es gibt 3 Datenströme (streams): **`stdin, stdout, stderr`**

Kommandozeile:

`ls -la > a.dat` Umleitung von stdout und stderr in die Datei a.dat

`find / -name *.c 2> err.dat` Umleitung von stderr in die Datei err.dat

`less < a.dat` Umleitung der Datei a.dat in less

`grep text a.dat | less` Umleitung der Ausgabe von grep in den Befehl less

Programmiersprachen: Bibliotheks-Funktionen $\longrightarrow$

`popen` (`man popen, pydoc os, pydoc subprocess`)

`popen(befehl,"r")` $\longrightarrow$ Filedescriptor zum Lesen der Ausgabe

`popen(befehl,"w")` $\longrightarrow$ Filedescriptor zum Schreiben von Daten

## 8.2 Signale

In UNIX-artigen Betriebssytemen:

Jeder Prozess erhält vom Betriebssytem eine eindeutige Nummer (=PID, process id)

Ermöglicht 64 Signale an einen Prozess zu senden $\longrightarrow$

Prozess führt eine vordefinierte Aktion durch

Der Befehl `kill -signal pid` sendet Signale an den Prozess.

`kill -l` listet alle möglichen Signale. Einige der Signale können vom Programmierer beeinflusst werden, andere führen vom Betriebssystem vorgegebene Aktionen durch.

| Signal | Name | Aktion | |
|---|---|---|---|
| 1 | SIGHUP | Einlesen der Konfigurationsdatei | User |
| 3 | SIGQUIT | Programmabbruch (<CTRL>C) | User |
| 9 | SIGUSR1 | Vom Programmierer definierte Aktion | User |
| 10 | SIGKILL | Programm wird bedingungslos beendet | BS |
| 12 | SIGUSR2 | Vom Programmierer definierte Aktion | User |
| 15 | SIGTERM | Prozess wird geordnet beendet | User |

## Signale im Programm

Signale können vom Programmierer behandelt werden. Siehe **`man 7 signal`**

Signal an Prozess $\longrightarrow$ Signal Handler wird aufgerufen

Signal Handler: Funktion, die bestimmte Anforderungen erfüllen muss

Vorgangsweise: (siehe **`man 2 signal`**)

1. Schreiben eines Signal Handlers: Funktion mit 1 (C) oder 2 (Python) Parametern

   1: `int signum`: Nummer des Signals, das behandelt wird

   2: interrupted stack frame

   Im Signal Handler dürfen nur sichere Funktionen (siehe man 7 signal) aufgerufen werden

2. Registrieren des Handlers durch den Systemaufruf

   `signal(int signum, action)`

   action: SIG_DFL $\rightarrow$ default Aktion

   SIG_IGN $\rightarrow$ wird ignoriert

   Name einer Funktion $\rightarrow$ dieser Signal Handler wird ausgeführt

## 8.3 Threads (Aktivitätsträger oder leichtgewichtiger Prozess)

Satz von Instruktionen, die unabhängig abgearbeitet werden können.

Der Thread läuft mit dem aufrufenden Prozess und teilt sich mit ihm die Resourcen

$\longrightarrow$ globale Variable

Threads des selben Prozesses können über globale Variablen Daten austauschen.

Startet ein Prozess weitere Threads wird nichts kopiert. Die Threads teilen sich den selben virtuellen Speicher, Dateideskriptoren und andere Systemresourcen. Ändert ein Thread den Wert einer Variablen, sieht der andere umittelbar deren neuen Wert.

Bibliotheken zum Erstellen von Threads: Java, python

Je nach Betreibsystem und Implemetierung sind Threads bei der Zuteilung von Resourcen (CPU) benachteiligt.

## 8.4 Prozess

Prozesse arbeiten isoliert voneinander. Jeder Prozess hat ein eigenes Daten- und Codesegment. Ein Prozess kann weitere “Child Prozesse” starten

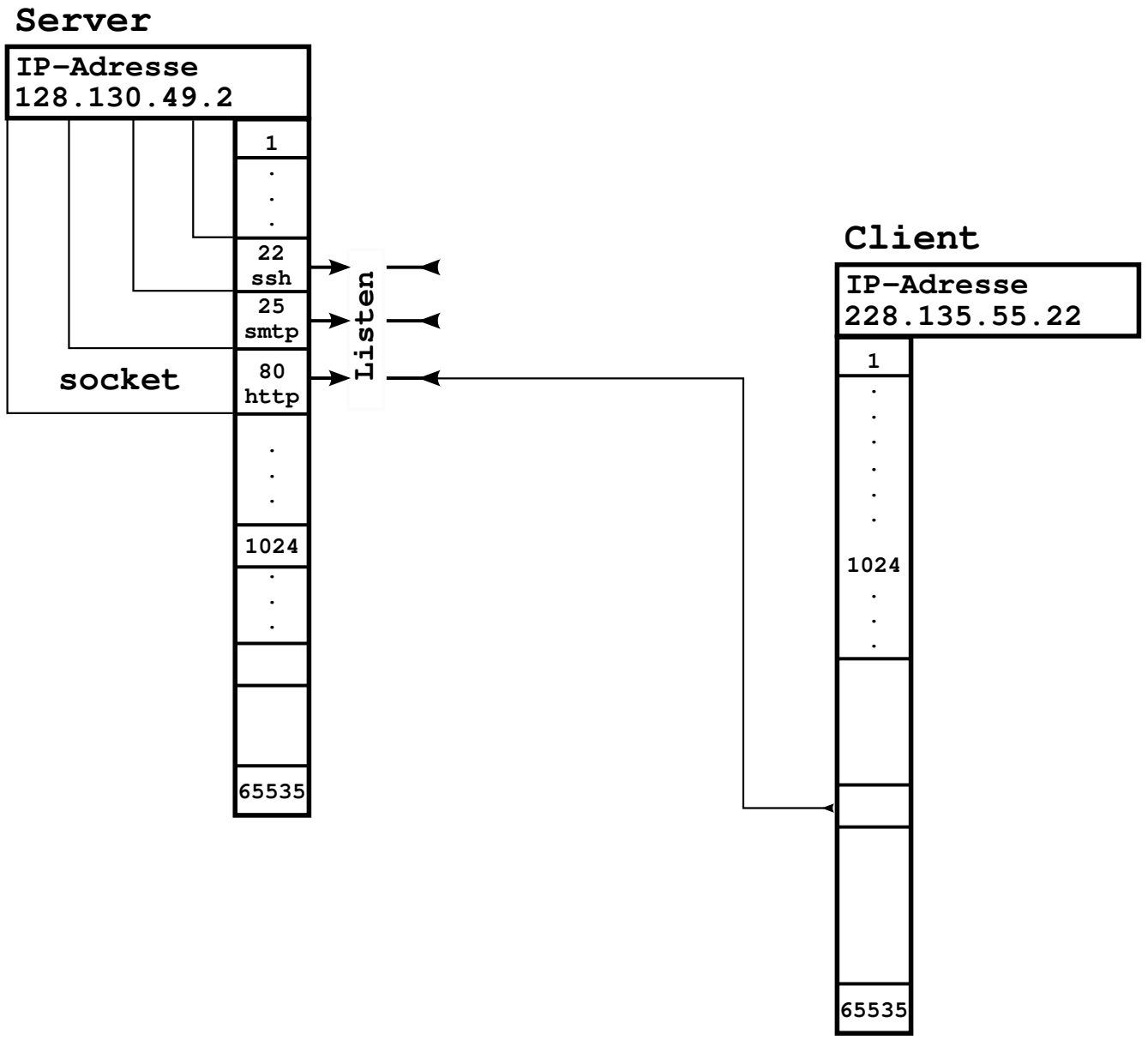
$\longrightarrow$ Kopie der Daten und des Codes wird angelegt; danach getrennt

Systemaufruf: `fork(void)`. Siehe **`man fork`**

Datenaustausch: Inter-process comunication (IPC) z.B.: shared memory

Bibliotheken zum Datenaustausch: Python multiprocessing

# 9 Socket


Server
IP-Adresse
128.130.49.2
1
22
ssh
25
smtp
80
http
1024
65535
socket
Listen
Client
IP-Adresse
228.135.55.22
1
1024
65535

Standardisierte Schnittstelle $\longrightarrow$ Datenaustausch über Netzwerk

Server $\longrightarrow$ Client

Parameter:

- IP-Adresse:

  IPv4: 4 Bytes $\longrightarrow$ 32 Bit $\longrightarrow$ $2^{32} \doteq 4.3 \cdot 10^9$

  IPv6: 128 Bit$\longrightarrow$ $2^{128} \doteq 3.4 \cdot 10^{38}$

  Nameserver: IP-Adresse $\longleftrightarrow$ Name

  **`host server3.physprak.tuwien.ac.at`**

  **`server3.physprak.tuwien.ac.at has address 128.130.49.15`**

- Port:

  mehrere Dienste (services) (ssh, smtp, http, ...) auf einem Server $\longrightarrow$

  65536 Ports $\longrightarrow$ jeder Dienst “horcht” auf einem Port

  Siehe: **`less /etc/services`**

  Ports 1 . . . 1024: “privilegierte” Ports für Serverdienste

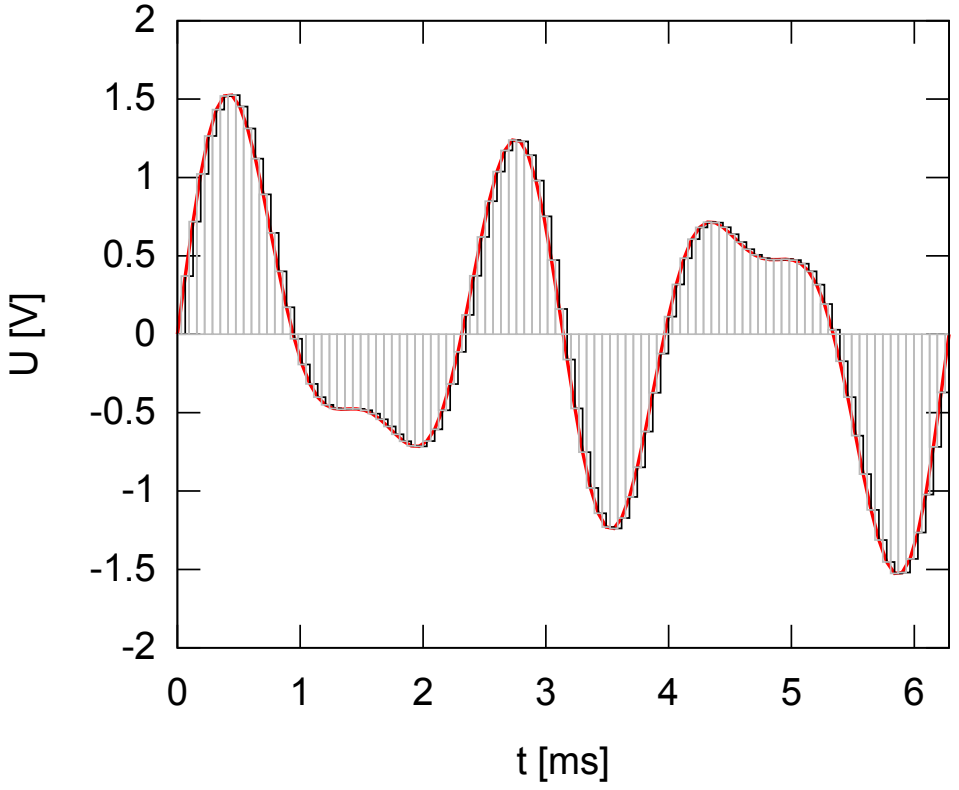
  restliche Ports: Für Clientverbindungen

# 10 Sound

Soundkarte erfasst Töne (Sprache, Musik,...) Eingang: Mikrophon $\longrightarrow$ Spannung wird in eine Zahlenfolge umgewandelt (ADC)

Ausgang: Zahlenfolge wird in Spannung umgewandelt (DAC) $\longrightarrow$ Lautsprecher

Umwandlung: PCM (Pulse Code Modulation) Zeitlich äquidistante Abtastung mit $t_a$




Abtasttheorem: $1/t_a > 2 \cdot f_{max}$

## 10.1 PCM Terminologie

- Sample

  PCM Audio (Eingang und Ausgang) besteht aus “samples”

  Einzelne Sample $\longrightarrow$ Amplitude eines Kanals (Tonspur) zu einem bestimmten Zeitpunkt

  Geräusch $\longrightarrow$ Vielzahl individueller Samples

  z.B.: Audio-CD: 44100 Samples/s

  Sample$\longrightarrow$ Zahlenwert, Genauigkeit 8-64 Bit $\longrightarrow$ Datentyp: int, float, BE,LE

  Musikalisch: Sample Size $\longrightarrow$ Lautstärke, Dynamik (Differenz laut-leise)

- Frame

  Ein “frame” ist eine Sample pro Kanal

  Mono: 1 Frame $\longrightarrow$ 1 Sample

  Stereo: Jedes Frame besteht aus zwei Samples

- Frame size

  Größe in Bytes für jedes Frame

  z.B.: 8 Bit/Sample, Mono $\longrightarrow$ Frame Size: 1 Byte

  64 Bit/Sample, 6 Kanäle $\longrightarrow$ Frame Size: 48 Bytes

- Rate

  PCM-Audio: Strom von Sound Frames

  Sound Rate: Wie oft wird das augenblickliche Frame erneuert

  z.B.: Rate von 8000 Hz $\longrightarrow$ 8000 mal je Sekunde wird ein Frame gespielt oder aufgezeichnet

- Datenrate (data rate)

  Anzahl der Bytes, die bei einer bestimmten Frame size und Rate je Sekunde aufgezeichnet oder wiedergegeben werden

  z.B.: 8000 Hz Mono 8 bit (1 byte) Samples (Telefon): Datenrate

  $8000 \cdot 1 \cdot 1 = 8\mathrm{kb/s} = 64k\mathrm{bit/s}$

  96000 Hz, 6 Kanäle, 64 bit (8 bytes) Samples: Datenrate of 96000 * 6 * 8 = 4608 kb/s

- Period

  Die Hardware verarbeitet die Daten in Chunks of Frames.

  Das Zeitintervall für eine Konversion $\longrightarrow$ Period

  $\longrightarrow$ direkter Einfluß auf die Latenzzeit (latency) bei der Ein/Ausgabe.

  Niedrige Latenzzeit: Period $\longrightarrow$ niedrig

  Schwache CPU $\longrightarrow$ höhere Latenzzeit

  ALSA: CPU-Auslastung nicht stark von der Latenzzeit betroffen $\longrightarrow$ Bufferung

- Period Size

  Die Größe der Period in Hz (Keine bytes, Hz!)

  ALSA: Wichtig!! Wenn die Period Size z.B. auf 32 gesetzt wurde, muß jeder Schreibvorgang genau 32 Frames enthalten und jeder Lesevorgang gibt ebenfalls 32 Frames zurück.

Siehe: `http://pyalsaaudio.sourceforge.net/terminology.html`

## 10.2 Analyse von Geräuschen

Geräusch: Trommelfell wird durch $p(t)$ in Schwingung versetzt

Lautstärke $\longrightarrow p^2(t)$

Wahrnehmung $\longrightarrow$ Frequenzanteile spielen wesentliche Rolle $\longrightarrow$ Fourieranalyse

**Frequenzspektrum:**

Gibt an mit welcher Amplitude die entsprechende Frequenz vertreten ist

**Fourier Reihe:**

**Diskretes Frequenzspektrum** einer **periodischen Funktion**

**Fourier Integral:**

**Kontinuierliches Frequenzspektrum** einer **beliebigen Funktion**

# 10.3 Fouriertransformation

$$f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} F(k)\, e^{ikx}\, dk$$

$$f(x) = \mathcal{F}^{-1}\{F(k)\}$$

inverse Fourier Transformation

$$F(k) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x)\, e^{-ikx}\, dx$$

$$F(k) = \mathcal{F}\{f(x)\}$$

Fourier Transformation

$$F(k) = A(k) + i\, B(k)$$

| | | |
|---|---|---|
| $x$ | $\rightarrow$ | $k$ |
| Ortsraum | $\rightarrow$ | Wellenzahlraum |
| $t$ | $\rightarrow$ | $\omega$ |
| Zeitdarstellung | $\rightarrow$ | Frequenzdarstellung |

## 10.4 Diskrete Fouriertransformation

Liegt die Funktion als $2N+1$ (äquidistant) abgetastete Datenwerte $f(nT)$ vor $\longrightarrow \int \rightarrow \sum$

$$F(\omega) \approx \frac{1}{\sqrt{2\pi}} \sum_{n=-N}^{N} e^{-i\omega nT} f(nT)\, T$$

Ist $N = 2^l$ (Zweierpotenz) $\longrightarrow$ effizienter Algorithmus zur Berechnung von $F(\omega)$

**FFT (fast fourier transformation)**

1805 von Carl Friedrich Gauß entwickelt; später mehrmals verbessert

Siehe z.B.:`http://de.wikipedia.org/wiki/Schnelle_Fourier-Transformation`

## Praktische Berechnung

Bibliotheksfunktionen (Python: scipy)

Ergebnis: Real- und Imaginärteil $\longrightarrow$ Wahrnehmung: Quadrat des Absoultbetrags

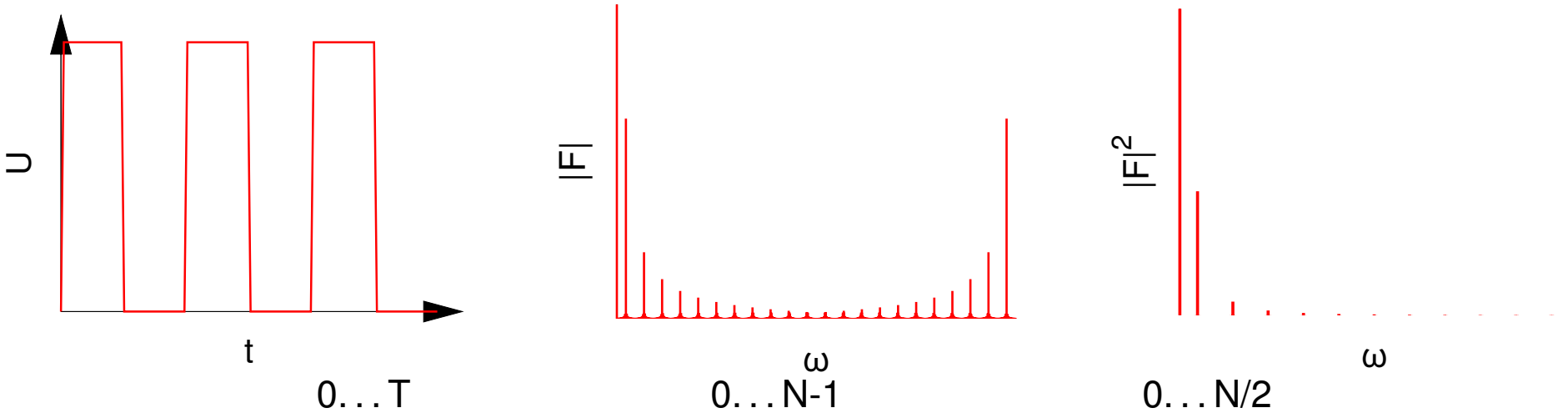
Führt man die Berechnung für $N$ Punkte mit $1/T$ Samples/s durch $\longrightarrow$

