

Python Project

Richardson Iteration from Scratch

applied to Heat Equation and the PageRank

– in german –

Dr. Christian Vollmann

2. März 2023

Übersicht

Leadsheet zum praktischen zweiten Teil der Veranstaltung “Programmierung für Mathematiker”. Das Kernziel ist der Aufbau einer integrierten Arbeitsumgebung, die ein verzahntes Arbeiten mit den folgenden Komponenten ermöglicht:

- Code: Entwicklung mit Python in PyCharm und Dokumentation mit Sphinx,
- Text: Mathematische Textverarbeitung mit L^AT_EX in TeXstudio,
- IT/Backup: Versionskontrolle mit git auf dem GitLab Server der Universität Trier und Online-Datensicherung mit Seafire auf Servern in RLP.

Vorbereitung auf:

- Numerik-Veranstaltungen (Code)
- Seminararbeiten (Vortrag, Text, Code, IT/Backup)
- Praktikumsbericht (Text, IT/Backup)
- Bachelor- und Master-Arbeiten (Text, Code, IT/Backup)
- Doktorarbeit und Forschung (Vortrag, Text, Code, IT/Backup,...)

Projekt:

- Um das Erlernte an einem Fallbeispiel praktisch anzuwenden, werden wir uns ein bestimmtes mathematisches Problem vornehmen und dieses mit geeigneten numerischen Methoden lösen.
- Genauer werden wir einen iterativen Löser für lineare Gleichungssysteme implementieren. Das wird bereits ein Vorgeschnack auf die spannenden Numerik-Veranstaltungen im weiteren Studiumsverlauf.

Inhaltsverzeichnis

1 Die mathematische Problemstellung	1
1.1 Eigenwerte und der Spektralradius	2
1.2 Lineare Fixpunktiteration	2
1.3 Das relaxierte Richardson-Verfahren	3
1.4 Speicherung dünnbesetzter Matrizen	5
1.5 Anwendung: PageRank	8
1.6 Anwendung: Optimierung	10
2 Projektaufbau planen	11
2.1 Code	11
2.2 Text (skip)	12
3 Secure Shell (SSH)	14
3.1 Verwendung	14
3.2 Quellen	15
4 Versionskontrolle: git und GitHub	16
4.1 git	16
4.2 GitHub	18
4.3 Quellen	19
5 Entwicklungsumgebung	20
5.1 Nicht integriert: Texteditor + Terminal + Interpreter +	20
5.2 Integriert: Integrated Development Environment (IDE)	21
5.3 Browserbasiert	21
5.4 Die IDE PyCharm von jetbrains	21
6 Code Dokumentation mit sphinx	23
6.1 Getting Started	23
6.2 Maßschneiderung in conf.py	24
6.3 Allgemeines Erscheinungsbild: html_theme	24
6.4 Struktur ausbauen	24
6.5 Synchronisation von Code und Dokumentation: autodoc	24
6.6 docstring Style: reStructuredText, NumPy, Google	25
6.7 Reference Guide: Automatisch alle Code-Bestandteile auflisten mit ... autosummary::	25
6.8 Links zwischen Quellcode und docs: [source], [docs]	25
6.9 Cross-References: Links innerhalb der Dokumentation	26
6.10 Mathematische Formeln .. math::	26
6.11 Andere Build-Formate	26
6.12 Veröffentlichen mit Github-Pages	26
7 Mathematische Textverarbeitung mit L^AT_EX und TeXstudio (Skip)	27
7.1 Einführung	27
7.2 Mögliche Arbeitsabläufe	27
7.3 Die .tex-Eingabedatei	28
7.4 Setzen von Text	31
7.5 Setzen von mathematischen Formeln	31
7.6 Setzen von Bildern	31
7.7 Seitenaufbau	31
7.8 Literatur	31
7.9 Modularisierung und Maßschneiderung	32
7.10 Ausgewählte Beispiele und Tipps	34
7.11 Sonstige Dokumentklassen	35
A Grundlagen der Informatik	37
A.1 Hardwarekomponenten	37
A.2 Betriebssysteme	46
A.3 Netzwerke	51
A.4 Virtualisierung	58

B Daten Management	61
B.1 Der Begriff Datensicherung (<i>backup</i>)	61
B.2 Outsourcing: Online-Datensicherung	62
B.3 Konzept 1: Externes Dateisystem einhängen	62
B.4 Konzept 2: Externes Dateisystem lokal spiegeln durch Dateisynchronisation	64

Abbildungsverzeichnis

1 Dünnbesetztheitsstruktur	6
2 Beispiel CSR Matrix	7
3 Mindmap zum Projektaufbau	11
4 Beispiel: Verzeichnisstruktur für code	12
5 Beispiel: Verzeichnisstruktur für text	13
6 https://xkcd.com/1597/	16
7 Schnittstellen	37
8 Hauptplatine	38
9 Intel Core i7	41
10 Treiber	45
11 Quelle: https://de.wikipedia.org/wiki/Betriebssystem	46
12 ping an Google	56

Tabellenverzeichnis

1 Für uns interessante Dokumentklassen	29
2 Für uns interessante Klassenoptionen	30
3 Eine kleine Auswahl häufig verwendeteter Pakete	30

1 Die mathematische Problemstellung

Wir möchten das folgende mathematische Problem numerisch lösen:

Input: Eine invertierbare Matrix $A \in \mathbb{R}^{n \times n}$ und ein Vektor $b \in \mathbb{R}^n$.
Output: Der Vektor $x \in \mathbb{R}^n$, sodass

$$Ax = b.$$

Bemerkungen:

- Da A invertierbar ist, existiert für jede rechte Seite b genau eine eindeutige Lösung $x = A^{-1}b$.
- Die Berechnung der inversen Matrix A^{-1} ist sehr teuer (z.B. n mal Lösen mit Einheitsvektoren) und wird in der Praxis stets vermieden, da man meist nur an $b \mapsto A^{-1}b$ interessiert ist.

Zur Lösung von linearen Gleichungssystemen gibt es im Wesentlichen zwei große Verfahrensklassen:

- **Direkte Methoden:** Faktorisieren und Lösen

- Gauss Elimination und LR -Zerlegung, Cholesky Zerlegung
 - Gram-Schmidt/Householder/Givens und QR -Zerlegung
 - ...
- Gut geeignet für kleine Probleme ($n \leq 10^5$)
→ Bei großen Problemen: Zu langsam, Rundungsfehler, ...
→ direkt = endlich viele Schritte

- **Iterative Verfahren:**

- Krylov-Unterraum Verfahren
 - Splitting-Verfahren (Fixpunktiteration)
 - * Richardson
 - * Jacobi
 - * Gauß-Seidel, SOR
- Brauchen i.d.R. nicht die Matrix selbst, sondern nur die Funktion $x \mapsto Ax$ (Matrix-Vektor Produkt)
→ Gut geeignet für große (dünn-besetzte) Probleme ($n \geq 10^5$), wie sie häufig in Anwendungen vorkommen
→ direkt \neq iterative = indirekt = ggf. unendlich viele Schritte

Der Ansatz für iterative Verfahren:

Wir möchten eine Folge $(x^k)_k$ von Vektoren $x^k \in \mathbb{R}^n$ konstruieren, sodass die folgenden Eigenschaften erfüllt sind.

- Die Folge $(x^k)_k$ konvergiert.
- Der Grenzwert x^* löst die Gleichung: $Ax^* = b$.

Ausgehend von einem (i.d.R. beliebigen) Startvektor $x^0 \in \mathbb{R}^n$ berechnen wir Schritte $p^k \in \mathbb{R}^n$ mit der Absicht, dass wir uns durch die Iterationsvorschrift

$$x^{k+1} = x^k + p^k$$

der Lösung schrittweise nähern.

Wünschenswert:

- Effizienz: Die Schritte p^k sollten leicht zu berechnen sein, sodass jeder Iterationsschritt effizient berechnet werden kann. (Zum Beispiel nur abhängig von einer kleinen Historie x_{k-m}, \dots, x_k)
- Konvergenz: Wir kommen der Lösung x^* mit jedem Schritt (bestenfalls) echt näher.

Die große Frage: Wie erhalten wir in jedem Iterationsschritt geeignete Schritte p^k ?

Dazu nun zwei kleine Theorie-Abschnitte.

1.1 Eigenwerte und der Spektralradius

Es sei $A = [a_1, \dots, a_n] \in \mathbb{R}^{n \times n}$ eine Matrix mit n Spalten $a_i \in \mathbb{R}^n$ und $x \in \mathbb{R}^n$ ein n -dimensionaler Vektor.
Betrachten wir das Matrix-Vektor Produkt

$$b := A \cdot x = \sum_{i=1}^n x_i a_i,$$

so erhalten wir einen Vektor $b \in \mathbb{R}^n$. Daher können wir eine Matrix stets als (lineare) Abbildung

$$\mathbb{R}^n \rightarrow \mathbb{R}^n, x \mapsto A \cdot x$$

verstehen.

Beispiel

$$A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

[Tafel: Grafik mit ein paar Beispielen $A \cdot x$]

Interessante Beobachtung:

- Es gibt Vektoren, deren Richtung von der Matrix nicht verändert werden. Diese werden allenfalls um einen bestimmten Faktor skaliert (vergrößert, verkleinert oder identisch abgebildet).
- Solche Vektoren nennen wir Eigenvektoren der Matrix und die zugehörigen Skalierungsfaktoren Eigenwerte. Sie werden später lernen, dass Eigenwerte wesentliche Eigenschaften der Matrix kodieren und damit von zentraler Bedeutung sind.

Definition. Es sei $A \in \mathbb{C}^{n \times n}$ eine Matrix. Dann nennen wir eine Zahl $\lambda \in \mathbb{C}$ Eigenwert von A , falls ein (Eigen-)Vektor $v \in \mathbb{C}^n$ mit $v \neq 0$ existiert, sodass

$$Av = \lambda v.$$

Bemerkung

- $Av = \lambda v \Leftrightarrow (A - \lambda I)v = 0 \Leftrightarrow v \in \ker(A - \lambda I) = \{v \in \mathbb{R}^n : (A - \lambda I)v = 0\}$
- Man kann zeigen, dass eine $n \times n$ Matrix maximal n verschiedene Eigenwerte besitzt (zum Beispiel mit dem Fundamentalsatz der Algebra).
- Den beträchtlich größten Eigenwert nennt man **Spektralradius** der Matrix A und wird mit $\rho(A)$ bezeichnet. Genauer

$$\rho(A) := \max\{|\lambda| : \lambda \text{ Eigenwert von } A\}.$$

- Siehe auch: [5, Def. 2.30]

Beispiel

Diagonalmatrix

$$\begin{pmatrix} 1 & 0 \\ 0 & -2 \end{pmatrix}$$

Dann $\lambda = 1, -2$ (Eigenvektoren hier Vielfachen der Einheitsvektoren) und $\rho(D) = 2$.

1.2 Lineare Fixpunktiteration

Grob gesprochen besagt der Banach'sche Fixpunktsatz (siehe [5, Kap. 2.4]) folgendes: Wenn eine Funktion f den Abstand (wir brauchen metrische Räume) zwischen zwei beliebigen Punkten verkleinert (**Kontraktion**), so besitzt f genau einen **Fixpunkt** $x^* = f(x^*)$.

Nützlich für die Numerik ist der konstruktive Beweis: Die Folge

$$x^0 \in \mathbb{R}^n \text{ bel., } x^{k+1} := f(x^k), \quad (\text{Fixpunktiteration})$$

konvergiert gegen diesen Fixpunkt. Diese Aussage ist in der Numerik von enormer Bedeutung und taucht in vielen Facetten immer wieder auf.

Bevor wir die Fixpunktiteration für unser Problem zur Anwendung bringen, konkretisieren wir diesen Sachverhalt zunächst durch folgenden Satz [5, Satz 4.5]:

Satz (Lineare Fixpunktiteration). Es seien $M \in \mathbb{R}^{n \times n}$, $N \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^n$ und $x^0 \in \mathbb{R}^n$ ein beliebiger Startvektor. Zudem definieren wir die Folge

$$x^{k+1} := f(x^k) := Mx^k + Nb.$$

Dann gilt

$$\rho(M) < 1 \quad (\text{f Kontraktion}) \quad \Leftrightarrow \quad \lim_{k \rightarrow \infty} x^k = x^* = f(x^*).$$

1.3 Das relaxierte Richardson-Verfahren

Um ein iteratives Verfahren zur Lösung von $Ax = b$ zu entwickeln, versuchen wir nun die Fixpunktiteration anzuwenden. Dazu brauchen wir eine geeignete Funktion $f: \mathbb{R}^n \rightarrow \mathbb{R}^n$ der Form

$$f(x) = Mx + Nb$$

mit

- Kontraktiv: $\rho(M) < 1$, sodass Fixpunkt $x^* = f(x^*)$ eindeutig existiert,
- Fixpunkt löst das Problem: $Ax^* = b$.

Um solch eine Funktion zu finden, re-formulieren wir das Problem $Ax = b$ kurzum als ein Fixpunkt-Problem:

$$\begin{aligned} Ax = b &\Leftrightarrow (x - x) + Ax = b \\ &\Leftrightarrow x = I \cdot x - A \cdot x + b \\ &\Leftrightarrow x = (I - A)x + b =: f(x). \end{aligned}$$

Mit obigem Satz (wähle $M = I - A$, $N = I$) gilt nun: Falls für den Spektralradius $\rho(I - A) < 1$ gilt, dann konvergiert die Folge

$$x^{k+1} := f(x^k) = (I - A)x^k + b \tag{1}$$

für jeden beliebigen Startvektor $x^0 \in \mathbb{R}^n$ gegen den eindeutigen Fixpunkt

$$x^* = f(x^*) \quad \Leftrightarrow \quad Ax^* = b.$$

Die Iterationsvorschrift in (1) beschreibt das sogenannte **Richardson-Verfahren**¹ (siehe [5, Kap. 4.1.4]) und lässt sich umformulieren zu

$$x^{k+1} = x^k - (Ax^k - b) = x^k + p^k, \quad \text{mit } p^k = -(Ax^k - b).$$

Entscheidend für die Konvergenz des Verfahrens ist die Bedingung

$$\rho(I - A) < 1,$$

welche im Allgemeinen nicht erfüllt sein muss; auch nicht für invertierbare Matrizen A . Das bedeutet: Das Gleichungssystem $Ax = b$ kann eine eindeutige Lösung haben, aber das Richardson Verfahren muss nicht konvergieren.

Dazu ein **Beispiel**

$$A = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 2 \\ 2 \end{pmatrix},$$

sodass

$$x^* = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

Allerdings

$$I - A = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix} = -I$$

mit Eigenwerten $\lambda_1 = -1, \lambda_2 = -1$, sodass

$$\rho(I - A) = 1 \quad (!).$$

In diesem Beispiel ergibt sich die Iteration

$$x^{k+1} = (I - A)x^k + b = -x^k + b.$$

¹Benannt nach dem britischen Meteorologen Lewis Fry Richardson (1881–1953), der dieses Verfahren 1910 entwickelte. (Wikipedia)

Betrachten wir beispielsweise $x^0 = 0$, dann ergeben sich für die ersten Iterationen

$$\begin{aligned}x^1 &= -x^0 + b = b \\x^2 &= -x^1 + b = -b + b = 0 \\x^3 &= -x^2 + b = 0 + b = b \\&\dots\end{aligned}$$

Offensichtlich konvergiert diese (alternierende) Folge nicht gegen die Lösung $x = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$.

[Tafel: Grafik dazu mit 2d-Iterierten]

1.3.1 Relaxierte Variante

[Tafel: Am Bild der alternierenden Folge motivieren]

Oftmals kann es ausreichen den Schritt hinreichend klein zu wählen, um ein konvergentes (aber ggf. langsames) Verfahren zu erhalten:

$$x^{k+1} = x^k + \theta p^k = x^k - \theta(Ax^k - b), \quad \theta > 0 \text{ klein.} \quad (2)$$

Dieses Verfahren nennt man **relaxiertes Richardson Verfahren**. Da

$$x^{k+1} = x^k - \theta(Ax^k - b) = (I - \theta A)x^k + \theta b,$$

konvergiert das rel. Richardson Verfahren genau dann, wenn

$$\rho(I - \theta A) < 1.$$

Das relaxierte Verfahren ist gerade das ursprüngliche Richardson-Verfahren angewendet auf das äquivalente Gleichungssystem

$$\theta Ax = \theta b \quad (\theta > 0) \quad \Leftrightarrow \quad Ax = b,$$

sodass wir auch sicher sein können, dass wir das originale Problems $Ax = b$ lösen.

Am **Beispiel** von oben:

Wählen wir beispielsweise $\theta = \frac{1}{2}$, so erhalten wir die Folge

$$x^{k+1} = (I - \theta A)x^k + \theta b = \begin{pmatrix} 1 - \theta 2 & 0 \\ 0 & 1 - \theta 2 \end{pmatrix} x^k + \theta \begin{pmatrix} 2 \\ 2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

Damit sind wir für einen beliebigen Startvektor $x^0 \in \mathbb{R}^n$ sogar schon nach einem Schritt fertig, denn

$$x^1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} = x^*.$$

Beachte: Multiplikation mit $\frac{1}{2}$ entspricht der Multiplikation mit der Diagonalmatrix $\text{diag}(\frac{1}{2})$, welche hier gerade die Inverse von A ist.

Bemerkung

In jedem Iterationsschritt ist im Wesentlichen nur ein Matrix-Vektor Produkt $A \cdot x$ zu berechnen.

- Wir brauchen also nicht einmal die Matrix selbst, um das Problem $Ax = b$ mit dem Richardson-Verfahren (Konvergenz vorausgesetzt) zu lösen, sondern lediglich die Funktion

$$x \mapsto Ax,$$

welche auch als eine Art Blackbox zur Verfügung stehen könnte.

- Für Matrizen mit vielen Nulleinträgen (=dünnbesetzt), kann das Produkt $A \cdot x$ selbst für große Matrizen ($n \geq 10^5$) effizient (schnell und ohne viel Speicherbedarf) implementiert werden. Dazu brauchen wir geeignete Formate mithilfe derer wir eine Matrix ohne ihre Nulleinträge speichern und dennoch die richtigen Operationen (wie $A \cdot x$) ausführen können. Wir implementieren mindestens eines dieser Formate.

1.3.2 Zur numerischen Umsetzung: Abbruchkriterien

Die Iterationsvorschrift (2) des rel. Richardson Verfahrens setzen wir in Form einer Schleife um. Dazu müssen wir ein Abbruchkriterium festlegen.

Fehlertoleranz

Idealerweise würden wir das Verfahren abbrechen, sobald die aktuelle Iterierte x^k "hinreichend nah" an der Lösung $x^* = A^{-1}b$ ist, also

$$\|e^k\|_2 < \text{tol}, \quad e^k := x^k - x^* = x^k - A^{-1}b,$$

für eine "kleine" Fehlertoleranz $\text{tol} > 0$ (ein gängiger Wert für iterative Verfahren ist $\text{tol}=1e-05$ oder $\text{tol}=1e-06$, siehe zum Beispiel `scipy.sparse`). Allerdings kennen wir die exakte Lösung x^* nicht. Stattdessen nehmen wir die Größe des **Residuums** r^k als Kriterium her:

$$\|r^k\|_2 < \text{tol}, \quad r^k := Ae^k = A(x^k - A^{-1}b) = Ax^k - b.$$

Im Falle von $x^k = x^*$ gilt $e^k = r^k = 0$. Außerdem lässt sich dies durch die Submultiplikativität der Spektralnorm von A^{-1} (Operatornorm bzgl. der Euklidischen Norm) rechtfertigen:

$$\|e^k\|_2 = \|A^{-1}r^k\|_2 \leq \|A^{-1}\|_2 \cdot \|r^k\|_2,$$

wobei die Spektralnorm (ausgedrückt mit extremalen Singulärwerten)

$$\|A^{-1}\|_2 := \max_{\|x\|_2=1} \|A^{-1}x\|_2 = \sigma_{\max}(A^{-1}) = \frac{1}{\sigma_{\min}(A)}$$

für eine feste Problemstellung als konstant angesehen werden kann. Mit anderen Worten: Falls $\|A^{-1}\|_2$ nicht extrem groß ist (das wäre der Fall wenn A "fast" singulär ist, also schlecht konditioniert ist), dann folgt aus $\|r^k\|_2$ klein, dass auch $\|e^k\|_2$ klein ist (siehe auch [5, Kap. 2.3]).

Maximale Iterationszahl

In der Praxis reicht das Residuums-Kriterium allerdings nicht aus, da nicht immer klar ist, ob das Verfahren tatsächlich konvergiert. Um eine Endlosschleife zu vermeiden, wird häufig ein optionaler Parameter `maxiter` übergeben, der die Anzahl von Schleifendurchläufen begrenzt.

1.3.3 Allgemeine Splitting–Verfahren

Allgemeiner könnte man das originale Gleichungssystem auch mit einer beliebigen invertierbaren Matrix $N \in \mathbb{R}^{n \times n}$ umformulieren zu

$$Ax = b \Leftrightarrow NAx = Nb.$$

Das resultierende Richardson–Verfahren angewendet auf das neue System würde dann wie folgt aussehen:

$$x^{k+1} = x^k - N(Ax^k - b)$$

mit Konvergenz, genau dann, wenn $\rho(I - NA) < 1$. Mit einer geeigneten Wahl von N , d.h. $N \approx A^{-1}$ (auch **Präkonditionierer** genannt), kann man ein konvergentes Verfahren erhalten. Unterschiedliche Wahl von N , liefert unterschiedliche Verfahren; sog. Splitting–Verfahren, welche allesamt als präkonditionierte Variante des Richardson–Verfahrens aufgefasst werden können [5, Kap. 4.1].

Bemerkung: In der Praxis werden Splitting–Verfahren meist nicht direkt als Löser verwendet, sondern als Präkonditionierer (genauer die Matrix N) von Krylov–Unterraum Verfahren [5, Kap. 5.3].

1.4 Speicherung dünnbesetzter Matrizen

Definition. Eine **dünnbesetzte** oder schwachbesetzte Matrix (engl. *sparse matrix*) ist eine Matrix mit "vielen" Nullen in den Einträgen². Das Gegenstück zu einer dünnbesetzten Matrix wird **vollbesetzte/dichte** Matrix (engl. *dense matrix*) genannt.

