Pharmacokinetic & pharmacodynamic modelling

Matthias König

2025-04-09

Table of contents

Preface

Welcome to the Course: **Pharmacokinetic and pharmacodynamic modelling** available from https://matthiaskoenig.github.io/dhpe-pkpd.

Have you ever wondered how medications travel through the human body, or how we determine the right dose for each patient? Understanding drug movement and action isn't just for pharmacologists—it's a key competency in modern healthcare and biomedical research.

This course invites you to explore the fascinating world of **pharmacokinetics (PK)**—how drugs are absorbed, distributed, metabolised, and excreted—and **pharmacodynamics (PD)**—how drugs produce their effects. You'll gain practical insights into the models that help us simulate and predict drug behavior in the body, from simple compartment models to advanced physiologically based models used in cutting-edge research and clinical decision-making.

https://www.youtube.com/embed/HPIw1wDLmaw

Through interactive content and hands-on examples, you'll learn how PK/PD modelling plays a critical role in:

- Understanding variability in patient responses
- Optimising drug dosing
- Evaluating drug-drug interactions

Built with open technologies and aligned with best practices in digital health education, this course empowers you to become part of a growing community of learners and educators who believe in transparency, reproducibility, and innovation.

Ready to begin? Dive in and transform the way you think about medicine.

Part I

Concept

1 Teaching concept

1.1 General idea

This open, practice-oriented course introduces the fundamentals of pharmacokinetic (PK) and pharmacodynamic (PD) modelling, with a strong emphasis on reproducible research, open science, and digital competencies in health professions education. Leveraging best practices from software development—such as version control via Git and GitHub, continuous integration (CI) for automated course building, and open formats like Markdown—the course is designed as a fully open educational resource. Hosted on the Open edX platform, it combines theoretical foundations with interactive, hands-on learning to equip participants with the skills needed to understand, simulate, and critically assess PK/PD models in clinical and research contexts.

The course consists of the following complementary parts which should allow as much offline studying as possible:

1. **Short lectures** about pharmacokinetics in the form of a book chapter providing the content. The book is available as HTML webpage, PDF, Epub and HTML slideshow. Quarto is used to render high quality books with professional figures, layouts, full text search, cross references, references, citation, and the possibiliy to compile to multiple output formats.
2. **Short presentations** for motivation and providing up to date use cases in the form of short videos and accompanying slides.
3. An **Open edX course** for the respective chapter with interactive content (quizzes, H5P components).
4. **Interactive Jupyter notebooks and simple apps** for active exploration of the course content. See for instance the indocyanine green application.

The course was developed as part of the Digital Health Professions Educator 2024/2025. The course is the digital teaching project of phase III (see Figure 1.1.)

1.2 Technology

In this section we provide an overview of the technology behind the course. The following figure provides an overview of the workflow of the course.

Figure 1.1: Overview of the Digital Health Professions Educator (DHPE) program.

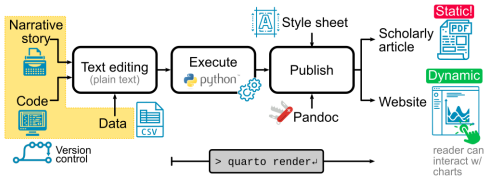


Figure 1: Course writing workflow, starting from plain text (narrative, code and data) all under version control for reproducibility.

1.2.1 Version control

Version control is a system that tracks changes to files over time, enabling users to manage revisions, collaborate efficiently, and maintain a complete history of their work. It allows users to revert to previous versions, compare changes, and resolve conflicts when multiple contributors edit the same content. Version control is essential for collaborative projects, ensuring transparency, consistency, and reproducibility.

In this course, all teaching materials are managed using version control through a GitHub repository: https://github.com/matthiaskoenig/dhpe-pkpd/. The content is written in Quarto QMD markdown files, and all updates made locally are systematically synchronized and pushed to the repository. This setup ensures that any changes to the course materials are tracked, making it easy to review updates, maintain a clean history of edits, and collaborate across contributors. Version control offers significant advantages for creating teaching materials: it supports iterative improvement, minimizes the risk of losing work, and provides a reliable way to roll back to previous versions if needed—all of which enhance the quality and reproducibility of educational content.

1.2.2 Continous integration (CI)

Continuous Integration (CI) is a development practice where changes to a project are automatically tested and integrated on a regular basis. CI helps identify issues early, ensures consistency across updates, and automates repetitive tasks such as building, testing, and deploying code. This leads to faster development cycles, higher quality output, and greater confidence in the stability of the project.

For this course, CI is implemented using GitHub Actions to automatically build and deploy the course content. Whenever changes are made to the repository, the content is rebuilt and published to the course webpage at https://matthiaskoenig.github.io/pkpd. This automated workflow ensures the site is always up to date with the latest teaching materials, improving reliability, reproducibility, and ease of maintenance.

1.2.3 Quarto

- Notebook and markdown based publishing tools
- Quarto is an open-source scientific and technical publishing system available from https://quarto.org/.
- Publish reproducible, production quality articles, presentations, dashboards, websites, blogs, and books in HTML, PDF, MS Word, ePub, and more.
- For a nice introduction to the features see: https://gael-close.github.io/posts/2209-tech-writing/2209-tech-writing.html

1.2.3.1 Interactivity

- Quarto supports interactivity via different methods: https://quarto.org/docs/interactive/
- Create custom JavaScript visualizations using Observable JS: https://quarto.org/docs/interactive/ojs/
- Incorporate Jupyter Widgets: https://ipywidgets.readthedocs.io/en/latest/
- Shiny for Python integration: https://quarto.org/docs/dashboards/interactivity/shiny-python/index.html

1.2.4 Jupyter notebooks

Jupyter Notebooks are interactive documents that combine code, text, and visual outputs, making them ideal for exploring and teaching computational topics in an engaging and reproducible way.

For this course, multiple sets of notebooks are generated to support different learning needs: content and teaching notebooks are rendered to the course webpage, while separate exercise notebooks offer hands-on practice, and solution notebooks provide detailed reference material.

1.2.5 reveal.js

- presentations are created with reveal.js
- reveal.js is an open source HTML presentation framework. It's a tool that enables anyone with a web browser to create fully-featured and beautiful presentations for free.
- Presentations made with reveal.js are built on open web technologies. That means anything you can do on the web, you can do in your presentation. Change styles with CSS, include an external web page using an `iframe` or add your own custom behavior using our JavaScript API.

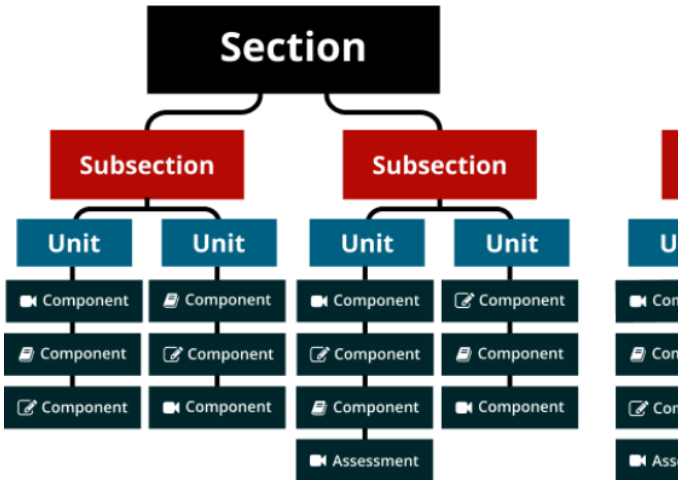
- The framework comes with a broad range of features including nested slides, Markdown support, Auto-Animate, PDF export, speaker notes, LaTeX support and syntax highlighted code.
- https://revealjs.com/course/
- nice integration with Quarto; https://quarto.org/docs/presentations/revealjs/demo/#/title-slide
- https://github.com/quarto-dev/quarto-web/blob/main/docs/presentations/revealjs/demo/index.qmd

1.2.6 OpenEdx

- https://openedx.org/
- Enable online campuses, instructor-led courses, degree programs, and self-paced courses using a single platform.
- Open edX is a thriving open source project, backed by a great community, for running an online learning platform at scale. Open edX comes with an LMS (Learning Management System) where students access course contents, a CMS (Content Management System) that course staff uses to design courses, and a few other components to provide more services to students, course staff, and platform administrators.

Allows integration of jupyter notebooks, H5P, ...

1.2.6.1 Course structure



The course is structured in sections, subsections and units.

1.2.6.2 Deployment

- Deployment based on docker and tutor
- OpenEDX is hosted using the de.NBI cloud (or HU resourses) via OpenStack

1.2.7 Generative AI

Generative AI refers to a type of artificial intelligence designed to create new content, such as text, images, music, or code, by learning patterns from existing data. It uses models like neural networks to generate outputs that resemble human creativity and reasoning.

All course content is supported by ChatGPT. Initial drafts are created, and materials are continuously updated and refined using generative AI.

1.2.8 Video recording (OBS Studio)

OBS Studio (Open Broadcaster Software) is a free, open-source software for video recording and live streaming. It allows users to capture and mix video/audio in real time with customizable scenes, sources, and transitions. Popular among streamers, educators, and content creators, OBS supports platforms like YouTube, Twitch, and Zoom.

Videos in this course are recorded using OBS Studio and subsequently hosted on youtube.

Part II

Course

2 Introduction

Welcome to the course **Introduction to pharmacokinetic and pharmacodynamic modelling**. Here we provide an overview of the course content and the topics which will be covered.

2.1 Summary

Pharmacokinetic modelling is the study of how drugs are absorbed, distributed, metabolised and excreted in the body. The pharmacokinetic modelling course covers topics such as pharmacokinetic principles, drug distribution, clearance and elimination and the factors that influence these processes. Students will learn about different models used to describe pharmacokinetics, such as compartmental models and physiologically based pharmacokinetic models, and how these models can be used to predict drug concentrations and optimise dosing regimens. Other topics that may be covered include pharmacodynamics, drug-drug interactions and the use of pharmacokinetic modelling in drug development and clinical practice. Overall, a course in pharmacokinetic modelling will provide students with a comprehensive understanding of the principles and techniques used to describe the movement of drugs through the body and how this knowledge can be applied to improve drug therapy.

> **i** Note: Help us improve the course
>
> This course is a work in progress, and we're always looking for ways to make it better. If you have suggestions, ideas, or feedback on what to improve or how to enhance your learning experience, please don't hesitate to get in touch. Your input is valuable and appreciated!

2.2 Topics

The following topics will be covered in the course:

- **Structural Models** (Chapter 3) Introduction to structural pharmacokinetic/pharmacodynamic (PK/PD) models, including model types (e.g., one- and multi-compartment), assumptions, and diagrammatic representation.

- **Ordinary Differential Equations (ODEs)** (**?@sec-ode**) Fundamentals of ODEs as the mathematical foundation of dynamic modeling in pharmacology; solving and interpreting ODEs in PK/PD contexts.
- **Compartment Model** (**?@sec-compartment-models**) Detailed exploration of compartmental models in pharmacokinetics, including one-, two-, and multi-compartment models with drug distribution and elimination.
- **Absorption** (**?@sec-absorption**) Modeling drug absorption processes including first-order and zero-order kinetics, and factors affecting bioavailability.
- **Multiple Dosing** (**?@sec-compartment-models**) Modeling and simulation of repeated drug administration; concepts such as steady-state concentration and accumulation.
- **Metabolism** (**?@sec-metabolism**)
  Representation of drug metabolism in models; modeling hepatic clearance and metabolite formation.
- **Pharmacokinetic Parameters** (**?@sec-pharmacokinetic-parameters**) Derivation and interpretation of key PK parameters such as clearance (CL), volume of distribution (Vd), half-life (t½), and area under the curve (AUC).
- **Variability** (**?@sec-variability**) Introduction to interindividual and intraindividual variability in PK/PD; sources of variability and population modeling concepts.
- **SBML (Systems Biology Markup Language)** (**?@sec-sbml**) Working with SBML for model exchange and simulation; structure of SBML files and integration with modeling tools.
- **Pharmacodynamics** (**?@sec-pharmacodynamics**) Modeling drug effects on the body; PD models such as Emax, sigmoid Emax, and indirect response models.
- **PBPK Tutorial (Physiologically Based Pharmacokinetic Modeling)** (**?@sec-pbpk**) Introduction to PBPK modeling with tutorial examples; structure, parameters, and application in translational and personalized medicine.

Tip: Customize Your Learning Path

The course does not have to be done linearly—you can explore the content in any order. Choose the learning path that best suits your goals, interests, or current knowledge level. This way, you stay engaged and make the most of your learning experience.

2.3 Learning Objectives

The main learning objectives are

- Understand key concepts in pharmacokinetics and pharmacodynamics.

- Learn to formulate and analyze models using differential equations.
- Develop and simulate compartment and PBPK models.
- Model drug absorption, metabolism, and multiple dosing scenarios.
- Interpret core PK parameters (e.g., clearance, half-life, AUC).
- Understand interindividual variability in drug response.
- Use SBML for model sharing and interoperability.
- Learn basic computational skills for model simulation and analysis.
- Apply PD models to quantify drug effects.

Tip: If you are interested in the technology behind this course

If you're curious about the technology powering this course, check out the technology overview in Chapter 1.2. For a deeper dive into how everything works behind the scenes, we've provided more detailed information Chapter 29.

2.4 Open edX

The Open edX course is available from http://apps.local.openedx.io.

Tip: This will not work for you

The Open edX platform is currently only running locally and we are working on deploying it asap. Please be patient.

2.5 Resources

A lot of great resources exist on the topic. Below we provide a few recommendations to get started. Some of the material was used as inspiration for the course

3 Structural Models as Algebraic Equations

A pharmacokinetic (PK) model can be simply represented by an algebraic equation. For instance, a one-compartment model with a single intravenous bolus dose can be described by the following equation:

$$C(t) = \frac{Dose}{V} e^{-\frac{CL}{V} \cdot t} \tag{3.1}$$

In this model, the equation defines the relationship between the independent variable, time `t`, and the dependent variable, concentration `C`. The notation `C(t)` indicates that `C` is a function of `t`. The parameters — Dose, clearance (CL), and distribution volume (V) — are constants and do not vary with time.

3.1 Understanding Variables and Parameters

It's important to distinguish between variables and parameters. Variables (dependent and independent) are used to extract information from the equation. In PK modeling, time is often the independent variable, but the equation can be rearranged for different analyses, such as using CL as the independent variable for sensitivity analysis, with time as a constant.

3.2 Exponential Decay and Curve Interpretation

This algebraic equation results in an exponential decay curve, representing the drug concentration over time. This type of curve is crucial for understanding how a drug is metabolized and eliminated from the body.

3.3 Practical Applications

PK modeling has several practical applications: - **Drug Development:** It helps in determining the optimal dosing regimen. - **Clinical Pharmacology:** It aids in understanding how

different patient factors (e.g., age, weight, renal function) affect drug kinetics. - **Personalized Medicine:** It supports tailoring drug therapy to individual patient needs, enhancing therapeutic outcomes while minimizing side effects.

3.4 Key Concepts

1. **One-Compartment Model:** Assumes the body acts as a single, homogeneous compartment where the drug distributes instantly.
2. **Intravenous Bolus Dose:** Refers to the entire dose of the drug administered at once into the bloodstream.
3. **Clearance (CL):** A measure of the body's efficiency in eliminating the drug.
4. **Volume of Distribution (V):** A theoretical volume that relates the amount of drug in the body to the concentration in the blood or plasma.

3.5 Beyond the Basics

While this equation represents a simplified model, more complex models exist that account for multiple compartments, non-linear kinetics, and various routes of administration (e.g., oral, subcutaneous). Advanced PK models can incorporate biological factors and mechanisms, such as enzyme kinetics and receptor binding.

4 Simulation of an Algebraic Equation

In the first task, we aim to use the algebraic equation to simulate the pharmacokinetics of a drug over time. We will start with a model of warfarin, a widely used anticoagulant.

4.1 Background on Warfarin

Warfarin is a commonly prescribed oral anticoagulant that helps prevent blood clots. It is often used in the treatment and prevention of deep vein thrombosis (DVT), pulmonary embolism (PE), and in patients with atrial fibrillation to reduce the risk of stroke. Warfarin works by inhibiting the synthesis of vitamin K-dependent clotting factors, which are essential for blood coagulation.

4.2 Importance of Warfarin

Warfarin is crucial in managing conditions that predispose individuals to thromboembolic events. However, it has a narrow therapeutic index, meaning that small changes in its concentration can lead to significant differences in therapeutic effect and toxicity. Monitoring and adjusting the dose of warfarin is essential to ensure its efficacy while minimizing the risk of bleeding complications.

4.3 Unique Pharmacokinetics of Warfarin

Warfarin's pharmacokinetics are complex due to several factors: - **Variable Absorption:** Although it is generally well-absorbed, factors such as diet and gastrointestinal health can affect its absorption. - **Protein Binding:** Warfarin is highly bound to plasma proteins, particularly albumin, which influences its distribution and free (active) concentration in the blood. - **Metabolism:** Warfarin is metabolized by the liver, primarily through the cytochrome P450 enzyme system. Genetic variations in these enzymes can lead to significant interindividual differences in drug clearance. - **Long Half-Life:** Warfarin has a long half-life, typically ranging from 20 to 60 hours, leading to a steady-state being reached only after several days of consistent dosing.

4.4 Simulation Task

To simulate the pharmacokinetics of warfarin using the algebraic equation, we need values for the pharmacokinetic parameters: clearance (CL) and volume of distribution (Vd).

4.4.1 Parameters for Warfarin

- **Clearance (CL):** The rate at which warfarin is removed from the body. For warfarin, typical clearance values range from 0.1 to 0.2 L/hour.
- **Volume of Distribution (Vd):** The theoretical volume in which the total amount of drug would need to be uniformly distributed to produce the observed blood concentration. For warfarin, Vd is approximately 10 L.

4.4.2 Simulation Procedure

1. **Select a Dose:** Choose a typical dose of warfarin, such as 10 mg, administered as a single intravenous bolus dose.
2. **Set Parameters:** Use the provided values for CL and Vd.
3. **Apply the Equation:** Use the one-compartment model equation to simulate the concentration of warfarin over time:

$$C(t) = \frac{Dose}{V} e^{-\frac{CL}{V} \cdot t} \tag{4.1}$$

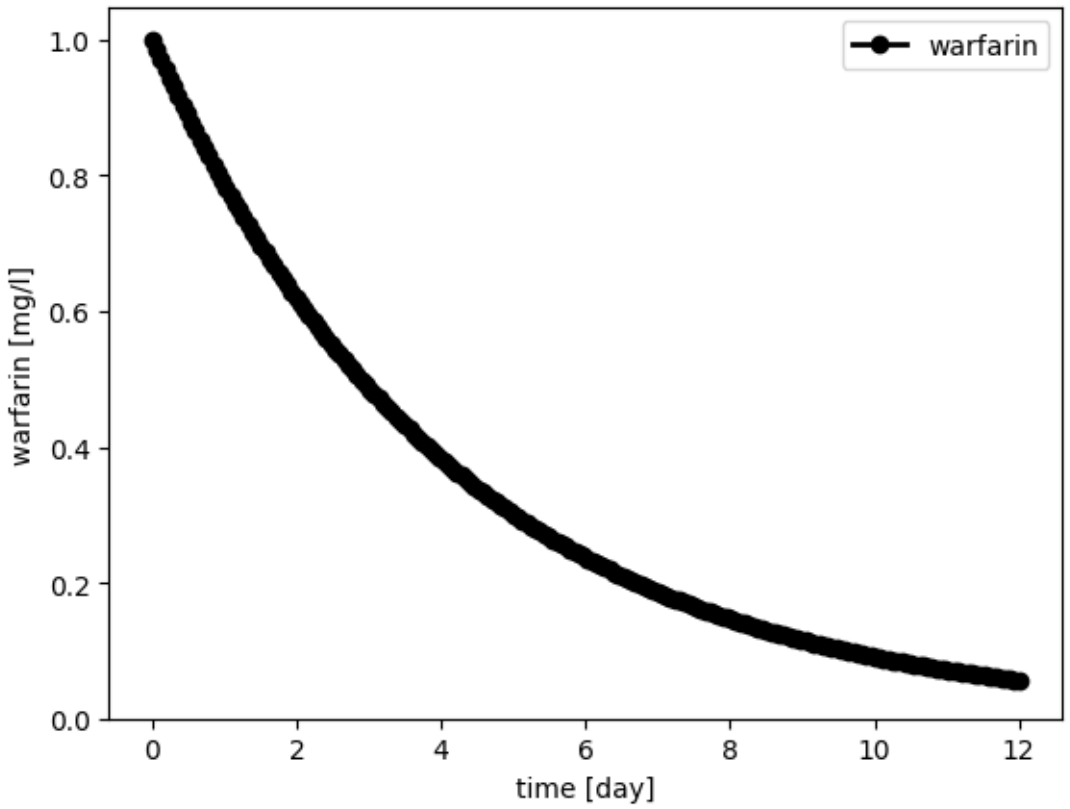
4. **Plot the Concentration Curve:** Generate a plot of warfarin concentration (C) versus time (t) to visualize the drug's pharmacokinetics.

4.5 Practical Insight

This simulation will help students understand how warfarin's concentration changes over time after administration, providing insights into its dosing regimen and the importance of monitoring to maintain therapeutic levels. Additionally, it highlights the significance of individual patient factors in determining the appropriate dose and frequency of administration.

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

# simulate
V = 10  # [l]
CL = 0.1  # [L/hr]
```

```
Dose = 10  # [mg]
t = np.linspace(start=0, stop=12*24, num=200)  # [hr]
C = Dose/V * np.exp(-CL/V * t)  # [mg/l]

# plot
f, ax = plt.subplots()
ax.plot(t/24.0, C, label="warfarin", color="black", linewidth=2.0, marker="o")
ax.set_xlabel("time [day]")
ax.set_ylabel("warfarin [mg/l]")
ax.set_ylim(bottom=0)
ax.legend()
plt.show()
```



5 Adaptation to Other Drugs

In this step, we will adapt the parameters (CL) and (Vd) to simulate the intravenous injection of other drugs, such as aspirin and chloroquine.

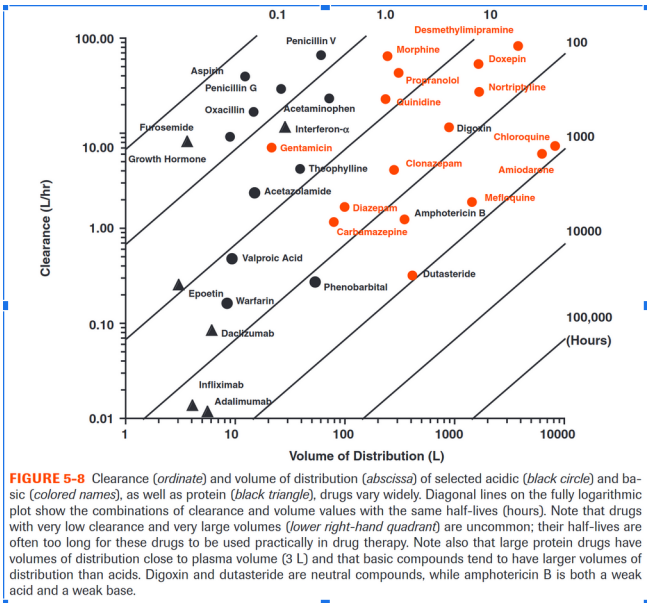


FIGURE 5–8 Clearance (*ordinate*) and volume of distribution (*abscissa*) of selected acidic (*black circle*) and basic (*colored names*), as well as protein (*black triangle*), drugs vary widely. Diagonal lines on the fully logarithmic plot show the combinations of clearance and volume values with the same half-lives (hours). Note that drugs with very low clearance and very large volumes (*lower right-hand quadrant*) are uncommon; their half-lives are often too long for these drugs to be used practically in drug therapy. Note also that large protein drugs have volumes of distribution close to plasma volume (3 L) and that basic compounds tend to have larger volumes of distribution than acids. Digoxin and dutasteride are neutral compounds, while amphotericin B is both a weak acid and a weak base.

Figure 5.1: Clearance vs Vd

5.1 Background on Aspirin and Chloroquine

Aspirin: Aspirin (acetylsalicylic acid) is a widely used medication with analgesic, anti-inflammatory, and antipyretic properties. It is also commonly used for its antiplatelet effect to prevent cardiovascular events such as heart attacks and strokes. Aspirin works by inhibiting the enzyme cyclooxygenase, which plays a key role in the synthesis of prostaglandins and thromboxanes.

Chloroquine: Chloroquine is an antimalarial drug that has also been used to treat autoimmune diseases such as rheumatoid arthritis and lupus. It functions by interfering with the growth of parasites in red blood cells. Chloroquine has a complex pharmacokinetic profile due to its extensive tissue distribution and long half-life.

5.2 Variability in Volume of Distribution and Clearance

The volume of distribution (Vd) and clearance (CL) of drugs can vary significantly between different medications due to factors such as: - **Drug Properties:** Lipophilicity, molecular size, and plasma protein binding affect how a drug distributes in the body. - **Patient Characteristics:** Age, weight, organ function (especially liver and kidneys), and genetic factors can influence drug metabolism and elimination. - **Pathophysiological Conditions:** Diseases and conditions affecting the liver, kidneys, and cardiovascular system can alter pharmacokinetic parameters.

Understanding this variability is crucial for tailoring drug therapy to individual patients and ensuring optimal therapeutic outcomes while minimizing adverse effects.

5.3 Exercises

5.3.1 Exercise 1: Solve the Algebraic Equation for Aspirin or Chloroquine

Using the respective CL and Vd parameters from the figure, solve the algebraic equation for the concentration of aspirin or chloroquine over time.

Aspirin Example Parameters: - **Clearance (CL):** Approximately 50 L/hour - **Volume of Distribution (Vd):** Approximately 10 L

Chloroquine Example Parameters: - **Clearance (CL):** Approximately 10 L/hour - **Volume of Distribution (Vd):** Approximately 5000 L

5.3.2 Exercise 2: Change the Dose and Observe the Effects

Modify the dose of the drug and observe the resulting effects on the concentration-time curves. Consider the following doses: - **Aspirin:** 500 mg, 1000 mg, 1500 mg - **Chloroquine:** 200 mg, 400 mg, 600 mg

Plot the concentration curves for each dose and analyze how changes in dosage affect the drug's pharmacokinetics.

5.4 Practical Insight

These exercises will help you understand the impact of different pharmacokinetic parameters on drug behavior in the body. By comparing the profiles of aspirin and chloroquine, you can appreciate the importance of individualized dosing and the challenges associated with drugs that have large volumes of distribution or low clearance rates.

6 Parameter Scans

To systematically study the effects of various parameters on the pharmacokinetic model, we can perform parameter scans. This involves changing individual parameters systematically to observe their impact on the drug concentration-time profile.

6.1 Background and Relevance

Parameter scans are a crucial tool in pharmacokinetic modeling. They allow us to understand how changes in dosing, volume of distribution, and clearance affect drug behavior. This knowledge is essential for optimizing drug dosing regimens, predicting therapeutic outcomes, and identifying potential risks of adverse effects.

By performing parameter scans, researchers and clinicians can: - **Optimize Dosing Strategies:** Determine the most effective and safe doses for different patient populations. - **Understand Drug Behavior:** Gain insights into how drugs distribute, metabolize, and eliminate from the body. - **Tailor Therapies:** Customize treatments based on individual patient characteristics and conditions. - **Predict Outcomes:** Forecast the effects of parameter variations, such as those due to drug interactions or genetic differences.

6.2 Dose Dependency

6.2.1 Concept

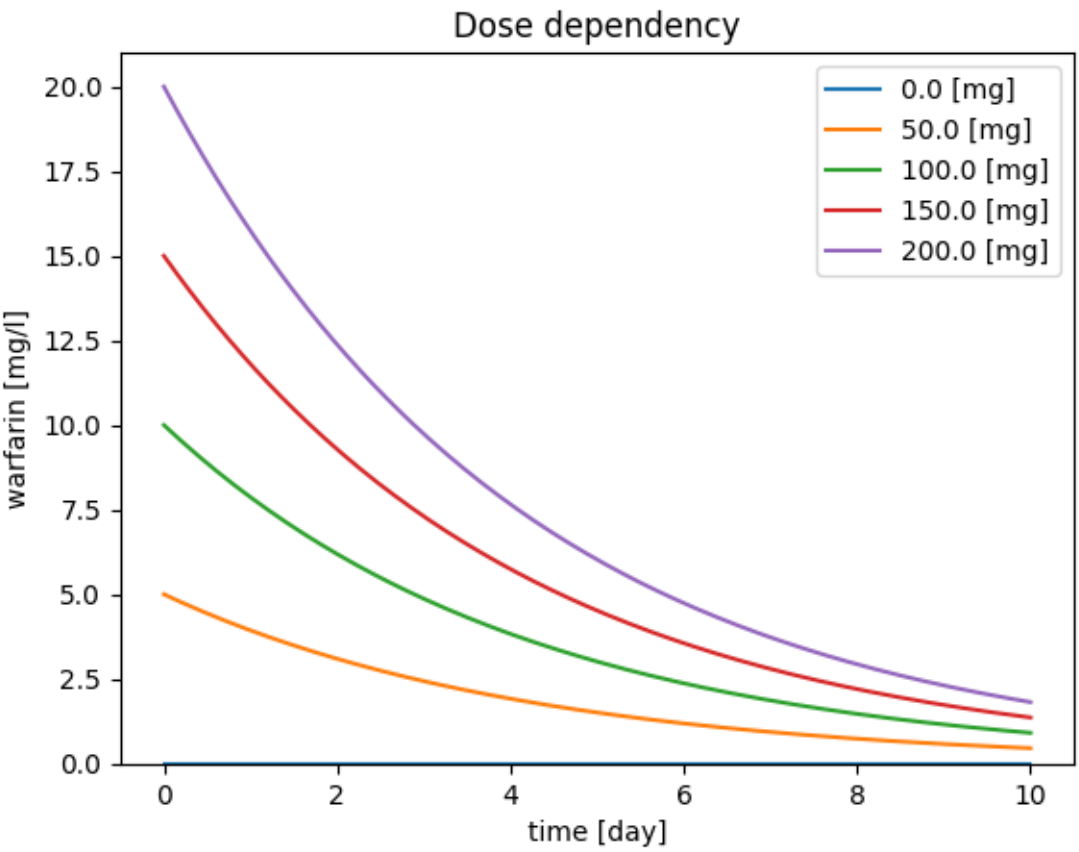
Dose dependency refers to how changes in the administered dose of a drug affect its concentration over time. Understanding dose dependency helps in determining the appropriate dose that achieves the desired therapeutic effect without causing toxicity.

6.2.2 Exercise

1. **Select a Range of Doses:** Choose a range of doses for the drug being studied (e.g., 0 mg, 50 mg, 100 mg, 150 mg, 200 mg).
2. **Simulate Concentration-Time Profiles:** Use the pharmacokinetic equation to simulate the concentration-time profiles for each dose.

3. **Analyze the Results:** Plot the concentration versus time for each dose and analyze how the peak concentration (Cmax) and area under the curve (AUC) change with different doses.

```
# warfarin
V = 10  # [l]
CL = 0.1  # [L/hr]
Dose = 100  # [mg]
t = np.linspace(start=0, stop=10*24, num=200)  # [hr]

# Dose dependency
f, ax = plt.subplots(nrows=1, ncols=1)
for Dose in np.linspace(0, 200, num=5):
    C = Dose / V * np.exp(-CL / V * t)  # [mg/l]
    ax.plot(t/24.0, C, label=f"{Dose} [mg]")
# reset dose
Dose = 100  # [mg]

# plot
ax.set_xlabel("time [day]")
ax.set_ylabel("warfarin [mg/l]")
ax.set_ylim(bottom=0)
ax.legend()
ax.set_title("Dose dependency")
plt.show()
```



6.3 Volume of Distribution Dependency

6.3.1 Concept

The volume of distribution (Vd) is a theoretical volume that relates the amount of drug in the body to its concentration in the blood or plasma. It provides insight into how extensively a drug disperses into body tissues. Changes in Vd can significantly affect drug concentration profiles, especially for drugs that distribute widely into tissues.

6.3.2 Exercise

1. **Select a Range of Vd Values:** Choose a range of Vd values for the drug (e.g., 5 L, 10 L, 20 L).
2. **Simulate Concentration-Time Profiles:** Use the pharmacokinetic equation to simulate the concentration-time profiles for each Vd value.

3. **Analyze the Results:** Plot the concentration versus time for each Vd value and observe how the distribution volume influences the drug's concentration, half-life, and overall exposure.

6.4 Practical Insight

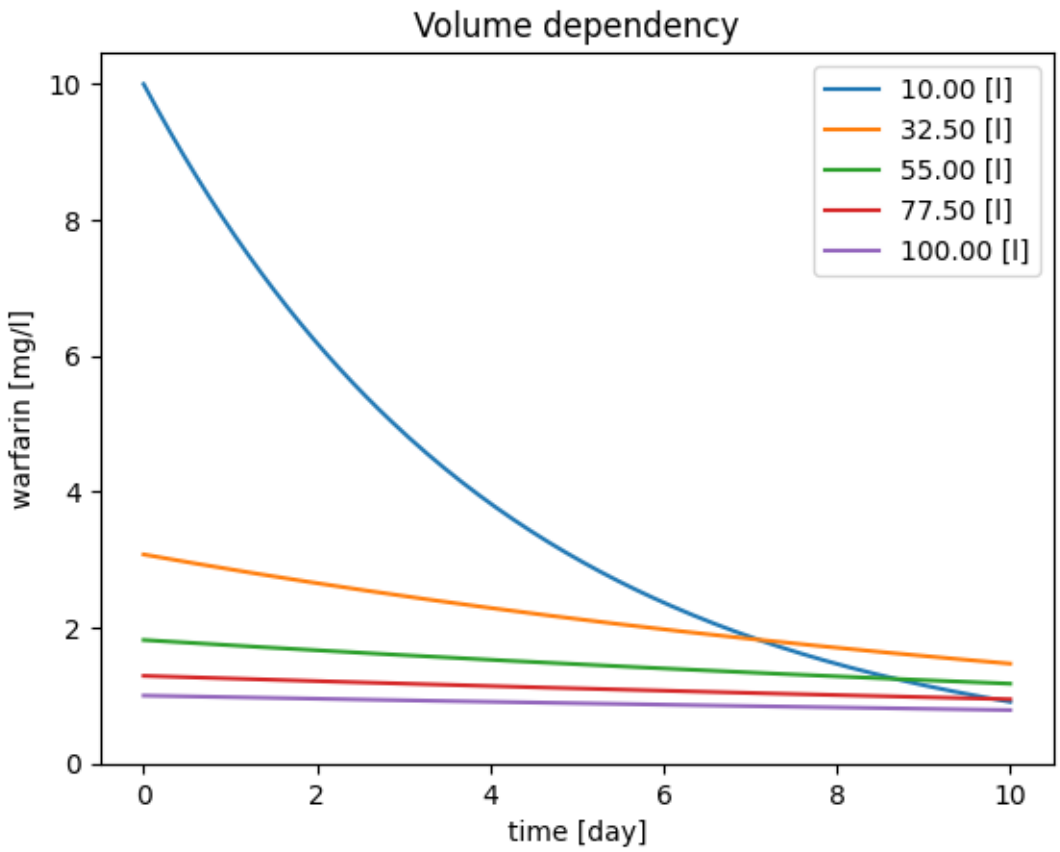
By performing parameter scans on dose and volume of distribution, students can gain a deeper understanding of the dynamic nature of pharmacokinetics. These exercises will illustrate: - **Non-Linear Relationships:** How parameters do not always change drug concentration linearly. - **Therapeutic Window:** The importance of maintaining drug concentrations within a therapeutic range. - **Individual Variability:** How patient-specific factors might necessitate adjustments in dosing or consideration of different pharmacokinetic profiles.

6.5 Example Applications

- **Dose Adjustment:** For patients with renal or hepatic impairment, dose adjustments may be required to avoid toxicity.
- **Drug Development:** In the early stages of drug development, parameter scans help in predicting human pharmacokinetics based on preclinical data.
- **Personalized Medicine:** Tailoring doses based on genetic makeup and other patient-specific factors to achieve optimal therapeutic outcomes.

By understanding and applying these concepts, students can appreciate the complexities involved in pharmacokinetic modeling and the importance of personalized approaches in clinical pharmacology.