`F`: Array [0...N-1]

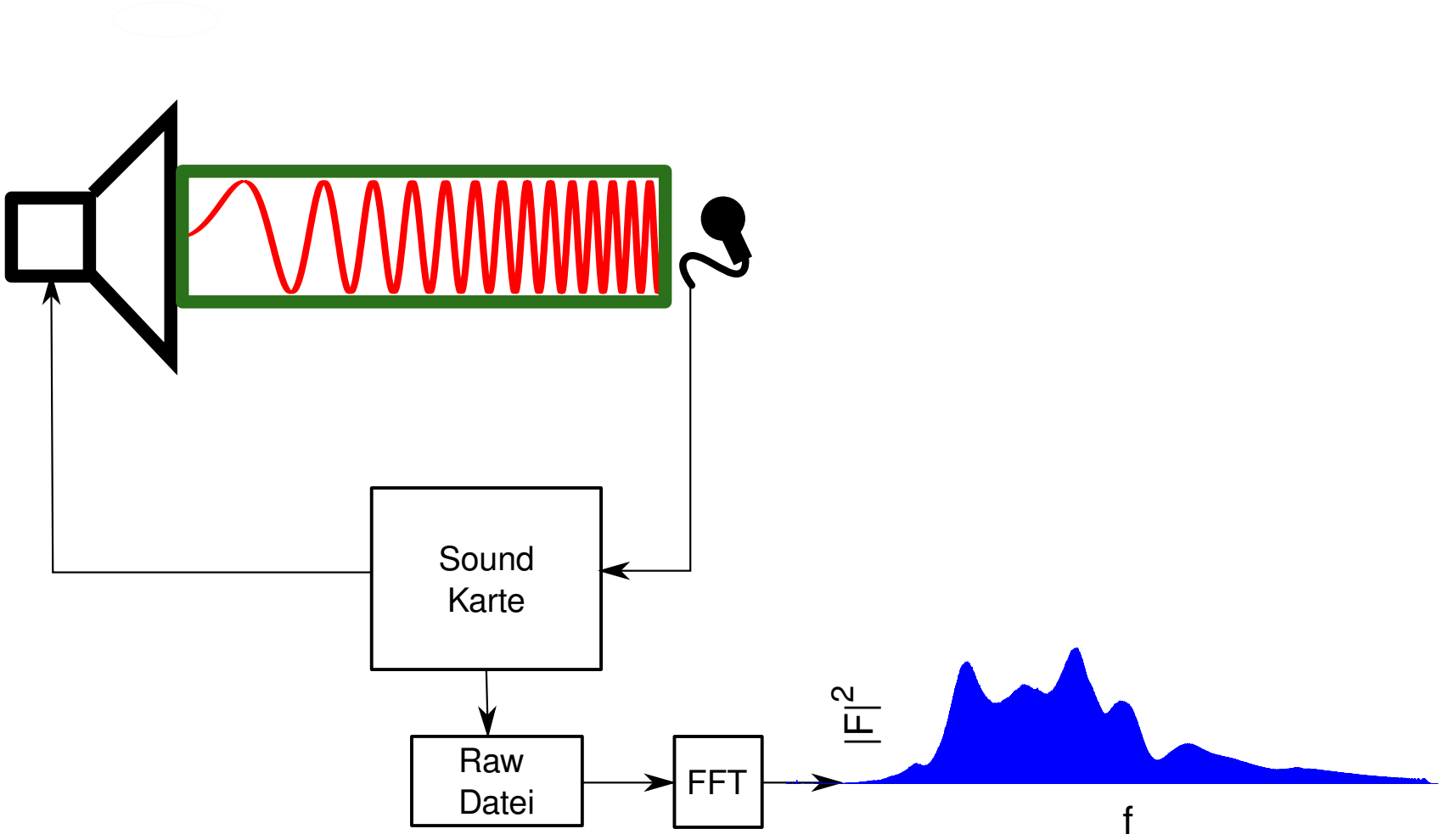
`F[0]`: Mittelwert (Term für $f = 0$)

`F[1:N/2+1]`: Terme für $f > 0$, `F[N/2+1:]`: Terme für $f < 0$

Das Spektrum ist bezüglich des Index's $N/2$ zentrisch symmetrisch



# Bestimmung Übertragungsfunktion



# 11 Modulation

Nachrichtentechnik:

Zu übertragendes Nutzsignal (Musik, Sprache, Daten) verändert ein Trägersignal

Ermöglicht eine hochfrequente Übertragung des niederfrequenten Nutzsignals

Sendesignal verwendet im Bereich der Trägerfrequenz eine vom Nutzsignal abhängige Bandbreite: $\Delta f_T = k \cdot \Delta f_N$

Zurückgewinnen der Nachricht $\longrightarrow$ **Demodulation**

Das Trägersignal selbst ist nur bedingt von Bedeutung und wird nach der Demodulation unterdückt

Das Trägersignal muss besimmte Bedingungen erfüllen:

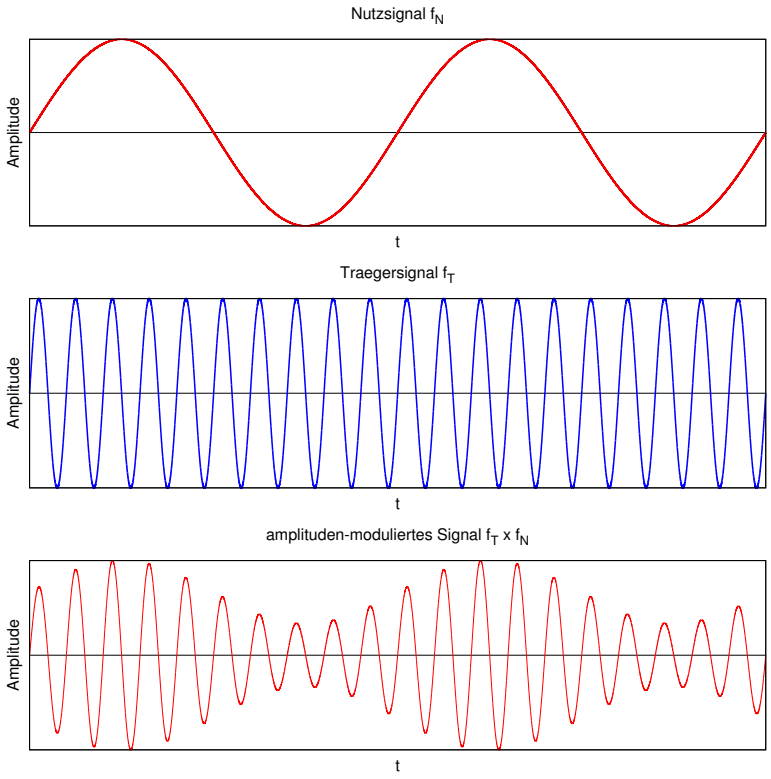
$$f_T >> f_{Nmax}$$

$f_T$: Trägerfrequenz, $f_{Nmax}$: maximale Frequenz im Nutzsignal

# 11.1 Amplitudenmodulation (AM)

Älteste und “einfachste” Modulation:

Ein hochfrequentes Trägersignal ändert die Amplitude im Takt des zu übertrageneden Signals




Nutzsignal: $u_{\mathrm{N}} = \hat{U}_{\mathrm{N}} \cos(\omega_N t)$

AM: Gleichanteil $\hat{U}_{\mathrm{T}}$ addieren und mit Trägerschwingung $\cos(\omega_T t)$ multiplizieren

$(\omega_T \gg \omega_N)$

$$u_{\mathrm{AM}} = \left(\hat{U}_{\mathrm{T}} + \hat{U}_{\mathrm{N}} \cos(\omega_N t)\right) \cdot \cos(\omega_T t)$$

$$u_{\mathrm{AM}} = \hat{U}_{\mathrm{T}} \cos(\omega_T t) + \hat{U}_{\mathrm{N}} \cos(\omega_N t) \cos(\omega_T t)$$

$$\cos\alpha \cos\beta = \frac{1}{2}\left(\cos(\alpha-\beta) + \cos(\alpha+\beta)\right)$$

ergibt:

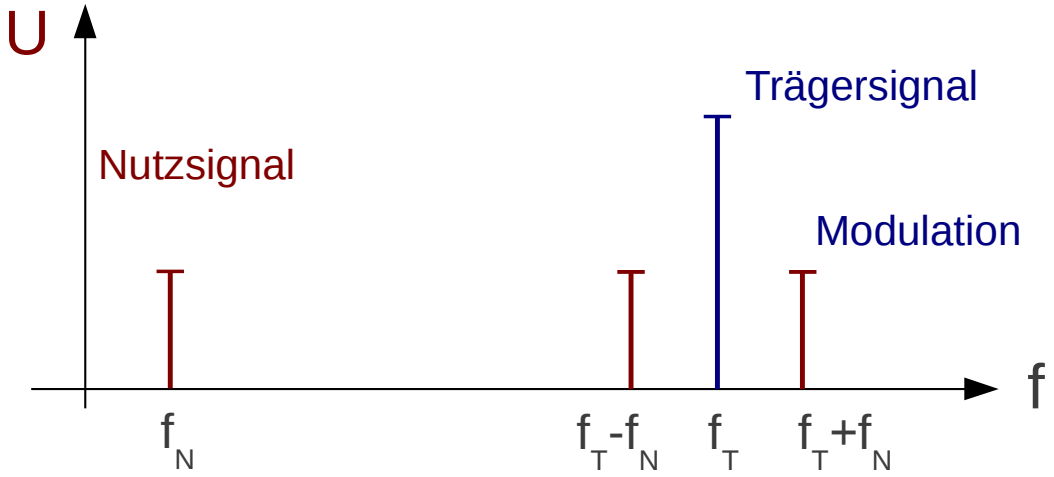
$$u_{\mathrm{AM}}(t) = \hat{U}_{\mathrm{T}} \cos\left(\omega_T t\right) + \frac{\hat{U}_{\mathrm{N}}}{2}\left(\cos\left((\omega_T - \omega_N)t\right) + \cos\left((\omega_T + \omega_N)t\right)\right)$$

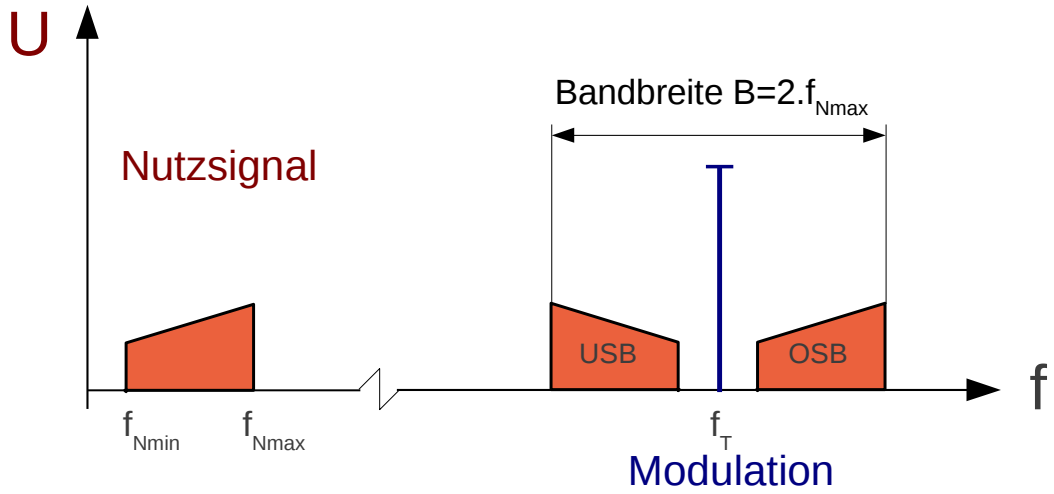
Frequenzspektrum:

Träger: $u_{\mathrm{T}} = \hat{U}_{\mathrm{T}} \cos\left(\omega_T t\right)$ mit der Trägerfrequenz $\omega_T$ und der Amplitude $\hat{U}_{\mathrm{T}}$

2 Seitenbänder: mit den Seitenfrequenzen $\omega_T - \omega_N$ und $\omega_T + \omega_N$ und der Amplitude $\frac{\hat{U}_{\mathrm{N}}}{2}$

Fourier-Transformation ⟶ Moduliertes Signals im Frequenzbereich



# Demodulation

## Einfachste Form:

Gesuchtes Frequenzband wird mit einem Bandpass herausgefiltert $\longrightarrow$

Gleichrichten mit einer Diode $\longrightarrow$

Glätten mit einem Tiefpass $\longrightarrow$

Entfernen des Gleichanteils mit einem Hochpass

Rundfunkübertragung auf Mittelwelle (300 kHz $\rightarrow$ 1000 m . . . 3000 kHz $\rightarrow$ 100 m)

Sprache und Musik:

Standardisiertes Frequenzband von 4.5 kHz Breite (von 0 Hz bis 4.5 kHz) wird übertragen

$\longrightarrow$ Bandbreite B = 9 kHz

Bildsignal beim Fernsehen: B $\approx$ 5.5 MHz

Aufwändigere Form: Trägerfrequenz wird lokal (im Empgänger) benötigt

Das empfangene Signal ($u_{\mathrm{AM}}$) wird mit dem lokalen Träger multipliziert:

$$u_{\mathrm{DAM}} = u_{\mathrm{AM}}(t) \cdot \cos(\omega_T t)$$

$$= \left(\hat{U}_{\mathrm{T}} \cos(\omega_T t) + \frac{\hat{U}_{\mathrm{N}}}{2} \left(\cos((\omega_T - \omega_N)t) + \cos((\omega_T + \omega_N)t)\right)\right) \cdot \cos(\omega_T t)$$

$$= \hat{U}_{\mathrm{T}} \cos^2(\omega_T t) + \frac{\hat{U}_{\mathrm{N}}}{2} \Big( \cos((\omega_T - \omega_N)t) \cos(\omega_T t) + \cos((\omega_T + \omega_N)t) \cos(\omega_T t) \Big)$$

Additionstheoreme: $\cos\alpha \cos\beta = \frac{1}{2}\Big(\cos(\alpha - \beta) + \cos(\alpha + \beta)\Big)$

$$\cos^2\alpha = \frac{1}{2}\Big(1 + \cos(2\alpha)\Big)$$

$$u_{\mathrm{DAM}} = \frac{\hat{U}_{\mathrm{T}}}{2}\Big(1 + \cos(2\omega_T t)\Big)$$

$$+ \frac{\hat{U}_{\mathrm{N}}}{4}\Big( \cos(-\omega_N t) + \cos((2\omega_T - \omega_N)t) + \cos(\omega_N t) + \cos((2\omega_T + \omega_N)t)\Big)$$

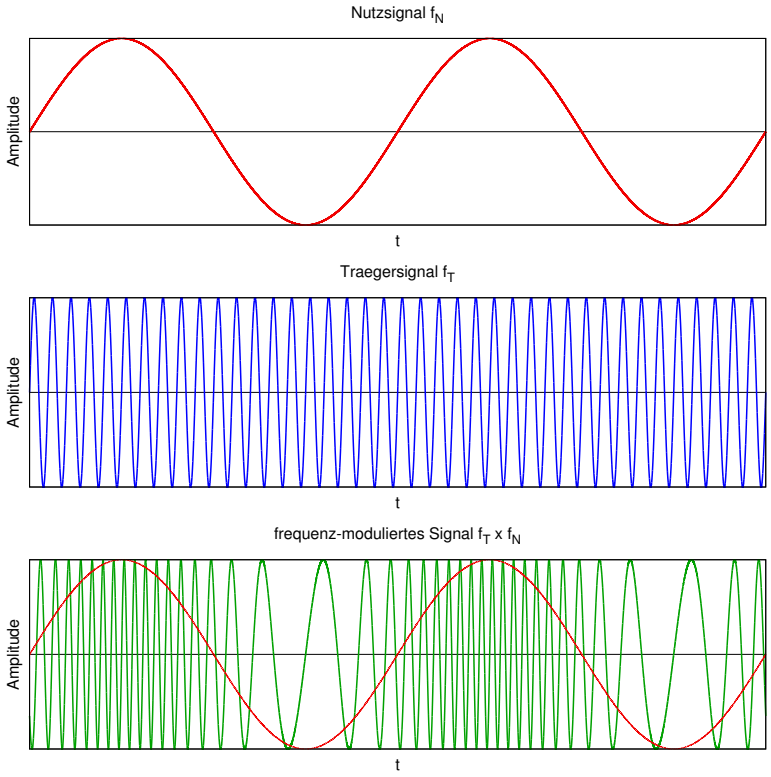
Filtern der unerwünschten hohen Frequenzanteile ($2\omega_T$) mit einem Tiefpass und des Gleichanteils mit einem Hochpass $\rightarrow$ Nutzsignal mit halber Amplitude:

$$u_{\mathrm{TP}} = \frac{\hat{U}_{\mathrm{N}}}{4}\left(\cos(-\omega t) + \cos(\omega t)\right) = \frac{\hat{U}_{\mathrm{N}}}{2}\cos(\omega t)$$

## 11.2 Frequenzmodulation (FM)

Weniger störanfällig als AM:

Ein hochfrequentes Trägersignal ändert die Frequenz im Takt des zu übertrageneden Signals




Trägersignal: $u(t) = \sin(\omega_0 t + p_0)$

$p(t) = (\omega_0 t + p_0)$: momentane Phase

$\omega_0$: Trägerkreisfrequenz, $p_0$: Phase bei $t = 0$

$p(t)$+Modulator $\longrightarrow$ Phasenmodulation:

$s(t) = \sin(\omega_0 t + p_0 + M_p m(t))$

$M_p$: Modulationstärke, $m(t)$: modulierende Funktion

Frequenzmodulation: stetige Änderung von $f$ $(\omega)$

$\longrightarrow$ momentane Kreisfrequenz: $\omega(t) = \frac{\mathrm{d}}{\mathrm{d}t} p(t)$

das ist die zeitliche Ableitung der Phasenfunktion

Frequenzmodulation: $\omega(t) = \omega_0 + M_f\, m(t)$

Berechnung der Kurvenform: Phasenfunktion (nicht die momentane Frequenz)

