This is my last summer semester 2024 at KIT

Raphael Sebastian Sutter

2024-09-08

Table of contents

Preface

Welcome to my tale of survival during the summer semester of 2024 at the Karlsruhe Institute of Technology (KIT). This book is a collection of stories, mishaps, and the occasional epiphany that I encountered while navigating the chaos of deadlines, lectures, while working full time at an internship and way too much coffee. If you're looking for inspiration, you might be in the wrong place—but if you want a good laugh at my expense, you're definitely in the right one. Here's to the ups, downs, and everything in between that made this semester unforgettable!

1 HOC

The smallest but coolest building blocks to fullfill all my credits for my Bachelors degree were the HOC courses. One of them was about Quarto

Quarto is an open-source scientific and technical publishing system that makes it easy to create high-quality documents, reports, presentations, and more. It's designed to be user-friendly, powerful, and flexible, allowing users to incorporate code, text, and multimedia content seamlessly.

1.1 Key Features

Here are some of the key features of Quarto:

- **Multi-format output**: You can render your documents into multiple formats, including HTML, PDF, and EPUB.
- **Integrated Code**: Quarto supports embedding code chunks in languages like Python, R, and Julia.
- **Markdown-based**: Quarto uses Markdown, making it easy to write and format text.
- **Cross-references**: Easily create cross-references for sections, figures, and tables.

1.2 Example of a Code Chunk

Quarto allows you to include code chunks directly in your document. Here's an example using Python:

```
import numpy as np
import matplotlib.pyplot as plt

r = np.arange(0, 2, 0.01)
theta = 2 * np.pi * r
fig, ax = plt.subplots(
  subplot_kw = {'projection': 'polar'}
)
ax.plot(theta, r)
```

```
ax.set_rticks([0.5, 1, 1.5, 2])
ax.grid(True)
plt.show()
```

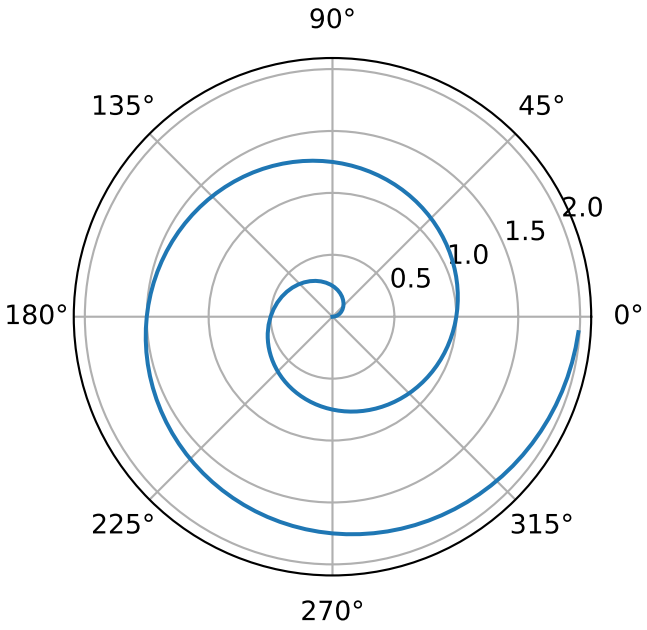

Figure 1.1: A line plot on a polar axis

1.3 Tables

The example table in the code displays a simple table with three columns. Each column contains the word "Data" repeated three times. The table is created using Markdown syntax, with each column separated by vertical bars (|). The table headers are specified in the first row, and the data is filled in subsequent rows.

Semester 1	Semester 2	Semester 3
Fun	Table	!
Let's	Have	Some
Fun	with	Code

2 Software Evolution

Software evolution refers to the process of modifying and improving software over time. It is an essential aspect of software development, as it allows software systems to adapt to changing requirements, fix bugs, and enhance functionality. In this quarto page, we will explore various techniques used in software evolution.

2.1 Version Control

One of the fundamental techniques in software evolution is version control. Version control systems, such as Git, enable developers to track changes made to the source code over time. By using version control, developers can easily collaborate, revert changes, and maintain a history of the software's evolution.

2.2 Refactoring

Refactoring is the process of restructuring existing code without changing its external behavior. It aims to improve the code's readability, maintainability, and performance. Common refactoring techniques include extracting methods, renaming variables, and eliminating code duplication. By refactoring code, developers can make it easier to understand and maintain, leading to more efficient software evolution.

2.3 Continuous Integration and Deployment

Continuous integration (CI) and continuous deployment (CD) are practices that automate the process of building, testing, and deploying software changes. CI ensures that changes made by multiple developers are integrated regularly, reducing the chances of conflicts and enabling faster feedback. CD automates the deployment of software changes to production environments, allowing for rapid and frequent releases. These practices facilitate the evolution of software by enabling quick iterations and reducing the time between development and deployment.

2.4 Testing

Testing is a crucial aspect of software evolution. It helps ensure that changes made to the software do not introduce new bugs or regressions. Different testing techniques, such as unit testing, integration testing, and acceptance testing, are used to validate the behavior and functionality of the software. By having comprehensive test suites, developers can confidently make changes to the software and ensure its stability and reliability.

2.5 Documentation

Documentation plays a vital role in software evolution. It provides information about the software's architecture, design decisions, and usage instructions. Well-documented software is easier to understand, maintain, and evolve. Techniques such as inline comments, code documentation, and user guides help developers and users navigate the software's evolution and make informed decisions.

In conclusion, software evolution is a continuous process that involves various techniques to modify, improve, and maintain software over time. Version control, refactoring, continuous integration and deployment, testing, and documentation are some of the key techniques used in software evolution. By employing these techniques, developers can ensure the software's adaptability, stability, and longevity.