```
# V dependency
f, ax = plt.subplots(nrows=1, ncols=1)
for V in np.linspace(10, 100, num=5):
    C = Dose / V * np.exp(-CL / V * t)  # [mg/l]
    ax.plot(t/24.0, C, label=f"{V:.2f} [l]")
# reset volume
V = 10  # [mg]

ax.set_xlabel("time [day]")
ax.set_ylabel("warfarin [mg/l]")
ax.set_ylim(bottom=0)
ax.legend()
ax.set_title("Volume dependency")
plt.show()
```



Exercise: Implement a Parameter Scan for Clearance (CL)

Objective: To understand how changes in clearance (CL) affect the pharmacokinetics of a drug, implement a parameter scan for CL values ranging from 0.1 to 3 L/hour.

7 Ordinary Differential Equations and Numerical Integration

7.1 Introduction

In pharmacokinetics, many models describing the behavior of drugs within the body are represented by ordinary differential equations (ODEs). These equations capture the dynamic changes in drug concentration over time, accounting for various processes such as absorption, distribution, metabolism, and excretion.

7.2 Background

7.2.1 Ordinary Differential Equations (ODEs)

An ODE is an equation that involves a function and its derivatives. In pharmacokinetics, ODEs describe how the concentration of a drug changes with respect to time due to different biological processes. The simplest form of an ODE in pharmacokinetics is a first-order equation representing the rate of change of drug concentration:

$$\frac{dC(t)}{dt} = -k \cdot C(t) \tag{7.1}$$

where: - C(t) is the concentration of the drug at time t. - k is the rate constant for the process being described (e.g., elimination).

More complex models may involve multiple compartments and multiple rate constants, leading to systems of ODEs.

7.2.2 Numerical Integration

In many cases, ODEs cannot be solved analytically, especially when dealing with complex pharmacokinetic models. Numerical integration techniques are employed to approximate the solutions of these equations. Methods such as Euler's method, the Runge-Kutta method, and more advanced algorithms provide approximate solutions by discretizing time and iteratively solving the equations.

7.3 Importance in Pharmacokinetics

Understanding and solving ODEs is crucial in pharmacokinetics for several reasons: - **Modeling Drug Behavior:** Accurately predict how drugs move through different compartments of the body over time. - **Dose Optimization:** Determine the optimal dosing regimen to achieve therapeutic levels without causing toxicity. - **Predicting Drug Interactions:** Understand how multiple drugs may interact and affect each other's pharmacokinetics. - **Personalized Medicine:** Tailor drug treatments based on individual patient characteristics and responses.

7.4 Applications

- **One-Compartment Models:** Describing the elimination of a drug from a single compartment.
- **Multi-Compartment Models:** Accounting for distribution between central (e.g., blood) and peripheral (e.g., tissues) compartments.
- **Non-Linear Models:** Handling cases where drug kinetics do not follow simple linear relationships, such as saturation kinetics.

7.5 Learning Objectives

By the end of this chapter, you should be able to: - Understand the basic principles of ODEs and their role in pharmacokinetics. - Apply numerical integration methods to solve pharmacokinetic equations. - Interpret the results of ODE models.

7.6 Euler method

We will mainly use ordinary differential equations of the form.

$$\frac{d\vec{x}}{dt} = \vec{f}(\vec{x}, \vec{p}, t) \tag{7.2}$$

Here $\vec{x}$ is a vector of state variables at time t. The parameters of the system are represented by the vector $\vec{p}$.

In one dimension, the system is written as

$$\frac{dx}{dt} = f(x) \quad \text{with} \quad x(t=0) := x_0 \tag{7.3}$$

The simplest way to solve the equation numerically is the Euler integration

$$f(x) = \frac{dx}{dt} \approx \frac{x(t+\Delta t) - x(t)}{\Delta t} \tag{7.4}$$

We obtain

$$x(t+\Delta t) \;=\; x(t) \;+\; \Delta t\; f(x(t)) \;+\; O(\Delta t^2) \tag{7.5}$$

Starting from an initial value x_0 at time $t = 0$ the solution can now be determined for later time points.

It is of importance to consider the error of the method. The Euler method introduces an error of $O(\Delta t^2)$ per integration step. To obtain the solution $x(t)$ at a time $t = T$, $N = T/\Delta t$ integration steps have to be performed. The total error is therefore of the order $O(T\Delta t)$ and decreases with decreasing Δt. Euler integration is a first-order method. The method is rarely used in real life (too inefficient).

The Euler Method

$$\frac{dC}{dt} = f(C) = -\frac{CL}{V}C \quad \text{with} \quad C(0) = \frac{DOSE}{V} \tag{7.6}$$

The analytical solution at time t is

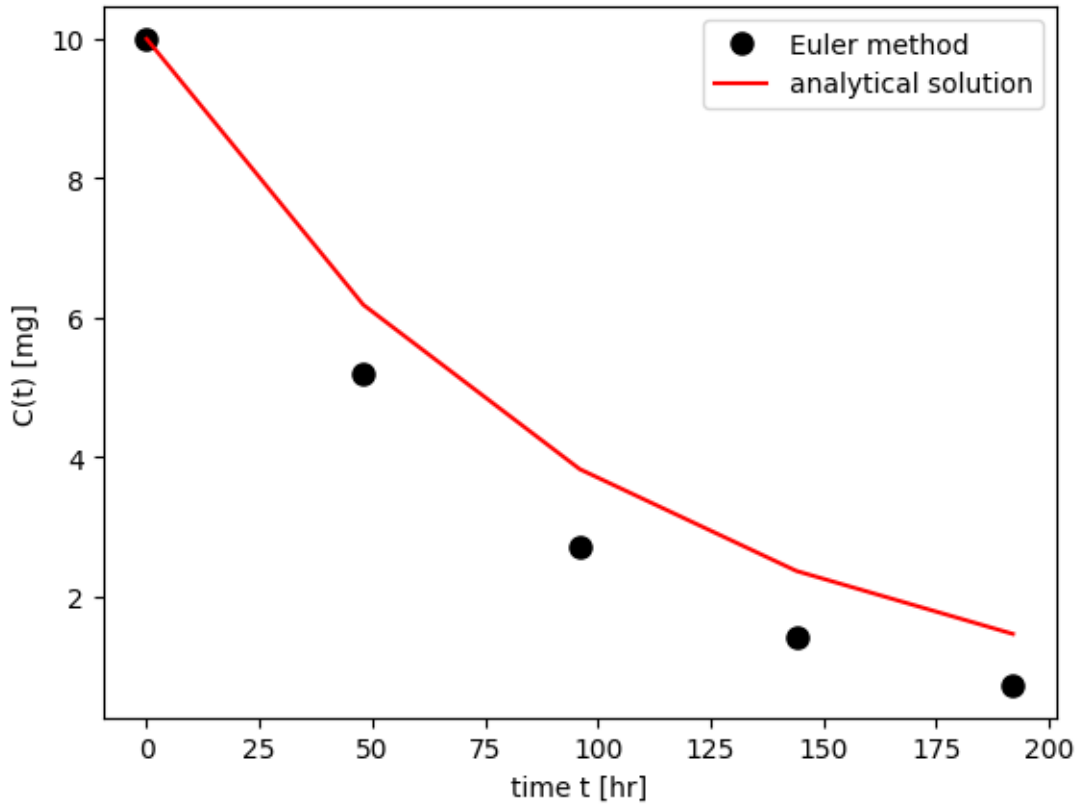
$$C(t) = \frac{DOSE}{V} \exp(-\frac{CL}{V}t) \tag{7.7}$$

An initial value is needed for this process. Computational errors are minimized by keeping the time increments very small. There has been extensive development of algorithms to solve differential equations numerically, and in most contexts the dif- ference between an analytical solution and the approximate numerical solution is inconsequential. However, solving a system of equations is computationally intensive and, even with automated, rapid processors, there is a time penalty for using differential equations to describe a model.

We now write a simple function that compares the numerical integration of the simple system with the (known) analytical solution.

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

def simple_euler(C0, tend, N, CL, V):
    """ The function integrates the simple
    system dx/dt = -k x to a time tend using the
    Euler method (N Steps) and initial condition x0.

    usage: C = simple_euler(C0, t, N)
    """
    k = 1 # set parameter k

    # some parameters
    dt = float(tend)/N
    timespan = np.arange(0, tend, dt)
    C = [float(C0)]

    # integration
    for i in range(1, N):
        # C(t) + dt * f
        C.append(C[i-1] + dt*(-CL/V*C[i-1]))

    # plot both solutions
    f, ax = plt.subplots(nrows=1, ncols=1)
    ax.plot(timespan, C,'ko', markersize=8, label='Euler method')
    ax.plot(timespan, C0*np.exp(-CL/V*timespan), 'r-', label='analytical solution')
    ax.set_xlabel('time t [hr]')
    ax.set_ylabel('C(t) [mg]')
    ax.legend(loc='upper right')
    plt.show()

    # return value of function
    cend = C[N-1]

    return cend, dt
```

```
V = 10  # [l]
CL = 0.1  # [l/hr]
DOSE = 100  # [mg]
C0 = DOSE/V

simple_euler(C0=C0, tend=24*10, N=5, CL=CL, V=V)
```



Exercise: We do know the true value $x(t = 1)$ using an analytical solution. Compare the numerical estimates for different values of `N` and plot the error as a function of `1/N`. What does `N` stand for? How does the graph look like? Why?

8 Numerical Integration in Python

The module `scipy.integrate` offers a variety of built-in functions for numerical integration. In this section, we will primarily use the function `odeint` to solve ordinary differential equations.

8.1 Example: Simple ODE

We consider a simple ODE of the form:

$$\frac{dC}{dt} = -\frac{CL}{V} \cdot C \tag{8.1}$$

with the initial condition:

$$C(0) = \frac{Dose}{V} \tag{8.2}$$

where CL (clearance) and V (volume of distribution) are parameters.

To solve this system numerically, we need to define the function representing the ODE. In this case, the function f(C, t) is:

$$f(C, t) = -\frac{CL}{V} \cdot C \tag{8.3}$$

This is implemented in a user-defined function in Python.

8.2 Understanding the Differential Equation

A differential equation describes the rate of change of a variable. In this example, $\frac{dC}{dt}$ represents the rate of change of drug concentration C with respect to time t. This rate of change is sometimes abbreviated as C'.

To solve a differential equation numerically, we must specify the initial value of the dependent variable. Here, the initial concentration C at time zero (C0) is given by:

$$C(0) = \frac{Dose}{V} \tag{8.4}$$

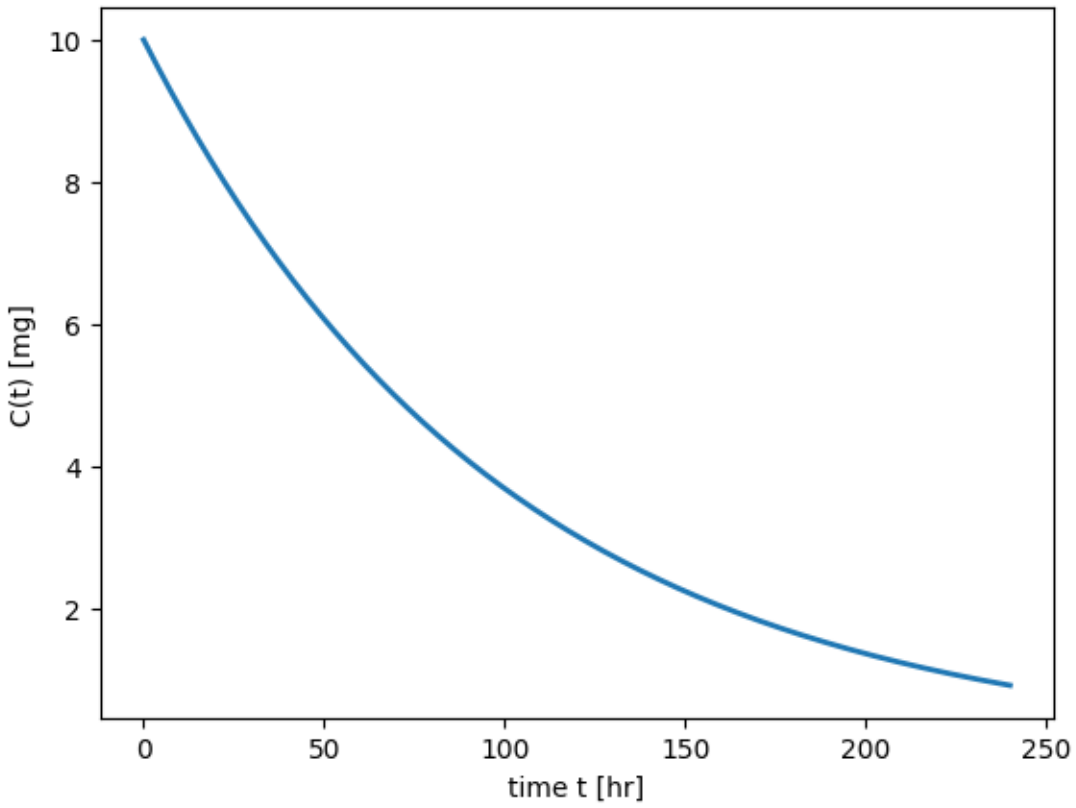
Numerical methods, such as those provided by `scipy.integrate.odeint`, are used to solve systems of differential equations where analytical solutions may not be feasible.

```
from scipy.integrate import odeint

def simple_ode(C, t, CL, V):
    """
    The function implements the simple linear
    ODE dCdt = -CL/V * C
    """

    dCdt = - CL/V *C

    return dCdt
```

To integrate the system numerically, we use `odeint`:

```
from scipy.integrate import odeint
import numpy as np

# parameters
V = 10  # [l]
CL = 0.1  # [l/hr]
DOSE = 100  # [mg]


# initial condition and time span
t = np.arange(0, 240, 0.1)  # [hr]
C0 = DOSE/V

C = odeint(simple_ode, C0, t, args=(CL, V))
```

```
f, ax = plt.subplots(nrows=1, ncols=1)
ax.set_xlabel('time t [hr]')
ax.set_ylabel('C(t) [mg]')
ax.plot(t, C, linewidth=2)
plt.show()
```



8.3 Insights and What We Learned

- **Defining the ODE:** We learned how to define a differential equation representing the pharmacokinetics of a drug. This involves identifying the parameters and the rate of change of the drug concentration.
- **Initial Conditions:** We understood the importance of initial conditions in solving differential equations. The initial concentration is determined by the dose and volume of distribution.

- **Numerical Integration:** We explored how to use numerical methods, specifically the `odeint` function from the `scipy.integrate` module, to solve ODEs when analytical solutions are not practical.
- **Interpreting Results:** By plotting the concentration-time profile, we can visually analyze how the drug concentration changes over time, providing insights into the drug's pharmacokinetics.

These skills are fundamental in pharmacokinetic modeling, allowing us to simulate and predict drug behavior under various conditions. This knowledge is crucial for optimizing drug dosing regimens, improving therapeutic efficacy, and minimizing adverse effects.

9 Compartment Model

Pharmacokinetic models are essential tools for understanding how drugs move through the body. One common approach is the compartment model, which divides the body into compartments where drugs are absorbed, distributed, metabolized, and eliminated.

9.1 Background

A compartment model simplifies the complex processes of drug distribution and elimination by grouping tissues and organs with similar drug kinetics into compartments. The most basic compartment models are:

- **One-Compartment Model:** Assumes the body is a single homogeneous unit where the drug distributes instantaneously.
- **Two-Compartment Model:** Divides the body into a central compartment (e.g., blood and highly perfused organs) and a peripheral compartment (e.g., muscle and fat tissues).
- **Multi-Compartment Models:** Include more compartments to capture more complex drug kinetics.

In this section, we will implement a simple compartment model consisting of absorption, elimination, and metabolization processes.

9.2 Relevance

Compartment models are highly relevant in pharmacokinetics for several reasons: - **Drug Development:** They help in predicting how new drugs behave in the body, guiding dosage form design and therapeutic regimens. - **Therapeutic Drug Monitoring:** Models aid in adjusting doses for individual patients to achieve optimal therapeutic levels. - **Understanding Drug Interactions:** They provide insights into how drugs interact with each other and with biological systems, influencing metabolism and elimination.

9.3 Simple Compartment Model

9.3.1 Components of the Model

1. **Absorption:** The process by which a drug enters the bloodstream from the site of administration (e.g., oral, intravenous).
2. **Elimination:** The removal of the drug from the body, primarily through metabolic conversion and excretion.
3. **Metabolization:** The chemical transformation of the drug into metabolites, which may be active or inactive.

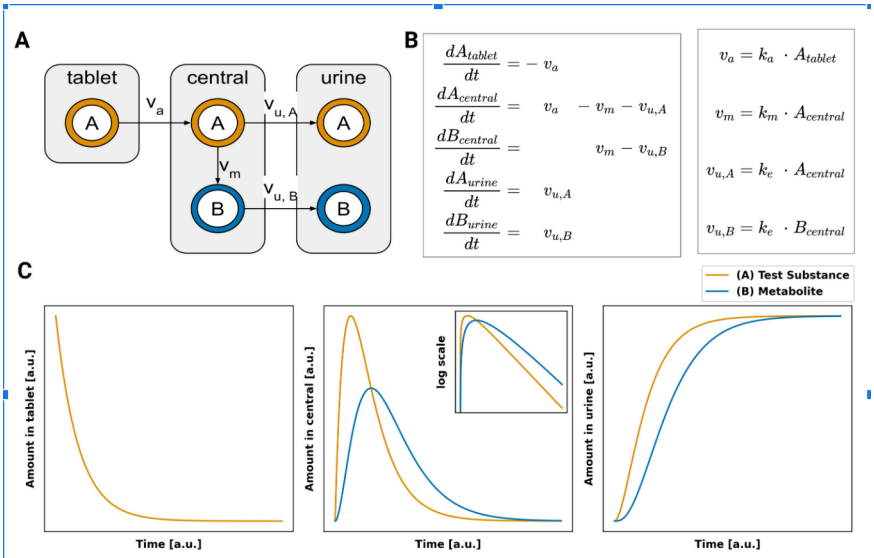


FIGURE 2.3: **Simple ODE-based pharmacokinetics model. A)** The system consists of three compartments (tablet, central, urine) that are connected via transport reactions. The model contains two substances the test substance A (orange); and the metabolite B (blue). The test substance A is metabolized to metabolite B in the central compartment. **B)** The resulting system of ordinary differential equations (ODEs). The rate of absorption, metabolism, and excretion (v_a, v_m, $v_{u,A}$, $v_{u,B}$) are modeled via irreversible mass-action kinetics. **C)** With an initial amount of $A_{tablet} = 10$ and rates $k_a = 1$, $k_m = 1$, and $k_e = 1$, all in [a.u.], the resulting amounts over time of the substances in the tablet, central, urine compartments are depicted.

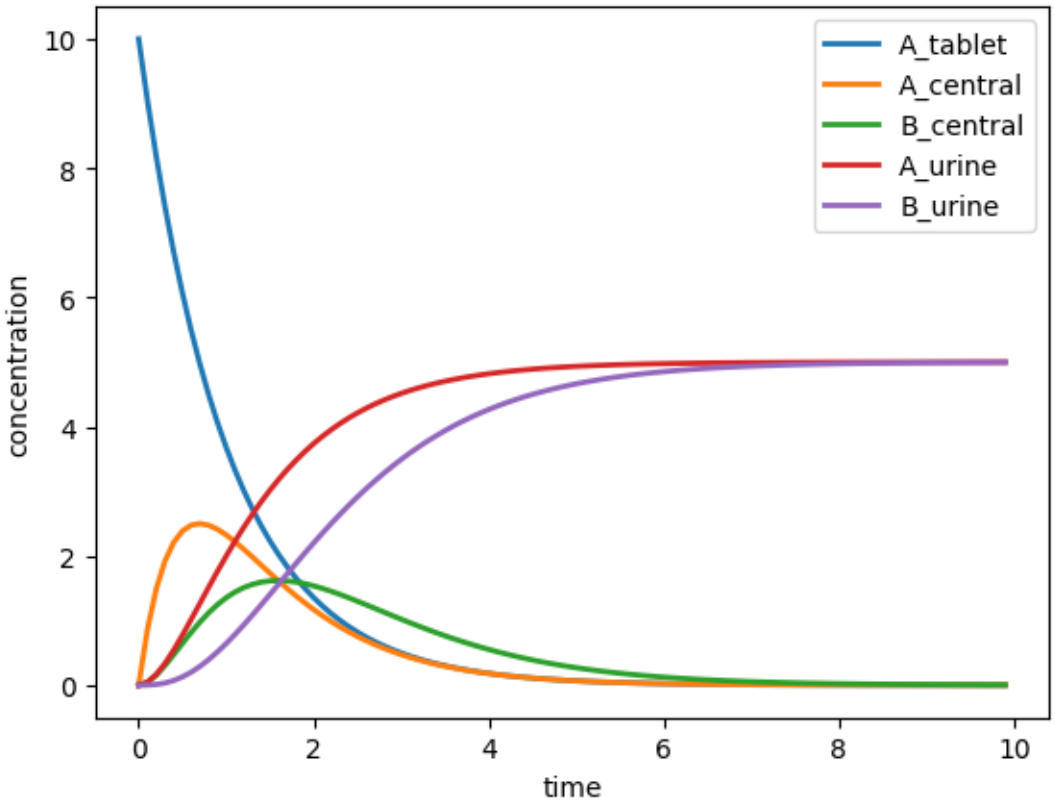
Figure 9.1: Compartment Model

10 Implementation of the compartment model

Now we implement the ordinary differential equation system corresponding to the processes and compartments depicted above. We can simulate the model with `odeint`.

```
import numpy as np
from scipy.integrate import odeint
from matplotlib import pylab as plt

def dydt_compartment_model(x, t, ka, km, ke):
    """
    System of ODEs of the compartment model.
    """
    # state variables
    A_tablet = x[0]
    A_central = x[1]
    B_central = x[2]
    A_urine = x[3]
    B_urine = x[4]

    # rates
    va = ka * A_tablet
    vm = km * A_central
    vuA = ke * A_central
    vuB = ke * B_central

    # odes (stoichiometric equation)
    return [
        -va,             # dA_tablet/dt
         va - vm - vuA,  # dA_central/dt
         vm - vuB,       # dB_central/dt
         vuA,            # dA_urine/dt
         vuB,            # dB_urine/dt
    ]


# initial condition and time span
```

```
t = np.arange(0, 10, 0.1)  # [hr]
Dose_A = 10.0  # [mg]
x0 = [
    Dose_A,  # A_tablet
    0.0,   # A_central
    0.0,   # B_central
    0.0,   # A_urine
    0.0,   # B_urine
]

# parameters
ka = 1.0
km = 1.0
ke = 1.0

x = odeint(dydt_compartment_model, x0, t, args=(ka, km, ke))
names = ["A_tablet", "A_central", "B_central", "A_urine", "B_urine"]

f, ax = plt.subplots(nrows=1, ncols=1)
for k, name in enumerate(names):
    ax.plot(t, x[:, k], linewidth=2, label=name)

ax.legend()
ax.set_xlabel("time")
ax.set_ylabel("concentration")

plt.show()
```



10.1 Insights and What We Learned

- **Compartmental Representation:** By dividing the body into compartments, we can model complex drug kinetics in a manageable way.
- **Absorption, Metabolization, and Elimination:** Understanding these processes helps in predicting drug behavior and designing effective dosing regimens.
- **Numerical Solutions:** Using numerical methods like `odeint` allows us to solve complex systems of ODEs that do not have straightforward analytical solutions.

This exercise provides a practical foundation in modeling drug kinetics using compartment models, which are crucial for various applications in pharmacology and medicine.

11 The Effect of Metabolism on Urinary Recoveries

We are interested in understanding how the metabolic rate affects the amounts of drug A and its metabolite B recovered in the urine. Specifically, we will systematically alter the parameter for the metabolization rate (k_m), which determines the conversion from A to B ((A →B)).

11.1 Background

In pharmacokinetics, the metabolism of a drug can significantly influence its pharmacological effects and the duration of its action. Metabolites can be either active or inactive, and their formation and elimination are crucial in determining the overall drug effect and safety profile. The rate of metabolism, represented by the rate constant (k_m), plays a key role in these processes.

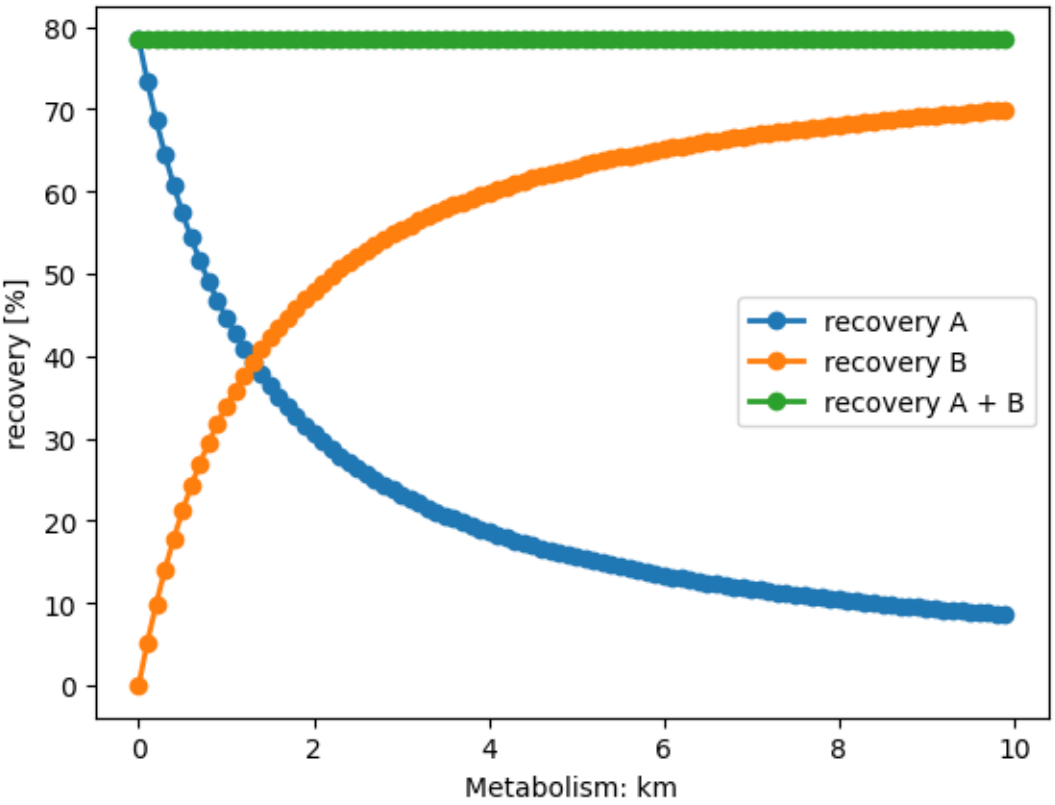
11.2 Relevance

Understanding the effect of metabolism on urinary recoveries is important for several reasons: - **Drug Efficacy and Safety:** The rate at which a drug is metabolized can affect its therapeutic efficacy and the risk of side effects. - **Dosing Regimens:** Adjustments to dosing regimens may be necessary based on metabolic rates to achieve optimal drug levels. - **Interindividual Variability:** Differences in metabolic rates among individuals can lead to variations in drug response and require personalized medicine approaches.

11.3 Systematic Parameter Scan

We will systematically alter the metabolization rate k_m to observe its effect on the amounts of A and B recovered in the urine. The range of k_m values will be chosen to cover a broad spectrum of metabolic rates, from slow to fast metabolism.

```
t = np.arange(0, 3, 0.1)  # [hr]
kms = np.arange(0, 10, 0.1)
A_recovery = np.zeros_like(kms)
B_recovery = np.zeros_like(kms)

for k, km_new in enumerate(kms):
    x = odeint(dydt_compartment_model, x0, t, args=(ka, km_new, ke))
    A_recovery[k] = x[-1, 3]  # A_urine(end)
    B_recovery[k] = x[-1, 4]  # B_urine(end)

A_recovery = A_recovery/Dose_A
B_recovery = B_recovery/Dose_A

f, ax = plt.subplots(nrows=1, ncols=1)
ax.plot(kms, A_recovery * 100, linewidth=2, label="recovery A", marker="o")
ax.plot(kms, B_recovery * 100, linewidth=2, label="recovery B", marker="o")
ax.plot(kms, (A_recovery + B_recovery) * 100, linewidth=2, label="recovery A + B", marker="o
ax.legend()
ax.set_xlabel("Metabolism: km")
ax.set_ylabel("recovery [%]")

plt.show()
```



11.4 Insights and What We Learned

- **Metabolism Rate Impact:** The rate of metabolism ((k_m)) significantly influences the amounts of the parent drug and its metabolite recovered in the urine.
- **Urinary Recovery Patterns:** By varying (k_m), we can observe how faster metabolism leads to higher levels of metabolite B and lower levels of drug A in the urine, and vice versa.
- **Clinical Implications:** Understanding these patterns is crucial for dose adjustment and optimizing therapeutic outcomes, especially for drugs with active metabolites.

Through this exercise, we gain valuable insights into the dynamic interplay between drug metabolism and elimination, enhancing our ability to predict drug behavior and tailor treatments to individual patient needs.

Exercise: Does the absorption rate of A have an effect on the urinary recovery of B?

12 Absorption

Absorption is a key principle of pharmacokinetics, which is the study of how the body interacts with a drug over time, including the processes of absorption, distribution, metabolism, and excretion (ADME).

In the context of drug therapy, absorption refers to the process through which a drug moves from the site of administration into the bloodstream. The rate and extent of absorption can significantly impact how quickly and effectively a drug acts.

Various factors can influence drug absorption, including:

1. Route of Administration: The route of administration significantly impacts how a drug is absorbed. For instance, drugs administered intravenously bypass the absorption process as they are introduced directly into the bloodstream. However, orally administered drugs must pass through the stomach and intestines, where they are absorbed into the bloodstream. This can be influenced by factors such as pH levels, presence of food, and gastrointestinal motility.

2. Drug Formulation: The physical and chemical properties of the drug can influence how well it is absorbed. For example, drugs formulated in a liquid solution are often absorbed more rapidly than those in a tablet or capsule.

3. Physiological Factors: Individual characteristics like age, sex, genetic factors, and health status can also influence drug absorption. For example, certain conditions like malabsorption syndromes or diseases affecting the liver or kidney can alter the absorption of drugs.

4. Drug Interactions: Certain drugs can interact in the body and affect absorption. For instance, some medications can increase stomach acidity, which can affect the absorption of other drugs.

Understanding absorption is crucial in drug therapy as it affects the onset, intensity, and duration of a drug's effect. A drug must be absorbed into the bloodstream before it can reach its site of action, and variations in the rate and extent of absorption can lead to differences in how individuals respond to the same dose of a drug. Consequently, understanding absorption can help in designing drug dosing regimens and predicting a drug's effect.

13 Simple absorption models

In the following we study a simple model for absorption and elimination. A can be absorbed from the tablet in the systemic circulation ($A_{tablet} \rightarrow A_{system}$) which can be eliminated in the urine via renal excretion ($A_{system} \rightarrow A_{urine}$).