Bemerkung

- "viel" ist zum Beispiel wie folgt zu verstehen: Bei einer dünnbesetzten quadratischen Matrix sollte die Anzahl der Nichtnulleinträge nur linear ($\mathcal{O}(n)$) von der Dimension n abhängen oder gemäß $\mathcal{O}(n \log n)$ (anstatt quadratisch: n^2 Einträge).

²Laut Wikipedia: Der Begriff wurde erstmals 1971 von James Hardy Wilkinson verwendet.

- Strukturierte Beispiele: Band-Matrizen (Diagonal, Tridiagonal,...)
- Unstrukturierte Beispiele: Die Diskretisierung von (partiellen) Differentialgleichungen führt typischerweise zu (hochdimensionalen) dünnbesetzten Systemen (siehe Abb. 1)
- Vorteil: Die Dünnbesetzungssstruktur können wir für die effiziente Implementierung von **Matrix-Operationen** sowie **Speicherung** der Matrizen ausnutzen.
- Iterative Verfahren (wie das Richardson Verfahren) brauchen häufig nur die Auswertung der Funktion $x \mapsto A \cdot x$.

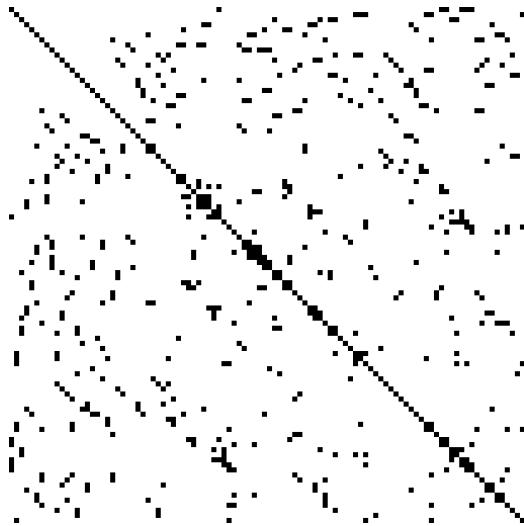


Abbildung 1: Dünnbesetzungssstruktur einer Finite Elemente Matrix.
Quelle: Oleg Alexandrov, https://de.wikipedia.org/wiki/Dünnbesetzte_Matrix

Unser **Ziel** ist es nun, ein Format zur Speicherung dünnbesetzter Matrizen zu entwickeln, welches die folgenden zwei Kriterien erfüllt:

1. Effiziente Speicherung: Im Wesentlichen nur Nichtrulleinträge und deren Koordinaten.
2. Effizientes Matrix-Vektor Produkt.

Ein Format, welches diese Kriterien erfüllt ist das:

1.4.1 Compressed Row Storage (CRS) Format

Um das CSR Format zu motivieren betrachten wir zunächst das Matrix-Vektor Produkt in der Perspektive "Zeile \times Spalte". Dazu sei $A \in \mathbb{R}^{m \times n}$ eine Matrix mit Zeilen $a_i^\top \in \mathbb{R}^{1 \times n}$ und $x \in \mathbb{R}^n$ ein Vektor gegeben. Dann gilt

$$A \cdot x = \begin{pmatrix} a_1^\top x \\ \vdots \\ a_m^\top x \end{pmatrix} =: \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix} = b,$$

wobei für $i \in \{1, \dots, m\}$ (Zeile für Zeile) die Skalarprodukte

$$b_i = a_i^\top x = \sum_{j=1}^n a_{ij} x_j = \sum_{j \in I_i} a_{ij} x_j , \quad I_i := \{j : a_{ij} \neq 0\},$$

berechnet werden müssen.

Beobachtung: Wir müssen für jede Zeile von A nur die Nichtrulleinträge und deren Spalten-Position (I_i) kennen. Dies führt zu folgendem Format mit den drei Listen

- **data**
Wir speichern die Nichtrulleinträge
- **indices**
Die zugehörigen Spalten-Indizes

- `indptr`
Wie viele Einträge pro Zeile? Für jede Zeile ein Adjazenzpaar `[start:stop]`
Differenz `stop-start` ist Anzahl Nichtnulleinträge in der jeweiligen Zeile

Beispiel: Siehe Abbildung 2

Sparse Matrix					data										
10	0	0	0	-2	10 -2 3 9 7 8 7 3 8 7 5 8 9 13										
3	9	0	0	0	(column) indices										
0	7	8	7	0	0 4 0 1 1 2 3 0 2 3 4 1 3 4										
3	0	8	7	5	row pointer (indptr)										
0	8	0	9	13	0 2 4 7 11 14										

Abbildung 2: Beispiel CSR Matrix

Können wir die **Matrix-Dimension** aus diesem Format ablesen?

- Zeilendimension: $m = \text{len(indptr)} - 1$
- Spaltendimension?
Man könnte vermuten $n = \max(\text{indices})$, aber was ist wenn die letzte Spalte der Matrix eine Nullspalte ist?
Stattdessen könnte man in `indptr[0]` die Spaltendimension speichern, da das erste Adjazenzpaar immer mit dem Index 0 beginnt.

Wann sparen wir **Speicherplatz**?

Wir definieren

$$\text{nnz} := \text{len(data)} = \text{Anzahl der Nichtnulleinträge.}$$

Dann muss in den entsprechenden Formaten folgende Anzahl an Zahlen gespeichert werden:

- Dense: $m \cdot n$
- CSR: $2 \cdot \text{nnz} + (m + 1)$

Und daher

$$2 \cdot \text{nnz} + (m + 1) < m \cdot n \Leftrightarrow \text{nnz} < \frac{(n - 1)m - 1}{2}$$

Beispiel: Tridiagonalmatrix

[Tafel: Grafik zu Tridiagonalmatrix]

- Dense: n^2
- Nichtnulleinträge: $\text{nnz} = n + 2(n - 1) = 3n - 1$
- CSR: $2 \cdot \text{nnz} + (m + 1) = 2(n + 2(n - 1)) + n + 1 = \dots = 7n - 3$ ((affin) linear in n)

[Tafel: Grafik n^2 VS $7n - 3$]

1.4.2 Sonstiges

Andere Formate

- **Compressed Sparse Column (CSC)**

Während CSR effizientes *row slicing* (man kann schnell die i -te Zeile berechnen; daher effizient: $x \mapsto A \cdot x$) ermöglicht, so ermöglicht CSC effizientes *column slicing* (daher effizient: $x \mapsto x^\top A = (A^\top x)^\top$). Sowohl CSR als auch CSC eignen sich nicht gut zur inkrementellen Manipulation/Erzeugung einer Matrix.

- **Coordinate list (COO)**

Speichert die Listen (`row`, `column`, `value`)

Zum Beispiel: Rating-Daten (Benutzer, Produkt (wie Film), Bewertung)

Die entsprechende Bibliothek in Python ist `scipy.sparse`
<https://docs.scipy.org/doc/scipy/reference/sparse.html>

1.5 Anwendung: PageRank

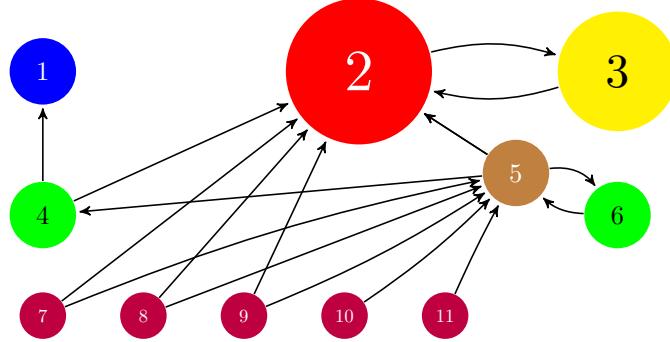
An Application of Eigenvectors: The *PageRank*

Aim: To rank results of a web search engine (such as Google) according to the “*importance*” of the web pages.

1998: For this purpose, Larry Page and Sergei Brin developed the PageRank algorithm as the basis of the Google empire.

Assumption: “*important*” means more links from other (important) web pages.

Idea: Let us think of the web as a directed graph, i.e., web pages are nodes and links from one page to another, i.e., from one node to another, are modeled as directed edges. For example a web structure consisting of 11 web pages could look as follows:



We now randomly place a random surfer according to the initial probability distribution $x^0 = (x_1^0, \dots, x_n^0)$ on this graph. Here, n (in the example above $n = 11$) denotes the number of web pages and x_i^0 denotes the probability that the random surfer starts at web page i . Further let $e := (1, \dots, 1)^T$, then the fact that x^0 is a probability distribution (i.e., probabilities sum up to 1) translates into $e^T x^0 = x_1^0 + \dots + x_n^0 = 1$. Now we make the assumption that the random surfer moves...

- (1) ...with probability $\alpha \in (0, 1)$ according to the link structure (with equal preferences to outgoing links)
- (2) ...with probability $(1 - \alpha)$ he can teleport to a random page (with equal probability) to prevent stranding in deadlocks

→ Pages, where the random surfer is more likely to appear in the long run based on the web's structure are considered more important.

These two movements can be described by multiplying the current probability distribution with the two following matrices:

$$P_1 = \left(\begin{array}{cccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 1 & 1 & & & & 1/2 & & & & & & \\ 2 & & & 1 & 1/2 & 1/3 & & 1/2 & 1/2 & 1/2 & & \\ 3 & & & & 1 & & & & & & & \\ 4 & & & & & & 1/3 & & & & & \\ 5 & & & & & & & 1 & 1/2 & 1/2 & 1/2 & 1 \\ 6 & & & & & & & & 1/3 & & 1 & 1 \\ 7 & & & & & & & & & & & \\ 8 & & & & & & & & & & & \\ 9 & & & & & & & & & & & \\ 10 & & & & & & & & & & & \\ 11 & & & & & & & & & & & \end{array} \right), \quad P_2 := \frac{1}{n}ee^T = \left(\frac{1}{n} \right)_{ij}$$

More precisely,

- (1) **Link structure:** P_1 is the probability matrix (column stochastic) defined by
 $P_1^{ij} :=$ Probability that random surfer moves from page j to page i defined by the link structure
- (2) **Jumps:** P_2 is the probability matrix (column stochastic) defined by
 $P_2^{ij} := \frac{1}{n} =$ Probability that random surfer jumps from page j to page i

The movement of the random surfer is then completely defined by the probability matrix

$$P = \alpha P_1 + (1 - \alpha) P_2.$$

This matrix is also known as the **Google Matrix**. For the next time instances we therefore obtain

$$\begin{aligned} x^1 &= \alpha P_1 x^0 + (1 - \alpha) P_2 x^0 = P x^0 \\ x^2 &= \alpha P_1 x^1 + (1 - \alpha) P_2 x^1 = P x^1 \\ x^{k+1} &= \alpha P_1 x^k + (1 - \alpha) P_2 x^k = P x^k = P^{k+1} x^0 \\ x^* &= \lim_{k \rightarrow \infty} x^k =: \textbf{PageRank} \end{aligned}$$

Observations:

- One can easily show that P_1 , P_2 and thus P are column stochastic (i.e., $e^T P = e^T$)
- Consequently, since x^0 is a probability distribution (i.e., $e^T x^0 = 1$), also $e^T x^k = 1$ for all k and $e^T x^* = 1$

Question:

Does this sequence $\{x^k\}_{k \in \mathbb{N}}$ of vectors converge (to a steady state)? More precisely, is there a $x^* = \lim_{k \rightarrow \infty} x^k = \lim_{k \rightarrow \infty} P^k x^0$, so that

$$P x^* = 1 x^*. \quad (3)$$

→ With other words, is there an **eigenvector** x^* to the **eigenvalue** 1 of the matrix P ?

→ **Eigenvalue algorithms** are developed to solve such problems. One of them is the **Power iteration**, which, applied to the eigenvalue problem above, produces precisely the sequence

$$x^k = P^k x^0.$$

Intuitively, in the limit most of the “mass” would be located at web pages that have many incoming links and would therefore be ranked as being more important. In fact, the i -th component of x^* is called the *PageRank* of the web page i .

Remark: *Perron Theorem*

A positive damping factor $\alpha > 0$ is also technically necessary as it assures that the matrix P has only strictly positive coefficients. The Perron Theorem then states that its largest eigenvalue is strictly larger than all other eigenvalues (in magnitude). Thus the convergence of the Power method is guaranteed. Since the matrix is column stochastic one can further show that the largest eigenvalue is 1.

1.5.1 Compute Page Rank matrix

<https://de.wikipedia.org/wiki/Google-Matrix>
adjazenz matrix, normalize, dangling nodes,..

1.5.2 Implementation Issues

How to implement $P_2 x$?

1.6 Anwendung: Optimierung

Let $A \in \mathbb{R}^{n \times n}$ be symmetric and positive definite (spd). Then A is in particular invertible, so that the linear system $Ax = b$ has a unique solution $x^* \in \mathbb{R}^n$ for all $b \in \mathbb{R}^n$. Let us relate this linear system to an optimization problem. For this purpose we define for a fixed spd matrix A and fixed right-hand side b the function

$$f := f_{A,b} : \mathbb{R}^n \rightarrow \mathbb{R}, \quad x \mapsto \frac{1}{2}x^T Ax - b^T x.$$

Then one can show the equivalence

$$Ax^* = b \iff x^* = \arg \min_{x \in \mathbb{R}^n} f(x).$$

In words, x^* solves the linear system on the left-hand side if and only if x^* is the unique minimizer of the functional f . In fact, you will learn in the next semester that the condition $Ax^* = b$ is the necessary first-order optimality condition:

$$0 = \nabla f(x) = Ax - b.$$

Due to the convexity of f this condition is also sufficient. Consequently, solving linear systems which involve spd matrices is equivalent to solving the associated optimization problem above, i.e., minimizing the function $f(x) = \frac{1}{2}x^T Ax - b^T x$. Thus, in this context iterative methods for linear systems, such as the Richardson iteration, can also be interpreted as optimization algorithms. Let us consider the (relaxed) Richardson iteration for $Ax = b$, i.e., $x_{k+1} = (I - \theta A)x_k + \theta b$. After some minor manipulations and making use of $\nabla f(x_k) = Ax_k - b$ we arrive at the equivalent formulation

$$x_{k+1} = x_k - \theta \nabla f(x_k).$$

The latter is what is called a gradient method. A step from x_k into (an appropriately scaled) direction of the gradient $\nabla f(x_k)$ yields a decrease in the objective function f , i.e., $f(x_{k+1}) \leq f(x_k)$. Along the Richardson aka Gradient method the scaling (also called step size) θ is fixed (in a machine learning context the step size θ is called the *learning rate*). However, one could also choose a different θ_k in each iteration step. This gives the more general version

$$x_{k+1} = x_k - \theta_k \nabla f(x_k). \tag{4}$$

The well known method of *steepest descent* is given by choosing

$$\theta_k = \frac{r_k^\top r_k}{r_k^\top A r_k}, \tag{5}$$

where $r_k := Ax_k - b$ is the k -th residual. This choice can be shown to be optimal in terms of convergence speed. Even more general, one can think of using a different preconditioner N_k in each iteration step

$$x_{k+1} = x_k - N_k \nabla f(x_k),$$

i.e., representing the gradient with respect to another inner product. This will later correspond to Newton-type optimization algorithms.

2 Projektaufbau planen

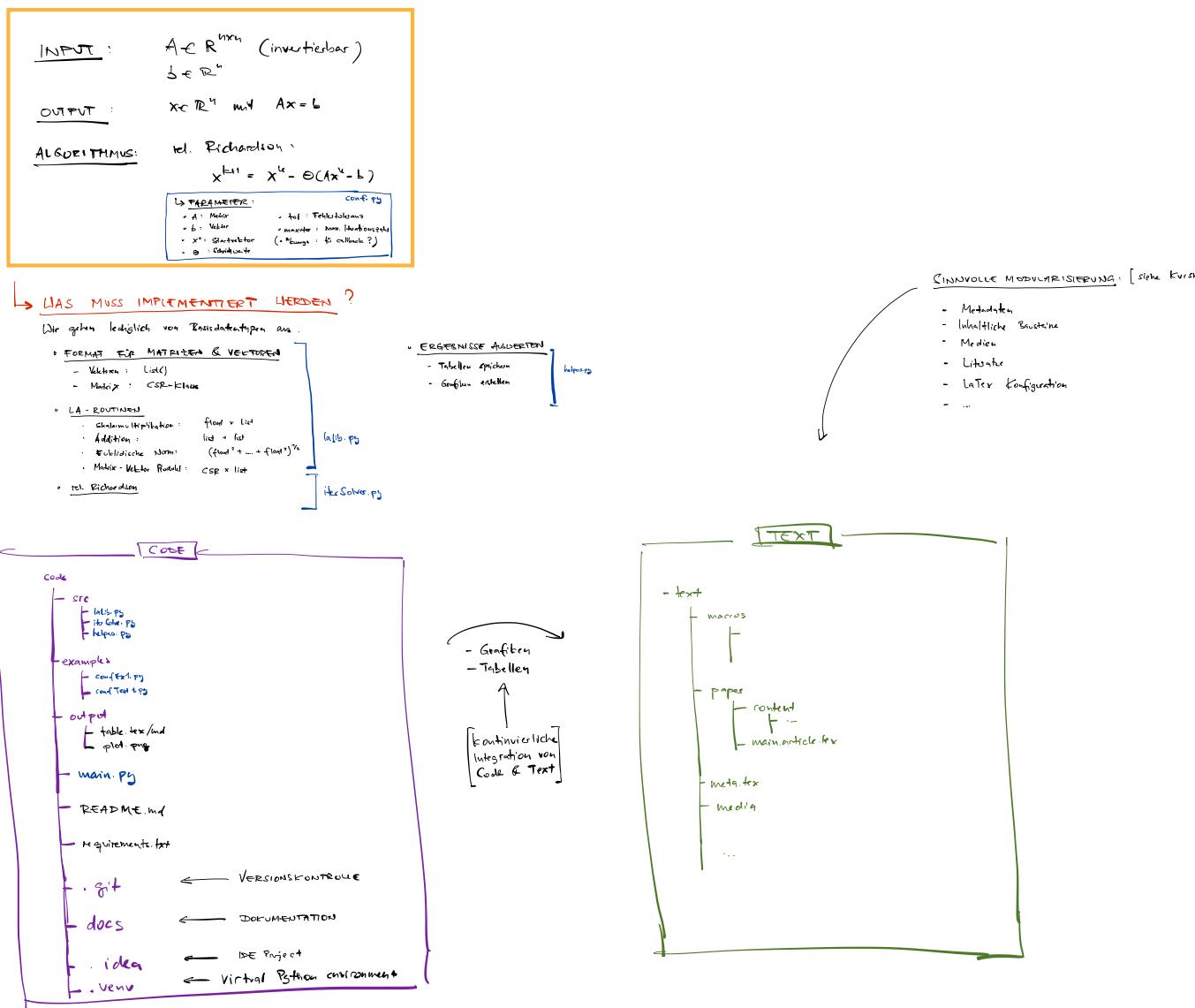


Abbildung 3: Skizze zu einem möglichen Projektaufbau.

Fallbeispiel: Sie müssen eine mathematische Arbeit über ein numerisches Thema verfassen (z.B. im Rahmen eines Seminars oder der Bachelor- bzw. Masterarbeit).

2.1 Code

Nachdem Sie die Fragestellung verstanden haben, versuchen Sie zunächst einmal numerische Ergebnisse zu produzieren. Dazu überlegen Sie sich idealerweise eine vernünftige Code-Infrastruktur, z.B.:

- Programmiersprache (hier Python)
- Arbeitsumgebung: jupyter, IDE wie Spyder, Texteditor+Konsole (hier IDE PyCharm)
- Versionskontrolle und backup
`git, github, (seafile?)`
- Was ist der Input? Hyperparameter? Welches Format? (Zahlen, Matrizen, Text, Grafik,...)
`Matrizen/Vektoren A, b, Verfahren?, maximale Iterationszahl, Fehlertoleranz,...`

- Was ist der Output? Welches Format? (Zahlen, Matrizen, Text, Grafik, Tabelle,...)
Vektoren x , Grafiken und Tabellen die den Konvergenzverlauf zeigen,...
- Was ist die Kernaufgabe des Codes? Welche Funktionen können modularisiert werden ? (linalg, src, helpers,...)
Verschiedene Funktionen für: Iteration, Erzeugung und speichern von Grafiken und Tabellen,...
- Was sind variable Parameter, die man ggf. komfortable in einer zentralen Konfigurationsdatei ändern möchte (conf.py)?
Beispiel: Problem A, b , Maximale Iterationsanzahl, Fehlertoleranz,...
- Was sind die Anforderungen an die Python-Umgebung? Werden externe Pakete benötigt?
requirements.txt
matplotlib zur Erstellung von Grafiken,...
- Sie schreiben Tests, welche der Code mindestens bestehen muss.
pytest, unittests
Sie testen ihren Löser auf verschiedenen Beispielen ($A, b; x$), bei denen Sie die Lösung $x = A^{-1}b$ kennen oder diese gar nicht existiert.
- Dokumentation und clean Code
PEP 8, sphinx,...

Wenn Ihr Betreuer in der Besprechung nach anderen Beispielen verlangt, können Sie diese nun bequem über Ihre Konfigurationsschnittstelle einspeisen.

Insgesamt erhalten Sie möglicherweise eine Verzeichnisstruktur wie in Abb. 4 für Ihren Projektblock “**Code**”.

```

|-- code
    |-- docs
    |-- src
        |-- linalg.py
        |-- iterative_solver.py
        |-- utils.py
    |-- examples
        |-- example_pagerank.py
        |-- example_heatequation.py
        |-- ...
    |-- data
    |-- test
    |-- main.py
    |-- README.rst
    |-- LICENSE
    |-- requirements.txt
    |-- .git
    |-- .idea

```

Abbildung 4: Eine mögliche Verzeichnisstruktur für das Code-Projekt. Beachten Sie, dass requirements.txt, .git und .idea automatisch erzeugt werden.