Frequenz: Ableitung der Phase $\rightarrow$ Phase: Integral der Frequenz

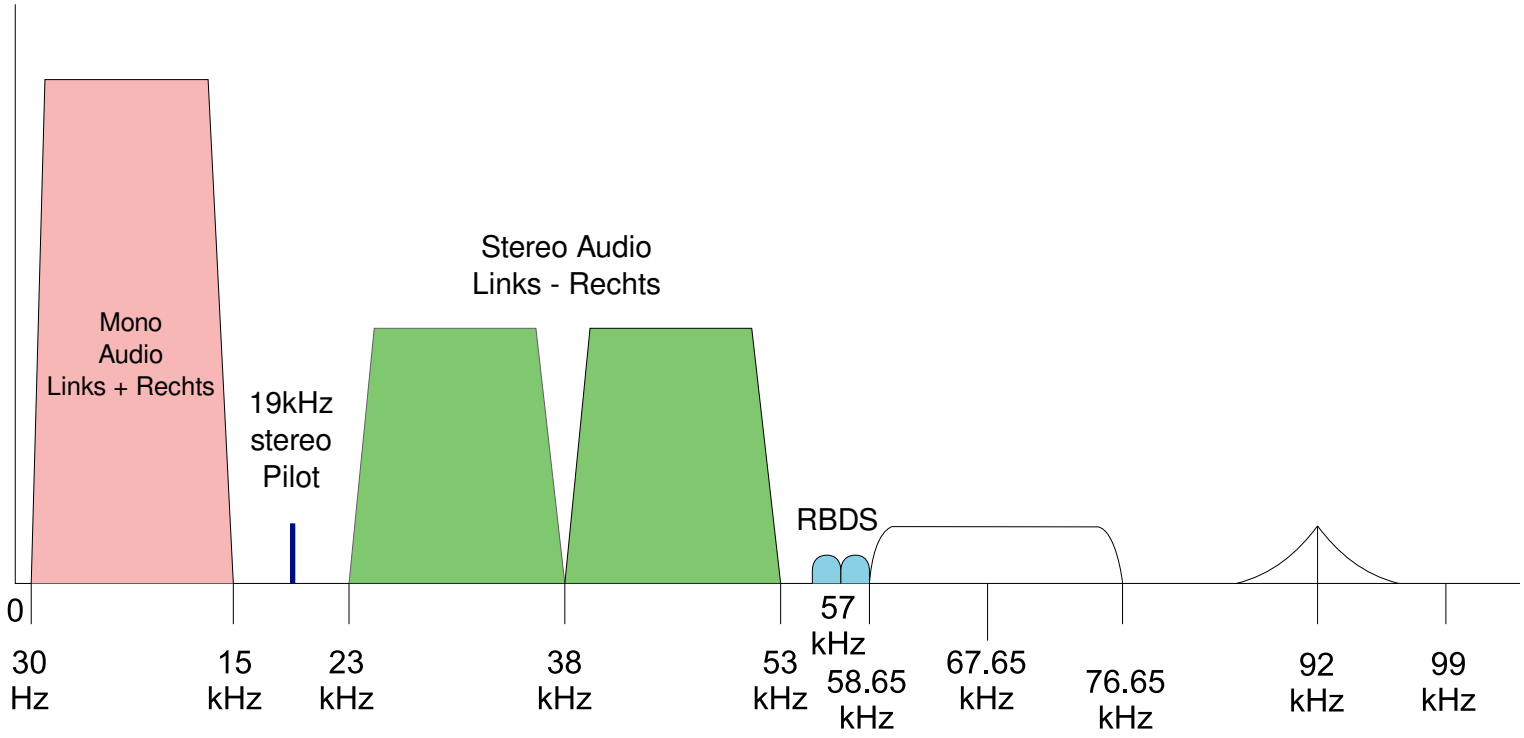
## 11.3 Anwendungen

- FM Rundfunk
- Stereo FM: Pilotton-Multiplexverfahren (FM MPX)
  - Das Summensignal aus linkem und rechtem Kanal im Basisband (Mono)
  - Das Differenzsignal aus linkem und rechtem Kanal mit AM (unterdrückter Träger) auf einer Frequenz von 38 kHz aufmoduliert.
  - 19-kHz Pilotton:
    Erkennung des Stereosignals und zur Demodulation des Differenzsignals

### Wird über FM gesendet

Mono-Empfänger: gibt alle Signale wieder, Differenzsignal und Pilotton (hohes $f$) ist unhörbar.
Stereo-Empfänger: Demoduliert $L - R$ und verwendet $L + R$:

$$(L + R) + (L - R) = (2)L \qquad (L + R) - (L - R) = (2)R$$




- Analoges Fernsehen

  Helligkeitinformation (schwarzweiß): Amplitudenmodulation

  Farbinformation: Phasenmodulation

## 11.4 Einteilung verschiedener Modulationsverfahren:

### 11.4.1 Lineare und nichtlineare Modulationsverfahren

Lineare Modulationsverfahren

Mathematische Funktion zwischen dem Nutzsignal und dem Sendesignal $\longrightarrow$ lineare Funktion

z.B.: Amplitudenmodulation $\longrightarrow$ Multiplikation im Zeitbereich

Nichtlineare Modulationsverfahren

Nichtlinearer Funktionszusammenhang zwischen Nutzsignal und Sendesignal

Die Analyse ist mit höherem Aufwand verbunden

z.B.: Frequenzmodulation $\longrightarrow$ Funktionaler Zusammenhang: Winkelfunktionen

### 11.4.2 Zeitkontinuierliche und zeitdiskrete Verfahren

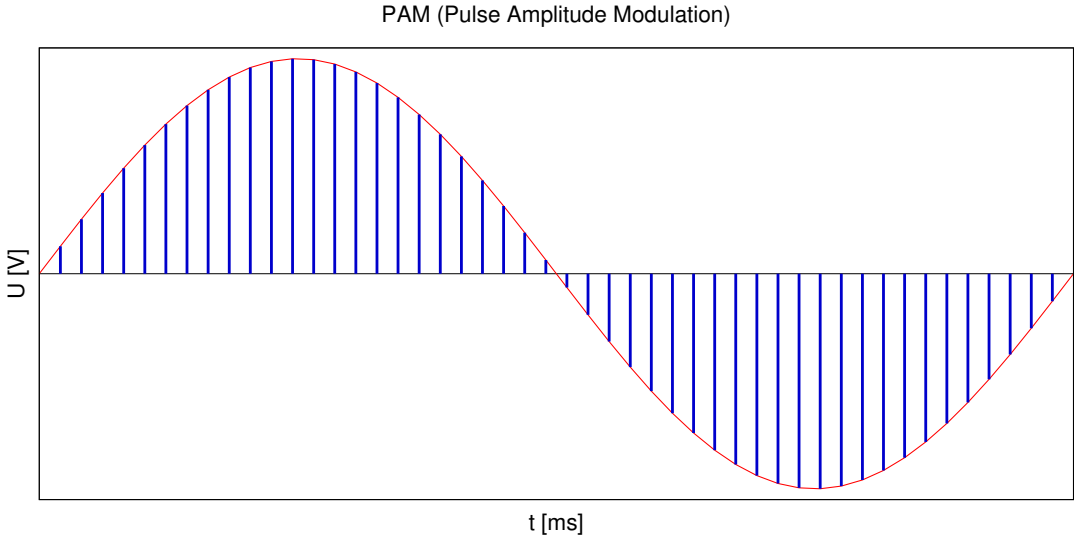
#### Zeitkontinuierliche Modulationsverfahren

Traĝer: kontinuierliches Signal z.B. Sinusschwingung
Das zu modulierende Informationssignal muss die Information nicht zeitkontinuierlich darstellen, nur das modulierte Signal am Modulatorausgang muss zeitkontinuierlich sein
Weitere Unterteilung: Wertkontinuierliche und wertdiskrete Modulation

#### Zeitdiskrete Modulationsverfahren

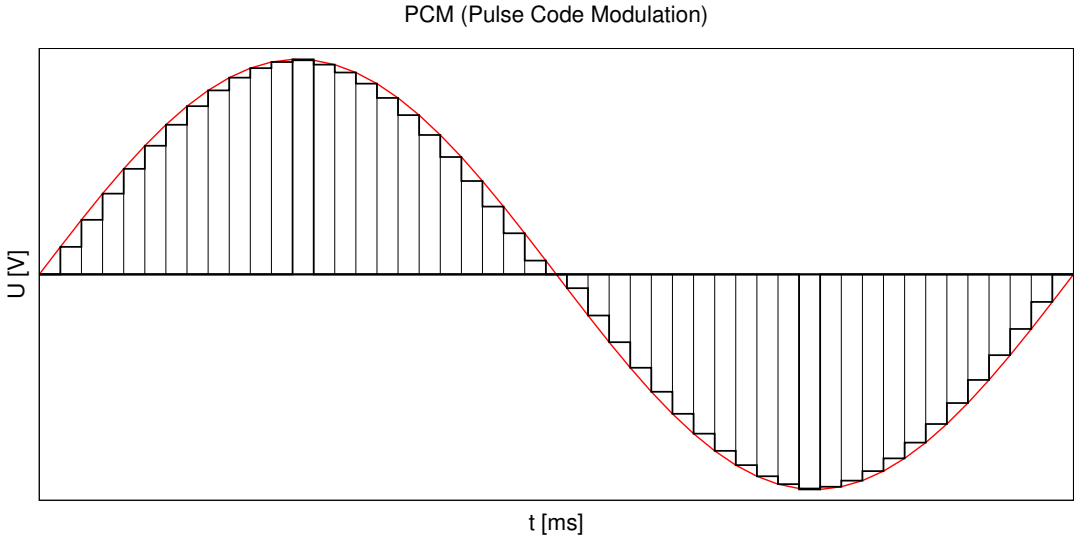
Liefern am Ausgang nur zu bestimmten Zeitpunkten ein definiertes Trägersignal.
Weitere Unterteilung: Wertkontinuierliche und wertdiskrete Modulationsverfahren

# Beispiele:




Wertkontinuierliches und zeitdiskretes Modulationsverfahren:

Pulsamplitudenmodulation (PAM)




Wertdiskretes und zeitdiskretes Modulationsverfahren:

Puls-Code-Modulation (PCM)

### 11.4.3 Analoge Modulation, *Analog Spectrum Modulation* (ASM)

Wertkontinuierliche und zeitkontinuierliche Verfahren: unpräzise $\longrightarrow$ analoge Modulation

Kontinuierliche Verarbeitung des Nutzsignals, keine Digitalisierung der Sendesignalwerte

Analoge Nutzsignale: z.B.: Sprach-, Musik oder Bildsignale

**Amplitudenmodulation (AM)**

analoger Rundfunk auf Mittelwelle

analoge Fernsehtechnik

AM mit unterdrücktem Träger:

Einseitenbandmodulation (SSB):

Amateurfunk

**Winkelmodulation**

Frequenzmodulation (FM):

UKW-Hörfunk

Phasenmodulation (PM)

Kombination aus Amplituden- und Winkelmodulation $\longrightarrow$ Vektormodulation

Information des Nutzsignals befindet sich in der Amplitude UND im Phasenwinkel der Trägerschwingung

Übertragung der Farbinformation beim PAL- bzw. NTSC-Farbbild-(FBAS)-Signal

Farbsättigung $\longrightarrow$ Amplitude

Farbton $\longrightarrow$ Phasenwinkel (Farbhilfsträger)

### 11.4.4 Digitale Modulation, *Digital Spectrum Modulation* (DSM)

Wertdiskrete und zeitkontinuierliche Verfahren: unpräzise $\longrightarrow$ digitale Modulation

Es werden Symbole übertragen, die für Sender und Empfänger eindeutig definiert sind

Analoge Signale (Sprache, Musik,...) müssen vor der digitalen Modulation digitalisiert werden

Es werden nur zu bestimmten Zeitpunkten gültige Werte geliefert

$\longrightarrow$ Abtastzeitpunkt $\longrightarrow$ zeitdiskret

Zeitliche Abstand der Abtastzeitpunkte $\longrightarrow$ Symbolrate

Dazwischen ist die Information des Sendesignals undefiniert

Demodulation: **Taktrückgewinnung**, Empfänger (Demodulator) muss erkennen können, zu welchen Zeitpunkten eine gültige Information vorliegt

Es kann nur eine endliche Anzahl unterschiedlicher Werte übertragen werden

$\longrightarrow$ wertdiskret

Geeignete Wahl der wertdiskreten Sendesymbole

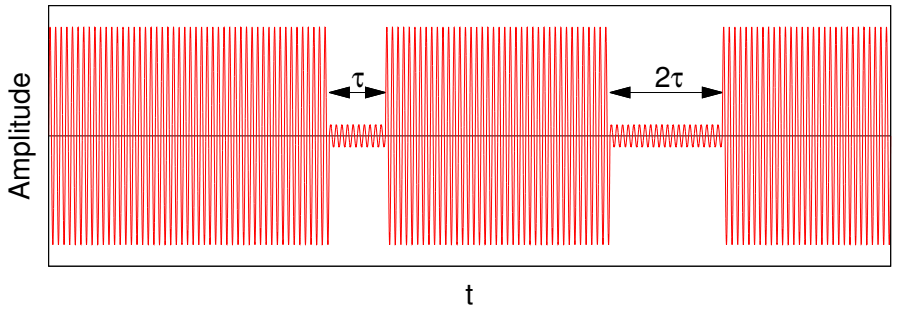
$\longrightarrow$ Abweichungen (Übertragungsfehler) können erkannt und kompensiert werden

$\longrightarrow$ höhere Störunempfindlichkeit

# Digitale Modulationsverfahren mit einem Träger

- **Amplitude Shift Keying (ASK):**

  Amplitude des Sendesignals wird in diskreten Schritten umgeschalten

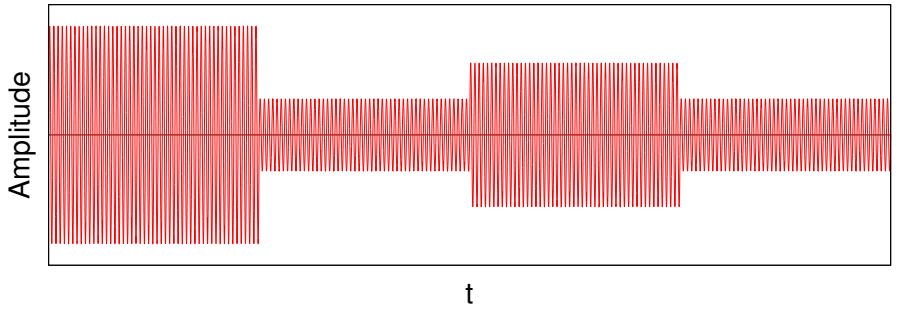



DCF77-Signal der Funkuhr:

Amplitude des Trägers (77.5 kHz) wird im Sekundentakt auf 25% abgesenkt

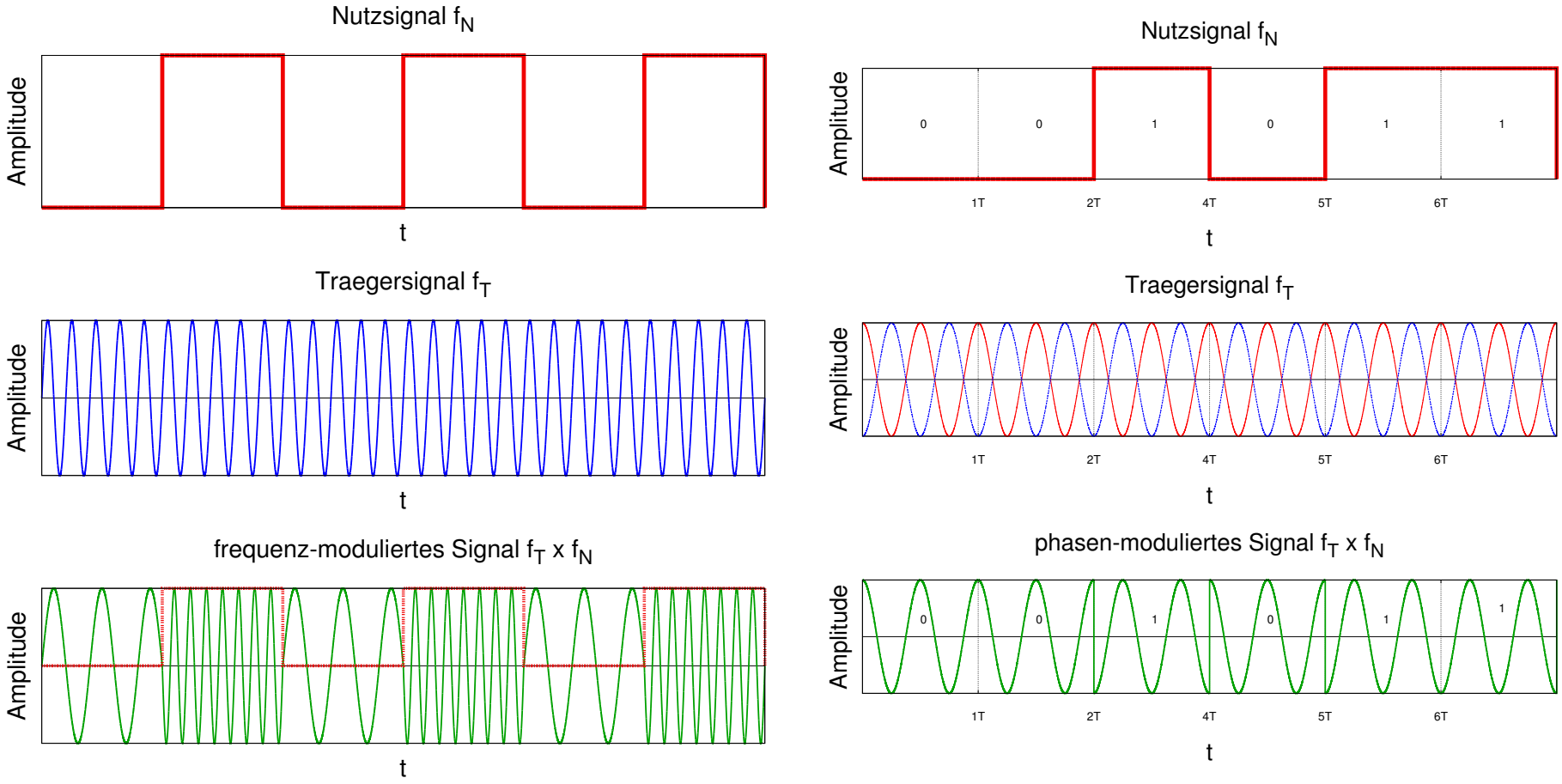
(0.1 s: logisch 0, 0.2 s: logisch 1)

Bei $n$ Sendesymbolen wird zwischen $n$ unterschiedlichen Amplitudenwerten gewählt



- **Frequency Shift Keying (FSK) und Phase Shift Keying (PSK):**

  Die Frequenz oder der Phasenwinkel des Trägersignals wird in diskreten Stufen umgeschaltet




Anwendungen: ältere Telefonmodems (1980er) bis zu 1200 bps

Analoge Faxgeräte verwenden diese Modulationsverfahren

- **Quadraturphasenumtastung, Quadrature Phase-Shift Keying (QPSK):**

  Es werden je zwei Bits (Dibits) verarbeitet

  Träger:

  Zwei sinusförmige Signale derselben Frequenz, eines ist um $90^\circ$ phasenverschoben

  Das QPSK-Signal ist damit die Addition zweier PSK-Signale

  Anwendung: GSM, DVB-S

- **Kombinationen aus Amplituden- und Winkelmodulationen:**

  Die Information (Nutzdatenfolge) wird sowohl in der Amplitude als auch in der Phasenlage des Trägers untergebracht

  Quadraturamplitudenmodulation (QAM, 16QAM, 32QAM, 64QAM, ..., $n$QAM)

  n: Anzahl der diskreten Datenpunkte (Sendesymbole) $\longrightarrow$ Bits pro Symbol

  Empfänger: Muss die einzelnen Symbole voneinander unterscheiden

  Probleme bei großem $n$

## Digitale Modulationsverfahren mit mehreren Trägern

Aufteilung des Nutzdatenstroms auf mehrere unterschiedliche Träger

$\longrightarrow$ Optimale Anpassung an die Eigenschaften des Übertragungskanals

Stehen aufgrund von Störungen einige Träger nicht zur Verfügung,
können die anderen Träger weiterverwendet werden