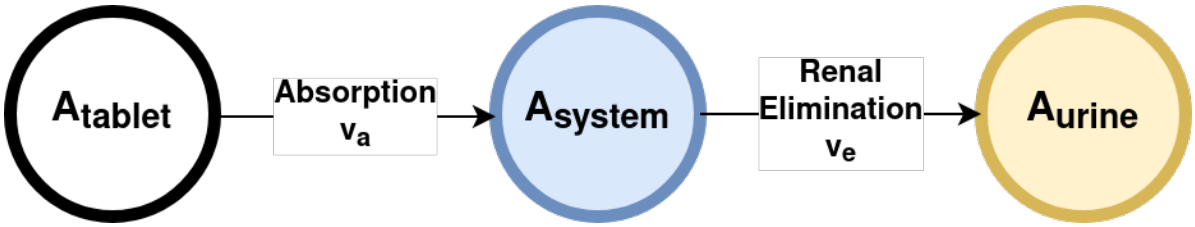


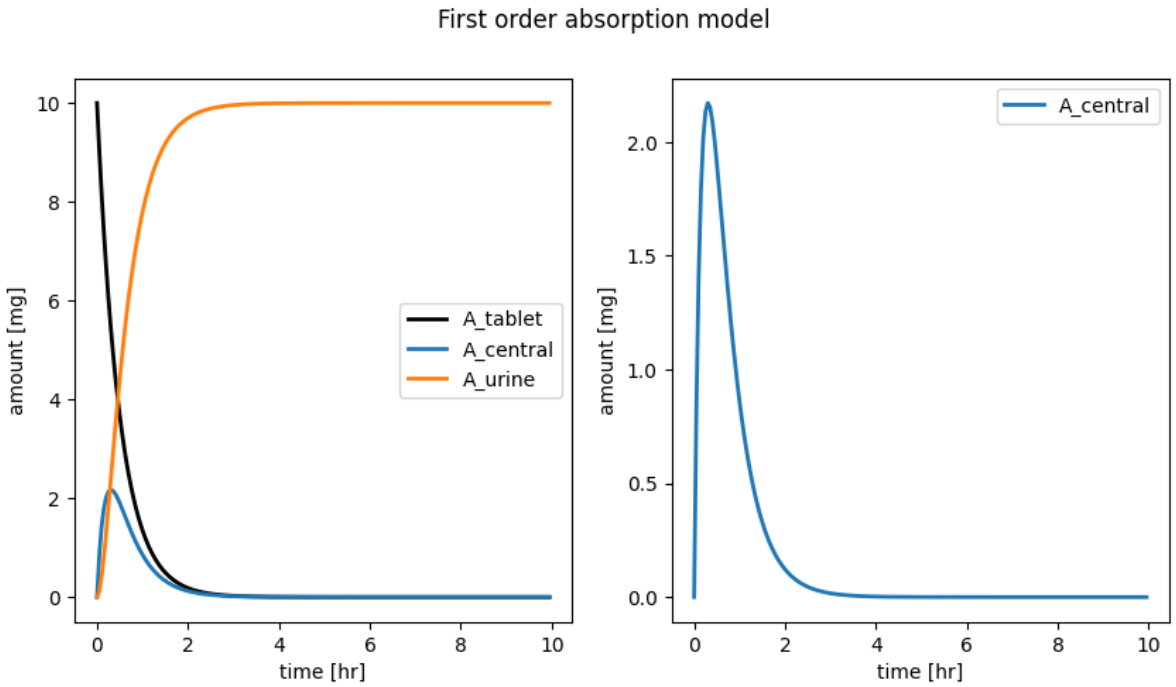
Figure 13.1: Absorption Elimination Model

Elimination and absorption are assumed to be Mass-Action, i.e., depending on a rate constant k and the amount or concentration of the respective substance.

The ordinary differential equation system (ODE) results in:

```
import numpy as np
from scipy.integrate import odeint
from matplotlib import pylab as plt

def dxdt_absorption_first_order(x, t, ka, ke):
    """
    First order absorption model
    """
    # state variables
    A_tablet = x[0]  # [mg]
    A_central = x[1] # [mg/l]
    A_urine = x[2] # [mg]

    # rates
    va = ka * A_tablet  # [mg/hr]
    ve = ke * A_central # [mg/hr]
```

```
    # odes (stoichiometric equation)
    return [
        -va,            # dA_tablet/dt  [mg/hr]
         va - ve,       # dA_central/dt [mg/hr]
         ve,            # dA_urine/dt   [mg/hr]
    ]


# initial condition and time span
t = np.arange(0, 10, 0.05) # [hr]
Dose_A = 10.0  # [mg]
x0 = [
    Dose_A,  # A_tablet  [mg]
    0,   # A_central [mg]
    0,   # A_urine [mg]
]

# parameters
ka = 2.0  # [1/hr]
ke = 5.0  # [1/hr]

x = odeint(dxdt_absorption_first_order, x0, t, args=(ka, ke))
names = ["A_tablet", "A_central", "A_urine"]
colors = ["black", "tab:blue", "tab:orange"]

# plot results
f, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(10,5))
f.suptitle("First order absorption model")
# all species
for k, name in enumerate(names):
    ax1.plot(t, x[:, k], linewidth=2, label=name, color=colors[k])

# only A_central
ax2.plot(t, x[:, 1], linewidth=2, label=names[1], color=colors[1])

for ax in (ax1, ax2):

    ax.legend()
    ax.set_xlabel("time [hr]")
    ax.set_ylabel("amount [mg]")

plt.show()
```

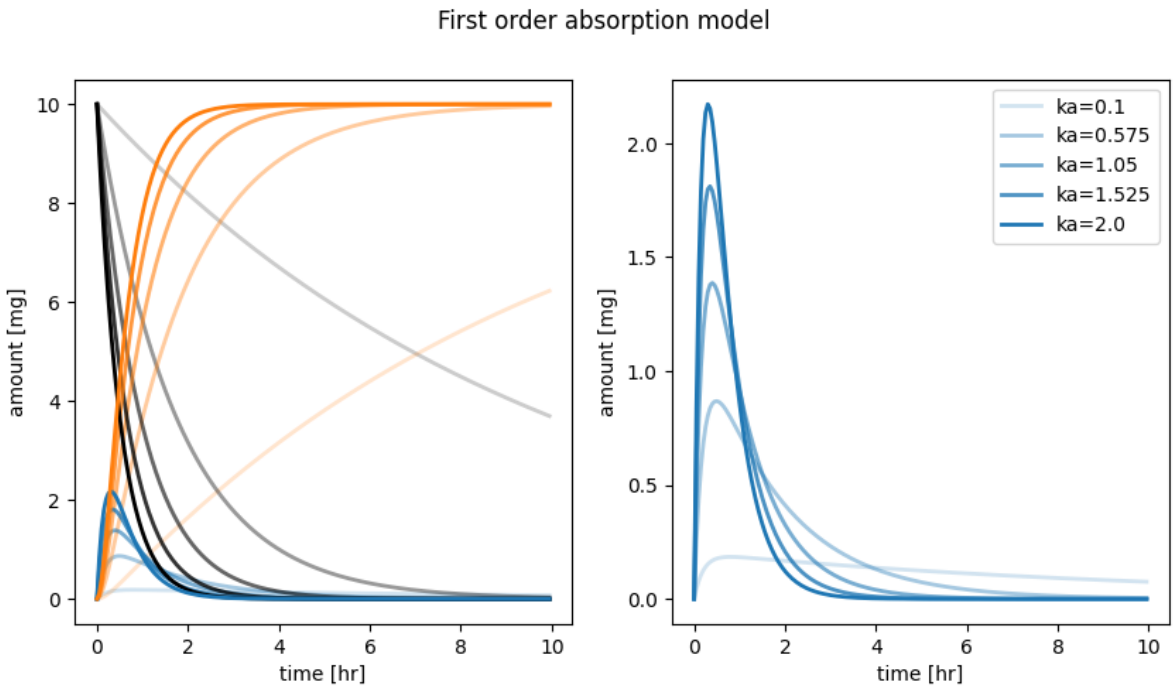


When `va = ve` the peak concentration in the central compartment is reached.

13.1 Effect of absorption parameter

```
# parameters
ka = 2.0  # [1/hr]
ke = 5.0  # [1/hr]

n_samples = 5
kas = np.linspace(0.1, 2.0, num=n_samples)  # [1/hr]

# plot results
f, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(10,5))
f.suptitle("First order absorption model")

for kp, ka in enumerate(kas):
    x = odeint(dxdt_absorption_first_order, x0, t, args=(ka, ke))

    # all species
    for k, name in enumerate(names):
```

```
        ax1.plot(t, x[:, k], linewidth=2, color=colors[k], alpha=(kp+1)/n_samples, label=f"{k

    # only A_central
    ax2.plot(t, x[:, 1], linewidth=2, color=colors[1], alpha=(kp+1)/n_samples, label=f"{ka=}

    for ax in (ax1, ax2):
        ax.set_xlabel("time [hr]")
        ax.set_ylabel("amount [mg]")
    ax2.legend()

plt.show()
```



14 Lag absorption models

Absorption can show a time lag due stomach passage or absorption in later regions of the intestine (e.g. ileum, jejunum). Such lags can be introduced either directly in absorption equations or via so called transit chains (i.e. chains of absorption reactions).

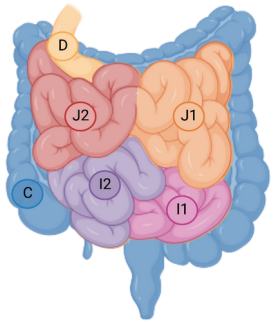
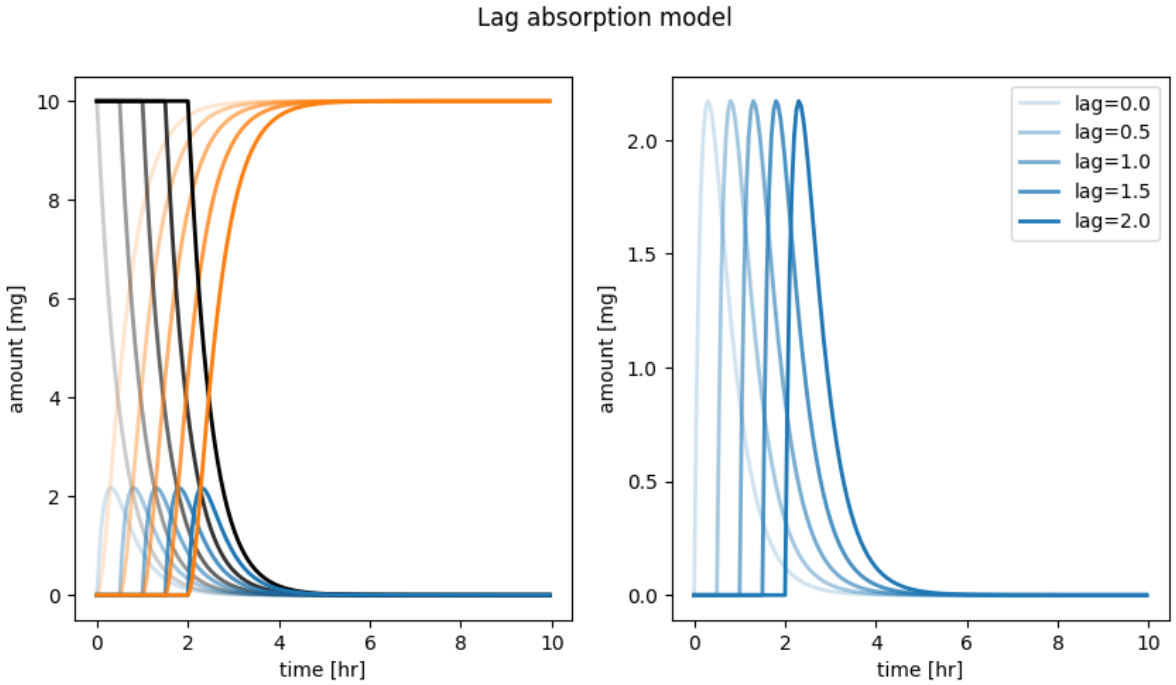


Figure 14.1: Intestine sections

First we have a look at models with an explicit delay in form of a lag time.

```
import numpy as np
from scipy.integrate import odeint
from matplotlib import pylab as plt

def dxdt_absorption_lag(x, t, ka, ke, lag=0.0):
    """
    First order absorption model with lag time
    """
    # state variables
    A_tablet = x[0]  # [mg]
    A_central = x[1] # [mg/l]
    A_urine = x[2] # [mg]

    # rates
    if t >= lag:
        va = ka * A_tablet  # [mg/hr]
    else:
```

```
        va = 0
    ve = ke * A_central # [mg/hr]

    # odes (stoichiometric equation)
    return [
        -va,            # dA_tablet/dt  [mg/hr]
         va - ve,       # dA_central/dt [mg/hr]
         ve,            # dA_urine/dt   [mg/hr]
    ]

ka = 2.0  # [1/hr]
ke = 5.0  # [1/hr]

n_samples = 5
lags = np.linspace(0.0, 2.0, num=n_samples)  # [hr]

# plot results
f, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(10,5))
f.suptitle("Lag absorption model")

for kp, lag in enumerate(lags):
    x = odeint(dxdt_absorption_lag, x0, t, args=(ka, ke, lag))

    # all species
    for k, name in enumerate(names):
        ax1.plot(t, x[:, k], linewidth=2, color=colors[k], alpha=(kp+1)/n_samples, label=f"{

    # only A_central
    ax2.plot(t, x[:, 1], linewidth=2, color=colors[1], alpha=(kp+1)/n_samples, label=f"{lag=

    for ax in (ax1, ax2):

        ax.set_xlabel("time [hr]")
        ax.set_ylabel("amount [mg]")
    ax2.legend()

plt.show()
```


Lag absorption model
amount [mg]
time [hr]
lag=0.0
lag=0.5
lag=1.0
lag=1.5
lag=2.0

15 Transit chain absorption models

Now we have a look at the example of a transit chain. By coupling a set of reactions delays can be introduced and curves are smoothend.

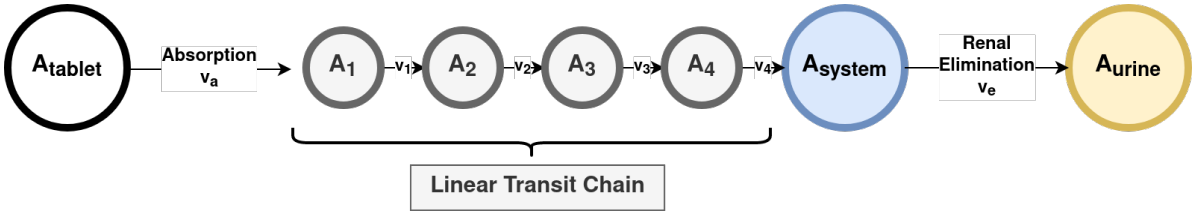


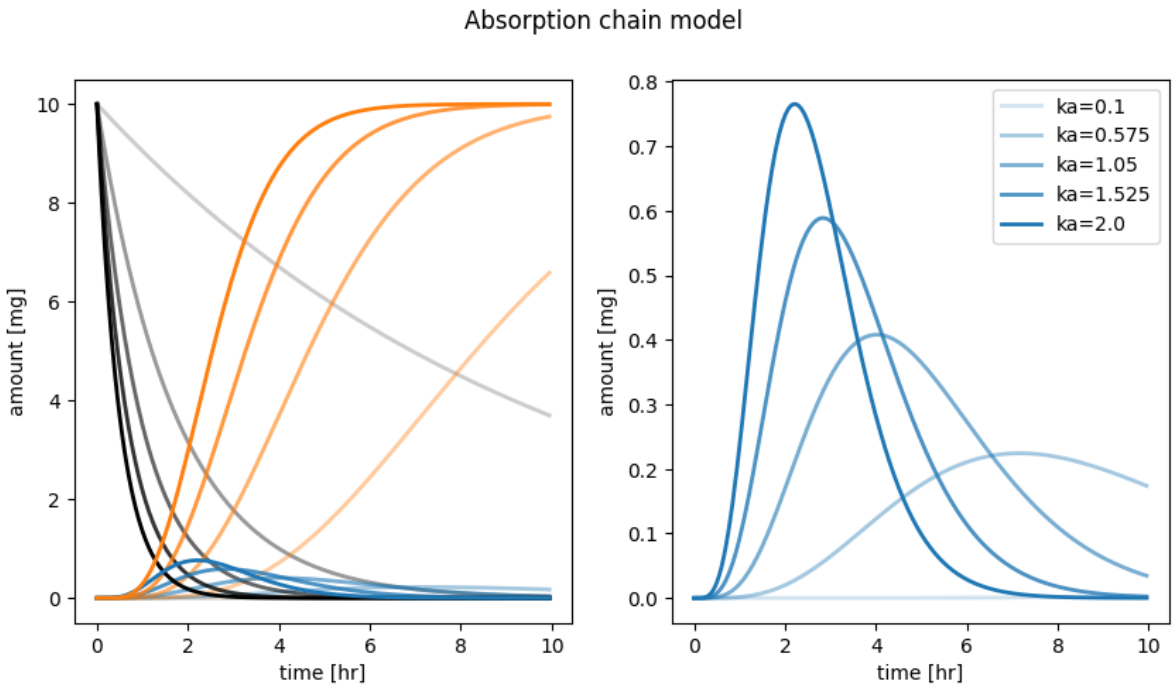
Figure 15.1: Transit chain

Due to the transit chain the substance appears delayed in the systemic circulation (central compartment).

```
def dxdt_absorption_chain(x, t, ka, ke):
    """
    First order absorption model for transit chain.
    """
    # state variables
    A_tablet = x[0]  # [mg]
    A_central = x[1] # [mg/l]
    A_urine = x[2] # [mg]
    A1 = x[3]
    A2 = x[4]
    A3 = x[5]
    A4 = x[6]

    # rates
    va = ka * A_tablet  # [mg/hr]
    ve = ke * A_central # [mg/hr]
    v1 = ka * x[3]
    v2 = ka * x[4]
    v3 = ka * x[5]
    v4 = ka * x[6]
```

```
    # odes (stoichiometric equation)
    dxdt = np.zeros(7)

    dxdt[0] = -va            # dA_tablet/dt  [mg/hr]
    dxdt[1] =  v4 - ve     # dA_central/dt [mg/hr] (end of chain)
    dxdt[2] =  ve            # dA_urine/dt  [mg/hr]
    dxdt[3] = va - v1
    dxdt[4] = v1 - v2
    dxdt[5] = v2 - v3
    dxdt[6] = v3 - v4

    return dxdt

# initial condition and time span
t = np.arange(0, 10, 0.05) # [hr]
Dose_A = 10.0  # [mg]
x0 = [
    Dose_A,  # A_tablet  [mg]
    0.0,   # A_central [mg]
    0.0,   # A_urine [mg]
    0.0,   # A1 [mg]
    0.0,   # A2 [mg]
    0.0,   # A3 [mg]
    0.0,   # A4 [mg]
]

names = ["A_tablet", "A_central", "A_urine"]
colors = ["black", "tab:blue", "tab:orange"]

# parameters
ka = 2.0  # [1/hr]
ke = 5.0  # [1/hr]

n_samples = 5
kas = np.linspace(0.1, 2.0, num=n_samples)  # [1/hr]

# plot results
f, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(10,5))
f.suptitle("Absorption chain model")

for kp, ka in enumerate(kas):
    x = odeint(dxdt_absorption_chain, x0, t, args=(ka, ke))
```

```
    # all species
    for k, name in enumerate(names):
        ax1.plot(t, x[:, k], linewidth=2, color=colors[k], alpha=(kp+1)/n_samples, label=f"{

    # only A_central
    ax2.plot(t, x[:, 1], linewidth=2, color=colors[1], alpha=(kp+1)/n_samples, label=f"{ka=}

    for ax in (ax1, ax2):
        ax.set_xlabel("time [hr]")
        ax.set_ylabel("amount [mg]")
    ax2.legend()

plt.show()
```



16 Multiple dosing

Multiple dosing in drug therapy refers to the practice of administering multiple doses of a drug or medication over a specific period of time to maintain a constant or effective drug concentration within the body. This approach is often used for conditions that require a continuous level of drug presence in the system, or for drugs with a short half-life that need to be administered frequently to maintain their effect.

Here are the main aspects of multiple dosing:

1. **Steady-state Concentration**: Over time, multiple doses of a drug lead to a steady-state concentration in the blood. This steady-state is reached when the amount of drug entering the body equals the amount being removed (through metabolism and excretion). This usually happens after approximately 4-5 times the half-life of the drug.
2. **Dosing Interval**: The dosing interval is the time between two consecutive doses. It's typically designed based on the half-life of the drug. For most drugs, it's aimed at maintaining the drug concentration within the therapeutic window.
3. **Therapeutic Window**: This is the range of drug concentrations in the blood that provide effective treatment without causing toxicity. The goal of multiple dosing is to keep the drug concentration within this range.
4. **Dose and Frequency**: The size of the dose and frequency of dosing are critical in multiple dosing strategies. They must be determined with care to avoid toxic levels while still providing therapeutic benefit.
5. **Accumulation and Elimination**: When drugs are administered repeatedly, they can accumulate in the body, leading to higher concentrations than after the first dose. This must be considered when planning a multiple dosing regimen.
6. **Interindividual Variability**: There can be a significant difference in how different individuals respond to the same drug and dosage due to genetic factors, age, gender, organ function, and the presence of other diseases or drugs. This factor is crucial in customizing a multiple dosing regimen for each patient.
7. **Compliance**: Compliance with the dosing regimen is essential for the success of the treatment. The easier the regimen (for example, fewer doses per day), the higher the chance the patient will adhere to it.

In short, multiple dosing is a sophisticated process and is a crucial part of many drug therapies. It involves a careful balance of dose, frequency, and timing to ensure that the drug is as effective as possible while minimizing side effects and toxicity. To achieve this, healthcare providers often need to monitor drug levels in the blood and adjust doses based on patient response.

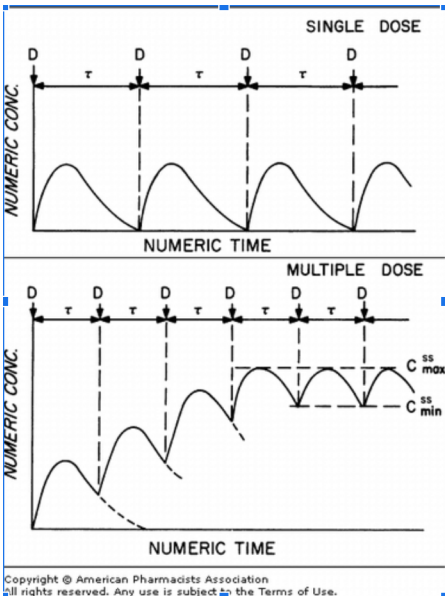


Figure 16.1: Multiple Dosing

16.1 Multiple dosing example

We now run a multiple dosing protocol with our simple model.

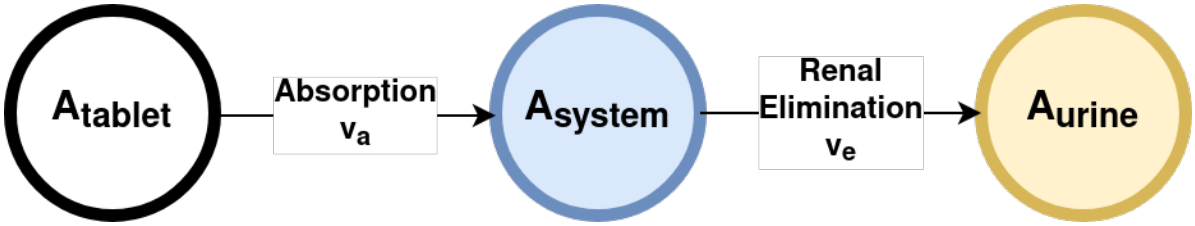


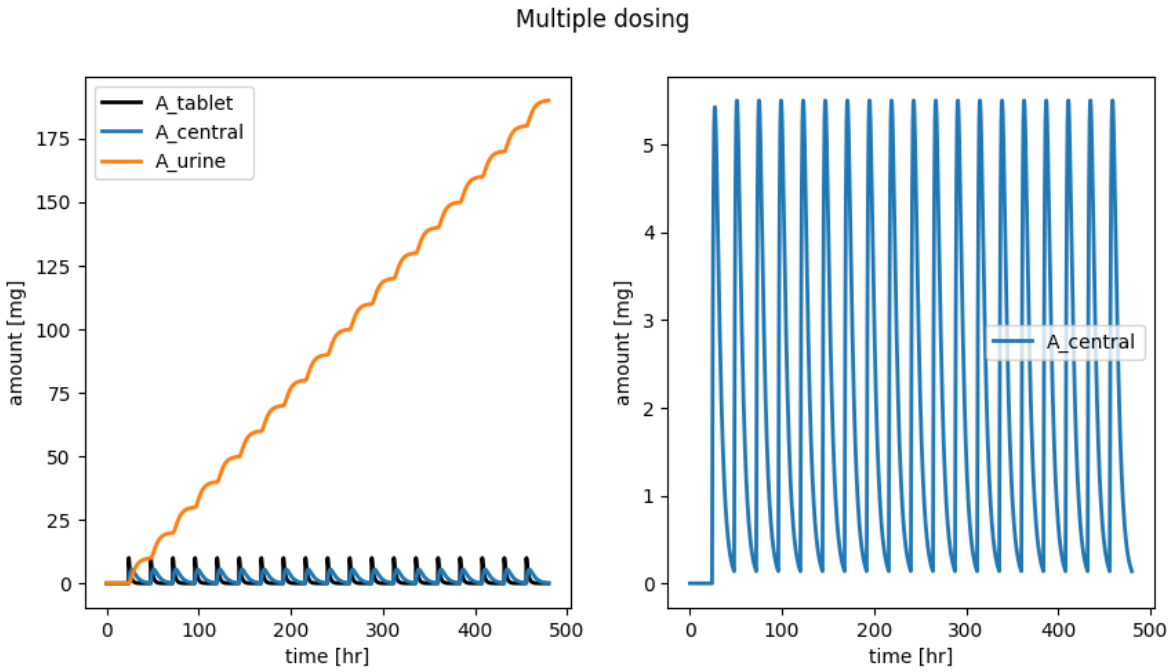
Figure 16.2: Absorption Elimination Model

```
import numpy as np
from scipy.integrate import odeint
from matplotlib import pylab as plt
import pandas as pd
```

```
def dxdt_absorption_first_order(x, t, ka, ke):
    """
    First order absorption model
    """
    # state variables
    A_tablet = x[0]  # [mg]
    A_central = x[1] # [mg/l]
    A_urine = x[2] # [mg]

    # rates
    va = ka * A_tablet  # [mg/hr]
    ve = ke * A_central # [mg/hr]

    # odes (stoichiometric equation)
    return [
        -va,            # dA_tablet/dt  [mg/hr]
         va - ve,       # dA_central/dt [mg/hr]
         ve,            # dA_urine/dt   [mg/hr]
    ]


def simulate_multi_dosing(Dose_A, ka, ke):
    """Helper function to run the multiple dosing simulation."""

    # initial condition
    names = ["A_tablet", "A_central", "A_urine"]
    x0 = [
        Dose_A,  # A_tablet  [mg]
        0.0,   # A_central [mg]
        0.0,   # A_urine [mg]
    ]

    # time span for single dose
    t = np.linspace(0, 24, num=100) # [hr]

    # multiple dose simulation
    n_doses = 20  # [hr]

    dfs = []
    for k in range(n_doses):
        if k == 0:
            x0[0] = 0
```

```
            tvec = t.copy()
        elif k > 0:
            x0 = x[-1, :]
            x0[0] = x0[0] + Dose_A
            tvec = t.copy() + tvec[-1]

        x = odeint(dxdt_absorption_first_order, x0, tvec, args=(ka, ke))
        df = pd.DataFrame(x, columns=names)
        df["time"] = tvec
        dfs.append(df)

    df_all = pd.concat(dfs)
    return df_all
```

```
# run simulation and plot results
Dose_A = 10.0  # [mg]
ka = 0.5  # [1/hr]
ke = 0.2  # [1/hr]
df = simulate_multi_dosing(Dose_A, ka, ke)

colors = ["black", "tab:blue", "tab:orange"]
names = ["A_tablet", "A_central", "A_urine"]
f, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(10,5))
f.suptitle("Multiple dosing")
# all species
for k, name in enumerate(names):
    ax1.plot(df.time, df[name], linewidth=2, label=name, color=colors[k])

# only A_central
ax2.plot(df.time, df["A_central"], linewidth=2, label=names[1], color=colors[1])

for ax in (ax1, ax2):

    ax.legend()
    ax.set_xlabel("time [hr]")
    ax.set_ylabel("amount [mg]")

plt.show()
```



Exercise: How do you have to change kel or ka to observe dose accumulation in the central compartment? Change these parameters and see what the effect is.

Does the Dose `Dose_A` play a role?

17 Therapeutic window

The **therapeutic window**, also known as the **therapeutic range** or **therapeutic index**, is a crucial concept in pharmacology and drug therapy. It is the range of drug dosages which can treat disease effectively while staying within the safety limits to avoid toxicity and side effects.

The therapeutic window is determined by two main factors: **the minimum effective concentration (MEC)** and the **maximum tolerable concentration (MTC)**.

- The **minimum effective concentration (MEC)** is the lowest concentration of a drug in the patient's bloodstream that still produces the desired therapeutic effect. If the concentration falls below this level, the drug may not be effective in treating the condition, leading to ineffective therapy.
- The **maximum tolerable concentration (MTC)** is the highest concentration of a drug that can be tolerated without causing significant toxic effects or side effects. If the drug concentration exceeds this level, the risk of side effects and toxicity increases.

The therapeutic window lies between these two concentrations. The goal in drug therapy is to maintain the drug concentration within this window: high enough to be effective (above the MEC) but low enough to be safe (below the MTC).

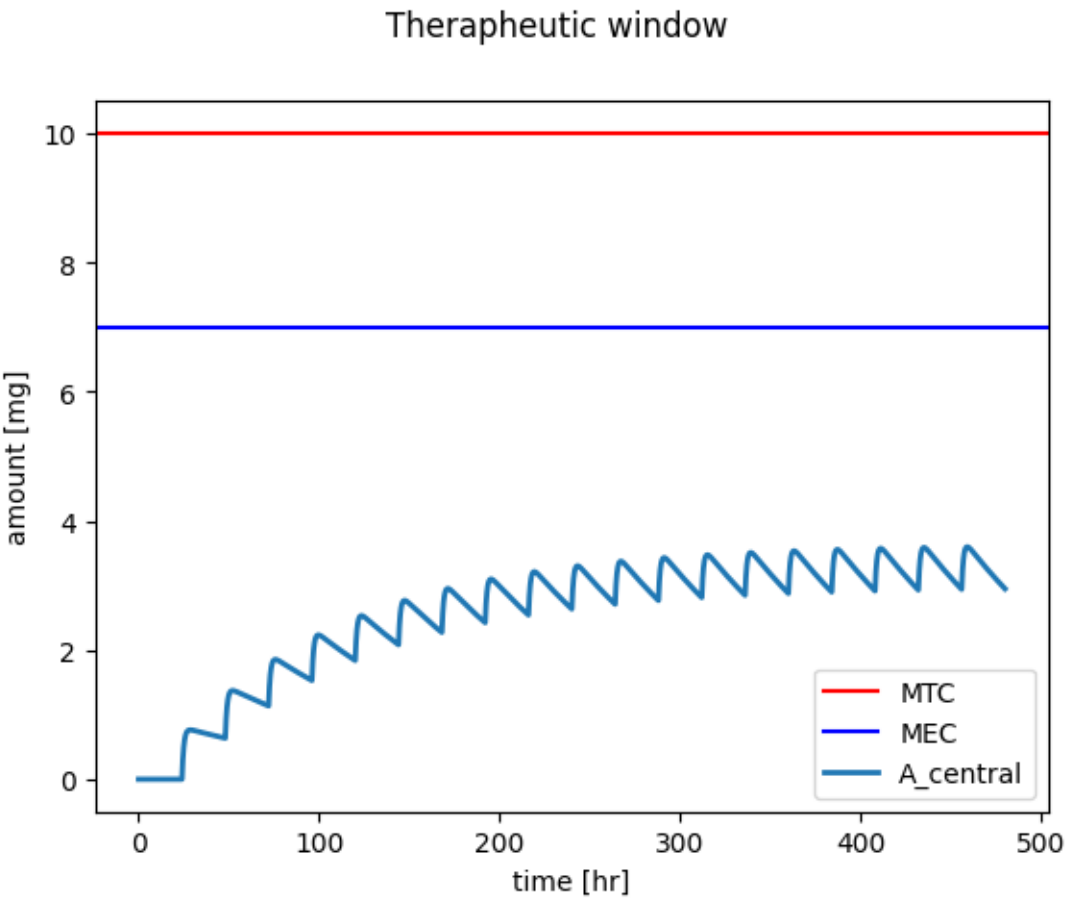
This balance is particularly important with drugs that have a narrow therapeutic window, meaning the difference between the effective dose and the toxic dose is small. For these drugs, small changes in the drug concentration can lead to either therapeutic failure or toxicity.

Monitoring drug levels in the body is often required for these drugs, and dosages may need to be adjusted based on individual patient response, metabolism, and excretion rates. This is especially relevant in multiple dosing regimens where the aim is to maintain a steady-state concentration of the drug within the therapeutic window.

In summary, the therapeutic window is critical in drug therapy. It provides a guideline for effective and safe drug dosages. Staying within this window helps to achieve the therapeutic benefits of the drug while minimizing the risk of side effects and toxicity.

We add the **MEC** and **MTC** lines to see if we can achieve the therapeutic range.

```
# plot results

# run simulation and plot results
Dose_A = 80.0  # [mg]
ka = 0.01  # [1/hr]
ke = 1.0  # [1/hr]
df = simulate_multi_dosing(Dose_A, ka, ke)

f, ax = plt.subplots(nrows=1, ncols=1)
f.suptitle("Therapeutic window")

ax.axhline(y=10, color='r', linestyle='-', label="MTC")
ax.axhline(y=7, color='b', linestyle='-', label="MEC")

ax.plot(df.time, df["A_central"], linewidth=2, label=names[1], color=colors[1])
ax.legend()
ax.set_xlabel("time [hr]")
ax.set_ylabel("amount [mg]")

plt.show()
```



Exercise: Change the dosing regime via `ka`, `ke` and `Dose_A` so that the concentrations are in the therapeutic window. See what is the effect of different combinations of these parameters.

18 Metabolization

In pharmacokinetics and enzymology, the rate at which reactions occur is crucial. Different mathematical models are used to describe these rates, with some of the most common being the Mass-Action model, the Michaelis-Menten model, and the Hill equation. Here's a brief summary of each:

1. Mass-Action Model: This model is one of the simplest and is based on the principle that the rate of a reaction is directly proportional to the concentration of the reacting substances. For a reaction A + B $\rightarrow$ C, the rate would be expressed as Rate = k[A][B], where k is the rate constant, and [A] and [B] are the concentrations of A and B.