3 Numerik

Numerical mathematics, also known as numerical analysis, is a branch of mathematics that focuses on developing algorithms and methods for solving mathematical problems using numerical approximations. In this section, we will explore some key concepts and techniques in numerical mathematics.

3.1 Root Finding

One important problem in numerical mathematics is finding the roots of a function. The root of a function is a value that makes the function equal to zero. There are several methods for root finding, such as the bisection method, Newton's method, and the secant method.

3.1.1 Bisection Method

The bisection method is a simple and robust method for finding roots. It works by repeatedly dividing an interval in half and narrowing down the interval until a root is found. Here's an example implementation in Python:

3.2 Upper Lower Matrix Method

The Upper Lower Matrix Method is a technique used to solve systems of linear equations. It involves decomposing a matrix into an upper triangular matrix and a lower triangular matrix. This decomposition allows for efficient and straightforward solutions to linear systems.

To illustrate the Upper Lower Matrix Method, let's consider a system of linear equations: ## Die LR-Zerlegung

Die LR-Zerlegung (auch LU-Zerlegung) ist eine Methode zur Zerlegung einer quadratischen Matrix in zwei Matrizen: eine untere Dreiecksmatrix **L** und eine obere Dreiecksmatrix **R** (oder **U**). Diese Methode wird häufig in der numerischen Mathematik zur Lösung von Gleichungssystemen, zur Inversion von Matrizen und zur Bestimmung der Determinante verwendet.

3.3 Definition

Gegeben eine Matrix **A**, kann die LR-Zerlegung wie folgt geschrieben werden:

$$A = L \cdot R$$

Dabei ist:

- **L** eine untere Dreiecksmatrix mit Einsen auf der Diagonalen.
- **R** eine obere Dreiecksmatrix.

3.3.1 Beispiel

Betrachten wir eine Matrix **A**:

$$A = \begin{pmatrix} 2 & 3 & 1 \\ 4 & 7 & 2 \\ 6 & 18 & 5 \end{pmatrix}$$

Die LR-Zerlegung von **A** ergibt:

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 3 & 1 \end{pmatrix}, \quad R = \begin{pmatrix} 2 & 3 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

3.4 Berechnung der LR-Zerlegung

Die LR-Zerlegung kann mit verschiedenen numerischen Verfahren berechnet werden. Eine der am häufigsten verwendeten Methoden ist der Gaußsche Eliminationsprozess.

```
import numpy as np
from scipy.linalg import lu

# Definieren der Matrix A
A = np.array([[2, 3, 1],
              [4, 7, 2],
              [6, 18, 5]])

# Durchführung der LR-Zerlegung
P, L, U = lu(A)
```

```
print("L =\n", L)
print("R =\n", U)
```

```
L =
 [[1.         0.          0.        ]
 [0.66666667 1.         0.        ]
 [0.33333333 0.6        1.        ]]
R =
 [[ 6.         18.          5.        ]
 [ 0.         -5.         -1.33333333]
 [ 0.          0.          0.13333333]]
```

4 Rechnerstrukturen

Die Rechnerstrukturen und die Produktion von Hardware Chips sind zwei wichtige Aspekte der Computertechnologie. In diesem Artikel werden wir einen Überblick über diese Themen geben.

4.1 Definition

Rechnerstrukturen beziehen sich auf die Architektur und Organisation von Computern. Sie umfassen die verschiedenen Komponenten eines Computersystems, wie Prozessor, Speicher, Eingabe- und Ausgabegeräte sowie die Verbindungen zwischen ihnen.

4.1.1 Von-Neumann-Architektur

Die Von-Neumann-Architektur ist eine der grundlegendsten Arten von Rechnerstrukturen. Sie besteht aus vier Hauptkomponenten:

1. **Speicher:** Speichert Daten und Programme.
2. **Rechenwerk (ALU):** Führt arithmetische und logische Operationen durch.
3. **Steuerwerk:** Interpretiert Befehle und steuert die anderen Komponenten.
4. **Eingabe-/Ausgabeschnittstellen:** Ermöglichen die Interaktion mit dem Computer.

> **Note**
>
> Die Von-Neumann-Architektur ist besonders bekannt für ihre Einfachheit und die Tatsache, dass sowohl Programme als auch Daten im selben Speicher gespeichert werden.

4.1.2 Harvard-Architektur

Die Harvard-Architektur unterscheidet sich dadurch, dass sie separate Speicher für Programme und Daten verwendet, was parallele Datenverarbeitung erleichtert und die Effizienz erhöht.

4.1.3 Diagramm: Datenfluss in einer einfachen Computerarchitektur

1. Program Memory (Programmspeicher) sendet Daten an die CPU.
2. Data Memory (Datenspeicher) sendet ebenfalls Daten an die CPU.
3. Die CPU verarbeitet die Daten und sendet die Ergebnisse an die Ausgabe.

4.1.4 Textdiagramm:

Program Memory –> CPU Data Memory ——> CPU | v Output

4.2 Interaktive Visualisierung mit Plotly

import plotly.express as px import pandas as pd

df = pd.DataFrame({ “x”: [1, 2, 3, 4, 5], “y”: [10, 15, 13, 17, 14], “z”: [“A”, “B”, “C”, “D”, “E”] })

fig = px.scatter(df, x=“x”, y=“y”, text=“z”, title=“Interaktive Plotly Grafik”) fig.update_traces(marker=dict(s color=“LightSkyBlue”), selector=dict(mode=‘markers’))

fig.show()

5 Next

Figure 5.1: Uni Zürich