$\longrightarrow$ Nur der Gesamtdatendurchsatz wird reduziert

Anwendung: ADSL, DVB-T, WLAN, Bluetooth

Auf den einzelnen Trägern werden schmalbandige digitale Modulationen (z.B. 16QAM)
verwendet
Große Anzahl der Träger (einige $10.000$)

$\longrightarrow$ Übertragungskanal kann optimal genutzt werden

Einige 10 kBit an Nutzdaten werden parallel (ein Taktschritt) übertragen

Modulation und Demodulation erfolgt durch digitale Signalprozessoren

## 11.5 Spezielle Modulationen

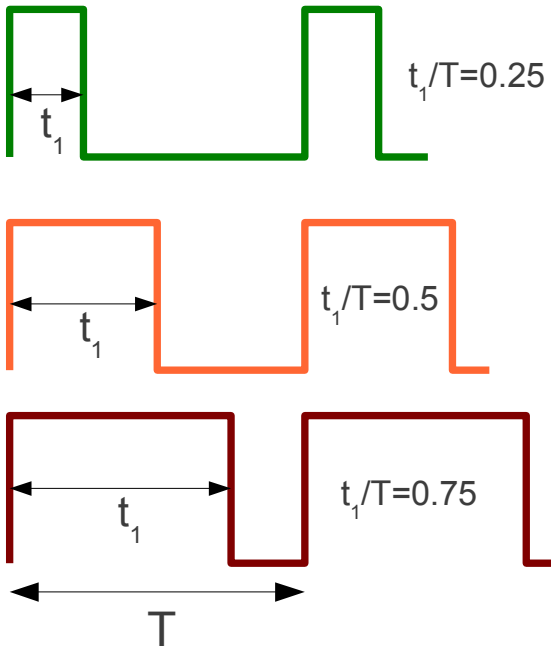
### Pulsmodulationen

Umwandlung eines kontinuierlichen analogen Signals in eine zeitdiskrete Signalfolge

bestehend aus einzelnen Imulsen

Amplitudenkontinuierliche Verfahren:

### 11.5.1 Pulsweitenmodulation (*pulse-width modulation* PWM)




Bei konstanter Frequenz wird der Tastgrad $t_1/T$ eines Rechteckpulses verändert (moduliert)

**Modulation:**

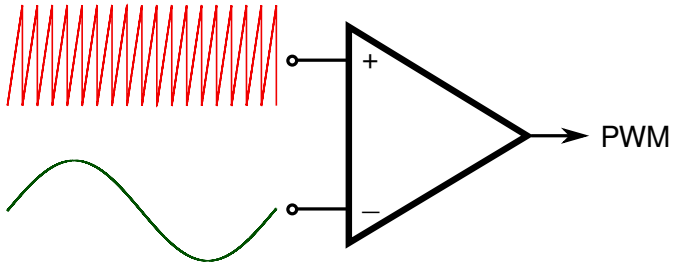
Aus digitalen Daten

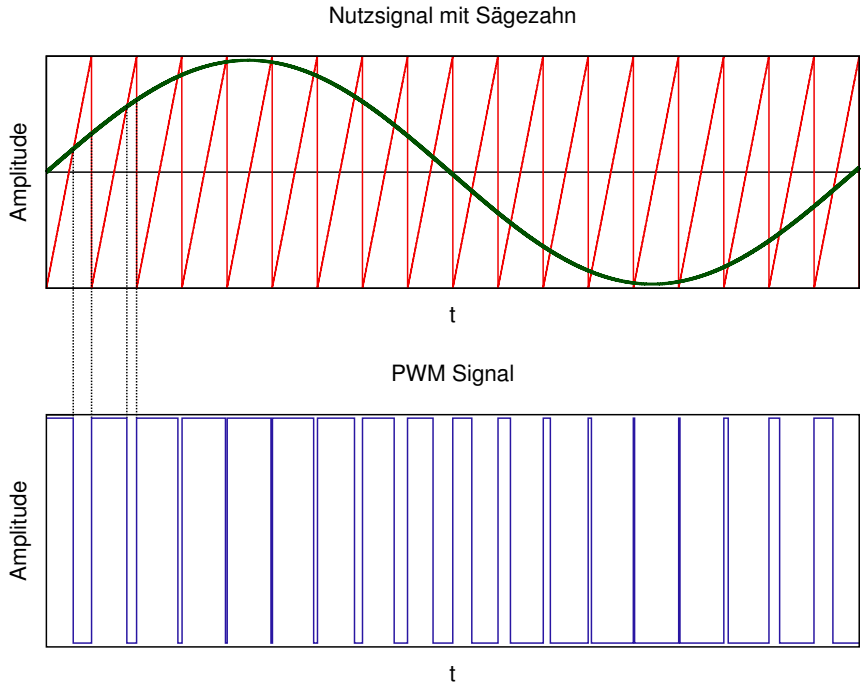
1. Geeignete Zähler/Vergleicherschaltungen
2. Mikrocontroller enthalten bereits PWM-Module

Aus analoger Spannung

1. Ein linear ansteigendes Signal (Dreieck- oder Sägezahn) wird mit dem analogen Eingangssignal verglichen. Je nach Wert liegt es eine längere Zeit unter dem Dreickssignal. An den Schnittpunkten wird das Ausgangssignal zwischen zwei Logikpegeln umgeschaltet

   $\longrightarrow$ Rechteck mit stufenlosveränderbarem Tastgrad




2. Die Zeitkonstanten einer astabilen Kippstufe werden mit dem analogen Eingangssignal beeinflusst

## Demodulation

Demodulation über einen Tiefpass $\longrightarrow$ arithmetischer Mittelwert des modulierten Signals

Z.B.: Schalter, der eine Heizung periodisch ein- und ausschaltet

Je nach Länge der Einschaltzeit zur Periodendauer ergibt sich die mittlere Heizleistung

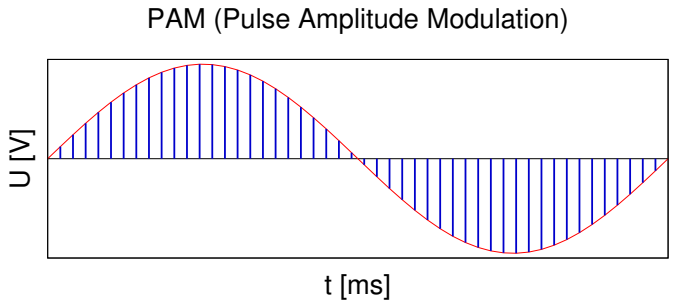
Die Temperatur des geheizten Objekts folgt langsam dem Ein- und Ausschaltvorgang

Durch die Wärmekapazität ergibt sich das Tiefpassverhalten zur Demodulation

## Anwendung

- Übertragung analoger Messwerte von Sensoren über lange Kabel oder Funk
- Dimmen von Leuchtdioden (LEDs); Schaltfrequenz (70 kHz), Wärmekapazität
- Leistungselektronik, Elektromotoren, Heizelemente, Dimmer, Schaltnetzteile, Klasse-D-Verstärker
  wenig Verlustenergie, da nur zwei Schaltpunkte: sperrend, durchgeschalten
- AD/DA-Umsetzer: Ein PWM-Signal kann einfach digital verarbeitet werden
  $\longrightarrow$ Positive Flanke setzt Zähler auf 0, negative Flanke liest den Zähler aus

### 11.5.2 Pulsamplitudenmodulation (PAM)




Der Amplitude des Signals werden in bestimmten Zeitabständen (time slots) einzelne “Proben” entnommen

$\longrightarrow$ das Signal wird abgetastet

**Anwendung:**

Übertragung mit Zeitmultiplexverfahren $\longrightarrow$ zwischen den PAM-Impulsen eines Kommunikationskanals können die PAM-Impulse anderer Kanäle übertragen werden

**Nachteil:** Hohe Störempfindlichkeit

**Verbesserung:**

Puls-Code-Modulation (PCM) $\longrightarrow$ Zeitdiskrete Werte der PAM werden quantisiert

$\longrightarrow$ wertdiskrete Folge

### 11.5.3 Pulsfrequenzmodulation (PFM)

Umso größer das Eingangssignal, desto mehr Pulse konstanter Dauer je Zeiteinheit

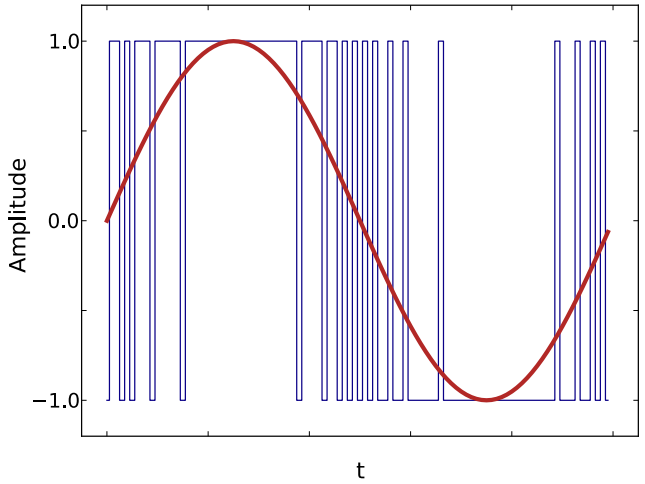
Deltamodulation:

Analoges Signal wird in gleichmäßigen Abständen abgetastet

Der Abtastwert wird mit dem vorherigen verglichen:

Ist er größer als sein Vorgänger $\longrightarrow$ 1-Signal

ist er kleiner $\longrightarrow$ 0-Signal




**Modulation:**

Für einen geeigneten Dynamikumfang

$\longrightarrow$ höhere Abtastfrequenz als als Nyquist-Shannon-Abtasttheorem

**Anwendung:**

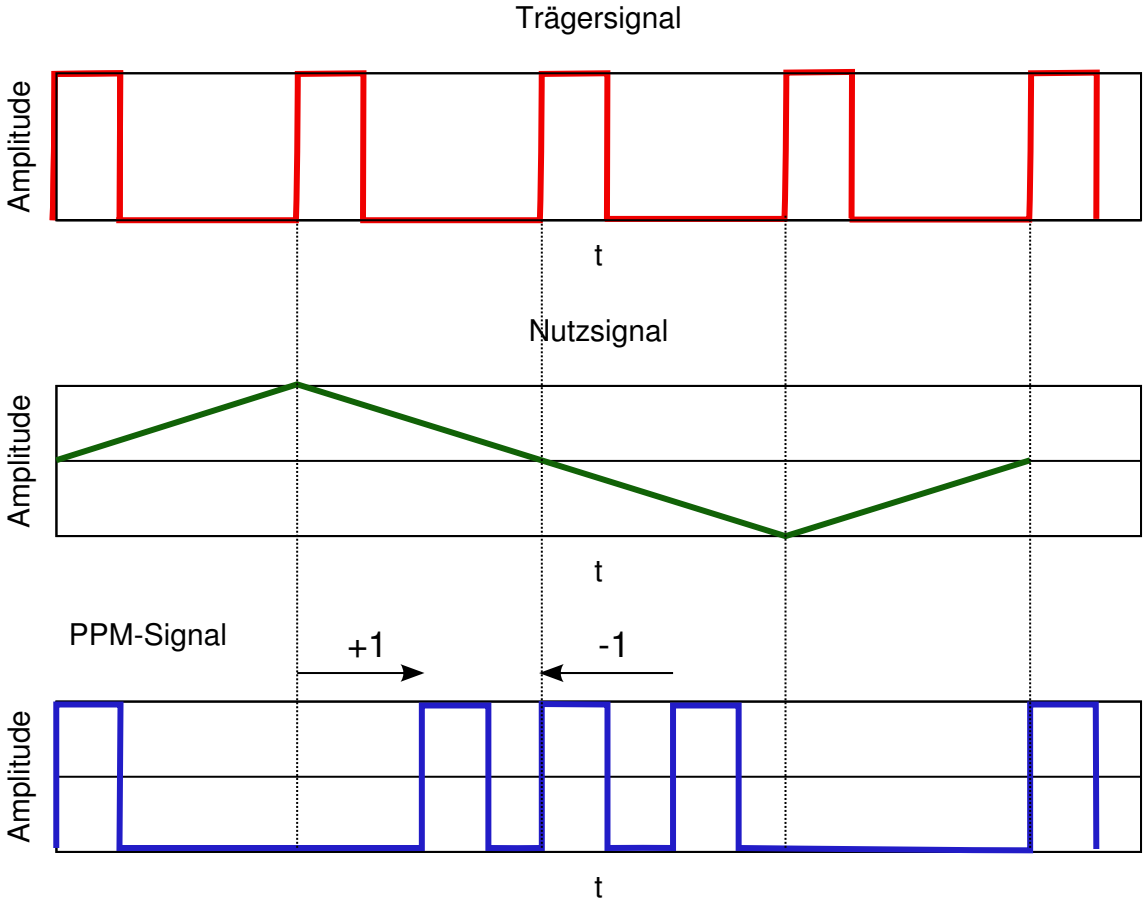
Steuerung von Schaltreglern und Gleichspannungswandlern

Umsetzung analoger Signalverläufe in eine binäre Folge

### 11.5.4 Pulsphasenmodulation (PPM)

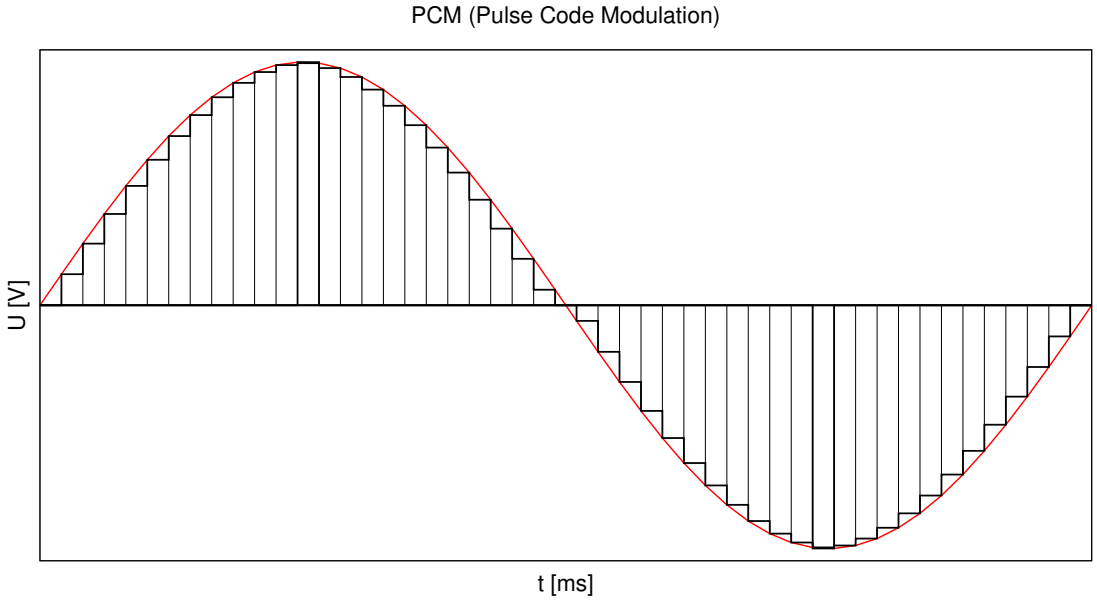
Pulspositionsmodulation (*pulse-position modulation*)

Die Information wird als zeitliche Verschiebung von Impulsen, relativ zu einem Referenztakt moduliert. Wird der Träger nicht moduliert, ergibt sich eine Folge von Rechteckimpulsen mit gleichem zeitlichen Abstand



### 11.5.5 Puls-Code-Modulation (PCM)

Amplitudendiskretes Verfahren




Ein Impulskamm (periodische Folge von kurzen Einzelpulsen) wird mit dem Eingangssignal multipliziert

Die Ausgabewerte werden quantisiert $\longrightarrow$ diskrete Stufen

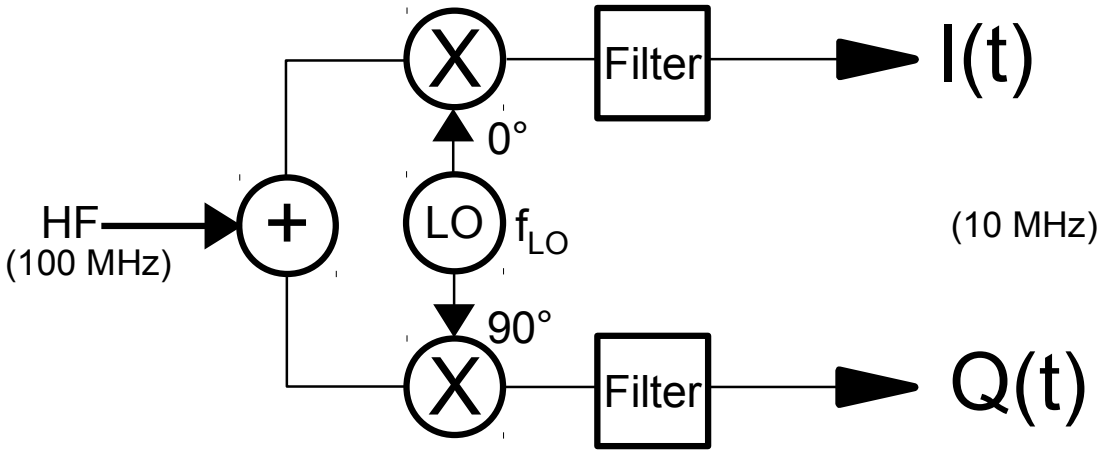
**Anwendung:**

Analog-Digital-Umsetzer für laufende Signalfolgen

Digitalisierung von Sprach- und Musiksignalen

### 11.5.6 IQ Modulation

Bei der Demodulation wird die Frequenz meist herabgesetzt:




Mischen (Multiplikation) des HF-Signals mit dem Sinussignal der lokalen Oszillators:

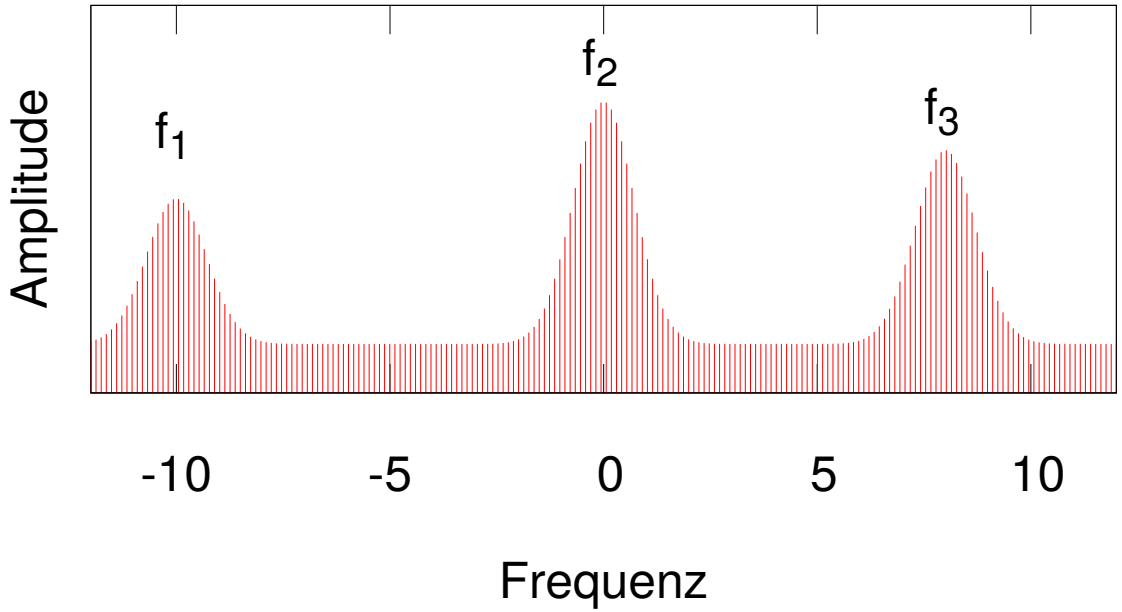
I-Signal (Phasenverschiebung $0^o$) und das Q-Signal (Phasenverschiebung $90^o$)

Komplexe Darstellung: $S_{\mathrm{IQ}}(t) = I(t) + j \cdot Q(t)$

Darin ist die gesamte Information des HF-Signals über eine bestimmte Bandbreite enthalten.