2. Michaelis-Menten Model: This model is used to describe enzyme-catalyzed reactions, particularly when enzyme concentrations are much lower than substrate concentrations. The rate of reaction (v) is given by the equation v = Vmax[S]/(Km + [S]), where Vmax is the maximum rate, [S] is the substrate concentration, and Km is the Michaelis constant (the substrate concentration at which the reaction rate is half of Vmax).

3. Hill Equation: This model is often used when there is cooperativity or interaction between multiple binding sites on a molecule (like a protein or enzyme). The equation is given by v = Vmax[S]^n/(Kd + [S]^n), where n is the Hill coefficient representing the degree of cooperativity.

For both the Michaelis-Menten model and Hill equation, values of Vmax, Km, and Kd are usually determined experimentally.

18.1 Inhibition and activition

Inhibition and activation also play crucial roles in metabolic models:

Inhibition: This occurs when a molecule binds to an enzyme and decreases its activity. Inhibitors can be competitive (bind to the active site and compete with the substrate), non-competitive (bind to a separate site and change the enzyme's shape), or uncompetitive (bind to the enzyme-substrate complex). Each type of inhibition changes the parameters (Vmax, Km) in distinctive ways.

Activation: This is when a molecule binds to an enzyme and increases its activity. This can lead to an increase in the maximum reaction rate (Vmax) or a decrease in the Km value, indicating an increased affinity of the enzyme for its substrate.

In summary, these models and concepts help scientists understand and predict the behavior of enzymes and other biochemical reactions, which is crucial in fields like drug design and metabolic engineering.

18.2 Rate equations

18.2.1 Mass-Action Model

This model is one of the simplest and is based on the principle that the rate of a reaction is directly proportional to the concentration of the reacting substances. For a reaction A + B $\rightarrow$ C, the rate would be expressed as

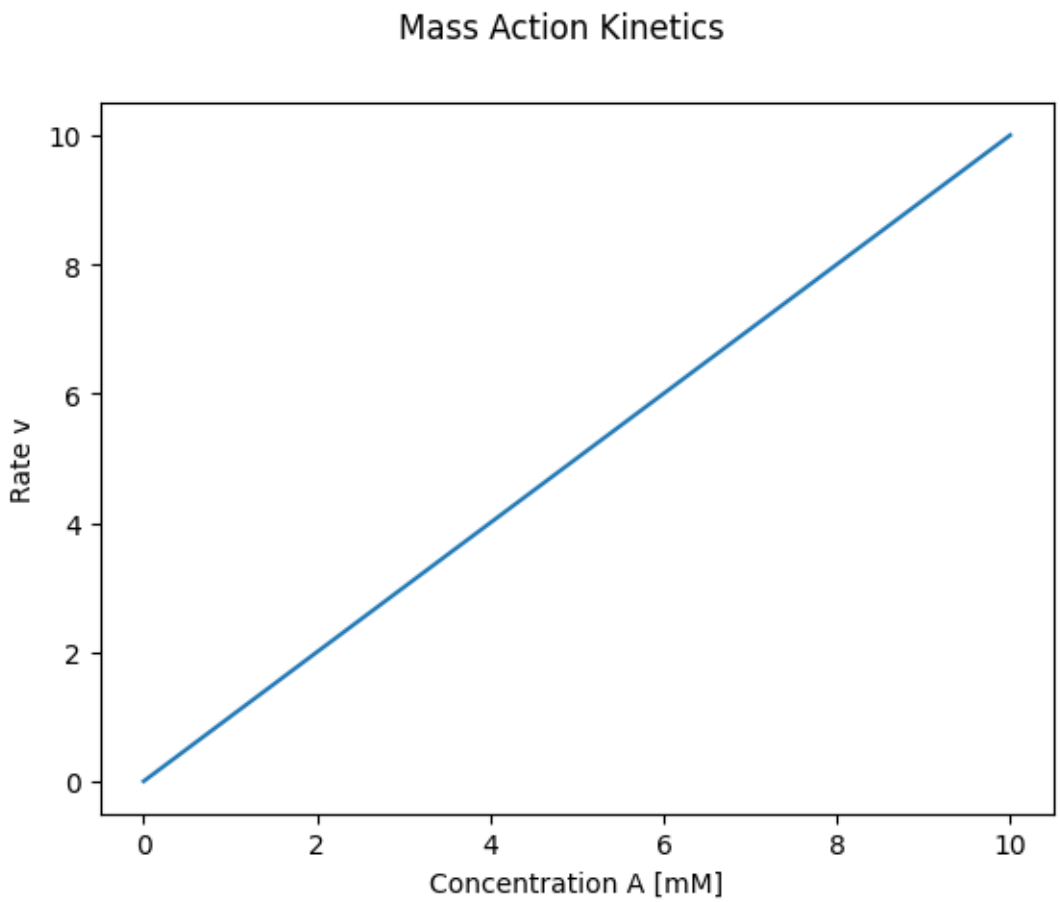
$$v = k \cdot A \cdot B \tag{18.1}$$

where k is the rate constant, and A and B are the concentrations of A and B.

In case of reaction A $\rightarrow$ B, the rate would be expressed as

$$v = k \cdot A \tag{18.2}$$

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

k = 1.0
A = np.linspace(0, 10, num=100)  # [mM]
v = k * A

f, ax = plt.subplots(nrows=1, ncols=1)
f.suptitle("Mass Action Kinetics")
ax.plot(A, v)
ax.set_xlabel("Concentration A [mM]")
ax.set_ylabel("Rate v")
plt.show()
```



Exercise: Plot the dependency of the rate v on the rate constant k. I.e. perform a parameter scan and show the rate curves for different k values.

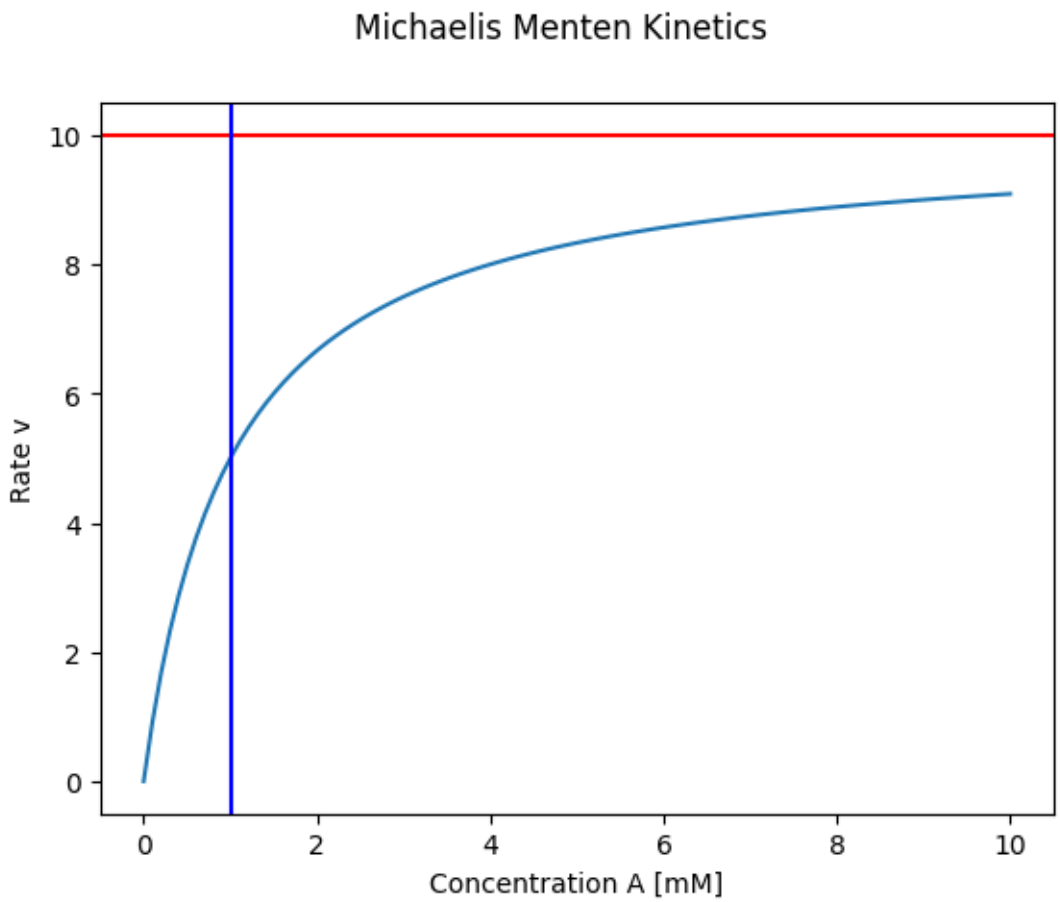
18.2.2 Michaelis-Menten Model

This model is used to describe enzyme-catalyzed reactions, particularly when enzyme concentrations are much lower than substrate concentrations. The rate of reaction (v) is given by the equation

$$v = \frac{Vmax \cdot A}{Km + A} \tag{18.3}$$

, where Vmax is the maximum rate, A is the substrate concentration, and Km is the Michaelis constant (the substrate concentration at which the reaction rate is half of Vmax).

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

Km = 1.0  # [mM]
Vmax = 10
A = np.linspace(0, 10, num=100)  # [mM]
v = Vmax * A/(Km + A)

f, ax = plt.subplots(nrows=1, ncols=1)
f.suptitle("Michaelis Menten Kinetics")
ax.plot(A, v)
ax.axhline(y=10, color='r', linestyle='-', label="Vmax")
ax.axvline(x=1, color='b', linestyle='-', label="Km")
ax.set_xlabel("Concentration A [mM]")
ax.set_ylabel("Rate v")
plt.show()
```



Exercise: Plot the dependency of the rate v on the parameters Km and Vmax. I.e. perform a parameter scan and show the curves for different Km and Vmax values.

18.2.3 Hill Equation

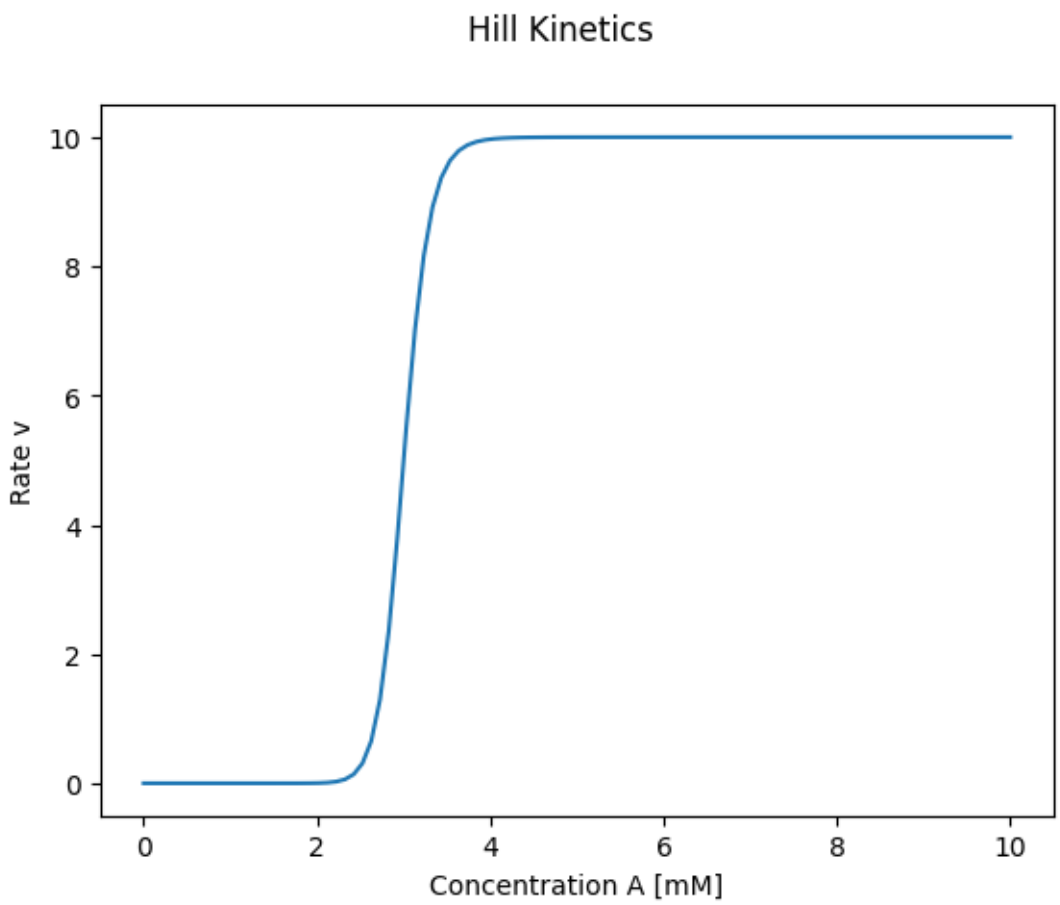
This model is often used when there is cooperativity or interaction between multiple binding sites on a molecule (like a protein or enzyme). The equation is given by

$$v = \frac{Vmax \cdot A^n}{Kd^n + A^n} \tag{18.4}$$

, where n is the Hill coefficient representing the degree of cooperativity.

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

```
Km = 3.0  # [mM]
Vmax = 10
n = 20
A = np.linspace(0, 10, num=100)  # [mM]
v = Vmax * A**n/(Km**n + A**n)

f, ax = plt.subplots(nrows=1, ncols=1)
f.suptitle("Hill Kinetics")
ax.plot(A, v)
ax.set_xlabel("Concentration A [mM]")
ax.set_ylabel("Rate v")
plt.show()
```



Exercise: Plot the dependency of the rate v on the parameters Km and n. I.e. perform a parameter scan and show the curves for different Km and n values.

18.2.4 Inhibition

A simple competitive inhibition term for a rate equation is

$$(\frac{1}{Ki + I}) \tag{18.5}$$

with Ki being the inhibition constant and I being the concentration of the inhibitor.

19 Pathway model of metabolism

Our next step are simple metabolic networks. The basic ingredients for model construction are the stoichiometry, the rate equations, and the parameters.

We start with a simple linear chain,

$$X \longrightarrow S_1 \longrightarrow S_2 \longrightarrow S_3 \longrightarrow Y \tag{19.1}$$

and irreversible Michaelis-Menten equations

$$v_i = \frac{V_{\max} S_i}{K_M + S_i} \tag{19.2}$$

The external metabolites X and Y are like (constant) parameters. The concentration of X is included into the (constant) influx value.

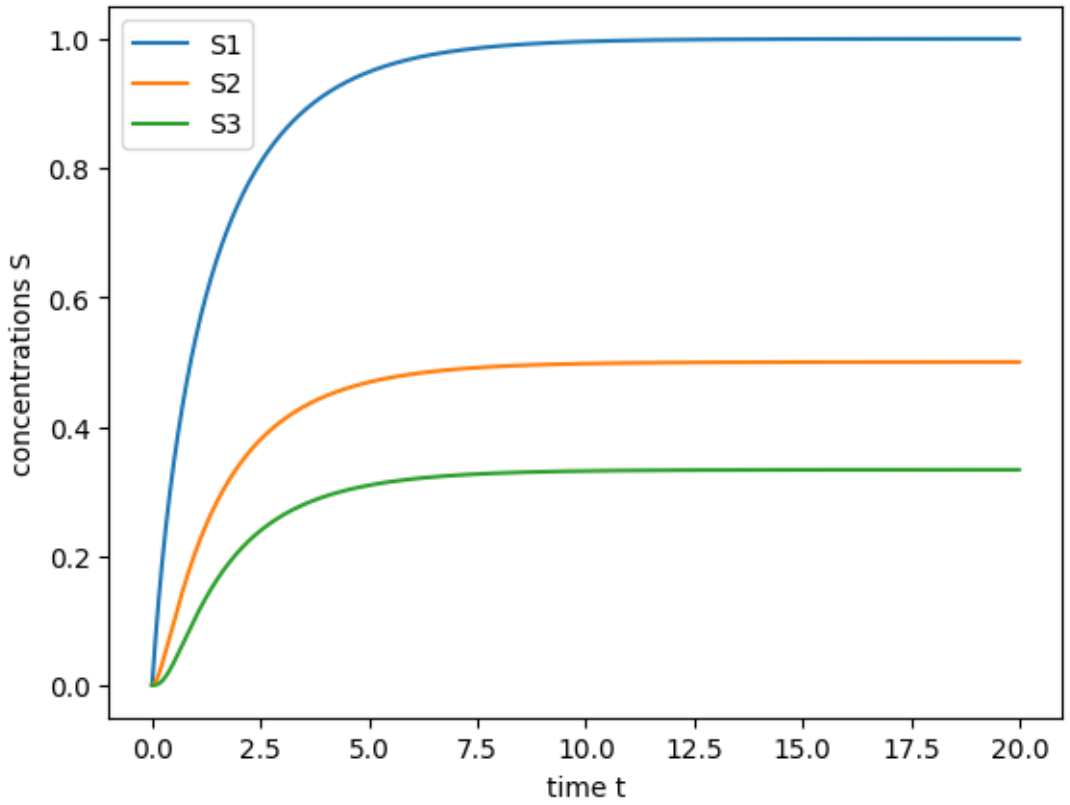
The python function might look like this:

```
def LinearChain(S,t):
    """ Implements the simple linear chain with irreversible MM kinetics. """

    # define metabolites
    S1 = S[0]
    S2 = S[1]
    S3 = S[2]

    # define parameters
    influx = 1.0
    Km1 = 1.0; Vm1 = 2.0
    Km2 = 1.0; Vm2 = 3.0
    Km3 = 1.0; Vm3 = 4.0

    # define rate functions
    v0 = influx # constant
    v1 = Vm1*S1/(Km1+S1)
    v2 = Vm2*S2/(Km2+S2)
```

```
    v3 = Vm3*S3/(Km3+S3)

    # define stoichiometry
    dS1dt = +v0 - v1
    dS2dt = +v1 - v2
    dS3dt = +v2 - v3

    return [dS1dt, dS2dt, dS3dt]
```

To integrate the pathway, we use `odeint`:

```
import numpy as np
from scipy.integrate import odeint
from matplotlib import pyplot as plt

T = np.arange(0,20,0.01)
S0 = [0, 0, 0]

S = odeint(LinearChain,S0,T)
plt.plot(T,S)
plt.legend(['S1','S2','S3'])
plt.xlabel('time t')
plt.ylabel('concentrations S')
plt.show()
```



Exercise: Implement the system using mass action kinetics instead of Michaelis-Menten Kinetics

Exercise: Add an inhibition term to v1 via S3 (inhibition via pathway product)

20 Pharmacokinetic parameters

Pharmacokinetic parameters are numerical values that describe how a drug behaves in the body. They play a vital role in determining the dosage and frequency of drug administration. Here's a summary of some of the key parameters:

1. **Absorption**: This parameter involves how the drug is absorbed into the bloodstream from the site of administration. The rate and extent of absorption can influence the onset, intensity, and duration of a drug's effect.
2. **Distribution**: This refers to how the drug spreads throughout the body. The volume of distribution (Vd) is a key parameter that quantifies the extent to which a drug is distributed in the body's tissues compared to its concentration in the blood.
3. **Metabolism (Biotransformation)**: Metabolism is how the drug is chemically modified or broken down in the body, primarily by liver enzymes. This can change the drug's activity and affects how quickly it's cleared from the body.
4. **Elimination (Excretion)**: This parameter refers to the removal of the drug from the body, primarily through the kidneys (urine) or liver (bile). The rate of elimination is usually expressed as the drug's half-life (t1/2), which is the time it takes for the concentration of the drug in the body to be reduced by half.
5. **Clearance (Cl)**: This is a measure of the body's efficiency in eliminating the drug, expressed as volume/time (like mL/min). It's a crucial parameter that determines the steady-state concentration of the drug for a given dosage regimen.
6. **Bioavailability (F)**: This is the fraction of the administered dose of a drug that reaches the systemic circulation in an unchanged form. It's a crucial parameter, especially for oral medications.
7. **Area Under the Curve (AUC)**: This is a measure of the total exposure of the body to the drug. It's calculated as the integral of the concentration-time curve, from administration to elimination.
8. **Peak Concentration (Cmax) and Time to Reach Peak Concentration (Tmax)**: Cmax is the highest concentration a drug achieves in the body after administration, and Tmax is the time it takes to reach this peak concentration.

These pharmacokinetic parameters are essential in understanding a drug's behavior and in designing optimal drug dosing regimens. By understanding these parameters, healthcare professionals can better predict how a drug will behave, enabling them to administer the drug safely and effectively.

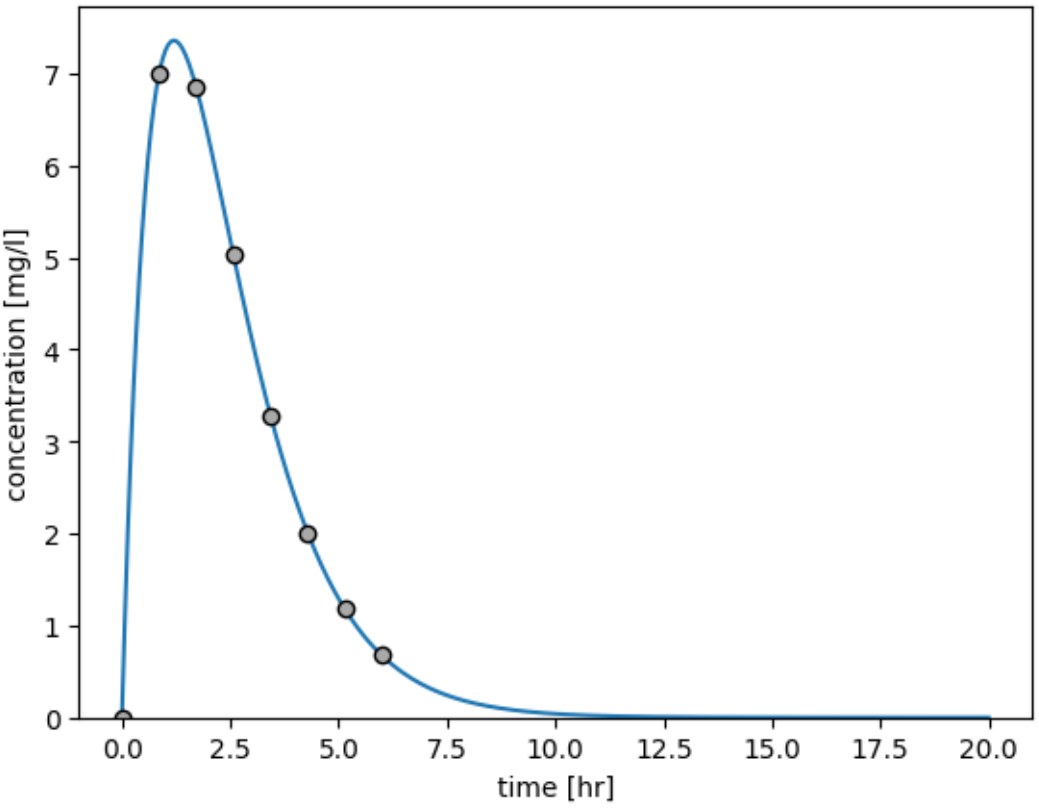
21 Algebraic equation

We calculate the pharmacokinetic parameters based on simple algebraic equation for oral absorption:

$$C(t) = \frac{Dose}{V} \cdot t \cdot e^{-\frac{CL}{V} \cdot t} \tag{21.1}$$

This model states the relationship between the independent variable, `time (t)`, and the dependent variable, concentration (`C`). The notation `C(t)` suggests that `C` depends on `t`. Dose, clearance (CL), and distribution volume (V) are param- eters (constants); they do not change with different values of t.

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

t = np.linspace(0, 20, num=300) # [hr]
t_points = np.linspace(0, 6, num=8)
dose = 100  # [mg]
V = 6.0  # [l]
CL = 5.0  # [L/hr]
C = dose/V * t * np.exp(-CL/V*t)  # [mg/l]
C_points = dose/V * t_points * np.exp(-CL/V*t_points)

f, ax = plt.subplots(nrows=1, ncols=1)
ax.plot(t, C)
ax.plot(t_points, C_points, linestyle="None", marker="o", markeredgecolor="black", color="da
ax.set_xlabel("time [hr]")
ax.set_ylabel("concentration [mg/l]")
ax.set_ylim(bottom=0)
plt.show()
```



21.1 Pharmacokinetic parameters

Based on the example curve we now calculate typical pharmacokinetic parameters of the curve.

```
# overview of the measurement data
import pandas as pd
t_unit = "hr"
c_unit = "mg/l"
df = pd.DataFrame({
    "t": t_points,
    "c": C_points,
})
print(df)
```

```
          t         c
0  0.000000  0.000000
```

```
1  0.857143  6.993452
2  1.714286  6.847172
3  2.571429  5.027964
4  3.428571  3.281864
5  4.285714  2.008261
6  5.142857  1.179753
7  6.000000  0.673795
```

21.1.1 AUC (area under the curve)

The AUC, or Area Under the Curve, is another important pharmacokinetic parameter.

In the context of pharmacokinetics, the "curve" in question is a graph of drug concentration in the bloodstream over time. The AUC is essentially the area under this graph, representing the total exposure of the body to the drug.

AUC is used to quantify the extent of drug absorption. When a drug is administered, it gets absorbed into the bloodstream and then gradually eliminated. A plot of the drug concentration in the blood over time typically forms a curve. The AUC of this curve from the time of administration to a specific time point (say, 24 hours) provides an estimate of the total drug exposure during this period. The AUC from time zero to infinity (AUC0–∞) is often used to estimate the total drug exposure over an infinite period.

AUC is directly proportional to the total amount of unchanged drug that reaches systemic circulation. Therefore, it's a valuable indicator of bioavailability (the extent and rate at which the drug is absorbed and becomes available at the site of action). It's also used to calculate other pharmacokinetic parameters, such as the clearance rate (how quickly the drug is eliminated from the body).

Moreover, AUC is often used in dose-response studies, where it helps in identifying the appropriate dosage for a therapeutic effect, and in comparative bioavailability studies, where it is used to compare the bioavailability of two different pharmaceutical products, such as a brand name drug and its generic version.

```python
def f_auc(t: np.ndarray, c: np.ndarray):
    """Calculate the area under the curve (AUC) via trapezoid rule.

    :param t = time array
    :param c = concentration array
    :param rm_nan = remove nan values array
    """
    auc = np.sum((t[1:] - t[0:-1]) * (c[1:] + c[0:-1]) / 2.0)
    return auc
```

```
auc = f_auc(t=df.t.values, c=df.c.values)
print(f"AUC: {auc:.2f} [{c_unit}*{t_unit}]")
```

```
AUC: 22.01 [mg/l*hr]
```

21.1.2 Time to maximum (tmax) and maximum concentration (cmax)

1. **Tmax (Time of maximum concentration):** This parameter represents the time it takes for a drug to reach its maximum concentration (Cmax) in the bloodstream after administration. It can provide insight into the rate of absorption of the drug. For example, a shorter Tmax could suggest faster absorption. However, many factors can affect Tmax, including the route of administration, the drug's formulation, and individual physiological differences.
2. **Cmax (Maximum or peak concentration):** This parameter refers to the highest concentration that a drug achieves in the body after administration, and before elimination processes reduce it. Cmax is a crucial factor in determining the efficacy and potential toxicity of a drug. This parameter can be influenced by factors such as the dose, rate of administration, and rate of absorption and elimination.

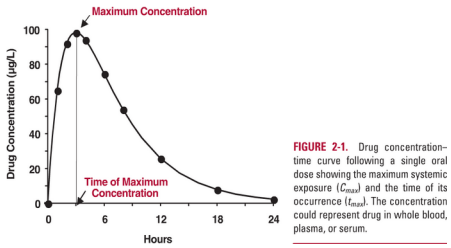


Figure 21.1: Cmax and tmax

```
def f_max(t, c):
    """Return timepoint of maximal value and maximal value based on curve.

    The tmax depends on the value of both the absorption rate constant (ka)
    and the elimination rate constant (kel).

    :return: tuple (tmax, cmax)
    """
    idx = np.nanargmax(c)
    return t[idx], c[idx]
```

```
tmax, cmax = f_max(t=df.t.values, c=df.c.values)
print(f"tmax: {tmax:.2f} [{t_unit}]")
print(f"cmax: {cmax:.2f} [{c_unit}]")
```

```
tmax: 0.86 [hr]
cmax: 6.99 [mg/l]
```

22 Elimination rate kel, Half-life thalf and AUCinf

The elimination rate constant (Kel) and half-life (T1/2) are critical pharmacokinetic parameters that describe how a drug is eliminated from the body.

1. **Elimination Rate Constant (Kel):** This parameter describes the rate at which a drug is removed from the body. It's usually expressed in units of time^-1 (such as hours^-1). Kel is calculated using the formula Kel = ln(2) / T1/2. A larger Kel means the drug is eliminated more rapidly.
2. **Half-Life (T1/2):** This is the time it takes for the concentration of the drug in the body (or in blood plasma) to reduce by half. It's a measure of how quickly the drug is being eliminated from the body. The half-life is calculated using the formula T1/2 = ln(2) / Kel. Drugs with shorter half-lives are eliminated from the body more quickly than those with longer half-lives. This parameter is particularly important when determining dosing intervals for a medication.

These two parameters provide vital information about how long a drug will remain in the body, which can guide decisions about the frequency and timing of doses to maintain the drug concentration within a therapeutic range. They can also help predict the time it will take to eliminate a drug from the body completely, which can be especially important in situations where a patient experiences toxic effects from a medication or needs to switch to a different drug.

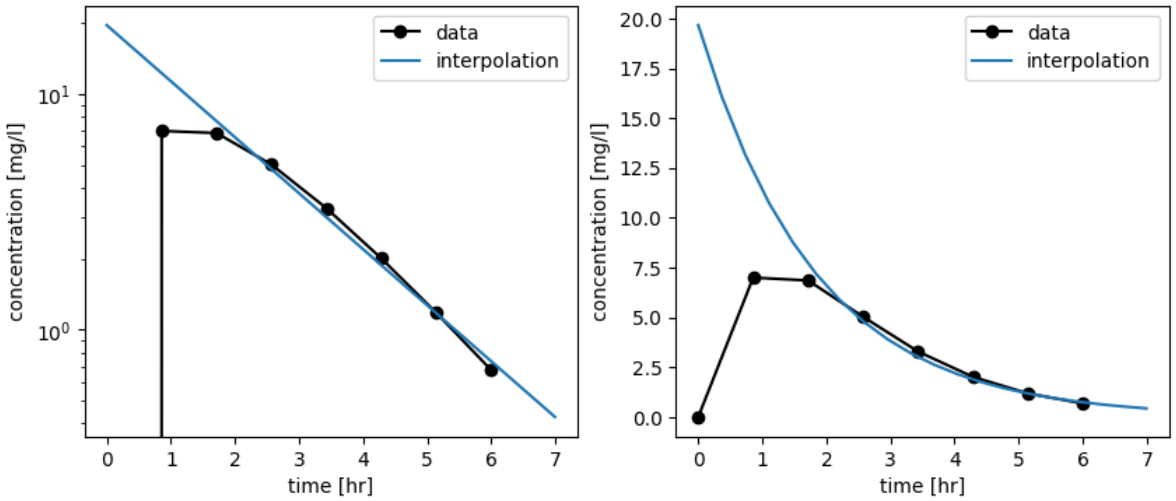
For the calculation of kel, thalf and auc_inf we have to perform a linear regression on the log pharmacokinetic timecourse.

```
from scipy import stats

def f_ols_regression(t, c):
    """Linear regression on the log timecourse after maximal value.

    The linear regression is calculated from all data points after the maximum.

    :return:
    """
```

```
    # linear regression start regression on data point after maximum
    max_index = np.nanargmax(c)
    x = t[max_index + 1 :]
    y = np.log(c[max_index + 1 :])

    # using mask to remove nan values
    mask = ~np.isnan(x) & ~np.isnan(y)
    slope, intercept, r_value, p_value, std_err = stats.linregress(x[mask], y[mask])

    return [slope, intercept]

[slope, intercept] = f_ols_regression(df.t.values, df.c.values)

print(f"slope: {slope:.2f} [{c_unit}/{t_unit}/]")
print(f"intercept: {intercept:.2f} [{c_unit}]")
```

```
slope: -0.55 [mg/l/hr/]
intercept: 2.98 [mg/l]
```

```
# plot the interpolation results
f, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(10, 4))

tvec = np.linspace(0, df.c.max(), num=20)
for ax in (ax1, ax2):
    ax.plot(df.t, df.c, "-o", color="black", label="data")
    ax.plot(tvec, np.exp(intercept + slope*tvec), label="interpolation")

    ax.set_xlabel("time [hr]")
    ax.set_ylabel("concentration [mg/l]")
    ax.legend()

ax1.set_yscale("log")
plt.show()
```



Based on the interpolation results we can calculate kel, thalf and aucinf.

```
[slope, intercept] = f_ols_regression(df.t.values, df.c.values)
kel = -slope
thalf = np.log(2) / kel
print(f"kel: {kel:.2f} [{1}/{t_unit}]")
print(f"thalf: {thalf:.2f} [{t_unit}]")

# by integrating from tend to infinity for c[-1]exp(slope * t) we get
auc = f_auc(df.t.values, df.c.values)
auc_delta = -df.c.values[-1] / slope
aucinf = auc + auc_delta
print(f"AUC: {auc:.2f} [{c_unit}*{t_unit}]")
print(f"AUCinf: {aucinf:.2f} [{c_unit}*{t_unit}]")
```

```
kel: 0.55 [1/hr]
thalf: 1.27 [hr]
AUC: 22.01 [mg/l*hr]
AUCinf: 23.24 [mg/l*hr]
```

22.0.1 Volume of distribution (Vd) and Clearance (CL)

Volume of Distribution (Vd) and Clearance (CL) are two more fundamental pharmacokinetic parameters:

Volume of Distribution (Vd): The volume of distribution is a theoretical volume that represents how a drug would be distributed in the body if the drug were uniformly distributed

and the concentration in all tissues was the same as in the blood. In reality, drugs are not uniformly distributed, but the concept of Vd allows for a simplified way to understand how drugs disperse throughout the body. It is usually expressed in liters (L).

A small Vd indicates the drug is primarily confined to the blood or plasma, while a large Vd suggests the drug is extensively distributed into the tissues. Factors such as the drug's lipid solubility, the extent of protein binding, and the drug's ability to cross cell membranes can influence the Vd.

Clearance (CL): Clearance describes the body's efficiency in eliminating the drug, essentially representing the volume of plasma from which the drug is completely removed per unit of time (e.g., mL/min or L/h).

Total body clearance can occur via different routes, primarily renal (kidneys) and hepatic (liver) clearance. A high clearance rate indicates the drug is rapidly removed from the body. Factors influencing clearance include organ blood flow (mainly liver and kidneys), the drug's affinity for metabolizing enzymes, and the extent of protein binding.

These parameters are crucial in understanding a drug's behavior in the body and in designing optimal dosing regimens. They are particularly important in situations such as renal or hepatic impairment, where clearance might be significantly reduced, necessitating dosage adjustments.