2.2 Text (skip)

Sie haben bereits erste numerische Ergebnisse produziert, die Sie Ihrem Betreuer oder der Welt in einem schönen Text präsentieren möchten. Als Mathematiker werden Sie feststellen, dass Sie unter Verwendung von Microsoft Office oder LibreOffice schnell an den Rande des Wahnsinns getrieben werden: Sie müssen mathematische Formeln bequem schreiben können, Nummerierungen sollten automatisch erzeugt werden, Grafiken und Tabellen sollten über generische Pfade importiert werden (ggf. gesteuert über Bash-Skripte), Sie brauchen ein Literaturverzeichnis etc. Spätestens ab diesem Punkt kommt \TeX bzw. \LaTeX ins Spiel, das nach einer kleinen Einarbeitungsphase Ihr Leben als Textautor nachhaltig vereinfachen wird.

Stellen Sie sich \LaTeX wie eine Programmiersprache vor: Auch hier gilt es Ihr (Text-)Projekt in geeignete Module zu unterteilen. Das dient nicht nur zur besseren Übersicht, sondern auch zur Wiederverwendbarkeit bestimmter Bausteine für andere Projekte (z.B. Bachelorarbeit → Masterarbeit). Eine Modularisierung könnte wie folgt aussehen:

- Inhaltliche Bausteine: Zusammenfassung, Kapitel 1, Kapitel 2, Listings,...
- Medien: Bilder, pdfs,...
- Literatur: pdfs, Literaturverzeichnis `literature.bib`,...
- Metadaten: Autename, Titel, Datum,...
- \LaTeX Spezifizierung: `usepackage`s, `commands`, `style`,...

Insgesamt erhalten Sie möglicherweise eine Verzeichnisstruktur wie in Abb. 5 für Ihren Projektblock “Text”.

```

| -- text/
|   |-- src/
|   |   |-- macros/
|   |   |   |-- meta.tex
|   |   |   |-- usepackage.tex
|   |   |   |-- style.tex
|   |   |   |-- commands.tex
|   |   |   |-- ...
|   |   |-- content/
|   |   |   |-- abstract.tex
|   |   |   |-- introduction.tex
|   |   |   |-- section1.tex
|   |   |   |-- section2.tex
|   |   |   |-- listing.py
|   |   |   |-- ...
|   |-- media/
|   |   |-- picture.png
|   |   |-- picture.jpg
|   |   |-- ...
|   |-- literature/
|   |   |-- literature.bib
|   |   |-- pdfs/
|   |   |   |-- book.pdf
|   |   |   |-- paper.pdf
|   |   |   |-- ...
|   |-- main.tex

```

Abbildung 5: Eine mögliche Verzeichnisstruktur für das Text-Projekt.

3 Secure Shell (SSH)

Was ist SSH?

- SSH ist ein Protokoll für *verschlüsselte* Verbindungen zwischen zwei Rechnern.
- SSH kann die Kommandozeile eines entfernten Rechners lokal zur Verfügung stellen.
- SSH kann Dateien über das Netzwerk transferieren.

Warum ist das für uns interessant?

- Für rechenintensive Programme verwenden wir *number cruncher* (Remote-Server mit viel Rechenleistung). Dafür müssen Dateien wie Quellcode, Inputdaten etc. verschlüsselt zwischen Rechnern transferiert werden.
- Wir verwalten unser Projekt auf github.com. Wir transferieren unsere Dateien sicher über einen verschlüsselten SSH-Tunnel zwischen diesem Server und unserer lokalen Maschine.

Demo:

- ssh auf remote
- remote: Programm ausführen
- scp zum kopieren von Daten

Weitere Bemerkungen:

- SSH wurde erstmals 1995 von dem finnischen Informatiker Tatu Ylönen entwickelt und ist heute die gängigste Methode für den Zugriff auf Remote-Linux-Server.
- SSH ist das allgemeine Protokoll. Die de facto Standardimplementierung ist OpenSSH (1999 von den OpenBSD-Entwicklern als Open-Source-Software veröffentlicht).
- Client Server Modell
 - client (lokal): ssh
 - server (entfernt): sshd

Wiederholung Public-Key-Verschlüsselungsverfahren (asymmetrisch)

- Schlüsselpaar ($k_{\text{pub}}, k_{\text{priv}}$)
 - öffentlicher Schlüssel k_{pub} kann geteilt werden
 - privater Schlüssel k_{priv} muss geheim gehalten werden
- Verschlüsselung mit öffentlichem Schlüssel: $y = E(x, k_{\text{pub}})$
- Entschlüsselung mit privatem Schlüssel: $D(y, k_{\text{priv}}) = x$
- Vorteil gegenüber symmetrischer Verschlüsselung (bei der für E und D derselbe Schlüssel verwendet wird): Alice und Bob müssen sich vor Kommunikation nicht nachts im Wald treffen, um sich auf ein Geheimnis zu einigen
- Berühmtes Beispiel: RSA (siehe etwa Zertifikate im Browser)

3.1 Verwendung

Passwort-basierte Authentifizierung:

```
$ ssh [USERNAME]@[IP_ADDRESS or HOSTNAME]
```

- IP_ADDRESS or HOSTNAME: Postanschrift des entfernten Rechners
- USERNAME: Benutzername des Accounts auf dem entfernen Rechner
- Passwort-Prompt

Schlüsselbasierte Authentifizierung:

- Erstellen eines Schlüsselpaars

```
ssh-keygen -t ed25519 -f [FILENAME] -C "your_email@example.com"
```

- Speicherort: \$HOME/.ssh
- Passphrase, um den privaten Schlüssel zu schützen

- Dateien anschauen

```
cd ~/.ssh
ls -al
cat [FILENAME]
```

- Schlüssel auf remote kopieren

```
ssh-copy-id -i [FILENAME] [IP_ADDRESS or HOSTNAME]
```

Server speichert den öffentlichen Schlüssel in .ssh/authorized_keys

- Nun einfach:

```
$ ssh [USERNAME]@[IP_ADDRESS or HOSTNAME]
```

ssh-agent (skip)

3.2 Quellen

- <https://www.digitalocean.com/community/tutorials/how-to-use-ssh-to-connect-to-a-remote-server>
- https://en.wikipedia.org/wiki/Secure_Shell
- <https://en.wikibooks.org/wiki/OpenSSH>
- https://en.wikipedia.org/wiki/Public-key_cryptography#

4 Versionskontrolle: git und GitHub

Was ist Versionskontrolle?

- Ein System, das Änderungen an Dateien im Laufe der Zeit aufzeichnet, sodass bestimmte Versionen später wieder hergestellt werden können.
- Engl.: Version Control System (VCS)

Warum ist das für uns interessant?

- Fehlersuche: Wir können jede vorgenommene Änderung im Zeitverlauf überprüfen und ggf. einen früheren (hoffentlich stabilen) Zustand (snapshot) wiederherstellen.
- Kollaboration: Wir können zusammen an einem Projekt arbeiten und unabhängige Beiträge zum Code verwalten.

Was ist git?

- git ist ein VCS und der de facto Standard
- Ursprünglich entwickelt von Linus Torvalds im Jahr 2005 für die Entwicklung des Linux-Kernels:
<https://www.youtube.com/watch?v=o8NP11zkFhE?t=07m40s>
- git ist ein verteiltes Versionskontrollsystem: Jeder Benutzer klont eine Kopie des Projekts und hat den gesamten Verlauf auf seiner eigenen Maschine.
- Der Ruf von git:



Abbildung 6: <https://xkcd.com/1597/>

4.1 git

[Tafel: DAG aufbauen]

Datenmodell

- blob : Datei
- tree : Verzeichnis

- `commit` : snapshot of top-level tree
 - Date
 - Author
 - Message
 - Parent
- Historie wird gespeichert als gerichteter azyklischer Graph (directed acyclic graph (DAG)): gerichteter Graph ohne gerichtete Zyklen (z.B. Stammbäume)
 - Knoten = `commits`
 - Kanten = Verwandtschaftsbeziehung (genauer: $(x,y) := \text{"commit } y \text{ kommt vor commit } x\text{"}$)
- `object` : blob | tree | commit
 - git speichert objects mithilfe ihrer SHA-1 Hashwerte (20 Bytes bzw. 40 Hexadezimalzahlen)
- `references` = menschenlesbare Namen für Hashwerte von `commits` (also Zeiger auf Knoten im Graphen)
 - Beispiele:
 - * `master/main`: typischerweise letzter commit im Hauptzweig
 - * `HEAD`: commit der dem aktuellen lokalen Zustand entspricht
 - Wichtig, da neue commits den Graphen vom aktuellen commit als Startpunkt aus erweitern
 - Daher spricht man auch nicht von Benamung, sondern von Abzweigung/Verästelung (branching)
 - Drei Varianten von `references`:
 - * local branch
 - * remote branch
 - * tag = reference, die sich nicht bewegt (für Softwareversionen)
- `repository` : Vereinigung von allen `objects` und `references`
 - Das ist im Wesentlichen alles was git auf der Festplatte speichert (in `.git`)

Staging area: Wie erstellen wir `commits`?

- Eine Möglichkeit wäre: Snapshot (`commit`) auf der Grundlage des aktuellen Zustands des Projektverzeichnisses erstellen.
- Warum das zu grobkörnig ist:
 - Beispiel 1: Angenommen wir haben zwei verschiedene Features implementiert und wollen dafür zwei `commits` erstellen, wobei der erste das erste Feature einführt und der nächste das zweite.
 - Beispiel 2: Angenommen wir haben print-Anweisungen zum Debuggen eingefügt, zusammen mit einer Fehlerbehebung. Wir möchten die Fehlerbehebung committen, während wir alle Druckanweisungen verworfen.
- git erlaubt durch einen Mechanismus namens “staging area” festzulegen, welche Änderungen in den nächsten Snapshot (`commit`) aufgenommen werden sollen.

git verwenden: CLI

[Demo und Hands-On]

Alle Git-Befehle entsprechen letztlich einer Manipulation des `commit`-DAG durch Hinzufügen von Objekten und Hinzufügen/Aktualisieren von Referenzen.

- `git init`: erstellt ein neues git repo, dessen Daten (`objects` und `references`) im Verzeichnis `.git` gespeichert werden
- `git status`: gibt Auskunft über den aktuellen Stand der Dinge
- `git add [FILENAME]`: fügt Dateien zur staging area hinzu
- `git commit -m <MESSAGE>`: erstellt neuen commit

- `git log -oneline -abbrev-commit -all -graph -decorate -color`: Zeigt Veränderung (in Graphenansicht)
- branch and merge: Erstellen von references
 - `git branch <branchname> [<start-point>]`
 - `<branchname>` Name für den commit des aktuellen Zustandes oder für den commit `<start-point>`
 - `<start-point>` ist optional (branch, tag, hash_value)
 - Mal in `.git/refs/heads` schauen: wir schreiben lediglich eine Textdatei mit 40 Hex-Zahlen (das geht sehr schnell)
 - Wir können Zweige zusammenbringen, in dem wir zwei commits verkleben
`git merge`
- `git checkout <commit>` : aktuelles Verzeichnis gemäß des commits `<commit>` herstellen, also Update von HEAD
 - `<commit>` ist optional (branch, tag, hash_value)

4.2 GitHub

Situation

- Mehrere Entwickler arbeiten zusammen an einem Projekt.
- Das bedeutet: Es bestehen mehrere Klone des Projekts auf den lokalen Platten der Entwickler.
- Den Kunden möchte man allerdings nur einen Ort als Quelle angeben
- Dafür gibt es sogenannte remote–Repositories: Ein gemeinsames Repository, das alle Entwickler zum Austausch ihrer Änderungen verwenden (Pivot-Repo).
- In den meisten Fällen wird ein solches Remote-Repository auf einem Code-Hosting-Dienst wie GitHub oder auf einem internen Server gespeichert.

Remotes

[Demo und Hands-On: Zweites Repo anlegen und erstes Repo als remote definieren]

- Nun kommen die oben erwähnten references der Kategorie `remote branches` ins Spiel
- `remote hinzufügen:`
`git remote add <name> <url>`
- wir können mehrere remotes hinzufügen, daher legen wir für die lokalen branches mit welchem `remote` sie verknüpft werden:
`git branch -set-upstream-to=<remote-name>/<remote-branch> <local-branch>`
- `git pull = git fetch + git merge`
- `git push`
- `git clone`

GitHub

[Demo and Hands-On]

- Account anlegen: <https://github.com>
- ssh Schlüssel hinterlegen
- `git repo erstellen`
- auf lokalen Rechner klonen
- Workflows testen

4.3 Quellen

- <https://missing.csail.mit.edu/2020/version-control/>
- <https://www.freecodecamp.org/news/learn-the-basics-of-git-in-under-10-minutes-da548267cc91/>
- <https://think-like-a-git.net>
- <https://en.wikipedia.org/wiki/Git>
- <https://www.youtube.com/watch?v=8JJ101D3knE>

5 Entwicklungsumgebung

Eine **integrierte Entwicklungsumgebung** (IDE, integrated development environment) ist eine Sammlung von Computerprogrammen, mit denen die Aufgaben der Softwareentwicklung möglichst ohne "Medienbrüche" (zB. zwischen Texteditor, Dateimanager, Terminal-emulator, Bildbetrachter...) bearbeitet werden können.

Integriert vs. nicht integriert: Um diesen Unterschied zu "erleben", setzen wir die folgenden möglichen Arbeitsabläufe der Softwareentwicklung um:

- Nicht integriert: Texteditor + Terminal + Interpreter + Bildbetrachter + ...
- Integriert: IDE
- Browserbasiert

Anschließend befassen wir uns intensiver mit der IDE PyCharm von JetBrains.

5.1 Nicht integriert: Texteditor + Terminal + Interpreter + ...

Hausaufgaben:

Programmieren über die Konsole

[Demo: CLI und GUI parallel]

1. VPN Verbindung herstellen
2. Anmelden auf syrma [ssh oder über Webinterface guacamole]
3. Im home-Verzeichnis:
 - Ein .py-Skript mit einem Editor (z.B. nano) erstellen.
 - Erste Zeile Shebang/Hash-Bang setzen: `#!<PATH-to-PythonInterpreter>` zum Beispiel `/usr/bin/python3`
 - Endlos-Schleife bauen mit einem print-Befehl im Schleifenkörper
4. Das .py-Skript mit dem Interpreter python3 ausführen.
5. Das Programm vorzeitig abbrechen mit strg+c.
6. Aus Endlos-Schleife eine "sehr lange" Schleife machen, die sicher abbricht.
7. Mit python3 ausführen und Standard-Output via ">" in eine Textdatei wegschreiben.
8. Nochmal mit python3 ausführen und Standard-Output diesmal via "> >" in eine Textdatei wegschreiben.

Als *nicht integriert*, trotzdem gelegentlich *Entwicklungsumgebung* genannt, gilt der Einsatz einzelner Programmierwerkzeuge innerhalb eines Arbeitsablaufes, zum Beispiel:

- Texteditor (zB nano)
- Terminal-emulator
- Compiler bzw. Interpreter (zB python3)
- Linker, Debugger,...
- Dateimanager, Bildbetrachter,....

→ Entwickler muss die einzelnen Arbeitsschritte gezielt anstoßen (siehe Hausaufgabenkasten oben)

5.2 Integriert: Integrated Development Environment (IDE)

Die einzelnen Programmierwerkzeuge (Texteditor, Terminal, Interpreter, Dateimanager,...) können auch in eine mächtige Software mit grafischer Oberfläche *integriert* werden; man spricht von einer *integrierten Entwicklungsumgebung*.

Typischerweise bieten diese noch diverse weitere sehr nützliche Features, wie Interpreter via ssh, Syntax-Highlighting, Virtuelle Python-Interpreter venv, Versionskontrolle (VC) mittels zB git, Debugging tools, History, Styleguide-Inspektion u.v.m.

Demo:

1. tmux
2. Spyder3
3. PyCharm

5.3 Browserbasiert

Wir haben bisher eine andere Programmierumgebung genutzt: **Jupyter Notebook**

- Jupyter Notebook ist eine web-basierte (=“läuft im Browser”) interaktive (=“wir lassen gleich einzelne Teile des Codes laufen”) Umgebung.
- Wurde entworfen um schnelle Code-Prototypen zu entwickeln, Text (markdown, Latex) einzubinden, Präsentationen zu erstellen, unterschiedliche Programmiersprachen zu nutzen,...
→ Daher besonders gut zum Einstieg und für die Lehre geeignet.
- Eher ungeeignet für die Entwicklung eines aufwendigeren Projekts.
- Siehe auch Nachfolger **JupyterLab**

5.4 Die IDE PyCharm von jetbrains

JetBrains ist ein russisch-tschechisches Software-Unternehmen, das 2000 gegründet wurde und u.a. folgende Produkte anbietet:

- **PyCharm:** IDE für Python
- Tutorials: PyCharm Edu
- **CLion:** IDE für C und C++
- **IntelliJ IDEA:** IDE für Java
- ...

PyCharm download and configure

PyCharm Projekt anlegen

PyCharm Projekt anlegen

1. PyCharm Projekt anlegen:

→ Dazu Ordner Seafile/<ProjektName>/code mit PyCharm als Anwendung öffnen

<https://www.jetbrains.com/help/pycharm/setting-up-your-project.html>

PyCharm legt einen Ordner **.idea** an

Der Projektordner (bei uns Seafile/<ProjektName>/code) beinhaltet das Verzeichnis **.idea** mit den folgenden Dateien:

- .iml Datei → Projektstruktur
- workspace.xml Datei → workspace Einstellungen
- Einige weitere .xml–Dateien: Jede .xml Datei ist verwantwortlich für die sich aus dem Dateinamen ableitbaren Einstellungen, zB vcs.xml für Version Control System (wir verwenden später git)

Alle Einstellungsdateien aus dem .idea Ordner sollten unter Versionskontrolle gesetzt werden; außer workspace.xml, da dort die *lokalen* Einstellungen gesetzt werden (von Programmierer zu Programmierer unterschiedlich). Die Datei workspace.xml sollte also vom VCS (bei uns git) ignoriert werden. Das regelt PyCharm für uns; siehe zB .idea/.gitignore

Interpreter und .venv [Siehe in diesem Zusammenhang auch: Das Python Kapitel über **Modularisierung**]

Dazu hier ein Ausschnitt aus den Python Tutorials zu **venv's**:

*Python applications will often use packages and modules that don't come as part of the standard library. Applications will sometimes need a specific **version of a library**, because the application may require that a particular bug has been fixed or the application may be written using an obsolete version of the library's interface.*

*This means it may **not be possible for one Python installation to meet the requirements of every application**. If application A needs version 1.0 of a particular module but application B needs version 2.0, then the requirements are in conflict and installing either version 1.0 or 2.0 will leave one application unable to run.*

*The solution for this problem is to create a virtual environment, a **self-contained directory tree** that contains a Python installation for a particular version of Python, plus a number of additional packages.*

Demo:

1. Vergleich: venv VS. systemweite Installation

```

1 >>> import sys
2 >>> sys.path
3 ['', '/usr/lib/python38.zip', '/usr/lib/python3.8', '/usr/lib/python3.8/lib-dynload',
   '/home/vollmann/.local/lib/python3.8/site-packages', '/usr/local/lib/python3.8/
   dist-packages', '/usr/lib/python3/dist-packages']

```

requirements.txt Die Pakete, die für unser Projekt benötigt werden, können hier erwähnt werden.

6 Code Dokumentation mit sphinx

Der folgende Abschnitt ist nur ein Leitfaden für die Präsentation im Unterricht. Einen geeigneten Einstieg finden wir in der offiziellen Dokumentation

<https://www.sphinx-doc.org/en/master/usage/quickstart.html>

- Beliebige Python–Dokumentation im Webbrowser finden: Unten im Footer meist Info über verwendete Software
- Allgemein wollen wir u.a.:
 - Automatisierung
 - Information nur an einer Stelle und daher Synchronisation von Code und Dokumentation
 - ggf. mathematische Formeln (brauchen entsprechende Markup Language)
- Wir verwenden dazu in diesem Kurs die Software sphinx (geschrieben in Python)
- Dokumentation: <https://www.sphinx-doc.org/en/master/index.html>
- Installation
 - GUI: strg+alt+s: project, interpreter -> install sphinx
 - CLI: \$ pip3 install sphinx

6.1 Getting Started

- PyCharm Terminal (.venv) \$
- Zum Beispiel Version checken: `sphinx-build --version`

in code/

- \$ mkdir docs
- \$ sphinx-quickstart <PATH-to-ROOTDIR> (hier einfach docs/)
 - Separieren source and build [y]
 - name: ...
 - release: 0.1
- Nun mal Verzeichnis code/docs/ inspizieren
 - code/docs/source/
 - * hier liegen die Quelldateien um die Dokumentation zu steuern
 - * conf.py
 - hier können wir die Dokumentation konfigurieren
 - u.a. wurden unsere Antworten der interaktiven Abfrage von sphinx-quickstart dort gespeichert
 - * index.rst ist unsere Startseite (unsere Hauptdatei/"Kleber")
 - rst wie reStructuredText (ähnlich wie markdown)
 - Direktiven und deren Optionen
 - Überschriften ==, ----
 - setze zB .. note::
 - code/docs/build/
 - * Dieses Verzeichnis ist noch leer
 - * Das ändern wir nun
 - build
 - * \$ sphinx-build -b html docs/source/ docs/build/html
 - * in build/ Verzeichnis schauen
 - * Wir finden nun einen Ordner html
 - * Wir öffnen die Datei index.html mit einer geeigneten Software (= Browser)

- * `index.rst` [in reStructuredText] wird zu `index.html`
The file `index.rst` created by `sphinx-quickstart` is the root document, whose main function is to serve as a welcome page and to contain the root of the “table of contents tree” (or `toctree`). alternativer build command ist nun: `make html` in `docs`/

- Bemerkung zu den sphinx-binaries:
 - Diese befinden sich in unserer virtuellen Python Umgebung unter: `venv/code/.<VENV-NAME>/bin`
 - siehe auch: `$ locate sphinx-*` (ggf. `$ sudo updatedb` notwendig) (alternativ: `$ type sphinx-*`)

6.2 Maßschneiderung in `conf.py`

```
extensions = []
```

- Erweiterungen stellen zusätzliche Funktionalitäten bereit.
- Hier können wir u.a. leicht Schalter setzen, um das Erscheinungsbild zu steuern.