Mittels Software kann das Nutzsignal errechnet werden $\longrightarrow$ SDR (software defined radio)

Das Frequenzspektrum des komplexen $S_{\mathrm{IQ}}(t)$-Signals zeigt die enthaltenen Trägerfrequenzen ($f_1,\ f_2,\ f_3$)



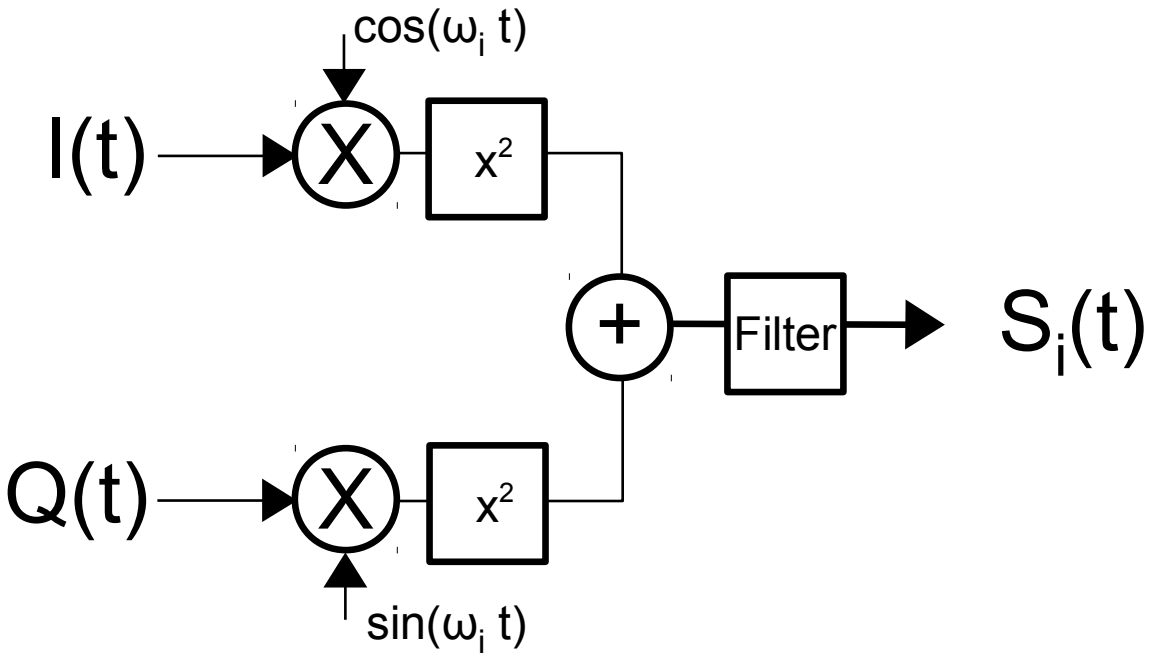

$f_2 = 0$ entspricht $f_{\mathrm{LO}}$

Die Nutzsignale $S_1(t)$, $S_2(t)$, $S_3(t)$ ergeben sich zu:

$S_i(t) = |\cos(\omega_i t) \cdot I(t) + j \sin(\omega_i t) \cdot Q(t)|$, wobei $S_2(t) = |I(t)|$, da $f_2 = 0$

### 11.5.7 IQ Demodulation

Ablauf der IQ-Demodulation für verschiedene Trägerfrequenzen:



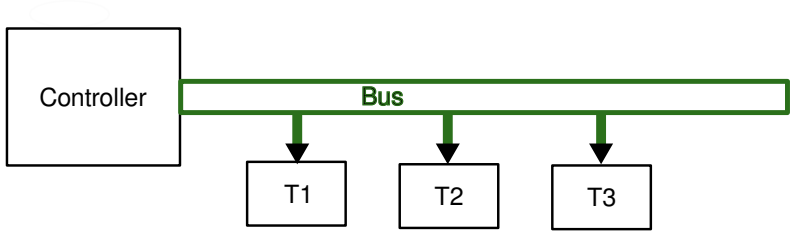

IQ-Signal: diskrete abgetastete Werte (Amplitude:8-16 Bit, Abtastrate: einige 100 kHz bis einige Mhz). In der Praxis wird noch ein Tiefpassfilter nachgeschalten (entfernt die Trägerfrequenz und reduziert das Rauschen). Meist wird auch noch die Abtastrate reduziert da das Nutzsignal im NF-Bereich liegt.

# 12 Datenverbindungen

## 12.1 Bus

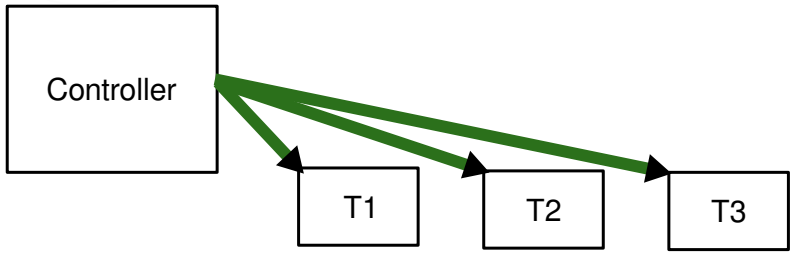
Datenübertragung zwischen mehreren Teilnehmern über einen Datenkanal

Die einzelnen Endgeräte werden durch eine Adresse unterschieden




## 12.2 Point to point

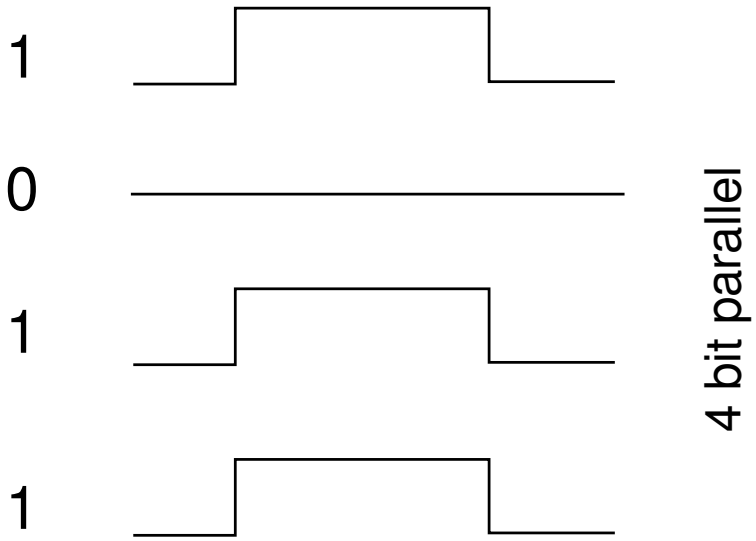
Für jeden Teilnehmer ist ein eigener Datenkanal (Leitung) notwendig

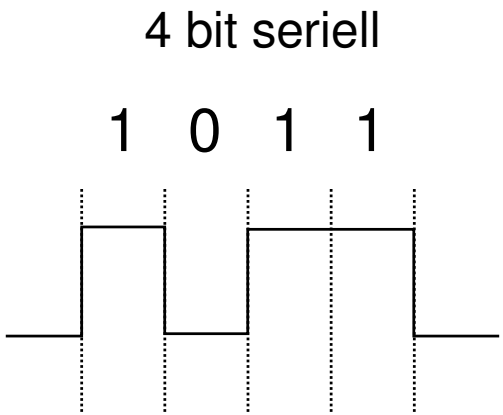



Datenaustausch zwischen den Endgeräten ist nicht vorgesehen

Unabhängig davon unterscheidet man zwischen:

## paralleler und serieller Verbindung



## 12.3 Parallele Datenverbindung

Jeweils ein Datenwort (z.B. ein Byte) wird gleichzeitig über die entsprechende Anzahl von Leitungen übertragen.

**Handshake-Leitungen** steuern zeitlichen Ablauf mehrerer Transaktionen

**Daten-Leitungen** übertragen die Daten

Meist viele Leitungen notwendig $\longrightarrow$ schnelle Verbindung

Beispiele:

Memory (RAM) $\longrightarrow$CPU (Bus)

Parallele Druckerschnittstelle (P2P)

GPIB (General purpose interface bus): Verbindet mehrere Messgeräte mit dem Computer

SCSI, PATA (IDE), ISA, PCI

### 12.3.1 GPIB

General Purpose Interface Bus (IEC625-Bus, IEEE488-Bus)

Parallelle Datenverbindung: 8 Bit Daten, 3 Steuerleitungen, 5 Busmangement
(ähnlich der Parallelen PC-Schnittstelle)

Ein Controller kann mit bis zu 15 Geräten am Bus kommunizieren.

Protokoll: ASCII-Code, Steuerzeichen zur Adressierung (LISTEN, TALK)

Übertragungsrate: bis zu 1Mbit/s

Genormte 24-polige Steckverbindung; max. Leitungslänge 20m (2m zwischen zwei Geräten)

Der Standard definiert nur den Verbindungsblauf und die gesendeten und empfangenen Daten sind vom Gerät abhängig.

## 12.4 Serielle Datenverbindung

Ein Datenwort (z.B. ein Byte) wird in einzelne Bits aufgelöst hintereinander über ein Leitungspaar übertragen.

**Handshake-Leitungen** möglich aber nicht notwendig.

Wenige Leitungen $\longrightarrow$ Taktrate bestimmt die Übertragungsrate

Beispiele:
Serielle (Modem)-Schnittstelle RS-232, RS-485
USB (Universal serial bus)
SATA, SAS

Man unterscheidet zwischen:

**synchroner und asynchroner serieller Übertragung**

### 12.4.1 Synchrone serielle Datenverbindung

Ein Master gibt einen Takt vor, mit einer Taktflanke liegen am Datenausgang die Daten an.

**I2C:**

Busverbindung bei der mehrere ICs mit einem Controller kommunizieren können.

Master sendet zuerst ein Adress-Byte, danach die Daten.

Adresse: vom Hersteller festgelegt, untersten 3 Bit durch Steuerpins veränderbar

Taktrate: einige kHz . . . 3.4 MHz

Vorteil: Ein Controller kann viele ICs mit 2 Pins ansprechen

Nachteil: Störanfällig, nur kurze Leitungen möglich; Steckverbindungen!!

## SPI (Serial Peripheral Interface):

Drei gemeinsame Leitungen, an denen jeder Teilnehmer angeschlossen ist:

SDO (Serial Data Out) bzw. MISO oder SOMI (Master in, Slave out)

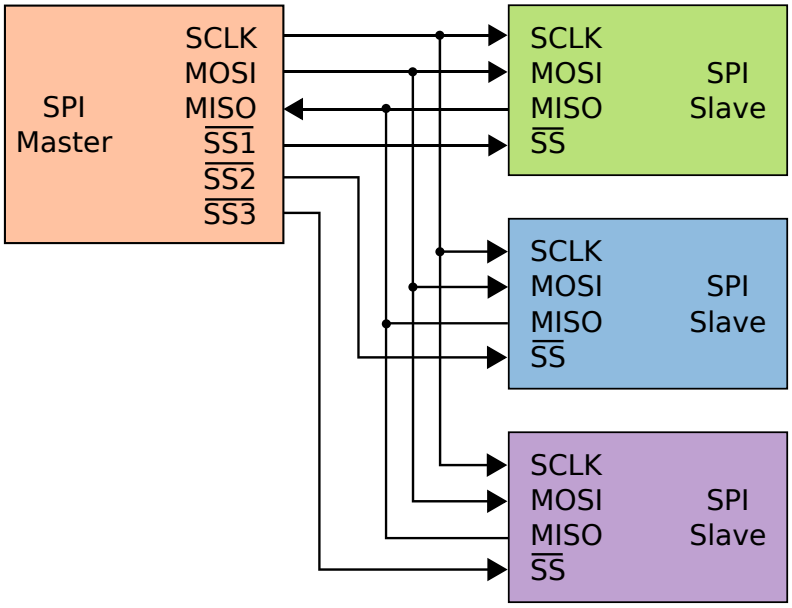
SDI (Serial Data In) bzw. MOSI oder SIMO (Master out, Slave in)

SCK (Serial Clock) bzw. SCLK, wird vom Master ausgegeben

Mehrere Chip-Select-Leitungen (SS, CS, STE) (vom Master gesteuert) zu jeden Slave.

Vollduplexfähig

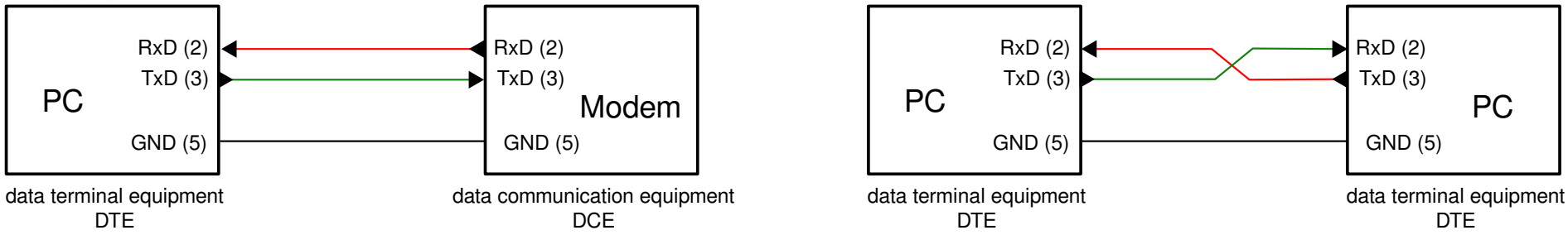
Taktrate: einige MHz



## 12.4.2 Asynchrone serielle Datenverbindung

### RS-232

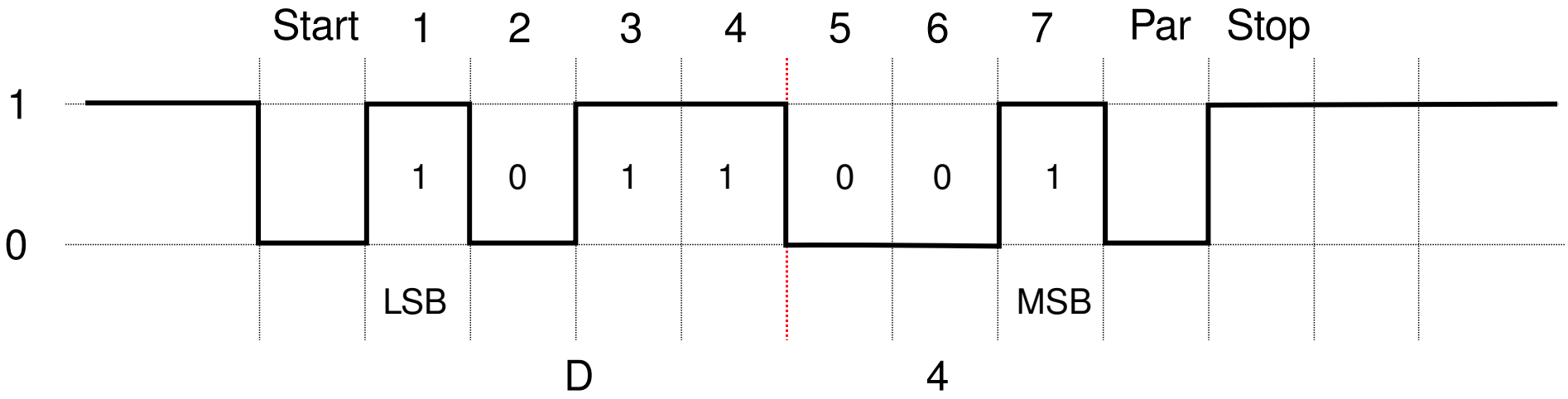
Zwei Geräte (z.B. PC-Modem, PC-PC) werden mit 2 Datenleitungen (TxD, RxD) verbunden




Übertragung: “Wörter” (5 - 9 Bits) $\longrightarrow$ Zeichen meist in ASCII (7/8 Bit) kodiert

Asynchrone (kein Taktsignal), bitserielle Übertragung über beide Datenleitungen

Spannungspegel: negative Logik: -3 V . . . -15 V $\longrightarrow$ logisch 1 +3 V . . . +15 V $\longrightarrow$ logisch 0




Beginn der Übertragung (Synchronisation des Empfängers) durch Startbit (logisch 0)

Danach 5-8 Datenbits

Ende der Übertragung durch 1-2 Stopbits (logisch 1)

Empfänger synchronisiert sich in die Mitte der einzelnen Datenbits und tastet folgende Bits mit eigener Bitrate ab

Bitrate (Baudrate) von Sender und Empfänger müssen nur annähernd gleich sein, da nach jedem Wort über das Startbit erneut synchronisiert wird.

Standard Baudraten: 110, 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400 bit/s

Zur Synchronisation sind weitere Steuerleitungen (DCD, DTR, DSR, RTS, CTS und RI) definiert, jedoch nicht notwendig

Stecker und Verkabelung:

Weitgehend genormt, jedoch viele Hersteller weichen ab!

DTE: Computer, 9 poliger Stecker DCE: Endgerät, 9 polige Buchse

Kabel DTE-DCE: Buchse $\longrightarrow$ Stecker (Pin 2 und 3 nicht ausgekreuzt)

Kabel DTE-DTE: Buchse $\longrightarrow$ Buchse (Pin 2 und 3 ausgekreuzt $\longrightarrow$Nullmodemkabel)

Identifizierung von DTE- und DCE-Geräten:

Messung der Spannung zwischen GND und TxD bzw. RxD

| | DTE | DCE |
|---|---|---|
| TxD - GND | -3. . . -15 V | ca. 0 V |
| RxD - GND | ca. 0 V | -3. . . -15 V |

Moderne Geräte: unbeschaltete Anschlüsse $\longrightarrow$ Ausgangstreiber stillgelegt

$\longrightarrow$ geeigneter Widerstand zwischen Signalanschluss und GND

## Programmierung

In UNIX und Linux: RS-232 (Serielle Schnittstelle) lange Tradition

Die meisten seriellen Bausteine (auch USB-Serial-Converter) werden durch den Kernel automatisch unterstützt

$\longrightarrow$ Device files **`/dev/ttySn`** oder **`/dev/ttyUSBn`** vorhanden (n: Zahl)

Ansprechen der Schnittstelle von der Kommandozeile:

1. Initialisieren: **`stty -F /dev/ttySn 9600 raw -echo`**
2. Senden: **`echo -en "text`**`\n`**`" > /dev/ttyS0`**
3. Empfangen: **`cat /dev/ttyS0`**

Programm:

Setzen der Schnittstellenparameter mittes Systemaufruf **`tcsetattr`**

Das Devicefile kann mittels jeder Programmiersprache als Datei geöffnet, gelesen und beschrieben werden. (Windows: Ähnlich; Name der Datei: **`COMn`**)

Python: Das Modul “serial” erlaubt auch eine Kontrolle über die Steuerleitungen.