```
vd = dose / (aucinf * kel)
vd_unit = "l"
cl = kel * vd
cl_unit = "l/hr"

print(f"AUC: {auc:.2f} [{c_unit}*{t_unit}]")
print(f"AUCinf: {aucinf:.2f} [{c_unit}*{t_unit}]")
print(f"Vd: {vd:.2f} [{vd_unit}]")
print(f"CL: {cl:.2f} [{cl_unit}]")
```

```
AUC: 22.01 [mg/l*hr]
AUCinf: 23.24 [mg/l*hr]
Vd: 7.86 [l]
CL: 4.30 [l/hr]
```

23 Application of pharmacokinetic parameters

We can now apply the calculation of pharmacokinetic parameters to evaluate the effect of changes in the system. First we define the function for calculation of the pharmacokinetic parameters in a file `helpers.py`. The functionality can then be imported and reused in the following

```
from scipy import stats
import numpy as np

def print_pk(pk):
    """Print pk information"""
    lines = []
    keys = [key for key in pk if not "unit" in key]
    for key in keys:
        unit = pk[f"{key}_unit"]
        lines.append(
            f"{key:<10}: {pk[key]:.2f} [{unit}]"
        )
    info = "\n".join(lines)
    print(info)

def f_pk(t, c, dose, show: bool = False):
    """Calculate PK information."""
    dose_unit = "mg"
    t_unit = "hr"
    c_unit = "mg/l"
    auc_unit = f"{c_unit}*{t_unit}"
    cl_unit = "l/hr"
    vd_unit = "l"

    auc = np.sum((t[1:] - t[0:-1]) * (c[1:] + c[0:-1]) / 2.0)
    idx = np.nanargmax(c)
    tmax, cmax = t[idx], c[idx]

    max_index = np.nanargmax(c)
    x = t[max_index + 1:]
```

```
    y = np.log(c[max_index + 1:])

    # using mask to remove nan values
    mask = ~np.isnan(x) & ~np.isnan(y)
    slope, intercept, r_value, p_value, std_err = stats.linregress(x[mask], y[mask])

    kel = -slope
    thalf = np.log(2) / kel

    auc_delta = -c[-1] / slope
    aucinf = auc + auc_delta

    vd = dose / (aucinf * kel)
    cl = kel * vd

    pk = {
        'dose': dose,
        'dose_unit': dose_unit,
        'auc': auc,
        'auc_unit': auc_unit,
        'aucinf': aucinf,
        'aucinf_unit': auc_unit,
        'tmax': tmax,
        'tmax_unit': t_unit,
        'cmax': cmax,
        'cmax_unit': c_unit,
        'thalf': thalf,
        'thalf_unit': t_unit,
        'kel': kel,
        'kel_unit': f"1/{t_unit}",
        'vd': vd,
        'vd_unit': vd_unit,
        'cl': cl,
        'cl_unit': cl_unit,
    }
    if show:
        print_pk(pk)

    return pk
```

```
from helpers import f_pk
f_pk(t=df.t.values, c=df.c.values, dose=dose, show=True);
```

```
dose      : 100.00 [mg]
auc       : 22.01 [mg/l*hr]
aucinf    : 23.24 [mg/l*hr]
tmax      : 0.86 [hr]
cmax      : 6.99 [mg/l]
thalf     : 1.27 [hr]
kel       : 0.55 [1/hr]
vd        : 7.86 [l]
cl        : 4.30 [l/hr]
```

In the following we study a simple model for absorption and elimination. A can be absorbed from the tablet in the systemic circulation ($A_{tablet} \rightarrow A_{system}$) which can be eliminated in the urine via renal excretion ($A_{system} \rightarrow A_{urine}$).

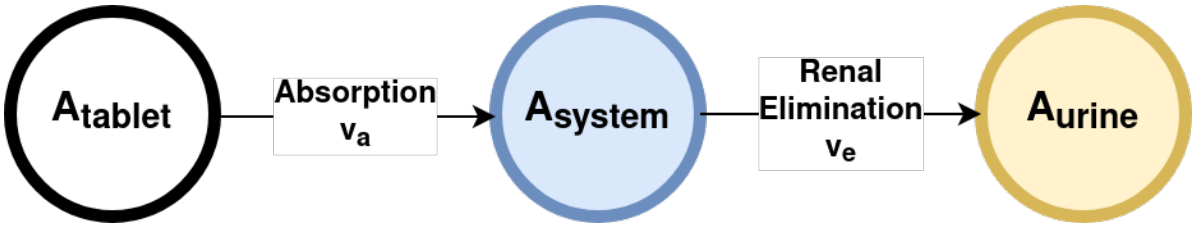


Figure 23.1: Absorption Elimination Model

Elimination and absorption are assumed to be Mass-Action, i.e., depending on a rate constant k and the amount or concentration of the respective substance.

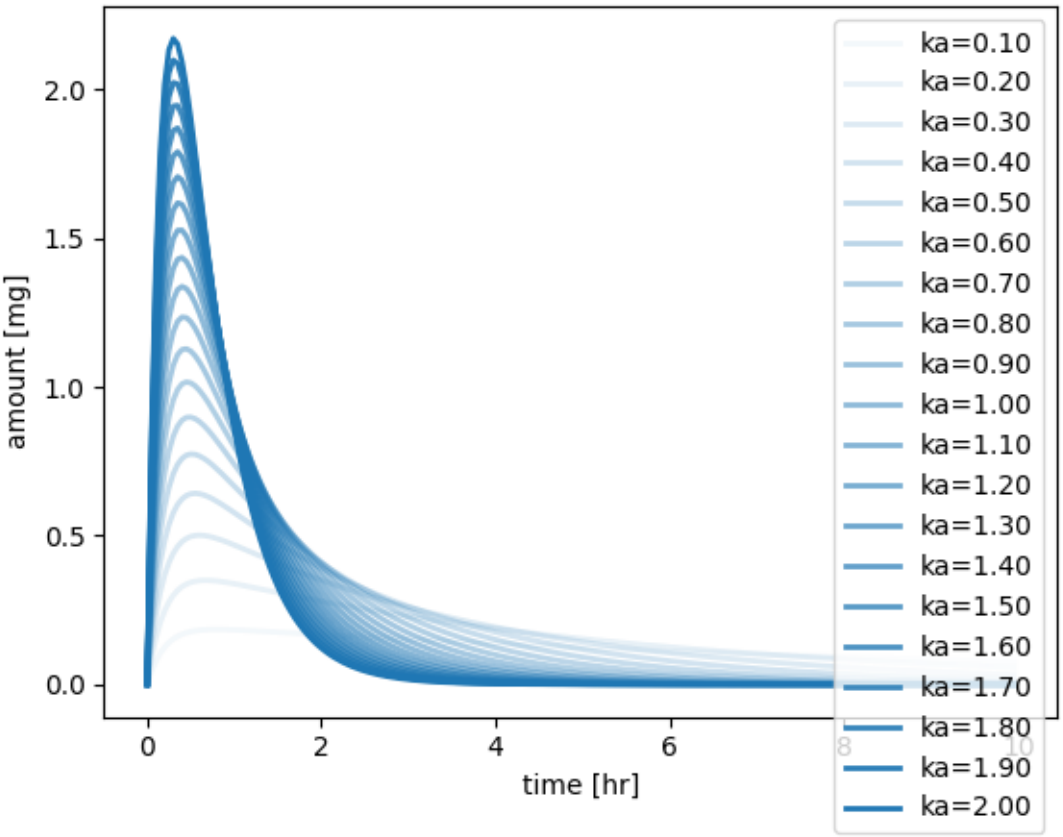
The ordinary differential equation system (ODE) results in:

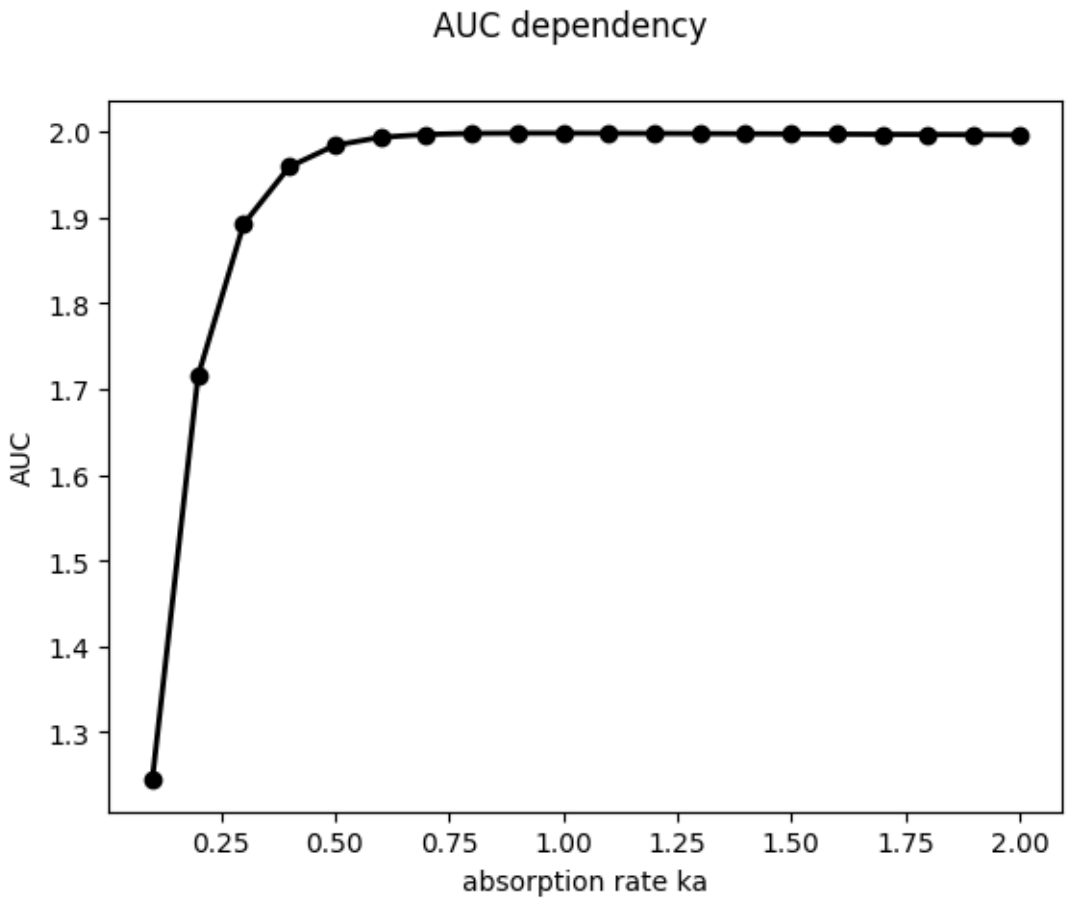
```python
# define the ode system for reuse in helpers.py

def dxdt_absorption_first_order(x, t, ka, ke):
    """
    First order absorption model
    """
    # state variables
    A_tablet = x[0]  # [mg]
    A_central = x[1] # [mg/l]
    A_urine = x[2] # [mg]

    # rates
    va = ka * A_tablet  # [mg/hr]
    ve = ke * A_central # [mg/hr]
```

```
    # odes (stoichiometric equation)
    return [
        -va,            # dA_tablet/dt  [mg/hr]
         va - ve,       # dA_central/dt [mg/hr]
         ve,            # dA_urine/dt   [mg/hr]
    ]
```

```
from helpers import dxdt_absorption_first_order, f_pk

import numpy as np
from scipy.integrate import odeint
from matplotlib import pylab as plt

# initial condition and time span
t = np.arange(0, 10, 0.05) # [hr]
Dose_A = 10.0  # [mg]
x0 = [
    Dose_A,  # A_tablet  [mg]
    0.0,   # A_central [mg]
    0.0,   # A_urine [mg]
]

# parameters
ka = 2.0  # [1/hr]
ke = 5.0  # [1/hr]

x = odeint(dxdt_absorption_first_order, x0, t, args=(ka, ke))

n_samples = 20
kas = np.linspace(0.1, 2.0, num=n_samples)  # [1/hr]
tcs = []
pks = []

# simulate all the different absorption
for kp, ka in enumerate(kas):
    x = odeint(dxdt_absorption_first_order, x0, t, args=(ka, ke))
    df = pd.DataFrame(x, columns=["A_tablet", "A_central", "A_urine"])
    df["time"] = t
    tcs.append(df)

    # calculate pharmacokinetics parameters on the curves
    # print(f"{ka=}")
```

```
    pk = f_pk(t=df.time.values, c=df.A_central.values, dose=Dose_A, show=False)
    # print("-" * 80)
    pks.append(pk)


# plot timecourse
f, ax = plt.subplots(nrows=1, ncols=1)
ax.set_xlabel("time [hr]")
ax.set_ylabel("amount [mg]")

for k, ka in enumerate(kas):
    tc = tcs[k]
    ax.plot(
        tc.time, tc.A_central, linewidth=2, color="tab:blue",
        alpha=(k+1)/n_samples, label=f"{ka=:.2f}"
    )

ax.legend()
plt.show()

# plot AUC
f, ax = plt.subplots(nrows=1, ncols=1)
f.suptitle("AUC dependency")
aucs = [pk["auc"] for pk in pks]
ax.plot(kas, aucs, linewidth=2, color="black", label=f"auc [{pk['auc_unit']}]", marker="o")
ax.set_xlabel("absorption rate ka")
ax.set_ylabel("AUC")
plt.show()
```


amount [mg]
2.0
1.5
1.0
0.5
0.0
0
2
4
6
8
10
time [hr]
ka=0.10
ka=0.20
ka=0.30
ka=0.40
ka=0.50
ka=0.60
ka=0.70
ka=0.80
ka=0.90
ka=1.00
ka=1.10
ka=1.20
ka=1.30
ka=1.40
ka=1.50
ka=1.60
ka=1.70
ka=1.80
ka=1.90
ka=2.00



Exercise 1: Study the dependency of the other pharmacokinetic parameters on the absorption rate ka. I.e. plot the dependency for Cmax, tmax, Cl, Vd, …

Exercise 2: What is the effect of changing the urinary excretion rate parameter `ke` on the pharmacokinetic parameters. Perform a similar analysis as for `ka`.

24 Variability in pharmacokinetics

Pharmacokinetics is the study of how a drug moves through the body. The four primary aspects of pharmacokinetics are absorption, distribution, metabolism, and excretion (ADME). Pharmacokinetic variability refers to the differences in drug response among individuals due to various factors. Understanding these factors is crucial for optimizing drug therapy and achieving personalized medicine.

Figure 24.1: Interindividual variability

Here are key points about the factors that result in pharmacokinetic variability:

1. **Variability in Protein Amounts of Transporters and Enzymes**:
    - **Role of Proteins**: Proteins play a critical role in drug transportation and metabolism. Transporter proteins help move drugs across cell membranes, while metabolic enzymes break down drugs in the body.
    - **Genetic Factors**: The expression levels and activity of these proteins can vary significantly among individuals due to genetic variations. For example, cytochrome P450 enzymes, responsible for metabolizing many drugs, can have genetic polymorphisms leading to different metabolic rates.
    - **Disease States**: Conditions such as liver or kidney disease can alter the expression and function of these proteins, impacting drug pharmacokinetics.
    - **Interactions**: Substances like grapefruit juice can inhibit certain enzymes, affecting drug metabolism.
2. **Pharmacogenomic Variants**:

- **Genetic Makeup**: Pharmacogenomics studies how genetic differences affect drug response. Variations in genes encoding drug-metabolizing enzymes, transporters, and receptors can influence drug efficacy and toxicity.
- **Examples**: Individuals with certain variants of the CYP2D6 gene may metabolize drugs like codeine either too quickly or too slowly, leading to ineffective treatment or adverse effects.

3. **Physiological Factors**:

   - **Age**: Older adults typically have decreased organ function and altered body composition, affecting drug metabolism and distribution.
   - **Sex**: Men and women may metabolize drugs differently due to hormonal differences and body composition.
   - **Body Weight**: Drug dosing often needs to be adjusted based on body weight to achieve the desired therapeutic effect.
   - **Organ Function**: Impaired liver or kidney function can significantly affect drug metabolism and excretion.
   - **Health Status**: Chronic conditions like diabetes or heart disease can alter drug pharmacokinetics.

4. **Environmental Factors**:

   - **Diet**: Certain foods can interact with drugs, influencing their absorption and metabolism. For instance, high-fat meals can enhance the absorption of lipophilic drugs.
   - **Lifestyle**: Alcohol consumption, smoking, and exposure to environmental toxins can induce or inhibit drug-metabolizing enzymes, altering drug levels in the body.
   - **Example**: Smoking induces CYP1A2, which can increase the metabolism of certain drugs like theophylline.

5. **Drug-Drug Interactions**:

   - **Mechanisms**: Some drugs can inhibit or induce the enzymes that metabolize other drugs, leading to altered drug levels and potential adverse effects.
   - **Examples**: Co-administration of ritonavir with other protease inhibitors can boost their levels by inhibiting their metabolism.

6. **Patient Compliance**:

   - **Adherence**: How well a patient follows their prescribed medication regimen can significantly impact drug efficacy. Missed doses, incorrect timing, or improper administration can all affect drug pharmacokinetics.
   - **Strategies**: Educating patients about the importance of adherence and simplifying dosing regimens can improve compliance.

24.0.1 Importance of Understanding Pharmacokinetic Variability

Understanding these factors is critical in personalized medicine as it allows for the optimization of drug therapy based on an individual's unique characteristics. This leads to improved drug efficacy, reduced adverse effects, and better overall patient outcomes. Healthcare professionals must consider these variables when prescribing medications and monitoring therapeutic responses.

24.1 Variability in Protein Amounts of Transporters and Proteins

A large part of the variability in drug metabolism within the population is due to individual protein amounts.

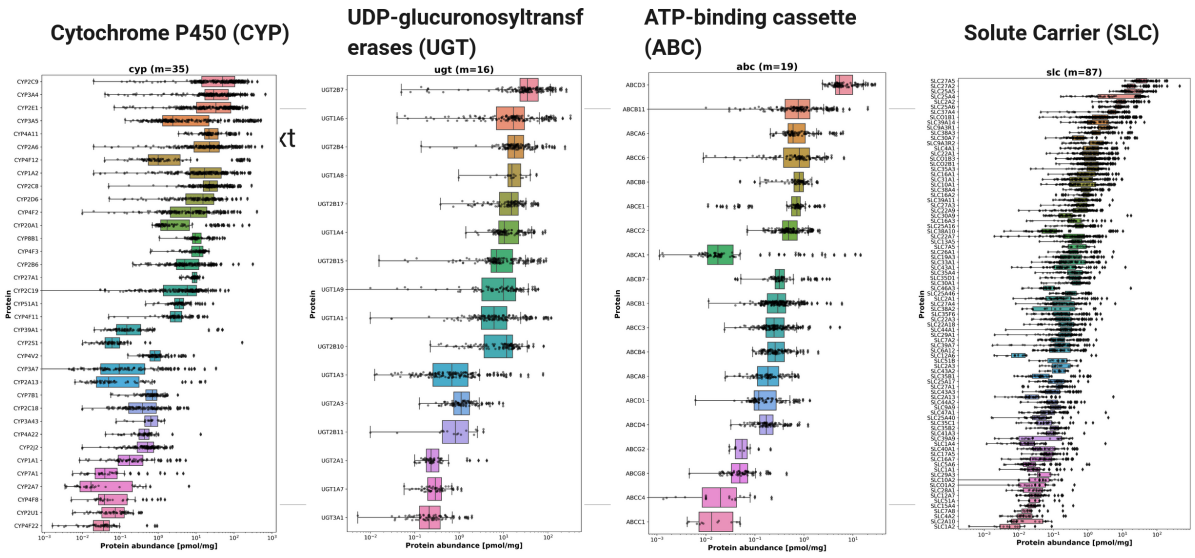


Figure 24.2: Protein variability

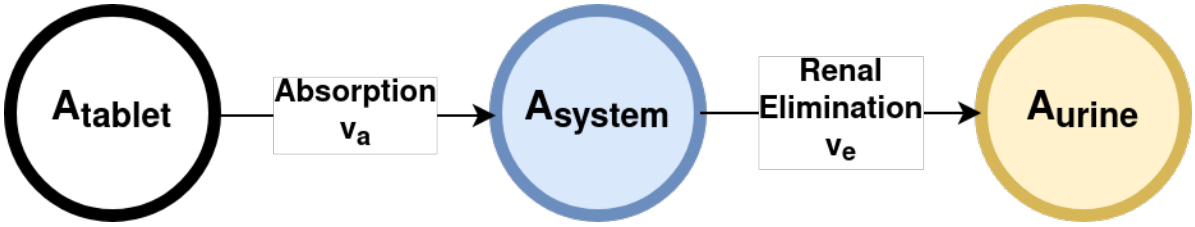


Figure 24.3: Absorption Elimination Model

24.1.1 Defining the Protein Distribution

To model the variability in protein amounts, we first need to define the distribution from which we will sample. We use a lognormal distribution for this purpose, as it is well-suited for representing biological data, including protein concentrations, which often exhibit right-skewed distributions.

A lognormal distribution is defined such that the logarithm of the variable is normally distributed. This characteristic makes it ideal for modeling data that cannot be negative and that exhibit multiplicative rather than additive variability.

For more details on the lognormal distribution and its properties, you can refer to the scipy.stats.lognorm documentation.

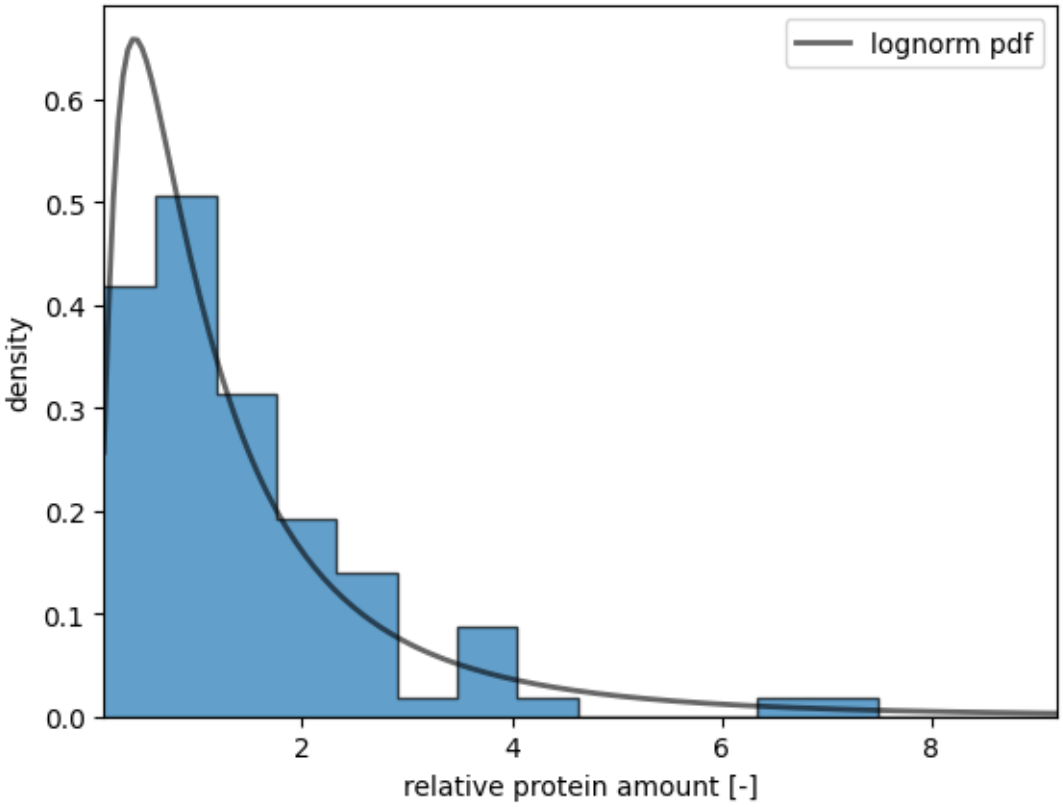
24.1.2 Why Use a Lognormal Distribution?

- **Biological Relevance**: Protein concentrations in biological systems are often right-skewed and span several orders of magnitude, characteristics that are well-captured by a lognormal distribution.
- **Non-Negativity**: Since protein amounts cannot be negative, the lognormal distribution is appropriate because it is defined for positive values only.
- **Multiplicative Effects**: Biological processes that affect protein levels (such as gene expression and degradation) often have multiplicative effects, aligning well with the lognormal distribution.

By using a lognormal distribution, we can more accurately model and simulate the variability observed in protein amounts across different samples or populations.

```
import numpy as np
from matplotlib import pyplot as plt
from scipy.stats import lognorm
np.random.seed(1234) # random seed for reproducibility

s = 0.954

# define the range and calculate the probability density function (pdf):
x = np.linspace(lognorm.ppf(0.01, s), lognorm.ppf(0.99, s), num=200)
y = lognorm.pdf(x, s)

# sample from distribution
n_samples = 100
f_proteins = lognorm.rvs(s, size=n_samples)
```

```
# plot a histogram of samples
fig, ax = plt.subplots()
ax.hist(f_proteins, density=True, bins='auto', histtype='stepfilled', alpha=0.7,
        edgecolor="black")
# plot the exact distribution
ax.plot(x, y, 'k-', lw=2, alpha=0.6, label='lognorm pdf')
ax.set_xlim([x[0], x[-1]])
ax.legend(loc='best')
ax.set_xlabel("relative protein amount [-]")
ax.set_ylabel("density")
plt.show()
```



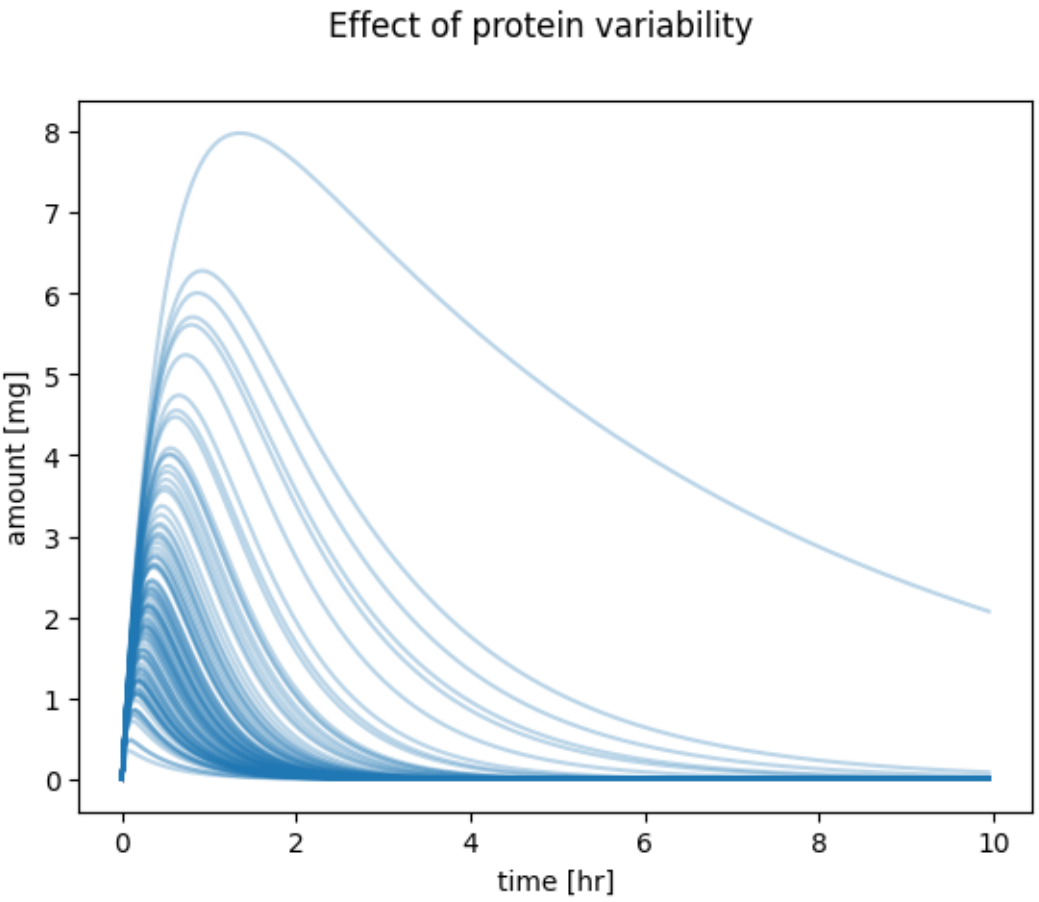
24.2 Calculating Individual Pharmacokinetics

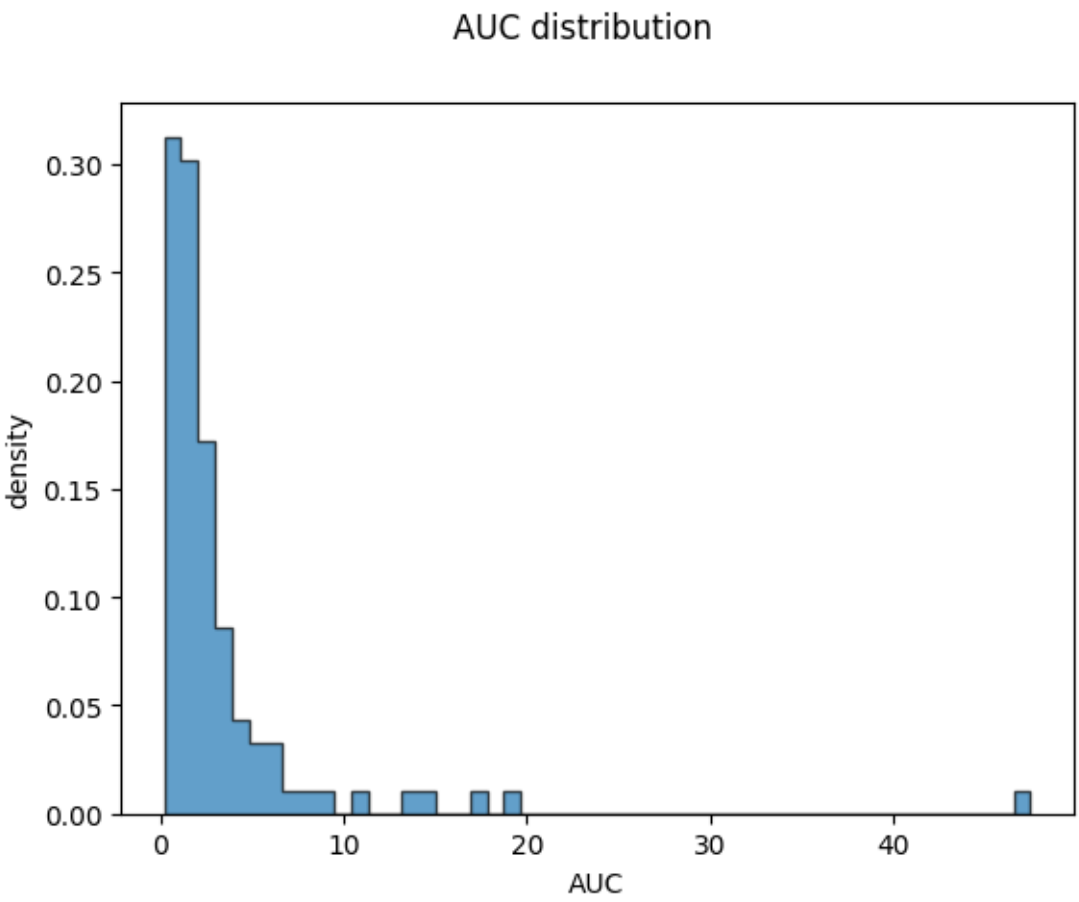
Next, we simulate the resulting distribution of pharmacokinetics based on the protein distribution. In our example, we sample the protein amount of a transporter responsible for the renal

excretion of the drug. By sampling from the lognormal distribution of this protein's amounts, we can model how variability in its expression impacts pharmacokinetic parameters such as absorption, distribution, metabolism, and excretion. This approach allows us to understand and predict the range of pharmacokinetic behaviors in a population, considering individual differences in protein expression.