6.3 Allgemeines Erscheinungsbild: `html_theme`

- **builtin schemes**

- siehe: <https://www.sphinx-doc.org/en/master/usage/theming.html#builtin-themes>
- Beispiel: `classic`
- in `conf.py`:


```
html_theme = 'classic'
```
- Neu bauen: `...docs/ $ make html`

- **third-party schemes**

- siehe: <https://sphinx-themes.org/>
- Beispiel: `sphinx-rtd-theme` (read the docs)
- Diese müssen extra installiert werden, da nicht in Standard-Sphinx-Installation enthalten
 - * `pip install sphinx-rtd-theme` oder über GUI
- in `conf.py`:


```
html_theme = 'sphinx_rtd_theme'
```

6.4 Struktur ausbauen

- Neue Unterseiten anlegen, zB `docs/source/usage.rst`
- Neu bauen: `...docs/ $ make html`
- Warnung:
`checking consistency... /.../usage.rst: WARNING: document isn't included in any toctree`
 » Diese Seite möchte noch verlinkt werden damit wir darauf zugreifen können
- In unserer Hauptdatei zum Inhaltsbaum hinzufügen:

```
.. toctree::  
   usage
```

6.5 Synchronisation von Code und Dokumentation: `autodoc`

- Nun wollen wir in der Dokumentation den Code beschreiben und dabei auf vorhandene docstrings zurückgreifen.
- Das ist eine zusätzliche Funktionalität, die durch eine Erweiterung bereitgestellt wird.
 - genauer autodoc: <https://www.sphinx-doc.org/en/master/usage/extensions/autodoc.html>
- aktivieren in `docs/source/conf.py`
 - zur Liste hinzufügen: `extensions = ['sphinx.ext.autodoc',]`

- sonst Fehler wie ERROR: Unknown directive type “autofunction”.
- Nun können wir zum Beispiel folgende Direktiven verwenden:


```
.. autofunction:: src.linalg.axpy
.. autoclass:: src.linalg.csr_matrix
```
- Wir bekommen weiterhin Fehler, da die relativen Pfade src.linalg nicht erkannt werden. Daher den absoluten Pfad zu <PATH>/code/ dem .venv Interpreter mitgeben. Anpassung in conf.py:


```
import pathlib
import sys
sys.path.insert(0, pathlib.Path(__file__).parents[2].resolve().as_posix())
```

» Der String `pathlib.Path(__file__).parents[2].resolve().as_posix()` sollte dann genau den Pfad zu `/.../code/` enthalten
- Wir bekommen weiterhin einen Fehler, da unsere docstrings nicht im reStructuredText Format sind, sondern NumPy Format

» Diesen Fehler beheben wir im nächsten Schritt

6.6 docstring Style: reStructuredText, NumPy, Google

- Dokumentation lesen: <https://sphinxcontrib-napoleon.readthedocs.io/en/latest/index.html>
- für NumPy Format: `sphinxcontrib-napoleon`
- Installation: GUI oder in CLI via (.venv) `$ pip install sphinxcontrib-napoleon`
- `conf.py`: extension hinzufügen
- Type Annotations: <https://sphinxcontrib-napoleon.readthedocs.io/en/latest/index.html#type-annotations>
- Examples in docstring, Code-Zeile hinter `>>>`
- Tipp: Docstring Format in PyCharm Settings setzen:
`str alt s | Tools | Python Integrated Tools | Docstrings > Docstring Format: NumPy`
- Nun auch nochmal **Indices** schauen: global, module, Search Page

6.7 Reference Guide: Automatisch alle Code-Bestandteile auflisten mit ... autosummary::

- `'sphinx.ext.autosummary'`
- API references/Reference Guide VS User Guide
- Wir legen dazu eine neue Unterseite an, zum Beispiel: `api.rst`
- `toctree`-Direktive in `index.rst` ergänzen
- Die Direktive `... autosummary::` wird durch die Extension `sphinx.ext.autosummary` freigeschaltet
- In `docs/source/api.rst` zum Beispiel:

```
Reference Guide
=====
... autosummary::
   :toctree: generated

   src.linalg
```

6.8 Links zwischen Quellcode und docs: [source], [docs]

- extension aktivieren: `sphinx.ext.viewcode`

6.9 Cross–References: Links innerhalb der Dokumentation

- Label setzen vor Überschrift . . _LABEL-TAG:
- Referenz auf den markierten Abschnitt: :ref:LINK-NAME <LABEL-TAG>

6.10 Mathematische Formeln . . math::

Zum Beispiel können wir dann L^AT_EX verwenden setzen:

. . math::

```
x\in\mathbb{R}^n
```

6.11 Andere Build-Formate

Zum Beispiel können wir auch mit dem Builder latex eine PDF–Dokumentation erstellen:

- Im Verzeichnis code/docs/: \$ make latexpdf
- Alternativ im Verzeichnis code/: \$ sphinx-build -b latex docs/source/ docs/build/latex/

6.12 Veröffentlichen mit Github-Pages

1. ggf. neuer branch "gh-pages"
2. index.html in /docs mit redirect auf /build/html/index
3. in docs/: .nojekyll
4. unter Setting/Pages "gh-pages" und "docs" auswählen

7 Mathematische Textverarbeitung mit \LaTeX und TeXstudio (Skip)

Ein Lehrbuch zu \LaTeX bietet die Quelle:

Schnell ans Ziel mit LaTeX 2e [4],

von Jörg Knappen, auf das wir mithilfe der ZIMK-Kennung beim De Gruyter Verlag Zugriff haben.
Eine Zusammenfassung finden wir hier:

LaTeX 2e Kurzbeschreibung

Klar strukturierte und verständliche Tutorials/Dokumentationen finden wir auf der Homepage des startup Unternehmens *overleaf*:

https://www.overleaf.com/learn/latex/Main_Page

7.1 Einführung

- \TeX : Computerprogramm zum Setzen von Texten und mathematischen Formeln von Donald E. Knuth; Erscheinungsjahr 1978

Ein auch hier sehr passendes Zitat von ihm:

Man versteht etwas nicht wirklich, wenn man nicht versucht, es zu implementieren.

- \LaTeX : Computerprogramm aufbauend auf \TeX von Leslie Lamport; Erscheinungsjahr 1984
- $\text{\LaTeX} 2\epsilon$: ist die aktuelle Version

7.1.1 Unterschied zu Programmen wie Word

- **\LaTeX :**

kann als Buch-Designer verstanden werden: Mithilfe von "Befehlen" geben wir dem Computer-Programm \LaTeX zu erkennen, was zB eine Überschrift sein soll (der Befehl lautet dann z.B. `\section{Überschrift}`), was ein Zitat (der Befehl lautet dann z.B. `\cite[Quelle]`), etc.

Sofern wir dem Programm \LaTeX diese logische Struktur zu erkennen geben, kann dieses für uns die Gestaltung übernehmen (dabei können wir das Design auch konfigurieren).

→ bei der Eingabe ins Programm (Texteditor), sehen wir in der Regel noch nicht sofort, wie der Text nach dem Formatieren aussehen wird

- **Visuell orientierte Textverarbeitungsprogramme (z.B. Word):**

Autor legt das Layout des Textes gleich bei der interaktiven Eingabe fest

→ nach dem Prinzip *wysiwyg* ("what you see is what you get")

7.1.2 Vorteile von \LaTeX

- wir geben nur logische Struktur vor, z.B. `\section{Kapitel}`, `\subsection{Unterkapitel}`... und \LaTeX übernimmt die sinnvolle Gestaltung für uns, zB soll die Schriftgröße der Kapitelüberschrift größer sein, als die des Unterkapitels. Zudem soll die Nummerierung automatisch ablaufen samt Erzeugung des Inhaltsverzeichnisses etc.
- Setzen von **mathematischen Formeln**, Fußnoten, Literaturverzeichnisse, Tabellen etc. sehr leicht

7.2 Mögliche Arbeitsabläufe

1. Nicht integriert über die Konsole:

- (a) Texteditor: Eingabedatei ("Quellcode") mit \LaTeX Befehlen schreiben
→ `text.tex`
- (b) Kompilieren: diese Datei mit \LaTeX -Compiler bearbeiten
→ dabei wird eine Datei erzeugt, die den gesetzten Text in einem geräteunabhängigen Format speichert
→ zB `text.pdf`
- (c) Vorschau: `text.pdf` mit einem PDF-Betrachter oder im Terminal anschauen

(d) Falls nötig: Zurück zu (a) und Korrekturen einarbeiten

2. Integriert: Eine IWE (Integrated Writing Environment): \LaTeX Editor

- Ähnlich wie eine IDE können die obigen Arbeitsschritte aus einer einzigen Software, eine IWE oder in diesem Fall auch einfach \LaTeX Editor genannt, heraus gesteuert werden. Zudem kommt diese mit einer ganzen Menge weiterer Features (Syntax-Highlighting, Tab Completion, ...) daher.
- Wir werden in dieser Veranstaltung das Programm TeXstudio verwenden.
- Andere beliebte Editoren: TeXmaker (multi-platform), TeXnicCenter (speziell für Windows entwickelt)

3. Webbrowserbasiert: Online \LaTeX Editor

- web-basiertes Arbeiten ist besonders hilfreich, wenn man zusammen am selben Dokument arbeitet und die Dateien auf einem zentralen Server bearbeiten möchten
- bei uns sehr beliebt ist zurzeit **Overleaf**:
"Overleaf is a collaborative cloud-based \LaTeX editor used for writing, editing and publishing scientific documents."

7.3 Die .tex-Eingabedatei

Das Eingabedatei für \LaTeX ist eine Textdatei mit der Endung .tex. Diese Datei wird mit einem Texteditor erstellt und enthält sowohl den **Text**, der gedruckt werden soll, als auch die **Befehle**, aus denen \LaTeX erfährt, wie der Text gesetzt werden soll.

7.3.1 Leerstellen

“Unsichtbare” Zeichen wie das Leerzeichen, mehrere Leerzeichen, Tabulatoren und das Zeilenende werden von \LaTeX einheitlich als Leerzeichen behandelt.

7.3.2 \LaTeX Befehle

Format von \LaTeX Befehlen:

1. Standard:

$\backslash\text{Befehlsname}\{\text{ggf. Parameter}\}$

- Startzeichen: Backslash “\”
- Befehlsname: Besteht nur aus Buchstaben (case-sensitive!)
- ggf. {Parameter}
- Ende: Leerzeichen oder Sonderzeichen

Wenn man nach einem Befehlsnamen eine Leerstelle erhalten will:

$\backslash\text{Befehlsname}\{\}$ oder $\backslash\text{Befehlsname}\backslash$ oder $\backslash\text{Befehlsname}\sim$

2. Backslash “\” + Sonderzeichen, z.B.

$\backslash\#, \backslash\&, \backslash\{, \dots$

7.3.3 Gruppen

Gruppen werden erzeugt durch Einrahmung mit geschweiften Klammern

{ ...
Gruppeninhalt
... }

- Befehle innerhalb einer Gruppe, wirken nur innerhalb der Gruppe
- Beispiel: $\backslash\text{bf}, \backslash\text{color}\{\text{Farbe}\}$

7.3.4 Umgebungen

Die Kennzeichnung von speziellen Textteilen, die anders als im normalen Blocksatz gesetzt werden sollen, erfolgt mittels sogenannter Umgebungen (environments) in der Form

```
\begin{Umgebungsname}
  ....Umgebungskörper....
\end{Umgebungsname}
```

- Umgebungen sind *Gruppen*.
- Verschachtlung möglich.

7.3.5 Kommentare

Alles, was hinter einem Prozentzeichen % steht (bis zum Ende der Eingabezeile), wird von L^AT_EX ignoriert.

Aus...

Die Funktion \$f\colon \mathbb{R} \rightarrow \mathbb{R}\$ % dieser Text wird ignoriert
wird...

Die Funktion $f: \mathbb{R} \rightarrow \mathbb{R}$

7.3.6 Aufbau

Die Eingabedatei hat folgende Struktur:

```
\documentclass[Dokumentklasse]
%so muss jede Eingabedatei starten; legt Art des Schriftstücks fest
:
Präambel % Makropakete laden, ...
:
\begin{document}
:
Setzen des Schriftstücks
:
\end{document} % alles nach diesem Befehl wird von LATEX ignoriert
```

7.3.7 Dokumentklassen

Der Start einer jeden Eingabedatei lautet

```
\documentclass[<optionen>]{<klasse>}
```

- **Klassen:** enthält Vereinbarungen über das Layout und die logischen Strukturen, z.B. die Gliederungseinheiten (Kapitel etc.), die für alle Dokumente dieses Typs gemeinsam sind
→ **muss** angegeben werden

article	für Artikel in wissenschaftlichen Zeitschriften, kürzere Berichte,...
beamer	für Präsentationen

Tabelle 1: Für uns interessante Dokumentklassen

Xpt wählt die normale Schriftgröße des Dokuments aus (default: X = 10)
a4paper für Papier im DIN A4-Format (default: amerikanisches Papierformat)

Tabelle 2: Für uns interessante Klassenoptionen

- **Optionen:** Zwischen den eckigen Klammern können, durch Kommas getrennt, *können* Optionen für das Klassenlayout angegeben werden

7.3.8 Pakete

Mit dem Befehl

`\usepackage[<optionen>]{<paket>}`

können in der Präambel ergänzende Makropakete (*packages*) geladen werden, die

- das Layout der Dokumentklasse modifizieren,
- oder zusätzliche Funktionalitäten bereitstellen.

In meinem Ubuntu-System habe ich sämtliche Pakete installiert via

`$ sudo apt-get install texlive-full.`

Ein Teil der Makropakete finde ich bei mir unter

`/usr/share/texlive/texmf-dist/tex/latex/`

Der Befehl

`\usepackage{amsmath}`

lädt dann das Makropaket `amsmath`, dessen `.sty` Datei unter

`/usr/share/texlive/texmf-dist/tex/latex/amsmath/amsmath.sty`

zu finden ist. In der Tabelle 3 finden wir weitere häufig verwendete Pakete.

<code>amsmath, amssymb</code>	Mathematischer Formelsatz mit erweiterten Fähigkeiten, zusätzliche mathematische Schriften und Symbole,...
<code>babel</code>	Anpassungen für viele verschiedene Sprachen. Die gewählten Sprachen werden als Optionen angegeben <code>\usepackage[ngerman]{babel}</code>
<code>fontenc</code>	font encoding: Erlaubt die Verwendung von Schriften mit unterschiedlicher Kodierung (Zeichenvorrat, Anordnung) <code>\usepackage[T1]{fontenc}</code>
<code>geometry</code>	Manipulation des Seitenlayouts.
<code>graphicx</code>	Einbindung von extern erzeugten Graphiken
<code>hyperref</code>	Ermöglicht Hyperlinks zwischen Textstellen und zu externen Dokumenten
<code>inputenc</code>	input encoding: Eingabekodierung <code>\usepackage[utf8]{inputenc}</code> (UTF-8 als Eingabekodierung)
<code>listings</code>	Zur Code Darstellung (Syntax-Highlighting kann festgelegt werden)

Tabelle 3: Eine kleine Auswahl häufig verwendeteter Pakete

\LaTeX Dokumentation lesen

7.4 Setzen von Text

<https://dante-ev.github.io/l2kurz/l2kurz.pdf#section.3>

7.5 Setzen von mathematischen Formeln

<https://dante-ev.github.io/l2kurz/l2kurz.pdf#section.4>

7.6 Setzen von Bildern

<https://dante-ev.github.io/l2kurz/l2kurz.pdf#section.5>

7.7 Seitenaufbau

<https://dante-ev.github.io/l2kurz/l2kurz.pdf#section.6>

7.8 Literatur

Zur Literaturverwaltung brauchen wir eine Datenbank und einen Mechanismus, der diese Datenbank verarbeiten kann.

7.8.1 Datenbank: Textdatei literature.bib

- In einer Textdatei mit Endung .bib können wir alle Quellen sammeln, die wir für ein bestimmtes Projekt zitieren möchten.
- Die Einträge müssen dabei mit einer bestimmten **Syntax** getätigter werden, siehe Listing 1. Welche Einträge möglich sind können wir beispielsweise auf https://de.overleaf.com/learn/latex/Bibliography_management_with_biblatex#Reference_guide nachlesen.
- Für die "automatisierte" Verwaltung von Literaturdatenbanken existieren auch Computerprogramme mit GUI.

```
1 % Latex
2 @book{latex1,
3   author="J. Knappen",
4   title="Schnell ans Ziel mit Latex 2e",
5   doi="https://www.degruyter.com/document/doi/10.1524/9783486850468/html",
6   year="2009",
7   publisher="De Gruyter",
8   address="Muenchen"
9 }
10 % Artikel
11 @article{einstein,
12   author      = "Albert Einstein",
13   title       = "Zur Elektrodynamik bewegter K\"orper",
14   journal     = "Annalen der Physik",
15   volume      = "322",
16   number      = "10",
17   pages       = "891--921",
18   year        = "1905",
19   DOI         = "http://dx.doi.org/10.1002/andp.19053221004"
20 }
21 % Sonstiges
22 @misc{knuthwebsite,
23   author      = "Donald Knuth",
24   title       = "Knuth: Computers and Typesetting",
```

```

25     url      = "http://www-cs-faculty.stanford.edu/~{}uno/abcde.html"
26 }

```

Listing 1: Beispiel Einträge in der .bib-Datei

Demo:

Oft kann man die .bib-Einträge beim Verlag (oder <https://tricat.uni-trier.de/>) exportieren (suchen nach "Cite this document") und muss diese nicht manuell eintragen.

Beispiel: <https://www.degruyter.com/document/doi/10.1524/9783486850468/html>

7.8.2 Verarbeitung: BibTEX und biblatex

- **BibTEX:** muss i.d.R nicht extra geladen werden

- klassische Variante in Verbindung mit einem Literaturverzeichnisstil
→ eine gute Übersicht zu den Stilen bietet:

https://de.overleaf.com/learn/latex/bibtex_bibliography_styles

- Anpassung an die eigenen Bedürfnisse schwierig

Im Dokumentkörper:

```

\cite{<bibfile-nametag>}
\bibliographystyle{<style>} % style ∈ {plain,...}, meist am Ende
\bibliography{<mybibfile>.bib}

```

- **biblatex:** wird als Makropaket geladen

- alternativ zu BibTEX kann dieses Makropaket das mächtigere Programm *biber* nutzen
- erlaubt die Manipulation des Literaturverzeichnisses auf LATEX-Ebene

– Vorteile gegenüber BibTEX:

- * man kann den Literaturverzeichnisstil leicht anpassen
- * kann UTF-8
- * man ist dadurch flexibel hinsichtlich "style guides" (Stilvorgaben des Verlags, Zeitschrift, Betreuer,...)

- siehe auch die Dokumentation von overleaf:

https://www.overleaf.com/learn/latex/bibliography_management_with_biblatex

In der Präambel

```

\usepackage[style=<somebiblatexstyle>,<other options>]{biblatex}
\addbibresource[options for bib resources]{<mybibfile>.bib}

```

Im Dokumentkörper:

```

\cite{<bibfile-nametag>}
\printbibliography[options for printing]{}

```

- es gibt auch noch das Makropaket **natbib**

7.9 Modularisierung und Maßschneiderung

Siehe hierzu auch

- das Kapitel "Planung eines LATEX-Projektes" in [4, Kapitel 11]
- https://de.overleaf.com/learn/latex/Management_in_a_large_project
- https://en.wikibooks.org/wiki/LaTeX/Modular_Documents

7.9.1 Aufteilung des Textes auf mehrere Dateien: `\input`, `\include`, `\usepackage`

- Der Befehl `\input{InputFile.tex}` fügt die Datei InputFile.tex an einer bestimmten Stelle des Quelltextes ein.
- Die eingefügte Datei wird als normaler Teil des Quelltextes mitverarbeitet und sollte daher ebenfalls aus einer Folge von LaTeX-Befehlen bestehen, jedoch ohne Präambel und ohne `\begin{document}`, `\end{document}`.
- Verschachtlung von mehreren `\input`-Befehlen ist möglich

Beispielsweise können wir das Projekt wie folgt aufteilen:

- **main**: Haupteingabedatei, auf die LATEX angewendet wird und alle Komponenten "zusammenklebt"
- **meta**: Metadaten wie Autorename, Datum, Titel,...
- **abstract**: Kurze Zusammenfassung des Texts
- **usepackages**: Laden aller benötigter (externer oder eigener) Makropakete
- **style**: Eigene Farben, Theorem-Umgebungen anpassen,...
(→ könnte man ggf. zu **usepackages** packen)
- **commands**: Eigene Befehle (Makros)
- **content**: Beispielsweise für jedes Kapitel eine eigene .tex-Eingabedatei

7.9.2 Definition eigener Makros: Der Befehl `\newcommand`, `\newenvironment`

Mit dem Befehl

```
\newcommand{<Befehlsname>}[<Anzahl>]{<Definition>}
```

können wir eigene Befehle/Makros definieren. Es sind maximal 9 Argumente möglich und diese können innerhalb der Befehlsdefinition mit #1 bis #9 benutzt werden.

Das ist aus vielen Gründen sinnvoll, z.B.,

- **Abkürzungen**: Häufig verwendete Makros wie³

```
\mathbb{R}^n
```

für \mathbb{R}^n , könnte man abkürzen durch

```
\newcommand{\Rn}{\mathbb{R}^n}
```

dann können wir mit `\Rn` das Symbol \mathbb{R}^n setzen.

- **Wichtige Variablennamen**: Variablennamen sollten stets durchdacht gewählt werden, sodass der Leser schon durch den Namen intuitiv das Objekt einschätzen kann. Zum Beispiel ist es typisch für Matrizen Großbuchstaben zu verwenden, etwa `S` für symmetrische Matrizen, `O` für orthogonale Matrizen,... Möchte man am Ende allerdings `Q` anstatt `O` für orthogonale Matrizen verwenden, so müsste man per search+replace mühsam die Stellen finden, an denen der Buchstabe "O" tatsächlich nur als Variablenname für orthogonale Matrizen auftaucht. Sinnvoll ist es daher wichtige (oder gleich alle) Variablennamen vorab als Befehl zu definieren, z.B.,

```
\newcommand{\oMat}{O}
```

Möchte wir nun aus "O" ein "Q" machen, so müssen wir dies nur an einer einzigen Stelle tun!