# 13 Zufall

Anwendung: Simulationen, Kryptographie, “Glücksspiel”

## 13.1 Erzeugung von Zufallszahlen

### 13.1.1 Software

Immer “Pseudozufallszahlen” $\longrightarrow$ deterministisch erzeugt

**Linearer Kongruenzgenerator**

$$y_i = (ay_{i-1} + c) \mod m$$

Modul: $m \in \{2, 3, 4, \ldots\}$ Faktor: $a \in \{1, \ldots, m-1\}$

Inkrement: $c \in \{1, \ldots, m-1\}$ Startwert: $y_1 \in \{0, \ldots, m-1\}$

Verwendung: Pseudozufallszahlen für Simulationen

Für Kryptographie ungeeignet $\longrightarrow$ aus wenigen Werten der Zahlenfolge a und b berechenbar

### 13.1.2 Hardware

"Echte" Zufallszahlen möglich

Rauschen, Radioaktiver Zerfall, freilaufende Oszillatoren, ...

Zugriffe auf PC-Hardware:

Linux: 2 Devicefiles, die "zufällige" Bytes liefern

**`/dev/random`** blockiert, wenn zuwenig "Zufall" an der Hardware anliegt

**`/dev/urandom`** liefert immer Bytes, wenn auch nicht mehr "zufällig"

Erzeugen einer Datei mit 1000 zufälligen Bytes:

```
dd if=/dev/urandom of=z.dat bs=1 count=1000
```

Häufiges praktisches Problem: Rasche Bereitstellung grosser Mengen an Zufallszahlen

## 13.2 Grundbegriffe

**Aufgabe:**

Einem Zufallsexperiment ein mathematisches Modell zuzuordnen, mit dem bestimmte Aussagen über den Ausgang des Experimentes getroffen werden können

**Zufallsexperiment:**

Vorgang, der unter reproduzierbaren, genau definierten Umständen abläuft und dessen Ausgang auf irgendeine Weise unbestimmt ist. (Klassisches Beispiel: Wurf einer Münze)

**Begriffe:**

Grundmenge $\Omega$: Alle möglichen Ausgänge des Experimentes

Jede Funktion $X : \Omega \to \mathbb{R}$ heißt **Zufallsgröße**

- Ist $\Omega$ endlich oder abzählbar unendlich: Diskretes Modell (z.B.: $\Omega = \{\mathrm{Kopf}, \mathrm{Zahl}\}$)
- Ist $\Omega$ überabzählbar unendlich: Kontinuierliches Modell

Zuordnung von Wahrscheinlichkeiten zu den einzelnen Elementen der Grundmenge

$\longrightarrow$ Wahrscheinlichkeitsfunktion $P : \Omega \to [0, 1]$

Sie muss normiert sein: $\sum_{\omega \in \Omega} P(\omega) = 1$

Erwartungswert:

$$\mu = \langle X \rangle = \sum_{x \in M_X} x P(x)$$

Varianz: mittlere quadratische Abweichung vom Mittelwert

$$\sigma_X^2 = \mathrm{Var}(X) = \sum_{x \in M_X} P(x)(x - \mu)^2$$

Verschiebungssatz:

$$\mathrm{Var}(X) = \langle X^2 \rangle - \langle X \rangle^2$$

$X$: Zufallsgröße, $M_X$: Grundraum, $P$: Wahrscheinlichkeitsverteilung

### 13.2.1 Diskrete Verteilungen

**Gleichverteilung:** Alle Werte sind gleich wahrscheinlich (Laplace Modell)

$X$ heißt gleichverteilt $X \sim D_m$ wenn:

$$M_X = \{1, \ldots, m\} \quad \text{und} \quad P(x) = \frac{1}{m} \quad \forall x \in M_X$$

$$\langle X \rangle = \frac{m+1}{2}, \qquad \mathrm{Var}(X) = \frac{m^2 - 1}{12}$$

Anwendungen: Würfel, Zufallszahlen, Roulette, usw.

**Alternativverteilung:** Versuch mit nur zwei möglichen Ausgängen

mit Wahrscheinlichkeit $\vartheta$ bzw. $1 - \vartheta$. Die Zufallsgröße kann mit 1 oder 0 beschrieben werden

X heißt alternativverteilt $X \sim A_\vartheta$ wenn:

$$M_X = \{0, 1\} \quad \text{und} \quad P(1) = \vartheta, \qquad P(0) = 1 - \vartheta$$

$$\langle X \rangle = \vartheta, \quad \mathrm{Var}(X) = \vartheta(1 - \vartheta)$$

Anwendungen: Münzwurf, Gut-Schlecht-Prüfungen, usw.

## Binomialverteilung:

Eine Folge von $n$ sich gegenseitig nicht beeinflussenden Alternativversuchen mit Erfolgswahrscheinlichkeit $\vartheta$. Betrachtet man die Anzahl der Erfolge $X$

$\longrightarrow$ Zufallsgröße $X$ ist binomialverteilt

Andere Sichtweise des Münzwurfes:

Man wirft eine Münze n-mal und zählt, wie oft man Kopf erhalten hat

$X$ heißt binomialverteilt $X \sim B_{n,\vartheta}$ wenn:

$M_X = \{0, 1, \ldots n\}$ und $P(x) = \binom{n}{x} \vartheta^x (1-\vartheta)^{n-x}$

$$\langle X \rangle = n\vartheta, \quad \mathrm{Var}(X) = n\vartheta(1-\vartheta)$$

Anwendungen: Ziehen mit Zurücklegen, Anzahl von gewürfelten Sechsen, Auftreten von einzelnen Zeichen in einem Text, Auftreten einzelner Lottozahlen (?!), ...

## Hypergeometrische Verteilung:

Verteilung des *Ziehens ohne Zurücklegen*

Aus einer Grundgesamtheit $N$ mit $A$ besonderen Stücken wird eine Stichprobe vom Umfang $n$ gezogen. Die Zufallsgröße $X$ beschreibt die Anzahl der besonderen Stücke in der Stichprobe.

## Poissonverteilung:

Wenn das Auftreten von Ereignissen in kleinen Zeitintervallen eine konstante, vom Zeitpunkt unabhängige, Auftrittswahrscheinlichkeit hat, die Ereignisse in den verschiedenen Zeitintervallen voneinander unabhängig sind und sie nur einzeln auftreten.

X heißt poissonverteilt $X \sim P_\lambda$ wenn: $M_X = \{0, 1, 2, 3 \ldots\}$ und $P(x) = \frac{\lambda^x}{x!} \mathrm{e}^{-\lambda}$

$$\langle X \rangle = \lambda, \quad \mathrm{Var}(X) = \lambda$$

Man erhält die Poissonverteilung auch aus einer Binomialverteilung für kleine $\vartheta$ und große $n$: Dann ist $\lambda = n\vartheta$. Ab $n \geq 30$ und $\vartheta \leq 0.1$ ist dieser Übergang für die meisten Anwendungen hinreichend genau.

Anwendungen: Zerfall von Teilchen, Meteoriten Einschläge auf Planeten, Tippfehler in einem Text, Anrufe an einem Telefonanschluss, ...

## Geometrische Verteilung:

Die Anzahl der Versuche, die benötigt werden, um bei einer Folge von unabhängigen Alternativversuchen mit konstanter Wahrscheinlichkeit den ersten Erfolg zu erzielen

X heißt geometrisch verteilt $X \sim G_\vartheta$ wenn:

$M_X = \{1, 2, 3 \ldots\}$ und $P(x) = (1-\vartheta)^{x-1}\vartheta$

$$\langle X \rangle = \frac{1}{\vartheta}, \quad \mathrm{Var}(X) = \frac{1-\vartheta}{\vartheta^2}$$

Anwendungen: Anzahl der Würfe eines Würfels, bis zum ersten Mal eine Sechs gewürfelt wird

**Bemerkung**: Wenn man gerade eine Sechs gewürfelt hat, so wird es weder wahrscheinlicher noch unwahrscheinlicher, dass bei den nächsten Würfen wieder eine Sechs geworfen wird. Dasselbe gilt wenn man schon lange keine Sechs mehr gewürfelt hat. Die Wahrscheinlichkeit ist immer $\frac{1}{6}$ und der Würfel erinnert sich nicht an das Ergebnis des letzten Wurfes.

$\longrightarrow$ Die Verteilung der Würfe ändert sich bis zum nächsten Erfolg nicht

Analoges gilt selbstverständlich auch für das Auftreten von Lottozahlen, radioaktiven Zerfällen, Einschlägen von Meteoriten, usw.

$\longrightarrow$ Gedächtnislosigkeit der geometrischen Verteilung

### 13.2.2 Stetige Zufallsgrößen und Verteilungen

Erweiterung der vorigen Begriffe auf Zufallsgrößen, die ein Kontinuum von Werten ($M_X$ ist reelles Intervall oder ganz $\mathbb{R}$) annehmen können

$\longrightarrow$ **stetige Zufallsgrößen**

Physik: Viele Messgrößen können stetige Werte annehmen

Diskrete Zufallsgröße $\longrightarrow$ Diskretisierung wird immer feiner $\longrightarrow$ Grenzfall

$\longrightarrow$ stetige Funktion, die **Dichtefunktion** $f$ oder $\phi$

Eigenschaften:

$$f : M_X \longrightarrow \mathbb{R}^+, \text{ integrierbar}$$

$$\int_{M_X} f(x)dx = 1$$

$$P(\{x \in M_X : a \leq x \leq b\}) = \int_a^b f(x)dx$$

Die Wahrscheinlichkeit einer stetigen Verteilung ist durch ein Integral definiert

$\longrightarrow$ Einem einzelnen Punkt ist keine Wahrscheinlichkeit zuzuordnen, sondern nur mehr einem Intervall

Zusätzlich lässt sich auch die Verteilungsfunktion $F$ einer Zufallsgröße X angeben

$$F(\xi) := P(\{x \in M_X : x \leq \xi\}) \quad \forall \xi \in \mathbb{R}$$

Sie gibt die Wahrscheinlichkeit aller Werte an, die unterhalb von $\xi$ liegen

Ist X diskret, so ist $F(x) = \sum_{x_i \leq x} P(x_i)$ eine Treppenfunktion

Ist X stetig, so ist F eine stetige Funktion mit:

$F'(x) = f(x), \quad F(x) = \int_{-\infty}^{x} f(t)dt$ und es gilt
$P(\{x \in M_X : a \leq x \leq b\}) = F(b) - F(a)$

## Wichtige Lage- und Streuparameter:

**$\alpha$-Fraktil $x_\alpha$:** $F(x_\alpha) = \alpha$

Jene Werte, unterhalb denen $(\alpha \cdot 100)$ %) aller Werte liegen

Wichtige Sonderfälle:

- Das 0.5-Fraktil (der Median)

  Jene Werte, unterhalb denen 50% aller Werte liegen

  Teilt den Wertebereich in zwei gleichwahrscheinliche Teile

- Das 0.25-Fraktil und das 0.75-Fraktil (unteres bzw. oberes Quartil)

  Jene Werte, unterhalb denen 25% bzw. 75% aller Werte liegen

**Erwartungswert $\langle X \rangle$:**

$$\langle X \rangle = \int_{M_X} x f(x) dx$$

bzw. für die Funktion einer Zufallsgröße

$$\langle g(X) \rangle = \int_{M_X} g(x) f(x) dx$$

**Varianz $\mathrm{Var}(X) = \sigma^2$:**

$$\mathrm{Var}(X) = \int_{M_X} (x - \langle X \rangle)^2 f(x) dx$$

$\sqrt[+]{\mathrm{Var}(X)} = \sigma$: Standardabweichung

## Stetige Gleichverteilung

Ganz analog zum diskreten Fall

$X$ heißt stetig gleichverteilt auf $[a, b]$ ($X \sim S_{a,b}$) wenn:

$$M_X = [a, b] \quad \text{und} \quad f(x) = \frac{1}{b-a} \quad \forall x \in M_X$$

$$\langle X \rangle = \frac{a+b}{2}, \quad \mathrm{Var}(X) = \frac{(b-a)^2}{12}$$

Anwendungen: Kontinuierliche Zufallszahlen, usw.

## Exponentialverteilung

Der stetige Fall der geometrischen Verteilung heißt Exponentialverteilung

$X$ heißt exponentialverteilt, wenn $M_X = \mathbb{R}^+$ und $f(x) = \lambda \mathrm{e}^{-\lambda x}$

$$\langle X \rangle = \frac{1}{\lambda}, \quad \mathrm{Var}(X) = \frac{1}{\lambda^2}$$

Anwendungen: Lebensdauern, Zerfallszeiten (Radio-Carbon-Datierung), Wartezeiten

## Normalverteilung (Gaußverteilung)

Die wahrscheinlich wichtigste und häufigste Verteilung. Sie ist der stetige Grenzfall der Binomialverteilung ($\longrightarrow$ „Galtonbrett“)

$X$ heißt standard-normalverteilt $X \sim N(0,1)$, wenn

$$M_X = \mathbb{R} \quad \text{und} \quad f(x) = \frac{1}{\sqrt{2\pi}} \mathrm{e}^{-\frac{x^2}{2}}$$

$$\langle X \rangle = 0, \quad \mathrm{Var}(X) = 1$$

Praktische Berechnung:

Das Maximum $\langle X \rangle = 0$ und die Wendepunkte $\langle X \rangle \pm \sqrt[+]{\mathrm{Var}(X)} = \pm 1$ der Dichtefunktion müssen passend zum jeweiligen Fall verschoben werden

Durch die Transformation $Y = \sigma X + \mu$ eines $X \sim N(0,1)$ erhält man wieder eine normalverteilte Zufallsgröße $Y \sim N(\mu, \sigma^2)$ mit $\langle Y \rangle = \mu$, $\mathrm{Var}(Y) = \sigma^2$ und der Dichtefunktion:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} \mathrm{e}^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Diese Funktion ist nicht analytisch integrierbar (Reihenentwicklung oder numerische Methoden). Das bedeutet, dass es nur mehr numerisch möglich ist eine Wahrscheinlichkeit zu berechnen. Durch die Transformation lässt sich jedoch immer eine Standardisierung erreichen. Die Werte der Standard-Normalverteilung N(0,1) sind in Tabellen zu finden, aber auch die meisten Tabellen-Kalkulationsprogramme verfügen über entsprechende Funktionen.

Anwendungen: Die Normalverteilung tritt immer dann auf, wenn sich eine Zufallsgröße aus einer Summe vieler unabhängiger Einflussgrößen ergibt.
z.B.: Messfehler, Abmessungen von Bauteilen, Länge von Grashalmen auf einem Feld, usw.
Aber auch z.B. die Antreffwahrscheinlichkeitsdichte $|\psi(x,t)|^2$ eines Teilchens in der Quantenmechanik ist eine Normalverteilung.

## 13.3 Testen von Zufallszahlen

Verteilung und Reihenfolge der Zufallszahlen testen

`http://dilbert.com/strips/comic/2001-10-25/`

### 13.3.1 Testen der Verteilung

- Zählen der 1(0):

  Testen auf Gleichverteilung und berechnen der Wahrscheinlichkeit der Abweichung

- Kategorisieren einzelnen Bytes:

  Gleichverteilung aller Ereignisse

  Binominalverteilung der Einzelereignisse

- Bestimmung der Informationsgehalts (Entropie)

  Ist $X$ eine Folge von diskreten Zufallszahlen $Z = \{z_1, z_2, \ldots, z_m\}$:

  Informationsgehalt $I(p) = -\log_2 p$ für ein Ereignis mit der Wahrscheinlichkeit $p$

  $z \in Z$, $p_z = P(X = z)$: Wahrscheinlichkeit mit der das Zeichen z des Alphabets auftritt

  Entropie eines Zeichens $\longrightarrow$ Erwartungswert des Informationsgehalts:

  $H_1 = \sum_{z \in Z} p_z \cdot I(p_z) = -\sum_{z \in Z} p_z \cdot \log_2 p_z \longrightarrow$ Anzahl der Bits für $Z$

- Bildung eines Zahlenpaars, berechnen eines bekannten Integrals mit MC-Simulation
- $\chi^2$-Test:

  Annahme: Die Abweichung von der berechneten Häufigkeit folgt einer Gaußverteilung

  Problem: Die Summe der Abweichungen verschwindet

  Lösung: Betrachtung der Abweichungsquadrate: Gaußverteilung $\longrightarrow \chi^2$-Verteilung
- Spektraltest:

  Jeweils 3 Zufallszahlen zu Tripel zusammenfassen; normieren $\longrightarrow$ Punkt $R_3$

  $\longrightarrow$ graphische Darstellung $\longrightarrow$ "gleichmäßig gefärbter" Würfel

  Beispiel:

  RANDU: $m = 2147483648 = 2^{31}; \quad a = 65539 = 2^{16} + 3; \quad c = 0; \quad y_0 = 1$

  C: $m = 2^{48}; \quad a = 25214903917; \quad c = 11; \quad y_0 = 0$

  (Beispiel: LinK.py c 5000; LinK.py randu 5000; gnuplot: splot datafile)

### 13.3.2 Testen der Reihenfolge

“ 1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6 . . . ” $\longrightarrow$ besteht alle Verteilungstests

- Binärer Runtest:

  | 0 | 111 | 00 | 1 | 00 | 1111 | $\longrightarrow$ Zählen der “runs” = Blöcke gleichen Ziffern in Folge

- Allgemeiner Runtest:

  Man bestimmt zunächst den Median der Stichprobe:

$$m = \begin{cases} x_i & mit\ i = \left(\frac{n+1}{2}\right) \qquad n\ ungerade \\ \\ \frac{x_j + x_l}{2} & mit\ j = \left(\frac{n}{2}+1\right),\ l = \left(\frac{n}{2}\right) \quad n\ gerade \end{cases}$$

  Nun bildet man eine Folge von 1 und 0, in der 1 an der i-ten Stelle für $x_i > m$ bzw. 0 wenn $x_i < m$ steht.

  Eine Unterfolge von gleichen Zahlen heißt *Run*.

  Die Gesamtzahl $R$ dieser Runs wird gezählt.

$\longrightarrow$ berechnen der Wahrscheinlichkeit der Abweichung

Für die Daten selbst muss keine Verteilung angenommen werden um ein Ergebnis zu erhalten
Es wird versucht zu bestimmen, ob die Daten korreliert sind, also nicht aus unabhängigen Beobachtungen stammen.