```
import numpy as np
import pandas as pd
from scipy.integrate import odeint
from matplotlib import pylab as plt

from helpers import f_pk, dxdt_absorption_first_order

# initial condition and time span
t = np.arange(0, 10, 0.05) # [hr]
Dose_A = 10.0  # [mg]
x0 = [
    Dose_A,  # A_tablet  [mg]
    0.0,   # A_central [mg]
    0.0,   # A_urine [mg]
]

# parameters
ka = 2.0  # [1/hr]
ke = 5.0  # [1/hr]

# simulate pharmacokinetics for indivdual proteins
tcs = []
pks = []

for f_protein in f_proteins:

    # the genetic variants effect the elimination/transport rate
    ke_protein = f_protein * ke
    x = odeint(dxdt_absorption_first_order, x0, t, args=(ka, ke_protein))
    df = pd.DataFrame(x, columns=["A_tablet", "A_central", "A_urine"])
    df["time"] = t
    tcs.append(df)

    # calculate pharmacokinetics parameters on the curves
    pk = f_pk(t=df.time.values, c=df.A_central.values, dose=Dose_A, show=False)
    pks.append(pk)
```

```
# plot timecourse
f, ax = plt.subplots(nrows=1, ncols=1)
f.suptitle("Effect of protein variability")
ax.set_xlabel("time [hr]")
ax.set_ylabel("amount [mg]")

for k, f_protein in enumerate(f_proteins):
    tc = tcs[k]
    ax.plot(tc.time, tc.A_central, color="tab:blue", alpha=0.3)
plt.show()

# plot AUC
f, ax = plt.subplots(nrows=1, ncols=1)
f.suptitle("AUC distribution")
aucs = [pk["auc"] for pk in pks]
ax.hist(aucs, density=True, bins='auto', histtype='stepfilled', alpha=0.7,
        edgecolor="black")

ax.set_xlabel("AUC")
ax.set_ylabel("density")
plt.show()
```


Effect of protein variability
amount [mg]
time [hr]
0
1
2
3
4
5
6
7
8
0
2
4
6
8
10



24.2.1 Exercise: Analyzing the Distribution of Pharmacokinetic Parameters

Objective: Investigate how the distribution of a protein transporter responsible for renal excretion affects other pharmacokinetic parameters in the population. This includes parameters such as the half-life $t_{1/2}$ or elimination rate constant k_{el}, and others.

Steps:

1. **Sample Protein Amounts**: Sample from the lognormal distribution of the protein transporter responsible for renal excretion.
2. **Calculate Pharmacokinetic Parameters**: Use the sampled protein amounts to calculate various pharmacokinetic parameters. For instance:
    - **Half-life $t_{1/2}$
    - **Elimination Rate Constant** k_{el}

3. **Plot Histograms**: Generate histograms for these pharmacokinetic parameters to visualize their distributions in the population.

24.2.2 Insights

By plotting these histograms, we can: - **Visualize Variability**: Understand how variability in protein transporter expression translates to variability in pharmacokinetic parameters. - **Predict Population Behavior**: Predict the range of pharmacokinetic responses in a population, which is critical for dosing strategies. - **Inform Personalized Medicine**: Use this information to tailor drug therapies based on individual differences in protein expression and pharmacokinetic responses.

24.3 Achieving the Therapeutic Range

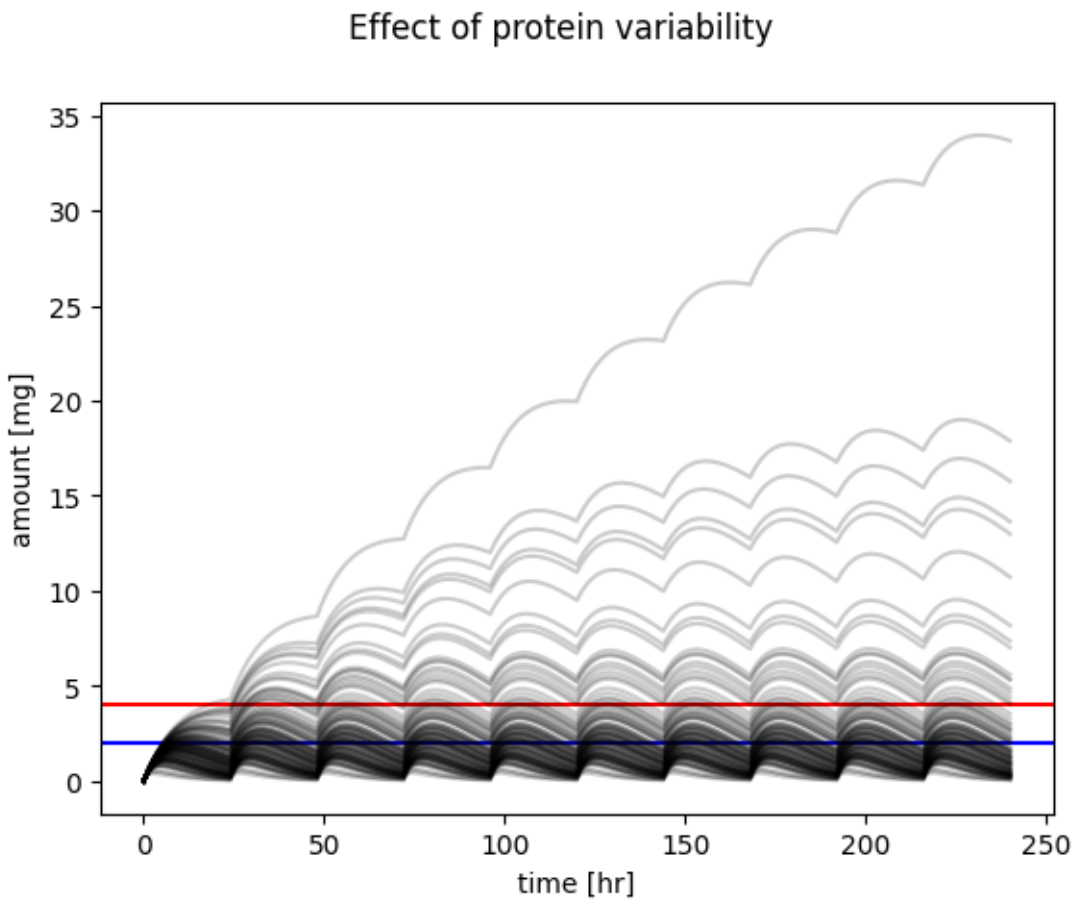
Achieving the therapeutic range is crucial for ensuring that a drug is both effective and safe. The therapeutic range refers to the concentration window within which a drug produces its desired effect without causing significant adverse effects. Various factors, including pharmacokinetic variability, influence a drug's ability to stay within this range. Understanding and managing these factors are essential for optimizing drug dosing and maximizing therapeutic efficacy.

```
def simulate_multi_dosing(Dose_A, ka, ke):
    """Helper function to run the multiple dosing simulation."""

    # initial condition
    names = ["A_tablet", "A_central", "A_urine"]
    x0 = [
        Dose_A,  # A_tablet  [mg]
        0.0,   # A_central [mg]
        0.0,   # A_urine [mg]
    ]

    # time span for single dose
    t = np.linspace(0, 24, num=100) # [hr]

    # multiple dose simulation
    n_doses = 10  # [hr]

    dfs = []
    for k in range(n_doses):
```

```
        if k == 0:
            # x0[0] = 400   # first dose
            tvec = t.copy()
        elif k > 0:
            x0 = x[-1, :]
            x0[0] = x0[0] + Dose_A
            tvec = t.copy() + tvec[-1]

        x = odeint(dxdt_absorption_first_order, x0, tvec, args=(ka, ke))
        df = pd.DataFrame(x, columns=names)
        df["time"] = tvec
        dfs.append(df)

    df_all = pd.concat(dfs)
    return df_all
```

```
# run simulation and plot results
Dose_A = 5.0  # [mg]
ka = 0.1  # [1/hr]
ke = 0.1  # [1/hr]


# simulate multi dosing
tcs = []
pks = []
for f_protein in f_proteins:
    # individual elimination rate
    ke_protein = f_protein * ke
    df = simulate_multi_dosing(Dose_A, ka, ke_protein)
    tcs.append(df)

# plot timecourse
f, ax = plt.subplots(nrows=1, ncols=1)
f.suptitle("Effect of protein variability")
ax.set_xlabel("time [hr]")
ax.set_ylabel("amount [mg]")
ax.axhline(y=4, color='r', linestyle='-', label="MTC")
ax.axhline(y=2, color='b', linestyle='-', label="MEC")

for k, f_protein in enumerate(f_proteins):
    tc = tcs[k]
    ax.plot(tc.time, tc.A_central, color="black", alpha=0.2)
```

```
plt.show()
```



24.3.1 Exercise: Dosing Optimization for the Population

Objective: Perform a dosing optimization for the population by adjusting the absorption rate constant k_a, the elimination rate constant k_e, and the administered dose $Dose_A$ to achieve optimal therapeutic outcomes.

24.3.2 Insights

- **Optimization Process**: Understand the process of optimizing pharmacokinetic parameters to achieve therapeutic goals.
- **Parameter Interdependence**: Learn how changes in one parameter affect the overall pharmacokinetic profile and how to balance multiple parameters for optimal therapy.

- **Therapeutic Outcomes**: Ensure drug concentrations remain within the therapeutic range for the majority of the population, maximizing efficacy and minimizing adverse effects.

This exercise helps you gain practical experience in pharmacokinetic modeling and dosing optimization, crucial for personalized medicine and effective drug therapy. It helps to understand the difference between optimizing the theraphy for an individual versus the population.

24.4 Stratification

In pharmacokinetic modeling, it is important to recognize that distributions of pharmacokinetic parameters can vary across different subsets of the population. This variability can be leveraged through stratification, which involves applying separate models to distinct subgroups within the population.

24.4.1 Gender-Based Stratification

For example, we can study the differences in pharmacokinetics between males and females by assuming that protein distributions differ between these groups. This approach allows us to more accurately predict drug behavior and optimize dosing for each subgroup.

24.4.2 Key Points:

- **Recognizing Differences**: Understand that pharmacokinetic parameters such as protein expression levels, enzyme activity, and drug transporter amounts can vary significantly between different demographic groups.
- **Subgroup Analysis**: By stratifying the population into subgroups (e.g., males and females), we can develop more precise pharmacokinetic models that account for these differences.
- **Improved Accuracy**: Stratified models can lead to better predictions of drug concentrations, efficacy, and safety profiles for each subgroup, enhancing personalized medicine.

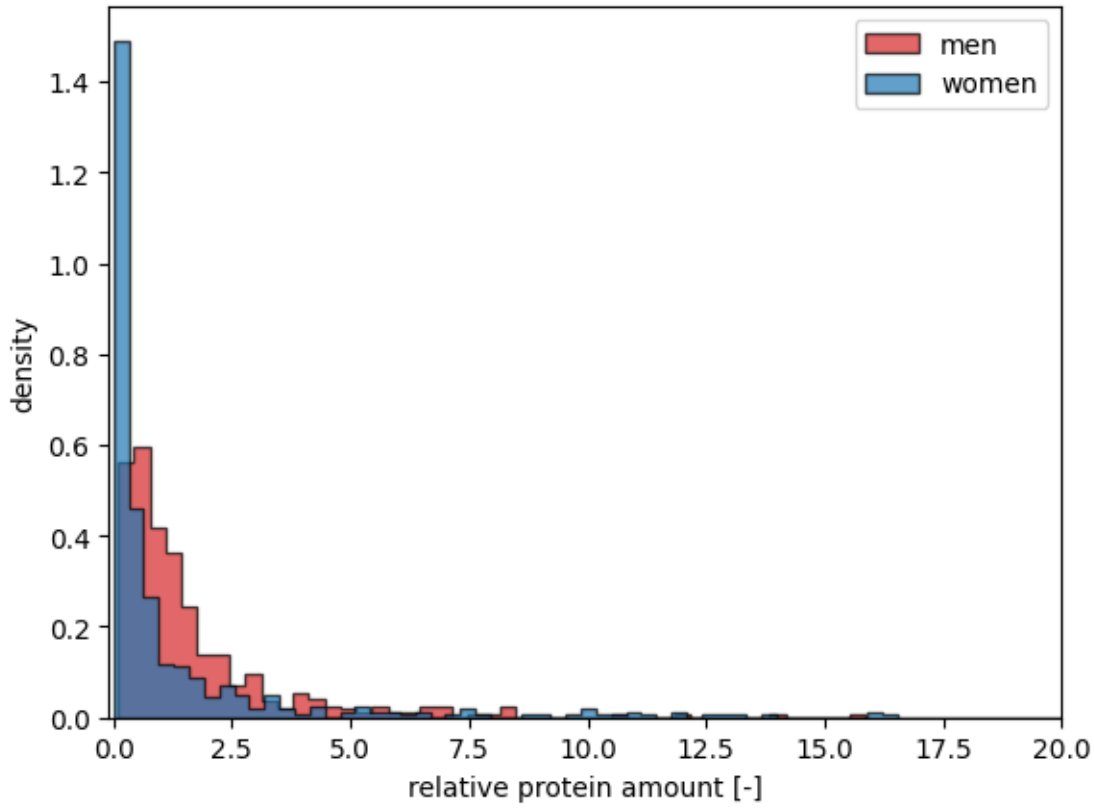
24.4.3 Example Application:

- **Gender Differences**: Protein distributions related to drug metabolism and transport can differ between males and females due to physiological and hormonal variations. By stratifying the population into male and female subgroups, we can tailor pharmacokinetic models to reflect these differences.

- **Implications for Dosing**: Stratification allows for more accurate dose adjustments based on subgroup-specific characteristics, reducing the risk of under- or over-dosing and improving therapeutic outcomes.

By incorporating stratification into pharmacokinetic modeling, we can enhance our understanding of drug behavior across different population subsets and optimize treatment strategies accordingly.

```
import numpy as np
from matplotlib import pyplot as plt
from scipy.stats import lognorm

fig, ax = plt.subplots(1, 1)
s_men = 1
scale_men=1
scale_women=0.4
s_women = 2

n_samples = 500
f_proteins_men = lognorm.rvs(s_men, scale=scale_men, size=n_samples)
f_proteins_women = lognorm.rvs(s_women, scale=scale_women, size=n_samples)

ax.hist(f_proteins_men, density=True, bins='auto', histtype='stepfilled', alpha=0.7,
        edgecolor="black", color="tab:red", label="men")
ax.hist(f_proteins_women, density=True, bins='auto', histtype='stepfilled', alpha=0.7,
        edgecolor="black", color="tab:blue", label="women")

ax.legend(loc='best')
ax.set_xlabel("relative protein amount [-]")
ax.set_ylabel("density")
ax.set_xlim(left=-0.1, right=20)
plt.show()
# print(f_proteins)
```

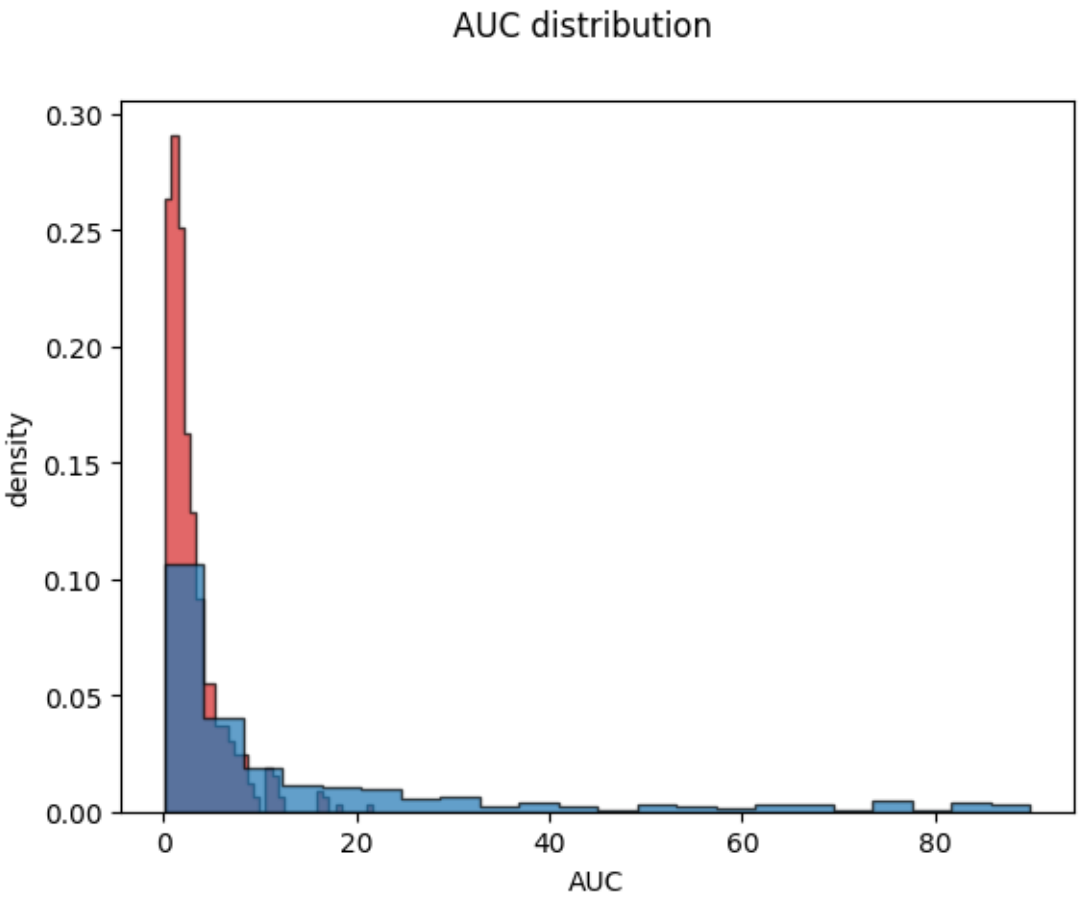



```
import numpy as np
import pandas as pd
from scipy.integrate import odeint
from matplotlib import pylab as plt

from helpers import f_pk, dxdt_absorption_first_order

# initial condition and time span
t = np.arange(0, 10, 0.05) # [hr]
Dose_A = 10.0  # [mg]
x0 = [
    Dose_A,  # A_tablet  [mg]
    0.0,   # A_central [mg]
    0.0,   # A_urine [mg]
]

# parameters
ka = 2.0  # [1/hr]
```

```
ke = 5.0  # [1/hr]

# simulate all genetic variants
tcs_dict = {}
pks_dict = {}

for key in ["men", "women"]:
    tcs = []
    pks = []

    if key == "men":
        f_proteins = f_proteins_men
    elif key == "women":
        f_proteins = f_proteins_women

    for f_protein in f_proteins:

        # the genetic variants effect the elimination/transport rate
        ke_protein = f_protein * ke
        x = odeint(dxdt_absorption_first_order, x0, t, args=(ka, ke_protein))
        df = pd.DataFrame(x, columns=["A_tablet", "A_central", "A_urine"])
        df["time"] = t
        tcs.append(df)

        # calculate pharmacokinetics parameters on the curves
        pk = f_pk(t=df.time.values, c=df.A_central.values, dose=Dose_A, show=False)
        pks.append(pk)
    tcs_dict[key] = tcs
    pks_dict[key] = pks

# plot AUC
f, ax = plt.subplots(nrows=1, ncols=1)
f.suptitle("AUC distribution")
for key in ["men", "women"]:
    pks = pks_dict[key]

    if key == "men":
        color = "tab:red"
    elif key == "women":
        color = "tab:blue"

    aucs = [pk["auc"] for pk in pks]
```

```python
    ax.hist(aucs, density=True, bins='auto', histtype='stepfilled', alpha=0.7,
            edgecolor="black", color=color)

ax.set_xlabel("AUC")
ax.set_ylabel("density")
plt.show()
```

```
/home/mkoenig/git/course-pharmacokinetic-modelling/src/mb19/notebooks/helpers.py:33: RuntimeW
  y = np.log(c[max_index + 1:])
/home/mkoenig/git/course-pharmacokinetic-modelling/src/mb19/notebooks/helpers.py:33: RuntimeW
  y = np.log(c[max_index + 1:])
/home/mkoenig/git/course-pharmacokinetic-modelling/src/mb19/notebooks/helpers.py:33: RuntimeW
  y = np.log(c[max_index + 1:])
/home/mkoenig/git/course-pharmacokinetic-modelling/src/mb19/notebooks/helpers.py:33: RuntimeW
  y = np.log(c[max_index + 1:])
/home/mkoenig/git/course-pharmacokinetic-modelling/src/mb19/notebooks/helpers.py:33: RuntimeW
  y = np.log(c[max_index + 1:])
/home/mkoenig/git/course-pharmacokinetic-modelling/src/mb19/notebooks/helpers.py:33: RuntimeW
  y = np.log(c[max_index + 1:])
/home/mkoenig/git/course-pharmacokinetic-modelling/src/mb19/notebooks/helpers.py:33: RuntimeW
  y = np.log(c[max_index + 1:])
/home/mkoenig/git/course-pharmacokinetic-modelling/src/mb19/notebooks/helpers.py:33: RuntimeW
  y = np.log(c[max_index + 1:])
```



24.5 Pharmacogenetic Variants

Pharmacogenomics is the study of how an individual's genetic makeup influences their response to drugs. This field combines pharmacology (the science of drugs) and genomics (the study of genes and their functions) to develop effective, safe medications and doses that will be tailored to a person's genetic makeup.

Pharmacogenomics is a part of the broader field of personalized medicine, which aims to tailor medical treatment to the individual characteristics of each patient.

With respect to pharmacokinetics (the study of how the body absorbs, distributes, metabolizes, and excretes drugs), pharmacogenomics can provide valuable insights. For instance:

1. **Absorption and Distribution:** Genetic differences can affect the expression and function of proteins involved in drug transport across cell membranes, influencing how quickly and effectively a drug is absorbed or distributed within the body.

2. **Metabolism:** A key aspect of pharmacokinetics is understanding how drugs are metabolized, primarily by enzymes in the liver. Individual genetic variations can affect the activity of these enzymes, leading to differences in how quickly a drug is metabolized. For example, some individuals may have genetic variations that cause certain enzymes to be overly active ("ultra-metabolizers") or underactive ("poor metabolizers"). This can significantly impact the concentration of drug in the body and therefore its efficacy and potential for side effects.
3. **Excretion:** Variations in genes can also impact the function of proteins involved in the excretion of drugs, primarily in the kidneys, affecting the rate at which a drug is removed from the body.

By understanding an individual's pharmacogenomic profile, healthcare providers can better predict how a patient will respond to a particular drug, informing decisions about which drug to prescribe and at what dose. This can improve drug efficacy, reduce the risk of adverse effects, and contribute to more efficient and safer healthcare. However, it's important to note that while pharmacogenomics holds great promise, its application in routine clinical practice is currently limited, though it's an area of active research and development.

The following defines the activity of the allels of the SLCO1B1 transporter. We focus on the subset of important genetic variants `*1a`, `*1b`, `*15` and possible combinations

```
# SLCO1B1
# f_sclo1b1
f_wildtype = 1.0  # *1a wildtype activity
f_increase = 1.1  # *1b has slightly increased activity
f_decrease = 0.5  # all variants with the 521T/C mutant have strongly reduced activity, e.g.

slco1b1_activity = {
    # strong increase (dark blue)
    "*1b/*1b": (f_increase + f_increase) / 2,

    # mild increase (light blue)
    "*1a/*1b": (f_wildtype + f_increase) / 2,

    # wildtype (black)
    "*1a/*1a": (f_wildtype + f_wildtype)/2,

    # minor decrease (orange)
    "*1b/*15": (f_increase + f_decrease) / 2,

    # mild decrease (red)
    "*1a/*15": (f_wildtype + f_decrease) / 2,
```

```
    # strong decrease (dark red)
    "*15/*15": (f_decrease + f_decrease) / 2,
}
slco1b1_color = {
        "*1b/*1b": "blue",
        "*1a/*1b": "tab:blue",
        "*1a/*1a": "black",
        "*1b/*15": "tab:orange",
        "*1a/*15": "tab:red",
        "*15/*15": "red",
}
```

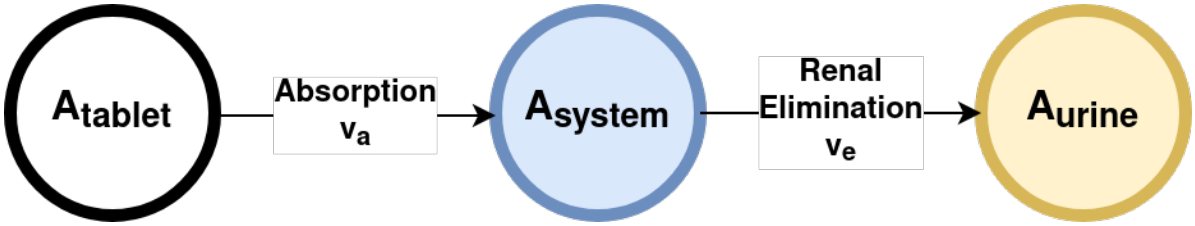
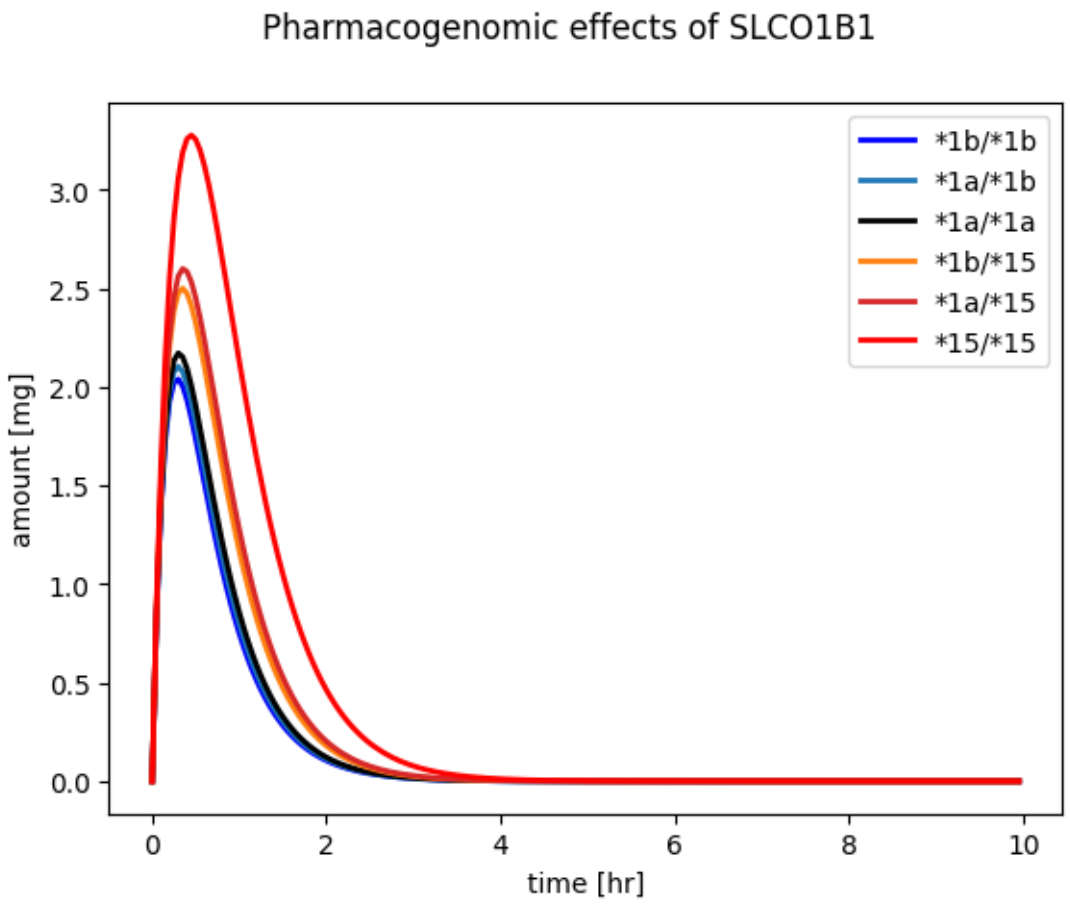


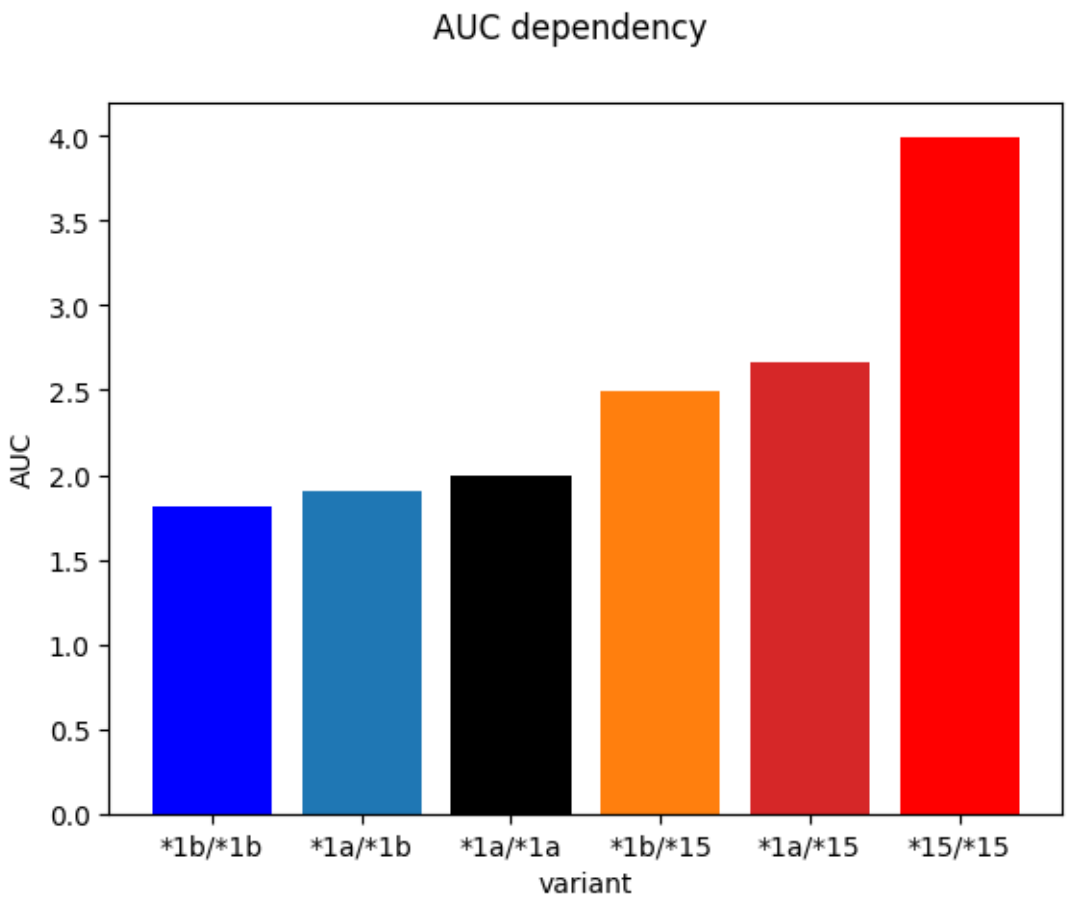
Figure 24.4: Absorption Elimination Model

```
import numpy as np
import pandas as pd
from scipy.integrate import odeint
from matplotlib import pylab as plt

from helpers import f_pk, dxdt_absorption_first_order

# initial condition and time span
t = np.arange(0, 10, 0.05) # [hr]
Dose_A = 10.0  # [mg]
x0 = [
    Dose_A,  # A_tablet  [mg]
    0.0,   # A_central [mg]
    0.0,   # A_urine [mg]
]

# parameters
ka = 2.0  # [1/hr]
ke = 5.0  # [1/hr]

# simulate all genetic variants
```

```
tcs = []
pks = []

for variant, f_activity in slco1b1_activity.items():

    # the genetic variants effect the elimination/transport rate
    ke_variant = f_activity * ke
    x = odeint(dxdt_absorption_first_order, x0, t, args=(ka, ke_variant))
    df = pd.DataFrame(x, columns=["A_tablet", "A_central", "A_urine"])
    df["time"] = t
    tcs.append(df)

    # calculate pharmacokinetics parameters on the curves
    pk = f_pk(t=df.time.values, c=df.A_central.values, dose=Dose_A, show=False)
    pks.append(pk)


# plot timecourse
f, ax = plt.subplots(nrows=1, ncols=1)
f.suptitle("Pharmacogenomic effects of SLCO1B1")
ax.set_xlabel("time [hr]")
ax.set_ylabel("amount [mg]")

for k, variant in enumerate(slco1b1_activity):
    color = slco1b1_color[variant]
    tc = tcs[k]
    ax.plot(tc.time, tc.A_central, linewidth=2, color=color, label=variant)

ax.legend()
plt.show()

# plot AUC
f, ax = plt.subplots(nrows=1, ncols=1)
f.suptitle("AUC dependency")

aucs = [pk["auc"] for pk in pks]
variants = list(slco1b1_activity.keys())
colors = [slco1b1_color[v] for v in variants]

ax.bar(
    x=range(len(aucs)), height=aucs,
    color=colors,
```

```
    tick_label=variants
)
ax.set_xlabel("variant")
ax.set_ylabel("AUC")


plt.show()
```





Exercise 1: Study the effect of the genetic variants on other pharmacokinetic parameters such as CL, thalf, …

Exercise 2: Add an additional newly discovered variant `*f` to the analysis which has 2.0 activity compared to the wildtype variant.

25 SBML Models

The Systems Biology Markup Language (SBML) is a computer-readable format designed for representing models of biological systems. It is an XML-based standard that encodes computational models in systems biology, a field focused on understanding the interactions and behaviors of biological components within a system.

25.1 Key Features and Benefits

- **Standardization**: SBML provides a standardized format for encoding biological models, ensuring consistency and interoperability across different software tools.
- **Model Exchange**: Facilitates the exchange of models between different research groups and software applications, allowing researchers to utilize the best tools available for various aspects of their work.
- **Collaboration and Reproducibility**: Enhances collaboration by enabling researchers to share models easily. It also ensures that models can be stored, archived, and reused, promoting reproducibility in scientific research.

25.2 Components of SBML

SBML models describe biochemical entities (such as species), the reactions between these entities, and the mathematical rules that govern the system. It can represent a wide range of biological processes, including metabolism, cell signaling, and gene regulation.

25.3 Development and Support

The development of SBML is coordinated by the SBML Project, a community-driven effort that includes researchers, software developers, and other stakeholders in systems biology. The project provides libraries, software tools, and educational resources for working with SBML.

25.4 Working with SBML

To work with SBML models, additional packages such as `sbmlutils` and `libroadrunner` are required. These packages facilitate the manipulation and simulation of SBML models. You can install them via pip:

pip install sbmlutils pip install libroadrunner

By using these tools, researchers can effectively create, modify, and analyze SBML models, enhancing their understanding of complex biological systems and improving the quality of their computational studies.

25.5 Encode SBML model

In a first step we encode our simple model with sbmlutils as SBML.

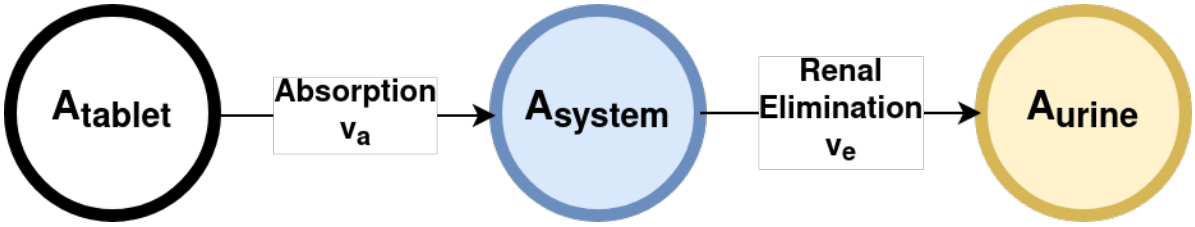


Figure 25.1: Absorption Elimination Model