³Der Befehl `\mathbb` ist im Makropaket `amssymb` enthalten.

7.10 Ausgewählte Beispiele und Tipps

7.10.1 Allgemein

- **Kapitelgliederung**
section, subsection,...
- **Referenzen**
(eq)ref, label, gute Namen: fig-, eq-, tab-
- **Tabellen**
table, tabular
- **Grafiken**
figure, includegraphics, insb width, trim, height,.
- **Beschreibung anpassen:** `\usepackage [font=small, labelfont=bf]{caption}`
captionsetupwidth=0.8 linewidth
- **Schriftstil**
textbf, texttt, textit, text
- **Code**
listing, algorithm
- **Aufzählungen**
itemize, enumerate
- **Hyperlinks**
url, hyperref
- **Verzeichnisse:**
Inhalt tableofcontent, Abb. listoffigures, Tabellen listoftables
- **Fußnote:** `\footnote{<Text>}`
- **Leerstellen:**
vspace, hspace*,...

7.10.2 Mathematik

LaTeX trennt das Setzen von normalem Text und mathematischen Formeln mithilfe von zwei Modi: *paragraph* und *math mode*

- **math mode:**
\$ und \$\$, \[, \]
- **Gleichungen:** `\usepackage{amsmath}`
equation(*), align(*),
- **Symbole:** `\usepackage{amssymb}`
griechische Buchstaben μ, η, \mathbb{R}
- **Summen/Produkt/Integral-Zeichen**
sum, prod, int,... $_$ und $^$
- **Quantoren**
forall, exist
- **Theoreme:** `\usepackage{amsthm}`
theorem, lemma,...
- **Mehrdimensionale Arrays:**
pmatrix, array,...

7.10.3 Arbeiten im Editor

- pdf preview: strg+enter
- input » Strg+Enter
- shortcuts: auskommentieren strg+u/t, fett strg+b, kursiv strg+i,...
- environments: Doppelklick und Name ändern
- Tab–Completion: strg+tab (Umgebungsname, Befehlsname,...)
- strg+Enter für \\
- copy-paste Dateien: pdf, png,...
- korrekte Anführungszeichen “ ”

7.10.4 Grafiken erstellen

inkscape

Hochwertige Vektorgrafiken können mit der freien Software **inkscape** erzeugt werden. Zudem bietet inkscape einen geeigneten \LaTeX -Export an, sodass man mathematische Formeln im *math mode* verwenden kann und die Grafik leicht in sein \LaTeX -Dokument einbinden kann.

7.11 Sonstige Dokumentklassen

7.11.1 Folienpräsentation erstellen

```
\documentclass{beamer}
```

7.11.2 Poster erstellen

```
\documentclass[tikzposter]{}
```

7.11.3 Lebenslauf

```
\documentclass[moderncv]{}
```

Literatur

- [1] H.-P. Gumm and M. Sommer. *Einführung in die Informatik*. Oldenbourg Wissenschaftsverlag GmbH, München, 2013.
- [2] H.-P. Gumm and M. Sommer. *Informatik – Band 1 Programmierung, Algorithmen und Datenstrukturen*. De Gruyter, Oldenbourg, 2016.
- [3] H.-P. Gumm and M. Sommer. *Informatik – Band 2 Rechnerarchitektur, Betriebssysteme, Rechnernetze*. De Gruyter, Oldenbourg, 2017.
- [4] J. Knappen. *Schnell ans Ziel mit Latex 2e*. De Gruyter, München, 2009.
- [5] A. Meister. *Numerik linearer Gleichungssysteme*. Springer Spektrum, Wiesbaden, 2015.

A Grundlagen der Informatik

In diesem Kapitel verschaffen wir uns einen groben Überblick zu grundlegenden Begrifflichkeiten aus der Informatik:

- Hardwarekomponenten und Bootvorgang.
- Betriebssystem: Wie wird die Hardware eines Computers dem Benutzer zur Anwendung nutzbar gemacht?
- Funktionsweise von Computernetzwerken, IPv4 und Konzepte wie das Client-Server-Modell.
- Virtualisierung: Virtuelle Maschinen, Container, virtuelle Netzwerke (VPN).

Die Inhalte sind fast baugleich entnommen aus den Lehrbüchern [1] und [3], dem Lernmaterial des Linux Professional Institute und teilweise Wikipedia.

A.1 Hardwarekomponenten

- Aus Benutzersicht präsentiert sich der Computer (Stand-PC, Laptop, Tablet, Smartphone, Fahrradcomputer,...) über die Anwendungssoftware (*apps*), die wir auf ihm ausführen und verwenden.
- Hardware verarbeitet die von der Software beschriebenen Befehle und stellt Mechanismen für Speicherung sowie Eingabe und Ausgabe bereit.
- Damit überhaupt eine Software auf einer bestimmten Hardware läuft, müssen geeignete Schnittstellen festgelegt werden; siehe dazu auch Abb. 7:



Abbildung 7: Zwischen diesen Schichten müssen geeignete Schnittstellen definiert und implementiert werden. Quelle: [3, Abb. 1.25]

Wir schauen uns einige Hardwarekomponenten als eigenständige physische Geräte und deren Standardschnittstellen genauer an. Die Standards sind relativ statisch, aber Form (Größe, Anschlüsse,...), Leistung (Zugriffsgeschwindigkeiten, Taktfrequenz,...) und Kapazität (Speichergröße,...) der Hardware entwickeln sich ständig weiter.

A.1.1 Netzteile

- Alle aktiven Komponenten in einem Computersystem benötigen zum Betrieb Strom.
- Netzteile normalisieren verfügbare Energiequellen.
- Standardisierte Spannungsanforderungen ermöglichen es Herstellern, Hardwarekomponenten zu entwickeln, die auf vielen Systemen laufen.
- Bei der Nutzung von Strom entsteht Wärme, die abgeführt wird durch:
 - Ventilator (erzeugt Luftstrom, am Netzteil und anderen Komponenten)
 - Kühlkörper (Rippenstruktur) mit guter Wärmeleitfähigkeit zur Vergrößerung der Oberfläche (Wärmeleitung an die Luft, welche dann über konvektiven Einfluss des Ventilators abgeführt werden kann)



Quelle: <https://de.wikipedia.org/wiki/PC-Netzteil>

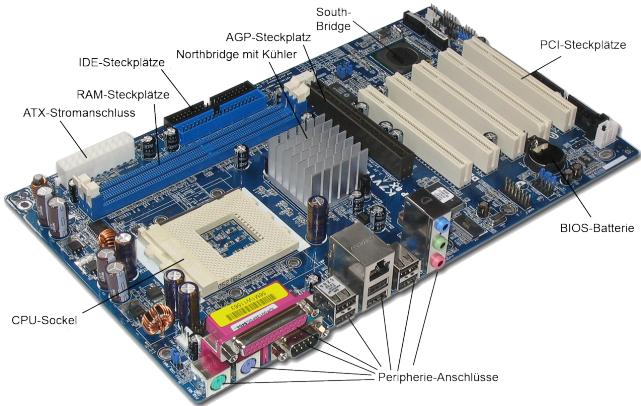


Abbildung 8: Hauptplatine. Quelle: <https://de.wikipedia.org/wiki/Hauptplatine>

A.1.2 Hauptplatine (Motherboard) und Bootvorgang

- Sämtliche Hardware eines Systems muss miteinander verbunden sein. Die Hauptplatine (auch *Motherboard* oder *Mainboard* genannt) normiert diese Verbindung mit standardisierten Steckverbindungen und Formfaktoren.
- Wird die Stromversorgung eingeschaltet, muss die mainboardspezifische Hardware konfiguriert und initialisiert werden, bevor das System laufen kann. Motherboards laden dazu Programme aus einem nichtflüchtigen Speicher (=bleibt erhalten unabhängig von Stromzufuhr), die als **Firmware** oder **Basis–Software** ([3, 2.1]) bezeichnet werden. Dabei handelt es sich noch nicht um Teile eines Betriebssystems.
- Zunächst werden u.a. die Komponenten des Rechners (CPU, Peripheriegeräte,...) getestet. Sind keine Fehler aufgetreten, wird von der Festplatte oder einem anderen Laufwerk das Betriebssystem geladen; man spricht bei diesem Vorgang von **booten**⁴.

In diesem Zusammenhang gibt es zwei wichtige Begriffe:

- **BIOS**

- Die ursprüngliche Form der Motherboard-Firmware wurde als BIOS (Basic Input/Output System) bezeichnet.
- Aufgaben: Grundlegende Konfigurationseinstellungen sowie die Identifizierung, das Laden und die Übertragung des Betriebs auf ein Betriebssystem wie Linux.

- **UEFI**

- Im Laufe der rasanten Hardwareentwicklung wurde auch die Firmware erweitert: Um größere Festplatten, Diagnosewerkzeuge, grafische Oberflächen, Netzwerke, usw.
- Mit dem Ziel die Schnittstelle zwischen Platinen–Firmware und Betriebssystem zu standardisieren wurde das Unified Extensible Firmware Interface (kurz UEFI, englisch für Vereinheitlichte erweiterbare Firmware–Schnittstelle) entwickelt (angeregt von Intel Ende der 90er mit EFI und vereinheitlicht zu UEFI von einem Konsortium von PC-Herstellern in 2006).
- UEFI hat sich als Nachfolger des BIOS etabliert und wird heute von den meisten Motherboards verwendet.

Unabhängig davon bezeichnen viele die Firmware des Motherboards immer noch als BIOS.

Der Bootvorgang (mit UEFI)

1. UEFI-kompatible Firmware des Motherboards (oft einfach als *UEFI* bezeichnet) wird ausgeführt. Je nach Hersteller kann dabei ein optionales **Einstellungsmenü (Bootmenü)** durch Drücken einer (F–)Funktionstasten eingeblendet werden. In der Praxis meist von Nöten, wenn man einen Rechner „neu aufsetzen“ oder inspizieren möchte und dazu die Bootreihenfolge ändern muss (um ein Betriebssystem-Image von einem USB-Stick zu laden).

⁴Da ein Computer während des Bootvorgangs schon ein Programm lädt, das er zum Funktionieren benötigt, zieht er sich bildlich gesprochen wie Münchhausen an den eigenen Haaren aus dem Sumpf; die englischen Redewendung dazu: an seinen Stiefelriemen (engl. bootstraps) selbst über einen Zaun.

2. Dann wird nach einem UEFI-kompatiblen **Bootloader** gesucht. Ein Bootloader (manchmal auch **Bootmanager** genannt) ist eine spezielle Software, die gewöhnlich durch die Firmware des Motherboards von einem wechselbaren veränderlichen Datenspeicher geladen und anschließend ausgeführt wird. Der Bootloader ist meist das erste Programm, das nach der unveränderlichen Firmware von einem wechselbaren veränderlichen Datenspeicher geladen wird.

Beispiel für Linux und weitere unixoide Betriebssysteme:

Grand Unified Bootloader (kurz **GRUB**, *Großer vereinheitlichter Bootloader*), aktuell **GRUB2**.

3. Der Bootloader lädt dann weitere Teile des Betriebssystems, gewöhnlich einen Kernel. Es können mehrere Betriebssysteme zur Verfügung stehen. In diesem Fall erscheint meist ein Prompt⁵, welches den Benutzer auffordert eine Auswahl zu treffen. Damit können erweiterte Funktionen und unterschiedliche Startmodi realisiert werden.

⁵<https://de.wikipedia.org/wiki/Prompt>: Als englisch prompt wird in der IT eine Aufforderung an den Benutzer bezeichnet, eine Eingabe (input) zu tätigen.

Demo:

```
UEFI firmware (Motherboard) [Einstellungsmenü]
> boot/efi/<distro>/grubx64.efi (Datenspeicher) [Bootmenü mit Kernelauswahl]
> kernel
```

Unter Linux (hier Ubuntu):

- Der einfachste Weg herauszufinden, ob UEFI oder BIOS verwendet wird, ist nachzuschauen, ob das Verzeichnis

```
/sys/firmware/efi
```

vorhanden ist.

- Auf modernen Systemen befindet sich der Bootloader dann auf der EFI (Extensible Firmware Interface) system partition (ESP), welche als

```
/boot/efi
```

eingehangen wird; siehe zum Beispiel

```
$ lsblk.
```

- Die UEFI Firmware des Motherboard startet dann die Datei

```
boot/efi/<distro>/grubx64.efi (für x64 UEFI System).
```

- Die Konfiguration des GRUB finden wir in der Datei

```
/boot/grub/grub.cfg
```

wo ein Eintrag beispielsweise den Kernel aufruft:

```
linux /boot/vmlinuz-5.11.0-44-generic root=UUID=(...) ro $vt_handoff
```

Die Datei wird automatisch erzeugt mit den Einstellungen aus

```
/etc/default/grub,
```

mit Einträgen wie

```
GRUB_CMDLINE_LINUX_DEFAULT='quiet splash',
```

die wir besser zu

```
GRUB_CMDLINE_LINUX_DEFAULT='',
```

setzen, um einen gesprächigen Bootvorgang zu beobachten.

- Im Verzeichnis /boot finden wir den Link auf den Kernel

```
vmlinuz-5.11.0-44-generic
```

A.1.3 Arbeitsspeicher (RAM)

- Der Arbeitsspeicher enthält die Daten und den Programmcode der aktuell laufenden Anwendungen.
- Physisch gesehen wird der Arbeitsspeicher in der Regel auf einzelnen Platinenmodulen aufgebracht, die in das Motherboard eingesteckt werden.
- 4 GB ist für die meisten Standardanwendungen der minimale Arbeitsspeicher.
- Swap Space: Was geschieht, wenn eine Anwendung mehr als den verfügbaren Arbeitsspeicher benötigt? In diesem Fall verschiebt Linux ungenutzte Anwendungen aus dem Arbeitsspeicher in einen speziellen Plattenbereich, den Swap Space, und ungenutzte Anwendungen aus dem Swap Space zurück in den Arbeitsspeicher, wenn sie ausgeführt werden müssen.

Demo: Unter Linux: Wie viel Speicherplatz steht zur Verfügung?

- Für einen Benutzer ist üblicherweise die Gesamtmenge des verfügbaren und des verwendeten Speichers von Interesse. Eine Informationsquelle ist die Ausführung des Befehls

```
$ free
```

mit dem Parameter -m (oder -g) zur Ausgabe der Werte in Megabytes (oder Gigabytes).

A.1.4 Prozessor und CPU



Abbildung 9: Rainer Knäpper, Free Art License
(<http://artlibre.org/licence/lal/en/>), https://commons.wikimedia.org/wiki/File:Intel_core_i7-975_bottom_R7309730_wp.jpg

- Das Wort "Prozessor" impliziert, dass etwas verarbeitet wird. In Computern geht es hauptsächlich um die Verarbeitung elektrischer Signale, die typischerweise so behandelt werden, dass sie einen der Binärwerte 1 oder 0 haben.
- Häufig werden der Begriff "Prozessor" und die Abkürzung CPU (Central Processing Unit) synonym gebraucht, was technisch nicht korrekt ist. Moderne Computer haben neben einer CPU oft auch weitere, aufgabenspezifische Prozessoren:
 - FPU (Floating Point Unit)
 - GPU (Graphical Processing Unit)
 - TPU (Tensor Processing Unit) von Google eigens entwickelt für TensorFlow
- Die CPU ist also ein Prozessor, aber nicht alle Prozessoren sind CPUs.

Die Kernmarkmale:

- **Befehlssatzarchitektur:**

- Definiert bestimmte Befehlssätze auf Chip-Ebene, die sogenannten Maschinenbefehle. Dazu gehören zum Beispiel arithmetische, logische oder Transfer-Operationen.
- Obwohl Intel und AMD Prozessoren herstellen, die dieselben Anweisungen unterstützen, ist es sinnvoll, nach Anbietern zu unterscheiden, da sie sich herstellerseitig in Bauart, Leistung und Stromverbrauch unterscheiden.
- **Beispiele**
 - * x86-Befehlssatzarchitektur
Verweist typischerweise auf die 32-Bit-Befehlssätze der 80386-Nachfolger, nach den Prozessoren der Intel 8086/8080-Reihe benannt.
 - * x64 (auch x86-64 genannt)
Referenzprozessoren, die sowohl die 32-Bit- als auch die 64-Bit-Anweisungen der x86-Familie unterstützen (rückwärtskompatibel).

- * AMD
Verweist auf die x86-Unterstützung durch AMD-Prozessoren.
- * AMD64, Intel64
Verweist auf die x64-Unterstützung durch AMD-(Intel-)Prozessoren.

- **Verarbeitungsbreite/Bitgröße (Bit size):**

Bei CPUs bezieht sich diese Zahl sowohl auf die native Größe der von ihr verarbeiteten Daten als auch auf die Menge an Speicher, auf die sie zugreifen kann. Die meisten modernen Systeme sind entweder 32-Bit oder 64-Bit. Grob: Die CPU arbeitet in (32) 64Bit-Blöcken.

- **Taktfrequenz (clock speed) [MHz/GHz]:**

- Sagt aus, wie schnell ein Prozessor Anweisungen verarbeitet.
- Prozessorgeschwindigkeit ist nur einer der Faktoren, die Systemreaktionszeiten, Wartezeiten und den Durchsatz beeinflussen.
- Da man bei Standard Office-Nutzern eines Rechners nur einen Bruchteil der Rechenleistung einer CPU verwendet, lohnt sich eine höhere Taktfrequenz lediglich bei rechenintensiven Anwendungen, welche die CPU auch 100% auslasten.

- **Cache:**

- Die Kosten und der Stromverbrauch eines Arbeitsspeichers mit mehreren Gigabyte, auf den mit CPU-Taktgeschwindigkeiten zugegriffen werden kann, sind unerschwinglich.
- Der CPU-Cache-Speicher ist auf dem CPU-Chip integriert, um einen schnellen Puffer zwischen CPUs und Arbeitsspeicher bereitzustellen. Der Cache ist in mehrere Schichten unterteilt, die üblicherweise als L1, L2, L3 und sogar L4 bezeichnet werden. In Fall von Cache gilt: Je mehr, desto besser.

- **Kerne (Cores):**

- "Kern/Core" bezieht sich auf eine einzelne CPU. Zusätzlich zum Core, der eine physische CPU darstellt, ermöglicht Hyper-Threading Technology (HTT) einer einzelnen physischen CPU, mehrere Anweisungen gleichzeitig zu verarbeiten, so dass sie praktisch als mehrere physische CPUs fungiert.
- Typischerweise werden mehrere physische Kerne als ein einziger physischer Prozessorchip verbaut.
- Auch hier: Standard Office-Anwendungen beanspruchen einen einzelnen Kern kaum. Daher bringt es für Standardanwendungen nicht viel weitere unbeschäftigte Kerne hinzufügen.
- Mehr Kerne führen nur dann zur Verbesserung des Durchsatzes wenn sie für die Ausführung von Anwendungen, die so geschrieben sind, dass sie mehrere unabhängige Funktionsabläufe aufweisen, wie z.B. Video-Frame-Rendering, Webseiten-Rendering oder parallele numerische Verfahren (HPC: High Performance Computing) mit mehreren Benutzern.

Demo:

Unter Linux:

Die Datei

`/proc/cpuinfo`

enthält detaillierte Informationen über den oder die Prozessoren eines Systems. Leider sind diese Details nicht allgemeinverständlich. Ein übersichtlicheres Ergebnis liefert der Befehl

`$ lscpu.`

A.1.5 Speicher

- Speichergeräte dienen der Aufbewahrung von Programmen und Daten.
- Festplattenlaufwerk *Hard Disk Drives* (HDDs) und Halbleiterlaufwerk *Solid State Drives* (SSDs) sind die häufigste Form von Speichergeräten in Servern und Desktops. USB-Speichersticks oder SD-Karten werden selten als primäre Speichermedien verwendet (zu unzuverlässig).



Hard Disk Drive (HDD)
Quelle: Eric Gaba, Wikimedia Commons user, <https://de.wikipedia.org/wiki/Festplattenlaufwerk>

- **HDD** (siehe Video):
 - Ein Festplattenlaufwerk speichert Informationen auf einer oder mehreren starren physischen Platten, die mit magnetischen Materialien bedeckt sind, um die Speicherung zu ermöglichen.
 - Die Platten befinden sich in einem abgedichteten Gehäuse, da Staub oder kleine Partikel die Lese/schreib-Fähigkeit der HDD beeinträchtigen würden.

- **SSD:**
 - SSDs sind deutlich anspruchsvollere Varianten von USB-Sticks mit wesentlich größerer Kapazität.
 - Sie speichern Informationen in Mikrochips, so dass es keine beweglichen Bauteile gibt.

Vergleichsparameter

- *Kosten:*
SSD bis zu 3-10 mal so teuer pro GB
- *Zugriffsgeschwindigkeit* (lesen/schreiben): SSD schneller als HDD, denn
HDD: Zum Lesen oder Schreiben muss sich eine bestimmte Stelle einer Festplatte an einen bestimmten Ort drehen.
SSD: Erlauben Direktzugriff.
- *Speicherapazität:* 5TB für HDD und 1TB SSD heutzutage üblich

Anschlüsse:

- SCSI (Small Computer System Interface)
- SATA (Serial AT Attachment)
- USB

Diese Interfaces werden typischerweise durch entsprechende Stecker auf der Hauptplatine unterstützt.

In den Firmware(UEFI/BIOS)-Einstellungen (Bootmenü) können wir die Reihenfolge festlegen, in der beim ersten Laden auf die Geräte zugegriffen wird (wie oben erwähnt: Notwendig, wenn man von einem anderem Speichermedium booten möchte.)

Demo:

Speichergeräte lesen und schreiben üblicherweise Daten in Blöcken von Bytes; man spricht daher auch von **Blockgeräten**.

Unter Linux:

Mit dem Befehl

```
$ lsblk # "list block devices"
```

können Sie die einem System zur Verfügung stehenden Blockdevices auflisten.