Annahme: Die Zahl der Runs $R$ ist annährend normal (binomial) verteilt
oder hypergeometrische Verteilung $\longrightarrow$ Wald–Wolfowitz Runtest

Stammt die Stichprobe aus unabhängigen Beobachtungen $\longrightarrow$ Die Anzahl der Runs $R$:

$$\langle R\rangle = \frac{n}{2} + 1; \quad \mathrm{Var}(R) = \frac{n-1}{4}.$$

Getestet wird: Hypothese: Unabhängigkeit $\leftrightarrow$ Alternative: positive Korrelation (weniger Runs)
Kriterium: der sog. *P-Wert*, $P(\{x \in \mathbb{R} : x \leq R\}) = F(R) \quad F: \ N, \ B$ oder $H$
P-Wert: Wahrscheinlichkeit, mit der bei diesen Test-Parametern ($\langle R\rangle, \mathrm{Var}(R)$) weniger oder gleich viele Runs entstehen, unter der Voraussetzung, dass die Daten nicht korreliert sind.
Man kann den P-Wert als Entscheidungskriterium für das Bestehen des Testes nehmen. Man beachte, dass der P-Wert nur die Aussage trifft, dass bei wiederholtem Ausführen des Tests auf einen Datensatz mit den gleichen Parametern, dieser mit einer Wahrscheinlichkeit $F(R)$ (zu $F(R)$%) besteht.

Man kann den Test noch zusätzlich verschärfen, indem man die Differenzen aufeinanderfolgender Stichproben-Werte betrachtet. Man berechnet

$$MQD = \frac{1}{n-1}\sum_{i=2}^{n}(x_i - x_{i-1})^2.$$

Gilt Unabhängigkeit, dann gilt auch $\langle MQD\rangle = 2\mathrm{Var}(X)$.
Man schätzt $\mathrm{Var}(X)$ mit der Stichprobenvarianz $s^2$ und bildet $d = \frac{MQD}{s^2}$.
Es gilt $d$ ist näherungsweise normalverteilt mit

$$\langle d\rangle = 2, \quad \mathrm{Var}(d) = \frac{n-2}{n^2}.$$

Der P-Wert kann analog der ursprünglichen Testvorschrift berechnet werden.

- Empirische Tests:

  z.B. Pokertest:

  Einteilung in 5er Gruppen (= 1 Blatt) Testen auf die Möglichkeiten beim Pokerspiel und berechnen der zugehörigen Wahrscheinlichkeit $\longrightarrow \chi^2$-Test

`http://de.wikipedia.org/wiki/Kryptographisch_sicherer_Zufallszahlengenerator`

Teile Zufallszahlen aus [0, 1) in d Gruppen

$$[i/d,\ (i+1)/d), i = 0, \ldots, (d-1))$$

Wahrscheinlichkeiten, dass $m$ verschiedene Werte in Auswahl vom Umfang $n$:

$$p_{m,n} = S(n,m)\frac{d(d-1)\cdot\ldots\cdot(d-m+1)}{d^n}$$

mit

$$S(n,m) = \frac{1}{m!}\sum_{i=0}^{m}(-1)^{m-i}\binom{m}{i}i^n$$

z.B.: Ziffern 0 . . . 9 in 5er Kombinationen

| | | |
|---|---|---|
| 5 verschiedene | 64321 | $\frac{10 \cdot 9 \cdot 8 \cdot 7 \cdot 6}{10^5}$ |
| 4 verschiedene (1 Paar) | 25672 | $\frac{\binom{5}{2} 10 \cdot 9 \cdot 8 \cdot 7}{10^5}$ |
| 3 verschiedene (2 Paare) | 24245 | $\frac{\binom{5}{2}\binom{3}{2}\frac{1}{2!} 10 \cdot 9 \cdot 8}{10^5}$ |
| 3 verschiedene (Drilling) | 59545 | $\frac{\binom{5}{3} 10 \cdot 9 \cdot 8}{10^5}$ |
| 2 verschiedene (Full) | 22266 | $\frac{\binom{5}{3} 10 \cdot 9}{10^5}$ |
| 2 verschiedene (Poker) | 44944 | $\frac{\binom{5}{1} 10 \cdot 9}{10^5}$ |
| 1 verschiedene (Grande) | 33333 | $\frac{10}{10^5}$ |

# 14 Laplace Gleichung

Berechnung des Potentials einer Elektrodenanordnung

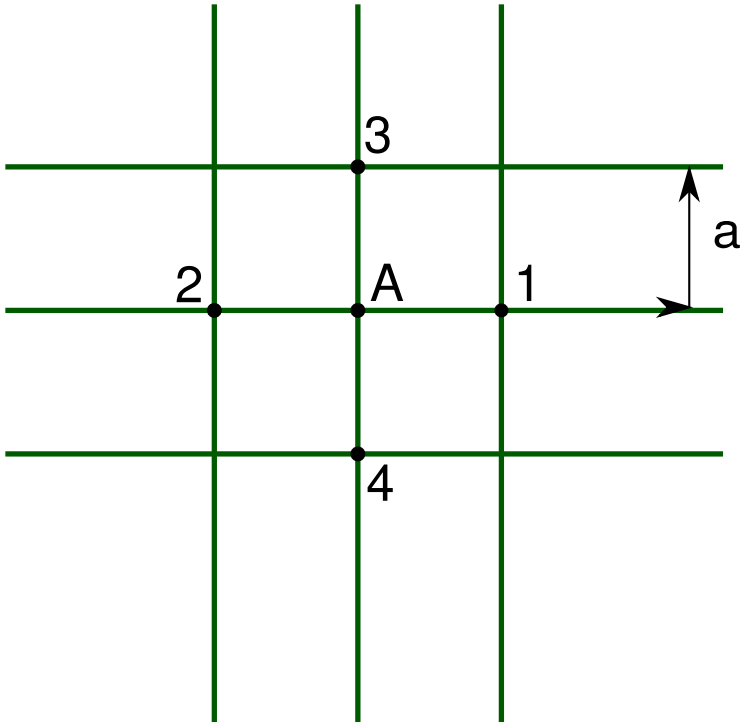
Zweidimensional:

$$\Delta U = \frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} = 0$$

$U = U(x, y)$... Potentialverteilung

Voraussetzung: Ladungsfreier Raum und Randbedingungen bekannt

# 14.1 Näherungslösung




Netz mit der Maschenweite $a$

Für jeden Punkt A gilt:

$$(\Delta U)_A = (\frac{\partial^2 U}{\partial x^2})_A + (\frac{\partial^2 U}{\partial y^2})_A = 0$$

Entwicklung für A um eine Koordinatenachse:

$$U_1 - U_A = a(\frac{\partial U}{\partial x})_A + \frac{a^2}{2}(\frac{\partial^2 U}{\partial x^2})_A$$

$$U_2 - U_A = -a(\frac{\partial U}{\partial x})_A + \frac{a^2}{2}(\frac{\partial^2 U}{\partial x^2})_A$$

Für die 2. Ableitung nach x und y gilt:

$$(\frac{\partial^2 U}{\partial x^2})_A = \frac{1}{a^2}[(U_1 - U_A) + (U_2 - U_A)]$$

$$(\frac{\partial^2 U}{\partial y^2})_A = \frac{1}{a^2}[(U_3 - U_A) + (U_4 - U_A)]$$

Einsetzen in die Differentialgleichung ergibt:

$$(\Delta U)_A = (\frac{\partial^2 U}{\partial x^2})_A + (\frac{\partial^2 U}{\partial y^2})_A = 0$$

$$U_A = \frac{U_1 + U_2 + U_3 + U_4}{4}$$

### 14.1.1 Iterative Berechnung

Gitter mit der Maschenweite $a$:

$i$: Index der Punkte in x-Richtung

$j$: Index der Punkte in y-Richtung

Die Potentialverteilung zu Beginn der Iteration ist vorgegeben.

Es gibt Punkte mit festem Potential (Elektroden) und variablen Potential (freier Raum)

Bei jedem Iterationsschritt wird für die freien Punkte das neue Potential $U$ berechnet:

$$U_{i,j}^{neu} = \frac{1}{4}(U_{i+1,j} + U_{i-1,j} + U_{i,j+1} + U_{i,j-1})$$

Nach jedem Iterationsschritt kann die Annäherung an die stabile Löung durch Berechnung der Residuen verfolgt werden: $r_{i,j} = U_{i,j}^{neu} - U_{i,j}$

Abbruchkriterium: $||r_{i,j}||$ oder $\max(r_{i,j})$

### 14.1.2 Verbesserung der Konvergenz

Simultanes Überrelaxations-Verfahren (SOR)

Die Residuen werden zur Berechnung der neuen Funktionswerte verwendet:

$$U_{i,j}^{neu} = U_{i,j} + \omega r_{i,j}$$

Der Parameter $\omega$ soll die Güte der Konvergenz beeinflussen:

Der Fehler steigt meist vorerst an, fällt jedoch anschließend

Weitere Optimierungen:

1. Durchlauf des Gitters
   (a) Sequentieller Durchlauf: Fehlerforpflanzung leichter möglich
   (b) In gerade und ungerade Punkte unterteilen:
   Die Funktionswerte an geraden Gitterpunkten sind nur von ungeraden Gitterpunkten abhägig und umgekehrt $\longrightarrow$ einfache Aufteilung auf mehrere Prozesse
2. Variation des Relaxationsparameters:
   Chebyshev-Beschleunigung: Relaxationsparameter ($\omega$) wird nach jedem Halbschritt geändert

## 14.2 Berechnung des E-Feldes

$$\mathbf{E} = -\nabla U = \begin{pmatrix} -\frac{\partial U}{\partial x} \\ -\frac{\partial U}{\partial y} \end{pmatrix}$$

Die partiellen Ableitungen von U erhält man aus den Differenzenquotienten in $x$- und $y$-Richtung:

$$\frac{\partial U}{\partial x} \sim \frac{\Delta U_x}{\Delta x} = \frac{U_{i,j} - U_{i-1,j}}{a} + O(\Delta x)$$
$$\frac{\partial U}{\partial y} \sim \frac{\Delta U_y}{\Delta y} = \frac{U_{i,j} - U_{i,j-1}}{a} + O(\Delta x)$$

Für “innere” Punkte kann auch der zentrierte Differenzenquotient verwendet werden:

$$\frac{\partial U}{\partial x} \sim \frac{U_{i+1,j} - U_{i,j-1}}{2a} + O(\Delta x^2)$$

## 14.3 Berechnung der Kapazität

Die Kapazität C ergibt sich aus $C = \frac{Q}{U}$ Q. . . Ladung, U. . . Spannung (Potentialdifferenz)

Ein Leiter ist im Inneren feldfrei $\longrightarrow$ das Potential ist konstant $\longrightarrow$
Ladung ausschließlich auf der Oberfläche der Leiter verteilt

Ladungsverteilung: $\nabla \cdot \mathbf{D} = \rho$, $\mathbf{D}$ . . . elektrische Flussdichte, $\rho$. . . Ladungsdichte

Flächenladungsdichte: $\sigma = \mathbf{D}_o \cdot \mathbf{n}$
$\mathbf{D}_o$. . . Flussdichte an der Leiteroberfläche, $\mathbf{n}$. . . Normalvektor (Leiteroberfläche)
$\mathbf{D}_i$, die Flussdichte im Inneren des Leiters verschwindet

Ladungsintegration über die Leiteroberfläche $O$:

$$Q = \iint_O \sigma \, dA = \iint_O \mathbf{D}_o \cdot \mathbf{n} \, dA = \epsilon \iint_O \mathbf{E} \cdot \mathbf{n} \, dA$$

# 15 SQL

“Structured Query Language”

Datenbanksprache:

- Definition von Datenstrukturen in relationalen Datenbanken
- Abfragen und Bearbeiten (Einfügen, Verändern, Löschen) der Daten
- Standardisiert, wird von vielen Datenbanksystemen unterstützt
- Nur grundlegende Operationen werden allgemein unterstützt
- Viele Erweiterungen von verschiedenen “Herstellern” unterschiedlich gehandhabt.

## 15.1 Datenbanksysteme:

Werden von vielen “Herstellern” zur Verfügung gestellt.

Arbeiten meist als Server-Client-Systeme:

Entweder lokal von einem Rechner oder über Netzwerk tw. platformunabhängig ansprechbar

Typische Vertreter:

Server Systeme: Open Source: MySQL, Postgres, ... Properitär: MicrosoftSQL, ...

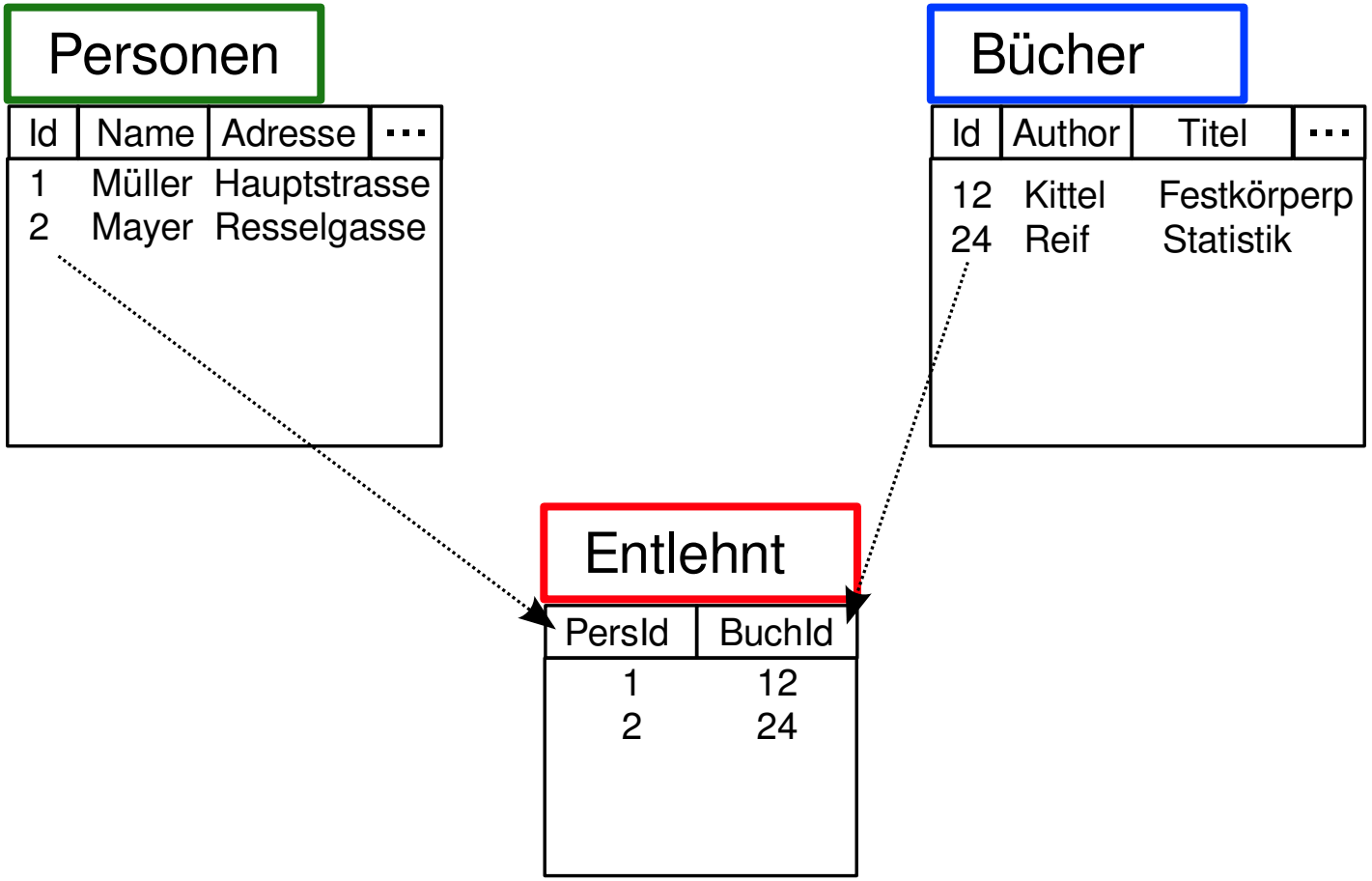
Einfaches Opensource System: sqlite

Vorallem für die Opensource-Systeme gibt es Schnittstellen (Bibliotheken) zu vielen Programmiersprachen (C/C++, perl, python, PHP, ...)

(Beispiel: xrd_create.sql, xrd_select.sql)

## 15.2 Relationale Datenbank:

Eine Tabelle nimmt (z.B. durch einen eindeutigen Index) Bezug auf eine andere Tabelle



## 15.3 Redundanz

Grundsatz des Datenbankdesigns:
In einer Datenbank sollen keine Redundanzen auftreten, jede Information (z. B. eine Adresse) wird genau einmal gespeichert

Z.B.: In der Tabelle “Entlehnt” werden die Adressen nicht erneut erfasst, sondern nur indirekt über PersonenId erfasst

Liste mit Adressen $\longrightarrow$ SELECT-Abfrage mit Verknüpfung von Personen-Tabelle mit Entlehnt-Tabelle

Komplexe Datenbanken mit mehreren Verknüpfungen:
Manchmal bessere Performance, wenn die Datenbank nicht vollständig normalisiert wird.
Redundanzen bleiben, um komplexe Abfragen zu verkürzen.

## 15.4 Schlüssel (primary key)

Jeder Datensatz erhält eine eindeutige Nummer oder ein anderes eindeutiges Feld, um ihn zu identifizieren $\longrightarrow$ **Schlüssel**

Sind Informationen auf viele Tabellen verteilt $\longrightarrow$ Schlüssel $\longrightarrow$ verknüpfen Informationen

Tabellen (Datensätze) die nur aus Schlüsseln bestehen $\longrightarrow$ Datensatz wird durch den Schlüssel referenziert

Primary Key: Der eigene Schlüssel des Datensatzes

Foreign Key: Schlüssel im Datensatz, der auf einen “primary key” anderer Tabellen verweist

Schlüssel können auch aus einer Kombination mehrerer Angaben bestehen

## 15.5 Referentielle Integrität

Einträge auf die von anderen Datensätzen Bezug genommen wird, müssen in der Datenbank auch vollständig vorhanden sein.

Obiges Beispiel:
In der Entlehnt-Tabelle dürfen nur PeronenIDs vorkommen, die es in der Personen-Tabelle auch gibt.