```
from sbmlutils.factory import *
from sbmlutils.metadata import *
from pathlib import Path
import warnings
warnings.filterwarnings("ignore")

m = Model(
    sid="absorption_first_order",
    name="Absorption model with first order kinetics",

)
m.compartments = [
    Compartment("body", 1.0, name="Body", sboTerm=SBO.SIMPLE_CHEMICAL)
]
m.parameters = [
    Parameter("Dose_A", 10.0, name="Dose of A"),
    Parameter("ka", 1.0, name="Absorption rate"),
```

```
    Parameter("ke", 1.0, name="Elimination rate"),
]
m.species = [
    Species("A_tablet", initialAmount=0.0, hasOnlySubstanceUnits=True,
            compartment="body", name="A (tablet)", sboTerm=SBO.SIMPLE_CHEMICAL),
    Species("A_central", initialAmount=0.0, hasOnlySubstanceUnits=True,
           compartment="body", name="A (body)", sboTerm=SBO.SIMPLE_CHEMICAL),
    Species("A_urine", initialAmount=0.0, hasOnlySubstanceUnits=True,
           compartment="body", name="A (urine)", sboTerm=SBO.SIMPLE_CHEMICAL),
]
m.assignments = [
    InitialAssignment("A_tablet", "Dose_A")
]
m.reactions = [
    Reaction(
        "ABSORPTION",
        name="absorption A",
        equation="A_tablet -> A_central",
        formula = "ka * A_tablet"
    ),
    Reaction(
        "ELIMINATION",
        name="elimination A",
        equation="A_central -> A_urine",
        formula = "ke * A_central"
    )
]

# save the model (uncomment the following lines to create the model)
# results = create_model(
#     model=m,
#     filepath=Path("absorption_first_order.xml"),
#     validation_options=ValidationOptions(units_consistency=False)
# )
```

Exercise: Explore the model with `https://sbml4humans.de`. Upload the model to the website and navigate the objects.

Exercise: Visualize the model with cytoscape and `cy3sbml`. I.e. download Cytoscape, install the app `cy3sbml` and load the model to explore the model.

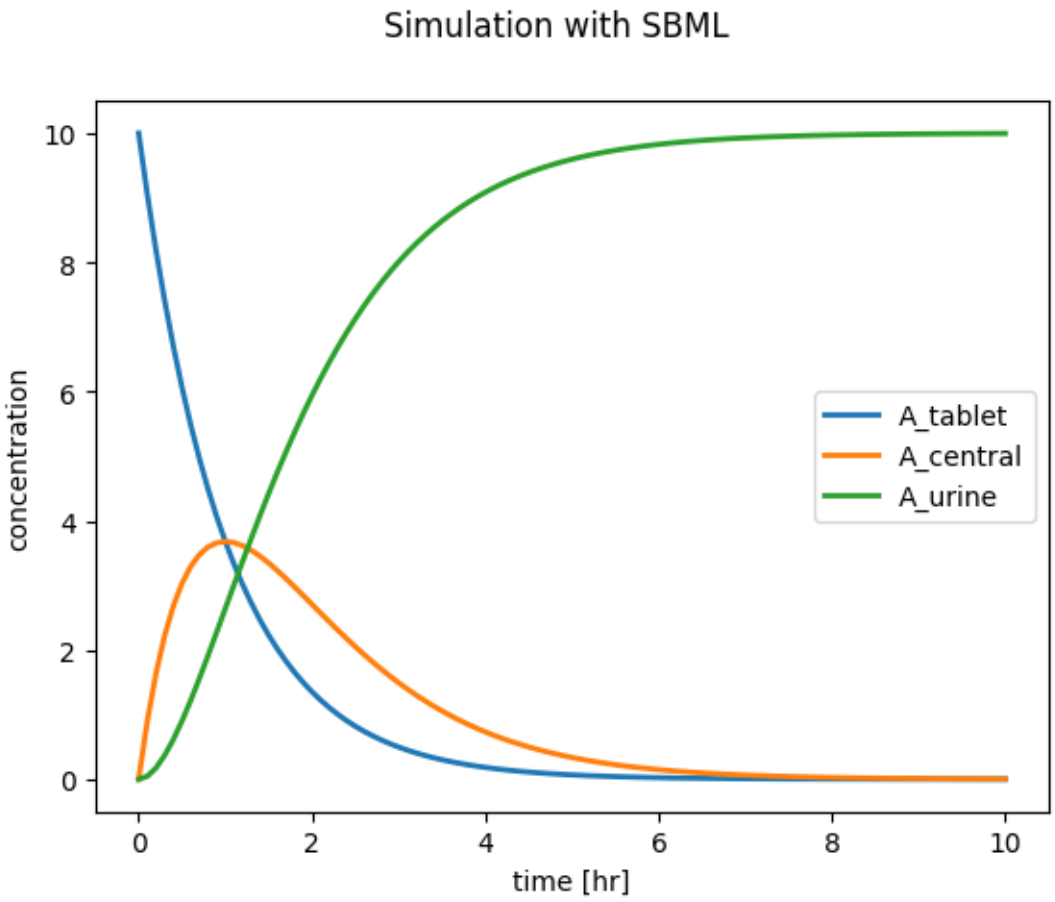
25.6 Simulate SBML model

In a second step we load the model and perform a simple simulation and visualization.

```
import roadrunner
import pandas as pd
r = roadrunner.RoadRunner("absorption_first_order.xml")
s = r.simulate(start=0, end=10, steps=100)
df = pd.DataFrame(s, columns=s.colnames)
print(df)
```

```
     time  [A_tablet]  [A_central]  [A_urine]
0     0.0   10.000000     0.000000   0.000000
1     0.1    9.048374     0.904837   0.046788
2     0.2    8.187307     1.637462   0.175231
3     0.3    7.408182     2.222455   0.369363
4     0.4    6.703201     2.681278   0.615521
..    ...         ...          ...        ...
96    9.6    0.000677     0.006502   9.992821
97    9.7    0.000613     0.005945   9.993443
98    9.8    0.000555     0.005434   9.994011
99    9.9    0.000502     0.004967   9.994531
100  10.0    0.000454     0.004540   9.995006

[101 rows x 4 columns]
```

```
from matplotlib import pyplot as plt
f, ax = plt.subplots(nrows=1, ncols=1)
f.suptitle("Simulation with SBML")

for name in ["A_tablet", "A_central", "A_urine"]:
    ax.plot(df.time, df[f"[{name}]"], lw=2, label=name)

ax.set_ylabel("concentration")
ax.set_xlabel("time [hr]")
ax.legend()
plt.show()
```


Simulation with SBML
concentration
time [hr]
A_tablet
A_central
A_urine
0
2
4
6
8
10

26 Pharmacodynamics

Pharmacodynamics is the study of how a drug affects the body. It involves understanding the biochemical, physiological, and molecular effects of drugs, as well as the body's response to them. Pharmacodynamics focuses on receptor binding, post-receptor effects, and chemical interactions. This field helps to explain how drugs work, their mechanisms of action, their therapeutic effects, and any adverse effects they might cause.

In pharmacodynamics, key concepts include:

1. **Dose-response relationships**: This describes how a body's response changes based on the dose of the drug. The relationship is typically plotted on a graph to understand the point at which the drug's effects plateau, and increasing the dose would not necessarily increase the response but could increase the risk of adverse effects.
2. **Therapeutic window**: This is the dosage range in which a drug is expected to be effective without causing unacceptable side effects. It's a balance between efficacy and toxicity.
3. **Receptors**: These are typically proteins in the body to which a drug binds to exert its effects. Drug-receptor interaction can stimulate (agonists) or block (antagonists) a physiological response.
4. **Pharmacological effect**: This refers to the specific biochemical interaction through which a drug substance produces its pharmacological effect.
5. **Drug potency and efficacy**: Potency refers to the amount of drug needed to produce an effect, while efficacy refers to the maximum effect a drug can produce.

Pharmacodynamics, along with pharmacokinetics (how the body processes the drug), forms the basis for the development, therapeutic use, and understanding of drugs.

26.1 Dose response relationship

Pharmacodynamics describes what a drug does to the body, and the dose-response relationship is a key aspect of this. The dose-response relationship, in basic terms, describes how the effect of a drug changes as its dose changes.

Here are the key points related to the dose-response relationship:

1. **Graded Dose-Response**: In this type of relationship, as the dose of a drug increases, the individual subject's response to the drug also increases until a maximum effect (Emax) is reached. At this point, increasing the dose further will not enhance the effect. This curve is often used to understand the drug's potency (amount of drug needed to produce a particular effect) and efficacy (maximum effect a drug can produce).
2. **Quantal Dose-Response**: This type of relationship looks at the percentage of a population that responds to a certain dose of a drug rather than the intensity of an individual's response. It's typically used to determine the median effective dose (ED50, the dose that produces the desired effect in 50% of the population), median toxic dose (TD50, the dose that is toxic in 50% of the population), and median lethal dose (LD50, the dose that is lethal in 50% of the population). These values are used in determining a drug's therapeutic index or safety margin.
3. **Threshold Dose**: This is the minimum dose of a drug that produces a measurable response. Below this dose, no significant effect is observed.
4. **Plateau or Saturation Point**: After a certain dose, increasing the quantity of the drug does not increase the therapeutic effect. This level is known as the plateau or saturation point, and it is here that the drug reaches its maximum efficacy.
5. **Drug Receptors and Binding**: The dose-response relationship also hinges on the interaction between the drug and its receptor. Drugs bind to receptors to produce their effects, and the binding is often reversible. The degree to which a drug activates a response when it binds to a receptor is called intrinsic activity.
6. **Agonists and Antagonists**: Drugs that bind to receptors and stimulate a response are called agonists. Those that prevent or dampen the receptor's response are called antagonists. Understanding how agonists and antagonists work helps in understanding the dose-response relationship.
7. **Inter-individual Variability**: There can be significant variability in dose-response relationships between different individuals due to factors such as genetics, age, gender, disease state, and the presence of other drugs.

These points highlight how the dose of the drug and the body's response to the drug are interconnected, and understanding this relationship is crucial in therapeutic drug use and development.

First we will study the dose response relationship based on the following equation

$$E = E_{max} \cdot \frac{C^{\gamma}}{E_{50}^{\gamma} + C^{\gamma}} \tag{26.1}$$

with E_{max} being the maximum effect or efficacy, C being the concentration at the site of action, and E_{50} being the dose required to produce 50% of the maximum effect, and γ being the steepness factor of the sigmoidal curve.

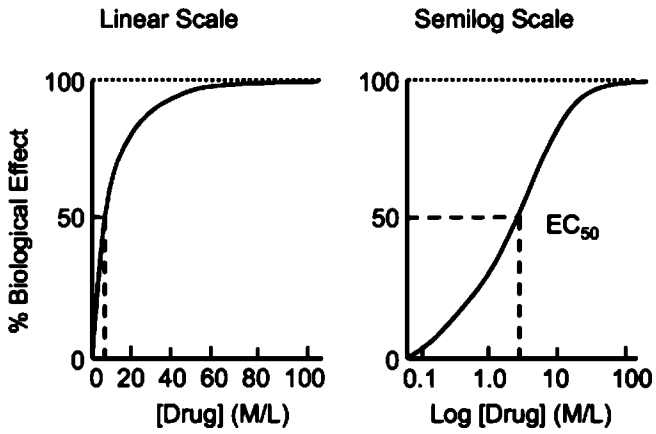
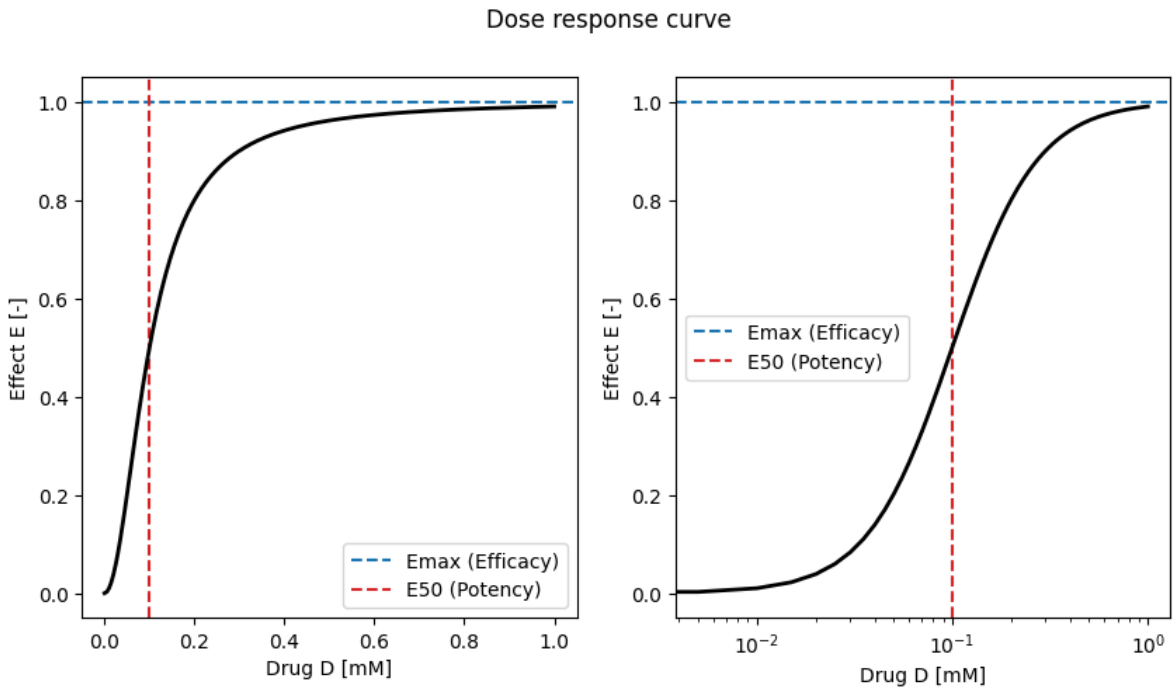


Figure 26.1: Dose response

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

E50 = 0.1  # [mM]
Emax = 1.0  # [-] arbitrary effect units
gamma = 2
D = np.linspace(0, 1, num=200)  # [mM]
E = Emax * D**gamma/(E50**gamma + D**gamma)  # [-]

f, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(10, 5))
f.suptitle("Dose response curve")
for ax in (ax1, ax2):
    ax.axhline(y=Emax, color="tab:blue", linestyle="--", label="Emax (Efficacy)")
    ax.axvline(x=E50, color="tab:red", linestyle="--", label="E50 (Potency)")
    ax.plot(D, E, color="black", linewidth=2.0)
    ax.set_xlabel("Drug D [mM]")
    ax.set_ylabel("Effect E [-]")
    ax.legend()

ax2.set_xscale("log")
plt.show()
```

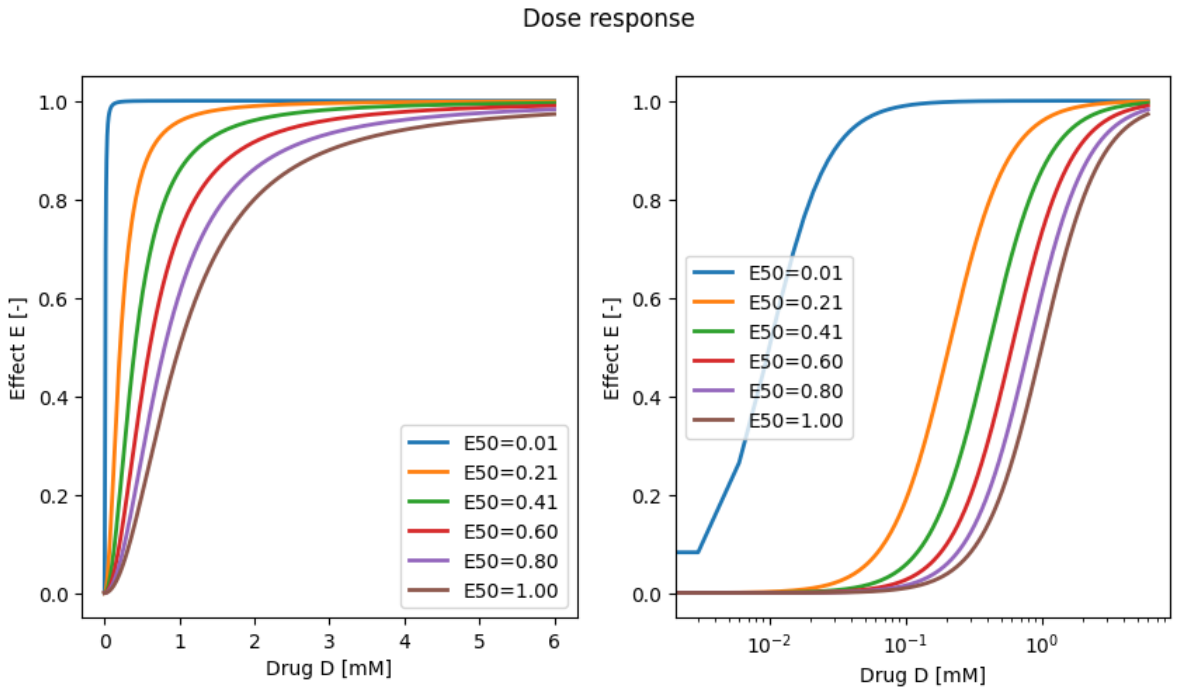


26.2 Potency

Now we perform a parameter scan to study the effect of potency (E50) on the drug dose-effect curve.

```
E50 = 0.1  # [mM]
Emax = 1.0  # [-] arbitrary effect units
gamma = 2
D = np.linspace(0, 6, num=2000)  # [mM]
E50s = np.linspace(0.01, 1.0, num=6)

f, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(10, 5))
f.suptitle("Dose response")
for ax in (ax1, ax2):
    for E50 in E50s:
        E = Emax * D**gamma/(E50**gamma + D**gamma)  # [-]
        ax.plot(D, E, linewidth=2.0, label=f"{E50=:.2f}")

    ax.set_xlabel("Drug D [mM]")
    ax.set_ylabel("Effect E [-]")
```

```
    ax.legend()

ax2.set_xscale("log")
plt.show()
```



Exercise: Analyse the dependency of the Effect `E` on the efficacy `Emax` and steepness `gamma`. I.e. perform a parameter scan similar to the potency `E50`.

26.3 Pharmacokinetics/pharmacodynamics (PK/PD)

Next we are interested in combining a pharmacokinetics model (concentration timecourse) with a pharmacodynamic model of the drug to be able to describe the effect of the drug.

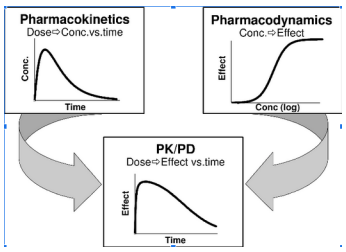


Figure 26.2: Pharmacokinetics and pharmacodynamics

We will couple the simple compartment model of absorption and elimination to a model pharmacodynamic model describing the desired effect `E` and the side effect `S`.

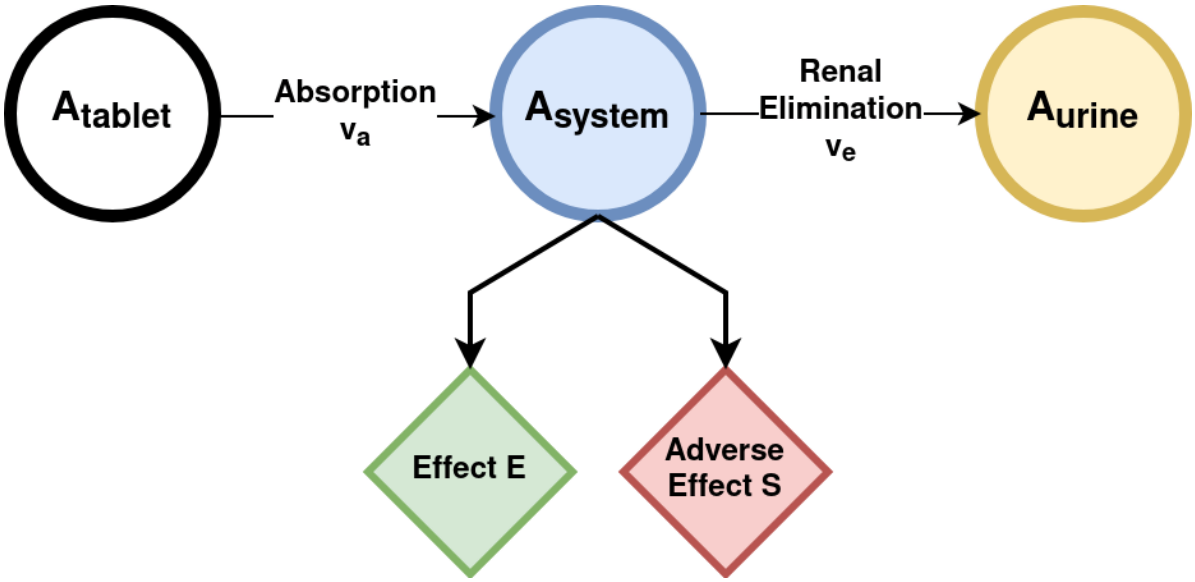
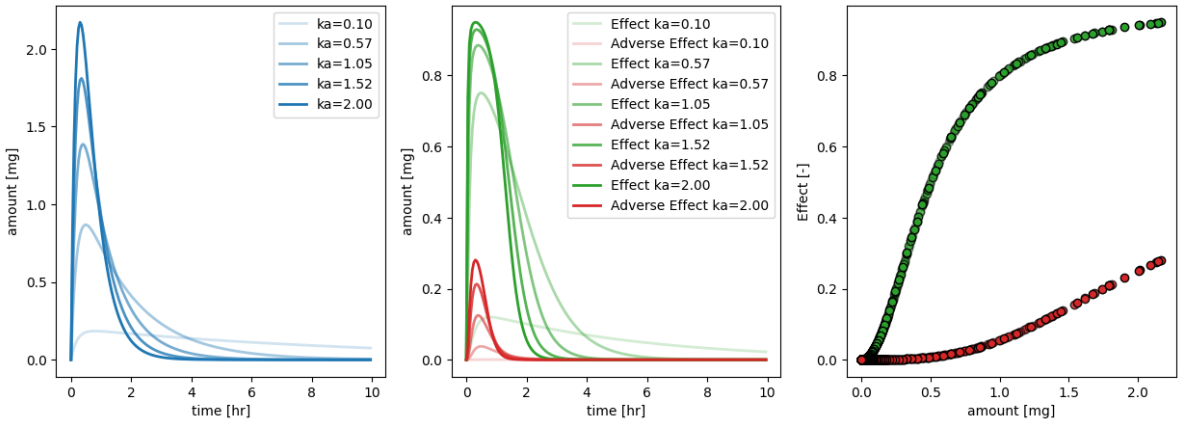


Figure 26.3: PK/PD model

```
import numpy as np
import pandas as pd
from scipy.integrate import odeint
from matplotlib import pylab as plt

from helpers import dxdt_absorption_first_order

# initial condition and time span
t = np.arange(0, 10, 0.05) # [hr]
Dose_A = 10.0  # [mg]
x0 = [
```

```
    Dose_A,  # A_tablet  [mg]
    0.0,   # A_central [mg]
    0.0,   # A_urine [mg]
]

# parameters of pharmacokinetics model
ka = 2.0  # [1/hr]
ke = 5.0  # [1/hr]

# parameters of pharmacodynamics model
# effect
E50 = 0.5  # [mg]
Emax = 1.0  # [-] arbitrary effect units
Egamma = 2

# adverse effect paramters
S50 = 2.0  # [mg]
Smax = 0.5  # [-] arbitrary effect units
Sgamma = 3

x = odeint(dxdt_absorption_first_order, x0, t, args=(ka, ke))

n_samples = 5
kas = np.linspace(0.1, 2.0, num=n_samples)  # [1/hr]
tcs = []
pks = []

# simulate all the different absorption
for kp, ka in enumerate(kas):
    x = odeint(dxdt_absorption_first_order, x0, t, args=(ka, ke))
    df = pd.DataFrame(x, columns=["A_tablet", "A_central", "A_urine"])
    df["time"] = t
    # calculate effect
    df["E"] = Emax * df.A_central**Egamma/(E50**Egamma + df.A_central**Egamma)
    # calculate side effect
    df["S"] = Smax * df.A_central**Sgamma/(S50**Sgamma + df.A_central**Sgamma)
    tcs.append(df)

# plot timecourse
f, (ax1, ax2, ax3) = plt.subplots(nrows=1, ncols=3, figsize=(15,5))

for k, ka in enumerate(kas):
```

```
    tc = tcs[k]
    # pharmacokinetics
    ax1.plot(
        tc.time, tc.A_central, linewidth=2, color="tab:blue",
        alpha=(k+1)/n_samples, label=f"{ka=:.2f}"
    )
    # desired effect
    ax2.plot(
        tc.time, tc.E, linewidth=2, color="tab:green",
        alpha=(k+1)/n_samples, label=f"Effect {ka=:.2f}"
    )
    ax3.plot(
        tc.A_central, tc.E, linewidth=2, color="tab:green", linestyle="", marker="o", markere
        alpha=(k+1)/n_samples, label=f"Effect {ka=:.2f}"
    )
    # adverse effect
    ax2.plot(
        tc.time, tc.S, linewidth=2, color="tab:red",
        alpha=(k+1)/n_samples, label=f"Adverse Effect {ka=:.2f}"
    )
    ax3.plot(
        tc.A_central, tc.S, linewidth=2, color="tab:red", linestyle="", marker="o", markeredg
        alpha=(k+1)/n_samples, label=f"Adverse Effect {ka=:.2f}"
    )

for ax in (ax1, ax2):
    ax.set_xlabel("time [hr]")
    ax.set_ylabel("amount [mg]")
    ax.legend()

ax3.set_xlabel("amount [mg]")
ax3.set_ylabel("Effect [-]")

plt.show()
```


ka=0.10
ka=0.57
ka=1.05
ka=1.52
ka=2.00
amount [mg]
time [hr]
Effect ka=0.10
Adverse Effect ka=0.10
Effect ka=0.57
Adverse Effect ka=0.57
Effect ka=1.05
Adverse Effect ka=1.05
Effect ka=1.52
Adverse Effect ka=1.52
Effect ka=2.00
Adverse Effect ka=2.00
amount [mg]
time [hr]
Effect [-]
amount [mg]

27 Physiologically based pharmacokinetic (PBPK)

Physiologically based pharmacokinetic (PBPK) models are mathematical models that can predict how a drug is absorbed, distributed, metabolized, and excreted by the body. PBPK models are built to be physiologically meaningful and incorporate information about the physiology of the body and the physical chemistry of the drug.

These models are structured to represent the body as a series of interconnected compartments that correspond to actual body organs and tissues (e.g., liver, kidneys, heart, blood, etc.). Each compartment can have unique characteristics such as volume, blood flow rate, and specific enzyme or transporter levels.

The inputs for a PBPK model include parameters like the physicochemical properties of the drug (e.g., partition coefficients, molecular weight, pKa), its dose and route of administration, and specific information about the physiology of the individual (e.g., body weight, age, sex, organ function, enzyme activity).

These models are quite complex, but they are valuable because they can be used to simulate and predict the behavior of drugs in the body under various conditions. This allows researchers to anticipate potential issues with drug-drug interactions, pharmacokinetics in different populations (e.g., children, the elderly, or patients with specific organ impairments), or under various dosing scenarios.

In other words, PBPK models serve as a tool that helps scientists understand how a drug will behave in the body before it's administered, thereby helping to optimize dosing regimens, predict potential side effects, and generally improve the safety and effectiveness of drug therapy.

Within this tutorial we will work with a simple PBPK model to study the pharmacokinetics of caffeine. Pharmacokinetics refers to how a drug is absorbed, distributed, metabolized, and excreted by the body. Here are the key points about the pharmacokinetics of caffeine:

1. **Absorption**: Caffeine is rapidly and almost completely absorbed from the gastrointestinal tract after oral ingestion, with peak plasma concentrations typically reached within 30 minutes to 2 hours.

2. **Distribution**: Caffeine is distributed throughout all body tissues and fluids, including the brain, and it crosses the placenta. It is highly water and lipid soluble, enabling its wide distribution throughout the body.

3. **Metabolism**: Caffeine is primarily metabolized in the liver by the cytochrome P450 oxidase enzyme system, specifically CYP1A2, into three dimethylxanthines, each having its own effects: paraxanthine (increases lipolysis), theobromine (dilates blood vessels and increases urine volume), and theophylline (relaxes smooth muscles of the bronchi, coronary arteries, and is responsible for caffeine's diuretic effect). The rate of caffeine metabolism can be affected by various factors such as genetic factors, liver function, and concurrent medications or substances that induce or inhibit CYP1A2.

4. **Excretion**: The metabolites of caffeine are excreted in the urine. Less than 1% of consumed caffeine is excreted unchanged in the urine.

5. **Half-life**: The half-life of caffeine in the human body is typically in the range of 3 to 5 hours. This can be influenced by factors such as age, liver function, pregnancy, and the level of enzymes in the liver. For example, in newborns, caffeine has a much longer half-life, and during pregnancy, particularly in the third trimester, caffeine clearance is significantly decreased.

Please note that while this is a general guide, individual responses to caffeine can vary based on genetics, body mass, age, medication interactions, and tolerance levels.

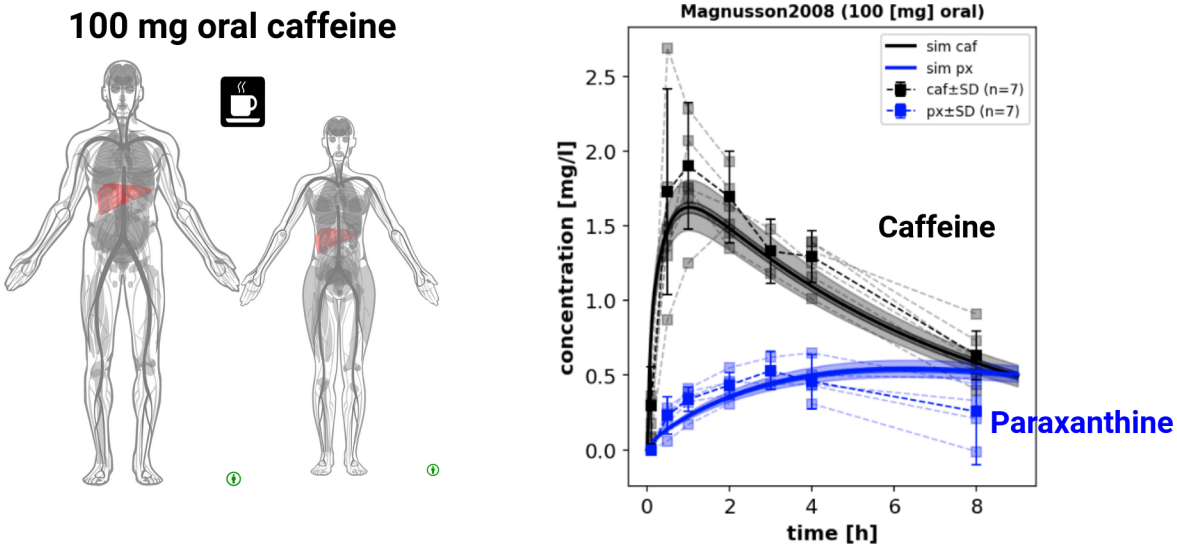


Figure 27.1: Caffeine Motivation

For questions contact konigmatt@googlemail.com. The latest version of the resources are available from https://github.com/matthiaskoenig/pkpd-course/releases

27.1 Install requirements

This tutorial works in a python environment with the following packages

```
numpy
scipy
matplotlib
pandas
libroadrunner
```

The packages can be installed via

```
pip install numpy scipy matplotlib pandas libroadrunner
```

or in `conda` via

```
conda install numpy scipy matplotlib pandas libroadrunner
```

The only additional requirement for this tutorial is `libroadrunner`

27.2 Download the model

The caffeine model can be downloaded from https://github.com/matthiaskoenig/pkpd-course/raw/develop/notebooks/models/caffeine_body_flat.xml. The model must be located in the same folder as the scripts.

```
# some magic (please ignore in spyder, only important for notebooks)
%load_ext autoreload
%autoreload 2
%matplotlib inline
```

```
import roadrunner
```

```
# general imports for ode integration
from pathlib import Path
import numpy as np
import pandas as pd
import roadrunner
from matplotlib import pylab as plt
from pprint import pprint

# global settings for plots (optional)
plt.rcParams.update({
        'axes.labelsize': 'large',
        'axes.labelweight': 'bold',
        'axes.titlesize': 'large',
        'axes.titleweight': 'bold',
        'legend.fontsize': 'small',
        'xtick.labelsize': 'large',
        'ytick.labelsize': 'large',
    })

# caffeine model
caffeine_sbml = str(Path(".").parent / "caffeine_body_flat.xml")


def simulate(r, start: float=0, end: float=10, steps: int=200, reset:bool=True) -> pd.DataFra
    """ Simulate given roadrunner model.

    :param r: roadrunner model
    :param start: start time
    :param end: end time
    :param steps: simulation steps
    :param reset: resets the model after simulation
    :return:
    """
    if reset:
        r.reset()
    s = r.simulate(start=start, end=end, steps=steps)
    return pd.DataFrame(s, columns=s.colnames)
```

27.3 Physiologically based pharmacokinetic (PBPK) model for caffeine

The model describes the clearance of caffeine by the liver in humans.

- Caffeine and the primary metabolite paraxanthine are removed from the blood by hepatic or renal clearance.
- Caffeine can be administered in the model either by intravenous injection or by oral dose.