Partitionen

- Ein Speichergerät ist praktisch eine lange Folge von Speicherplätzen. Partitionierung ist der Mechanismus, mit dem man diese Folge in mehrere unabhängige Sequenzen zerlegen kann. Jede Partition wird wie ein einzelnes Gerät behandelt.
- Vorteile:
 - Verwaltung des verfügbaren Speichers oder die Unterstützung mehrerer **Dateisysteme** (ext4, vFAT, NTFS,...).
 - Multi-System: Partitionen ermöglichen es, ein einziges Speichergerät zu haben, das unter verschiedenen Betriebssystemen booten kann
- **Formatierung:**
 - Zwar erkennt das Betriebssystem (wie Linux) die Speichersequenz eines Raw-Device, aber ein Raw-Device kann nicht "einfach so" verwendet werden.
 - Um ein Raw-Device zu nutzen, muss es formatiert, d.h., ein Dateisystem muss auf das Gerät geschrieben werden.
 - Dann erst sind Dateioperationen ausgeführt werden.

A.1.6 Peripheriegeräte

- Ein Computersystem braucht lediglich eine Kombination aus CPU, Arbeitsspeicher und Speicher.
 - Aber diese grundlegenden Komponenten interagieren nicht direkt mit der Außenwelt.
 - Peripheriegeräte sind die Geräte, die den Systemen Input, Output und Zugriff auf den Rest der realen Welt ermöglichen.
- Beispiele: Tastatur, Maus, Sound, Video und Netzwerk

A.1.7 (Geräte-)Treiber

Wenn eine CPU mit den Endgeräten (z.B. den Laufwerken, Drucker,...) verschiedenster Hersteller zusammenarbeiten soll, dann muss man sich zunächst auf eine gemeinsame Schnittstelle verstündigen: Eine Konvention, die eine Verbindung verschiedener Bauteile festlegt.

Beispielproblem:

- Ein Textverarbeitungsprogramm kann nicht im Voraus alle verschiedenen Drucker kennen, die es einmal bedienen soll.
- Wenn der Benutzer den Befehl "Drucken" auswählt, soll der Druck funktionieren, egal welcher Drucker angegeschlossen ist.
- Dazu bedient sich das Programm einer Schnittstelle, die im Betriebssystem definiert wird; genauer installiert man ein Programm (Treiber), das der Druckerhersteller für das entsprechende Betriebssystem mitliefert und welches die Druckbefehle in Signale für den Drucker umsetzt.

Allgemein

- Gerätetreiber (*Device Driver* oder kurz *Driver*) akzeptieren einen Standardsatz von Anfragen und übersetzen diese dann in die entsprechenden Steuerungsaktivitäten des Geräts.
- Ein Treiber ermöglicht einem Anwendungsprogramm die Benutzung einer Komponente, **ohne** deren detaillierten Aufbau zu kennen.
- Die Anforderungen eines Anwendungsprogramms an das zugehörige Gerät werden dann vom Betriebssystem an den entsprechenden Treiber umgeleitet, dieser wiederum sorgt für die korrekte Ansteuerung des Druckers.

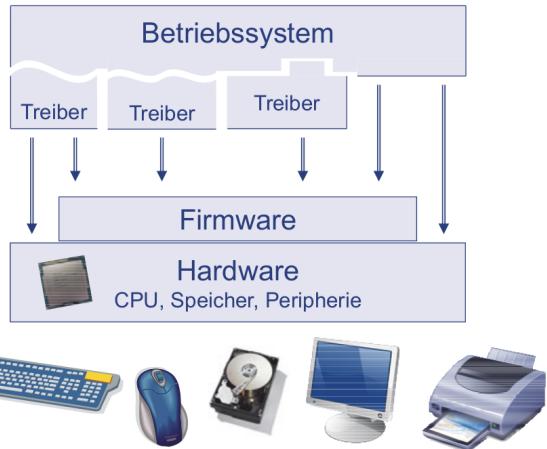


Abbildung 10: Hardware, Treiber und Betriebssystem; Quelle: [2, Abb. 1.27]

Anderes Beispiel: Temperatur- und Luftdrucksensor für einen Raspberry Pi. Der Betriebssystemkernel braucht entsprechende Kerneltreiber, um diesen auszulesen und die Daten dem Benutzer über eine geeignete Schnittstelle zur Verfügung zu stellen (zum Beispiel Textdateien in einem designierten Verzeichnis, das ausgelesen werden kann, um dann in Python damit zu arbeiten).

Demo:

Unter Linux:

- \$ lshw -short
- \$ lscpu
- \$ hwinfo --short
- Für weitere Befehle unter Linux zur Ermittlung von Hardware Informationen siehe zum Beispiel: <https://www.binarytides.com/linux-commands-hardware-info/>

Hausaufgaben

1. Welche Firmware (UEFI oder BIOS) wird auf Ihrem System verwendet?
2. Wo liegt der Bootloader?
3. Finden Sie heraus, welcher Prozessor in Ihrem Rechner verbaut ist. Um welche Architektur handelt es sich?
4. Wie viel GB Arbeitsspeicher (RAM) hat Ihr Rechner?
5. Welche Festplatten sind eingehangen?

A.2 Betriebssysteme

(siehe auch [3, Kap. 2], [1, Kap. 6])

Zugeschnitten auf die vorhandene Prozessor-Architektur werden Betriebssysteme (genauer Betriebssystemkerne) entwickelt, die Hardware-Ressourcen des Rechners (Speicher, Prozessorleistung, externe Geräte) dem Benutzer zur Verfügung stellen; siehe auch Abb. 7. Ein Betriebssystem ist damit letztlich ein Programm, das dem Benutzer (*user*) und den Anwendungsprogrammen (*app*) grundlegende Dienste bereitstellt. Diese Dienste machen das aus, was der Rechner in den Augen eines Nutzers kann.

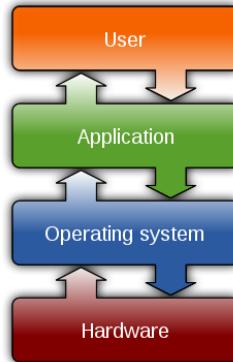


Abbildung 11: Quelle: <https://de.wikipedia.org/wiki/Betriebssystem>

Beispiele

Die meisten Rechner werden zusammen mit einem bereits vorinstallierten Betriebssystem verkauft.

- Auf PCs dominiert immer noch **Microsoft Windows** (aktuell (2022) mit Version Windows 11)
- Gefolgt von **macOS** (exklusiv auf den Rechnern der Firma Apple).
- **Linux** (ein freies, quelloffenes Betriebssystem) dominiert auf Servern und im wissenschaftlichen Bereich.
- Auf kleineren Rechnern (Netbooks, Tablets oder Smartphones) ist das auf Linux basierende **Android** der Firma Google sehr beliebt.

Wir werden uns die folgenden Komponenten eines Betriebssystem näher anschauen und anhand von Linux praktisch kennenlernen:

- **Betriebssystemkern**
- **Dateisystem**
- **Mensch-Computer-Schnittstelle:**
 - CLI: **Kommandointerpreter** (Shell)
 - optionale GUI: **Desktopumgebung** (inkl. Dateimanager)

Demo:

- Der Dateiname eines Betriebssystem-Images beinhaltet häufig Information über die vorgesehene Prozessorarchitektur: z.B.

ubuntu-20.04.2.0-dekstop-amd64.iso.

<https://ubuntu.com/#download>

A.2.1 Betriebssystemkern (*kernel*)

Ein Betriebssystemkern, auch **Kernel** genannt, ist der zentrale Bestandteil eines Betriebssystems, auf dem alle weiteren Softwarebestandteile des Betriebssystems aufbauen. Er bildet somit die unterste Softwareschicht des Betriebssystems und hat direkten Zugriff auf die Hardware.

- Der Kernel ist in der Regel das erste Programm, dass nach dem Bootloader (siehe oben) gestartet wird.
- Er muss mehr Rechte besitzen, um notfalls ein „abgestürztes“ Programm zu beenden, Speicher wieder freizugeben oder den Zugriff auf eine Ressource zu verweigern. Um zu verhindern, dass Benutzerprogramme sich ähnliche Rechte anmaßen, muss der Prozessor verschiedene Privilegierungsstufen vorsehen. Die höchste Stufe steht nur dem Kern zu.
- Er hat völlige Kontrolle über das System.
- Befindet sich stets im Speicher (RAM).

Demo:

- Um herauszufinden welcher Linux-Kernel verwendet wird:

```
$ uname -r #(kernel release)
```

oder

```
$ uname -a #(all)
```

Output:

5 : Kernel version
8 : Major revision
0 : Minor revision
44 : Patch level
generic : Linux distro/kernel specific additional info

- Das GitHub Repository zum Kernel verwaltet vom Linux-Vater Linus Torvalds:

<https://github.com/torvalds/linux>

A.2.2 Dateisysteme (*file system*)

Unter den elementarsten Diensten eines jeden Betriebssystems befinden sich solche, die den Umgang mit **Dateien** (engl.: *files*) (=Folge von Bytes) organisieren.

- Dateien müssen erzeugt, verändert, abgespeichert, umbenannt und gelöscht werden
- In die Fülle der existierenden Dateien muss eine Ordnung gebracht werden
- Rechte: Der Zugang zu ihnen muss den anderen Benutzern des Rechners ermöglicht oder blockiert werden

Das Betriebssystem muss eine Übersetzung zwischen den von der Hardware angebotenen Blöcken von Bytes und den vom Benutzer gewünschten Dateien gewährleisten. Dazu verwendet es ein **Dateisystem**.

Dateien können zu einem **Verzeichnis** (engl. **directory**) zusammengefasst werden. Darin findet sich für jede Datei ein Eintrag mit kennzeichnenden Informationen (**Attribute** genannt) wie:

- Dateiname (ggf. auch die „Erweiterung“),
- Dateityp (Normaldatei, ausführbare Datei, Katalogdatei)
- Länge in Bytes,
- zugehörige Blöcke (meist reicht ein Verweis auf den ersten Block),
- Zugriffsrechte (Besitzer, ggf. Passwort, wer hat Lese- oder Schreibrechte)
- Datum (Erstellung, Änderung, evtl. Verfallsdatum).

Demo:

- Partitionen nochmal anschauen, Dateisystem?

```
$ lsblk -f      # ( | grep sd)
```

("-f": Output info about filesystems.)

A.2.3 Schnittstelle zwischen Mensch und Betriebssystem: CLI VS GUI

Wir klären zunächst folgende Begrifflichkeiten:

- **CLI:** *Command Line Interface*

Eine Kommandozeilen-Schnittstelle (*command-line interface*) (CLI) bezeichnet die Verwendung von Kommandozeilen (Textzeilen) für die Interaktion mit einem Benutzer. Das Programm, welches diese Schnittstelle verwaltet nennt man **Kommandozeileninterpreter** (z.B. Shell für Betriebssysteme).

- **GUI:** *Graphical User Interface*

Grafische Benutzeroberfläche oder auch grafische Benutzerschnittstelle bezeichnet eine Benutzerschnittstelle mittels grafischer Symbole und Steuerelemente (Widgets). Dies geschieht

- bei Computern: mittels einer Maus als Steuergerät,
- bei Smartphones, Tablets und Kiosksystemen: durch Berührung eines Sensorbildschirms.

CLI: Shell

Als **Shell** wird der Kommandozeileninterpreter bezeichnet, mittels dessen ein Benutzer über Kommandozeilen mit einem Betriebssystem interagiert. Die Shell (englisch für „Schale“ oder „Hülle“) ist anschaulich die Außenschicht des Betriebssystemkerns.

- Kommandozeilen waren historisch die ersten Methoden zur Interaktion mit Betriebssystemen und benötigen lediglich den Textmodus.
- Kommandozeileninterpreter sind bei modernen Betriebssystemen auch im Grafikmodus verfügbar, etwa als Terminal-emulation (siehe unten).

GUI: Desktop-Umgebung

Heute starten die meisten Betriebssysteme unmittelbar mit einer grafischen **Benutzeroberfläche** (**GUI:** Graphical User Interface), der sogenannten **Desktopumgebung**; die technische Umsetzung der sogenannten **Schreibtischmetapher**⁶:

- Der Rechner präsentiert grafisch einen Schreibtisch (desktop), auf dem Akten und Ordner (Dateien und Verzeichnisse) herumliegen. Diese Akten können geöffnet und verändert (edit), kopiert (copy) oder in einen Papierkorb geworfen werden (delete).
- Man kann die Objekte des Schreibtisches anfassen, verschieben oder aus Ordnern neue Akten herausholen.
- Erst den grafischen Betriebssystemoberflächen ist es zu verdanken, dass heute jeder einen Rechner irgendwie bedienen kann!
- Genau genommen handelt es sich bei den grafischen Oberflächen um Betriebssystemaufsätze
→ Es gibt Betriebssysteme meist in einer desktop und server version (mit oder ohne Desktop-Umgebung; siehe zB <https://ubuntu.com/#download>)
- Beispiel GNOME (GNU Network Object Model Environment) 3.36 ist Standard bei Ubuntu 20.04

⁶Die Idee der graphischen Benutzeroberfläche und des Desktops entstand in den 1970er bei Xerox im Palo Alto Research Center (Xerox PARC). Das Konzept von Xerox wurde von der damals noch kleinen Firma Apple übernommen. Nach einem anfänglichen Misserfolg mit dem System Apple Lisa trat der Macintosh seit den frühen 80er Jahren seinen Erfolgzug an.

Terminalemulator: "Shell in der GUI"

Eine Terminalemulation⁷ ist ein Computerprogramm, das die Funktion einer Shell innerhalb einer Desktopumgebung grafisch nachbildet, also ein Vermittler zwischen einer Textanwendung und einer grafischen Oberfläche.

Demo:

Beispiel UNIX: **Gnome Terminal** (A terminal emulator for the GNOME desktop)

- Gnome Terminal

(Navigatorische) Dateimanager (*file manager or browser*)

Ein Dateimanager (englisch File Manager) ist ein Computerprogramm zum Verwalten von Inhalten auf Dateisystemen, die sich auf unterschiedlichen Speichermedien befinden können. Navigatorische Dateimanager stellen die Inhalte eines beliebigen Verzeichnisses umschaltbar in einem Fenster dar.

Aufgaben:

- übersichtliche Darstellung in Form einer (oft grafischen) Benutzerschnittstelle
- Navigation (über Buttons; ähnlich wie im Webbrowser)
- Auflisten, das Umbenennen und Verschieben, das Kopieren und das Löschen von Dateien und Verzeichnissen zu den Grundfunktionen
- Bearbeitung von Metadaten unterstützter Dateisysteme, wie beispielsweise Dateiattribute, Dateiberechtigungen und Verknüpfung
- Netzwerkverbindungen via protocols, such as FTP, HTTP, NFS, SMB or WebDAV. How? browse for a file server (connecting and accessing the server's file system like a local file system) or by providing its own full client implementations for file server protocols.

Bekannte Beispiele:

- **Nautilus** ist ein freier **Dateimanager** für unixoide Systeme. Standard-Dateimanager der Desktop-Umgebung **Gnome**. Der Quelltext ist gemäß der GNU General Public License (GPL) frei verfügbar.
- Der **Windows-Explorer** (https://en.wikipedia.org/wiki/File_Explorer) bildet den voreingestellten Dateimanager in der Windows-Betriebssystemsfamilie seit Windows 95.
- Der **apple Finder** ist der Standard-Dateimanager des aktuellen Apple-Betriebssystems macOS (OS X, Mac OS X; ab 2001) und des klassischen Mac OS (1984–2001).

Demo:

- Features anhand von Nautilus demonstrieren.

⁷Als Emulator (von lateinisch *aemulari*, „nachahmen“) wird in der Computertechnik ein System bezeichnet, das ein anderes in bestimmten Teilespekten nachbildet.

Hausaufgaben

1. Welches Betriebssystem verwenden Sie? Welche Version? Wie viel kostet eine Lizenz für Ihr aktuelles Betriebssystem?
2. Welches Dateisystem befindet sich auf Ihren Partitionen?
3. Welche Desktopumgebung benutzen Sie?
4. Welchen Dateimanager benutzen Sie und wie können Sie darin die Attribute einer Datei abrufen?
5. Rufen Sie die Shell (bzw. dessen Emulation als Terminal) Ihres Betriebssystems auf:
 - Linux: Strg+Alt+T
 - Windows Shell cmd.exe
 - macOS Terminal.app

A.3 Netzwerke

(siehe auch [3, Kap. 3])

Rechner (Workstation, Compute-Server, Tablet, Laptop, Smartphone, Drucker, Fahrradcomputer,...) sind heute meist über ein Netzwerk verbunden und können über Protokolle miteinander kommunizieren.

A.3.1 Rechner-Verbindungen

Die Voraussetzung für die Vernetzung von Rechnern aller Art ist die *physikalische* Verbindung von Rechnern untereinander. Ist dieser Schritt erst einmal geschafft, kann man mehrere Rechner zu einem Netz zusammenfassen. Wir unterscheiden zwei verschiedene Arten:

- **wired**: Kabel (Ethernet, Glasfaser,...)
- **wireless**: "Luft" (WPA, Bluetooth, NFC,...)

A.3.2 Netze

Unter einem Rechnernetz versteht man eine Gruppe von Rechnern, die untereinander verbunden sind, um miteinander zu kommunizieren oder gemeinsame Ressourcen nutzen zu können.

Beispiele:

- **Lokales Netz (LAN = Local Area Network)**: dient zur Verbindung von Rechnern und Servern in einem räumlich begrenzten Gebiet (Ausdehnung von wenigen Kilometern) über Leitungen, für die nur der Betreiber (also nicht etwa die Telekom) verantwortlich ist.
- **Drahtloses lokales Netz (WLAN = Wireless Local Area Network)**: Kommunikation nicht über Leitungen sondern drahtlos.
- **Internet**: Spezielles *globales* Netz, das mittlerweile weltweit leitungsgebunden und drahtlos nutzbar ist.
- **Virtuelles privates Netz (VPN = Virtual Privat Network)**: verbindet eine Benutzergruppe innerhalb eines "unkontrollierten" Netzwerks (meist im Internet) durch ein spezielles Zugangs- und Übertragungsprotokoll derart, dass nur *berechtigte* Teilnehmer Zugang zu diesem VPN erhalten.

A.3.3 Netze von Netzen

Einerseits ist es in vielen Fällen nicht möglich, alle potentiellen Teilnehmer an ein einziges Netz anzuschließen. Andererseits wird eine direkte Kopplung über ein einziges Netz manchmal nicht gewünscht, z.B. um eine *Netzüberlastung* zu vermeiden oder aus Gründen der *Sicherheit* gegen Eindringlinge.

Beispiel: Von besonderem Interesse ist in den letzten Jahren der Anschluss an das weltweite Internet geworden (*Heimnetz* → *Internet*).

Die Verbindung zwischen Netzen erfolgt durch **Vermittlungsrechner (VR)**:

- **Switches** verbinden mehrere Subnetze (=Segmente eines LANs).
- **Router** (auch IP-Switch) haben dieselbe Funktion wie Switches und verbinden darüberhinaus Subnetze des Internets (typischerweise (*Heim*)Netz → *Internet*).
 - haben die Aufgabe der optimalen Paketvermittlung für das IP-Protokoll
 - bestimmen den nächsten Rechner im Internet (möglichst optimal), zu dem ein gegebenes IP-Paket auf seinem Weg zum endgültigen Ziel geschickt wird
→ Vorgang nennt man **routing**
- **Gateway** nennt man ganz allgemein einen Rechner, der die Verbindung zu einem bestimmten Netz (z.B. Internet) herstellt (meist ein Router).

A.3.4 Protokolle

Zu einer Kommunikation zwischen Rechnern gehört neben einer physikalischen Verbindung noch eine Vereinbarung über Art und Abfolge des Datenaustausches, ein sogenanntes **Kommunikationsprotokoll** (=Algorithmus).

Beispiel: Das Protokoll "https"

Sie können über Ihren Webbrowser den Server `math.uni-trier.de` via `https` kontaktieren, um den Inhalt einer Mathe-Webseite abzurufen.

Demo:

- `https://math.uni-trier.de`
- Alternativ: `136.199.235.16 (= host)`

Das **OSI-Modell** beschreibt in verschiedenen Schichten (unterteilt in *Transport* (Bitübertragung,...) und *Anwendung* (`https`, `smtp`,...)) die Kommunikation über unterschiedlichste technische Systeme hinweg. Klare Schnittstellen ermöglichen den einfachen Austausch von Netzwerkprotokollen.

A.3.5 Internet und die TCP/IP Protokolle

Das Internet ist ein Netz von Netzen. Diese sind untereinander durch Vermittlungsrechner verknüpft. Basis des Internets ist eine Familie von Protokollen, die bis etwa 1982 spezifiziert wurden und unter dem Namen TCP/IP bekannt sind.

- **TCP (transmission control protocol)** ist verantwortlich für das *Verpacken* der zu übertragenden Daten (Email,...) in eine Folge von Datenpaketen. Diese werden mit einer Adresse versehen und an die niedrigere Schicht, IP, weitergereicht.
- **IP (internet protocol)** ist für den *Versand* der einzelnen Pakete zuständig. Jedes Datenpaket wandert zu dem nächsten Vermittlungsrechner und wird von diesem weitergeleitet, bis es über viele Zwischenstationen an einem Vermittlungsrechner ankommt, in dessen Bereich (*domain*) sich der Zielrechner befindet.

A.3.6 Internetadressen

Jeder Rechner, der am Internet angeschlossen ist, benötigt eine Adresse. Überwiegend wird noch die Version 4 des IP-Protokolls benutzt, abgekürzt **IPv4**. Die Zahl der mit diesem Protokoll adressierbaren Rechner (=4,294,967,296) bzw. Netze hat sich als zu klein erwiesen. Daher wurde ein neues Protokoll, IPv6 (IP Version 6) entwickelt.

Eine IPv4 Adressse setzt sich zusammen aus:

- **Netzadresse des Subnetzes**, in dem der Rechner sich befindet, z.B.

`136.199.*.* = uni-trier.de.`

Paketvermittlung im Internet zwischen verschiedenen Netzwerken erfolgt ausschließlich über die Netzadressen.

- **Hostadresse:** Adresse innerhalb dieses Subnetzes, z.B.

`*.*.235.16 = math.uni-trier.de.`

Sobald das Paket den Vermittlungsrechner des Zielnetzwerkes erreicht hat, kann dieser sich mit dem dezidierten Zielgerät über die Hostadresse verbinden.