Überwachung dieser Aufgabe, z. B.:

- nur vorhandene PersonenIDs dürfen in die Entlehnt-Tabelle eingetragen werden können
- der Datensatz einer Person, die in die Entlehnt-Tabelle eingetragen ist, darf nicht gelöscht werden, oder er muss automatisch aus der Entlehnt-Tabelle entfernt werden
- Eine PersonenID, die bereits in der Entlehnt-Tabelle vorkommt, darf in der Personen-Tabelle nicht verändert werden oder sie muss dort ebenfalls aktualisiert werden

Fehlerhafte (widersprüchliche) Datensätze konnen durch bestimmte SELECT-Abfragen gefunden werden

# 16 Lineare Gleichungssysteme

Lineares Gleichungssystem: $\mathbf{A}x = b$

$x = (x_1,\ x_2,\ \ldots x_n)^T$ Lösungsvektor $b = (b_1,\ b_2,\ \ldots b_m)^T$:

$$\begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}$$

1. $m < n$ oder $m = n,\ \det(A) = 0$: weniger Gleichungen als unbekannte $\longrightarrow$ System unterbestimmt, unendlich viele oder keine Lösungen
2. $m = n\ \det(A) \neq 0 \longrightarrow$ eindeutige Lösung
3. $m > n$ mehr Gleichungen als Unbekannte $\longrightarrow$ System überbestimmt, Lösung mit kleinster Abweichung (Ausgleichsrechnung)
   $\mathbf{A} \cdot x = b \quad \Rightarrow \quad \mathbf{A}^T \cdot \mathbf{A} \cdot x = \mathbf{A}^T \cdot b$

## 16.1 Gaußsches Eliminationsverfahren

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\
&\cdots\cdots\cdots\cdots \\
a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n
\end{aligned}
\quad \longrightarrow \quad
\begin{aligned}
\tilde{a}_{11}x_1 + \tilde{a}_{12}x_2 + \cdots + \tilde{a}_{1n}x_n &= \tilde{b}_1 \\
\tilde{a}_{22}x_2 + \cdots + \tilde{a}_{2n}x_n &= \tilde{b}_2 \\
&\cdots\cdots\cdots\cdots \\
\tilde{a}_{nn}x_n &= \tilde{b}_n
\end{aligned}
$$

1. Vorwärtselimination durch elementare Zeilenumformungen:
   (a) Vielfaches einer Zeile zu einer anderen Zeile dazu addieren
   (b) Vertauschung von zwei Zeilen
2. Rückwärtseinsetzen (Rücksubstitution)

Elementare Zeilenumformungen $\longrightarrow$

Gleichungssystem wird transformiert $\longrightarrow$ besitzt dieselbe Lösungsmenge

**Algorithmus:**

- Start: $a_{21} \ldots a_{n1}$ der 1.Spalte sollen Null werden $\longrightarrow$
  Jeweils das geeignete Vielfache der ersten Gleichung addieren
  Multiplikator: das zu eliminierende $a_{i1} \quad i{=}2 \ldots n$ durch das Pivotelement $-a_{11}$ dividieren
- Iteration: Das Verfahren auf die nächsten $n-1$ Spalten und Zeilen der restlichen Matrix anwenden

Problem: Pivotelement $a_{jj} = 0 \longrightarrow$ Pivotisierung

- Wähle ein Matrixelement der jeweiligen Spalte, das ungleich 0 ist
- Vertausche die jeweilige Zeile mit der Pivotzeile

Kann kein Matrixelement ungleich 0 mehr gefunden werden $\longrightarrow$

Singuläre Matrix $\longrightarrow$ Gleichungssystem nicht eindeutig lösbar

## Numerische Berechnung:

Stabiler Algorithmus: Pivotelement $\longrightarrow$ Matrixelement mit dem größten Betrag

- Pivotelement aus der aktuellen Spalte: Spaltenpivotisierung
- Pivotelement aus der aktuellen Zeile: Zeilenpivotisierung

  $\longrightarrow$ entsprechender Tausch der Spalten notwendig

  Rückwärtseinsetzen: Variablen haben die Position gändert
- Pivotelement aus der gesamten Restmatrix: Totalpivotisierung

  $\longrightarrow$ Zeilen- und Spaltenvertauschungen notwendig

Pivotisierung benötigt Rechenzeit daher meist Spaltenpivotisierung

## 16.2 LR-Zerlegung (auch LU-Zerlegung oder Dreieckszerlegung)

Gaußsches Eliminationsverfahren $\longrightarrow$ LR-Zerlegung

Zerlegung der regulären Matrix $\mathbf{A}$ in das Produkt einer linken ("left") unteren Dreiecksmatrix $\mathbf{L}$ und einer rechten oberen Dreiecksmatrix $\mathbf{R}$ (rechts, auch $\mathbf{U}$ "upper")

$$\mathbf{A}x = b \quad \rightarrow \quad \mathbf{LR}x = b$$

Obere Dreiecksmatrix:

- Einträge unterhalb der Hauptdiagonale = 0
- Hauptdiagonale: keine Beschränkungen
- $i > j \Rightarrow a_{ij} = 0$

Untere Dreiecksmatrix:

- Einträge oberhalb der Hauptdiagonale = 0

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & \cdots & & 0 \\ l_{21} & 1 & 0 & \cdots & 0 \\ l_{31} & l_{32} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ l_{N1} & l_{N2} & \cdots & l_{NN-1} & 1 \end{pmatrix} \mathbf{R} = \begin{pmatrix} r_{11} & r_{12} & \cdots & & r_{1N} \\ 0 & r_{22} & \ddots & \cdots & r_{2N} \\ 0 & 0 & \ddots & & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & r_{NN-1} \end{pmatrix}$$

$\mathbf{LR}x = b$ kann durch vorwärts/rückwärtseinsetzen gelöst werden

$$y_1 = b_1;\ y_i = (b_i - \sum_{k=1}^{i-1} l_{ik} \cdot y_k)$$

$$x_n = \frac{y_N}{r_{NN}};\ x_i = \frac{1}{u_{ii}}(y_i - \sum_{k=i+1}^{N} u_{ik} \cdot x_k)$$

## 16.3 Tridiagonale Matrix

$$\mathbf{A}x = \begin{pmatrix} b_1 & c_1 & & & 0 \\ a_2 & b_2 & c_1 & & 0 \\ & \ddots & \ddots & \ddots & \\ & & a_{N-1} & b_{N-1} & c_{N-1} \\ 0 & & & a_N & b_N \end{pmatrix} x = \mathbf{L}(\mathbf{U}x) = \mathbf{L}y = f$$

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & & & 0 \\ l_2 & 1 & 0 & & 0 \\ & l_3 & 1 & \ddots & \\ & & \ddots & \ddots & 0 \\ 0 & & & l_N & 1 \end{pmatrix} \quad \mathbf{R} = \begin{pmatrix} d_1 & c_1 & 0 & & 0 \\ 0 & d_2 & c_2 & & \\ & \ddots & \ddots & & \\ & & 0 & d_{N-1} & c_{N-1} \\ 0 & 0 & \cdots & 0 & d_N \end{pmatrix}$$

$$d_1 = b_1;\ l_j = a_j/b_{j-1};\ d_j = b_j - l_j \cdot c_{j-1}$$

$$y_1 = f_1;\ y_i = f_i - l_i \cdot y_{i-1};\ x_N = y_N/d_N;\ x_i = (y_i - c_i \cdot x_{i+1})/d_i$$

# 17 Interpolation

Daten als diskrete Funktionswerte, Zwischenwerte möglichst “gut” berechnen (approximieren)

- Wenige Mess-/Datenpunkte
- Hoher Aufwand zur Berechnung eines Punktes $\longrightarrow$ Zwischenpunkte $\longrightarrow$ Interpolation
- Numerische Methode verlangt ein bestimmtes Gitter, Punkte liegen dazwischen

Datenwerte: $y_i = f(x_i)$ $\longrightarrow$ “günstiges” $f(x)$ finden

Gegeben: $f(x)$ an $N+1$ unterschiedlichen $x_i,\ i = 0 \ldots N \mapsto f_i = f(x_i)$

Aufgabe: Für eine Funktionenklasse $\phi(x; a_0, a_1 \ldots a_N)$ die Parameter $a_i$ so zu bestimmen, dass $\phi(x_i; a_0, a_1 \ldots a_N) = f_i \ \forall\ i = 0 \ldots N$, mit den Stützstellen $(x_i, f_i)$

Meist wird nicht auf ein passendes Modell Rücksicht genommen, es geht oft nur um ein “glatte” Kurve.

## 17.1 Lineare Interpolation

2 Datenpunkte $(x_i, f_i),\ (x_{i+1}, f_{i+1}) \longrightarrow$ Gerade: $\phi = a_1 x + a_0 \longrightarrow$ Zwischenpunkte

Spezialfall von:

## 17.2 Polynominterpolation

$\phi$: Polynom (reell oder komplex) mit dem Grad $p \leq N$

$$\phi(x_i; a_0, a_1 \ldots a_N) = p(x) = a_0 + a_1 x + a_2 x^2 + \cdots a_N x^N$$

Für $N + 1$ beliebig vorgegebene Punkte:

**Es existiert ein eindeutiges Polynom p(x) vom Grad N, sodass** $p(x_i) = f_i$

verschiedene Berechnungsmethoden $\longrightarrow$ verschiedene Darstellungen des selben Polynoms

### 17.2.1 Newtonscher Algorithmus

Newton-Basisfunktionen: $N_0(x) = 1,\ N_i(x) = \prod_{j=0}^{i-1}(x - x_j)$

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

$$a_0 + a_1\,(x - x_0) + a_2\,(x - x_0)\,(x - x_1) + \cdots + a_n\,(x - x_0)\cdots(x - x_{n-1})$$

$\longrightarrow$ Gleichungssystem $p(x_i) = f_i$

$$\begin{pmatrix} 1 & & & 0 \\ 1 & (x_1 - x_0) & & \\ 1 & (x_2 - x_0) & (x_2 - x_0)(x_2 - x_1) & \\ \vdots & \vdots & & \ddots \\ 1 & (x_n - x_0) & \cdots & \prod_{i=0}^{n-1}(x_n - x_i) \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} f_0 \\ \vdots \\ f_n \end{pmatrix}$$

$\longrightarrow$Untere Dreiecksmatrix $\longrightarrow$ leicht zu berechnen

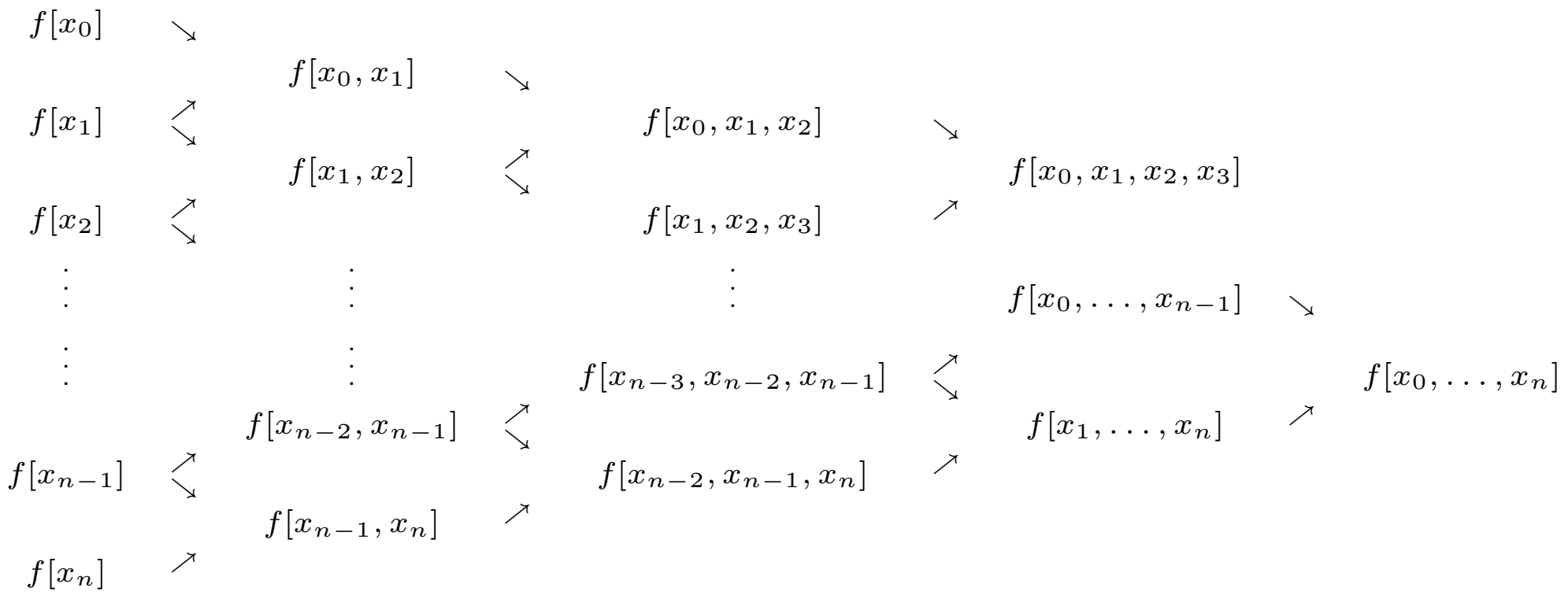
## Schema der dividierten Differenzen

$a_i$ nicht aus Gleichungssystem bestimmt $\longrightarrow$ dividierte Differenzen (effizienter)

$a_i = f[x_0, \ldots, x_i]$ dividierte Differenz: $f[x_i, \ldots, x_j]$ für $i < j$

Rekursiv definiert durch: $f[x_i] = f_i, \quad f[x_i, x_j] = \frac{f[x_i] - f[x_j]}{x_j - x_i},$

$$f[x_i, \ldots, x_j] = \frac{f[x_{i+1}, \ldots, x_j] - f[x_i, \ldots, x_{j-1}]}{x_j - x_i}$$



Ergänzung der Wertepaare $(x_i, f_i)$ um einen weiteren Punkt $\longrightarrow$ eine Zeile hinzufügen

Vorherige Koeffizienten $a_i$ müssen nicht neu berechnet werden

Auswertung des Polynoms $\longrightarrow$Horner Schema $\longrightarrow$ effizienter Algorithmus

$$p(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_n x^n = (\ldots(b_n x + b_{n-1})x + \cdots)x + b_0$$

z.B.: $p(x) = 5x^4 + 3x^3 - 7x^2 + 4x + 2 = x(x(x(5(x) + 3) - 7) + 4) + 2$

```
// Hornerschema
for(i=N-1,y=b[N]; i>=0; i--)y=x*y+b[i];
// N Multiplikationen und Additionen
```

```
// Straight forward
for(i=1,y=b[N];i<=N;i++)y+=pow(b[i],i);
// N-mal berechnung der Potenz (Rechenzit)
```

siehe z.B. `http:`

`//de.wikipedia.org/wiki/Polynominterpolation#Newtonscher_Algorithmus`

(Beispiel: horner.py, polynom.py)

Beispiel: $(x_i, f_i)$ : (3.2, 22.0), (2.7, 17.8), (1.0, 14.2), (4.8, 38.3), (5.6, 51.7)

Die dividierten Differenzen sind hier:

| $x_i$ | $f(x_i)$ | | | | |
|---|---|---|---|---|---|
| $3.2$ | $22.0$ | | | | |
| $2.7$ | $17.8$ | $\frac{17.8-22.0}{2.7-3.2} = 8.400$ | | | |
| $1.0$ | $14.2$ | $\frac{14.2-17.8}{1.0-2.7} = 2.118$ | $\frac{2.118-8.400}{1.0-3.2} = 2.856$ | | |
| $4.8$ | $38.3$ | $\frac{38.3-14.2}{4.8-1.0} = 6.342$ | $\frac{6.342-2.118}{4.8-2.7} = 2.012$ | $\frac{2.012-2.856}{4.8-3.2} = -0.5280$ | |
| $5.6$ | $51.7$ | $\frac{51.7-38.3}{5.6-4.8} = 16.750$ | $\frac{16.750-6.342}{5.6-1.0} = 2.263$ | $\frac{2.263-2.012}{5.6-2.7} = 0.0865$ | $\frac{0.0865-(-0.5280)}{5.6-3.2} = 0.256$ |

Interpolationspolynom:

$$
\begin{aligned}
P_{\text{Newton}}(x) =& 22.0 \\
&+ 8.400(x - 3.2) \\
&+ 2.856(x - 3.2)(x - 2.7) \\
&- 0.528(x - 3.2)(x - 2.7)(x - 1.0) \\
&+ 0.256(x - 3.2)(x - 2.7)(x - 1.0)(x - 4.8)
\end{aligned}
$$

## 17.3 Splineinterpolation

Nachteil der Polynominterpolation $\longrightarrow$Oszillationen bei Polynomen höherer Ordnung

Spline: engl. Biegsames Kurvenlineal; glatte Kurve durch mehrere Punkte (Handwerk)

Es wird lokal über jeweils zwei Stützstellen $(x_i, f_i)$ und $(x_{i+1}, f_{i+1})$ $i = 0, .., N-1$ interpoliert.

Linearer Spline: Polynom 1.Ordnung $\longrightarrow$Gerade

Kubische Spline: Polynom 3.Ordnung

Für die Splinefunktion $s(x_i)$ gilt:

1. $s(x_i) = f_i = y_i,\ i = 0, 1, \ldots, N$
2. $s$ ist in $[x_0, x_N]$ zweimal stetig differenzierbar
3. die Gesamtkrümmung von $s$ ist minimal

Für jedes Teilintervall $\longrightarrow$ Polynom $s_j(x)$ in Newtondarstellung:

$$s_j(x) = a_j + b_j \cdot (x - x_j) + c_j \cdot (x - x_j)^2 + d_j \cdot (x - x_j)^3,\ x_{j-1} \leq x \leq x_j,\ j = 1, \ldots, N$$

$\longrightarrow$ Gleichungssystem $\longrightarrow$Lösung: $4N$ Bedingungen

Für $N$ Intervalle sind zwei Interpolationsbedingungen zu erfüllen:

$$s_j(x_{j-1}) = y_{j-1} \qquad j = 1, \ldots, n \tag{4}$$

$$s_j(x_j) = y_j \qquad j = 1, \ldots, N \tag{5}$$

$\longrightarrow 2N$ Bedingungen, $s$ muss an allen $N - 1$ inneren Stützstellen zweimal stetig differenzierbar sein:

$$s_j'(x_j) = s_{j+1}'(x_j) \qquad j = 0, \ldots, n - 1 \tag{6}$$

$$s_j''(x_j) = s_{j+1}''(x_j) \qquad j = 1, \ldots, N - 1 \tag{7}$$

$\longrightarrow 2N - 2$ Bedingungen

2 Randbedingungen:

- freier Rand oder natürlicher Spline: $s_0''(x_0) = 0,\ s_N''(x_N) = 0$
- eingespannter Rand: $s_1'(x_0) = y_0',\ s_n'(x_n) = y_n'$ $y_0'$ und $y_n'$ vorgegeben
  (durch die Ableitung der zu interpolierenden Funktion f oder durch eine Approximation)

1. Ableitung (Steigung): $s_j'(x) = b_j + 2 \cdot c_j \cdot (x - x_j) + 3 \cdot d_j \cdot (x - x_j)^2$
2. Ableitung (Krümmung): $s_j''(x) = 2 \cdot c_j + 6 \cdot d_j \cdot (x - x_j)$

$\longrightarrow$ tridiagonales Gleichungssystem

Praktische Lösung meist durch Bibliotheksfunktionen

siehe z.B.

`http://de.wikipedia.org/wiki/Spline-Interpolation#Der_kubische_C2-Spline`

(Beispiel: Spline.py)

## 17.4 B-Splines

Widersprechen den Voraussetzungen der Interpolation:

$y_i \neq f(x_i)$: Kurve geht nicht durch die Stützpunkte

Basieren ebenfalls auf Polynomen (k-1)-ten Grades

Änderung einzelner Punkte wirken sich nur lokal (nicht auf alle Intervalle) aus

Darstellung von Computergrafiken: Bezierkurven (Glättung in Zeichenprogrammen)

## 17.5 Praktische Vorgehensweise

Messdaten: Normalerweise mit Messfehlern und Rauschen behaftet

Problem: Punkte streuen $\longrightarrow$ überschwingende Polynome

**Lösung:**

- Eine Funktion anpassen (fitten), die dem zugrundeliegenden Modell entspricht
- Ist das nicht möglich:

  Orthogonale Funktionen verwenden (z.B.: Tschebyscheff)

  Vor dem Fit meist Normierung der Messdaten notwendig
- Wertetabelle $(x_i, f_i)$ erstellen und gesuchte Zwischenwerte interpolieren