This tutorial demonstrates some simple use cases and analysis of the model.

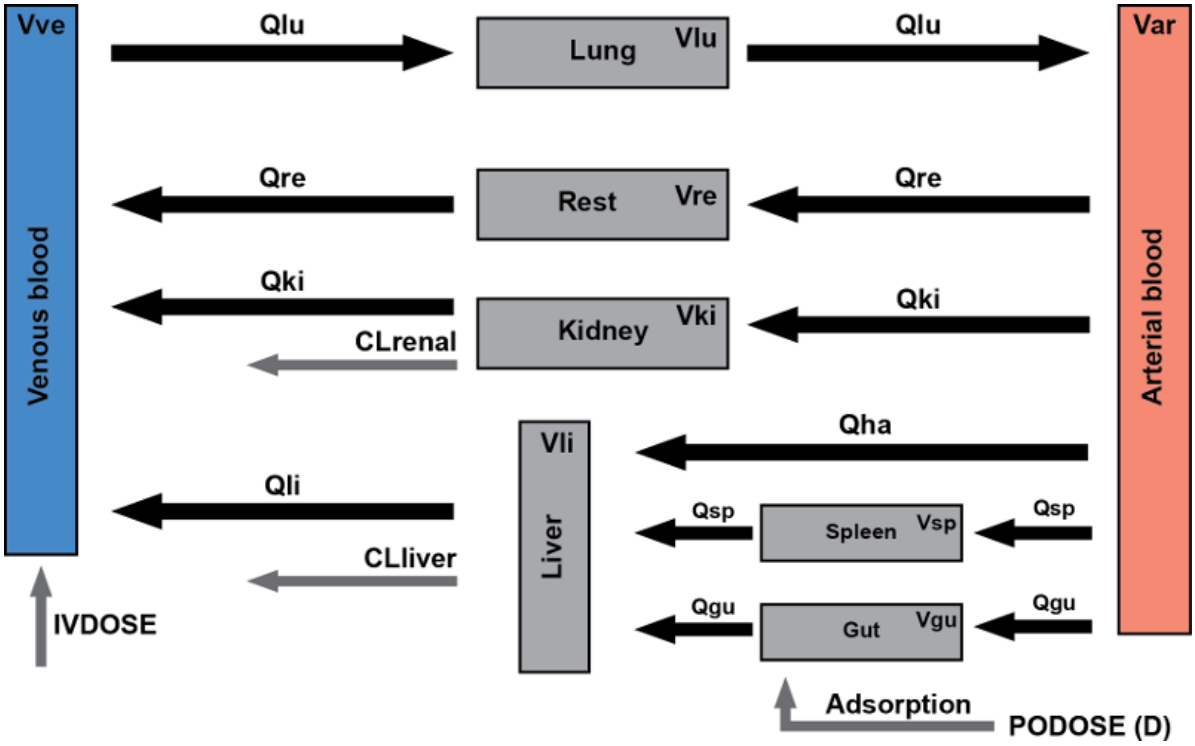


Figure 27.2: Fig.1 Caffeine PKPD Model

27.4 Load the caffeine model

```
# from helpers import *
print(caffeine_sbml)

# load the model
```

```
r = roadrunner.RoadRunner(caffeine_sbml)
# set variables in result
r.timeCourseSelections = ["time"] + r.model.getFloatingSpeciesIds() + r.model.getGlobalParame
# pprint(r.timeCourseSelections)
```

```
caffeine_body_flat.xml
```

The model is an SBML model and you can view the corresponding file and HTML report. This gives a good overview of the contents of the model. This report can be generated from https://sbml4humans.de

27.5 Example simulation

We now simulate a 100 [mg] oral dose (p.o) of caffeine. In a first step we perform this simulation and look at some state variables of the model.

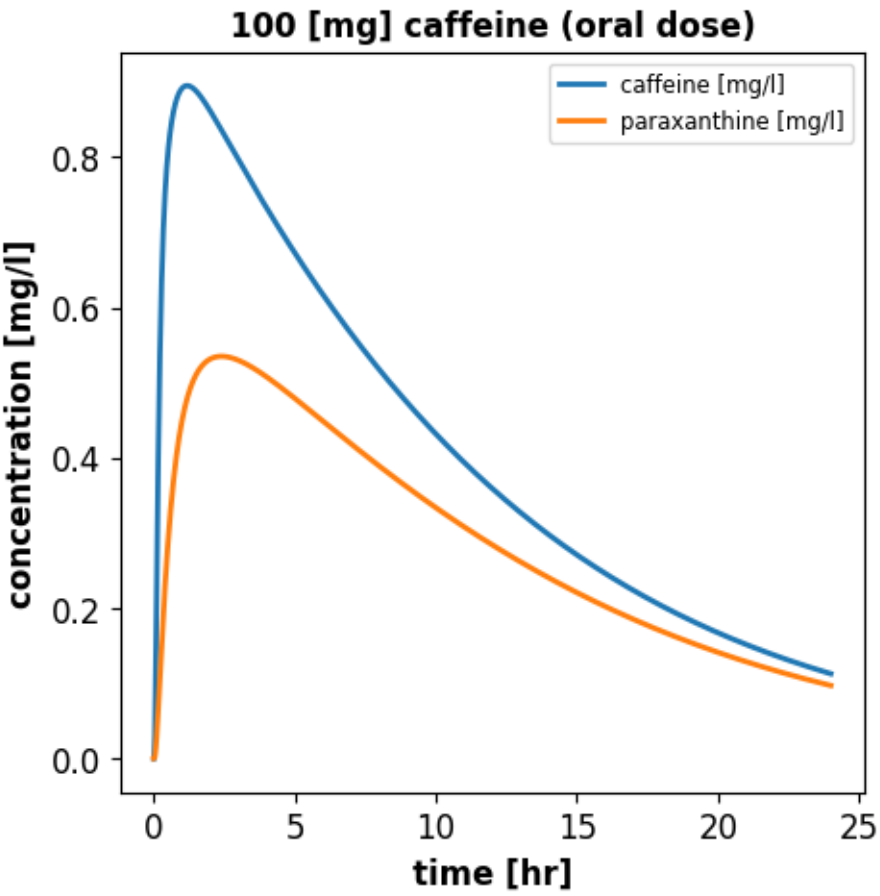
The model time is in [h]. The simulation result is a pandas DataFrame which can easily be accessed.

```
# reset model to initial state
r.resetToOrigin()

# set the oral dose
r["init(PODOSE_caf)"] = 100  # [mg]

# set parameters
r["BW"] = 100  # [kg] bodyweight
# r["LI__CAF2PX_Vmax"] = 2E-3  # [mmole/min/l] rate caf -> px
# r["KI__PXEX_f"] = 5  # [-] renal clearance paraxantine

# simulate the model for 24 [hr], model time is in [min]
s = simulate(r, start=0, end=24*60, steps=1200)  # [min]

# show the resulting DataFrame
print(s.head())
```

```
   time  Cgu_plasma_caf  Cki_plasma_caf  Cli_plasma_caf  Clu_plasma_caf  \
0   0.0        0.000000    0.000000e+00        0.000000    0.000000e+00
1   1.2        0.000019    3.142602e-08        0.000003    1.359226e-07
2   2.4        0.000058    4.265750e-07        0.000017    1.097020e-06
```

```
3   3.6        0.000102    1.360261e-06        0.000038    2.943392e-06
4   4.8        0.000144    2.659598e-06        0.000062    5.310823e-06

   Cre_plasma_caf   Car_caf   Cve_caf   Cpo_caf   Chv_caf  ...     Mve_px  \
0    0.000000e+00  0.000000  0.000000  0.000000  0.000000  ...  0.000000
1    1.867982e-07  0.000013  0.000043  0.000024  0.000006  ...  0.000297
2    4.140970e-06  0.000127  0.000310  0.000074  0.000030  ...  0.002833
3    1.934535e-05  0.000358  0.000802  0.000130  0.000069  ...  0.008538
4    5.119269e-05  0.000661  0.001422  0.000185  0.000112  ...  0.016942

         Apo_px   Cpo_free_px    Xpo_px    Mpo_px        Ahv_px  Chv_free_px  \
0  0.000000e+00  0.000000e+00  0.000000  0.000000  0.000000e+00     0.000000
1  2.403103e-08  6.392825e-07  0.000004  0.000115  4.188096e-07     0.000011
2  3.926420e-07  1.044521e-05  0.000071  0.001882  2.678790e-06     0.000071
3  1.405600e-06  3.739231e-05  0.000253  0.006737  6.484786e-06     0.000173
4  3.018005e-06  8.028611e-05  0.000544  0.014464  1.096036e-05     0.000292

     Xhv_px    Mhv_px  dissolution_caf
0  0.000000  0.000000         0.085827
1  0.000075  0.002007         0.070269
2  0.000483  0.012839         0.057531
3  0.001168  0.031080         0.047103
4  0.001975  0.052530         0.038564

[5 rows x 164 columns]
```

```
# plot venous caffeine & paraxanthine concentrations against time
f1, ax1 = plt.subplots(1, 1, figsize=(5, 5))

time_hr = s.time/60  # [min] -> [hr]

# caffeine concentration, venous blood
ax1.plot(time_hr, s.Mve_caf, linewidth=2, label="caffeine [mg/l]", color="tab:blue")
# paraxanthine concentration, venous blood
ax1.plot(time_hr, s.Mve_px, linewidth=2, label="paraxanthine [mg/l]", color="tab:orange")
ax1.set_title('100 [mg] caffeine (oral dose)')
ax1.set_ylabel('concentration [mg/l]')
ax1.set_xlabel('time [hr]')
ax1.legend()
plt.show()
```

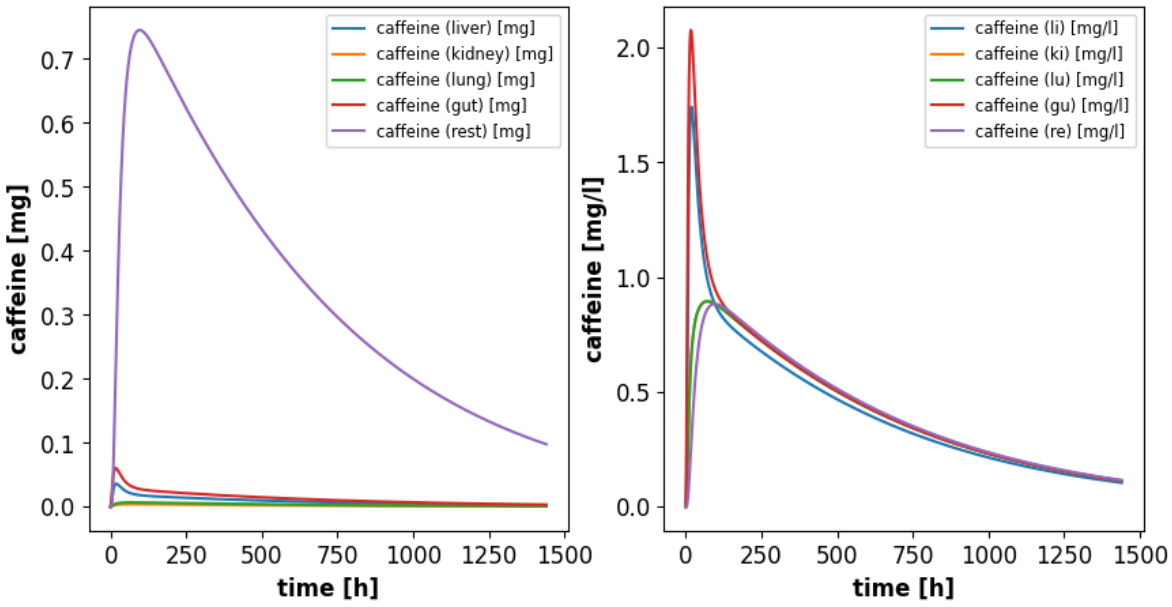


27.6 Compare amounts in different organs

In the following we compare the amount of caffeine in the different organs. For this we select all the columns in the solution which belong the amount of caffeine.

```
# plot caffeine and paraxanthine amounts
f1, (ax1, ax2) = plt.subplots(1, 2, figsize=(10,5))
axes = (ax1, ax2)
organs = {
    "li": "liver",
    "ki": "kidney",
    "lu": "lung",
    "gu": "gut",
    "re": "rest",
}
```

```
for organ, name in organs.items():
    sid = f"X{organ}_plasma_caf"
    label = f"caffeine ({name}) [mg]"
    ax1.plot(s.time, s[sid], label=label)

    sid = f"M{organ}_plasma_caf"
    label = f"caffeine ({organ}) [mg/l]"
    ax2.plot(s.time, s[sid], label=label)


ax1.set_ylabel('caffeine [mg]')
ax2.set_ylabel('caffeine [mg/l]')
for ax in axes:
    ax.set_xlabel('time [h]')
    ax.legend()

plt.show()
```



27.7 Plot the organ volumes

In the following we plot the organ volumes by querying the model. Model variables and parameters are available via the dot syntax (`.`) or the bracket access (`[]`).

```
for oid in organs:
    vid = f"V{oid}"
    print(f"V{oid} = {r[vid]} [l]")
```

```
Vli = 2.1 [l]
Vki = 0.44 [l]
Vlu = 0.76 [l]
Vgu = 2.97 [l]
Vre = 86.02000000000001 [l]
```

```
print("relative volume:", r.FVli)
print("relative perfusion:", r.FQh)
```

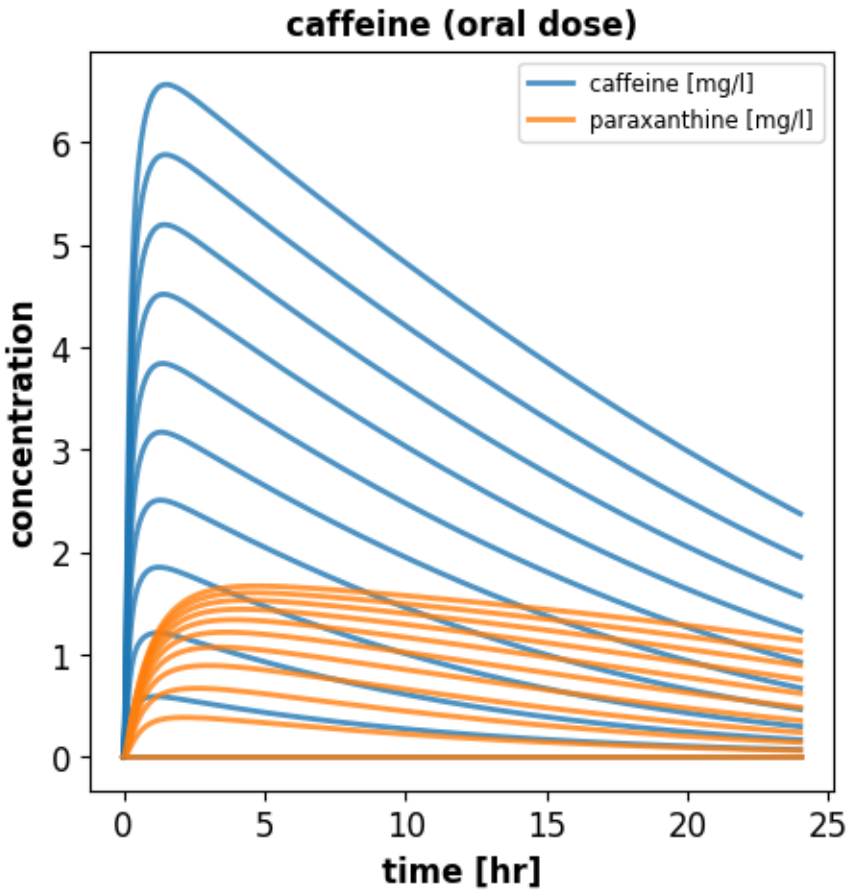
```
relative volume: 0.021
relative perfusion: 0.215
```

27.8 Stepwise increase of the caffeine dose

Now we see what happens if we drink more coffee every day

```
doses = np.linspace(0, 500, num=11)
results = []

for dose in doses:
    # reset model to initial state
    r.reset()

    # set the oral dose
    r['init(PODOSE_caf)'] = dose  # [mg]

    # simulate the model for 24[h]
    s = simulate(r, start=0, end=24*60, steps=500)
    results.append(s)

# plot venous caffeine & paraxanthine against time
f1, ax1 = plt.subplots(1, 1, figsize=(5, 5))
for k, s in enumerate(results):
    time_hr = s.time/60
    if k == 1:
```

```
        label = "caffeine [mg/l]"
    else:
        label = "__nolabel__"
    ax1.plot(time_hr, s.Mve_caf, linewidth=2, color="tab:blue", label=label, alpha=0.8)
    if k == 1:
        label = "paraxanthine [mg/l]"
    else:
        label = "__nolabel__"
    ax1.plot(time_hr, s.Mve_px, linewidth=2, color="tab:orange", label=label, alpha=0.8)
ax1.set_title('caffeine (oral dose)')
ax1.set_ylabel('concentration')
ax1.set_xlabel('time [hr]')
ax1.legend()
plt.show()
```



28 Exercises

28.1 E1 Your caffeine level

The first exercise is to calculate the timecourse of the expected venous caffeine level after you drink some caffeinated beverage. To estimate your oral dose of caffeine you can estimate the caffeine content from the following chart

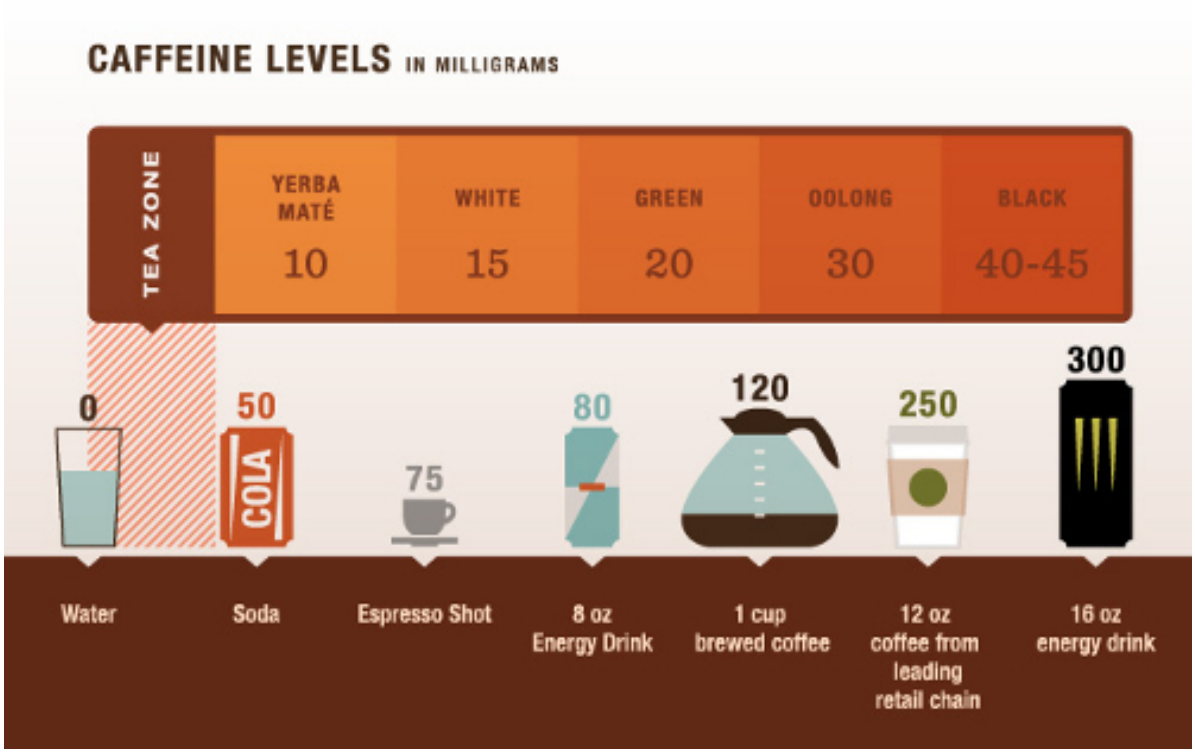


Figure 28.1: Fig.2 Caffeine Content of Beverages

The oral dose is defined in the model via parameter `r['init(PODOSE_caf)'] = 100` [mg]. In addition you can also adjust the bodyweight to get a more realistic estimation via the parameter `r['BW'] = 70` [kg].

- What would be your level of caffeine now, if you had two cups of coffee for breakfast this morning?
- How would your time course and level of caffeine look if you would take the same amount of caffeine intravenously (I.V)? (Hint: you have to set the set the i.v. dose via `r['init(IVDOSE_caf)'] = 100`)
- What is the peak time of caffeine in venous blood? What is the peak concentration?

28.2 E2 Interindividual variability

We saw that there is a large variability in caffeine kinetics in the population. Depending on if you are a fast or slow metabolizer of caffeine the timecourses can look very different. In E1 you calculated the mean timecourse for the population. Now we will look at the interindividual differences.

Your caffeine clearance by the liver depends on the activity of CYP1A2 in the liver, the main enzyme metabolizing caffeine. The activity is defined via the liver parameter (`r['LI__cyp1a2'] = 1` [dimensionless]).

- How would your time course / level of caffeine change if you are a slow metabolizer (small apparent clearance), or if you are a fast metabolizer (large apperent clearance)?
- Simulate the effect of lifestyle changes on your caffeine clearance via adjusting the caffeine clearance accordingly. For instance simulate changes in your coffee intake or smoking habit. An overview over the changes in apparent clearance are given in Tab.1.
- How would your caffeine timecourse change if you smoke >20 cigarettes per day and drink 1 liter of coffee (the effects are additive) compared to being abstinent?
- Also the bodyweight has a strong influence on the distribution of caffeine. What happens when setting your body weight? (`r.BW = 75` [kg])

Table 4. Parameter estimates of covariates obtained for logarithmic clearance values using the paraxanthine/caffeine ratio method (equation 1)

Covariate	*Symbol used in equation 5*	*Estimate*	*95% Confidence interval*		*Mean resulting change of clearance (factor)*
			Lower bound	*Upper bound*	
–	Intercept	0.264	–0.015	0.542	–
Coffee intake (litre day^{-1})	$Slope_{coffee}$	0.368	0.287	0.449	1.445
Body mass index (kg m^{-2})	$Slope_{BMI}$	–0.010	–0.018	–0.002	0.990
Cigarettes/day					
Non-smokers	$V_{smoking\ habit\ index}$	0	–	–	Reference
1–5		0.195	0.065	0.324	1.215
6–10		0.383	0.253	0.509	1.467
11–20		0.504	0.386	0.621	1.655
>20		0.543	0.430	0.655	1.721
Oral contraceptives					
No	$V_{oral\ contraceptive\ index}$	0	–	–	Reference
Yes		–0.332	–0.236	–0.428	0.717
Country					
Germany	$V_{country\ of\ residence\ index}$	0	–	–	Reference
Bulgaria		–0.209	–0.356	–0.061	0.811
Slovakia		–0.303	–0.450	–0.156	0.739
Sex					
Male		0	–	–	Reference
Female	$V_{sex\ index}$	–0.111	–0.178	–0.044	0.895

Figure 28.2: Tab.1 Lifestyle Effects

Part III

Technology

29 Technology details

This section provides information on how to setup the different parts of the course. This is a working document to keep track of the technology stack and how to configure the parts.

29.1 uv

As part of the course python code has to be executed. All requirements are managed via virtual environments. The dependencies are managed using uv. For installation information see: https://docs.astral.sh/uv/getting-started/installation/

The virtual environment can be setup via:

```
uv venv
uv sync
```

To activate the virtual environment use

```
source .venv/bin/activate
```

or the corresponding command for your operating system. The environment should be set as python interpreter in vscode and pycharm.

29.2 Quatro

Information about installation is available from https://quarto.org/docs/get-started/.

This documents were created with the prerelease version `quarto-1.7.21-linux-amd64.deb` available from https://quarto.org/docs/download/prerelease.html.

Documentation for the pre-release is available at https://prerelease.quarto.org/.

29.2.1 VS Code Plugin

Information on the visual studio code integration is available here: https://quarto.org/docs/tools/vscode.html

For executing the python code the python interpreter must be set and the jupyter plugin must be installed.

29.2.2 Github actions for publishing

The output can be automatically generated using github actions. https://quarto.org/docs/publishing/github-pages.html#github-action

29.2.3 Rendering and preview

The content can be rendered to the respective output format using the `render` command.

```
quarto render notebooks/hello.ipynb --to html
quarto render notebooks/hello.ipynb --to pdf
quarto render notebooks/hello.ipynb --to revealjs
quarto render notebooks/hello.ipynb --to pptx
```

To execute the notebooks use the `execute` flag

```
quarto frender notebook.ipynb --execute
```

To preview the HTML use the `preview` command.

```
quarto preview notebooks/hello.ipynb
```

29.2.4 Render to reveal.js and book

Necessary to render to different target types: Presentation (reveal.js) -> hosted on Github.io; necessary to render to a different target; see https://github.com/quarto-dev/quarto-cli/discussions/1433; see https://github.com/quarto-dev/quarto-cli/issues/2200. By using different profiles these information can be combined.

29.3 Open edX

Open edX is a thriving open source project, backed by a great community, for running an online learning platform at scale. Open edX comes with an LMS (Learning Management System) where students access course contents, a CMS (Content Management System) that course staff uses to design courses, and a few other components to provide more services to students, course staff, and platform administrators.

A local installation of OpenEdx can be setup with tutor: https://docs.tutor.edly.io/index.html. This requires a working docker installation: https://docs.docker.com/engine/install/.

```
source .venv/bin/activate
tutor local launch
```

This results in

```
All services initialised.
The platform is now running and can be accessed at the following urls:

    http://local.openedx.io
    http://studio.local.openedx.io
    http://meilisearch.local.openedx.io
    http://apps.local.openedx.io
```

29.3.1 Creating a new user with staff and admin rights

You will most certainly need to create a user to administer the platform. Just run:

```
tutor local do createuser --staff --superuser matthiaskoenig konigmatt@googlemail.com
```

29.3.2 Importing the demo course

After a fresh installation, your platform will not have a single course. To import the Open edX demo course, run:

```
tutor local do importdemocourse
```

29.3.3 View status of containers

You can view your platform's containers:

```
tutor local status
```

29.3.4 Open edX notebook/jupyter integration

Tutor plugin for running Jupyter notebooks.

```
- name: jupyter
  src: tutor-jupyter>=19.0.0,<20.0.0
  url: <https://github.com/overhangio/tutor-jupyter>
  author: Edly <hello@edly.io>
  maintainer: Edly <abdul.muqadim@arbisoft.com>
  description: |
    Run Jupyter notebooks right in your LMS.

    Launch a JupyterHub single-node cluster and install the Jupyter XBlock in the Open
    edX LMS/CMS. This makes it very easy to launch student-editable code editors rights
    in your courses.
```

- openedx notebook integration: https://github.com/parmentelat/nbhosting ?!
- https://github.com/overhangio/jupyter-xblock JupyterHub hosts Jupyter instances with authentication (e.g. OAuth; IDK about using edX users as JupyterHub users with individual Docker image instance containers with nbgrader/xblock) The new jupyter-viewer-xblock (https://github.com/ibleducation/jupyter-viewer-xblock) allows to embed notebooks dynamically from a public URL. Demo here! https://www.youtube.com/watch?v=K8jhWgQnxvI
- https://github.com/overhangio/tutor-jupyter; This is a plugin for Tutor that makes it easy to integrate Jupyter notebooks in Open edX. It achieves the following: 1. Install the official jupyter-xblock in the Open edX LMS and Studio. 2. Run a Docker-based JupyterHub instance with a Docker spawner.

29.3.5 H5P

Open edX supports integration of H5P. - Play H5P content in Open edX using h5pxblock - https://github.com/edly-io/h5pxblock

29.4 edX course creation

29.4.1 HarvardX Course Template Builder

Allows to create some template structure of the course https://harvardx.github.io/edx_course_templater/index.html

A while ago I created the HarvardX Course Template Builder for our project leads. You answer some questions about your course (length, structure, desired template) and it builds you a blank course that you can import to Studio. It really speeds up the process of clicking "New Unit" over and over.

You are welcome to use it as much as you like. My bosses have ok'd releasing this into the wild, including our usual boilerplate (intro pages and a few sample items). The code is available on a GitHub repo if you want to make your own custom version or just see how it works.

29.4.2 XNF conversion

https://git.upv.es/serpucga/xnf2edx_cli/-/tree/master https://discuss.openedx.org/t/script-to-create-complete-open-edx-courses-from-a-template-in-an-excel-file/8520

https://discuss.openedx.org/t/programmatically-create-courses-including-all-xblock-children/9206

29.4.3 Mu

29.4.4 Common Cartrige (CC)

https://github.com/openedx/cc2olx

29.4.5 OBS

- add video source
- add filter (chroma key green)

30 Technology review

Here we performed a search of available open technologies which could be used for the course. Some of these are employed in the course. Overview of technology relevant for the course.

30.1 Interactive plots

An important part are interactive plots in the python notebooks. Most of the static plots are currently be generated with matplotlib.

30.1.1 plotly

- https://plotly.com/python/

30.1.2 altair

- https://altair-viz.github.io/

30.2 Interactive web applications

To demonstrate simple relationships between parameters/settings and outputs interactive small web applications with minimal overhead are a great solution. E.g. are the indocyanine green application

30.2.1 Shiny for python

- possible integration with Quarto

30.2.2 Voila

- voila notebooks: https://voila.readthedocs.io/en/stable/
- Voilà allows you to convert a Jupyter Notebook into an interactive dashboard that allows you to share your work with others. It is secure and customizable, giving you control over what your readers experience.

30.2.3 Streamlit

- https://streamlit.io/
- Turn your data scripts into shareable web apps in minutes. All in pure Python. No front-end experience required.

30.2.4 Dash

- Developed by Plotly, Dash is ideal for creating complex, interactive web applications using Python.
- Supports multi-page apps and scales well for large datasets and multiple users.
- Requires knowledge of HTML, CSS, and JavaScript for advanced customization

30.2.5 Panel

- https://panel.holoviz.org/
- Panel is an open-source Python library designed to streamline the development of robust tools, dashboards, and complex applications entirely within Python.

30.2.6 Framework

- https://github.com/observablehq/framework
- Observable Framework is a free, open-source, static site generator for data apps, dashboards, reports, and more. Framework combines JavaScript on the front-end for interactive graphics with any language on the back-end for data analysis. Framework features data loaders that precompute static snapshots of data at build time for dashboards that load instantly.

30.3 Deployment of notebooks

30.3.1 binderhub

- very slow
- https://binderhub.readthedocs.io/en/latest/
-

30.3.2 jupyterhub

- JupyterHub brings the power of notebooks to groups of users. It gives users access to computational environments and resources without burdening the users with installation and maintenance tasks. Users - including students, researchers, and data scientists - can get their work done in their own workspaces on shared resources which can be managed efficiently by system administrators.
- https://jupyter.org/hub
- https://tljh.jupyter.org/en/latest/howto/index.html

https://education.github.com/globalcampus/teacher?email_referrer=true

nbgrader: https://nbgrader.readthedocs.io/en/stable/ https://www.youtube.com/watch?v=bEcxnR2V-_8 Otter grader & gradescope

31 GitHub Global Campus

Focus on teaching with GitHub Classroom Managing and organizing your class is easy with GitHub Classroom. Track and manage assignments, automate grading, and help students when they get stuck— all while using GitHub.

31.1 Educational resources

Resources on building a digital course, Course structure and best practise

31.2 Teaching and Learning with Jupyter

This handbook is for any educator teaching a topic that includes data analysis or computation in order to support learning. - https://jupyter4edu.github.io/jupyter-edu-book/

31.3 Open edX Educators

- Material on how to build and design courses: https://docs.openedx.org/en/latest/educators/index.html
- Instructional Design Concepts: https://docs.openedx.org/en/latest/educators/navigation/creating_course design-concepts

31.4 Text to speech

31.4.1 Whisper AI

- https://openai.com/index/whisper/

31.4.2 ElevenLabs

- https://elevenlabs.io/

32 Mu

See also markdown xblock! https://github.com/citynetwork/markdown-xblock

Writing courses in markdown.

Mu Course authoring for humans https://discuss.openedx.org/t/introducing-mu-course-authoring-for-humans/9314

In a nutshell, Mu allows you to write your courses in Markdown or HTML5, and then convert them to OLX, and vice-versa.

https://github.com/overhangio/mu

32.1 Installation

pip install mu-courses

Install up to date pandoc version https://github.com/jgm/pandoc/releases/ pandoc 3.6.4

mkdir olx mu ./course.md ./olx/

Package the course

```
tar -czf mycourse.tar.gz ./olx
```

Part IV

TODO

33 Open issues

Here we collect open issues and things which are still to do. This is by no means comprehensive but more of an infomration drop.

33.1 DHPE course

DHPE teaching project overview.

- ☐ update the teaching concept: `teaching_concept.qmd`
- ☐ create workflow picture

33.2 Proof of principle implementation

Create a proof of principle for a first lecture: **structural models**. This should demonstrate the complete learning path with the different used technologies

- ☒ Create script/text + teaching notebook (for visualization)
- ☒ Create slides 5-15 min
- ☐ Create video of slides
- ☐ Create teaser videos
- ☐ interactive tutorial notebook –> reveal.js
- ☐ interactive tutorial notebook solutions –> reveals.js

33.3 OpenEdX

Create course section Open edX (from script content).

- ☒ setup local platform for development and testing
- [~] create initial course
- ☐ test content from script (import course content; qmd -> OLX(mu) -> course format?; other options? xmarkdown))
- ☐ test interactivity
- ☐ test quizz

- ☐ test h5p integration
- ☐ deployment

33.4 Feature demos

The following features work, but have to still be implemented or activated

- ☐ references from zotero with workflow
- ☐ add button “edit on github to the pages”

33.5 Open issues

- ☐ How to properly greenscreen videos (canva remove background not very convincing; OBS chroma key filter)?
- ☐ How to programmatically generate Open edX course from markdown and XML (interactive components). Various scripts and proof-of-principles available, but nothing convincing so far. We want to version control all course content and be able to transfer lessions/sections content to other courses easily.

33.6 Bugs

- ☐ section headers in notebooks not working
- ☐ PDF compilation in CI not working