Internetadresse (IPv4-Adresse) = Netzadresse + Hostadresse.

Diese wird als 32-Bit-Zahl codiert und der besseren Lesbarkeit wegen in der Form

`A.B.C.D`

notiert:

- dabei stehen A, B, C, D jeweils für die dezimalen Äquivalente der vier Bytes
- Per Konvention findet sich
 - die **Netzadresse** am Anfang der Gesamtadresse
 - die **Hostadresse** ergibt sich aus den restlichen Bits

Beispiel

Die Netzadresse aller IPs von Rechnern im Netz der Uni Trier lautet

136.199,

für die Subnetze des Fachs Mathematiks haben wir die Hostadressen:

- 123.* (Mitarbeiter selbst verwaltet → sichtbar nur innerhalb VPN),
- 234.* (Mitarbeiter zentral verwaltet → komplett isoliert),
- 235.* (www-Server → nach außen sichtbar),
- 236.* (Testwiese).

Wir können im internen Mathe-Netz also maximal $4 \cdot 256 = 1024$ Rechner mit *festen* IPs versehen.

Demo:

- ping syrma (math316, enif)
- ping 192.168.178.45 (mein Smartphone) – WLAN an/aus

Netzmaske

Die 32Bits von IP Adressen können an einer beliebigen Stelle in Netz- und Hostadresse geteilt werden. Die *Netzmaske* definiert wo dieser Schnitt stattfindet.

Beispiel

Die Deutsche Telekom z.B. hat den den Netzadressbereich

217.224.0.0 bis 217.255.255.255,

also die Binärzahlen

11011001.11100000.00000000.00000000	217.224.0.0
11011001.11100000.00000000.00000001	217.224.0.1
:	:
11011001.11100000.00000000.11111111	217.224.0.255
11011001.11100000.00000001.00000000	217.224.1.0
:	:
11011001.11100000.00000001.11111111	217.224.1.255
11011001.11100000.00000010.00000000	217.224.2.0
:	:
11011001.11100001.00000000.00000000	217.225.1.0
:	:
11011001.11100010.00000000.00000000	217.225.0.0
:	:
11011001.11111111.00000000.00000000	217.255.0.0
:	:
11011001.11111111.00000000.00000000	217.255.0.0
:	:
11011001.11111111.11111111.11111111	217.255.255.255

Wir beobachten:

- Alle IP-Adressen haben die Form

11011001.111xxxxx.xxxxxxxxxx.xxxxxxxxxx,

d.h. die ersten 11 Bits sind gleich. Diese 11 Bits bilden die Netzadresse.

- Mit dem logischen AND ($\text{AND}(x,1)=x$, $\text{AND}(x,0)=0$) gilt für all diese IP-Adressen

11011001.111xxxxx.xxxxxxxxxx.xxxxxxxxxx AND 11111111.11100000.00000000.00000000
 $= 11011001.11100000.00000000.00000000,$

d.h. wir können mit der Bitfolge **11111111.11100000.00000000.00000000** diese ersten 11 Bits filtern.

Als **(Sub)netzmaske** bezeichnet man die 32-Bit Binärzahl, die an den zur Netzadresse gehörenden Bitpositionen eine 1 hat, überall sonst eine 0. Für den Netzadressbereich der Deutschen Telekom können wir also verkürzt schreiben (*Classless Inter-Domain Routing (CIDR)*).

217.224.0.0/11,

wobei **11** die Anzahl der mit 1 besetzten ersten Bits der Netzmase angibt, dh. die Subnetzmaske lautet

11111111.11100000.00000000.00000000 = 255.224.0.0.

Demo:

- Beispiel: Der Netzadressbereich der Uni Trier ist

136.199.0.0/16

mit Netzmase

255.255.0.0 also 11111111.11111111.00000000.00000000.

Für Subnetze wie das 123-er Mathenetz

136.199.123.0/8

mit Netzmase

255.255.255.0 also 11111111.11111111.11111111.00000000.

- Beispiel: Manuelle (bzw. statische) IPv4 Einstellungen für einen Rechner im Mathe-Subnetz 136.199.123.*

DHCP/IPv4 Methode:	manuell
IPv4 Adresse:	136.199.123.108
Netzmase:	255.255.255.0
Gateway:	136.199.123.1
DNS:	136.199.8.101, 136.199.8.129

Siehe auch auf meinem Rechner im Büro die Dateinamen im Verzeichnis (Hinweis: pwd im Klartext!!!)

/etc/NetworkManager/system-connections

Private Subnetze

Eine Teilmenge des IPv4 Adressraums ist für bestimmte Zwecke reserviert; zum Beispiel für private IPv4 Adressen. Die reservierten Teilmengen für private Adressen sind:

- 10.0.0.0/8
- 172.16.0.0/12

- 192.168.0.0/16

Da jedes Netzwerk Adressen aus diesen Teilmengen verwenden kann, sind sie nicht eindeutig und dienen nicht dem Routing über das Internet. Die meisten Netzwerke (insbesondere zuhause) nutzen diese privaten Adressen. Mit der Außenwelt wird dann über einer einzigen "extern eindeutigen" IP vermittelt, welche zuhause typischerweise Ihrem Router zugeordnet ist.

- Router mappen dann alle internen Adressen auf diese einzige externe Adresse, wenn sie Pakete ins Internet weiterleiten.
- Diesen Vorgang nennt man "*Network Address Translation (NAT)*"

Konfiguration von IP-Adressen

Jedem Subnetz steht ein begrenztes **Kontingent von Hostadressen** zur Verfügung. Diese können bestimmten Rechnern *fest/statisch* oder *temporär/dynamisch* vergeben werden.

- **Statisch:** Rechner, die ständig im Netz sind, z.B. Server (wie `vpn.uni-trier.de`), erhalten immer eine feste Adresse.
- **Dynamisch:** Rechner, die nicht ständig online sind, kann bei Bedarf eine gerade freie Hostadresse zugeordnet werden (zB ein Smartphone Ihres Besuches in ihrem WLAN).
 - geschieht meist mit dem **Dynamic Host Configuration Protocol (DHCP)**
 - zuhause spielt meist der Router die Rolle des DHCP-Servers
 - Vorteil: Bei begrenzter Anzahl von Hostadressen, kann so dynamisch eine potenziell größere Zahl von Rechnern bedient werden

Das System der Domain-Namen.

Anstatt der 32-Bit IP-Adresse möchte man einen Rechner lieber über einen Namen in Form von Text, den sog. **hostname**, ansprechen:

- Zu Beginn hat man das manuell gelöst über Textdatei `hosts.txt`.
- Ab etwa 1980 wurde als Alternative eine verteilte Datenbanklösung zur Verwaltung von Namen im Internet entwickelt.
- 1984 wurde sie unter dem Namen **DNS (domain name service)** eingeführt

Beim Konzept **DNS** hatte man die Vorstellung, dass jedes Subnetz des Internets einen *Bereich (domain)* von Namen verwaltet und selbst einen *Bereichsnamen (domain name)* hat.

Domain-Namen im Internet bestehen aus mindestens zwei Komponenten, die durch Punkte getrennt sind:

`domain.ToplevelDomain`

oder

`subdomain.domain.ToplevelDomain.`

- Am weitesten rechts steht der Name einer **Toplevel-Domain** (abgekürzt: TLD). Zum Beispiel Länderspezifisch `.de` oder generisch wie `.com` für *commercial* (ursprünglich für Unternehmen)
 - Die zweite Komponente (von rechts) beinhaltet den eigentlichen **Domain-Namen**, zB `uni-trier`
 - dann folgt eine Einteilung in Unterbereiche (**subdomains**), zB `math`
- Beachte: Groß- und Kleinbuchstaben werden nicht unterschieden.

Firewall.

Eine Firewall realisiert u.a. die folgenden Schutzmaßnahmen:

- Der Zugriff aus dem Intranet (= ein in sich geschlossenes Netz) auf Rechner im Internet wird nur für bestimmte Netzadressen ermöglicht.
- Verbindungen zu den von außen zugänglichen Servern werden gefiltert, um Angriffe auf diese Rechner zu verhindern; Beispiel: Im Falle eines E-Mail-Servers unerwünschte Mails filtern.

A.3.7 Dienste im Internet

Rechner sind heute fast immer mit einem Netzwerk verbunden. Etabliert ist das **Client-Server-Konzept** einer dezentralen Rechnerversorgung mit:

- **Dienstleistern (server)**: Bieten Dienste (*services*) wie Datei-Verwaltung, E-Mail, WWW, Datenbankenbindung, Cloud Computing, etc.
- **Kunden (client)**: Nehmen diese Dienste in Anspruch.

Häufig benutzte Dienste:

- `https://` Hypertext Transfer Protocol Secure (Homepages)
- `smtp`: Simple Mail Transfer Protocol (E-Mails)
- `ssh`: secure shell
 - Man kann, sofern man die Benutzerberechtigung hat, zu entfernten Host-Rechnern (host = Gastgeber) mit
`ssh [options] [host]`
eine verschlüsselte Verbindung aufbauen
 - Programme auf entfernten Rechnern ausführen
- für die Datei-Übertragung (unten mehr):
 - `ftp [options] [host]`: file transfer protocol
Rechner gestatten einen eingeschränkten Zugang mit dem Dienst ftp, um Dateien untereinander auszutauschen
 - `sftp` secure file transfer protocol (ftp mit ssh)
 - `scp`
 - `nfs://` network file system
 - `smb://` Server Message Block
→ in einer Ur-Version auch als Common Internet File System (CIFS)
→ wird vom frei verfügbaren Softwareprojekt Samba
 - `dav://` WebDAV
 - `rsync` Netzwerkprotokoll zur unidirektionalen Synchronisation von Daten (lokal oder über Rechner in einem Rechnernetz)
→ effizient da mitunter nur geänderten Teile einer Datei übertragen werden (Delta-Kodierung)
- `ping` ist ein Diagnose-Werkzeug, mit dem überprüft werden kann, ob ein bestimmter Host in einem IP-Netzwerk erreichbar ist.

```
wollmann@zanta:~$ ping google.de
PING google.de(fra24s07-in-x03.1e100.net (2a00:1450:4001:82a::2003)) 56 data bytes
64 bytes from fra24s07-in-x03.1e100.net (2a00:1450:4001:82a::2003): icmp_seq=1 ttl=119 time=16.3 ms
64 bytes from fra24s07-in-x03.1e100.net (2a00:1450:4001:82a::2003): icmp_seq=2 ttl=119 time=16.0 ms
^C
--- google.de ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
```

Abbildung 12: ping an www.google.de.

Demo:

- `ssh syrma.uni-trier.de`
- `rsync` remote (e.g. `ssh syrma.uni-trier.de`) to local
- `ping`

Hausaufgaben

1. Welche Verbindungsart nutzen Sie gerade, um ins Internet zu kommen?
2. Welche IP-Adresse hat Ihr Gerät gerade?
3. Wurde diese IP-Adresse statisch oder dynamisch vergeben?

Analysieren Sie Ihr Heimnetz

- Welche VRs (zB Router, Repeater, Powerlink) sind vorhanden?
- Welche netzwerkfähigen Geräte sind in Ihrem Heimnetz? Rechner, TV, Smartphone, Synology/NAS/, Heizung, Drucker, Herdplatte, Lampen, Fahrrad-Tacho,...
- Wie sind diese Geräte miteinander verbunden (*wire(less)*)? Welches Protokoll verwenden Ihre Geräte im WLAN? Was ist eigentlich ein WLAN-Schlüssel?
- Finden Sie heraus, welche IP-Adresse Ihr Rechner oder Smartphone von Ihrem Router aktuell zugeordnet bekommen hat. Wie lautet die Netzadresse? Geben Sie diese Netzadresse in Google ein: Was ist die Besonderheit dieser Netzadresse (Stichwort: Private IP-Adresse)?
Meiner besitzt (aktuell) als Netzadressen-Anteil: 217.251 mit Gateway 217.251.34.1, ein Gateway in Trier von der Deutsche Telekom AG
- Falls Sie Zugang haben sollten: Schauen Sie sich auch mal in der Browser-Maske Ihres Routers um (zB <http://fritz.box/>). Welche Internetadresse hat Ihr Router (Freigaben festgelegt für Erreichbarkeit von außen)?
- Falls Sie Zugang haben sollten: Können Sie Ihren Geräte feste IPs zuweisen? Zum Beispiel sinnvoll, um den eigenen Netzwerk-Drucker stets unter der selben IP zu erreichen.
- Schauen Sie in ihrem Dateimanager nach den Optionen nfs://, smb://, sftp://, ftp://, dav://

A.4 Virtualisierung

Virtualisierung⁸ bezeichnet in der Informatik die Nachbildung (Emulation) eines Hard- oder Software-Objekts durch ein ähnliches Objekt vom selben Typ mit Hilfe einer **Abstraktionsschicht**.

Zur Unterscheidung echter und virtueller Umgebungen werden diese Wirt (*host*) und Gast (*guest*) genannt:

- **Wirt** (*host*): bezeichnet immer die Ebene (oder Schicht), welche der physischen Hardware am nächsten ist
- **Gast** (*guest*): die auf dem Wirt ausgeführte Umgebung

Anwendungsbeispiel: Betriebssystem innerhalb eines anderen ausführen.

- Mithilfe einer geeigneten Virtualisierer-Software (zB Virtualbox oder VMWare) können wir beispielsweise andere Betriebssysteme (zB Gast = Ubuntu 20.04) in einem Fenster unter unserem derzeitigen Betriebssystem (zB Host = Windows 10) nutzen und Dateien zwischen Host- und Gast-Betriebssystem austauschen.
- Dabei *emuliert* die Virtualisierer-Software die fehlende PC-Hardware des virtuellen Betriebssystems und leitet Hardware-Anfragen an die echte Hardware (CPU, Grafikkarte, Festplatte) des physisch vorhandenen PCs weiter.
- Es wird dem Gast-Betriebssystem die Alleinnutzung eines Computers vorgegaukelt, wobei es tatsächlich innerhalb eines anderen Betriebssystems als gewöhnliche Anwendung läuft.

Weitere Anwendungsbeispiele und Vorteile:

- Computer-Ressourcen/Dienste aufteilen oder zusammenfassen
- Agilität durch **Unabhängigkeit** von der physischen Hardwareinfrastruktur
→ Abstraktionsschicht täuscht (andere) physische Gegebenheiten vor
- Agilität durch **Modularisierung** (eine virtuelle Machine bekommt nur eine einzige Aufgabe; kann schnell ausgetauscht werden)

Demo:

- Fallbeispiel: Mathe-Netz (abstrakt und proxmox)

Wir schauen uns in diesem Zusammenhang noch kurz etwas um:

- Softwarevirtualisierung: Virtuelle Maschinen und Container
- Hardwarevirtualisierung
- Netzwerkvirtualisierung: VLAN und VPN

A.4.1 Softwarevirtualisierung

Wir unterscheiden: Virtuelle Maschinen und Container.

Virtuelle Maschine: Systemvirtualisierung mittels Hypervisor

Als virtuelle Maschine (VM) wird in der Informatik die Software-technische Kapselung eines Rechnersystems innerhalb eines lauffähigen Rechnersystems bezeichnet.

- Das Betriebssystem der VM verwendet dabei seinen eigenen Betriebssystemkern.
- Die Abstraktionsschicht zwischen Host und Guest wird **Hypervisor** oder **Virtual Machine Monitor** genannt.
- Bekannte Umsetzungen:
 - Oracles VirtualBox,
 - VMwares vSphere.

⁸“Virtuell” wie “nicht-physisch”

Container: Systemvirtualisierung mittels OS-Container

Containervirtualisierung (oder Containering) ist eine Methode, um mehrere Instanzen eines Betriebssystems (als Gäste) isoliert voneinander den Kernel des Hosts nutzen zu lassen.

- Im Gegensatz zur Virtualisierung mittels eines Hypervisors gilt Containervirtualisierung als besonders ressourcenschonend.
- Container (Sandbox) haben nur Zugriff auf einen Teil der **Systemressourcen**:
 - Nutzbare Hardware(komponenten), wie CPU und Netzwerk
 - Speicher (Lesen/Schreiben), Ordnerstrukturen und Netzwerkspeicher
 - Peripheriegeräte wie Tastatur, Webcam, Scanner und Drucker.
- Bekannte Umsetzungen:
 - **LXC** (Linux Container)
 - **Docker**; benutzerfreundliche Werkzeuge (u.a. eine Beschreibung von Images (Dockerfiles) oder ein Repository, das solche Images verwaltet); seit ca. 2013 sehr populär.

VMs

1. VirtualBox installieren (gui): <https://www.virtualbox.org/wiki/Downloads>
2. Ubuntu 20.04 als VM installieren
3. Ggf müssen Sie Virtualisierung auf Ihrer CPU zunächst erlauben:
Enable Virtual Hardware in BIOS/UEFI.
Press F2 (or other F- ..) key at startup BIOS/UEFI Setup.
Switch to the Advanced Mode tab then choose CPU Configuration option.
Scroll down to the "Intel Virtualization Technology" value, and then change [Disabled]->[Enabled].
Save Changes and Exit.

Container

1. docker desktop/server installieren; siehe <https://docs.docker.com/engine/install/>
2. Von Docker Hub ein Docker Image (instanz) ausführen; zB nextcloud (https://hub.docker.com/_/nextcloud)

A.4.2 Hardwarevirtualisierung

Hierfür können entweder das ganze System oder nur einzelne seiner Komponenten, wie z.B. CPU oder GPU, virtualisiert werden.

Anwendungsbeispiel: GPU eines leistungsfähigen GPU-Servers virtuell partitionieren und Anwendern (zB Studierenden) diese für einen gewissen Zeitraum (zB Abschlussarbeit) für rechenintensive Probleme isoliert zur Verfügung stellen.

A.4.3 Netzwerkvirtualisierung

Durch die physikalisch angeklemmte Hardware entsteht über Router, Switches, Rechner, Drucker etc. ein physikalisches Netzwerk (zB innerhalb der Uni).

Virtual Local Area Network (VLAN)

Um das Netzwerk in einzelne Teilbereiche (sog. Subnetze) *unabhängig von der physischen Gegebenheit* zu unterteilen, bedarf es einer geeigneten Virtualisierung. Damit bleibt man auch hier agil und flexibel. Ein Virtual Local Area Network (VLAN) ist ein logisches Subnetz welches sich über mehrere Switches hinweg ausdehnen kann.

Anwendungsbeispiel: Das Netz der Universität Trier kann folgende IPs vergeben

136.199.XXX.YYY.

Dabei können feste Zahlen XXX als Subnetze/VLANs verstanden werden, in denen dann 256 verschiedene IPs für Rechner, Drucker, Server etc vergeben werden können. Zum Beispiel betreiben wir in der Mathematik die Subnetze:

- 136.199.123.YYY (Mitarbeiter selbst verwaltet),
- 136.199.234.YYY (Mitarbeiter zentral verwaltet, CIP Pools),
- 136.199.235.YYY (www-Server),
- 136.199.236.YYY (Testwiese).

Durch geeignete Firewall-Regeln kann zudem der Zugriff aus und auf die Subnetze geregelt werden. So sollte zum Beispiel ein Web-Server im Subnetz 136.199.235.YYY nach außen sichtbar sein und http(s)-Anfragen o.ä. von beliebigen IP-Adressen beantworten. Auf Knoten im Mitarbeiter-Netz 136.199.234.YYY können hingegen nur einige privilegierte IP-Adressen außerhalb der Range 136.199.234.YYY zugreifen.

Virtual Private Network (VPN)

Andererseits möchte man auch über Knoten (wie Router, Switches,...), die nicht im eigenen Hardwarebestand sind, ein geschütztes isoliertes Netz administrieren (zB das Netz der Universität Trier; Unternehmensnetz).

Ein Virtual Private Network (VPN) bildet ein nach außen abgeschirmtes Netzwerk über fremde oder nicht vertrauenswürdige Netze. Virtuell in dem Sinne, dass es sich nicht um eine eigene physische Verbindung handelt, sondern um ein bestehendes Kommunikationsnetz, das als Transportmedium verwendet wird.

Anwendungsbeispiel:

- Der Computer eines Studierenden kann von zu Hause aus Zugriff auf das Uni-Trier-Netz erlangen. Aus Sicht der VPN-Verbindung dienen die dazwischen liegenden Netze (sein Heimnetz sowie das Internet) als Verlängerungskabels, das den Computer (VPN-Partner) ausschließlich mit dem zugeordneten Netz verbindet (VPN-Gateway).
- Der sich daraus ergebende Nutzen eines VPNs kann je nach verwendetem VPN-Protokoll durch eine Verschlüsselung ergänzt werden, die eine abhör- und manipulationssichere Kommunikation zwischen den VPN-Partnern ermöglicht.
- Ein verschlüsseltes VPN über ein unverschlüsseltes Netzwerk herzustellen, ist mit einer der Hauptgründe für die Verwendung eines VPNs.

Sobald ein Computer eine VPN-Verbindung aufbaut, ist der Vorgang vergleichbar mit dem Umstecken seines Netzwerkkabels von seinem ursprünglichen Netz an das neu zugeordnete Netz, mit allen Auswirkungen wie geänderten IP-Adressen und Unterschieden beim Routing.

Eigenes VPN

- Wer Lust und Zeit hat: VPN Heimnetz konfigurieren mithilfe ihres Routers, zB fritz.box. Damit könnten Sie zB von unterwegs mittels ihres Smartphones über das Internet (LTE, ...) getunnelt einen Druckauftrag o.ä. nach Hause senden oder ihre eigene Nextcloud (die sie zuvor einfach als Docker image installiert haben) erreichen.

Hausaufgabe:

VPN + Anmeldung auf dem Remote Server

1. Verschaffen Sie sich Zugang zum VPN der Universität Trier über Ihre ZIMK-Kennung.
2. Melden Sie sich auf dem Server syrma an. [Siehe Anleitung auf der Titelseite.]

B Daten Management

Wir wollen uns gegen den Verlust wertvoller Daten durch bspw. Hardware-Versagen, Hausbrand oder irreversibler Datei-Löschen schützen. Zudem möchten wir mit mehreren Endgeräten auf einem einzigen synchronen Datenbestand arbeiten. Dazu werden wir im Folgenden einige Begrifflichkeiten und Konzepte in diesem Kontext erklären und letztlich Seafile als unseren Lösungsansatz implementieren.

B.1 Der Begriff Datensicherung (*backup*)

Datensicherung *bezeichnet das Kopieren von Daten in der Absicht, diese im Fall eines Datenverlustes zurückkopieren zu können*. Wir unterscheiden die Funktionen

- **backup** (Sicherungskopie): Daten redundant auf einem Speichermedium sichern.
- **restore** (Datenwiederherstellung): Originaldaten aus einer Sicherungskopie wiederherstellen.

Bemerkung: Wir werden uns nicht mit der Sicherung im Sinne von Verschlüsselung beschäftigen.

B.1.1 Arten der Datensicherung

- Vollsicherung:
Die reinen Daten (ohne Dateisystem usw.) einer Festplatte/Partition o.ä. werden komplett auf das Speichermedium kopiert.
 - + Sehr einfach zu verstehen und implementieren
 - Speicher- und Zeitbedarf
- Differentielle Sicherung:
Nur die Dateien werden gespeichert, die seit der letzten Vollsicherung geändert wurden oder neu hinzugekommen sind.
 - + Reduzierter Speicher- und Zeitbedarf
 - Sichert trotz nur kleiner Änderungen die ganze Datei nochmals
- Inkrementelle Sicherung:
Nur die Dateien oder Teile von Dateien werden gespeichert, die seit der letzten inkrementellen Sicherung geändert wurden oder neu hinzugekommen sind. Das kann bei Blockgeräten (Festplatte) auf Block-Level passieren.
Tools unter Linux: `duplicity`, `rdiff-backup`
 - + Sehr geringe Speicher- und Zeitbedarf; eignet sich daher für die Datensicherung in Netzwerken/Cloud
 - Prinzipbedingt sind alle Inkremeante miteinander verkettet:
Originaldaten = `restore`(Komplettsicherung, inkrement. Sicherung 1,...,n)
→ Rechenaufwand bei `restore`
- Speicherabildsicherung (image backup):
Der komplette Datenträger wird durch ein 1-zu-1-Abbild gesichert, also das gesamte Dateisystem, inklusive Betriebssystem, Benutzereinstellungen etc.
 - + Klonen oder Wiederherstellen eines kompletten Systems möglich
 -

B.1.2 Speicher-Hardware

https://en.wikipedia.org/wiki/Backup#Storage_media

B.1.3 Strategien der Datensicherung

<https://de.wikipedia.org/wiki/Datensicherung#Backupstrategien> 3-2-1 backup Regel

B.1.4 Software

https://de.wikipedia.org/wiki/Liste_von_Datensicherungsprogrammen
`duplicity`, `rdiff-backup`, `rsync`, `sync-clients`, `unison`,...

B.1.5 Datenmanipulation

https://en.wikipedia.org/wiki/Backup#Manipulation_of_data_and_dataset_optimization
Kompression, Verschlüsselung, usw.

Hausaufgaben

1. Lesen Sie die Informationen vom ZIMK unter IT-Dienste und -Services/Datensicherung:
<https://www.uni-trier.de/universitaet/wichtige-anlaufstellen/zimk/gemeinsame-seiteninhaltse/datensicherung>
2. Siehe auch 3-2-1 backup Regel

B.2 Outsourcing: Online-Datensicherung

Als Online-Datensicherung (*Cloud-Backup*) bezeichnet man eine Datensicherung über das Internet (...) auf Datenspeichern eines Internetdienstanbieters/Unternehmens. Wir können Drittanbieter bezahlen (hftl. vertrauenswürdige Spezialisten), die Datensicherung für uns zu übernehmen.

Technische Voraussetzungen

- Internetzugang (idealerweise mit verschlüsselter Verbindung; z.B. [https](https://))
- Dienste/Protokolle auf Server und Client Seite mit Benutzerkennung über ein Verzeichniszugriffsprotokoll (ldap, AD, auch in Kombination mit Shibboleth o.ä), um Zugriffsrechte etc. zu verwalten.
- Datenmanipulation: Datenübertragung über Internet langsamer als über direkte Schnittstellen zum Speichermedium, daher
 - Datenkompression
 - Inkrementelle Datensicherung

Das Konzept des delokalisierten Speichers (Stichwort: Modularisierung) bringt neben der Sicherung generell viele weitere Vorteile mit sich: Hochverfügbarkeit, Verfügbarkeit auf vielen Endgeräten, Zusammenarbeit, Webbasiertes Zugreifen usw. (in Abschnitt B.4.2 mehr dazu)

B.3 Konzept 1: Externes Dateisystem einhängen

Hier: mounten via nfs und smb und der sftp Prompt

In einem Netzwerk (z.B. das Uni-Netz) befindet sich ein zentraler, massiv abgesicherter Speicher, auf dem alle Nutzer im Netzwerk Daten (Text, Bilder, Emails, Kalender,...) ablegen können oder auf dem die Administratoren bestimmte Software zur Verfügung stellen können. Wir dazu brauchen unter anderem die folgenden Komponenten:

- **Verzeichniszugriffsprotokoll:**
Damit user Bob nicht die Daten von user Alice einsehen kann, werden bestimmte Zugriffsrechte über eine Benutzer-Kennung samt Passwort (z.B. ZIMK-Kennung) von einem dafür zuständigen (vermutlich virtuellen; Stichwort: Modularisierung) Server verwaltet (z.B. Active Directory Service).
- **Fileserver:**
Um den zentralen Speicher zu verwalten, gibt es auf Seiten des Dienstleisters einen sogenannten Fileserver, der sich u.a. mit dem Verzeichnisverwalter über die Zugriffsrechte austauscht.
- **Protokoll und dessen Implementierung:**
Für die Datenübertragung zwischen zwei Rechnern im Netzwerk werden bestimmte Netzwerkprotokolle verwendet (z.B. smb, nfs,...). Diese Protokolle werden von einer Software implementiert (z.B. Samba für smb, nfs-kernel-server/nfs-common)

Ein Nutzer (*client*), also ein Rechner im Netzwerk mit einer bestimmten IP-Adresse, kann dann über die vom Fileserver angebotenen Protokolle (z.B. smb, nfs,...) Daten zwischen seinem und dem zentralen Speicher transferieren. Andererseits kann man auch gleich auf diesem (eingehangen; man spricht von *mount*⁹) Datenbestand arbeiten: Das

⁹engl. anbringen, montieren

externe Dateisystem wird einfach in das lokale Dateisystem eingehangen (zB in Windows als neues Laufwerk) und ist aus Nutzersicht allenfalls an der Zugriffsgeschwindigkeit von den anderen Dateien zu unterscheiden.

Beispiel: Das "U-Laufwerk"

- Studierende der Universität Trier möchten Rechner-Ressourcen (CIP-Pools,..) nutzen. Dabei sollte es idealerweise möglich sein, sowohl auf dem Campus im CIP-Pool, als auch zuhause am Privatrechner auf demselben Datensatz zu operieren ohne diesen über Email, USB-Stick o.ä. mühselig hin und her zu kopieren.
- Das Rechenzentrum der Universität hat einen zentralen Speicher, auf dem die Studierenden ihre Daten ablegen. Die Zugriffsrechte auf diese Daten werden durch die ZIMK-Kennung über ein Active Directory sichergestellt. Diese dort abgelegten Daten werden von der Universität sogar (mittels geeigneter Netzwerkprotokolle) über das Internet exportiert, sodass Angehörige der Uni auch zuhause auf Ihre Daten zugreifen können: Das sog. U-Laufwerk.

Vorteil: Sofern die Datenübertragung hinreichend schnell ist (zB Glasfasernetz innerhalb eines Unternehmens) braucht man am Client lokal kaum noch Speicherplatz. Das kann man auf die Spitze treiben (Software, OS etc. alles auslagern) und landet bei dem Konzept der sogenannten *thin clients*.

Nachteil: Man kann nur bei bestehender Verbindung (zB via Internet) zum Fileserver auf den Daten arbeiten.
→ Daher: Zusätzliche Synchronisation.

Demo: smb

smb-share: U-Laufwerk der Uni mounten über smb (unter Linux wird cifs synonym verwendet)

Voraussetzung: VPN Verbindung, Uni-LAN oder ZIMKFunkLan

Die entsprechenden **Parameter** für das smb-Protokoll lauten:

```
address=ifsp.uni-trier.de/u      # (uni-trier.de/dfs anderer Ort auf den ich noch Zugriff habe)
domain=urt
vers=3.0
username=<ZIMK-Nutzerkennung>
password=<Pwd Ihrer ZIMK-Nutzerkennung>
```

- **Terminal**
(\$ sudo apt-get install smbclient)
\$ sudo apt-get install cifs-utils # notwendige Software installieren
\$ mkdir <myMountPoint> # Montagepunktbaum einhängen erstellen
\$ df -Th
\$ sudo mount -t cifs -o username=vollmann, domain=urt, vers=3.0 //ifsp.uni-trier.de/u <myMountPoint>
\$ df -Th
\$ sudo umount <myMountPoint> # un-mount
\$ df -Th

Bemerkung: CIFS wird öfter als Synonym für das SMB-Protokoll oder sogar dessen ganze Protokollfamilie verwendet.

- **Dateimanager** (<https://www.uni-trier.de/index.php?id=73271>)
"mit Server verbinden": smb://ifsp.uni-trier.de/u

Demo: nfs

nfs-share: Mathe-Software /usr/local

[wird bei Ihnen nicht funktionieren da fehlende Rechte]

- **Terminal**
\$ showmount -e mathetest
\$ sudo apt install nfs-common
\$ sudo mkdir <myMountPoint>
\$ sudo mount -t nfs mathetest.uni-trier.de:/usrlocal <myMountPoint>

- **Dateimanager**
analog zu smb

Bemerkung: Damit wir unser home-Verzeichnis von syrma per nfs/smb auf unsere lokale Maschine mounten können, müssten die entsprechenden Dienste auf Server und Client installiert und konfiguriert werden! (\$ sudo apt-install nfs-common nfs-kernel-server #(?) ...)

Demo: sftp

sftp: Dateitransfer zwischen local und syrma

- **Terminal**

```
$ sftp vollmann@syrma.uni-trier.de  
# es öffnet sich ein sftp prompt, siehe zB dieses Tutorial; navigieren mit cd, ls, ...  
$ get remoteFile  # Kopie ins aktuelle Verzeichnis, von dem aus man sftp prompt geöffnet hat  
$ get remoteFile localFile  # Andere Richtung mit "put"
```

- **Dateimanager**

```
$ sftp://vollmann@syrma.uni-trier.de/home/vollmann
```

Hausaufgabe:

U-Laufwerk

1. Hängen Sie Ihr sogenanntes "U-Laufwerk" von der Universität Trier manuell auf Ihrem Privatrechner ein:

https://www.uni-trier.de/fileadmin/urt/IT-Dienste-Home-Office-Telearbeit/Netzlaufwerke_verbinden_MSWindows.pdf

B.4 Konzept 2: Externes Dateisystem lokal spiegeln durch Dateisynchronisation

Dateisynchronisation (*Filesyncing*) ist die Übertragung von Dateien (...) zur Erzeugung eines einheitlichen Datenbestands an unterschiedlichen Orten.

Insbesondere bei mobilen Geräten von Vorteil. Fest installierte Rechner im Intranet haben in der Regel über nfs/smb stabile Zugriff auf die Daten.

Wir diskutieren zwei Umsetzungen:

B.4.1 Unter Linux: mount/ssh+rsync oder app wie unison

Mit dem Linux-Tool rsync lässt sich unidirektionale Dateisynchronisation leicht über Kommandozeilen umsetzen.

- Variante 1: Manuell

- Verbindung herstellen:

```
mount: (physisch) externes Dateisystem mounten per nfs, smb,...
```

ssh: Falls man Zugriff auf den externen Server hat, so kann auch eine ssh Verbindung verwendet werden

- Dateisynchronisation:

Das eingehangene Dateisystem kann nun mithilfe von rsync mit einem (physisch) lokalen Dateisystem synchronisiert werden

(eventuell direkt Zugriff per rsync)

Demo:

– ssh+rsync:

Anwendungsbeispiel: Die Seiteninhalte auf dem externen web-Server möchte ich lokal erstellen. Die Daten sollten synchron sein.

- * executable erstellen "pushWWW.sh"

```
$ nano/gedit pushWWW.sh
```

```
pdflatex ... [.tex to .pdf]  
rsync ... [push from local to remote]
```

- * Ausführbar machen

```
$ chmod +x pushWWW.sh
```

- * Lokaler Rechner: Änderungen im Texteditor tätigen

- * Auf WWW-Server "pushen":

```
$ ./pushWWW.sh
```

- * Beobachten auf remote:

```
https://www.math.uni-trier.de/~vollmann/prog/script/prog-teil-2.pdf
```

– mount+rsync:

U-Laufwerk mit einem Ordner lokal per rsync synchronisieren

- Variante 2: (Halb-)Automatisiert

In der ersten Variante haben wir die Schritte zu Fuß einmalig umgesetzt. In der Praxis sollten diese Schritte besser anhand von bestimmten Auslösern (Zeitrhythmus, Dateiänderung,...) automatisiert ausgeführt werden.

- Ggf. Automatisches mounten: Die Konfigurationsdatei fstab (Abkürzung für file system table, deutsch „Dateisystemtabelle“) wird während des Bootens vom Befehl mount zeilenweise gelesen. Diese Konfigurationsdatei liegt im Verzeichnis etc und enthält eine Liste aller zu einzhängender Datenspeicher.
- Synchronisation des eingehangenen Dateisystem per rsync (ggf. über ssh) automatisierten mithilfe des Cron-Daemon¹⁰. Dieser dient allgemein der zeitbasierten Ausführung von Prozessen in unixartigen Betriebssystemen, um wiederkehrende Aufgaben (=Cronjobs) zu automatisieren.

Demo: \$ cat /etc/fstab

- Variante 3: Automatisiert

- Für diesen Einsatzzweck geschaffene Anwendungen wie unison verwenden, die ggf noch mit einer GUI daherkommen.

Vorteil: Eine Anwendung wie unison läuft auch auf dem externen System und kann den Stand jeder Datei zentral monitoren. Dadurch entsteht deutlich weniger Netzwerkverkehr.

B.4.2 Sync Clients mit browser backend

Was durch die bisherigen 3 Varianten noch nicht umgesetzt sind u.a.: Zugriffsrechteverwaltung (share-link etc), browserbasiertes Arbeiten, OS-Unabhängigkeit usw. Daher jetzt:

Variante 4: Eine mächtige Software wie Nextcloud, ownCloud, Seafile bzw. die von Anbietern wie Dropbox¹¹, Google Drive, iCloud oder MS OneDrive nutzen, die folgende Features bereitstellt:

- **Dateisynchronisation** mit einem externen Speicher
- **Sync clients** für verschiedene Betriebssysteme → können auf jeglichen Endgeräten arbeiten
- **Versionskonflikte:** Wenn verschiedene Benutzer gleichzeitig dieselbe Datei verändern oder Internetverbindung an einem Client ausfällt, wird das von dem System bemerkt und es speichert Kopien der veränderten Datei unter einem modifizierten Namen, der auf die Inkonsistenz aufmerksam macht
- **Zusammenarbeit:** Statt Dateien per Email zu verschicken, kann man dem Empfänger bspw. einfach einen link auf diese Dateien senden.

¹⁰Als Daemon bezeichnet man unter unixartigen Systemen einen Prozess, der im Hintergrund abläuft und bestimmte Dienste zur Verfügung stellt.

¹¹Bemerkung: Der Dropbox service unter linux in Python3 geschrieben

- **Dateiversionierung und Snapshots**
- **Online Bearbeitung von Dateien:** Zusätzlich werden Browserbasierte Programme bereitgestellt um E-Mails, Textdokumente, Präsentationen und Tabellenkalkulationen zu bearbeiten, ohne dazu eigene Verarbeitungsprogramme zu verwenden.
→ Browserbasierte (und damit OS unabhängige) Konzepte sind aktuell häufig vorzufinden (zB jupyter notebooks)
- **APIs:**¹² Es gibt Schnittstellen zu anderen Anwendungen, zB können Sie evtl. die Daten Ihres Fahrradcomputers oder von Gootnotes in Ihrer Dropbox ablegen
- **Unterstützung weitere Protokolle:** Oftmals wird der Datenbestand auch mittels anderer Protokolle exportiert. So können wir auch via WebDAV (Vorteil: Verbindung geht über den meist freigeschalteten http-Port 80) auf unsere Seafile-Daten zugreifen.

Demo: \$ vi/nano/gedit /bin/dropbox

Vorsicht: Datenschutzgesetze in anderen Ländern

Die Speicherung von Daten in einer Cloud setzt Vertrauen zu den Anbietern dieser Dienste voraus, wenn man ihnen Daten und Dokumente *ohne eigene Verschlüsselung*¹³ anvertraut.

- Einige Lösungen unterhalten Ihre Serverstruktur ausschließlich in Deutschland, weshalb der gesamte Datenverkehr den strengen deutschen Datenschutzgesetzen untersteht, mit der Folge zusätzlicher Sicherung gegen nicht autorisierten Zugriff.
- Wenige Lösungen bieten darüber hinaus noch eine Ende-zu-Ende-Verschlüsselung, durch welche die Daten bereits lokal auf dem eigenen Rechner verschlüsselt werden und somit bereits auf dem Weg zum Server hin sicher verschlüsselt sind.
 - schützt gegen Abfangen der Daten in Form eines man-in-the-middle Angriffs
 - Aber: Anbieter können Daten entschlüsseln und zum Anfertigen von Benutzerprofilen verwenden – das ist der Preis für kostenlosen Speicher inkl. Tools. Alleinige Verschlüsselung durch den Anbieter ist daher für wichtige Daten nicht zu empfehlen.

“IT as a service” Softwareanbieter brauchen ihre Programme nicht mehr zu verkaufen, sie stellen sie auf ihren eigenen Rechnern zur Verfügung und der Benutzer zahlt je nach Nutzung. Auch Hardware kann man sich nach Bedarf aus dem Netz holen. Rechenintensive Tasks werden unsichtbar auf Großrechner in der Wolke verlagert und der Benutzer zahlt nach CPU-Sekunden. Solche und ähnliche Angebote gibt es von Firmen wie Google, Amazon, IBM und anderen. Das Geschäftsmodell heißt: “IT as a service”.

Hausaufgaben

1. Siehe zum Beispiel die Angebote des ZIMKs: <https://www.uni-trier.de/index.php?id=53078>

B.4.3 Seafile

Bei den Anbietern Dropbox, Google Drive, iCloud oder MS OneDrive lernen wir in der Regel nur die Kundenseite kennen. Es gibt jedoch auch freie Software, die exakt diese Dienste bereitstellen und uns damit ermöglicht einen eigenen unabhängigen Cloud-Speicher auf eigenen Servern mit völliger Kontrolle über die Daten aufzusetzen.

Bekannte Software ist: Nextcloud, ownCloud und eben auch Seafile ¹⁴.

Die Rechenzentrumsallianz Rheinland-Pfalz (RARP) administriert einen eben solchen Seafile-Server und alle Angehörigen einer Universität oder Hochschule in Rheinland-Pfalz haben Zugriff darauf. Ähnlich wie Sie sich mit Ihrem Google-Account bei Google-Drive anmelden, so können sie sich auf

¹²application programming interface, für Programmierschnittstelle

¹³Es gibt Möglichkeiten: cryFS, veracrypt

¹⁴seafile ist ein chinesisches Unternehmen mit Sitz in Beijing, China, first release 2012; Siehe auch <https://www.seafile.com/en/about/>

<https://seafolder.rlp.net/>

mit ihrer ZIMK Kennung anmelden und es stehen Ihnen als StudentIn 16GB Cloud-Speicher zur Verfügung. Dazu wird über Shibboleth das Active Directory der Uni Trier angezapft.

Hausaufgabe:

Seafolder einrichten

1. Lesen Sie die Informationen auf den Seiten der RARP:

<https://rarp.rlp.net/services/seafolder>

2. Mit ZIMK Kennung anmelden auf:

<https://seafolder.rlp.net/>

- My Libraries + **New Library** <ProjektName>
- Erstellen Sie eine Markdown-, Excel- sowie Word-Datei im Browser zum testen.
- Welche Optionen haben Sie, um diese Bibliothek mit anderen zu teilen?

3. Download Desktop **Syncing** Client (**nicht** Desktop Drive Client):

<https://www.seafolder.com/en/download/>

- Beim Konfigurieren müssen Sie die Server-Adresse von oben angeben
- **Vorsicht:** Nur Libraries und deren Unterordner werden vom Desktop Client synchronisiert.
- Suchen Sie Ihre Library /home/user/Seafolder/<ProjektName>/

4. In Ihrem **Dateimanager** unter /home/user/Seafolder/<ProjektName>/

- Teilweise Verzeichnisstruktur von Abb. 4 und 5 anlegen

5. Richten Sie WebDAV ein: Anleitung der RARP

- WebDav Passwort setzen: Settings > WebDav Password
- Die benötigten WebDAV Parameter für Ihren entsprechenden Client (zb Browser, Dateimanager) sind dann:
 - server=seafolder.rlp.net/seafdav
 - username=<ZIMK-Kennung>@uni-trier.de
 - password=<das-oben-gesetzte-Pwd>
- Zugriff
 - im Browser: <https://seafolder.rlp.net/seafdav>
 - im Dateimanager Nautilus: davs://seafolder.rlp.net/seafdav

6. Durchstöbern Sie den versteckten Ordner .seafolder-data

7. Auch nützlich: Mobile Client für Android oder iOS installieren.

Zusammenfassung:

- Speichermedien und backup: Wo liegen letztlich die Daten physisch. Kriterien: Redundanz, Geografisch getrennt, regelmäßige Speicherungen etc.
- Dateiübertragungsprotokolle und Dateisynchronisation: Wie können Dateien zwischen zwei Computern übertragen werden? Wie können Datenbestände synchron gehalten werden? (smb, nfs, ftp, webdav, rsync,...)
- Umsetzung und Arbeiten auf dem Datenbestand: Wie wird letztlich auf den Datenbestand zugegriffen und damit gearbeitet? (sync client, browser-basierte Tools, share-Funktionalitäten,...)